THE UNIVERSITY OF CHICAGO

NEW ABSTRACTIONS FOR 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
CASEY DUCKERING

CHICAGO, ILLINOIS
DECEMBER 2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

TODO

ABSTRACT

The field of quantum computing is at an exciting time where we are building devices, running programs, and finding out what works best. As qubit technology grows and matures, we need to be ready to design and program larger quantum computer systems. An important aspect of systems design is layered abstractions to reduce complexity and guide intuition. Classical computer systems have built up many abstractions over its history including the layers of the hardware stack and programming abstractions like loops. Researchers initially ported these abstractions with little modification when designing quantum computer systems and only in recent years have some of those abstractions been broken in the name of optimization and efficiency.

We argue that new or quantum-tailored abstractions are needed to get the most benefit out of quantum computer systems. We keep the benefits gained through breaking old abstraction by finding abstractions aligned with quantum physics and the technology. This dissertation is supported by three examples of abstractions that could become a core part of how we design and program quantum computers: third-level logical state as scratch space, memory as a third spacial dimension for quantum data, and hierarchical program structure.

CHAPTER 1

INTRODUCTION

Moore's Law and the expectation that computers double in speed every 18 months is at an end, so hard problems in chemistry, physics simulation, and combinatorial optimization cannot be solved by waiting for a faster computer. Since the end of Moore's Law, researchers have been developing special-purpose accelerators to squeeze better performance out of each transistor. However once fully realized, quantum computers can solve specific classes of problems in simulation and cryptography exponentially faster.

Quantum computers work by harnessing quantum physics instead of classical Newtonian physics. Because quantum physics is a superset of classical physics, we often treat quantum computers as classical computers with the additional features of *superposition*, *entanglement*, and *interference*. This view is apparent in Shor's algorithm (Shor [1997]) which creates a quantum superposition, followed by classical arithmetic, and finishes with quantum phase estimation.

Seeing quantum programming through a classical lens can be limiting and sometimes harmful. It is common for programmers who are new to quantum to invent a "quantum algorithm" that is simply a randomized classical algorithm run on a quantum computer. More subtly, concepts such as binary representation of data, random access memory, and hierarchical modularity of programs when used in the design of quantum computers limit the performance due to mismatches with the underlying technology. Even classical concepts of causality and movement of data can be limiting; quantum teleportation, described in Nielsen and Chuang [2011], moves quantum data long distances by pre-transferring another resource *before* the data exists.

When we design quantum architectures and compilers, the abstractions we use are key to a good design. The abstraction of two-level *bits* is very beneficial for classical computer reliability but is yet to be decided for quantum. Early classical computers used base-10 addresses and arithmetic until early computer architects settled on binary as the most efficient and reliable design. This history informs the general assumption that binary (base-2) is best for quantum computers but that is not necessarily the case. We discuss this further in Chapter 2.

Because quantum computing is a rapidly developing field with many competing technologies there is no clear “best” for any use case. Each quantum technology has capabilities and constraints that inform a variety of architectures that show how to turn a qubit technology into a practical quantum computer. The principles of abstraction and modularity we use to build any complex system still apply when we design a quantum computer architecture, instruction set, compiler, and programming language, but we must tailor the abstractions to best fit the physics and the technology or we will limit future efficiency.

This dissertation presents three cases of new or old abstractions that we have tailored for quantum computing. We discuss the methodologies to select these abstractions and how we use them with a particular class of quantum architectures. We show that good abstractions can allow more space efficient algorithms or more effective compilers.

This dissertation comprises three core papers introducing three abstractions covered in the following chapters. Additional content from other work is included that show further benefits and refinement to the abstractions. We start in Chapter 2 by introducing three-level quantum *trits* and d-level quantum *dits*: Asymptotic Improvements to Quantum Circuits via Qutrits, Gokhale et al. [2019] and Efficient Quantum Circuit Decompositions via Intermediate Qudits, Baker et al. [2020a]. These abstractions replace and augment the use of binary qubits with three-level *qutrits* or d-level *qudits*, but require us to completely rethink how algorithms and compilers allocate and use scratch space. Most quantum technologies can reliably support three or more quantum states with minor changes to the control signal design and no change to the hardware design. Supported technologies include superconducting transmon, ion trap, and neutral atom, but notably not some types of photonic qubits.

Chapter 3 considers abstractions that spatially separate quantum data storage or memory from computation on that data: Virtualized Logical Qubits: A 2.5D Architecture for Error-Corrected Quantum Computing, Duckering et al. [2020]. Classical computers contain high speed buses that can transfer data between memory (RAM) and computation (CPU), but this extreme separation of memory from compute does not make sense either for current small (NISQ) or for future (fault-tolerant) quantum computers. The typical abstraction for both kinds is a monolithic 2D array of qubits because NISQ computers cannot sacrifice the data-parallelism and fault-tolerance requires constant error correction to prevent errors. Compiler design is simple in this monolithic model because there is no heterogeneity; just place related data nearby in the plane. But we compare an alternative to the monolithic model. We redesign the surface code to use small amounts of distributed memory and find that it improves the space efficiency of fault-tolerant algorithms.

Classical programmers have used a hierarchy of function calls and modules in the design of a program to great effect. Hierarchy gives structure to what would otherwise be a very long list of primitive instructions. Compilers use this structure to guide optimizations and to avoid duplicate work of repeated components. However, quantum programmers currently trend toward highly hand-optimized programs with no hierarchy to maximize flat, program-wide optimizations that erase hierarchy. Chapter 4 introduces Orchestrated Trios: Compiling for Efficient Communication in Quantum Programs with 3-Qubit Gates, Duckering et al. [2021], to show that hierarchy can guide quantum compiler heuristics even for small- to mid-size programs. Program hierarchy enables sequences of compiler passes to repeat for each level, improving heuristic performance and allowing new kinds of passes like our connectivity-aware split pass. This is key for quantum where data locality constraints restrict data movement and can inform program structure.

Picking the right abstractions are crucial for quantum programming, compiling, and execution. Chapter 5 concludes with a discussion and other places where we still need better abstractions.

CHAPTER 2
BEYOND BINARY

2.1 Introduction

Recent advances in both hardware and software for quantum computation have demonstrated significant progress towards practical outcomes. While early research efforts focused on longer-term systems employing full error correction to execute large programs for algorithms like Shor [1997] and Grover [1996], recent work has focused on NISQ (Noisy Intermediate Scale Quantum, Preskill [2018]) computation. The NISQ regime considers near-term machines with just tens to hundreds of quantum bits (qubits) and moderate errors.

In the NISQ regime, quantum programs rely directly on the individual qubits in the quantum device and severe resource constraints prohibit the use of error correction. Given the severe constraints on quantum resources, it is critical to fully optimize the compilation of a quantum program in order to have successful computation. Prior architectural research on techniques such as mapping, scheduling, and parallelism (Ding et al. [2018], Javadi-Abhari et al. [2017], Guerreschi and Park [2018]) have helped to extend the amount of useful computation possible, but without error correction, programs are exposed to noise and errors in their qubits. On the flip side, program in the NISQ regime can directly take advantage of typically unused technology capabilities.

This chapter shows how to greatly reduce resource requirements by replacing the binary abstraction required by two-level qubits with a new abstraction enabled by three-level *qutrits* or multi-level *qudits*. Qutrits and qudits are natural features of technologies in the NISQ regime, which we evaluate, but the takeaways from this chapter may require further research to apply in an error-corrected setting.

While quantum computation is typically expressed as a two-level binary abstraction of qubits, the underlying physics of quantum systems are not intrinsically binary. Whereas classical computers operate in binary states at the physical level (e.g. clipping above and below a threshold voltage), quantum computers have natural access to an infinite spectrum of discrete energy levels. In fact, hardware must actively suppress higher level states in order to achieve the two-level qubit approximation. Hence, using three-level qutrits is simply a choice of including an additional discrete energy level, albeit at the cost of more opportunities for error.

Prior work on qutrits (or more generally, d-level *qudits*) identified only constant factor gains from extending beyond qubits. In general, the prior work Pavlidis and Floratos [2017] has emphasized the information compression advantages of qutrits. For example, N qubits can be expressed in base-3 ternary as $\frac{N}{\log_2(3)}$ qutrits, which leads to $\log_2(3) \approx 1.6$-constant factor improvements in space and runtime.

This chapter evaluates the benefits of a novel abstraction that uses qutrits in a novel fashion. We use the first two states as usual to represent computed values in binary but use the third state as temporary storage when needed. The per-operation error rate of qutrit operations is higher but the runtime (i.e. circuit depth or critical path) is *asymptotically* faster, and the overall reliability of computations is improved due to the novel temporary storage. Moreover, this abstraction only applies qutrit operations in an intermediary stage: the input and output are still qubits, which is important for initialization and measurement on real devices (Randall et al. [2015, 2018]) and reduces the burden to transition to the new abstraction.

We consider the benefits of different applications of this temporary qutrit abstraction. The first application we consider is a novel implementation of the generalized Toffoli circuit by Gokhale et al. [2019], a subroutine used in many quantum algorithms. By cleverly storing intermediate computations the third state of input qubits, our implementation avoids the use of costly additional temporary qubits (called ancilla), but it achieves the speed of the fastest implementations that require many ancilla qubits.

In contrast, we also consider potential automated uses of temporary qutrits. The hand-designed generalized Toffoli implementation makes excellent use of one additional logical state and, while hand-optimization can be a good way to squeeze performance out of resource-constrained devices, codifying manual strategies into our compilers can have wider performance benefit and free most programmers to think at a higher level. By intelligently "compressing" the data in groups of idle qubits into smaller groups of qutrits as in Baker et al. [2020a] (using the $\log_2(3)$ compression ratio) or qudits ($\log_2(d)$ ratio), similar benefits for resource-constrained quantum computers can be achieved for a wider range of quantum programs.

The main benefit of compression is to produce ancilla, specifically clean ancilla, *generated* locally during the compilation of an algorithm into a quantum circuit. That is, we propose a new circuit which performs qubit-qudit compression storing the information of many qubits as a small number of qudits at the cost of some gate overhead. These compression circuits produce clean ancilla in the $|0\rangle$ state. The stored data can be retrieved later when needed since all quantum operations are reversible (this is technically a re-encoding, not compression). Essentially, when certain groups of qubits will be unused for a long period of time, we can repurpose them by compressing them and using the produced ancilla. This "compression" is a rearrangement of the stored binary values into higher states, letting us store more information into the same number of physical quantum devices and free up qubits for computation. We evaluate this compression strategy in the design of an improved quantum adder circuit.

The key result of this chapter is that use of this abstraction by quantum subroutines or compilers extends the frontier of what limited-size quantum computers can compute. In particular, the frontier is defined by the zone in which every machine qubit is a data qubit, for example a 100-qubit program running on a 100-qubit machine. In this frontier zone, we do not have room for non-data workspace qubits known as ancilla. The lack of ancilla in the frontier zone is a costly constraint that generally leads to inefficient circuits. For this reason, typical circuits instead operate well below the frontier zone, with many machine qubits used as ancilla. This chapter demonstrates that ancilla can be substituted with qutrits, enabling us to operate efficiently within the ancilla-free frontier zone.

We highlight the primary contributions of this chapter:

1. A circuit construction for the generalized Toffoli using temporary qutrits and no ancilla qubits. This is an asymptotically faster circuit ($633N \to 38 \log_2 N$) than equivalent qubit-only ancilla-free constructions.

2. Qutrit and qudit "compression" circuit designs.

3. A circuit construction for arithmetic addition in binary using qudit compression and no ancilla qubits.

4. An open-source qudit circuit library and simulator, now a core feature of Google's Cirq (Cirq).

This chapter is organized as follows: Section 2.2 presents relevant background about quantum computation and Section 2.3 outlines related prior work that we benchmark our work against. Section 2.4 demonstrates our key circuit construction, and Section 2.5 surveys applications of this construction toward important quantum algorithms. Section 2.6 introduces our open-source qudit circuit simulator. Section 2.7 explains our noise modeling methodology, and Section 2.8 presents simulation results for the generalized Toffoli circuits under these noise models.

In the remainder of the chapter, we present an application of this technique to give logarithmic depth decompositions of quantum arithmetic circuits—a carry lookahead adder and, by extension, addition by a constant. In Section 2.9 we present two compression circuits for qubit-qutrit and qubit-ququart ($d = 4$) compression and evaluate advantages of various compression schemes. In Section 2.10 we present our decomposition of the zero-ancilla, in-place $A + B$ adder which takes as input two registers A and B of qubits and possibly carry-in and carry-out; any fresh $|0\rangle$ states used are generated locally. We then evaluate the costs of this decomposition. We end with extensions to our arithmetic decomposition in Sections 2.10.1 and 2.10.2 and finish with a discussion and summary in Section 2.11.

2.2 Background

A qubit is the fundamental unit of quantum computation. Compared to their classical counterparts which take values of either 0 and 1, qubits may exist in a superposition of the two states. We designate these two basis states as $|0\rangle$ and $|1\rangle$ and can represent any qubit as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ with $\|\alpha\|^2 + \|\beta\|^2 = 1$. $\|\alpha\|^2$ and $\|\beta\|^2$ correspond to the probabilities of measuring $|0\rangle$ and $|1\rangle$ respectively.

Quantum states can be acted on by quantum gates which (a) preserve valid probability distributions that sum to 1 and (b) guarantee reversibility. For example, the X gate transforms a state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ to $X|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$. The X gate is also an example of a classical reversible operation, equivalent to the NOT operation. In quantum computation, we have a single irreversible operation called measurement that transforms a quantum state into one of the two basis states with a given probability based on α and β.

In order to interact different qubits, two-qubit operations are used. The CNOT gate appears both in classical reversible computation and in quantum computation. It has a control qubit and a target qubit. When the control qubit is in the $|1\rangle$ state, the CNOT performs a NOT operation on the target. The CNOT gate serves a special role in quantum computation, allowing quantum states to become entangled so that a pair of qubits cannot be described as two individual qubit states. Any operation may be conditioned on one or more controls that act like the conditions of an if-statement, only performing the operation on the states where all controls are $|1\rangle$.

Many classical operations, such as AND and OR gates, are irreversible and therefore cannot directly be executed as quantum gates. For example, consider the output of 1 from an OR gate with two inputs. With only this information about the output, the value of the inputs cannot be uniquely determined. These operations can be made reversible by the addition of extra, temporary workspace bits initialized to 0. Using a single additional ancilla, the AND operation can be computed reversibly as in Figure 2.1.

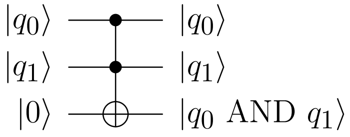


Figure 2.1: Reversible AND circuit using a single ancilla bit. The inputs are on the left, and time flows rightward to the outputs. This AND gate is implemented using a Toffoli (CCNOT) gate with inputs q_0, q_1 and a single ancilla initialized to 0. At the end of the circuit, q_0 and q_1 are preserved, and the ancilla bit is set to 1 if and only if both other inputs are 1.

Classical operations are fed an input state and produce an output state but quantum operations do more. Quantum operations take a *superposition* state, a complex linear combination of some or all 2^n classical (binary) input states and produce an output *superposition* state. For example, the quantum CNOT gate applied to a pair of control and target qubits $|ct\rangle$ transforms an input superposition $\alpha_1 |00\rangle + \alpha_2 |01\rangle + \alpha_3 |10\rangle + \alpha_4 |11\rangle$ to the output superposition $\alpha_1 |00\rangle + \alpha_2 |01\rangle + \alpha_3 |11\rangle + \alpha_4 |10\rangle = \alpha_1 |00\rangle + \alpha_2 |01\rangle + \alpha_4 |10\rangle + \alpha_3 |11\rangle$.

Physical systems in classical hardware are typically binary. However, in common quantum hardware, such as in superconducting and trapped ion computers, there is an infinite spectrum of discrete energy levels. The qubit abstraction is an artificial approximation achieved by suppressing all but the lowest two energy levels. Instead, the hardware may be configured to manipulate the lowest three energy levels by operating on qutrits. In general, such a computer could be configured to operate on any number of d levels, except as

d increases the number of opportunities for error, termed error channels, increases. Here, we focus on $d = 3$ and later $d = 4$ with which we achieve the desired improvements to the Generalized Toffoli gate and qudit "compression". For a complete guide to superconducting qubits we refer to Krantz et al. [2019].

In a three level system, we consider the computational basis states $|0\rangle$, $|1\rangle$, and $|2\rangle$ for qutrits. A qutrit state $|\psi\rangle$ may be represented analogously to a qubit as $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle + \gamma |2\rangle$, where $\|\alpha\|^2 + \|\beta\|^2 + \|\gamma\|^2 = 1$. Qutrits are manipulated in a similar manner to qubits; however, there are additional gates which may be performed on qutrits.

For instance, in quantum binary logic, there is only a single X gate. In ternary, there are three X gates denoted X_{01}, X_{02}, and X_{12}. Each of these X_{ij} for $i \neq j$ can be viewed as swapping the amplitudes of $|i\rangle$ and $|j\rangle$ and leaving the third basis element unchanged. For example, for a qutrit $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle + \gamma |2\rangle$, applying X_{02} produces $X_{02} |\psi\rangle = \gamma |0\rangle + \beta |1\rangle + \alpha |2\rangle$. Each of these operations' actions can be found in the left state diagram in Figure 2.2.

There are two additional non-trivial operations on a single trit. They are the $+1$ and -1 (sometimes referred to as a $+2$) operations (with $+$ meaning addition modulo 3). These operations can be written as $X_{01}X_{12}$ and $X_{12}X_{01}$, respectively; however, for simplicity, we will refer to them as X_{+1} and X_{-1} operations. A summary of these gates' actions can be found in the right state diagram in Figure 2.2.

When we use qudits with more than three levels, there are many mores gates which can be used depending on d. For a single qudit we have access to every permutation of the d basis states, or $d! - 1$ nontrivial operations, but in practice, many of these operations are unnecessary and only a small number are needed for universal computation. We make use of the increment permutations, denoted X_{+k} where $+$ is addition modulo d, which rotates a state $|i\rangle$ to $|i + k \mod d\rangle$ and the flip permutations denoted X_{ij} which flip or switch the states $|i\rangle$ and $|j\rangle$, leaving all others unchanged. X_{01} is equivalent to the qubit X gate.

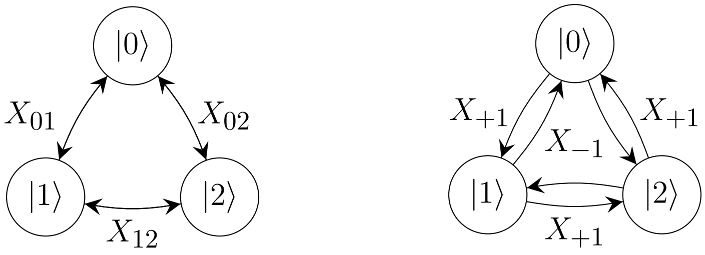


Figure 2.2: The five nontrivial permutations on the basis elements for a qutrit. (Left) Each operation here switches two basis elements while leaving the third unchanged. These operations are self-inverses. (Right) These two operations permute the three basis elements by performing a $+1 \mod 3$ and $-1 \mod 3$ operation. They are each other's inverses.

Other, non-classical, operations may be performed on a single qudit. For example, the Hadamard gate (Nielsen and Chuang [2011]) can be extended to work on qudits in a similar fashion as the X gate was extended. In fact, all single qubit gates, like rotations, may be extended to operate on qudits. In order to distinguish qubit, qutrit, and qudit gates, all non-qubit gates will appear with an appropriate subscript.

Each of these operations can be extended to two qudits as a controlled operation that applies the single-qudit operation conditioned on the control qudit being in a certain state. For example, consider applying an X_{+2} operation on a $d = 4$ level system conditioned on a control qudit being in the $|3\rangle$ state. These controlled qudits have been physically realized and they are universal for qudit computation, as shown by Muthukrishnan and Stroud [2000]. This can be extended to any number of controls but only two-qudit gates can be directly executed on typical quantum hardware; any use of a multi-controlled gate has a decomposition into one and two qudit gates since these gates are universal. We only require a single 3-qubit, 2-controlled gate (Toffoli-like) whose decomposition is given by Di and Wei [2011] into basic one- and two-qubit gates. We represent these gates in circuit diagrams with the control types indicated by circles with the control values inside. The applied gates, specifically the increment (X_{+i}) and flip gates (X_{ij}) will be given as a square labeled with the name of the gate.

One question concerning the feasibility of using higher states beyond the standard two is whether these gates can be implemented and perform the desired manipulations. Qudit gates have been successfully implemented by Di and Wei [2011], Muthukrishnan and Stroud [2000], Klimov et al. [2003], Chi et al. [2022], indicating that it is possible to consider higher level systems apart from qubit only systems.

In order to evaluate an implementation of a quantum circuit, we consider quantum circuit costs. Quantum circuits consist of a sequence of operations, also called gates, applied to a set of input qubits. These circuits do not have fan-in or fan-out and so when represented each horizontal line in the circuit diagram corresponds to a single qubit and time flows from left to right from inputs to outputs. The space cost of a circuit is therefore the number of qubits (or qudits) and this cost is referred to as circuit *width*. Requiring ancilla increases the circuit width and therefore the space cost of a circuit. The time cost for a circuit is the *depth* of a circuit. The depth is the length of the critical path (in number of gates) from input to output.

2.3 Prior Work

2.3.1 Qudits

Qutrits, and more generally qudits, have been been studied in past work both experimentally and theoretically. Experimentally, d as large as 10 has been achieved (including with two-qudit operations) by Kues et al. [2017], and $d = 3$ qutrits are commonly used internally in many quantum systems, including Bækkegaard et al. [2018], Fedorov et al. [2011].

However, in past work, qudits have conferred only an information compression advantage. For example, n qubits can be compressed to $\frac{n}{\log_2(d)}$ qudits, giving only a constant-factor advantage in Pavlidis and Floratos [2017] at the cost of greater errors from operating qudits instead of qubits. Under the assumption of linear cost scaling with respect to d, Greentree et al. [2003], Khan and Perkowski [2007] demonstrated that $d = 3$ is optimal, although as we show in Section 2.7 the cost is generally superlinear in d.

The information compression advantage of qudits has been applied specifically to Grover's search algorithm by Fan [2008], Li et al. [2011], Wang and Perkowski [2011], Ivanov et al. [2012] and to Shor's factoring algorithm by Bocharov et al. [2016]. Ultimately, the trade-off between information compression and higher per-qudit errors has not been favorable in past work. As such, the past research towards building practical quantum computers has focused on qubits.

We introduce qutrit-based, ancilla-free circuits which are *asymptotically* better than equivalent qubit-only, ancilla-free circuits. Unlike prior work, we demonstrate a compelling advantage in both runtime and reliability, thus justifying the use of qutrits.

2.3.2 Generalized Toffoli Gate

We start by focusing on the Generalized Toffoli gate, which simply adds more controls to the Toffoli circuit in Figure 2.1. The Generalized Toffoli gate is an important primitive used across a wide range of quantum algorithms, and it has been the focus of extensive past optimization work. Table 2.1 compares past circuit constructions for the Generalized Toffoli gate to our construction, which is presented in full in Section 2.4.2.

Among prior work, the Gidney [2015], He et al. [2017], and Barenco et al. [1995] designs are all qubit-only. The three circuits have varying trade-offs. While Gidney and Barenco operate at the ancilla-free frontier, they have large circuit depths: linear with a large constant for Gidney and quadratic for Barenco. The Gidney design also requires rotation gates for very small angles, which can pose an experimental challenge. While the He circuit achieves logarithmic depth, it requires an ancilla for each data qubit, effectively halving the effective potential of any given quantum hardware. Nonetheless, in practice, most circuit implemen-

	Depth	Ancilla	Qudit Types	Constants
This Work	$\log n$	0	Controls are qutrits	Small
Gidney [2015]	n	0	Qubits	Large
He et al. [2017]	$\log n$	n	Qubits	Small
Barenco et al. [1995]	n^2	0	Qubits	Small
Wang and Perkowski [2011]	n	0	Controls are qutrits	Small
Lanyon et al. [2008], Ralph et al. [2008]	n	0	Target is $d = n$-level qudit	Small

Table 2.1: Asymptotic comparison of n-controlled gate decompositions. The total gate count for all circuits scales linearly (except for Barenco et al. [1995], which scales quadratically). Our construction uses qutrits to achieve logarithmic depth without ancilla. We benchmark our circuit construction against Gidney [2015], which is the asymptotically best ancilla-free qubit circuit.

tations use these linear-ancilla constructions due to their small depths and gate counts.

As in our approach, circuit constructions from Lanyon et al. [2008], Ralph et al. [2008], and Wang and Perkowski [2011] have attempted to improve the ancilla-free Generalized Toffoli gate by using qudits. Both the Lanyon et al. [2008] and Ralph et al. [2008] constructions, which have been demonstrated experimentally, achieve linear circuit depths by operating the target as a $d = n$-level qudit. Wang and Perkowski [2011] also achieves a linear circuit depth but by operating each control as a qutrit.

Our circuit construction, presented in Section 2.4.2, has similar structure to the He design, which can be represented as a binary tree of gates. However, instead of storing temporary results with a linear number of ancilla qubits, our circuit temporarily stores information directly in the qutrit $|2\rangle$ state of the controls. Thus, no ancilla are needed.

In our simulations, we benchmark our circuit construction against the Gidney [2015] construction because it is the asymptotically best qubit circuit in the ancilla-free frontier zone. We label these two benchmarks as QUTRIT and QUBIT. The QUBIT circuit handles the lack of ancilla by using *dirty* ancilla, which unlike *clean* (initialized to $|0\rangle$) ancilla, can have an unknown initial state. Dirty ancilla can therefore be bootstrapped internally from a quantum circuit. However, this technique requires a large number of Toffoli gates which makes the decomposition particularly expensive in gate count.

Augmenting the base Gidney construction with a single ancilla or dirty ancilla does reduce the constants for the decomposition significantly, although the asymptotic depth and gate counts are maintained. For completeness, we also benchmark our circuit against this augmented construction, QUBIT+ANCILLA. However, the augmented circuit does not operate at the ancilla-free frontier, and it can conflict with parallelism.

2.4 Circuit Construction

In order for quantum circuits to be executable on hardware, they are typically decomposed into single- and two-qudit gates. Performing efficient, low depth, and low gate count decompositions is important in both the NISQ regime and beyond. Our circuits assume all-to-all connectivity, same as prior work. If the quantum device does not support all-to-all connectivity, additional gates would be inserted by the compiler as needed, but this should not change our results.

2.4.1 Key Intuition

To develop intuition for our technique, we first present a Toffoli gate decomposition (the first step of implementation) which lays the foundation for our generalization to multiple controls. In each of the following constructions, all inputs and outputs are qubits, but we may occupy the $|2\rangle$ state temporarily during computation. Maintaining binary input and output allows these circuit constructions to be inserted into any preexisting qubit-only circuits.

In Figure 2.3, a Toffoli decomposition using qutrits is given. A similar construction for the Toffoli gate is known from past work Lanyon et al. [2008], Ralph et al. [2008]. The goal is to perform an X operation on the last (target) input qubit q_2 if and only if the two control qubits, q_0 and q_1, are both $|1\rangle$. First a $|1\rangle$-controlled X_{+1} is performed on q_0 and q_1. This elevates q_1 to $|2\rangle$ iff q_0 and q_1 were both $|1\rangle$. Then a $|2\rangle$-controlled qubit-X gate is applied to q_2. Therefore, X is performed only when both q_0 and q_1 were $|1\rangle$, as desired. The controls are restored to their original states by a $|1\rangle$-controlled X_{-1} gate, which undoes the effect of the first gate. The key intuition in this decomposition is that the qutrit $|2\rangle$ state can be used to store temporary information.

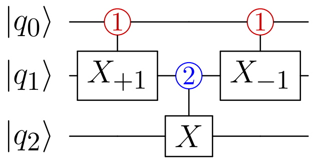


Figure 2.3: A Toffoli decomposition via qutrits. Each input and output is a qubit. The red controls activate on $|1\rangle$ and the blue controls activate on $|2\rangle$. The first gate temporarily elevates q_1 to $|2\rangle$ if both q_0 and q_1 were $|1\rangle$. We then perform the qubit-X operation only if q_1 is $|2\rangle$. The final gate restores q_1 to its original state.

2.4.2 Generalized Toffoli Gate

We now present our circuit decomposition for the Generalized Toffoli in Figure 2.4. The decomposition is expressed in terms of three-qutrit gates (two controls, one target) instead of single- and two-qutrit gates, because the circuit can be understood as purely classical reversible operations at this granularity. For implementation and in our simulation, we use a decomposition from Di and Wei [2011] that requires 6 two-qutrit and 7 single-qutrit physically implementable quantum gates.

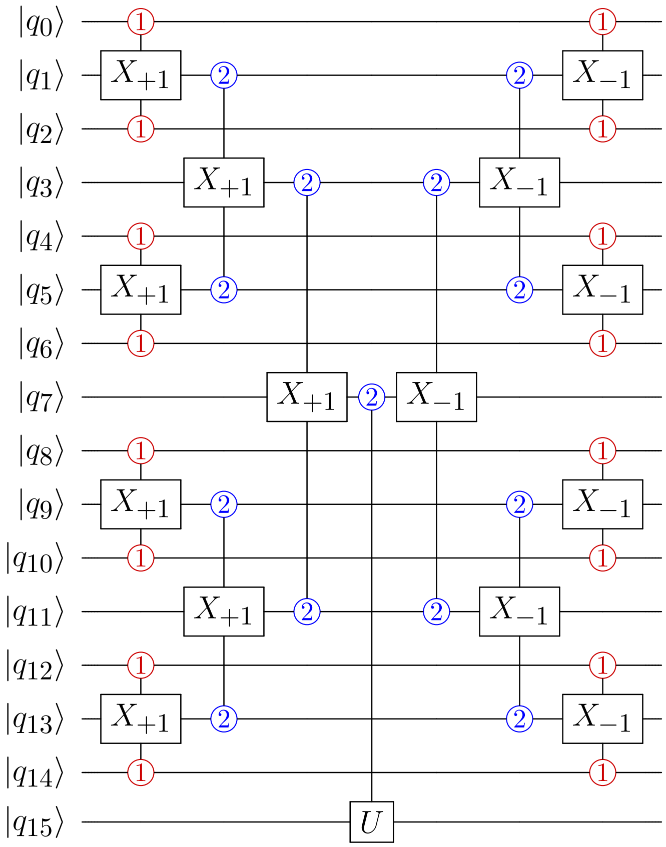


Figure 2.4: Our circuit decomposition for the Generalized Toffoli gate is shown for 15 controls and 1 target. The inputs and outputs are both qubits, but we allow occupation of the $|2\rangle$ qutrit state in between. The circuit has a tree structure and maintains the property that the root of each subtree can only be elevated to $|2\rangle$ if all of its control leaves were $|1\rangle$. Thus, the U gate is only executed if all controls are $|1\rangle$. The right half of the circuit performs uncomputation to restore the controls to their original state. This construction applies more generally to any multiply-controlled U gate. Note that the three-input gates are decomposed into 6 two-input and 7 single-input gates in our actual simulation, as based on the decomposition in Di and Wei [2011].

Our circuit decomposition is most intuitively understood by treating the left half of the circuit as a tree structure. The desired property is that the root of the tree, q_7, is $|2\rangle$ if and only if each of the 15 controls was originally in the $|1\rangle$ state. To verify this property, we observe the root q_7 can only become $|2\rangle$ iff q_7 was originally $|1\rangle$ and q_3 and q_{11} were both previously $|2\rangle$. At the next level of the tree, we see q_3 could have only been $|2\rangle$ if q_3 was originally $|1\rangle$ and both q_1 and q_5 were previously $|2\rangle$, and similarly for the other triplets. At the bottom level of the tree, the triplets are controlled on the $|1\rangle$ state, which are only activated when the even-index controls are all $|1\rangle$. Thus, if any of the controls were not $|1\rangle$, the $|2\rangle$ states would fail to propagate to the root of the tree. The right half of the circuit performs *uncomputation* to restore the controls to their original state.

After each subsequent level of the tree structure, the number of qubits under consideration is reduced by a factor of ~ 2. Thus, the circuit depth is logarithmic in n. Moreover, each qutrit is operated on by a constant number of gates, so the total number of gates is linear in n.

Our circuit decomposition still works in a straightforward fashion when the control type of the top qubit, q_0, activates on $|2\rangle$ or $|0\rangle$ instead of activating on $|1\rangle$. These two constructions are necessary for the Incrementer circuit in Section 2.5.2.

We verified our circuits, both formally and via simulation. Our verification scripts are available on our GitHub (Qutrits Code).

2.5 Application to Algorithms

The Generalized Toffoli gate is an important primitive in a broad range of quantum algorithms. In this section, we survey some of the applications of our circuit decomposition.

2.5.1 Grover's Algorithm

Grover's Algorithm for search over M unordered items requires just $O(\sqrt{M})$ oracle queries. However, each oracle query is followed by a post-processing step which requires a multiply-controlled gate with $N = \lceil \log_2 M \rceil$ controls (Nielsen and Chuang [2011]). The explicit circuit diagram is shown in Figure 2.5.

Our log-depth circuit construction directly applies to the multiply-controlled gate. Thus, we reduce a $\log M$ factor in Grover search time complexity to $\log \log M$ via our ancilla-free qutrit decomposition.

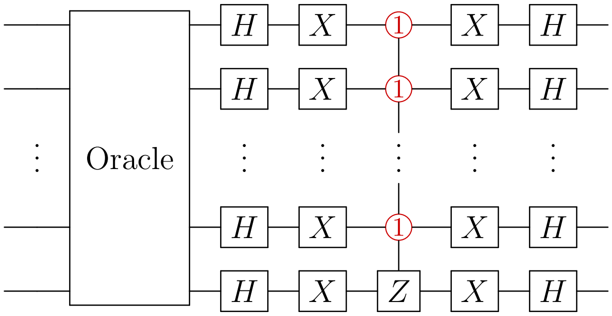


Figure 2.5: Each iteration of Grover Search has a multiply-controlled Z gate. Our logarithmic depth decomposition, reduces a $\log M$ factor in Grover's algorithm to $\log \log M$.

2.5.2 Incrementer

The Incrementer circuit performs the $+1 \mod 2^N$ operation to a register of N qubits. While logarithmic circuit depth is achieved with linear ancilla qubits by Draper [2000], the best ancilla-free incrementers require either linear depth with large linearity constants as in Gidney [2017] or quadratic depth in Barenco et al. [1995]. Using alternate control activations for our Generalized Toffoli gate decomposition, the incrementer circuit is reduced to $O(\log^2 N)$ depth with no ancilla, a significant improvement over past work.

Our incrementer circuit construction is shown in Figure 2.6 for an $N = 8$ wide register.

The multiple-controlled X_{+1} gates perform the job of computing carries: a carry is performed iff the least significant bit generates (represented by the $|2\rangle$ control) and all subsequent bits propagate (represented by the consecutive $|1\rangle$ controls). We present an $N = 8$ incrementer here and have verified the general construction, both by formal proof and by explicit circuit simulation for larger N.

The critical path of this circuit is the chain of $\log N$ multiply-controlled gates (of width $\frac{N}{2}$, $\frac{N}{4}$, $\frac{N}{8}$, $\ldots$) which act on $|a_0\rangle$. Since our multiply-controlled gate decomposition has log-depth, we arrive at a total circuit depth circuit scaling of $\log^2 N$.

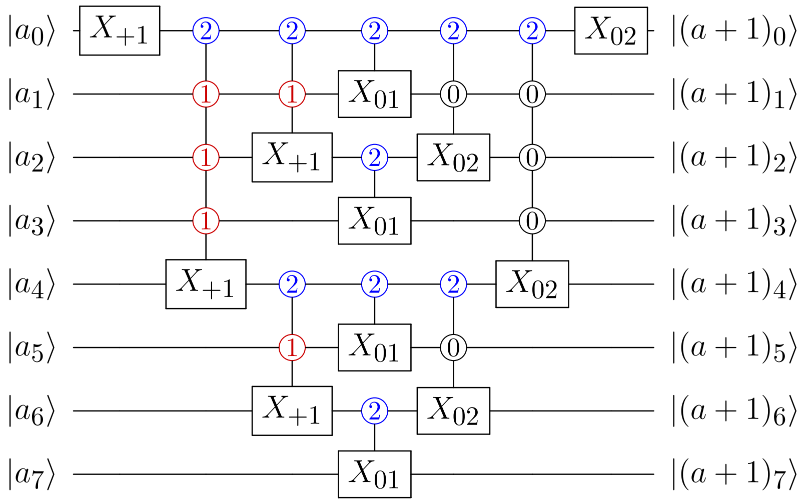


Figure 2.6: Our circuit decomposition for the Incrementer. At each subcircuit in the recursive design, multiply-controlled gates are used to efficiently propagate carries over half of the subcircuit. The $|2\rangle$ control checks for carry generation and the chain of $|1\rangle$ controls checks for carry propagation. The circuit depth is $\log^2 N$, which is only possible because of our log depth multiply-controlled gate primitive.

2.5.3 Arithmetic Circuits and Shor's Algorithm

The Incrementer circuit is a key subcircuit in many other arithmetic circuits such as constant addition. By adding a control to the first and last X gates, this circuit can be used for addition, modular multiplication, and modular exponentiation. Modular exponentiation was shown to be a bottleneck in the runtime for executing Shor's algorithm for factorization by Gidney [2017], Häner et al. [2016]. While a shallower Incrementer circuit alone is not sufficient to reduce the asymptotic cost of modular exponentiation (and therefore Shor's algorithm), it does reduce constants relative to qubit-only circuits. Qudit arithmetic circuits using qudit "compression" are discussed in Section 2.9.

2.5.4 Error Correction and Fault Tolerance

One possible benefit for qutrits in error correction and error mitigation is as an error *flag* (Chao and Reichardt [2018]). Flag qubits are extra qubits that are toggled by carefully placed extra gates in a way that does not change program outcome. If no errors occur during execution, these qubits are always returned to the $|0\rangle$ state but if an error occurs, this often leads to a flag qubit measured in the $|1\rangle$ state, indicating an error occurred. Qutrits can be used in the same way but on resource constrained-devices with no qubits to spare.

The Generalized Toffoli gate has applications to circuits for both error correction (Cory et al. [1998]) and fault tolerance (Dennis [1999]). We foresee two paths of applying these circuits. First, our circuit construction can be used to construct error-resilient *logical qubits* more efficiently. This is critical for quantum algorithms like Grover's and Shor's which are expected to require such logical qubits. In the nearer-term, NISQ algorithms are likely to make use of limited error mitigation. For instance, recent results have demonstrated that error correcting a single qubit at a time for the Variational Quantum Eigensolver algorithm can significantly reduce total error (Otten and Gray [2018]). Thus, our circuit construction is also relevant for NISQ-era error correction.

2.6 Simulator

To simulate our circuit constructions, we developed a qudit simulation library, built on Google's Cirq Python library (Cirq). Cirq is a qubit-based quantum circuit library and includes a number of useful abstractions for quantum states, gates, circuits, and scheduling.

Our work extends Cirq by discarding the assumption of two-level qubit states. Instead, all state vectors and gate matrices are expanded to apply to d-level qudits, where d is a circuit parameter. We include a library of common gates for $d = 3$ qutrits. Our software adds a comprehensive noise simulator, detailed below in Section 2.6.1.

In order to verify our circuits are logically correct, we first simulated them with noise disabled. We wrote a Cirq simulator for classical subcircuits that allows gates to specify their action on classical non-superposition input states without considering full state vectors. Therefore, each classical input state can be verified in space and time proportional to the circuit width. By contrast, Cirq's default simulator relies on a dense state vector representation requiring space and time exponential in the circuit width. Reducing this scaling from exponential to linear dramatically improved our verification procedure, allowing us to verify circuit constructions for all possible classical inputs across circuit sizes up to widths of 14.

Our software is fully open source (Qutrits Code) and the core qudit support has also been added to Cirq.

2.6.1 Noise Simulation

Figure 2.7 depicts a schematic view of our noise simulation procedure which accounts for both gate errors and idle errors, described below. To determine when to apply each gate and idle error, we use Cirq's scheduler which schedules each gate as early as possible, creating a sequence of `Moment`'s of simultaneous gates. During each `Moment`, our noise simulator applies a gate error to every qudit acted on. Finally, the simulator applies an idle error to every qudit. This noise simulation methodology is consistent with previous simulation techniques such as Miller et al. [2018] which accounted for gate errors and Khammassi et al. [2017] for idle errors.

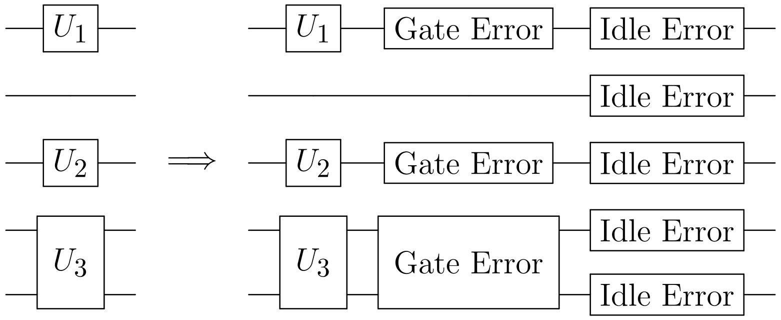


Figure 2.7: This `Moment` comprises three gates executed in parallel. To simulate with noise, we first apply the ideal gates, followed by a gate error noise channel on each affected qudit. This gate error noise channel depends on whether the corresponding gate was single- or two-qudit. Finally, we apply an idle error to every qudit. The idle error noise channel depends on the duration of the `Moment`.

Gate errors arise from the imperfect application of quantum gates. Two-qudit gates are noisier than single-qudit gates (IBM Devices), so we apply different noise channels for the two. Our specific gate error probabilities are given in Section 2.7.

Idle errors arise from the continuous decoherence of a quantum system due to energy relaxation and interaction with the environment. The idle errors differ from gate errors in two ways which require special treatment:

1. Idle errors depend on duration, which in turn depend on the schedule of simultaneous gates (`Moments`). In particular, two-qudit gates take longer to apply than single-qudit gates. Thus, if a `Moment` contains a two-qudit gate, the idling errors must be scaled appropriately. Our specific scaling factors are given in Section 2.7.

2. For the generic model of gate errors, the error channel is applied with probability

independent of the quantum state. This is not true for idle errors such as T_1 amplitude damping, which only applies when the qudit is in an excited state. This is treated in the simulator by computing idle error probabilities during each `Moment`, for each qutrit.

Gate errors are reduced by performing fewer *total gates*, and idle errors are reduced by decreasing the circuit *depth*. Since our circuit constructions asymptotically decrease the depth, this means our circuit constructions scale favorably in terms of asymptotically fewer idle errors.

Our full noise simulation procedure is summarized in Algorithm 1. The ultimate metric of interest is the mean *fidelity*, which is defined as the squared overlap between the ideal (noise-free) and actual output state vectors. Fidelity expresses the probability of overall successful execution. We do not consider initialization errors and readout errors, because our circuit constructions maintain binary input and output, only occupying the qutrit $|2\rangle$ states during intermediate computation. Therefore, the initialization and readout errors for our circuits are identical to those for conventional qubit circuits.

We also do not consider crosstalk errors, which occur when gates are executed in parallel. The effect of crosstalk is very device-dependent and difficult to generalize. Moreover, crosstalk can be mitigated by breaking each `Moment` into a small number of sub-moments and then scheduling two-qutrit operations to reduce crosstalk, as demonstrated in Venturelli et al. [2017], Booth et al. [2018].

2.6.2 Simulator Efficiency

Simulating a quantum circuit with a classical computer is, in general, exponentially difficult in the size of the input because the state of N qudits is represented by a state vector of d^N complex numbers. For 14 qutrits, with complex numbers stored as two 8-byte floats (`complex128` in NumPy), a state vector occupies 77 megabytes.

A naive circuit simulation implementation would treat every quantum gate or `Moment` as

```
$|\Psi\rangle \leftarrow$ random initial state vector
$|\Psi\rangle_{\text{ideal}} =$ circuit applied to $|\Psi\rangle$ without noise
foreach Moment do
    foreach Gate $\in$ Moment do
        $|\psi\rangle \leftarrow$ Gate applied to $|\psi\rangle$
        GateError $\leftarrow$ DrawRand(GateError Prob.)
        $|\psi\rangle \leftarrow$ GateError applied to $|\psi\rangle$
    end
    foreach Qutrit do
        if Moment has 2-qudit gate then
            IdleErrors $\leftarrow$ long-duration idle errors
        else
            IdleErrors $\leftarrow$ short-duration idle errors
        end
        Prob. $\leftarrow [\|\texttt{M}\,|\Psi\rangle\,\|^2$ for M $\in$ IdleErrors$]$
        IdleError $\leftarrow$ DrawRand(Prob.)
        $|\psi\rangle \leftarrow$ IdleError applied to $|\psi\rangle$
        Renormalize($|\psi\rangle$)
    end
end
return $\langle\Psi_{\text{ideal}}|\Psi\rangle^2$, fidelity between ideal & actual output;
```

Algorithm 1: Pseudocode for each simulation trial, given a particular circuit and noise model.

a $d^N \times d^N$ matrix. For 14 qutrits, a single such matrix would occupy 366 terabytes—out of range of simulability. While the exponential nature of simulating our circuits is unavoidable, we mitigate the cost by using a variety of techniques which rely only on state vectors, rather than full square matrices. For example, we maintain Cirq's approach of applying gates by Einstein Summation (Biamonte and Bergholm [2017]), which obviates computation of the $d^N \times d^N$ matrix corresponding to every gate or `Moment`.

Our noise simulator only relies on state vectors, by adopting the quantum trajectory methodology of Brun [2001], Schack and Brun [1996], which is also used by the Rigetti PyQuil noise simulator (Smith et al. [2016]). At a high level, the effect of noise channels like gate and idle errors is to turn a coherent quantum state into an incoherent mix of classical probability-weighted quantum states (for example, $|0\rangle$ and $|1\rangle$ with 50% probability each). The most complete description of such an incoherent quantum state is called the density matrix and has dimension $d^N \times d^N$. The quantum trajectory methodology is a stochastic approach—instead of maintaining a density matrix, only a single state is propagated and the error term is drawn randomly at each timestep. Over repeated trials, the quantum trajectory methodology converges to the same results as from full density matrix simulation (Smith et al. [2016]). Our simulator employs this technique—each simulation in Algorithm 1 constitutes a single quantum trajectory trial. At every step, a specific `GateError` or `IdleError` term is picked, based on a weighted random draw.

Finally, our random state vector generation function was also implemented in $O(d^N)$ space and time. This is an improvement over other open source libraries, Johansson et al. [2011, 2012], which perform random state vector generation by generating full $d^N \times d^N$ unitary matrices from a Haar-random distribution and then truncating to a single column. Our simulator directly computes the first column and circumvents the full matrix computation.

With optimizations, our simulator is able to simulate circuits up to 14 qutrits in width. This is in the range as other state-of-the-art noisy quantum circuit simulations (since 14 qutrits $\approx$ 22 qubits, Chernyavskiy et al. [2018]). While each simulation trial took several minutes (depending on the particular circuit and noise model), we were able to run trials in parallel over multiple processes and multiple machines, as described in Section 2.8.

2.7 Noise Models

In this section, we describe our noise models at a high level. We chose noise models which represent realistic near-term machines. We first present a generic, parametrized noise model roughly applicable to all quantum systems. We then present specific parameters, under the generic noise model, which apply to near-term superconducting quantum computers. Finally, we present a specific noise model for trapped ion quantum computers.

2.7.1 Generic Noise Model

Gate Errors

The scaling of gate errors for a d-level qudit can be roughly summarized as increasing as d^4 for two-qudit gates and d^2 for single-qudit gates. For $d = 2$, there are 4 single-qubit gate error channels and 16 two-qubit gate error channels. For $d = 3$ there are 9 and 81 single- and two-qutrit gate error channels respectively. Consistent with other simulators, Smith et al. [2016], Khammassi et al. [2017], we use the symmetric depolarizing gate error model, which assumes equal probabilities between each error channel. Under these noise models, two-qutrit gates are $(1 - 80p_2)/(1 - 15p_2)$ times less reliable than two-qubit gates, where p_2 is the probability of each two-qubit gate error channel. Similarly, single-qutrit gates are $(1 - 8p_1)/(1 - 3p_1)$ times less reliable than single-qubit gates, where p_1 is the probability of each single-qubit gate error channel.

Idle Errors

Our treatment of idle errors focuses on the relaxation from higher to lower energy states in quantum devices. This is called amplitude damping or T_1 relaxation. This noise channel irreversibly takes qudits to lower states. For qubits, the only amplitude damping channel is from $|1\rangle$ to $|0\rangle$, and we denote this damping probability as λ_1. For qutrits, we also model damping from $|2\rangle$ to $|0\rangle$, which occurs with probability λ_2.

2.7.2 Superconducting QC

We chose four noise models based on superconducting quantum computers expected in the next few years. These noise models comply with the generic noise model above and are thus parametrized by p_1, p_2, λ_1 and λ_2. The λ_i probabilities are derived from two other experimental parameters: the gate time Δt and T_1, a timescale that captures how long a qudit persists coherently.

As a starting point for representative near-term noise models, we consider parameters for *current* superconducting quantum computers. For IBM's public cloud-accessible superconducting quantum computers, we have $3p_1 \approx 10^{-3}$ and $15p_2 \approx 10^{-2}$. The duration of single- and two-qubit gates is $\Delta t \approx 100ns$ and $\Delta t \approx 300ns$ respectively, and the IBM devices have $T_1 \approx 100\mu s$ (IBM Devices, Linke et al. [2017]).

However, simulation for these current parameters indicates an error is almost certain to occur during execution of a modest size 14-input Generalized Toffoli circuit. This motivates us to instead consider noise models for better devices which are a few years away. Accordingly, we adopt a baseline superconducting noise model, labeled as SC, corresponding to a superconducting device which has 10x lower gate errors and 10x longer T_1 duration than the current IBM hardware. This range of parameters has already been achieved experimentally in superconducting devices for gate errors by Barends et al. [2014], Barnes et al. [2016] and for T_1 duration by Reagor et al. [2015], Earnest et al. [2017] independently. Faster gates (shorter Δt) are yet another path towards greater noise resilience. We do not vary gate speeds, because errors only depend on the $\Delta t/T_1$ ratio, and we already vary T_1. In practice however, faster gates could also improve noise-resilience.

We also consider three additional near-term device noise models, indexed to the SC noise model. These three models further improve gate errors, T_1, or both, by a 10x factor. The specific parameters are given in Table 2.2. Our 10x improvement projections are realistic extrapolations of progress in hardware. In particular, Schoelkopf's Law—the quantum analogue of Moore's Law—has observed that T_1 durations have increased by 10x every 3 years for the past 20 years Girvin. Hence, 100x longer T_1 is a reasonable projection for devices that are $\sim$ 6 years away.

Noise Model	$3p_1$	$15p_2$	T_1
SC	10^{-4}	10^{-3}	1 ms
SC+T1	10^{-4}	10^{-3}	10 ms
SC+GATES	10^{-5}	10^{-4}	1 ms
SC+T1+GATES	10^{-5}	10^{-4}	10 ms

Table 2.2: Noise models simulated for superconducting devices. Current publicly accessible IBM superconducting quantum computers have single- and two-qubit gate errors of $3p_1 \approx 10^{-3}$ and $15p_2 \approx 10^{-2}$, as well as T_1 lifetimes of 0.1 ms (IBM Devices, Linke et al. [2017]). Our baseline benchmark, SC, assumes 10x better gate errors and T_1. The other three benchmarks add a further 10x improvement to T_1, gate errors, or both.

2.7.3 Trapped Ion $^{171}Yb^{+}$ QC

We also simulated noise models for trapped ion quantum computing devices. Trapped ion devices are well matched to our qutrit-based circuit constructions because they feature all-to-all connectivity (Brown et al. [2016]), and many ions that are ideal candidates for QC devices are naturally multi-level systems.

We focus on the $^{171}\mathrm{Yb}^+$ ion, which has been experimentally demonstrated as both a qubit and qutrit by Randall et al. [2015, 2018]. Trapped ions are often favored in QC schemes due to their long T_1 times. One of the main advantages of using a trapped ion is the ability to take advantage of magnetically insensitive states known as "clock states". By defining the computational subspace on these clock states, idle errors caused from fluctuations in the magnetic field are minimized—this is termed a DRESSED_QUTRIT, in contrast with a BARE_QUTRIT. However, compared to superconducting devices, gates are much slower. Thus, gate errors are the dominant error source for ion trap devices. We modelled a fundamental source of these errors: the spontaneous scattering of photons originating from the lasers used to drive the gates. The duration of single- and two-qubit gates used in this calculation was $\Delta t \approx 1$ μs and $\Delta t \approx 200$ μs respectively (Brown and Brown [2018]). The single- and two-qudit gate error probabilities are given in Table 2.3.

Noise Model	p_1	p_2
TI_QUBIT	6.4×10^{-4}	1.3×10^{-4}
BARE_QUTRIT	2.2×10^{-4}	4.3×10^{-4}
DRESSED_QUTRIT	1.5×10^{-4}	3.1×10^{-4}

Table 2.3: Noise models simulated for trapped ion devices. The single- and two-qutrit gate error channel probabilities are based on calculations from experimental parameters. For all three models, we use single- and two-qudit gate times of $\Delta t \approx 1$ μs and $\Delta t \approx 200$ μs respectively.

2.8 Simulation Results

Figure 2.8 plots the exact circuit depths for all three benchmarked circuits. The qubit-based circuit constructions from past work are linear in depth and have a high linearity constant. Augmenting with a single borrowed ancilla reduces the circuit depth by a factor of 8. However, both circuit constructions are surpassed significantly by our qutrit construction, which scales logarithmically in N and has a relatively small leading coefficient.

Figure 2.9 plots the total number of two-qudit gates for all three circuit constructions. As noted in Section 2.4, our circuit construction is not asymptotically better in total gate count—all three plots have linear scaling. However, as emphasized by the logarithmic vertical axis, the linearity constant for our qutrit circuit is 70x smaller than for the equivalent ancilla-free qubit circuit and 8x smaller than for the borrowed-ancilla qubit circuit.

Our simulations under realistic noise models were run in parallel on over 100 n1-standard-4 Google Cloud instances. These simulations represent over 20,000 CPU hours, which was sufficient to estimate mean fidelity to an error of $2\sigma < 0.1\%$ for each circuit-noise model pair.

The full results of our circuit simulations are shown in Figure 2.10. All simulations are for the 14-input (13 controls, 1 target) Generalized Toffoli gate. We simulated each of the three circuit benchmarks against each of our noise models (when applicable), yielding the 16 bars in the figure.

Figure 2.10 demonstrates that our QUTRIT construction (orange bars) significantly outperforms the ancilla-free QUBIT benchmark (blue bars) in fidelity (success probability) by more than 10,000x.

For the SC, SC+T1, and SC+GATES noise models, our qutrit constructions achieve between 57–83% mean fidelity, whereas the ancilla-free qubit constructions all have almost 0% fidelity. Only the lowest-error model, SC+T1+GATES achieves modest fidelity of 26% for the QUBIT circuit, but in this regime, the qutrit circuit is close to 100% fidelity.

The trapped ion noise models achieve similar results—the DRESSED_QUTRIT and the BARE_QUTRIT achieve approximately 95% fidelity via the QUTRIT circuit, whereas the TI_QUBIT noise model has only 45% fidelity. Between the

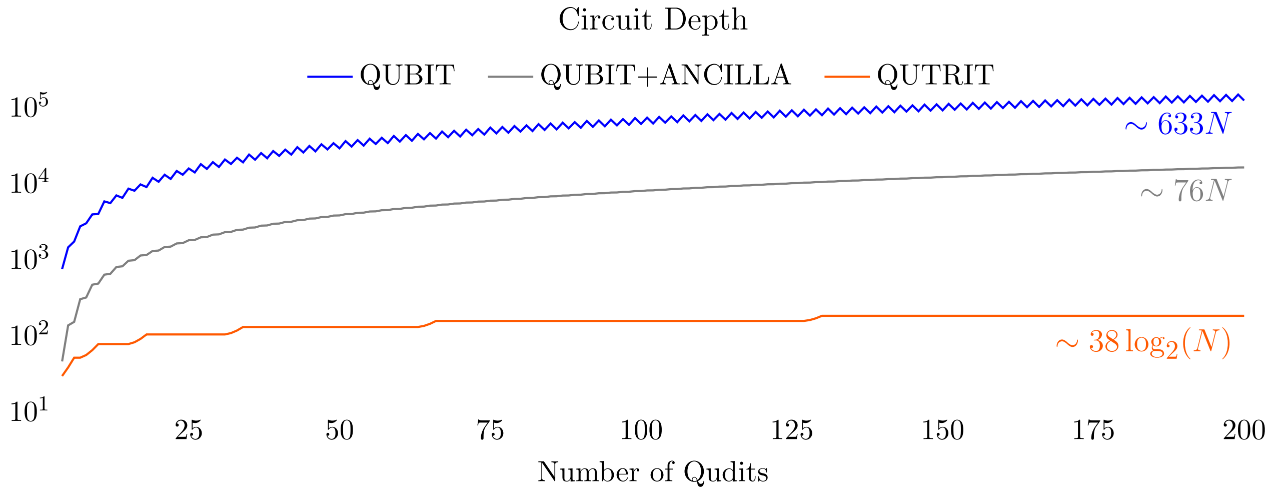


Figure 2.8: Exact circuit depths for all three benchmarked circuit constructions for the N-controlled Generalized Toffoli up to $N = 200$. Both QUBIT and QUBIT+ANCILLA scale linearly in depth and both are bested by QUTRIT's logarithmic depth.

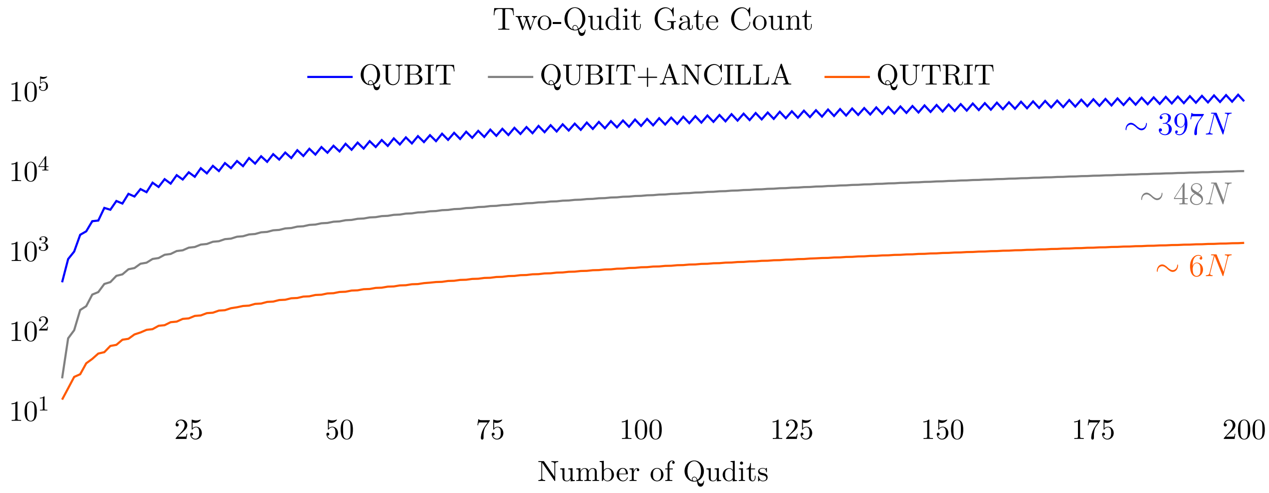


Figure 2.9: Exact two-qudit gate counts for the three benchmarked circuit constructions for the N-controlled Generalized Toffoli. All three plots scale linearly; however the QUTRIT construction has a substantially lower linearity constant.

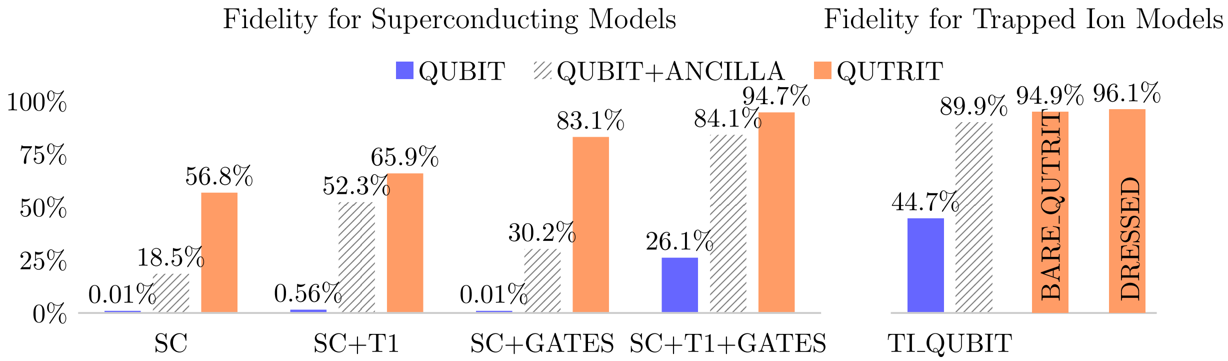


Figure 2.10: Circuit simulation results for all possible pairs of circuit constructions and noise models. Each bar represents 1000+ trials, so the error bars are all $2\sigma < 0.1\%$. Our QUTRIT construction significantly outperforms the QUBIT construction. The QUBIT+ANCILLA bars are drawn with dashed lines to emphasize that it has access to an extra ancilla bit, unlike our construction.

dressed and bare qutrits, the dressed qutrit exhibits higher fidelity than the bare qutrit, as expected. Moreover, the dressed qutrit is resilient to leakage errors, so the simulation results should be viewed as a lower bound on its advantage over the qubit and bare qutrit.

Based on these results, trapped ion qutrits are a particularly strong match to our qutrit circuits. In addition to attaining the highest fidelities, trapped ions generally have all-to-all connectivity (Brown et al. [2016]) within each ion chain, which is critical as our circuit construction requires operations between distant qutrits.

The superconducting noise models also achieve good fidelities. They exhibit a particularly large advantage over ancilla-free qubit constructions because idle errors are significant for superconducting systems, and our qutrit construction significantly reduces idling (circuit depth). However, most superconducting quantum systems only feature nearest-neighbor or short-range connectivity. Accounting for data movement on a nearest-neighbor-connectivity 2D architecture would expand the qutrit circuit depth from $\log N$ to $\sqrt{N}$ (since the distance between any two qutrits would scale as $\sqrt{N}$). However, Naik et al. [2017] has experimentally demonstrated fully-connected superconducting quantum systems via random access memory.

Such systems would also be well matched to our circuit construction.

For completeness, Figure 2.10 also shows fidelities for the QUBIT+ANCILLA circuit benchmark, which augments the ancilla-free QUBIT circuit with a single dirty ancilla. Since QUBIT+ANCILLA has linearity constants $\sim$ 10x better than the ancilla-free qubit circuit, it exhibits significantly better fidelities. While our QUTRIT circuit still outperforms the QUBIT+ANCILLA circuit, we expect a crossing point where augmenting a qubit-only Generalized Toffoli with enough ancilla would eventually outperform QUTRIT. However, we emphasize that the gap between an ancilla-free and constant-ancilla construction for the Generalized Toffoli is actually a fundamental rather than an incremental gap, because:

- Constant-ancilla constructions prevent circuit parallelization. For example, consider the parallel execution of N/k disjoint Generalized Toffoli gates, each of width k for some constant k. An ancilla-free Generalized Toffoli would pose no issues, but an ancilla-augmented Generalized Toffoli would require $\Theta(N/k)$ ancilla. Thus, constant-ancilla constructions can impose a choice between serializing to linear depth or regressing to linear ancilla count. The Incrementer circuit in Figure 2.6 is a concrete example of this scenario—any multiply-controlled gate decomposition requiring a single clean ancilla or more than 1 dirty ancilla would contradict the parallelism and reduce runtime.

- Even if we only consider serial circuits, given the exponential advantage of certain quantum algorithms, there is a significant practical difference between operating at the ancilla-free frontier and operating just a few data qubits below the frontier.

While we only performed simulations up to 14 inputs in width, we would see an even bigger advantage in larger circuits because our construction has asymptotically lower depth and therefore asymptotically lower idle errors. We also expect to see an advantage for the circuits in Section 2.5 that rely on the Generalized Toffoli, although we did not explicitly simulate these circuits.

2.9 Qubit-Qudit Compression

We see, using qudits as temporary storage can benefit our circuit for the Generalized Toffoli but can the abstraction of using extra levels as temporary storage be useful in other contexts? In this section, we show how to re-encode, or "compress" idle data to free-up extra workspace ancilla, without requiring a larger quantum computer.

Typically, when using a higher radix computing paradigm, we express a circuit entirely in the specified base, that is all inputs and outputs are in the designated radix. An alternative approach is to fix the input and output radix but allow the use of higher level states temporarily during the computation, i.e. we are permitted to occupy any level up to a specified d during a computation with the guarantee that we return to the specified radix.

What does this gain for us? It is known that by simply fully encoding a computation into a higher radix we obtain a constant space and time advantage over binary-only circuits. However, as we showed earlier, the use of these higher states can act as temporary storage, similar to the use of an ancilla, and can convey an asymptotic reduction in circuit depth. This circuit construction suggests we can obtain better circuits while using fewer qubits by accessing higher states temporarily.

We take this a step further and *generate* ancilla temporarily out of input qubits in order to take advantage of previously known efficient binary circuit decompositions like that of Draper et al. [2006]. Using this method, we can reduce the number of external ancilla needed from $O(n)$ to 0 while keeping the same asymptotic circuit depth. To do this, we allow subsets of qubits to temporarily store higher values, becoming qudits, to store the information of many qubits within a few qudits. As a concrete example, consider three qubits. There are $2^3 = 8$ total basis states while for two qutrits there are $3^2 = 9$ basis states. Therefore all the information of 3 qubits can be stored in two qutrits and the third qubit can be left in a chosen state, $|0\rangle$, a clean ancilla. We refer to this process as *compression*, that is storing the information of many qu*bits* in a smaller number of qu*dits*. While a better term for this process might be *re-encoding* with a different radix, its behavior from a systems perspective is similar to lossless compression of data to save memory.

We consider various reversible compression schemes labeled x-y-z compression, where x is the radix of the input qudits, y is the radix of the output qudits, and z is the number of ancilla generated. Such operations exist if $x^m \leq y^n$ with $0 < n < m$ and $m - n = z$ for some integers m, n, the number of input qudits and the number of non-ancilla outputs, respectively. Put more simply, these proposed compression circuits exist if the number of basis states of the inputs is fewer than the number of basis states of the non-ancilla outputs and the number of non-ancilla outputs is strictly smaller than the number of inputs. Ideally, we choose compression schemes with a good compression ratio, i.e. those for which $x^m / y^n \approx 1$.

In this section, we consider 2-3-1 and 2-4-1 compression as methods of generating ancilla for simplicity. Many other schemes such as 2-8-2 and 3-9-1 are possible but require increasingly complex compression circuits.

2.9.1 2-3-1 Compression

In 2-3-1 compression we take as input three qubits and output 2 qutrits and a single ancilla, a qubit guaranteed to be in the $|0\rangle$ state. First, consider the truth table of Table 2.4. We note the partial function represented by this truth table is invertible, implying there exists a reversible circuit that realizes it. The third output, C', is guaranteed to be in the $|0\rangle$ state, an ancilla. By storing qubit information used infrequently we can generate an extremely useful ancilla to be used elsewhere in the circuit. Because we ensure all inputs are binary, we do not need to consider the inputs with value 2 to the ternary circuit. An example circuit realizing this truth table is given in Figure 2.11. When a compression circuit of this type is applied, we need to keep track of which pair of qutrits encodes the three qubits, in order. When the compressed data is needed, we can decompress by applying the inverse of this function. The inverse circuit is simply the gates in reverse order with $+1$ replaced with -1.

A	B	C	A’	B’	C’
0	0	0	0	0	0
0	0	1	2	2	0
0	1	0	0	1	0
0	1	1	0	2	0
1	0	0	1	0	0
1	0	1	2	1	0
1	1	0	1	1	0
1	1	1	1	2	0

Table 2.4: Truth table for 2-3-1 Compression

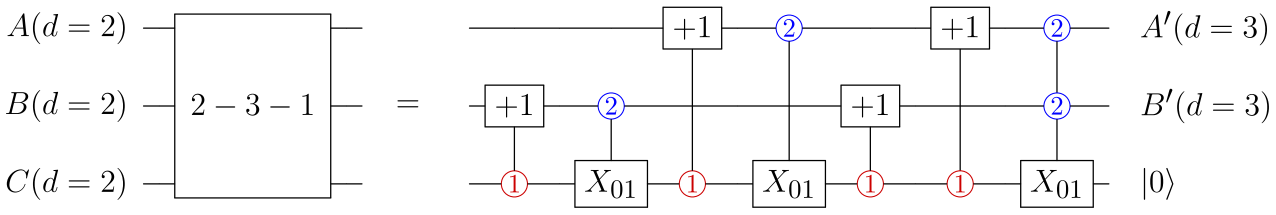


Figure 2.11: The compression of 3 qubits into 2 qutrits and an ancilla, $|0\rangle$. All $+1$ gates are done modulo 3. Using a sequence of qutrit gates, we can transform three input qubits into the desired ancilla. When A, B and C are not going to be used for a long time in the circuit, they can be temporarily repurposed as an ancilla bit elsewhere in the circuit. When we want to operate on these stored bits, we run the inverse of this circuit using *any* ancilla for the third qubit.

Notably, this inversion requires an ancilla as input. To retrieve the information, the inverse should be applied taking in any free ancilla and then the stored bits can be computed on as normal.

This circuit, while accomplishing what is desired, can be rather inefficient. For example, in architectures with limited connectivity this circuit requires some number of expensive communication operations since every input qubit must be adjacent at some point. Furthermore, this circuit requires the use of a two-controlled qutrit gate which is typically decomposed into a sequence of 6 two-qutrit gates and 10 single-qutrit gates as in Di and Wei [2011]. In total this compression requires 22 gates, 12 two-qutrit and 10 single-qutrit gates.

2.9.2 2-4-1 Compression

While 2-3-1 compression required a fairly substantial number of gates, the 2-4-1 compression circuit can convert qubit inputs into ancilla more simply and with few gates. This does not come for free. In quantum computing, we subject our computation to a greater probability of error by using higher radix gates and by persisting for longer durations in higher energy states. In Table 2.5, we show that two qubits can be compressed into a single ququart and one ancilla. 2-4-1 compression is simpler than 2-3-1 compression because 2^2 states fit evenly in a single ququart with 4 states. In Figure 2.12, we show a compression circuit using only 3 two-ququart gates in total, a substantial reduction over the 2-3-1 counterpart. In the next section, we show how compression and decompression can be used to design efficient circuits requiring no ancilla.

A	B	A’	B’
0	0	0	0
0	1	2	0
1	0	1	0
1	1	3	0

Table 2.5: Truth table for 2-4-1 Compression

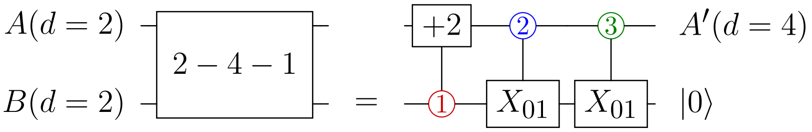


Figure 2.12: The compression of 2 qubits into a single ququart and generating an ancilla, $|0\rangle$. The +2 gate here is done modulo 4. This operation takes as input two qubits, A and B, and produces a single ququart and an ancilla $|0\rangle$. To do this, we need only 3 two-ququart gates. Similarly, to retrieve the stored information, we can do the inverse of this operation using *any* ancilla for the second qubit.

2.10 A+B Adder

We now present our $A+B$ adder. This circuit takes as input two equal-sized registers of qubits, A and B, and optionally carry-in or carry-out bits. This decomposition uses no ancilla and instead generates ancilla locally when needed by sub-components. In prior work, to achieve a logarithmic depth decomposition, $O(n)$ many ancilla were required where n is the size of the input register. We will demonstrate how this efficient decomposition can be used along with our new compression technique to obtain an $O(\log n)$ depth decomposition of the same adder in-place without the extra use of ancilla.

We first briefly review the work of Draper et al. [2006] which gives a qubit-only in-place adder with ancilla which we will refer to as $(A+B)_2$. We give the decomposition for registers of size 4 in Figure 2.13. One of the key contributions of this prior work is to demonstrate how, in logarithmic depth, the carry bits could be computed and used (and subsequently uncomputed to restore input ancilla back to the $|0\rangle$ state). This decomposition requires $2m - w(m) - \lfloor \log m \rfloor$ ancilla, where $w(m)$ is the number of 1's appearing in the binary expansion of the number of inputs, m. We will use this number later to determine how many ancilla to generate via compression. This same prior work demonstrates several variants of this circuit. We require those with either a carry-in bit, a carry-out bit, or both.

We will now present our decomposition shown in Figure 2.14. Let $A = (a_1 a_2 \dots a_n)$ and $B = (b_1 b_2 \dots b_n)$ be the input registers with a_1, b_1 the least significant bits of each register. We divide these registers into c blocks $R_1, \dots, R_c$ each of size $2n/c$. We assume for clarity that n is a multiple of c but our constructions will work for any n, with one additional block containing the remaining $2(n \mod c)$ qubits. Take

$$R_i = (a_{(i-1)(c/n)+1} b_{(i-1)(c/n)+1} \dots a_{i(c/n)} b_{i(c/n)})$$

then notice for $i > 1$ we can perform an addition circuit $(A+B)_2$ with carry-in and carry-out on block R_i in $O(\log(n/c)) = O(\log n)$ depth by generating the proper number of ancilla out of the other input qubits, specifically $2(n/c) - w(n/c) - \lfloor \log n/c \rfloor$ ancilla. We will assume a worst case scenario of $2n/c$ ancilla to simplify the analysis. Suppose we are performing $(A+B)_2$ on block R_i while every other block is unused. We can perform compression on the currently unused qubits in all other blocks $\{R_j | j \neq i\}$ to obtain generated ancilla which can then be used by the current adder subcircuit.

Recall 2-3-1 compression takes 3 qubits and outputs a single ancilla. Let a 2-3-1 *Compress circuit* be a circuit which takes any number of qubits m as input and applies 2-3-1 compression to triplets resulting in $\lfloor m/3 \rfloor$ ancilla. Then applying 2-3-1 compression to all qubits in $\{R_j | j \neq i\}$ we obtain $\lfloor (c-1)2n/3c \rfloor$ ancilla. We now have constraints on what the constant c

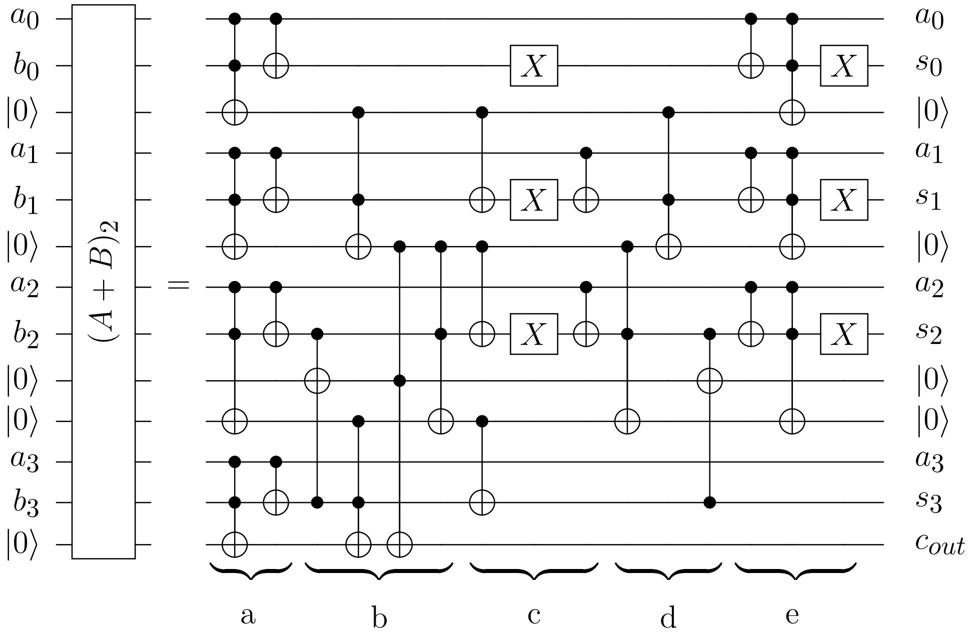


Figure 2.13: An adder circuit from Draper et al. [2006] on two four-bit registers A and B with a carry-out bit using ancilla. The sum S is computed in-place on register B while A is untouched and the ancilla are restored to $|0\rangle$. We use this as a sub-component of our general decomposition. Each of the ancilla in this circuit can be generated from other input qubits not shown here via our compression circuits. Part a of the circuit computes carry, generate, and propagate for each bit position. Part b computes the carry-in for every bit position. Part c does the addition, storing the output in register B. Parts d and e uncompute b and a respectively, restoring the ancilla back to $|0\rangle$.

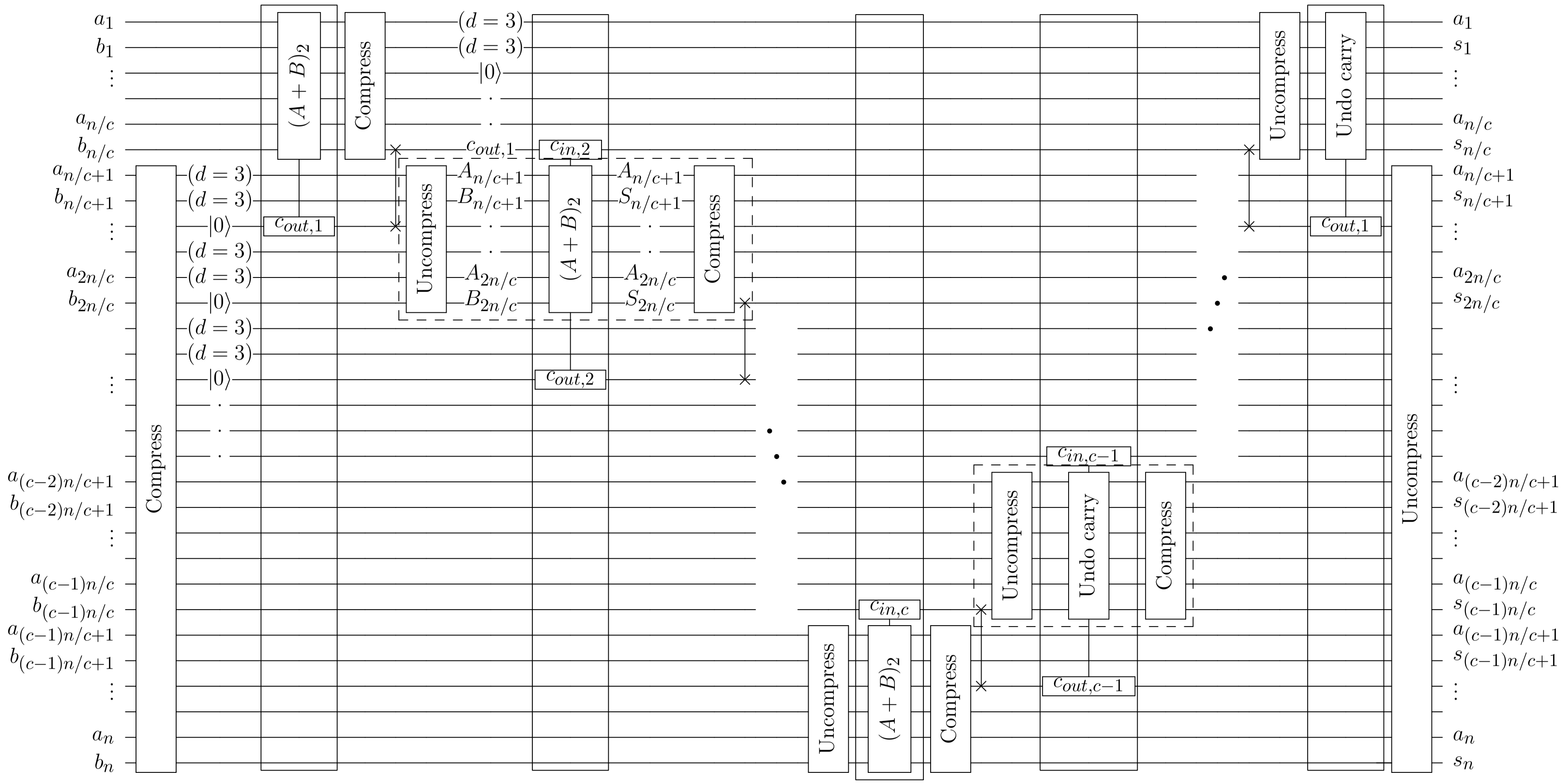


Figure 2.14: Our $A + B$ adder that uses no external ancilla. The variant shown here for $c = 5$ uses 2-3-1 compression to generate one ancilla (marked as $|0\rangle$) for every three unused qubits, storing their values in two qutrits (marked as $d = 3$). A box is drawn around every $(A + B)_2$ and Undo carry gate to indicate that they use all the generated ancilla across the circuit. $c_{out,i}$ or $c_{in,i}$ is included on some of the gates to indicate when the carry-in and carry-out versions are used and on which ancilla the carry-out is stored. The SWAP gates (pairs of × in the diagram) simply move a carry-out bit to another ancilla where it is used as the next carry-in. The two blocks of gates shown with dashed lines are repeated $c - 2 = 3$ times along the diagonal indicated. If 2-4-1 compression is used, an ancilla is generated for every two unused qubits so only $c = 4$ blocks are needed. The depth of this circuit is $O(\log n)$.

should be for our decomposition to be feasible. That is we must have $\lfloor (c-1)2n/3c \rfloor \geq 2n/c$. Because we must store intermediate carry values between each $(A+B)_2$, we will actually require an additional $c-1$ ancilla, giving us $\lfloor (c-1)2n/3c \rfloor \geq 2n/c + c - 1$. By solving the inequality, this implies our construction is feasible for $c = 5$ and $n \geq 30$. An alternative adder that is ancilla-free but does not scale well asymptotically, like the $O(n)$-depth adder by Cuccaro et al. [2004], may be used where our construction is infeasible on small problem sizes with $n < 30$.

The circuit construction now goes as follows, first considering the case when we have no carry-in and no carry-out. To add in these additional features requires only minor adjustments, discussed later. First, we compress the qubits in blocks $\{R_j | j \neq 1\}$. Then we apply $(A+B)_2$ with carry-out to the block R_1 using the newly generated ancilla. The compression block is constant depth ($O(1)$) and the adder is logarithmic depth ($O(\log(n/c)) = O(\log n)$). The qubits $b_1, \ldots, b_{n/c}$ now store the first n/c bits of the addition, $s_1, \ldots, s_{n/c}$. Also note the adder circuit restores all ancilla (except the carry-out) to $|0\rangle$. Then, apply a compression block to R_1. Swap the carry-out, $c_{out,1}$, to any of the ancilla generated to hold on to whether a carry should be applied to the next block (these carries are where the additional $c-1$ term come from above). Next, we uncompress all of the bits in R_2 so we can apply $(A+B)_2$ with carry-out *and* carry-in ($c_{in} = c_{out,1}$) to block R_2 using the other generated ancilla. We repeat this process until the last block, R_c. In this case, since we do not have a carry-out bit we apply $(A+B)_2$ with only carry-in ($c_{in} = c_{out,c-1}$).

We have now computed the sum $A + B$ and now must cleanup the intermediate carry bits. This can be done by working in reverse to uncompute each carry-out without undoing the addition. One intuitive way would be to simply apply the inverse of the $(A+B)_2$ circuit we applied to block R_{c-1} which will uncompute the addition and $c_{out,c-1}$ and then re-apply it *without* carry-out. Now the ancilla storing $c_{out,c-1}$ is restored to $|0\rangle$. We repeat this process on each of the blocks in reverse order. Finally, after $c_{out,1}$ has been uncomputed and the ancilla restored to $|0\rangle$, we uncompress all of the qubits. The resulting output will be the sum S in register B with register A left unchanged from the input.

Uncomputing the intermediate carry-out bits can be improved dramatically by noticing that by applying the inverse of $(A+B)_2$ with carry-in and carry-out and the subsequently applying $(A+B)_2$ with only carry-in is unnecessary. Instead we can uncompute the carry-out by only applying the inverse of the second half of $(A+B)_2$ with carry-out and then executing the second half of $(A+B)_2$ with a few extra gates in Figure 2.13d to cancel the carry-out.

Earlier, we show our decomposition only works when $c = 5$ using 2-3-1 compression. However, due to page size constraints, we do not show some of the repeated blocks in Figure 2.14. The block of gates surrounded by a dashed line is simply repeated in a block diagonal pattern indicated by the ellipsis. If we instead used 2-4-1 compression, the factor of 3 in the earlier inequality would be replaced with 2 making $c = 4$ feasible with a constraint of $n \geq 12$.

Our decomposition performs addition in-place with zero ancilla, taking advantage of qutrits (qudits in general) to obtain ancilla instead of extra qubits for ancilla. Each of the $(A+B)_2$ blocks has depth $O(\log n)$ for input register size n and we perform only a constant $2c-1$ of them so our decomposition also has $O(\log n)$ depth.

2.10.1 Carry-in and Carry-out

We can extend the above decomposition to allow for carry-in quite simply. When computing the $(A+B)_2$ and Undo carry on R_1 we simply use the $(A+B)_2$ circuit with carry-in. Similarly, we can allow for carry-out by simply substituting an $(A+B)_2$ with carry-in *and* carry-out on block R_c.

2.10.2 +K Adder

The method used to construct the $A + B$ adder shown above can be applied to any circuit that can be divided into blocks while only needing to pass a constant number of bits to the input of the following block. One example that follows from $A + B$ is the $+K$ adder. The $+K$ adder acts on a single register of qubits B and computes the sum $B + K$ in-place where K is a classical constant known when creating the circuit.

The design of our $+K$ adder will use as subcircuits the $(+K)_2$ circuit derived from $(A + B)_2$ from Draper et al. [2006] and described earlier. The design of $(+K)_2$ is the same as $(A + B)_2$ except the qubits of register A are removed and all CNOT gates with a control on a_i are removed and only replaced with X gates if $k_i = 1$. Similarly, the Toffoli gates (controlled-controlled-not gates) are removed and replaced with CNOT gates in the same way. Depending on the value of K, some of the ancilla may also be removed but in the worst case, $(+K)_2$ may still require $2n/c - w(n/c) - \lfloor \log n/c \rfloor - 1$ ancilla for input size n/c which we upper bound by $2n/c$. The circuit still has $O(\log n)$ depth.

We use the same diagonal block structure as $A + B$ but now we define

$$R_i = (b_{(i-1)(c/n)+1} \cdots b_{i(c/n)})$$

At step i, the number of ancilla generated by applying 2-3-1 compression to all qubits in $\{R_j | j \neq i\}$ is $\lfloor (c-1)n/3c \rfloor$. From this, we obtain the inequality $\lfloor (c-1)n/3c \rfloor \geq 2n/c + c - 1$ which determines when there are enough unused qubits to generate the required ancilla. The extra $c - 1$ ancilla are needed to store intermediate carry values. When we solve this inequality, we find that $c = 8$ blocks are required and the circuit will only have enough ancilla when $n \geq 168$. Both the number of blocks and the minimum n are larger than for $A + B$ because the input to $+K$ is only a single register so the ancilla required per input qubit is doubled, resulting in a higher minimum n.

2-3-1 compression is not the only option. If we use 2-4-1 compression instead, more ancilla can be generated per input qubit and we obtain the inequality $\lfloor (c-1)n/2c \rfloor \geq 2n/c + c - 1$. The solution to this tells us that the minimum $c = 6$ and we can use the circuit for $n \geq 60$.

2.11 Discussion and Summary

We have shown a new use of qudits in circuit designs, to generate ancilla in-place, and its application to the class of quantum circuits that can be split into blocks. We give a new construction for an in-place addition circuit that uses no ancilla but still obtains the same $O(\log n)$ asymptotic depth as the qubit circuit it was based on that needed $O(n)$ ancilla. The new circuit can be used as a drop-in replacement in algorithms to use significantly fewer total qubits. These results should encourage further use of the temporary-qudit abstraction in qudit-assisted quantum computing.

A number of useful quantum circuits, especially arithmetic circuits, make extensive use of multiply-controlled gates. However, these circuits are typically pre-compiled into single- and two-qubit gates using one of the decompositions from prior work, usually one that involves ancilla qubits. Revisiting these arithmetic circuits from first principles, with our qutrit circuit as a new tool, could yield novel and improved circuits like our Incrementer circuit in Section 2.5.2 and Adder circuit in Section 2.10.

It still remains to be seen what the most intuitive way for quantum programmers to use qudits. We have only shown hand-designed subroutines and compression strategies to use temporary qutrits. The hand-designed generalized Toffoli implementation makes excellent use of one additional logical state and, while hand-optimization can be a good way to squeeze performance out of resource-constrained devices, codifying manual strategies into our compilers can have wider performance benefit and free most programmers to think at a higher level. The qubit "compression" strategy shows benefit in the design of the quantum adder arithmetic circuit, indicating that this strategy could have wider uses. For example, a compiler could intelligently "compress" binary quantum data (stored in groups of idle qubits) into smaller groups of qutrits (using the $\log_2(3)$ compression ratio) or qudits ($\log_2(d)$ ration). This has the potential to automate the benefits of temporary qutrits to all quantum programs.

Our circuit constructions and compression strategies point towards the benefit of temporary qudit abstraction. Researchers will undoubtedly find more uses for qudits as a form of temporary data storage, enabled by this new way of thinking about quantum data beyond the binary representation.

CHAPTER 3

SPATIALLY LOCAL MEMORY

3.1 Introduction

From the previous chapter, when we abstract multi-level qudits as a two-level qubit with additional storage we enable improvements in new circuit designs and design strategies. In this chapter, we look at a different form of storage, viewed through the lens of spacial constrains in 2D and 3D. These constraints come from 2D chip limitations and the 3D world we live in due to the fragility of qubit-to-qubit communications.

Here we focus on error correction, the underlying architecture needed for future fault-tolerant quantum computing. We are currently in the NISQ (Noisy Intermediate-Scale Quantum, Preskill [2018]) era where great progress has been made at the software level such as improved compilation procedures reducing required overhead for program execution. However, these machines will be too small for error correction and unable to run large-scale programs due to unreliable qubits. The ultimate goal is to construct fault-tolerant machines capable of executing thousands of gates and, in the long-term, to execute large-scale algorithms such as Shor [1997] and Grover [1996] with speedups over classical algorithms. There are a number of promising error correction schemes which have been proposed such as the color code from Landahl et al. [2011] or the surface code from Fowler et al. [2012a], Horsman et al. [2012], Gidney and Ekerå [2019]. The surface code is a particularly appealing candidate because of its low overhead, high error threshold, and its reliance on few nearest-neighbor interactions in a 2D array of qubits, a common feature of superconducting transmon qubit hardware. In fact, Google's next milestone is to demonstrate error corrected qubits (Arute et al. [2019], Martinis [2019]).

Current architectures for both NISQ and fault-tolerant quantum computers make no distinction between the memory and processing of quantum information (typically represented in qubits). While currently viable, as larger devices are built, the engineering challenges of scaling up to hundreds of qubits becomes readily apparent. For transmon technology used by Google, IBM, and Rigetti, some of these issues include fabrication consistency and crosstalk during parallel operations. Every qubit needs dedicated control wires and signal generators which fill the refrigerator the device runs in. To scale to the millions of qubits needed for useful fault-tolerant machines like in Gidney and Ekerå [2019], we need a some kind of memory-like abstraction to manage the massive scale these devices will need to achieve. Memory, in essence, decouples the amount of data storage (qubit-count) from the scale of the instruction processor (transmon-count or laser waveguide size, this is classically the CPU). Finding the most appropriate memory-like abstraction will enable quantum computers to scale more effectively.

In this chapter, we evaluate a recently realized qubit memory technology which stores quantum data in a superconducting cavity local to each qubit (Naik et al. [2017]). This technology, while new, is expected to become competitive with existing transmon devices. Stored in cavity, qubits have a significantly longer lifetime (coherence time) but must be loaded into a transmon for computation. This longer lifetime can increase the total amount of computation before data is lost to errors. Although the basic concept of a compute qubit and associated memory has been demonstrated experimentally, the contribution of the chapter is to design and evaluate a system-level organization of these components within the context of a novel surface code embedding and fault-tolerant quantum operations. We provide a proof of concept in the form of a practical use case motivating more complex experimental demonstrations of larger systems using this technology.

Our proposed 2.5D memory-based design is a typical 2D grid of transmons with memory added as shown in Figure 3.1. This can be compared with the traditional 2D error correction implementation in Figure 3.2, where the checkerboards represent error-corrected logical qubits. The logical qubits in this system are stored at unique virtual addresses in memory

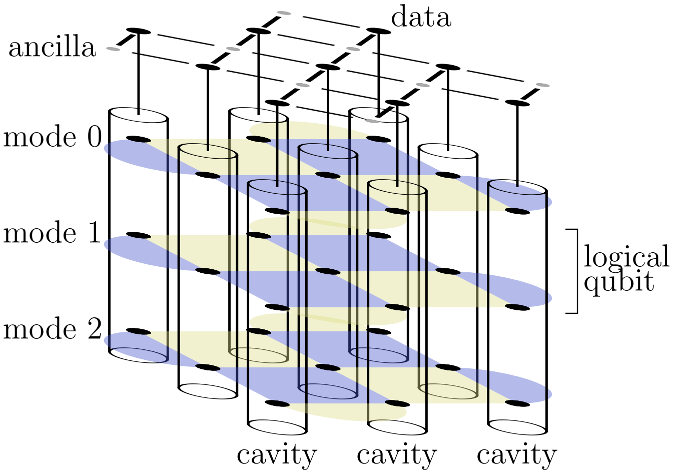


Figure 3.1: Our fault-tolerant architecture with random-access memory local to each transmon. On top is the typical 2D grid of transmon qubits. Attached below each data transmon is a resonant cavity storing error-prone data qubits (shown as black circles). This pattern is tiled in 2D to obtain a 2.5D array of logical qubits. Our key innovation here is storing the qubits that make up each logical qubit (shown as checkerboards) across many cavities to enable efficient computation.

cavities when not in use. They are loaded to a physical address in the transmons and made accessible for computation on request and are periodically loaded to correct errors, similar to DRAM refresh. This design allows for more efficient operations such as the transversal CNOT between logical qubits sharing the same physical address, i.e. co-located in the same cavities. This is not possible on the surface code in 2D which requires methods such as braiding or lattice surgery for a CNOT operation.

We introduce two embeddings of the 2D surface code to this new architecture that spread logical qubits across many cavities. Despite serialization due to memory access, we are able to store and error-correct stacks of these logical qubits. Furthermore, we show surface code operations via lattice surgery can be used unchanged in this new architecture while also enabling a more efficient CNOT operation. Similarly, we are able to use standard and architecture-specific magic-state distillation protocols like Litinski [2019b] in order to ensure universal computation. Magic-state distillation is a critical component of error-corrected algorithms so any improvement will directly speed up algorithms including Shor's and Grover's.

We discuss several important features of any proposed error correction code, such as the threshold error rate (below which the code is able to correct more errors than its execution causes), the code distance, and the number of physical qubits to encode a logical qubit. In many codes, the number of physical qubits can be quite large. We develop an embedding from the standard representation to this new architecture which reduces the required number of physical transmon qubits by a factor of approximately k, the number of resonant modes per cavity. We also develop a Compact variant saving an additional 2x. This is significant because we can obtain a code distance $\sqrt{2k}$ times greater or use hardware with only $\frac{1}{2k}$ the required physical transmons for a given algorithm. In the near-to-intermediate term, when qubits are a highly constrained resource, this will accelerate a path towards fault-tolerant computation. In fact, the smallest instance of Compact requires only 11 transmons and 9 cavities for k logical qubits.

We evaluate variants of our architecture by comparing against the surface code on a larger 2D device. Specifically, we determine the error correction threshold rates via simulation for each and find they are all close to the baseline threshold. This shows the additional error sources do not significantly impact the performance. We explore the sensitivity of the threshold to many different sources of error, some of which are unique to the memory used in this architecture. We end by evaluating magic-state distillation protocols which have a large impact on overall algorithm performance and find a 1.22x speedup normalized by the number of transmon qubits.

In summary, we make the following contributions:

- We introduce a 2.5D architecture where qubit-local memory is used for random access to error-corrected, logical qubits stored across different memories. This allows a simple virtual and physical address scheme that exemplifies exposing native data locality to the application level. Error correction is performed continuously by loading each from memory.

- We give two efficient adaptations of the surface code in this architecture, Natural and Compact. Unlike a naive embedding, both support fast transversal CNOTs in addition to lattice surgery operations with improved connectivity between logical qubits.
- We develop an error correction implementation optimized for Compact and designed to maximise parallelism and minimize the spread of errors.
- Via simulation, we determine the surface code adapted to our 2.5D architecture is still an effective error correction code while greatly reducing hardware requirements.

3.2 Background

In this section we briefly introduce the basics of quantum computation. We review current superconducting qubit architectures and memory technology our proposed design takes advantage of. We then discuss the noise present in these physical systems. Next, we introduce the basics of quantum error correction and give a detailed introduction to the surface code and lattice surgery. We conclude with a review of the basic procedure for decoding physical errors.

3.2.1 Basics of Quantum Computing

The fundamental unit of quantum computing is the qubit. Like the classical bit, it can exist in the $|0\rangle$ or $|1\rangle$ state, but it may also exist in a coherent superposition of the two states and n qubits may exist in a superposition of all 2^n bit strings. For example, a single qubit state is $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where $|\alpha|^2 + |\beta|^2 = 1$ and $\alpha, \beta \in \mathbb{C}$. To manipulate these bits we apply quantum operations, often called gates. Single qubit gates like X (bit flip), Z (phase flip), H (Hadamard basis change), and T ($\frac{\pi}{4}$ phase) and two-qubit gates like CNOT (reversible XOR with output $b' = a \oplus b$) are unitary and reversible (invertible). We may measure a qubit to obtain either a 0 or a 1 outcome with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively. Multi-qubit operations like CNOT can create entanglement between qubits. Using only CNOT and single qubit gates, universal computation is possible, meaning any reversible multi-qubit operation is possible. The three-qubit Toffoli (reversible AND gate with output $c' = (a \wedge b) \oplus c$), a common primitive in error-corrected algorithms, can be implemented by performing a few CNOT, H, and T gates. See Nielsen and Chuang [2011] for a more comprehensive background.

3.2.2 Superconducting Qubit Architectures

In contrast to other leading qubit technologies such as trapped ion devices with one or more fully-connected qubit chains, superconducting qubits are typically connected in nearest-neighbor topologies, often a 2D mesh on a regular square grid as shown in Figure 3.2. For near-term computation, this limitation makes engineering these devices easier but results in high communication costs, increasing the chance of errors on NISQ devices and communication congestion for error corrected operations. This is a leading technology in industry, used by Rigetti, IBM, and Google.

3.2.3 Qubit Memory Technology

Recently studies, including Naik et al. [2017], Hann et al. [2019], have demonstrated random access memory for quantum information. Qubit states can be stored in the resonant modes of physical superconducting cavities attached to a transmon qubit as depicted in Figure 3.3. In these devices, transmon-transmon interactions are essentially the same as other superconducting transmon technology and transmon-cavity interactions are expected to perform similarly. Currently demonstrated error rates are promising, and there is nothing fundamen-

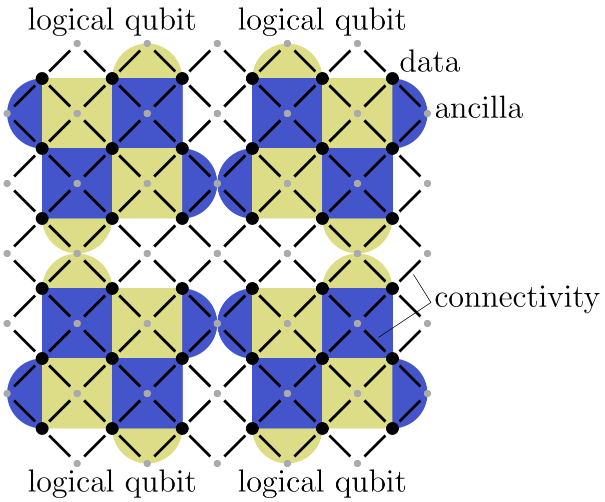


Figure 3.2: A typical 2D superconducting qubit architecture. The dots are transmon qubits where black are used as data and gray are used as ancilla for error correction. The lines indicate physical connections between qubits that allow operations between them. Four logical qubits, each consisting of 9 error-prone data qubits, are shown here in the rotated surface code with distance 3. Z parity checks are shaded yellow (light) and X parity checks are shaded blue (dark) where checks on only 2 data are drawn as half circles.

tal preventing this technology from becoming competitive with other transmon devices. We expect operation error rates to improve, cavity sizes and coherence times to increase and in general expect performance to improve as it has with other quantum technologies.

Local memory is not free. Stored qubits cannot be operated on directly. Instead, operations on this information are mediated through the transmon. Furthermore, to operate on qubits stored in memory, we first load the qubit from memory. Then we perform the desired operation on the transmons, and store the qubit back in its original location. A two-qubit operation such as a CNOT can also be performed directly between the transmon and a qubit in its connected cavity by manipulating higher states of the transmon. We use this transmon-mode CNOT later.

In this architecture, qubits stored in the same cavity cannot be operated on in parallel.

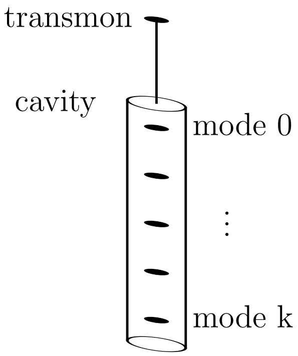


Figure 3.3: A close-up representation of the qubit memory technology we use. On top is a superconducting transmon qubit physically connected to a resonant superconducting cavity. This cavity has many resonant modes each used to store a qubit. These qubits can be loaded and stored (with random access) via the transmon.

For example, consider two qubits stored in different modes of the same cavity (two virtual addresses corresponding to the same physical address). If we want to perform an H gate on each of them in parallel, this would not be possible. Instead, we serialize these operations. There are two primary benefits of this technology. First, we are able to quickly perform two-qubit interactions between any pair of qubits stored in the same cavity because we have star-graph connectivity between the transmon and its cavity modes. Second, qubits stored in the cavity are expected to have longer coherence times by about one order of magnitude i.e. there will be 10x fewer idle errors when qubits are stored in the cavity.

3.2.4 Quantum Errors

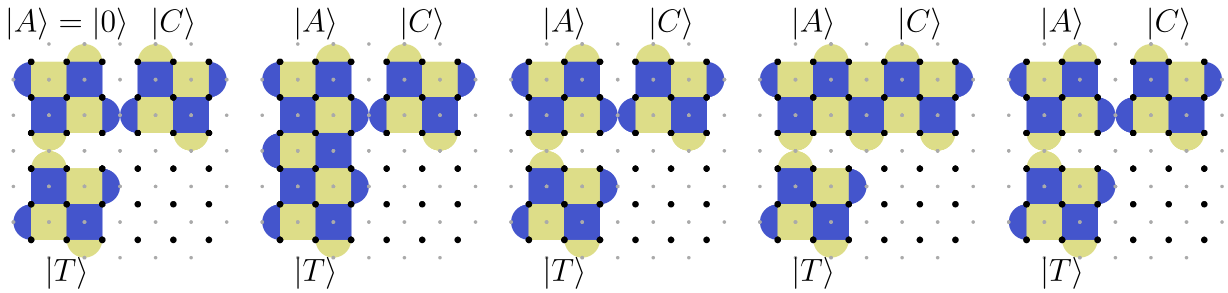
Quantum systems are inherently noisy, subject to a variety of coherent and non-coherent error. For example, when attempting to apply some gate U to a qubit we may actually apply some other gate U' which is close to the desired operation but may include an additional undesired operation. Fortunately, this type of coherent error is fairly easy to model. Since every single-qubit unitary can be expressed as a linear combination of the Pauli matrices[1]

I, X, Y, Z we can express this coherent error as a combination of bit flip (X) and phase flip (Z) errors where I is no error and Y is simultaneous bit and phase errors ($Y = iXZ$). For a quantum error correcting code this will play a part in digitizing errors, meaning we will be able to simply detect and correct X and Z errors.

Errors such as decoherence errors can be attributed to interaction with the environment. These errors are inevitable because manipulating qubits requires they not be perfectly isolated. When modeling and simulating this type of error we require the use of full density matrix simulation. In this paper, we opt not to model coherence errors in this way because simulation of this class of errors is hard (density matrices have size exponential in the number of qubits), we instead also model storage errors as Pauli errors. This is a common simplification and a conservative overestimate for the error causing our error threshold estimation to be slightly more conservative. For example, when decoherence resets a qubit to $|0\rangle$, this causes an error to a qubit in the $|1\rangle$ state but not to a qubit already in the $|0\rangle$ state whereas a Pauli X error causes a bit flip which is an error on either state.

The above errors apply to all superconducting systems and we often assume consistent error rates across the device. We treat all two-qubit interactions equally so gates like a CNOT incur some fixed error cost, a fixed chance of some error $U_1 \otimes U_2$ is applied to $|\psi\rangle$ where $U_1, U_2 \in \{I, X, Y, Z\}$. In traditional superconducting architectures (our baseline), we consider a few error sources — storage error, one and two-qubit gate error, and measurement error. In superconducting architectures with resonant cavities such as our design, there is more nuance. We consider cavity storage and transmon storage error rates separately since each has its own coherence time and we separate transmon-transmon two-qubit gates and transmons-cavity two-qubit gates. We detail this and our other assumptions for simulation in experimental setup.

[1] The Pauli matrices along with I form a complete basis over complex matrices so any single-qubit unitary $U = aI + bX + cY + dZ$ where $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$, $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.



(a) Create ancilla

(b) Merge A and T to measure A⊕T (X-basis)

(c) Split

(d) Merge A and C to measure A⊕C (Z-basis)

(e) Split then measure X-basis of A

Figure 3.4: The lattice surgery operations to perform a logical CNOT on the standard surface code (and directly supported in our architecture). Given control and target qubits $|C\rangle$ and $|T\rangle$, a CNOT is performed by enabling and disabling the parity checks as shown across 6 timesteps ((e) is two steps). We show this complex process to contrast with the fast transversal CNOT enabled by our architecture (described later in Section 3.3.2).

3.2.5 Surface Codes, Error Decoding, and Lattice Surgery

The surface code by Fowler et al. [2012a] is one of the most promising quantum error correction protocols because it requires only nearest neighbor connectivity between physical qubits. The surface code is implemented on a two-dimensional array of physical qubits. These qubits are either data, where the state of the logical qubit is stored, or ancilla used for syndrome extraction (parity checks). These ancilla qubits are measured to stabilize the entangled state of the data. These ancilla fall into two categories, measure-Z and measure-X for Z syndromes and X syndromes designed to detect bit and phase errors respectively. Data qubits not on the boundary are adjacent to two measure-Z and two measure-X qubits.

In Figure 3.2 we show four logical qubits with code distance 3 mapped to a 2D lattice of superconducting qubits. Dark physical qubits are used as data and light qubits are used as measure qubits. In this paper, we opt to explicitly indicate qubits in order to make clear how logical qubits, formed of many square and half-circle plaquettes, are mapped directly to hardware. In our diagrams however, we use customary notation by shading X-plaquettes blue (dark) and Z-plaquettes yellow (light). Half-plaquettes contain only 2 data qubits and are shown as half circles.

Each X (Z) plaquette corresponds to a single measure-X (Z) qubit and the four data which it interacts with. The corners of each plaquette are the data qubits. For the baseline, we use standard Z and X syndrome extraction (parity measurement) circuits where the qubits of this circuit are physical qubits. The Z-syndrome measures the bit-parity of its corner qubits and the X-syndrome measures their phase-parity. By repeatedly performing syndrome extraction and detecting parity changes we are able to locate errors. This repeated syndrome extraction collapses any error to a correctable Pauli error and forces the data to remain in what is called the code, or quiescent, state. Once the qubits are in this state, subsequent syndrome extraction should result in the same outcomes. If errors occur, we detect them as changes in measurement outcomes.

Errors are decoded by running a classical algorithm on the measured syndromes as specified by Fowler et al. [2012b]. In the surface code, when an error occurs on a data qubit, for example a single X bit-flip error, we see this as a change in the measurement outcome of *both* of the Z-syndrome ancilla adjacent to it. If an error occurs on every data qubit in a chain of neighbors, only the two syndromes at the ends will detect a change. The standard way of performing error decoding is to collect all of these changed syndromes into a complete graph with edge weights given by the log-probability of that chain of errors occurring. We perform a maximum likelihood perfect matching of this graph to find the most probable set of error locations which we correct or track in the classical control. If errors are sufficiently low these error chains will be well isolated and this decoding algorithm will be able to determine the correct set of corrections to be made. If errors are less sparse, this matching algorithm may misidentify which error chains have actually occurred and this can result in a logical error, that is a *logical* bit flip or phase flip is applied. These logical errors cannot be detected because they result from misidentifying the physical errors.

There are two primary ways to manipulate the logical qubits of the surface code to perform desired logical operations — braiding and lattice surgery. In this paper we will primarily consider lattice surgery which has been shown to have some advantages over braiding like using fewer physical qubits. For a more thorough introduction to lattice surgery we refer the reader to Horsman et al. [2012], Litinski [2019b], Lao et al. [2018]. In our proposed scheme, all primitive lattice surgery operations can be used such as split and merge which together perform a logical CNOT as shown in Figure 3.4. For universal quantum computation in surface codes we allow for the creation and use of magic states such as $|T\rangle$ or $|CCZ\rangle$. These states are necessary because the T and CCZ operations cannot be done transversely (using physical gates on the data in parallel to reliably perform the logical gate) in this type of code. However, high fidelity versions of these states can be generated via distillation as in Bravyi and Haah [2012], Litinski [2019b] where many error-prone copies of the state are combined to generate the state with low error probability. Our scheme permits the use of these methods in the same way as other surface code schemes and also allows more efficient implementations.

3.3 Virtualized Logical Qubits

In this section we describe in detail our proposed architecture, an embedding of the surface code which virtualizes logical qubits, saving over 10x in required number of transmons. This takes advantage of quantum resonant cavity memory technology described above to store *logical* qubits, in the form of surface code patches, in memory local to the computational transmons. In this section we describe how we can embed surface code tiles in two variations, Natural and Compact. We show the hardware operations needed to perform efficient syndrome extraction for both in our new fault-tolerant architecture. We then describe how typical lattice surgery operations are translated into operations in this new scheme, and finally how our system supports fault-tolerant transversal interactions between logical qubits sharing the same virtual address. We verify these operations via process tomography. We briefly describe how magic state distillation, an important primitive for algorithms, is translated to our system.

3.3.1 Natural Surface Code Embedding

Our goal here is to take logical qubits stored in a plane and find an embedding of that plane in 3D where the third dimension (our transmon-local memory) is a limited size, k. The intuitive answer is to simply fold the surface k times. While this works, it does not have the benefits of a more clever embedding. We propose slicing the plane into many pieces, storing them flat in memory to enable them to stitch together on-demand. This embedding enables the fast transversal CNOT and high connectivity we will describe later.

Consider the high-level three dimensional view of the quantum memory architecture presented in Naik et al. [2017]. For every transmon in this architecture (the compute qubits in the top layer of Figure 3.1) there is a cavity attached with a fixed number of resonant modes, k. Each cavity can store k qubits, one per mode. Each transmon can load and store qubits from its attached cavity by performing a transmon mediated iSWAP. We assume all transmons can be operated on in parallel as is the case in most superconducting hardware (i.e. from IBM or Google). For example, we can load qubit q_{iz} to transmon t_i and load q_{jz} to transmon t_j in parallel, simultaneously execute single qubit operations on each qubit, then store in parallel. Any other qubits stored in cavities i or j will be unaffected by these operations. We expect this technology to allow cavity size k on the order of 10 to 100 qubits and it will likely not be practical to scale k along with the size of the 2D grid as hardware improves so we cannot implement a true 3D code such as Bombín [2015]. For our analysis, we conservatively assume $k = 10$ and view this as a 2.5D architecture where we expect the width and height of the grid to scale while the depth, k, remains small.

We demonstrate how our system is sensitive to the length of these cavities in Section 3.6 where the amount of time between error correction cycles is directly a function of this cavity size k. As the size of the cavity becomes very large, the physical qubits stored are expected to be subject to more and more decoherence errors which will reduce our ability to properly decode the errors.

Consider the rotated surface code of Figure 3.2 and the high level view of this architecture in Figure 3.1. We imagine mapping each of the physical qubits of this logical qubit $q_{L,1}$ to the same mode z of each cavity in this memory architecture. Another logical qubit $q_{L,2}$ can be mapped to mode $z_2 \neq z$ of the same set of cavities. We view this as stacking the surface code patches, the logical qubits, together under the same set of transmon qubits. The transmons themselves are only used for logical operations and error correction cycles performed on the patches.

For logical qubits with code distance d we define patches on the architecture, contiguous grids of size $d \times d$ data qubits and $d \times d$ ancilla qubits. Logical qubits are mapped to multiples of d coordinates on the grid and a specific mode, z, for storage. For example, logical qubit q_L is mapped to a pair (P_{xy}, z) where P_{xy} refers to the square patch of data transmons $q_{d \cdot x, d \cdot y}$ to $q_{d \cdot x+d-1, d \cdot y+d-1}$ and z indicates which cavity mode it is stored in. A *virtual memory address* of a logical qubit refers to exactly the pair (transmon patch, index). We sometimes refer to all pairs with the same transmon patch collectively as a stack where *transmon patch* is the physical memory address where a patch is loaded.

In this memory architecture, recall we are unable to operate on qubits stored in the same cavity in parallel, however we *are* permitted to operated on qubits stored in different cavities in parallel. This implies for two logical qubits $q_{L,1}$ and $q_{L,2}$ stored in the same stack we are only able to perform syndrome extraction on at most one of these qubits at a time. In order to detect measurement errors, we typically require d rounds of syndrome extraction before we perform our decoding algorithm and correct errors. If all indices are occupied by logical

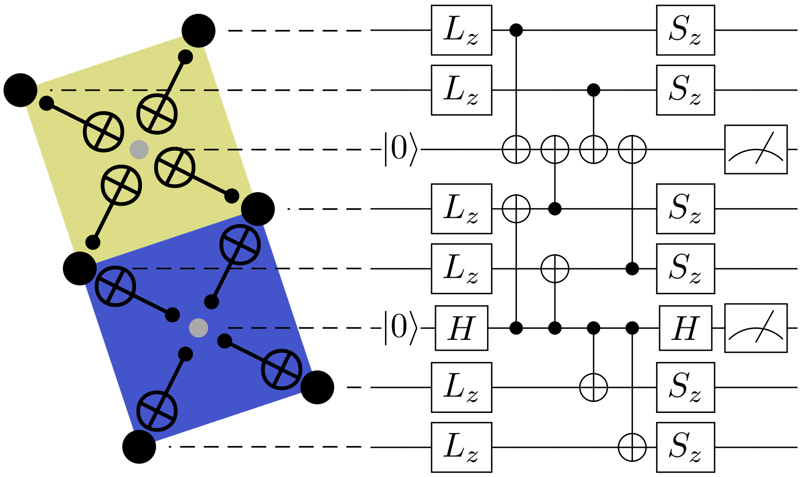


Figure 3.5: Circuit showing how to execute our Natural embedding on hardware. Left: The layout of six data (black) and two ancilla (gray) in hardware. CNOT operations between qubits are drawn between. Right: A circuit diagram of the operations applied over time where each horizontal line corresponds to a qubit and each box or symbol is an operation. The steps are L_z: load from memory mode z, $|0\rangle$: reset ancilla, CNOTs: compute the Z or X parity, Meter: measure the result, S_z: store back to memory.

qubits and we want to perform d rounds of correction to each one we have two primary strategies. We can load a logical qubit (meaning load all data in parallel to each transmon), perform all d rounds of extraction, then store the qubit.

Alternatively, we can Interleave the extraction cycles by loading the logical qubit in index 0, performing one syndrome extraction step, then storing. We execute this same procedure for every logical qubit in the stack and repeat d times. We expect this latter procedure to be less efficient, subjecting the data qubits to d load and store errors per d cycles as opposed to performing exactly one set of loads and stores when collecting all d measurements at once. We study the effect of this choice of syndrome extraction on the error threshold in Section 3.5. We detail these extraction protocols for each syndrome in Figure 3.5. Here we use L_z (S_z) to indicate loading (storing) the data from (to) index z of the attached cavity.

Intuitively, this scheme is stacking many different logical tiles together in a single location.

This includes mapping measure-Z/X ancilla to cavity modes. However, this is unnecessary, because measure ancilla do not actually store any data and are reset before every extraction step. Therefore, we can reduce the number of cavities required for this system by simply omitting any cavity where ancilla are stored. Instead, every patch in the same stack shares the same ancilla, the transmons at the top layer with no attached cavity.

In our system, up to k logical qubits share the same set of transmons, more efficiently storing these qubits than on a single large surface. In order to interact logical qubits in different stacks we load them in parallel to the transmons then interact them via lattice surgery operations like the CNOT shown in Figure 3.4. In these cases, all of the other stacks' transmons between the interacting logical qubits act as a single (possibly large) logical ancilla. In typical planar architectures, we are unable to execute transversal two-qubit operations due to limited connectivity. We can perform physical operations between qubits in the same cavity, mediated by the transmon. Therefore, in our system, we *are* able to perform transversal two-qubit interactions if the logical qubits are co-located in the same stack. We describe this next.

3.3.2 Transversal CNOT

A major advantage of this 2.5D architecture, enabled by our embedding of patches across memories, is the ability to do two-qubit operations transversely using the third dimension. The logical operation is performed directly by doing the same physical gate to every data qubit and correcting any resulting errors. On typical 2D architecture error correcting codes like the surface code, the only transversal operations are single-qubit Clifford operations like X or Z. Two-qubits operations are not possible because the corresponding data qubits of two logical patches cannot all be made adjacent. However, with memory, it is possible to load one patch into the transmons and apply two-qubit gates mediated by each transmon onto the data qubits for a second qubit stored in one mode of the cavities. This works in both

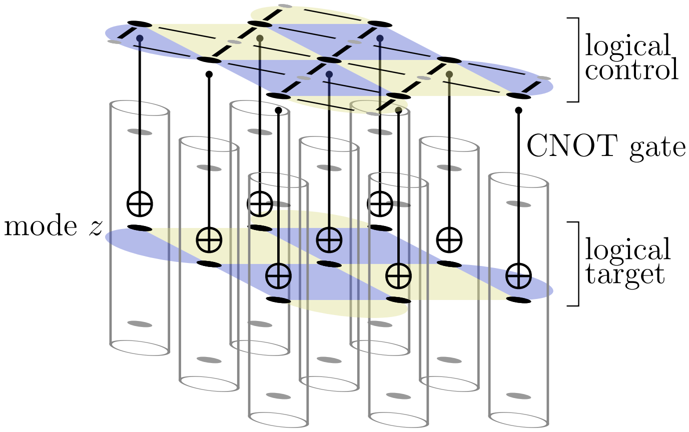


Figure 3.6: The transversal CNOT enabled by our 2.5D architecture. The data qubits for the control logical qubit are loaded into the transmons. Transmon-mediated CNOTs to mode z for every data qubit perform the logical operation. This takes one timestep to perform, 6x better than a lattice surgery CNOT.

Natural and Compact (described later).

Figure 3.6 demonstrates this for the transversal CNOT gate which we verified via process tomography (Neeley [2010], Nielsen and Chuang [2011]) to apply the expected CNOT unitary in simulation. This can be performed in a single round of d error correction cycles while the lattice surgery CNOT shown in Figures 3.4 (and later 3.9) takes 6 rounds. This can translate to major savings in runtime for algorithms.

The transversal CNOT is not limited to logical qubits currently stored in the same 2D address. With an extra step it is possible to transversely interact any two logical qubits. To do this one of the qubits must be *moved* to the same 2D address as the other using the move operation described in Litinski [2019b]. The move operation involves growing the patch toward the move target in one step by adding new plaquettes along the entire path and performing d cycles, one timestep, of error correction. Once grown, the patch can be shrunk from the other end back to its original size. The data qubits freed during the shrink are measured and used to determine any fixup operation. Once the two qubits are in the same 2D address, the transversal CNOT can be applied. It can then be moved back, left where it is, or moved somewhere else as determined during compilation. This process takes 2 timesteps or 3 if including the second move.

3.3.3 Compact Surface Code Embedding

In the previous scheme, half of the transmons did not have attached cavities (or they did not make use them). An ancilla and data qubit could share a transmon because the data are stored in the cavity the majority of the time and the ancilla are reset every cycle. This leads to a more efficient, Compact embedding which halves the required number of transmons. We will see that this comes at the cost of additional loads and stores from memory due to contention during error correction, effectively trading some error and time for significant space savings.

In the above memory architecture, because we do not store any logical qubits in the transmon layer, these qubits can act as the measurement ancilla, rather than have separate transmons only there to act as the syndrome measurement ancilla. With this observation, we can pack the data qubits of the surface code patch of Figure 3.7a more efficiently with *every* transmon having a cavity attached. Each plaquette of the rotated surface code has a single ancilla at its center, interacting with each data qubit. For Z plaquette (yellow or light) in this mapping scheme we colocate the upper-right data and the ancilla; the upper-right data is located in the cavity attached to the transmon corresponding to the ancilla. Similarly, for each X plaquette (blue or dark) we colocate the lower-left data and the ancilla; the lower-left data is located in the cavity attached to the transmon corresponding to the ancilla.

This mapping results in plaquettes which resemble triangles rather than squares, where the center of the hypotenuse of each triangle corresponds to both the ancilla qubit and the data qubit, stored “beneath” in its cavity. Every data qubit is still mapped to the *same* index. Notice in this scheme every data (sans the boundary) is still adjacent to two

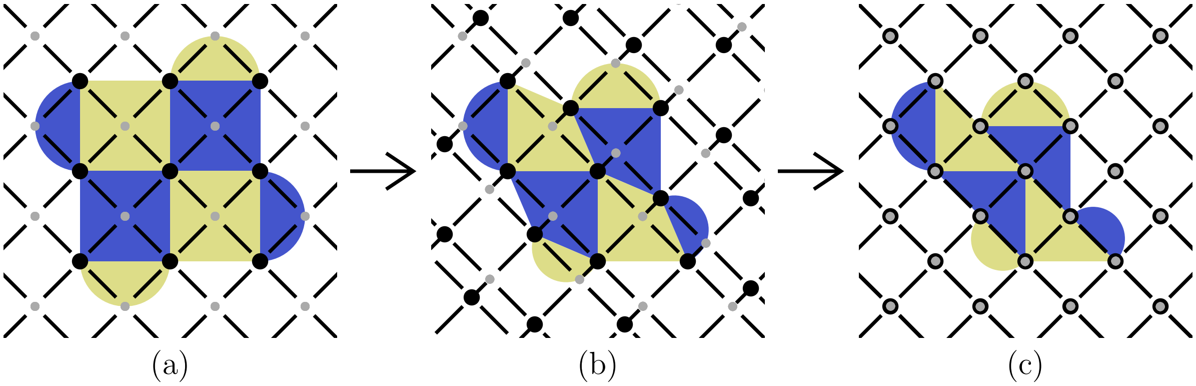


Figure 3.7: Transformation from Natural to Compact. (a) Natural embedding: Only data have attached cavities (not shown). (b) The transformation: Z ancilla (over yellow/light areas) merge with the upper-right data transmon and X ancilla (over blue/dark areas) merge with the lower-left data transmon. The opposite parings are key to keeping 4-way grid connectivity. (c) Compact embedding: All ancilla transmons without attached cavities have been removed. All remaining transmons have cavities and are used as both data and ancilla.

measure-Z and two measure-X ancilla where adjacent means either in the cavity of the ancilla or in a cavity adjacent to the ancilla. We illustrate this transformation from our undistorted Natural surface code patch to Compact in Figure 3.7 and a diagram of this architecture with a cavity for every transmon in Figure 3.8. If a different ancilla location were chosen, for example all sharing with the upper-right data, some of the syndrome extraction gates in the resulting arrangement would require six-way connectivity, two diagonal to the grid, which would be much more difficult to engineer with low noise. This scheme where X and Z ancilla share with data in opposite directions is the best scheme we found to satisfy the hardware connectivity.

In Natural, we assign square patches to predetermined square patches on the hardware. In Compact, we assign square patches to predetermined rhombus or diamond patches on the hardware. Previously, operations on the virtualized patches closely resembled the original operations because the shape was unchanged, except with the addition of loads and stores to

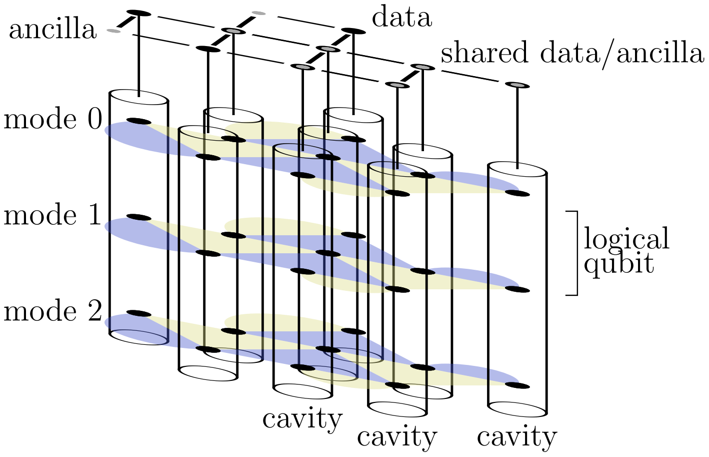
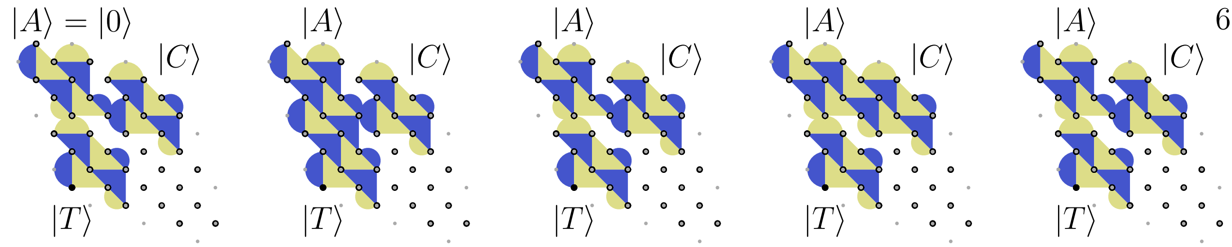


Figure 3.8: A 3D view of our Compact embedding. Shown at the top is the 2D grid of transmon qubits. Attached below every transmon is a resonant cavity. Compact surface code patches are shown stored, one in each mode. This deformed patch can be tiled in 2D.

retrieve the logical qubit from memory. The same operations apply here. We can examine the original, unmapped surface code patch and perform the same sequence of operations modulo loads and stores, on the transformed coordinates of the mapped version.

This new mapping also requires a new syndrome extraction procedure because data cannot be loaded while a transmon is in use as an ancilla. A single round of syndrome extraction can be executed by dividing the plaquettes into four groups, with each group containing non-interfering plaquettes. Two plaquettes are non-interfering if they do not share their ancilla with any data qubits of the other plaquette. This process is detailed explicitly in Figure 3.10. It is imperative this process use both the minimum number of loads and stores and keep data qubits loaded for as short a time as possible as the error incurred during this circuit directly impacts the error threshold for the code. This has a similar cost as Natural, Interleaved where a higher numbers of load and store gates were also required.

Error correction can be performed Interleaved or All-at-once just as with Natural. This



(a) Create ancilla (b) Merge A and T to measure A⊕T (X-basis) (c) Split (d) Merge A and C to measure A⊕C (Z-basis) (e) Split then measure X-basis of A

Figure 3.9: The Compact lattice surgery operations to perform a CNOT. The logical operations performed are identical to Figure 3.4 but the corresponding physical operations are arranged as shown in Figure 3.7. This uses half as many transmons as Natural. As before, it takes 6 timesteps of d error correction cycles each.

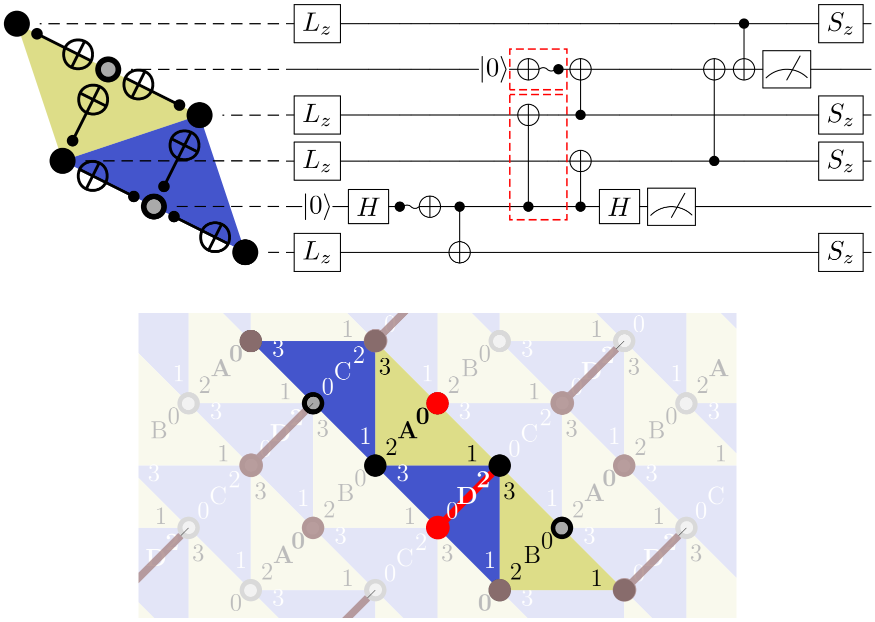


Figure 3.10: The CNOT sequence for parity checks in Compact. Top: A quantum circuit showing the hardware operations over time. Bottom: The CNOT execution order repeats A_0D_2, A_1D_3, A_2C_0, A_3C_1, B_0C_2, B_1C_3, B_2D_0, B_3D_1. The AB and CD sequences run in parallel but offset to ensure ancilla and data use do not conflict. CNOTs for A_0D_2 are marked in red where an isolated circle indicates a transmon-mediated CNOT.

should be chosen dependent on how likely storage errors and gate errors are. For example, if storage errors are expected to be significant, we may opt to use Interleaved syndrome extraction. This will cost more loads and stores so if gate errors are more significant than storage errors we may opt for All-at-once.

3.3.4 Architectural Considerations

When compiling and executing programs in our system there are several important architectural features to keep in mind. First, it is always possible to execute a transversal two-qubit interaction, rather than requiring use of split and merge. In surface code architectures, the logical qubits are not bound to a specific hardware location and are free to move around on the grid. This qubit movement is fairly cheap requiring only a single round of d error correction cycles (usually referred to as a single timestep) to move any distance. However, we require a clear area of unused patches to move through; typically, this requires about 1/3 to 1/2 of the total area to be kept as open channels to allow for distant qubit interactions. In our architecture this translates to keeping one of the resonant modes in every stack unused ($1/k$ of total qubits for cavity depth $k \approx 10$) and loading this mode along a path when a logical qubit needs to move, i.e. there is an index in the stack which has no logical qubit mapped to it. This enables our system to transport logical qubits between stacks to execute more time and space efficient transversal CNOTs. The empty mode is necessary for Compact because data is always stored back to the cavity during syndrome extraction but not required for Natural, All-at-once where the transmons themselves can act as the unused qubits to move the logical qubit through.

Unfortunately, this qubit movement is not entirely free. During the compilation process if we request many logical qubits to move in parallel this can be expensive due to serialization of intersecting move paths. Just as in current quantum systems without error correction where it is imperative to map and schedule multi-qubit interactions in a way which minimizes total execution time, it is also important in our system that logical qubits which interact heavily be located close by for similar reasons. The mapping problem on the system presented here is interesting because there is now a tradeoff between locality and serialization between operations with qubits sharing the same 2D address.

Second, we stress even though the logical qubits are stored in memory, they are still subject to errors and it is critical that every logical qubit be error corrected regularly. In the case of Interleaved syndrome extraction, every logical qubit of a stack will be roughly guaranteed to get a round of correction every k time steps, where k is the cavity depth. This rate is during steady state, when qubits are idle. When logical operations are being executed, this rate may be reduced slightly. When compiling and executing on this system, we may need to delay some operations in order to ensure stored logical qubits get the required amount of error correction and are not left so long that errors accumulate and error correction becomes less likely to succeed.

Finally, many lattice surgery operations require the use of ancilla logical qubits, for example to measure specific stabilizers which are done to execute a particular set of operations in Litinski [2019b]. This restriction requires our architecture and any compiler to guarantee one free mode of every stack be allocated to temporarily obtain large logical qubits. This free mode may be shared with qubit movement or separate if many ancilla logical qubits are used.

3.4 Evaluation

In this section, we outline our error model and experimental setup used to determine error thresholds for our mapping and syndrome extraction schemes. We compare to the surface code on a typical 2D architecture. Our goal is to demonstrate the error thresholds for various error correction schemes, i.e. to determine the necessary *physical* error rate required to begin obtaining exponentially better *logical* error rate as the code distance increases.

Currently, neither transmon devices nor transmon-memory devices used for our schemes have consistently achieved physical error rates below this threshold and instead the threshold serves as a goal or checkpoint.

3.4.1 Error Model and Noise Assumptions

For our experiments we make the following further assumptions about how noise and errors behave in both a typical 2D architecture and our 2.5D cavity memory architecture since both have the same fundamental underlying transmon technology:

- The error rates in the device do not fluctuate appreciably over time.
- Transmon qubits can be actively reset and reinitialized to $|0\rangle$ efficiently and without significant error.
- All errors are independent. No leakage errors and no correlated noise.
- All classical processing of the syndromes is instantaneous and error-free.
- Every n-qubit gate with the same n is equally error-prone. For example, every one qubit operation has the exact same chance of failure regardless of which actual physical qubit it is applied to.
- All errors are Pauli, i.e. drawn from the set $\{I, X, Y, Z\}^{\otimes n}$. For example, if a one-qubit error occurs with probability p then we apply an X, Y, or Z with probability $p/3$ and I (no error) with probability $1 - p$.
- We detect and correct X and Z errors independently. A Y error is both an X and Z error.

For each of our experiments we rely on realistic device data for current superconducting devices, provided by IBM Devices. For the memory hardware, we use experimental data from Naik et al. [2017]. These parameters are listed in Table 3.1, where $T_{1,c}$ is the coherence time of the cavity, $T_{1,t}$ is the coherence time of the transmon, Δ_t is the single qubit gate time, Δ_{t-t} is the two-qubit transmon-transmon gate time, Δ_{t-m} is the two-qubit gate time of transmon-mode interactions, and $\Delta_{l/s}$ is the load and store times. In every experiment, the gate durations for one- and two-qubit interactions is fixed. In a first set of experiments, we vary all gate errors and coherence times together, all derived from a single probability of error p given as the probability of an SC-SC (Transmon-Transmon gates) two-qubit gate error. We consider T_1 times of both cavities and transmons to determine the probability of storage error given as $\lambda = 1 - \exp\{-\Delta t/T_1\}$, where Δt is the duration stored. We consider the same potential gate error rates for each of these devices since the underlying technology behaves very similarly. While typical coherence errors are not generally Pauli, we model them as Pauli errors here as a worst-case approximation since correcting Pauli errors is harder than correcting coherence errors in general.

Table 3.1: Starting point coherence times and constant gate times for the hardware models.

Hardware Parameter	Baseline Transmons	Transmons with Memory
$T_{1,t}$	100 μs	100 μs
$T_{1,c}$	—	1 ms
Δ_{t-t}	200 ns	200 ns
Δ_t	50 ns	50 ns
Δ_{t-m}	—	200 ns
$\Delta_{l/s}$	—	150 ns

3.4.2 Experimental Setup

In every experiment, we run 2,000,000 simulated trials per data point with each trial consisting of a round of error correction. We compute the logical error rate as the number of logical errors (misidentified error chains) over the total number of trials. The large number of trials is required to estimate logical error rates to 10^{-5}. To determine the error threshold values for different surface code schemes, we vary the physical error rate over several different code sizes. The goal is to find an intersection point for each of these lines which gives a physical error rate below which we expect our logical error rate to get better as the physical error rate improves. Below the threshold we also expect the logical error rate to get better exponentially in the code distance d.

We study 5 setups to determine initial error thresholds.

- The surface code on a 2D superconducting architecture as our baseline.
- Our Natural embedding with either the All-at-once or Interleaved syndrome extraction.
- Our Compact embedding with either the All-at-once or Interleaved syndrome extraction.

In our designs, the possible sources of error are more nuanced and we study the thresholds' sensitivity to variation in the parameters. In all threshold experiments, we assume cavity depth of 10 but later study sensitivity to cavity size. The simulation code used to generate our results is available on GitHub (VLQ Code).

3.5 Error Threshold Results

We detail our threshold results in Figure 3.11. We study 5 different code distances in order to obtain the physical error threshold value. The threshold value indicates at which point increasing the code distance, d, improves the logical error rate instead of hurting it. This

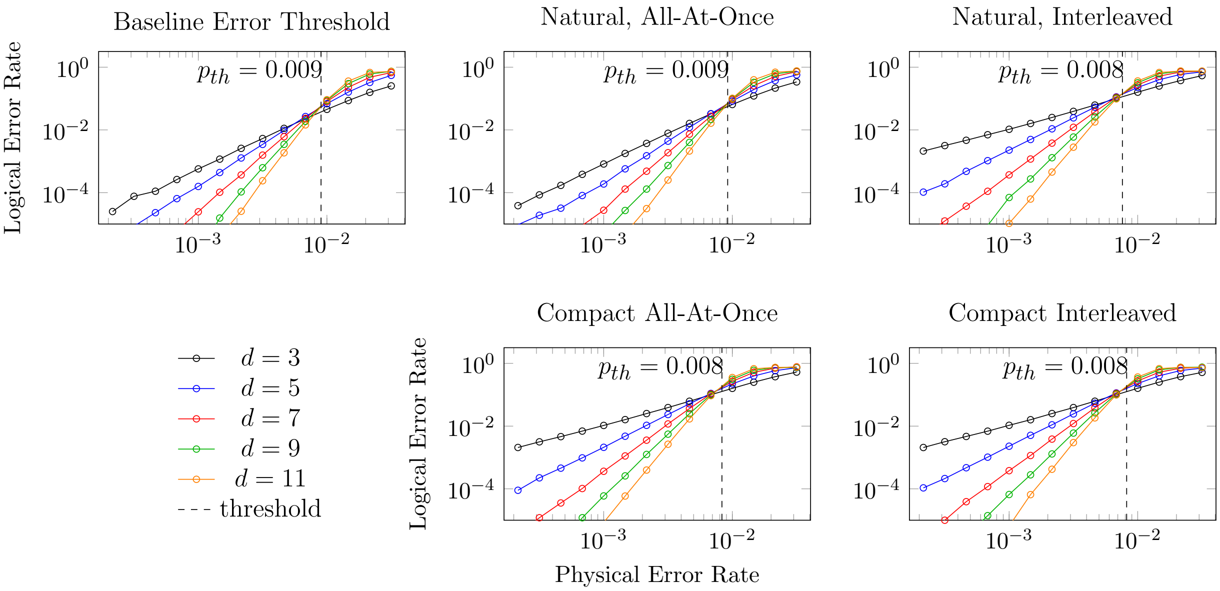


Figure 3.11: Error thresholds for the baseline 2D architecture and Natural and Compact variants of our 2.5D architecture. The thresholds are comparable to the baseline indicating the space savings obtained in our system does not substantially reduce the error thresholds. The slopes of the lines in this figure indicate, post-threshold, how much improvement in physical error rates improve logical error rate. Except for the baseline, all use a cavity size of 10.

threshold is a function of both the physical system model, the chosen syndrome extraction circuit, and the specific decoding procedure. For example, decoding procedures which do not accurately represent the probability of certain error chains occurring will do a poor job of correcting those errors. The decoding process should be directly informed by the error model. In systems with more complicated error models, the decoder should be aware of these further details to inform its decision about which types of errors occurred and the proper way to correct them. We use the usual maximum likelihood decoder because we use standard assumptions in our error model.

The major difference in each procedure is the additional error sources and different syndrome extraction procedures. For example, the baseline is not subject to any of the effects related to cavity storage or transmon-mode operations. These syndrome extraction procedures differ by the amount of storage time of data qubits in different locations (cavity vs. transmon) as well as the number of different physical gate operations applied to them. These differences however, do not cause substantial variation in the error threshold for the different setups which is extremely promising. Second, the slopes for each code distance compared across the various schemes is stable, indicating each scheme improves at a similar rate, post error threshold, and showing that the logical error rate decays exponentially with d as desired. This is significant because it means we will be able to save on total number of transmons without major shifts in the error threshold. Since transmon memory technology is expected to perform as well as other competing transmon technology, we obtain higher distance codes, and hence better logical error rate, with fewer total transmons.

3.6 Error Sensitivity Results

In this section, we study the effects of different sources of error on the thresholds obtained in Section 3.5. Specifically, we show how different system-level details affect the threshold of the code. Here we focus on Compact, Interleaved as the most efficient physical qubit mapping and subject to a wide variety of errors. In these studies, the physical error rates of all but a single error source are fixed at a typical operating point below the threshold obtained previously, 2×10^{-3} and the cavity depth is fixed at 10. Gate times are fixed while we vary the physical error rate of SC-SC gates, SC-Cavity gates, Load-Store gates or the coherence times of the cavity and the transmon. We additionally study the duration of load/store, the gates unique to memory technology. We note the effect of the SC-Cavity gate duration will be a similar, smaller effect since it occurs only once per qubit per error correction cycle. Finally, we study the effect of cavity size by varying the number of modes per cavity, causing a proportional delay between error correction cycles.

The results of these sensitivity studies are found in Figure 3.12. The logical error rate

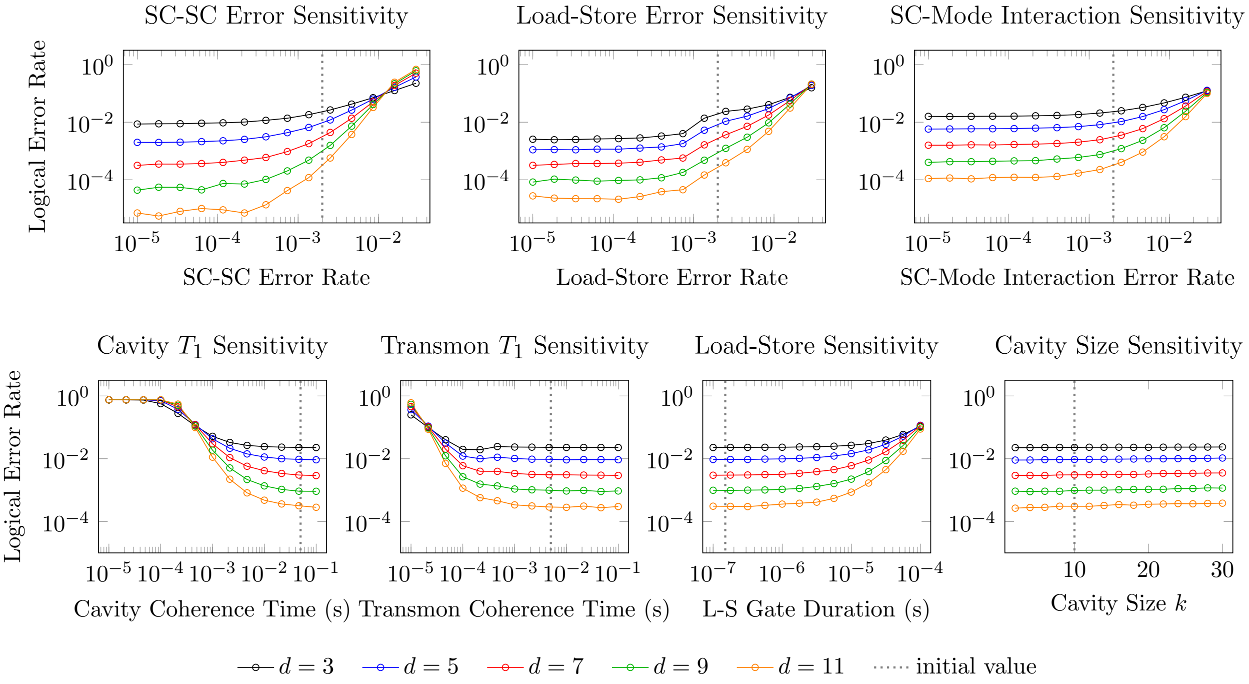


Figure 3.12: Sensitivity of logical error rate to various error sources in Compact, Interleaved. The logical error rates are most sensitive to physical error of Loads/Stores and SC-SC gates. The logical error rate is less sensitive to the coherence times and mostly insensitive to effects of load-store duration and cavity size.

is sensitive to a particular error source's probability if the slope of the line is pronounced at the marked reference value. The logical error rate for Compact, Interleaved is sensitive to all changes in system-level details to some degree. The gate error rates show the highest sensitivity, indicating improvement in these will give the greatest benefit. Coherence times are not quite as sensitive but the slightly over 10x offset between the cavity and transmon plots shows that there is no benefit in transmon T_1 being longer than 1/10 cavity T_1 when the cavity size is 10. The lines taper off, indicating other errors sources eventually dominate. Initially, we expected the cavity size to have a large impact on the logical error rate. However, when coherence times are high and gate error rates are fairly low below the threshold, the logical error rate does increase proportional to the length of the cavity but the effect is very minor. This indicates, given cavities with good coherence times, our proposed system will be able to scale smoothly into the future as cavity sizes increase.

While larger cavity sizes will make this architecture even more advantageous, there will be a point at which it has a vanishing benefit because the delay between error correction becomes too long and decoherence error dominates. For the error rates used in the evaluation, we find that cavity decoherence error starts dominating after cavity size $k \approx 150$. After this point, it would be more beneficial to improve cavity coherence time.

3.7 Magic State Distillation Resource Estimates

Now that we have shown error correction is effective in our virtualized qubit architecture, we analyze how the transversal CNOT and memory connectivity can benefit the performance of an algorithm overall. In error-corrected quantum algorithms, the dominating cost (commonly $> 90\%$) in both space and time resources is magic state distillation (Tannu et al. [2017], Ding et al. [2018], Gidney and Ekerå [2019]). For this analysis we consider how T-state distillation, a commonly used magic state, is improved. Any improvements here will translate directly to improvements in important algorithms like Shor's and Grover's.

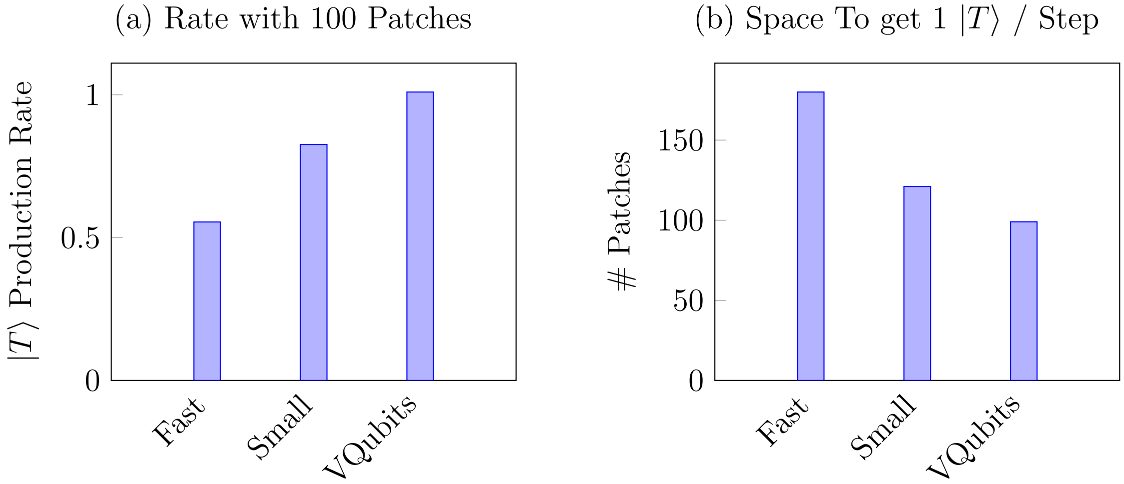


Figure 3.13: (a) The T-state generation rates of three different circuits. Higher generation rate is better. (b) The space, in terms of number of patches, required to produce a single $|T\rangle$ per time step. Lower is better. Fast Litinski [2019a] and Small Litinski [2019b] work in the surface code and do not use memory. VQubits is implemented with transversal CNOTs in our 2.5D architecture. All are based on Bravyi and Haah [2012].

We take the 15-to-1 distillation circuit of Bravyi and Haah [2012] to generate a T magic state using a single patch of transmons with 6 logical qubits stored in the attached cavities. This circuit consists of 16 qubit initializations, 15 measurements, 35 CNOT gates and a few other operations. It takes a total of 110 surface code timesteps to generate a T-state using only a single patch of transmons. If pairs of these circuits are executed in lock-step, they only take 99 timesteps.

In Figure 3.13 we compare the T-state generation rate with memory against two representative extremes designed for speed or size, Fast Lattice from Litinski [2019a] and Small Lattice from Litinski [2019b] (also based on Bravyi and Haah [2012]). Fast Lattice generates a T-state every 6 timesteps but uses 30 patches of space whereas Small Lattice, generates a T-state every 11 timesteps using only 11 patches of space. We compare these results by computing the T-state generation rate per timestep if we filled 100 patches with copies of

Table 3.2: Transmon, depth-10 cavity, and total qubit costs of each T-state generation protocol for $d = 5$.

Protocol	# transmons	# cavities	total qubits
Fast Lattice from Litinski [2019a]	1499	—	1499
Small Lattice from Litinski [2019b]	549	—	549
VQubits (Natural)	**49**	**25**	**299**
VQubits (Compact)	**29**	**25**	**279**

the circuit running in parallel. Table 3.2 show the qubit cost of each and chip area will be proportional to the number of transmons. Using our VQubits protocol generates 1.82x as many T-states as Fast Lattice and 1.22x as many as Small Lattice. This improvement allows an algorithm like Shor's to run roughly 1.22x faster or work on smaller hardware.

3.8 Summary

Realizable quantum error correction protocols are a critical step in the path towards fault-tolerant quantum computing. There has been great progress in NISQ-era devices, but it is equally critical to look towards designing architectures for QEC. In this paper, we introduce a system which virtualizes logical, error corrected qubits and is both space and time efficient without sacrificing in terms of fault tolerance.

By taking advantage of recent advances in quantum memory technology, we present a new architecture that substantially reduces hardware requirements by storing logical qubits distributed in spatially-local memory. This technology allows memory to be separated but local to computation in a quantum system. By finding a memory abstraction that keeps application data spatially local, we find application-level efficiencies in communication of logical qubit data.

We provide two direct mappings of the surface code to this new system with virtual addressing and illustrate how syndrome extraction and error correction procedures can be executed efficiently on the embedded surface code. Our embedding, combined with the random-access nature of the memory is important for several reasons. It enables fast transversal gates like the CNOT which can reduce program execution time by allowing faster operations and indirectly through improved magic-state distillation protocols. It significantly reduces the total number of transmon qubits required (10x for our analysis) which allows larger code distance patches while using 10x fewer transmon qubits and classical control wires. This allows error correction to be realized much sooner on small architectures while also enabling these devices to scale. Our results show superconducting cavity-based technology offers a promising path towards realizing spatially local memory and quickly scaling fault-tolerant quantum computation. This design can be evaluated with 10 logical qubits using as few as 11 transmons and 9 cavities which we hope motivates further experimental efforts and prompts industry to adopt and scale-up this architecture.

CHAPTER 4

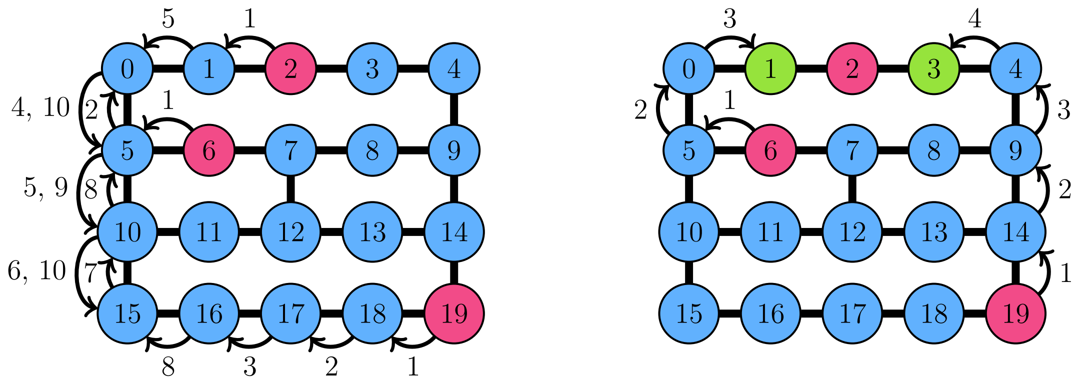
HIERARCHICAL PROGRAM STRUCTURE

4.1 Introduction

The two previous chapters cover abstractions that allow us to conceptualize storage of quantum data in the same qubit devices or nearby. Because quantum information is delicate and prone to error from interactions with the environment, it is challenging to move. This is in part why the previous abstractions have shown benefit due to allowing the data-locality of the application to influence the implementation and match the data layout of the technology.

In this chapter, we explicitly look at data movement. When we compile quantum programs with arbitrary data usage patterns, the compiler will insert explicit data movement operations. Typically, a quantum program is flattened to its basic executable gates before data movement is determined but this leads to ineffective heuristics or impractical optimization problems for the compiler. This chapter takes a small example of hierarchy, the three-qubit operation called the Toffoli gate, and shows how even using a little hierarchy, used properly, can have large gains in compiled program performance.

Quantum program compilation involves many passes of transformations and optimizations similar in many ways to classical compilers. Some optimizations occur at the abstract circuit level, independent of the underlying hardware, such as the gate cancellation from Nam et al. [2018]. One of the first steps usually taken is to convert an input program into a gate set (ISA) supported by the target hardware. For example, on IBM devices, gates are typically rewritten using only gates in the set $\{u1, u2, u3, cx\}$ (IBM Devices, single-qubit gates and the common CNOT gate described later). One critical limitation of many current available architectures is the inability to execute more complex multi-qubit operations, like the Toffoli, directly; instead, these gates must be decomposed into the supported one- and two-qubit gates. Furthermore, many current superconducting architectures only support two



(a) Expensive Qiskit routing

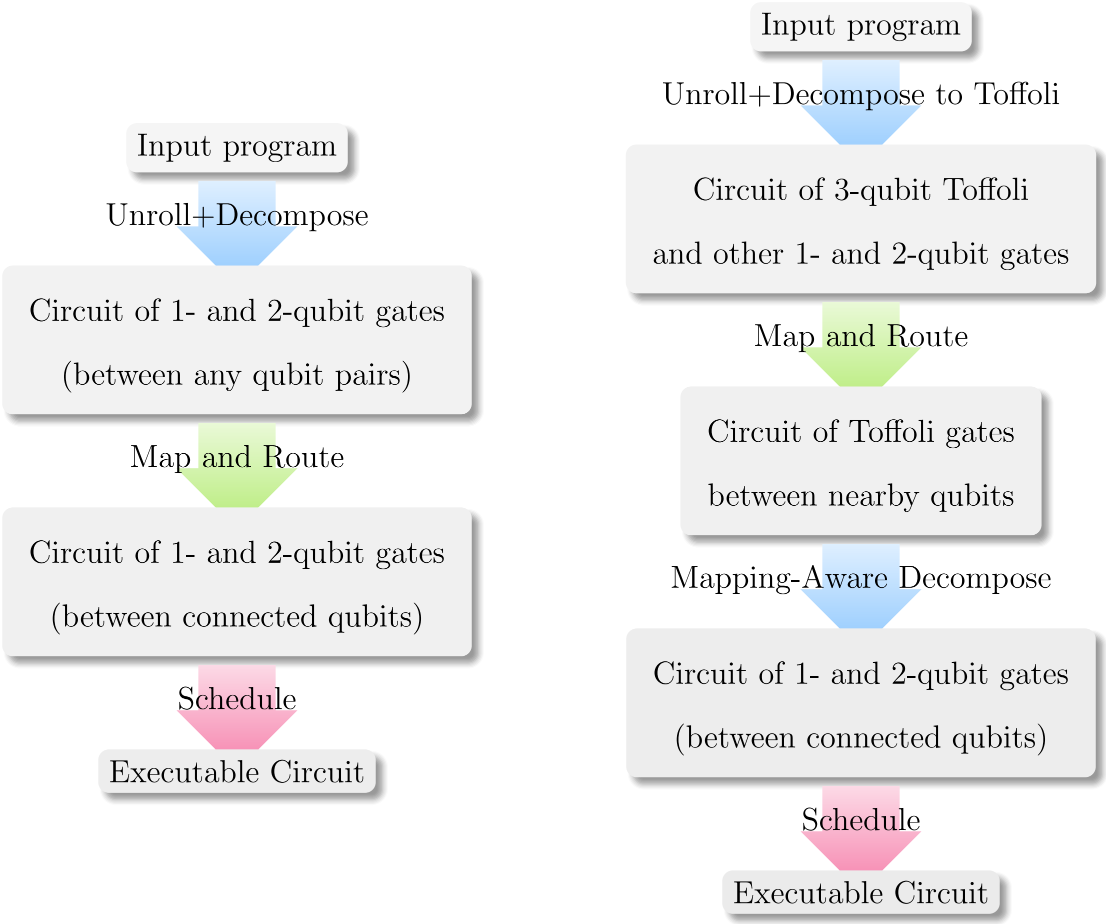
(b) Efficient Trios routing

Figure 4.1: Example routing from Qiskit (a) vs. Trios (b) for a single Toffoli operation. Circles represent qubits and lines indicate two qubits are connected. Input qubits are highlighted in red. SWAP arrows are labeled by timestep. The routed locations for Trios routing are highlighted in green while Qiskit moves them several times. Qiskit adds 16 SWAPs (=48 CNOTs), some during the Toffoli, while Trios adds only 7 SWAPs (=21 CNOTs) all before the Toffoli. Performing multiple passes of decomposition allows direct routing and enables this huge reduction in communication, increasing the probability of program success.

qubit operations on adjacent hardware qubits wired together with a coupler. This requires the insertion of additional operations called SWAPs to move the data onto adjacent qubits (which are connected with a coupler).

The process of transforming an optimized and decomposed program to the desired target is typically broken down into three distinct steps: decomposing the program into basic gates, mapping the abstract qubits of a program to specific hardware qubits and routing interacting quantum data so that they occupy adjacent qubits on hardware when they have an operation, and scheduling operations in order to minimize total program run time (depth) or to minimize errors due to crosstalk as in Murali et al. [2020b]. Each of these steps is critical to the success of the input program. A well-mapped and well-routed program will reduce the total number of communication (SWAP) operations added and subsequently reduce the compiled program's depth, both of which will increase the chance of success. Conventionally, these three steps occur sequentially. By doing so, current strategies are unable to account for any hierarchical structure in the input program, resulting in inefficient routing of qubits. An optimal compiler could find the best routing despite the lack of structure but at the cost of much slower, typically impractical, compilation. Consider the SWAP sequences inserted by IBM's Qiskit compiler for a single Toffoli compiled to IBM's Johannesburg device in Figure 4.1a. This baseline strategy adds a large number of unnecessary SWAPs as it individually routes each CNOT composing the Toffoli, dramatically reducing the probability of successful execution.

Our approach, Orchestrated Trios (or simply Trios) decomposes and routes qubits in multiple stages, as seen in Figure 4.2b. For example, first decompose an input program to one- two-, *and* three-qubit gates (e.g. do not decompose Toffoli gates) and route as before except for three-qubits, route all three to a common location with minimal SWAPs. This new program can then undergo a second round of decomposition to produce a circuit containing only hardware-permitted one- and two-qubit gates. The second round may use information


Input program
Unroll+Decompose
Circuit of 1- and 2-qubit gates
(between any qubit pairs)
Map and Route
Circuit of 1- and 2-qubit gates
(between connected qubits)
Schedule
Executable Circuit
Input program
Unroll+Decompose to Toffoli
Circuit of 3-qubit Toffoli
and other 1- and 2-qubit gates
Map and Route
Circuit of Toffoli gates
between nearby qubits
Mapping-Aware Decompose
Circuit of 1- and 2-qubit gates
(between connected qubits)
Schedule
Executable Circuit

(a) Conventional compilation

(b) Trios compilation

Figure 4.2: (a) Typical compilation passes used by Qiskit (simplified). (b) Trios compilation passes. The Unroll+Decompose pass is split into two parts: decompose into medium-size operations (Toffoli gates), later finish decomposition, but using information from the Map and Route pass.

from previous passes (i.e. locations of data qubits on the device) to generate fine-tuned decompositions for the architecture.

This layered approach has a major advantage over current routing techniques: we are better able to capture program structure by inspecting intermediate complex operations for routing. This better informs how data should be placed and moved around the device during program execution. In Figure 4.1, the Trios strategy reduces the total number of SWAPs added to 21: fewer than half compared to Qiskit. This was an extreme example we selected to present the issue, not an average case.

We specifically propose a split-pass approach to circuit decomposition. We will focus on superconducting hardware systems like IBM's cloud accessible devices, but our strategy can easily be adapted to other systems. An overview of our compilation structure is found in Figure 4.2b. This strategy has a substantial benefit on the overall success rate of programs. We demonstrate these improvements by executing Toffoli gates on a real IBM quantum computer and estimating success probability of a suite of benchmarks via simulation.

Our contributions are as follows:

- A new compiler structure, Trios, with two passes for decomposition with a modified routing pass in between which greatly improves qubit routing.

- A simple method for architecture-tuned Toffoli decompositions during the second decompose pass that allows for a new kind of location-aware optimization.

- On Toffoli-only experiments, Trios reduces the total number of gates by 35% geomean (geometric mean) resulting in 23% geomean increase in success rate when run on real IBM hardware as compared to Qiskit.

- On near-term algorithms shown in Figure 4.11 (4 to 20 qubit benchmarks), Trios reduces total gate count by 37% geomean resulting in 344% geomean increase in (or 4.44x) simulated success rate on IBM Johannesburg with noise rates of near-future hardware as compared to programs compiled without Trios. A sensitivity analysis over four architecture types shows the benefit range from 133% to 3020% increase in success rate.

4.2 Background

4.2.1 Quantum Computing Basics

The most basic object in quantum computing is the quantum bit (qubit). Unlike a classical bit which is either 0 or 1, the qubit has two basis states $|0\rangle$ and $|1\rangle$ and can exist as a linear superposition over these two states, i.e. for a quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ with $\alpha, \beta \in \mathbb{C}$ and $\|\alpha\|^2 + \|\beta\|^2 = 1$. In general, a quantum system consisting of n qubits can exist in a linear superposition of 2^n basis states in contrast to a classical system of n bits which can exist as exactly a single of these states. An important feature which gives quantum computing its power is the ability to *entangle* qubits via two qubit operations like the CNOT. This, along with quantum interference between the complex amplitudes, allows quantum programs to solve certain problems faster than classical computers. Another common two-qubit gate is the SWAP gate which switches quantum data, in-place, between two qubits.

While a qubit system can exist in these superpositions during computation, at the end of the computation, the qubits are measured producing a classical binary outcome. The probability of each outcome depends on the amplitude of each basis state (the values of $\alpha, \beta, \gamma, \dots$). Consequently, since the outcome of a quantum program is a classical bitstring and because quantum systems are inherently noisy, programs are usually run thousands of times to obtain a distribution over possible answers. A comprehensive background can be found in Nielsen and Chuang [2011].

4.2.2 Quantum Circuits

Quantum programs are typically represented as a circuit which, like a classical program, is an ordered list of instructions. Here the instructions are quantum logic gates applied to qubits. The input circuit may not be expressed in the instruction set supported by the underlying hardware or it might even be structured as hierarchical modules.

Quantum circuits have a single line for each qubit, with time flowing from left to right. Gates in a quantum circuits have the same number of inputs and outputs and gates on disjoint sets of lines can be executed in parallel. Single qubit gates are represented as boxes labeled with the indicated operations. Controlled operations, like the CNOT and Toffoli, have one or two control qubits respectively indicated by $\bullet$ and a target qubit given by $\oplus$.

Currently available superconducting quantum hardware, like that of IBM and Rigetti, only supports one-qubit gates and two-qubit gates on specific pairs. Therefore, more complex instructions must be decomposed into multiple simpler, supported operations and SWAPs must me inserted to move quantum data around the device. For example, many quantum algorithms and subroutines make use of the Toffoli gate, a three-input gate which performs the logical AND between two control bits and writes the output onto the target bit. This gate cannot typically be executed directly on available hardware and instead is decomposed into an equivalent sequence of one- and two-qubit operations. Two such decompositions are given in Figures 4.3 and 4.4. There are two key distinctions in these decompositions illustrating a more general trade off. The first, taught in Nielsen and Chuang [2011], is the ubiquitous decomposition using the minimum 6 CNOT gates, but it requires CNOTs between all three pairs of qubits. This would require either inserted SWAPs or a device connectivity containing a triangle. The second, originally discovered by Schuch [2002], uses a total of 8 CNOT gates but requires all three inputs be only linearly connected (only two of the three qubit pairs are required to be connected). While the first is apparently more efficient, this is not true if the connectivity of the underlying hardware does not directly

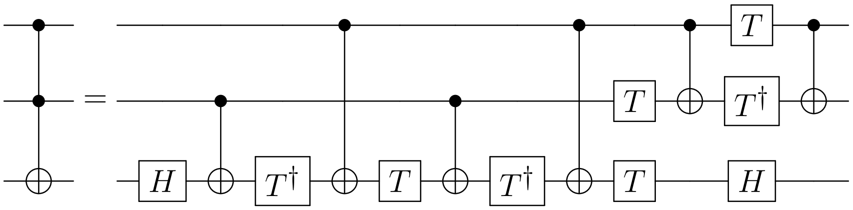


Figure 4.3: The common 6-CNOT decomposition of the Toffoli gate.

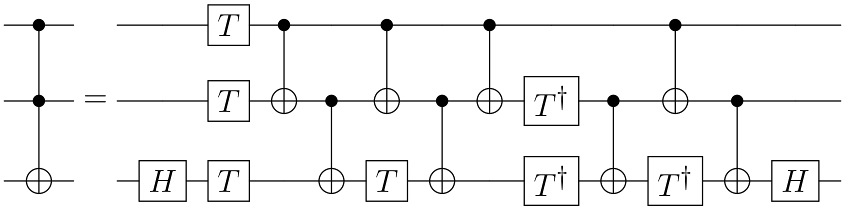


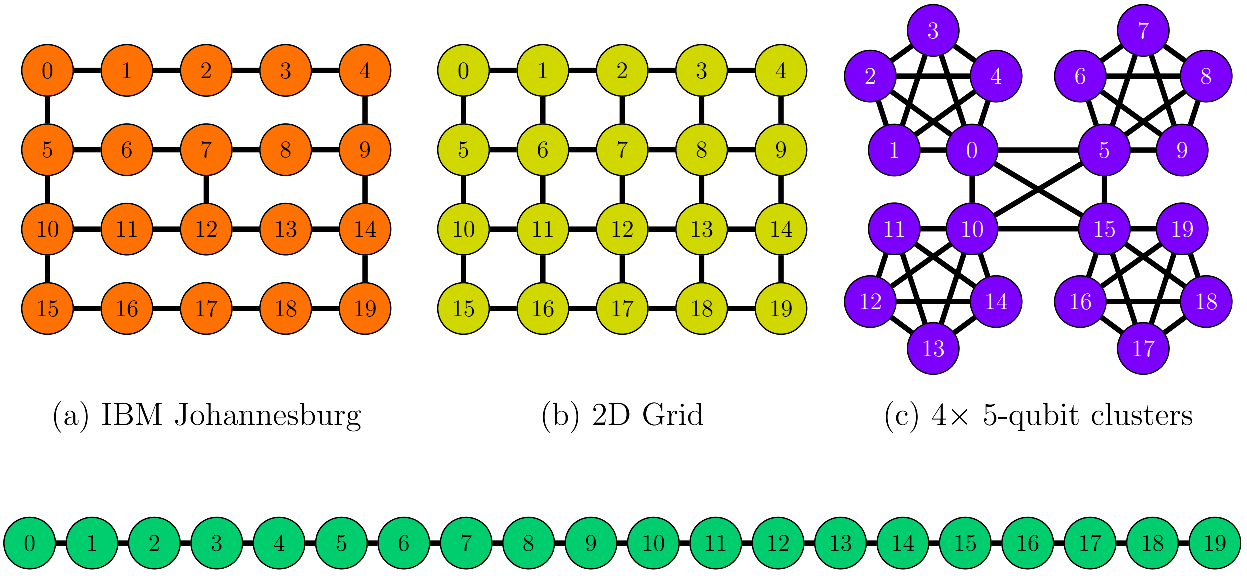
Figure 4.4: An 8-CNOT decomposition of the Toffoli gate with the same behavior.

support it. It is more efficient to use the 8-CNOT version than to use the 6-CNOT version with SWAPs added for feasibility.

For superconducting qubits, current quantum computers support gates only between adjacent hardware qubits. In order to use qubits which are currently mapped far apart on the hardware, extra SWAP operations must be inserted, each of these SWAPs is usually decomposed as a series of 3 CNOT gates (equivalent to a classical memory in-place swap using 3 alternating XORs). In the case of the 6-CNOT Toffoli decomposition above, when mapped to a device with linear or square grid connectivity, no triangles exist so extra SWAPs will need to be inserted, resulting in a greater total number of CNOTs due to the mismatch with hardware details.

4.2.3 Current Quantum Devices

In this paper we focus primarily on currently available superconducting quantum devices. This type of hardware is the primary focus of many industry players like IBM, Rigetti, and Google (Smith et al. [2016], IBM Devices, Google Bristlecone). We show some representative topologies for superconducting devices in Figure 4.5abd. For completeness, we include a clustered device shown in Figure 4.5c representative of a QCCD ion trap device such as Moses



(a) IBM Johannesburg

(b) 2D Grid

(c) 4× 5-qubit clusters

(d) Linear

Figure 4.5: Example topologies of near-term quantum devices. Orange (a): IBM Johannesburg. Yellow (b): 2D Grid. Purple (c): four groups of five fully connected clusters. Green (d) Linear. Our real experiments run on Johannesburg and our simulations explore all of these topologies. Colors correspond with the bars in Figures 4.9, 4.10, and 4.11.

et al. [2020]. These systems exhibit all of the properties previously discussed. They have a small universal supported gate set which all programs must be transformed into and only support local two-qubit operations. The connectivity of these devices is given as a *coupling graph* specifying which pairs of qubits can execute CNOTs.

Furthermore, these systems are subject to a wide variety of noise which cause programs to fail. Some noise is due to manufacturing imperfections or calibration error. Some is inherent to quantum program execution resulting from the imperfect physical isolation of the qubits from the environment required to manipulate the quantum state (Krantz et al. [2019]). In IBM machines, the experimental devices of this work, single-qubit gate errors are small, occurring on average 1 in 2000 operations. CNOT gate errors are more significant occurring on roughly 1 in 100 gates. Measurement error is also significant, with errors on the same order of magnitude as CNOT gates. Finally, qubit lifetimes (coherence times) are relatively short, allowing on the order of 50 CNOT gate durations before the qubit state is lost (but gates can often run in parallel while imposing additional crosstalk error, Gambetta and Sheldon [2019]). Therefore, quantum compilation is essential to reduce both of these sources of error: add as few extra gates as possible and minimize total execution time.

4.2.4 The Compilation Problem

In the NISQ era, quantum programs are highly optimized in order to reduce the effect on errors and maximize the probability the correct answer is observed. Similar to many classical programs, compilation uses a pass structure, where a set of transformation and optimizations are applied in a fixed order resulting in the compilation of an input quantum program to an executable for the target hardware (JavadiAbhari et al. [2015], Nam et al. [2018]). For the most part, these optimizations take place at the circuit-gate level. Some optimizations are hardware independent, for example, reducing total number of gates via commutativity-aware gate cancellation or find-and-replace with circuit identities. Other passes are focused on decomposing gates into the hardware's ISA (Sasanian and Miller [2012], Amy [2013], Maslov et al. [2008]).

One of the most important parts of this compilation process is mapping and routing the optimized program to one executable on the target hardware, typically done post-decomposition. Quantum mechanics imposes new constraints on these, different from classical compilation or logic synthesis. By the no cloning theorem, quantum states cannot be copied, only entangled, which prevents fan-out or fan-in. Instead, the data must be routed sequentially (i.e. moved with SWAP gates) to each place it is needed.

Compilation involves three main steps. First, mapping program qubits to hardware qubits in order to minimize the total distance between qubits that will need to be close by in the future (Murali et al. [2019], Tannu and Qureshi [2019], Wille et al. [2019]). Second, routing pairs of CNOT inputs to be adjacent by inserting SWAPs (Cowtan et al. [2019], Hirata et al. [2011]). Finally, scheduling operations to minimize total execution time (Guerreschi and Park [2018], Murali et al. [2020b]). In general, the compilation problem is computationally hard and, while some attempts at optimal solutions have been pursued by Tan and Cong [2020], Siraichi et al. [2018], Wille et al. [2014], the dominant approach is heuristics. In this work we focus on two pieces of this compilation problem: decomposition and routing.

IBM's Qiskit compiler, the standard for compiling programs to execute on an IBM device, has a default sequence of passes. First, all high level optimization and analysis passes are performed and all gates are unrolled and decomposed to the target gate set. Then single passes of mapping, routing, and scheduling are performed (Abraham et al. [2019]).

4.2.5 Evaluation Metrics

When evaluating compiler methods, we use a few metrics to compare our results. Our primary metric is program success rate, the fraction of circuit executions that result in the correct output. Others use fidelity, which can stand-in for success rate when evaluating sub-circuits where the output is not measured. When executing a quantum algorithm, the corresponding quantum circuit is typically executed thousands of times to gather output statistics or identify the error-free result.

Program success rate is highly dependent on the noise characteristics of the quantum computer the program runs on. The rates of these device errors can fluctuate day-to-day so we also use the simpler metric of two-qubit gate count. The number of two-qubit operations in the final compiled circuit is inversely correlated with the success rate because they are usually the largest source of noise.

4.2.6 Simulation

Simulating general quantum systems is exponentially expensive in the size of the system and therefore it is difficult to realistically model all of the errors during the execution of a quantum program. We use a simplified model for simulation to predict, specifically to obtain a close upper bound on, the success rate of a program with specified gate error rates and qubit coherence times. In our simplified model, we compute the probability of a program succeeding as the probability that no gate errors occur: $(p_{gate})^{n_{gates}}$ times the probability no coherence errors occur, $p_{coherence}$, where the latter is computed as $e^{\Delta/T_1+\Delta/T_2}$, Δ is the total program duration, and T_1 and T_2 are the relaxation and dephasing times (collectively decoherence).

Current error rates, while rapidly improving, are still insufficient to obtain high probabilities of success, making it difficult to compare our mid-size benchmarks that are large enough to need many SWAPs. For our simulations we use error rates 20x improved over current IBM Johannesburg error rates to obtain reasonable success rates and we study sensitivity to this choice later.

4.3 Motivation: Conventional Compilation

In this section we motivate the need for a split decomposition pass with routing in between. We look closely at the Qiskit compiler which does not effectively account for the structure in programs. It often produces circuits with an excessive number of swaps, suggesting room for improvement.

The default compilation framework in Qiskit, used to transform input circuits to be executed on their hardware, ensures a fully decomposed circuit before mapping, routing, and scheduling occur. As a simple example, consider three qubits placed fairly distant on IBM's Johannesburg device, but for which we need to execute a Toffoli gate on them as in Figure 4.1a. Qiskit decomposes this Toffoli as Figure 4.3 with 6 CNOTs. Each CNOT acts on distant qubits so the many SWAPs inserted for all 6 CNOTs gets expensive quickly. When routing, we first SWAP the first interacting pair together (usually by adding SWAPs from control to target or the reverse, but a meet-in-the-middle strategy is also possible) and the qubit mapping is updated. The next CNOT is also distant so we add SWAPs to move them together and there is an even chance that the SWAPs for the second CNOT separate the two qubits that were just brought together.

Ideally, we move the third qubit to the already adjacent pair, but Qiskit cannot recognize this situation and could just as well move the other way. This is clearly sub-optimal and could continue on for the other four CNOTs. Even in the case where it makes the correct decision to move the distant third qubit, there are problems. Because each pair of qubits needs to interact, we may need single additional SWAPs as the qubits compete to be neighbors. This causes the 6-CNOT Toffoli decomposition to use many more than 6 CNOTs when there is not a triangle in the qubit connectivity graph. The core idea is that the routing strategy fails to take advantage of two things. First, it has effectively forgotten the desired operation is a Toffoli (which requires all three qubits be adjacent) and second, that a more efficient Toffoli decomposition could be chosen that was more suitable for the underlying device architecture.

In the example, inefficient compilation adds a total of 16 SWAPS or 48 CNOTs in total.

Some approaches in the past have attempted to solve the first of these problems, for example Wille et al. [2016], Baker et al. [2020c] use lookahead when choosing routing strategies and while this helps to treat the symptoms of pre-decomposing all operations it does not remedy the underlying problem.

4.4 Orchestrated Trios

In this section we describe our proposed compilation structure compared to the conventional one as outlined in Figure 4.2 earlier. Specifically, we focus on improving the routing and decomposition stages of compilation. Previously, we identified a key problem in current methods: decomposing the program to one- and two-qubit gates up front hinders the ability of heuristic-based compilers to effectively minimize the communication cost, i.e. the number of SWAPs added, and eliminates the possibility of location-aware decompositions.

We propose a new pass structure. Rather than performing a single round of decomposition and routing, we propose a split approach. Any program processing prior to decomposition stays the same. The decomposition pass is then divided so the majority of decomposition occurs next but any Toffoli gates are left as-is before moving on to mapping and routing.

The mapping and routing passes come next like normal but must be modified slightly to handle three-qubit gates. The mapper can simply treat the non-decomposed Toffoli as it would the equivalent 6 CNOTs for the purposes of determining which qubits most need to be placed nearby. We then do the modified routing pass, moving *groups* of qubits together instead of only pairs where all or all-but-one of the group are moved into a single neighborhood via SWAPs. This greatly improves the effectiveness of the routing heuristics when applied to this modified routing pass. There are some subtleties when coordinating the routing of multiple qubits to the same place to ensure the paths don't overlap. For the purposes of our evaluations we do the following but many similar heuristic strategies are possible.

Taking the next operation to apply, we first find the shortest paths (using any shortest path algorithm on a graph) between all the pairs of qubits. We choose the qubit with the shortest sum of paths to the other two qubits as the destination. SWAPS following these two paths are then inserted into the circuit. The two shortest paths are checked for overlap. If the ending points overlap, the second is only routed to the penultimate hardware location along the swap path and the first becomes the middle qubit adjacent to both others. This can save one valuable SWAP but doesn't affect the correctness. Once they are adjacent, the Toffoli gate is now on adjacent qubits and routing can continue to the next operation.

Finally, the second decomposition pass is run. This is different from normal decomposition as there are only Toffoli gates to decompose and they are already mapped to neighboring qubits. We could use the default 6-CNOT decomposition and still get the above benefit of improved routing but now that we have more information, this can be exploited to further reduce SWAPs due to a mismatch between the decomposition and the hardware connectivity. If all three pairs of qubits are connected, then the 6-CNOT Toffoli of Figure 4.3 is best, otherwise use the 8-CNOT Toffoli of Figure 4.4, ensuring the middle qubit is used for the middle of the decomposition (Any of the three qubits can be the target by simply moving the two H gates to that qubit).

When routing complex operations like the Toffoli, we recognize the underlying hardware does not usually support triangles in the connectivity graph but linear connectivity is sufficient for a decent decomposition. Since we are creating operations on three qubits, the qubits must be routed into a valid linear connectivity. That is, a configuration where each qubit is connected with at least one of the other qubits.

This method can be easily extended to be noise-aware like previous work, Murali et al. [2019], Tannu and Qureshi [2019], by using a noise-aware mapper with the simple modification described earlier where the path-finding graph has weighed edges with the $-\log$ value of the CNOT success rate. The path distance represents the $-\log$ probability of success of that particular path where lower values indicate a higher success rate and the shortest path can be found just as before and the routing steps are unchanged. Any routing strategy designed for one and two-qubit gates can be modified to work for one, two, and three-qubit gates and used as the first routing step of Trios.

In programs where there are no three qubit gates as in the typical NISQ benchmark, Bernstein and Vazirani [1993], which is specified directly as CNOT gates, our strategy will have no effect. Many benchmarks, however, are written using Toffoli gates because they are the quantum analog of the AND gate, ubiquitous in arithmetics and other common subroutines.

Trios can naturally be extended to any multi-qubit operation of three or more qubits but this introduces the challenges of simultaneously routing many qubits and of designing decompositions that are efficient with whichever grouping the simultaneous router can achieve. It is not obvious how to route more than three qubits into a line or other desired shape. As many NISQ benchmarks are not typically written with more complex structures and usually phrase them in terms of one-, two-, and three-qubit gates, this extension may only be desirable for larger-scale quantum computing.

4.5 Evaluation

4.5.1 Toffoli Only Circuits

We first evaluate the effect of our new compilation strategy by studying simple circuits containing only a single Toffoli gate. In these experiments, we place the three input qubits at random locations on the target hardware to emulate the potential locations of the qubits at some intermediate point in the execution of a more complex circuit.

We study these circuits on a real IBM device, namely IBM Johannesburg, a 20-qubit device with limited connectivity, shown in Figure 4.5a. We use the default Qiskit compiler which decomposes the Toffoli gates before doing shortest path routing compared to our proposed method where we do shortest path routing first and then decompose the Toffoli. We study the use of two different Toffoli implementations: a 6 CNOT decomposition with full qubit connectivity and an 8 CNOT decomposition with linear qubit connectivity.

In all four configurations, we compare the total compiled CNOT counts which correlates with the total success probability of a program. For execution on Johannesburg, we prepare the qubits in the states $|110\rangle$, perform the compiled Toffoli, then measure the three qubits of interest and compute the success rate as the probability of obtaining the correct answer (here the $|111\rangle$ state), where each experiment is performed with 8192 trials.

4.5.2 NISQ Benchmarks and Quantum Subroutines

We also study Trios on real quantum benchmarks of moderate size using simulation only. The error rates of current devices are still too high to run benchmarks of these sizes but are expected to run on current devices as errors improve in the near future. We choose error rates 20x better than Johannesburg rates as this make the estimated success probabilities within a reasonable range and is a realistic near-term estimate. We discuss sensitivity to this choice later.

We study four implementations of the many-controlled-NOT (CnX) gate. This subroutine has many use cases from Grover's algorithm to various arithmetics. The implementations take advantage of differing numbers of ancilla and are chosen based on the number of available qubits on hardware. We study three adder implementations: Cuccaro, Takahashi, and QFT. The first two have many uses of the Toffoli gate while the latter has no such gates, for comparison. We study a small version of Grover's algorithm as well which makes use of the `cnx_logancilla` subroutine. Finally, we compile two common NISQ benchmarks: QAOA for Max-Cut and Bernstein Vazirani (BV). We expect no gain on these benchmarks since

Benchmark	Qubits	Toffolis	CNOTs*
`cnx_dirty`, Baker et al. [2019]	11	16	128
`cnx_halfborrowed`, Gidney [2015]	19	32	256
`cnx_logancilla`, Barenco et al. [1995]	19	17	136
`cnx_inplace`, Gidney [2015]	4	54	490
`cuccaro_adder`, Cuccaro et al. [2004]	20	18	190
`takahashi_adder`, Takahashi et al. [2009]	20	18	188
`incrementer_borrowedbit`, Gidney [2015]	5	50	448
`grovers`, Grover [1996]	9	84	672
`qft_adder`, Ruiz-Perez and Garcia-Escartin [2017]	16	0	92
`bv`, Bernstein and Vazirani [1993]	20	0	19
`qaoa_complete`, Farhi et al. [2014]	10	0	90

Table 4.1: Details about our benchmarks both NISQ programs and other quantum subroutines. We consider circuits with and without Toffoli gates where we expect advantage only for circuits containing Toffoli gates. For BV we assume the all 1-bit string. The different CnX (many-controlled-NOT) gates use various numbers of ancilla. *The total number of CNOT gates is after decomposition with the 8-CNOT Toffoli but does not including any SWAPs for routing.

they do not contain any Toffoli gates. A summary of our benchmarks is found in Table 4.1 using implementations found in Baker et al. [2020b].

As noted previously, the connectivity of the underlying hardware has a significant impact on the number of required SWAPs. For example, on a completely connected set of qubits, no SWAPs are ever needed. In architectures with greater connectivity, we may opt for a more efficient Toffoli decomposition using 6 CNOTs. With simulation we study the effect of connectivity on the overall expected success rates and gate counts. We study four different connectivity models, all shown in Figure 4.5, each with 20 qubits, the topology of IBM's Johannesburg device containing four connected rings, a 2D mesh, a line, and a small clustered architecture representative of a QCCD ion trap.

We use error rates reported by IBM obtained via randomized benchmarking on a daily basis; for simulations we use error numbers obtained from Johannesburg obtained on 8/19/2020 with an average T1 time of $70.87\mu s$, T2 time of $72.72\mu s$, two qubit gate time of $0.559\mu s$, a one qubit gate time of $0.07\mu s$, two qubit gate error of 0.0147, one qubit gate error of 0.0004. Source code for all experiments is available on GitHub (Trios Code). Experiments using IBM are tested with version 0.14.0 through their Python API. When compiling with Qiskit for the single Toffoli experiments, we use the default settings for the `transpile` function while specifying the Johannesburg backend. This means light optimization is performed: a stochastic routing policy is chosen, and some simple optimizations such as single qubit gate consolidation is performed. We fix the initial mapping to force routing to occur.

4.6 Results and Discussion

4.6.1 Trios Reduces Total Number of Gates

In both sets of experiments, the total number of gates required to make the input programs executable is much less than when using the default Qiskit compiler. When compiling our simple programs consisting of a single Toffoli gate with qubits mapped in random locations, we reduce the average number of gates by 35% geomean.

In Figure 4.7 we show 35 different triplets of hardware qubits for each of the four strategies. For each triplet, we note the total distance between the qubits on the hardware, given by the shortest path distance in the underlying topology. Even when the distance is relatively small, Trios outperforms Qiskit, reducing overall gate count. As the distance increases, this performance margin tends to increase. In the small distance cases, this can be attributed to Trios choosing the better Toffoli decomposition for a linearly connected topology. This is significant for two reasons. First, the fewer the gates, the less likely an error occurs due to qubit manipulation. Second, fewer gates, especially long sequential chains of SWAPs, often means lower circuit depth, meaning fewer chances for decoherence errors. Together this translates into faster and more successful programs.

This advantage extends to our NISQ benchmarks which contain various numbers of Toffoli gates. In Figure 4.10 we note substantial reductions in total gates across all benchmarks containing Toffoli gates across all underlying topologies. The only exception is the two smallest benchmarks (on 4 and 5 qubits) for the clustered topology because they could be compiled with zero SWAPs.

An extreme of the clustered topology is a single cluster with all-to-all connected qubits. On this device, Orchestrated Trios would have no benefit as operations can be performed between any pair of qubits so no SWAPs are needed and routing is trivial. However, as quantum technologies scale to more than a few qubits, fully-connected architectures hits physical limitations and must be re-engineered. As trapped ion qubit chains get longer, for example, gate operations become slower and lower fidelity. Murali et al. [2020a] showed that the optimal trap size is 15–25 ions interconnected similar to our cluster model with cluster sizes of 15–25 where Trios does benefit.

On average, for Toffoli-containing programs we reduce gate count 37%, 36%, 48%, 26% for Johannesburg, Grid, Line, and Cluster topologies respectively with the maximum gain obtained for linear devices.

4.6.2 Trios Improves Overall Success Rate

In general, we expect programs with fewer total two-qubit gates to succeed with higher probability. In devices with limited connectivity, the addition of routing operations like SWAPs, usually decomposed to 3 CNOTs, can severely reduce the chance an input program

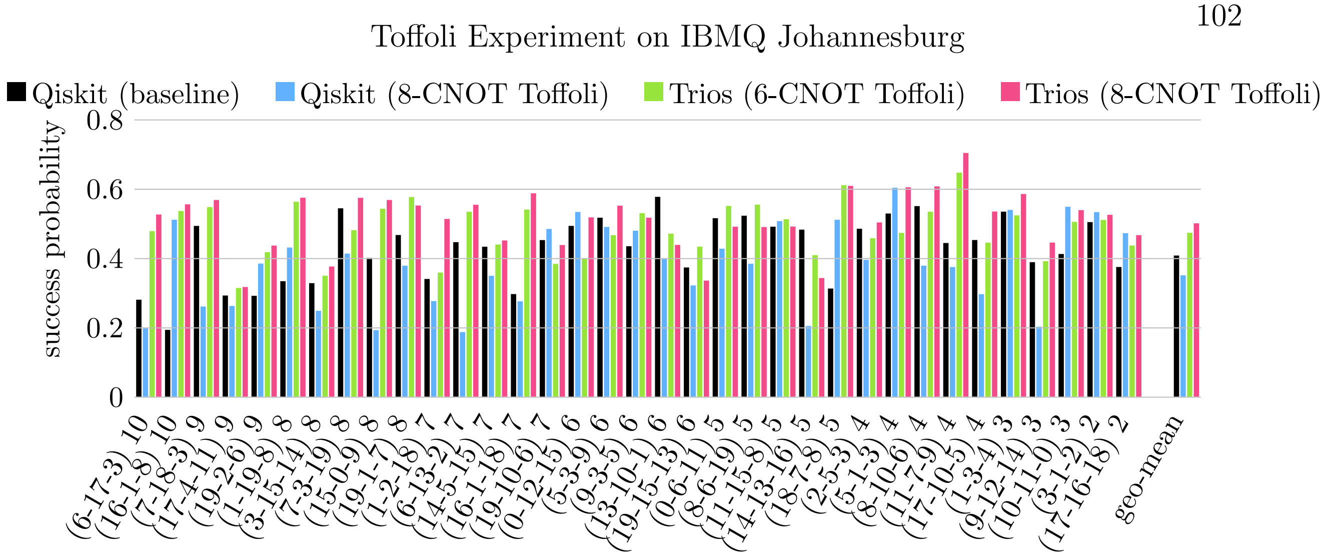


Figure 4.6: Success probabilities of Toffoli gates between random triplets of qubits. Higher is better. The x-labels specify the three qubits and total swap distance. The geometric mean success rates for each compiler are 41%, 35%, 47%, and 50% respectively. Trios (8-CNOT) improves average success rate by 23% vs. the Qiskit baseline.

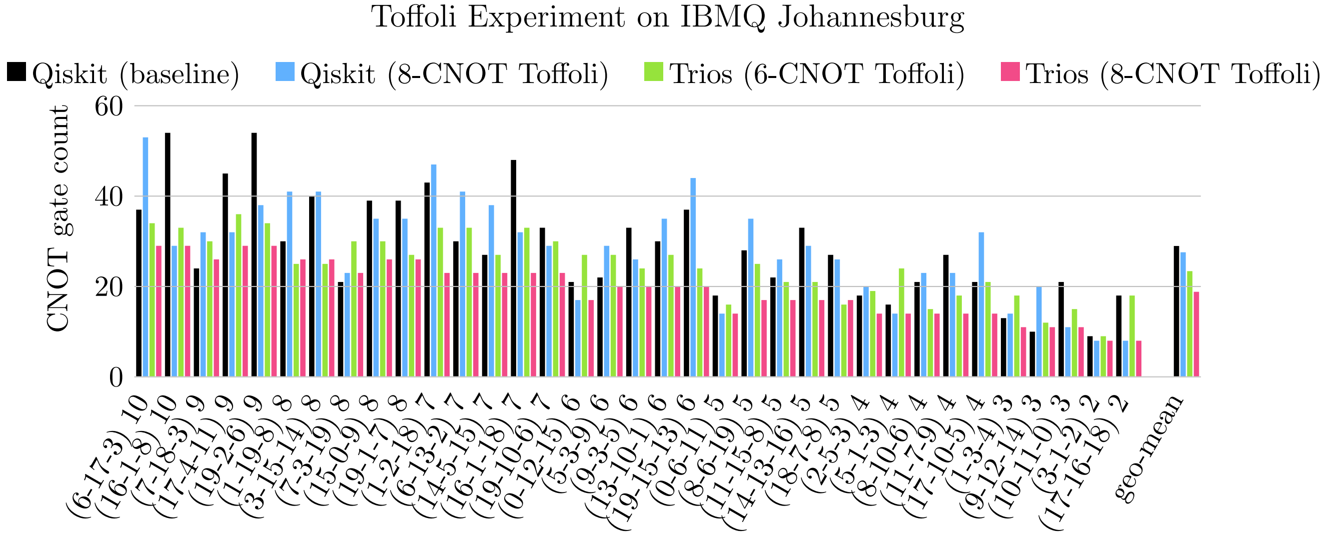


Figure 4.7: Total number of two-qubit (CNOT) gates required to execute a Toffoli gate between various distant qubits. Lower is better. The x-labels specify the three qubits and total swap distance. The geometric mean gate counts for each compiler are 29, 28, 23, and 19 respectively. Trios (8-CNOT) reduces average gate count by 35%.

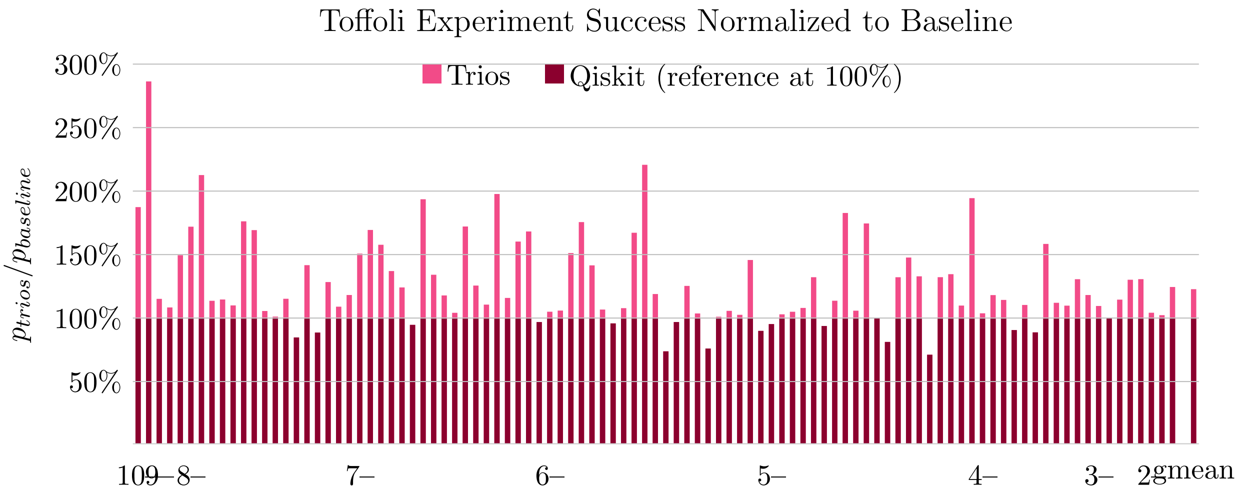


Figure 4.8: Normalized success probabilities of Toffoli gates between triplets of qubits. Higher is better. Bars below 100% indicate lower success rate for Trios. The geometric mean increase in success rate is 23%. The x-labels indicate the qubit distance for a range of bars.

can succeed. While success rate is inversely correlated with number of gates, gate error is not the only reason a program can fail and reducing gate counts does not *guarantee* improved success rates.

In Figure 4.6 we show the success rates of our Toffoli-only experiments when the two controls are initialized to $|1\rangle$ and the target is initialized to $|0\rangle$ so we measure the probability of obtaining $|111\rangle$. These results are obtained from Johannesburg on 8/19/2020. The x-axes of both Figures 4.6 and 4.7 line up to compare gate counts and resulting success rate. In general, experimentally, fewer gates results in substantial improvements to success rates. For example, a Toffoli on qubits 6, 17, and 3 compiled with Trios improves success rate from around 30% to over 50%. On average, we improve success rates by 23 % geomean with max of 286%. In Figure 4.8, we show improvements compiled with Trios normalized to baseline for 99 different triplets of varying total distance on Johannesburg.

Trios on average improves the probability of success for these circuits. However, there are a small number of cases where Trios performs worse despite having a smaller number of total gates. This can be attributed to several different factors. For example, the chosen edges for SWAP paths may be more noisy, or on pairs of edges with greater crosstalk, or the final qubits which are measured have worse readout error. Regardless, reducing the overall gate count of a program is an important contributing factor to improving expected success rate.

For our simulated NISQ benchmarks, we see even larger gains. The reduced gate counts in Figure 4.10 translate to major improvements in simulated success rate in Figure 4.9 (normalized success rates in Figure 4.11). For example, in `cnx_logancilla-19`, Trios more than doubles the expected success rates when compiled to each of the architectures. In many cases, the expected success rate of programs compiled with Qiskit is effectively zero while Trios has a realistic chance of obtaining the correct answer. As expected, on programs containing no Toffoli gates, Trios has no effect on success showing that it introduces no measurable overhead. This suggests Trios can easily be added to other quantum compilation tool flows.

4.6.3 Trios Routes Complex Interactions Better

Trios improves gate counts, and consequently improves success rates, by routing more efficiently and choosing more appropriate Toffoli decompositions based on the underlying architecture's connectivity. Current compilers, like Qiskit, perform routing on fully decomposed and unrolled programs, and while this must eventually be done, it leads to less efficient routing policies and relies on assumptions that a theoretically good decomposition (fewest CNOTs) is the best decomposition for the hardware. Trios eliminates this by choosing a context-dependent Toffoli decomposition and routing multi-qubit gates as single units.

Trios greatly improves effectiveness compared to a *heuristic-based* compiler by applying similar heuristics to the higher abstraction level Toffoli gates. An optimal routing of the decomposed circuit would be better except it cannot select the best architecture location-

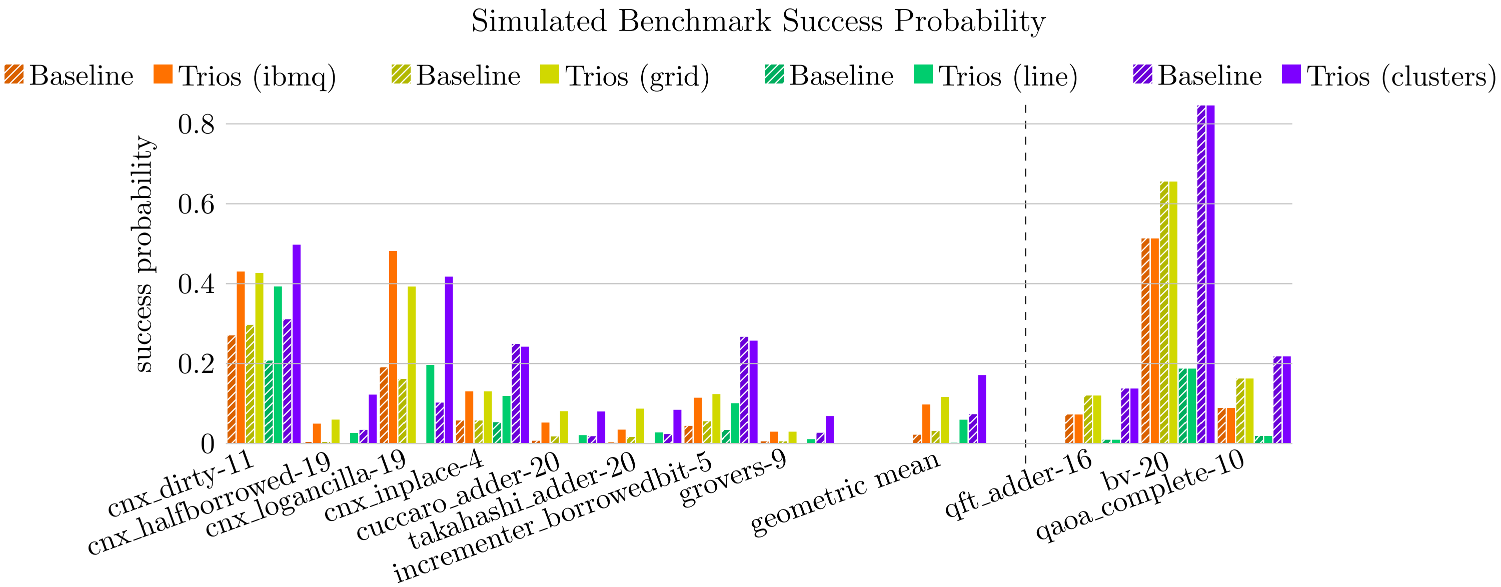


Figure 4.9: Simulated upper-bounds on the program execution success probability on various hardware (using 20x lower idle and gate errors than Johannesburg). Neighboring pairs of bars compare the baseline with Trios compiled for Johannesburg. Higher is better when comparing pairs of bars with the same color. The geometric mean success rates over the benchmarks that use Toffoli gate for each device type respectively are 2.2%→9.8%, 3.2%→12%, 0.19%→6.0%, 7.3%→17%. The rightmost three benchmarks contain zero Toffoli gates so have no change vs. the baseline.

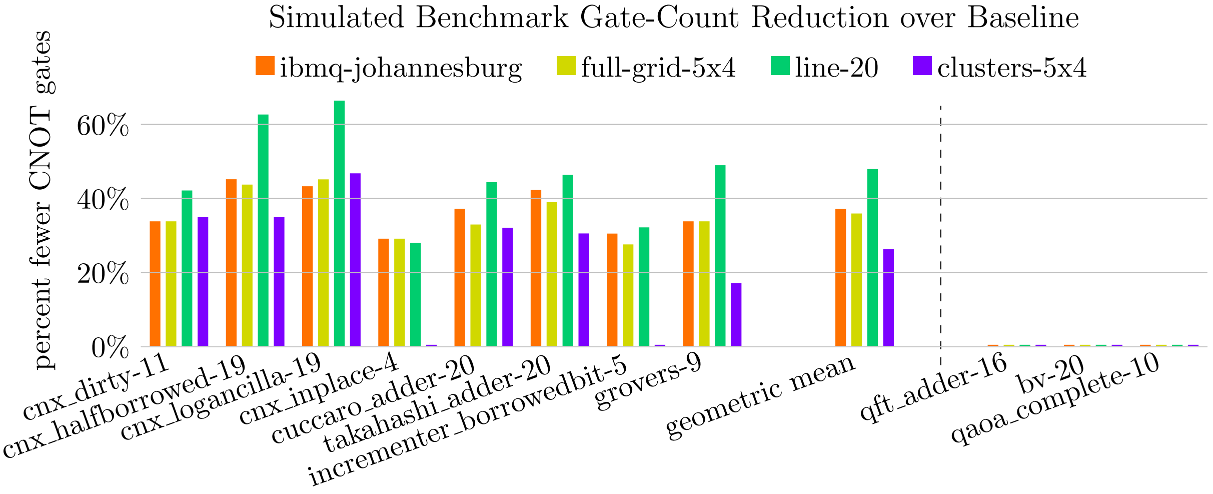


Figure 4.10: A comparison between the baseline and Trios for various hardware. Above 0% indicates benefit. All two-qubit gates (for communication and computation) are counted. The geometric mean reductions in gate counts are 37%, 36%, 48%, and 26% respectively. The rightmost three benchmarks contain zero Toffoli gates so have no change vs. the baseline.

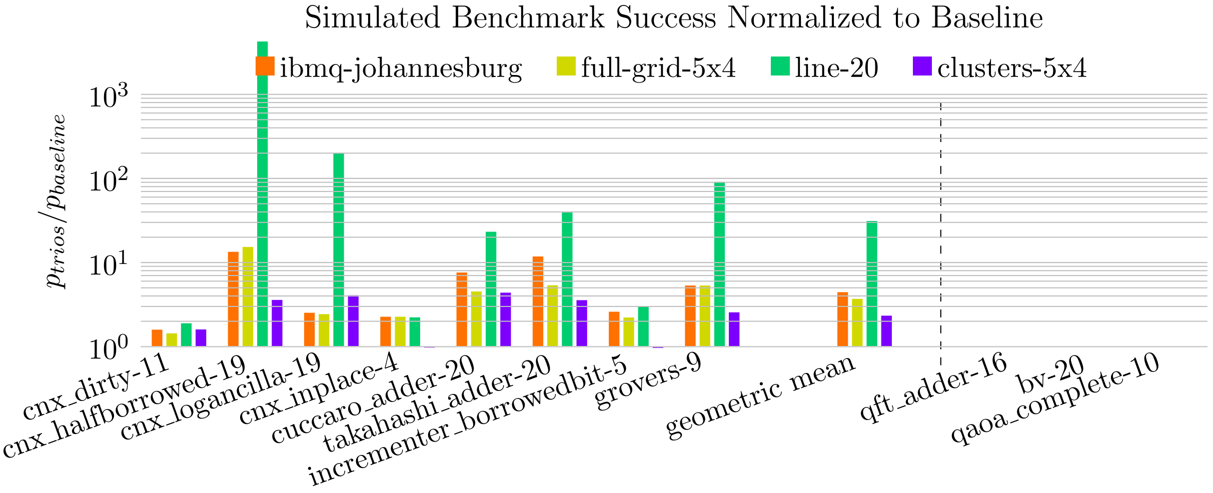


Figure 4.11: Normalized Figure 4.9 to show our consistent increase in program success with Trios. Above 10^0 indicates benefit. Some improvement factors are huge due to near-zero baseline success rates. The geometric mean increases in success rate are 4.4x, 3.7x, 31x, and 2.3x respectively. The rightmost three benchmarks contain zero Toffoli gates so have no change vs. the baseline.

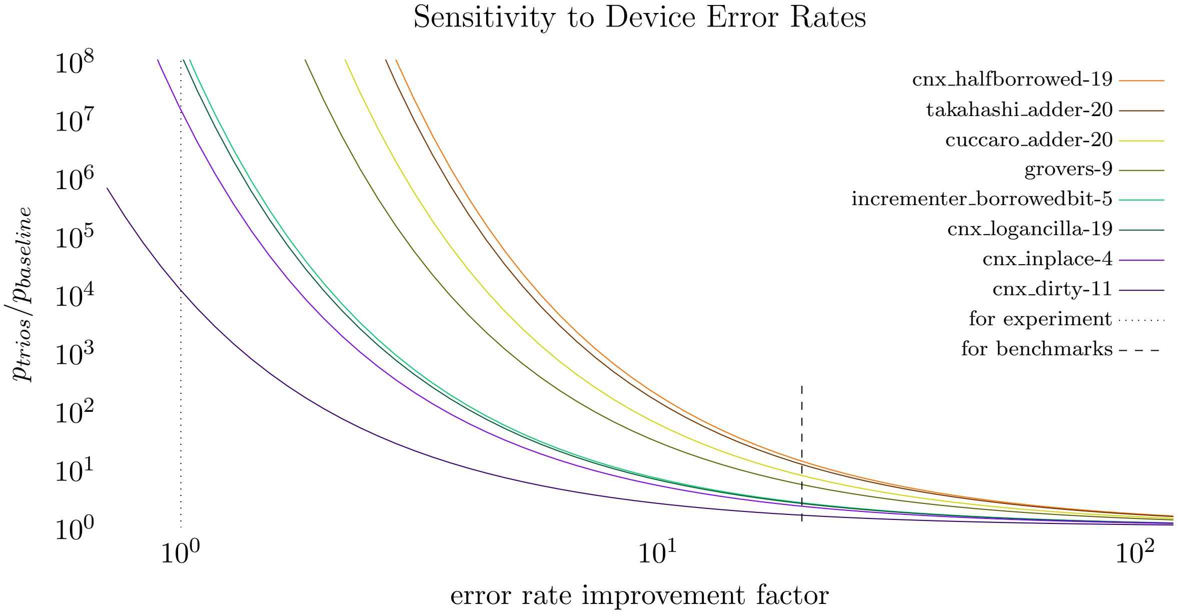


Figure 4.12: Factor of improvement in success rate in Trios over baseline for scaling gate error rates. The dotted line indicates current error rates on IBM Johannesburg and the dashed line (20x improvement) indicates values of the near future used in simulation. In our approximation of success rate factors of improvement in gate error rates lead to an exponential fall off in success ratios, as expected. In the very near term, we expect Trios to drastically improve the execution of quantum programs.

specific decomposition. This makes a huge difference specifically with Toffolis on any square-grid-based device. One might choose to improve the solution found by an optimal compiler by always decomposing Toffolis to the 8-CNOT version before optimally routing, but this will still limit the solution. There are multiple possible qubit orders for the decomposition and the best can only be selected after the routing pass.

4.6.4 Simulation Sensitivity to Error Rates

For our simulations we use an error model (20x better than current errors on Johannesburg) which is forward looking. As errors improve, we expect Trios to have a reduced impact on program success rates since gate errors will contribute less and less to program failure though

Trios will never perform worse than the baseline. In Figure 4.12 we study the sensitivity of simulation results to two qubit error rates beginning with current IBM error rates. For poor error rates, the benefit of Trios is extremely large, owed to the fact that programs compiled with the baseline have probabilities of success very close to 0. In our simplified simulation framework, as error rates improve we expect an exponential drop off in improvement with the most advantage obtained with current error rates.

4.7 Summary

We present a new quantum compilation structure, Trios, with a split decomposition pass to greatly reduce compiled communication cost and enable architecture-tuned decompositions. We specifically target the three-qubit Toffoli operation to capture program structure enabling more optimal compiled circuits. Because current quantum computers are especially error prone, they require high levels of optimization to reduce gate counts and maximize the probability the compiled program will succeed. Prior optimization strategies discarded hierarchy to better maximize flat program optimization but Trios shows that the extra structural information from program hierarchy can be more helpful than the increased flexibility of a flattened program structure.

Orchestrated Trios both greatly improves the effectiveness of qubit routing given newly exposed program structure and, additionally, improves decompositions with a connectivity-aware second pass. These both greatly benefit the program success rate, a critical metric for today's error-prone and resource-constrained quantum computers. We hope this inspires more hierarchically designed NISQ algorithms now that we have shown that keeping the abstraction of hierarchy and reorganizing compilation passes can help bridge the gap between these noisy quantum hardware and practical applications.

CHAPTER 5

CONCLUSION

From the three cases we present and evaluate, we find that the right abstraction enables system design, compiler design, and program design to align. Each of the abstractions intrinsically represented quantum computation in a way that preserved the data locality of the application in ways that reduced communication costs during program execution. This aspect of keeping related data close-by emerged from fundamental constraints of quantum computing technology and we found useful abstractions that worked within this. First, we treat additional logical levels as scratch space to enable program subroutines with simplified communication patterns. Qubit-local memory enables a third spacial dimension, improving fault-tolerant operations and fault-tolerant data movement. And re-introducing the hierarchical program abstraction, in a small way, can greatly improve compiler heuristics and enable smarter compiler passes.

These abstractions are just the tip of the iceberg of what will eventually be an entire ecosystem of abstractions that fit together and be integrated into the quantum computer scientist's programming languages, hardware stack, and toolchain. As the stack evolves, the computing paradigm around quantum computing will grow more into its own, fundamentally new niche and be more than a classical computer under superposition.

REFERENCES

Héctor Abraham, AduOffei, Rochisha Agarwal, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Eli Arbel, Abraham Asfaw, Carlos Azaustre, AzizNgoueya, Aman Bansal, Panagiotis Barkoutsos, George Barron, Luciano Bello, Yael Ben-Haim, Daniel Bevenius, Lev S. Bishop, Sorin Bolos, Samuel Bosch, Sergey Bravyi, David Bucher, Artemiy Burov, Fran Cabrera, Padraic Calpin, Lauren Capelluto, Jorge Carballo, Ginés Carrascal, Adrian Chen, Chun-Fu Chen, Edward Chen, Jielun (Chris) Chen, Richard Chen, Jerry M. Chow, Spencer Churchill, Christian Claus, Christian Clauss, Romilly Cocking, Abigail J. Cross, Andrew W. Cross, Simon Cross, Juan Cruz-Benito, Chris Culver, Antonio D. Córcoles-Gonzales, Sean Dague, Tareq El Dandachi, Marcus Daniels, Matthieu Dartiailh, DavideFrr, Abdón Rodríguez Davila, Anton Dekusar, Delton Ding, Jun Doi, Eric Drechsler, Drew, Eugene Dumitrescu, Karel Dumon, Ivan Duran, Kareem EL-Safty, Eric Eastman, Pieter Eendebak, Daniel Egger, Mark Everitt, Paco Martín Fernández, Axel Hernández Ferrera, Romain Fouilland, FranckChevallier, Albert Frisch, Andreas Fuhrer, MELVIN GEORGE, Julien Gacon, Borja Godoy Gago, Claudio Gambella, Jay M. Gambetta, Adhisha Gammanpila, Luis Garcia, Shelly Garion, Austin Gilliam, Aditya Giridharan, Juan Gomez-Mosquera, Salvador de la Puente González, Jesse Gorzinski, Ian Gould, Donny Greenberg, Dmitry Grinko, Wen Guan, John A. Gunnels, Mikael Haglund, Isabel Haide, Ikko Hamamura, Omar Costa Hamido, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Stefan Hillmich, Hiroshi Horii, Connor Howington, Shaohan Hu, Wei Hu, Rolf Huisman, Haruki Imai, Takashi Imamichi, Kazuaki Ishizaki, Raban Iten, Toshinari Itoko, JamesSeaward, Ali Javadi, Ali Javadi-Abhari, Jessica, Madhav Jivrajani, Kiran Johns, Jonathan-Shoemaker, Tal Kachmann, Naoki Kanazawa, Kang-Bae, Anton Karazeev, Paul Kassebaum, Spencer King, Knabberjoe, Yuri Kobayashi, Arseny Kovyrshin, Rajiv Krishnakumar, Vivek Krishnan, Kevin Krsulich, Gawel Kus, Ryan LaRose, Enrique Lacal, Raphaël Lambert, John Lapeyre, Joe Latone, Scott Lawrence, Christina Lee, Gushu Li, Dennis Liu, Peng Liu, Yunho Maeng, Aleksei Malyshev, Joshua Manela, Jakub Marecek, Manoel Marques, Dmitri Maslov, Dolph Mathews, Atsushi Matsuo, Douglas T. McClure, Cameron McGarry, David McKay, Dan McPherson, Srujan Meesala, Thomas Metcalfe, Martin Mevissen, Antonio Mezzacapo, Rohit Midha, Zlatko Minev, Abby Mitchell, Nikolaj Moll, Michael Duane Mooring, Renier Morales, Niall Moran, MrF, Prakash Murali, Jan Müggenburg, David Nadlinger, Ken Nakanishi, Giacomo Nannicini, Paul Nation, Edwin Navarro, Yehuda Naveh, Scott Wyman Neagle, Patrick Neuweiler, Pradeep Niroula, Hassi Norlen, Lee James O'Riordan, Oluwatobi Ogunbayo, Pauline Ollitrault, Steven Oud, Dan Padilha, Hanhee Paik, Yuchen Pang, Simone Perriello, Anna Phan, Francesco Piro, Marco Pistoia, Christophe Piveteau, Alejandro Pozas-iKerstjens, Viktor Prutyanov, Daniel Puzzuoli, Jesús Pérez, Quintiii, Rafey Iqbal Rahman, Arun Raja, Nipun Ramagiri, Anirudh Rao, Rudy Raymond, Rafael Martín-Cuevas Redondo, Max Reuter, Julia Rice, Diego M. Rodríguez, RohithKarur, Max Rossmannek, Mingi Ryu, Tharrmashastha SAPV, SamFerracin, Martin Sandberg, Ritvik Sapra, Hayk Sargsyan, Aniruddha Sarkar, Ninad Sathaye, Bruno Schmitt, Chris Schnabel, Zachary Schoenfeld, Travis L. Scholten, Eddie Schoute, Joachim Schwarm, Ismael Faro Sertage, Kanav Setia, Nathan Shammah, Yunong Shi, Adenilton Silva, Andrea Simonetto, Nick Singstock, Yukio

Siraichi, Iskandar Sitdikov, Seyon Sivarajah, Magnus Berg Sletfjerding, John A. Smolin, Mathias Soeken, Igor Olegovich Sokolov, SooluThomas, Starfish, Dominik Steenken, Matt Stypulkoski, Shaojun Sun, Kevin J. Sung, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Soolu Thomas, Mathieu Tillet, Maddy Tod, Miroslav Tomasik, Enrique de la Torre, Kenso Trabing, Matthew Treinish, TrishaPe, Wes Turner, Yotam Vaknin, Carmen Recio Valcarce, Francois Varchon, Almudena Carrera Vazquez, Victor Villar, Desiree Vogt-Lee, Christophe Vuillot, James Weaver, Rafal Wieczorek, Jonathan A. Wildstrom, Erick Winston, Jack J. Woehr, Stefan Woerner, Ryan Woo, Christopher J. Wood, Ryan Wood, Stephen Wood, Steve Wood, James Wootton, Daniyar Yeralin, David Yonge-Mallo, Richard Young, Jessie Yu, Christopher Zachow, Laura Zdanski, Helena Zhang, Christa Zoufal, Zoufalc, a kapila, a matsuo, bcamorrison, brandhsn, chlorophyll zz, dekel.meirom, dekool, dime10, drholmie, dtrenev, ehchen, elfrocampeador, faisaldebouni, fanizzamarco, gadial, gruu, hhorii, hykavitha, jagunther, jliu45, kanejess, klinvill, kurarrr, lerongil, ma5x, merav aharoni, michelle4654, ordmoj, rmoyard, saswati qiskit, sethmerkel, strickroman, sumitpuri, tigerjack, toural, vvilpas, welien, willhbang, yang.luh, yotamvakninibm, and Mantas Čepulkovskis. Qiskit: An open-source framework for quantum computing, 2019.

Matthew Amy. Algorithms for the optimization of quantum circuits. Master's thesis, University of Waterloo, 2013. URL `http://hdl.handle.net/10012/7818`.

Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019. doi: 10.1038/s41586-019-1666-5. URL `https://doi.org/10.1038/s41586-019-1666-5`.

T. Bækkegaard, L. B. Kristensen, N. J. S. Loft, C. K. Andersen, D. Petrosyan, and N. T. Zinner. Superconducting qutrit-qubit circuit: A toolbox for efficient quantum gates, 2018.

Jonathan M. Baker, Casey Duckering, Alexander Hoover, and Frederic T. Chong. Decomposing quantum generalized toffoli with an arbitrary number of ancilla. *arXiv preprint*, April 2019.

Jonathan M. Baker, Casey Duckering, and Frederic T. Chong. Efficient quantum circuit decompositions via intermediate qudits. In *2020 IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 303–308, 2020a. doi: 10.1109/ISMVL49045.2020.9345604.

Jonathan M. Baker, Casey Duckering, Pranav Gokhale, and Andrew Litteken. Quantum circuit benchmarks. `https://github.com/jmbaker94/quantumcircuitbenchmarks`, 2020b.

Jonathan M. Baker, Casey Duckering, Alexander Hoover, and Frederic T. Chong. Time-sliced quantum circuit partitioning for modular architectures. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*, pages 98–107, 2020c. doi: 10.1145/3387902.3392617.

Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, November 1995. doi: 10.1103/PhysRevA.52.3457.

R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov, A. N. Cleland, and John M. Martinis. Logic gates at the surface code threshold: Superconducting qubits poised for fault-tolerant quantum computing. 2014. doi: 10.1038/nature13171.

Edwin Barnes, Christian Arenz, Alexander Pitchford, and Sophia E. Economou. Fast microwave-driven three-qubit gates for cavity-coupled superconducting qubits. 2016. doi: 10.1103/PhysRevB.96.024504.

Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 11–20, June 1993. ISBN 0897915917. doi: 10.1145/167088.167097.

Jacob Biamonte and Ville Bergholm. Tensor networks in a nutshell, 2017.

Alex Bocharov, Martin Roetteler, and Krysta M. Svore. Factoring with qutrits: Shor's algorithm on ternary and metaplectic quantum architectures. 2016. doi: 10.1103/PhysRevA.96.012306.

Héctor Bombín. Gauge color codes: optimal transversal gates and gauge fixing in topological stabilizer codes. *New Journal of Physics*, 17(8):083002, 2015.

Kyle E. C. Booth, Minh Do, J. Christopher Beck, Eleanor Rieffel, Davide Venturelli, and Jeremy Frank. Comparing and integrating constraint programming and temporal planning for quantum circuit compilation, 2018.

Sergey Bravyi and Jeongwan Haah. Magic-state distillation with low overhead. *Physical Review A*, 86(5):052329, 2012.

Kenneth R Brown, Jungsang Kim, and Christopher Monroe. Co-designing a scalable quantum computer with trapped atomic ions. *npj Quantum Information*, 2:16034, 2016.

Natalie C. Brown and Kenneth R. Brown. Comparing zeeman qubits to hyperfine qubits in the context of the surface code: ^{174}Yb^{+} and ^{171}Yb^{+}. *Phys. Rev. A*, 97:052301, May 2018. doi: 10.1103/PhysRevA.97.052301. URL `https://link.aps.org/doi/10.1103/PhysRevA.97.052301`.

Todd A. Brun. A simple model of quantum trajectories. 2001. doi: 10.1119/1.1475328.

Rui Chao and Ben W. Reichardt. Quantum error correction with only two extra qubits. *Phys. Rev. Lett.*, 121:050502, Aug 2018. doi: 10.1103/PhysRevLett.121.050502. URL `https://link.aps.org/doi/10.1103/PhysRevLett.121.050502`.

A. Yu. Chernyavskiy, Vad. V. Voevodin, and Vl. V. Voevodin. Parallel computational structure of noisy quantum circuits simulation. *Lobachevskii Journal of Mathematics*, 39(4):494–502, May 2018. ISSN 1818-9962. doi: 10.1134/S1995080218040042. URL `https://doi.org/10.1134/S1995080218040042`.

Yulin Chi, Jieshan Huang, Zhanchuan Zhang, Jun Mao, Zinan Zhou, Xiaojiong Chen, Chonghao Zhai, Jueming Bao, Tianxiang Dai, Huihong Yuan, Ming Zhang, Daoxin Dai, Bo Tang, Yan Yang, Zhihua Li, Yunhong Ding, Leif K. Oxenløwe, Mark G. Thompson, Jeremy L. O'Brien, Yan Li, Qihuang Gong, and Jianwei Wang. A programmable qudit-based quantum processor. *Nature communications*, 13(1):1–10, 2022.

Cirq. Cirq: A python framework for creating, editing, and invoking noisy intermediate scale quantum (nisq) circuits. `https://github.com/quantumlib/cirq`, 2018.

D. G. Cory, W. Mass, M. Price, E. Knill, R. Laflamme, W. H. Zurek, T. F. Havel, and S. S. Somaroo. Experimental quantum error correction. 1998. doi: 10.1103/PhysRevLett.81.2152.

Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. On the qubit routing problem. 135:5:1–5:32, February 2019. ISSN 1868-8969. doi: 10.4230/LIPIcs.TQC.2019.5. URL `http://drops.dagstuhl.de/opus/volltexte/2019/10397`.

Steven A. Cuccaro, Thomas G. Draper, Samuel A. Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit. *arXiv preprint*, October 2004.

Eric Dennis. Toward fault-tolerant quantum computation without concatenation. 1999. doi: 10.1103/PhysRevA.63.052314.

Yao-Min Di and Hai-Rui Wei. Elementary gates for ternary quantum logic circuit, 2011.

Yongshan Ding, Adam Holmes, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic Chong. Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 828–840. IEEE, 2018.

Thomas G. Draper. Addition on a quantum computer, 2000.

Thomas G Draper, Samuel A Kutin, Eric M Rains, and Krysta M Svore. A logarithmic-depth quantum carry-lookahead adder. *Quantum Information & Computation*, 6(4):351–369, 2006.

Casey Duckering, Jonathan M. Baker, David I. Schuster, and Frederic T. Chong. Virtualized logical qubits: A 2.5d architecture for error-corrected quantum computing. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 173–185, 2020. doi: 10.1109/MICRO50266.2020.00026.

Casey Duckering, Jonathan M. Baker, Andrew Litteken, and Frederic T. Chong. Orchestrated trios: Compiling for efficient communication in quantum programs with 3-qubit gates. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS 2021, pages 375–385, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383172. doi: 10.1145/3445814.3446718. URL https://doi.org/10.1145/3445814.3446718.

Nathan Earnest, Srivatsan Chakram, Yao Lu, Nicholas Irons, Ravi K. Naik, Nelson Leung, Leo Ocola, David A. Czaplewski, Brian Baker, Jay Lawrence, Jens Koch, and David I. Schuster. Realization of a λ system with metastable states of a capacitively-shunted fluxonium. 2017. doi: 10.1103/PhysRevLett.120.150504.

Yale Fan. Applications of multi-valued quantum algorithms. 2008. doi: 10.1109/ISMVL.2007.3.

Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint*, November 2014.

Arkady Fedorov, Lars Steffen, Matthias Baur, M. P. da Silva, and Andreas Wallraff. Implementation of a toffoli gate with superconducting circuits. 2011. doi: 10.1038/nature10713.

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

Austin G Fowler, Adam C Whiteside, and Lloyd CL Hollenberg. Towards practical classical processing for the surface code. *Physical review letters*, 108(18):180501, 2012b.

Jay Gambetta and Sarah Sheldon. Cramming more power into a quantum device, March 2019. URL https://www.ibm.com/blogs/research/2019/03/power-quantum-device/.

Craig Gidney. Constructing large controlled nots, 2015. URL http://algassert.com/circuits/2015/06/05/Constructing-Large-Controlled-Nots.html.

Craig Gidney. Factoring with n+2 clean qubits and n-1 dirty qubits, 2017.

Craig Gidney and Martin Ekerå. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *arXiv preprint arXiv:1905.09749*, 2019.

Steven M. Girvin. Circuit qed: Superconducting qubits coupled to microwave photons.

Pranav Gokhale, Jonathan M Baker, Casey Duckering, Natalie C Brown, Kenneth R Brown, and Frederic T Chong. Asymptotic improvements to quantum circuits via qutrits. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 554–566. ACM, 2019.

Google Bristlecone. A preview of Bristlecone, Google's new quantum processor, March 2018. URL https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html.

Andrew D. Greentree, S. G. Schirmer, F. Green, Lloyd C. L. Hollenberg, A. R. Hamilton, and R. G. Clark. Maximizing the hilbert space for a finite number of distinguishable quantum states. 2003. doi: 10.1103/PhysRevLett.92.097901.

Lov 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. ACM, 1996. doi: 10.1145/237814.237866.

Gian Giacomo Guerreschi and Jongsoo Park. Two-step approach to scheduling quantum circuits. *Quantum Science and Technology*, 3(4):045003, July 2018. doi: 10.1088/2058-9565/aacf0b.

Thomas Häner, Martin Roetteler, and Krysta M. Svore. Factoring using 2n+2 qubits with toffoli based modular multiplication. 2016.

Connor T Hann, Chang-Ling Zou, Yaxing Zhang, Yiwen Chu, Robert J Schoelkopf, Steven M Girvin, and Liang Jiang. Hardware-efficient quantum random access memory with hybrid quantum acoustic systems. *arXiv preprint arXiv:1906.11340*, 2019.

Y. He, M.-X. Luo, E. Zhang, H.-K. Wang, and X.-F. Wang. Decompositions of n-qubit Toffoli Gates with Linear Circuit Complexity. *International Journal of Theoretical Physics*, 56: 2350–2361, July 2017. doi: 10.1103/PhysRevA.75.022313.

Yuichi Hirata, Masaki Nakanishi, Shigeru Yamashita, and Yasuhiko Nakashima. An efficient conversion of quantum circuits to a linear nearest neighbor architecture. *Quantum Information and Computation*, 11(1):142–166, January 2011. ISSN 1533-7146.

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

IBM Devices. IBM quantum devices. https://quantumexperience.ng.bluemix.net/qx/devices, 2018. Accessed: 2018-05-16.

S. S. Ivanov, H. S. Tonchev, and N. V. Vitanov. Time-efficient implementation of quantum search with qudits. 2012. doi: 10.1103/PhysRevA.85.062321.

Ali Javadi-Abhari, Pranav Gokhale, Adam Holmes, Diana Franklin, Kenneth R. Brown, Margaret Martonosi, and Frederic T. Chong. Optimized surface code communication in superconducting quantum computers. 2017. doi: 10.1145/3123939.3123949.

Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. ScaffCC: Scalable compilation and analysis of quantum programs. *Parallel Computing*, 45:2–17, 2015. doi: 10.1016/j.parco.2014.12.001.

J. R. Johansson, P. D. Nation, and Franco Nori. Qutip: An open-source python framework for the dynamics of open quantum systems. 2011. doi: 10.1016/j.cpc.2012.02.021.

J. R. Johansson, P. D. Nation, and Franco Nori. Qutip 2: A python framework for the dynamics of open quantum systems. 2012. doi: 10.1016/j.cpc.2012.11.019.

N. Khammassi, I. Ashraf, X. Fu, C. G. Almudever, and K. Bertels. Qx: A high-performance quantum computer simulation platform. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '17, pages 464–469, 3001 Leuven, Belgium, Belgium, 2017. European Design and Automation Association. URL http://dl.acm.org/citation.cfm?id=3130379.3130487.

Mozammel H. A. Khan and Marek A. Perkowski. Quantum ternary parallel adder/subtractor with partially-look-ahead carry. *J. Syst. Archit.*, 53(7):453–464, July 2007. ISSN 1383-7621. doi: 10.1016/j.sysarc.2007.01.007. URL http://dx.doi.org/10.1016/j.sysarc.2007.01.007.

A. B. Klimov, R. Guzmán, J. C. Retamal, and C. Saavedra. Qutrit quantum computer with trapped ions. *Phys. Rev. A*, 67:062313, Jun 2003. doi: 10.1103/PhysRevA.67.062313. URL https://link.aps.org/doi/10.1103/PhysRevA.67.062313.

Philip Krantz, Morten Kjaergaard, Fei Yan, Terry P. Orlando, Simon Gustavsson, and William D. Oliver. A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews*, 6(2):021318, June 2019. doi: 10.1063/1.5089550.

Michael Kues, Christian Reimer, Piotr Roztocki, Luis Romero Cortés, Stefania Sciara, Benjamin Wetzel, Yanbing Zhang, Alfonso Cino, Sai T. Chu, Brent E. Little, David J. Moss, Lucia Caspani, José Azaña, and Roberto Morandotti. On-chip generation of high-dimensional entangled quantum states and their coherent control. *Nature*, 546:622 EP –, 06 2017. URL http://dx.doi.org/10.1038/nature22986.

Andrew J Landahl, Jonas T Anderson, and Patrick R Rice. Fault-tolerant quantum computing with color codes. *arXiv preprint arXiv:1108.5738*, 2011.

B. P. Lanyon, M. Barbieri, M. P. Almeida, T. Jennewein, T. C. Ralph, K. J. Resch, G. J. Pryde, J. L. O'Brien, A. Gilchrist, and A. G. White. Quantum computing using shortcuts through higher dimensions. 2008. doi: 10.1038/nphys1150.

Lingling Lao, Bas van Wee, Imran Ashraf, J van Someren, Nader Khammassi, Koen Bertels, and Carmen G Almudever. Mapping of lattice surgery-based quantum circuits on surface code architectures. *Quantum Science and Technology*, 4(1):015005, 2018.

H. Y. Li, C. W. Wu, W. T. Liu, P. X. Chen, and C. Z. Li. Fast quantum search algorithm for databases of arbitrary size and its implementation in a cavity QED system. *Physics Letters A*, 375:4249–4254, November 2011. doi: 10.1016/j.physleta.2011.10.016.

N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Experimental comparison of two quantum computing architectures. 2017. doi: 10.1073/pnas.1618020114.

Daniel Litinski. Magic state distillation: Not as costly as you think. *arXiv preprint arXiv:1905.06903*, 2019a.

Daniel Litinski. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum*, 3:128, 2019b.

John Martinis. Quantum supremacy using a programmable superconducting processor, 11 2019. URL `https://youtu.be/FklMpRiTeTA`. Institute for Quantum Information and Matter Seminar at the California Institute of Technology.

Dmitri Maslov, Gerhard W. Dueck, D. Michael Miller, and Camille Negrevergne. Quantum circuit simplification and level compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(3):436–444, March 2008. doi: 10.1109/TCAD.2007.911334.

Daniel Miller, Timo Holz, Hermann Kampermann, and Dagmar Bruß. Propagation of generalized pauli errors in qudit clifford circuits, 2018.

Steven Moses, Juan Pino, Joan Dreiling, Caroline Figgatt, John Gaebler, Michael Allman, Charles Baldwin, Michael Foss-Feig, David Hayes, Karl Mayer, Ciaran Ryan-Anderson, and Brian Neyenhuis. Demonstration of the QCCD trapped-ion quantum computer architecture. *Bulletin of the American Physical Society*, June 2020.

Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1015–1029, April 2019. doi: 10.1145/3297858.3304075.

Prakash Murali, Dripto M. Debroy, Kenneth R. Brown, and Margaret Martonosi. Architecting noisy intermediate-scale trapped ion quantum computers. In *2020 ACM/IEEE 47th*

Annual International Symposium on Computer Architecture (ISCA), pages 529–542, June 2020a. doi: 10.1109/ISCA45697.2020.00051.

Prakash Murali, David C. McKay, Margaret Martonosi, and Ali Javadi-Abhari. Software mitigation of crosstalk on noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1016, March 2020b. doi: 10.1145/3373376.3378477.

Ashok Muthukrishnan and C. R. Stroud. Multivalued logic gates for quantum computation. *Phys. Rev. A*, 62:052309, Oct 2000. doi: 10.1103/PhysRevA.62.052309. URL `https://link.aps.org/doi/10.1103/PhysRevA.62.052309`.

R. K. Naik, N. Leung, S. Chakram, Peter Groszkowski, Y. Lu, N. Earnest, D. C. McKay, Jens Koch, and D. I. Schuster. Random access quantum information processors using multimode circuit quantum electrodynamics. *Nature Communications*, 8(1):1904, 2017. doi: 10.1038/s41467-017-02046-6. URL `https://doi.org/10.1038/s41467-017-02046-6`.

Yunseong Nam, Neil J. Ross, Yuan Su, Andrew M. Childs, and Dmitri Maslov. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information*, 4(1):1–12, March 2018. doi: 10.1038/s41534-018-0072-4.

Matthew Gary Neeley. *Generation of three-qubit entanglement using Josephson phase qubits.* PhD thesis, University of California, Santa Barbara, 2010.

Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, New York, NY, USA, 2011. ISBN 1107002176, 9781107002173.

Matthew Otten and Stephen Gray. Accounting for errors in quantum algorithms via individual error reduction, 2018.

Archimedes Pavlidis and Emmanuel Floratos. Arithmetic circuits for multilevel qudits based on quantum fourier transform, 2017.

John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2, August 2018. ISSN 2521-327X. doi: 10.22331/q-2018-08-06-79.

Qutrits Code. Code for asymptotic improvements to quantum circuits via qutrits. `https://github.com/epiqc/qutrits`, 2019.

T. C. Ralph, K. J. Resch, and A. Gilchrist. Efficient toffoli gates using qudits. 2008. doi: 10.1103/PhysRevA.75.022313.

J. Randall, S. Weidt, E. D. Standing, K. Lake, S. C. Webster, D. F. Murgia, T. Navickas, K. Roth, and W. K. Hensinger. Efficient preparation and detection of microwave dressed-state qubits and qutrits with trapped ions. *Phys. Rev. A*, 91:012322, 01 2015. doi: 10.

1103/PhysRevA.91.012322. URL `https://link.aps.org/doi/10.1103/PhysRevA.91.012322`.

J. Randall, A. M. Lawrence, S. C. Webster, S. Weidt, N. V. Vitanov, and W. K. Hensinger. Generation of high-fidelity quantum control methods for multilevel systems. *Phys. Rev. A*, 98:043414, 10 2018. doi: 10.1103/PhysRevA.98.043414. URL `https://link.aps.org/doi/10.1103/PhysRevA.98.043414`.

Matthew Reagor, Wolfgang Pfaff, Christopher Axline, Reinier W. Heeres, Nissim Ofek, Katrina Sliwa, Eric Holland, Chen Wang, Jacob Blumoff, Kevin Chou, Michael J. Hatridge, Luigi Frunzio, Michel H. Devoret, Liang Jiang, and Robert J. Schoelkopf. A quantum memory with near-millisecond coherence in circuit qed. 2015. doi: 10.1103/PhysRevB.94.014506.

Lidia Ruiz-Perez and Juan Carlos Garcia-Escartin. Quantum arithmetic with the quantum Fourier transform. *Quantum Information Processing*, 16(6):152:1–152:14, April 2017. doi: 10.1007/s11128-017-1603-1.

Zahra Sasanian and D. Michael Miller. Reversible and quantum circuit optimization: A functional approach. In *International Workshop on Reversible Computation*, pages 112–124. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-36315-3. doi: 10.1007/978-3-642-36315-3_9.

Ruediger Schack and Todd A. Brun. A C++ library using quantum trajectories to solve quantum master equations. 1996. doi: 10.1016/S0010-4655(97)00019-2.

Norbert Schuch. Implementation of quantum algorithms with Josephson charge qubits. *Universität Regensburg*, December 2002. URL `https://epub.uni-regensburg.de/1511/`.

Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997. ISSN 0097-5397. doi: 10.1137/S0097539795293172. URL `http://dx.doi.org/10.1137/S0097539795293172`.

Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintão Pereira. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, pages 113–125, February 2018. doi: 10.1145/3168822.

Robert S. Smith, Michael J. Curtis, and William J. Zeng. A practical quantum instruction set architecture. *arXiv preprint*, 2016.

Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. Quantum addition circuits and unbounded fan-out. *arXiv preprint*, October 2009.

Bochen Tan and Jason Cong. Optimal layout synthesis for quantum computing. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, November 2020.

Swamit S. Tannu and Moinuddin Qureshi. Ensemble of diverse mappings: Improving reliability of quantum computers by orchestrating dissimilar mistakes. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 253–265, October 2019. doi: 10.1145/3352460.3358257.

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

Trios Code. Source code for orchestrated trios. `https://github.com/cduck/orchestrated-trios`, 2021.

Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. Compiling quantum circuits to realistic hardware architectures using temporal planners. 2017. doi: 10.1088/2058-9565/aaa331.

VLQ Code. Simulation source code for virtualized logical qubits. `https://github.com/cduck/VLQ`, 2020.

Y. Wang and M. Perkowski. Improved complexity of quantum oracles for ternary grover algorithm for graph coloring. In *2011 41st IEEE International Symposium on Multiple-Valued Logic*, pages 294–301, May 2011. doi: 10.1109/ISMVL.2011.42.

Robert Wille, Aaron Lye, and Rolf Drechsler. Optimal SWAP gate insertion for nearest neighbor quantum circuits. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 489–494. IEEE, January 2014. doi: 10.1109/ASPDAC.2014.6742939.

Robert Wille, Oliver Keszocze, Marcel Walter, Patrick Rohrs, Anupam Chattopadhyay, and Rolf Drechsler. Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 292–297. IEEE, January 2016. doi: 10.1109/ASPDAC.2016.7428026.

Robert Wille, Lukas Burgholzer, and Alwin Zulehner. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, June 2019.