THE UNIVERSITY OF CHICAGO

ADAPTING COMPILATION STRATEGIES FOR ARCHITECTURE-SPECIFIC FEATURES 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
ANDREW LITTEKEN

CHICAGO, ILLINOIS
JUNE 2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I want to first thank my advisor Fred Chong for his guidance, flexibility and opportunities he has provided me with over the course of my PhD. I'd also like my committee Hank Hoffmann and Robert Rand who have given their time to advise me on my disseration and previous projects. I'd like to also thank all of my collaborators - Jonathan Baker, Lennart Maximillian Siefert, Jason Chadwick, Natalia Nottingham, Casey Duckering, and Hannes Bernien.

I would also like to thank all of those who have encouraged, supported and offered their advice as I've worked towards my PhD. Through all of my studies, I have had many incredible teachers, professors, colleagues and friends who prepared me for whatever came next. And, I would like to thank all of my family, who have always been on my side and made it possible to be where I am today.

ABSTRACT

Quantum Computing is in an era of growing pains. Quantum algorithms become more tuned to the computational capabilities of near-term quantum devices, and the need for compilation techniques that fully utilize the architecture while avoiding potential pitfalls are a necessity. Many different types quantum devices are being developed each with its own advantages and unique features It is unclear which architecture is dominant, and all are in active development. Some problems, such as limited connectivity, imperfect gate execution, and lower coherence times are shared across architectures, and mapping, routing and communication algorithms have been developed to circumvent and avoid these issues. But, each architecture has its own set of specific problems to overcome. It is not simply the case that a single compilation pipeline or set of algorithms will be able to make the best use of each of these architectures. This work proposes an exploration into adapting a general quantum compiler algorithm to several different architectures.

This thesis will focus on two main architectures, Neutral Atom and Superconducting qudit devices. Neutral Atom architectures have additional flexibility in routing qubits beyond nearest neighbor connectivity, while incurring some serialization cost due to increasingly large “areas of restruction”. But, we are still able to take advantage of these more flexible interactions through careful mapping and routing. Additionally, we can craft strategies that make use of these features to circumvent Neutral Atom Architectures’ biggest execution downfall: atom loss. Many quantum architectures have natural extensibility to extra computational states beyond the traditional $|0\rangle$ and $|1\rangle$ used in classical computing and most quantum computation. These extra states can be used to reduce the number of operations, but at the cost of slower operations and decreased stability of the computational units. Once again, we can adapt an existing base algorithm to mix-and-match qubit-only, mixed-radix, and higher-radix operations to develop efficient program that make use less resources and increase the chances of successful quantum computation.

CHAPTER 1

INTRODUCTION

Quantum computing has seen a renewed explosion in development, investment and interest in the past decade as new technologies have been built and architectures with more qubits have been constructed such as those from IBM [40], Google [5, 55], Rigetti [89], IonQ [23] and many more. New machines have seen marked success over prior designs, and new applications have been developed to take advantage of the current state of quantum computing such as QAOA [35] and VQE [106], beyond the flagship algorithms such as Grover's Search [45] and Shor's factoring algorithm [97]. There are ongoing collaborations between scientific and technology-based communities as they continually push to develop this fundamentally different form of computation.

In classical computing, there is a fairly well defined strategy for general purpose computation. While there are certainly variations in certain processor architectures and instruction set differences, there has been consolidation on the Von-Neumann based architectures. This has allowed for the development of device-independent optimizations at a higher, intermediate representation level [65], where compilers have been able to take advantage of the similarities of a shared model to reduce program metrics such as runtime, memory usage, and code size. After device specific optimization has been performed, the program is compiled for a specific processor, using optimizations designed for those platforms, such as register allocation, instructions selection and specific assembly language optimizations for those architectures. It is a very sequential system due to the general principles underlying most computing platforms following the model demonstrated in Figure 1.1.

This remains true for many accelerators, such as GPUs and FPGAs. Programs may have to be written in certain dialects specific for these devices, but they still have a general strategy for computation. That is, strategies that work on one device, will work on another. There can be device-independent optimizations, before using device-specific features that

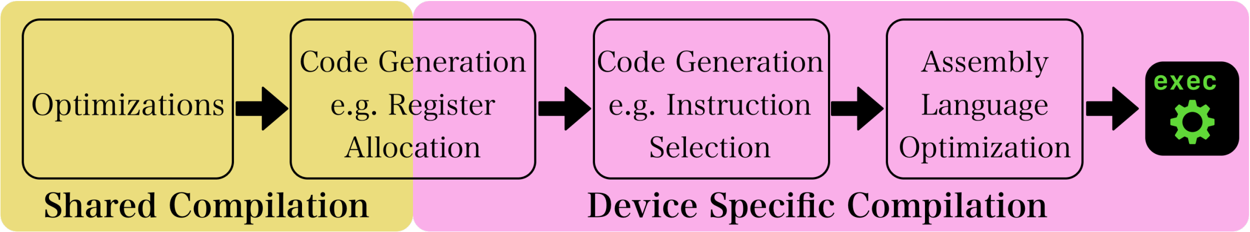

Optimizations
Code Generation e.g. Register Allocation
Code Generation e.g. Instruction Selection
Assembly Language Optimization
exec
Shared Compilation
Device Specific Compilation

Figure 1.1: A representation of a classical compilation pipeline, where multiple device independent optimizations are performed before passing the program off to a more device specific compilation.

take advantage of the design decisions for a specific piece of hardware.

At first glance, it may seem that quantum computers should be able to take advantage of the accelerator model. Each use the same general framework for quantum computation, but use different strategies to achieve the same goal. However, the current era of quantum computing, commonly referred to as the Noisy Intermediate-Scale Quantum Computing (NISQ) era [85], is defined by error-prone gates, unstable qubits and low qubit counts, and the best path towards fault-tolerant quantum computing is still unclear. There is active development among many different styles architectures such as superconducting, trapped ions, neutral atoms and photonic computing. Each has a different set of trade offs, whether that be ease of control, fidelity of operations, qubit connectivity, number of qubits, or qubit lifetime. There are some device independent optimizations that can be performed, such as gate cancellation and resynthesis, but can only reduce gate count and circuit duration at a high level. To be useful in the current era, each architectural style uses different optimization strategies throughout the entirely of the stack in order to be used most effectively.

Most quantum compilation frameworks such as tket [99], Qiskit [4], Cirq [31], include some amount of device independent optimizations to process a quantum circuit, translating it from the general circuit model to a new circuit that fits the connectivity and native gate constraints of the architecture in a compilation flow detailed in Figure 1.2. Usually, these optimization passes are parameterized by the details of the machine such as the qubit connectivity and the native gate set, but the optimization algorithms themselves are device

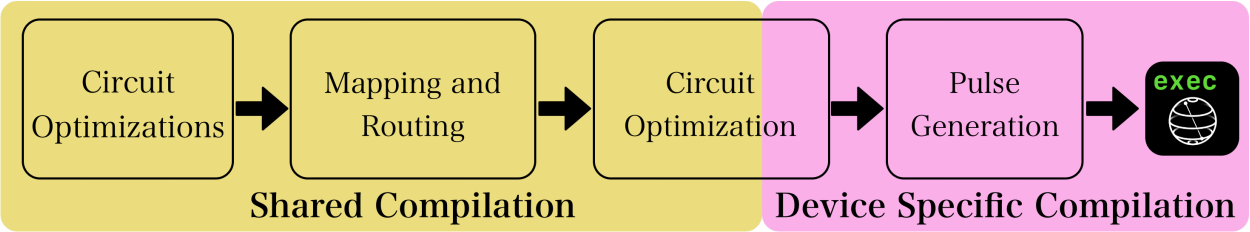


Figure 1.2: A representation of a classically inspired quantum compilation pipeline, where multiple device independent optimizations are performed before passing the program off to a more device specific compilation.

independent. These strategies fail to take into account the benefits and drawbacks of the architecture itself. While these strategies reduce gate count and circuit duration to some degree, they may actively work against the architecture in other cases. One example is the need for a specific router that schedules qubit communication to avoid crosstalk [75]. But simply, general purpose optimizations only get us so far. But, it is not practical to write routers and optimizations for each architecture-style, and the different features across architectures with the same underlying technology. This is especially true since the foundations of each algorithm will likely be very similar to one another across each architecture.

We can instead adapt general algorithms for quantum computing to be specific to hardware circuits are compiled for instead of inventing new technologies whole cloth. Instead of having device-independent optimizations feed into a device specific pipeline, we can share a common optimization aglorithm "shell" that is adapted to the specific, unique and efficient features of an architecture. This structure is visualized in Figure 1.3. While this work does not propose a compiler framework that successfully achieves this goal, it describes different compilation algorithms and how they can be adapted for several different architectural features.

We first describe the basics of quantum computing. We start with the general strategy for computation and how it can achieve expected the computational speed-ups. We also describe the circuit model that is usually used during compilation to describe quantum programs and is used when developing optimizations at the circuit level. We follow this with a description

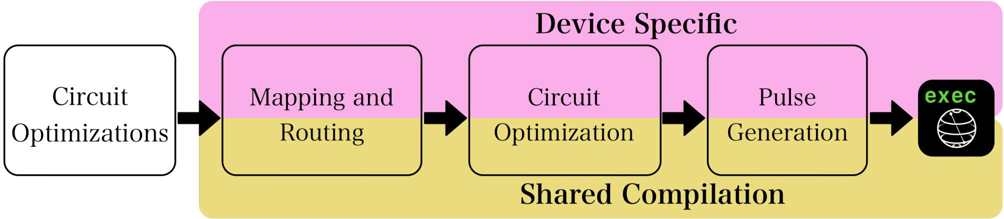


Figure 1.3: A representation of a quantum compilation pipeline that uses a similar algorithmic backing for each optimization step, but can be altered, using the specific of the device, for better compilation results.

of a general compiler algorithms developed previously and is used in several other works, but usually with a focus on a specific architecture or avoiding a specific kind of error [10, 73].

Second, we explore Neutral Atom architectures, an emerging architecture with unique features, such as long distance interactions, and the ability to perform native multi-qubit gates. But, the use of these features can induce serialization of gate execution causing circuit slowdowns. The unique features of neutral atoms have significant opportunities for compilation to take advantage of. Additionally, they also uncover a solution to a major detriment of this kind of architecture, atom loss. An inidivudal atom represents a single qubit in the system, and if an atom is lost, it could require an entire reloading of the architecture, a time intensive process. The unique features of neutral atoms allow for runtime and compiler opportunities to recover failed shots in software, preventing the need for a full reset of the device.

Third, we adapt our techniques for three different instances of higher-radix qudits. That is, quantum devices that can access the states beyond the traditional $|0\rangle$ and $|1\rangle$ states that are used for qubits. Higher radix devices are more error prone, requiring more calibration and longer gate times to accurately alter the state. This presents an interesting trade off for compilation. The extra computational space provides a reduction in physical devices required to perform computation. In theory, with access to four states, it would be possible to represent two times the information that an architecture with the same number of devices, but with only access to two level per device. Finding a trade-off between the extra coherence error and circuit duration versus fewer gates and saved space is a difficult compiler problem to solve. In the first application, we stay within the bounds of the native gate sets translated from qubit computation for higher radix qudits, extending work from [42] where qutrits were used for gate and circuit depth reduction. The second and third applications cover a compression strategy where we take advantage of the fact two qubits can be encoded into the four levels of a ququart quantum device. This significantly increases qubit connectivity, and the speed of some gates between two qubits encoded in the same ququart, as well as preventing breaking down of larger multiqubit gates into more smaller gates in some cases. However, qudits have lower lifetimes, and the speed of gates across ququarts are increased. We adapt compiler algorithms, and alter the native gate set to dynamically establish the best tradeoff between these different features and drawbacks.

These varied architecture demonstrate the need for more adaptable compiler frameworks as in an era of varied technologies and error. Fundamentally, it is important to develop quantum compiler pipelines built from common pieces that can be strategically customized to fit the unique features of a quantum architecture.

CHAPTER 2
GENERAL COMPILATION STRATEGIES FOR QUANTUM ARCHITECTURES

This chapter introduces the basics of how quantum programs are constructed and the constraints we will be working within when compiling quantum programs to quantum hardware.

2.1 Background: Quantum Programs

The fundamental unit of quantum computing is the quantum bit, or qubit, which exists as a linear superposition between the $|0\rangle$ and the $|1\rangle$ states. Most quantum programs manipulate a register of N quantum bits where the state of the qubits is given as a linear superposition of the 2^N basis vectors. This state evolves by the application of operations or gates. For example, the single qubit X gate transforms $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ into $X|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$. Some gates, such as the Hadamard gate, push the qubit into a state of superposition, existing in the $|0\rangle$ and $|1\rangle$ state at the same time. For example, the Hadamard gate transforms $|\psi\rangle = |0\rangle$ into $H|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. Half the time, we will measure this qubit to be in the $|0\rangle$ state, and half the time we will measure it to be in the $|1\rangle$ state.

Gates can also operate on multiple qubits at a time. For example, the CNOT gate applies an X gate to the target qubit if and only if the control qubit is in the $|1\rangle$ state. This CX gate transforms the bitstring $|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$ into $CX|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \delta|10\rangle + \gamma|11\rangle$. These multiqubit operators are able to produce entanglement between the qubit states. The measured state of one qubit will influence the measured of qubits it is entangled with. Together, superposition and entanglement are central to the expected power of quantum computation and to solving problems that are intractable on classical computers.

While most quantum compilation focuses on manipulating the one- and two-qubit gates,

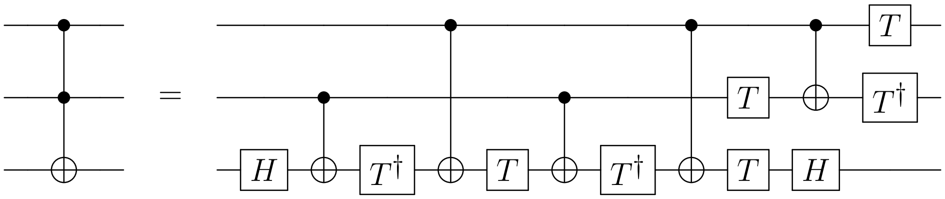


Figure 2.1: The decomposition of the three-qubit Toffoli gate into one- and two-qubit gates.

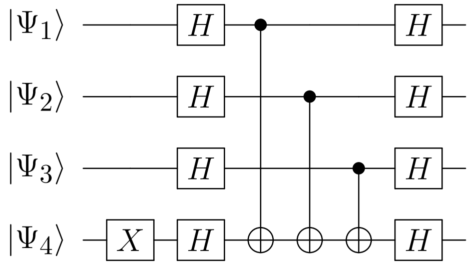


Figure 2.2: A Bernstein-Vazarani example circuit.

an operation can operate across any number of qubits. An example of this is the three-qubit Toffoli gate, which changes the $|0\rangle$ and $|1\rangle$ position of a target qubit if both of the control qubits are in the $|1\rangle$ state. These $N-$qubit operators can be decomposed into a set of universal one- and two-qubit gates that represent the same set of operations. However, this can often insert many extra gates into a circuit as replicating a gate exactly can be an expensive process and is an open problem [82]. An example of a decomposition of the Toffoli gate can be found in Figure 2.1.

Most quantum programs for gate-based quantum computation are expressed in the quantum circuit model. In this model, each qubit is represented by a wire in a diagram, and each gate is presented by a box or circle on a qubit with a progression of operations from left to right. A multi-qubit gate is represented by a box that touches multiple qubits. Generally, when we have a controlled operation, like a CX gate, the qubit that controls the target is represented by a filled in dot. A small Bernstein-Vazarani circuit is found in Figure 2.2.

There are many more aspects to quantum computing, such as error correction, but this foundation is all that is necessary for this thesis. For a complete introduction to quantum computing see [79].

2.2 Background: Quantum Hardware

Quantum architectures are in a time of rapid development and severe constraints. In this era, there are several limitations facing quantum hardware, and different architectural techniques, such as superconducting, ion traps, and neutral atoms have different trade offs.

The first of these constraints is imperfect gates. Each of the listed architectures have error prone gates, and cannot perform operations perfectly. For superconducting architectures one-qubit gates can be performed with 99.9% fidelity, and two-qubits gates with 99.4% fidelity [2]. In some cases, this may even be variable depending on the which qubits are interacting. This is similar for ion traps with 99.99936% and 99.99987% [19]. Neutral Atoms, which are earlier in development, have the lowest with 99.6% and 96.5% [66]. Due to these error prone gates, programs cannot contain too many operations. As more operations are added, the probability of a successful circuit decreases as well as each additional operation could fail. When we begin compiling circuits to hardawre, we must be conscious of this fact and add as few gates as possible, and remove as many redundant gates as we can.

A second constraint is gate durations and qubit coherence time. Current qubits are fragile, and can only maintain a non-ground energy state for a certain period time before it decoheres, and the quantum state collapses. In most cases, this will result in the wrong measurement. Similar to how more gates increases the probability of error, the longer a quantum program runs, the more likely a qubit is to decohere. Where superconducting qubits have relatively high gate success rates, it has the shortest coherence times of 121.1 μs [2], due in part to the difficulty of manufacturing high fidelity superconducting chips [78]. On the other hand neutral atoms and ion traps have much longer coherence times on the order of minutes [34, 109]. Consequently, the speed of gates is much more important for superconducting devices as compared to ion traps and neutral atoms which use single atoms to represent qubits, using lasers and optical tweezers to keep each in position.

Another significant issue facing quantum computers is interacting qubits with one another

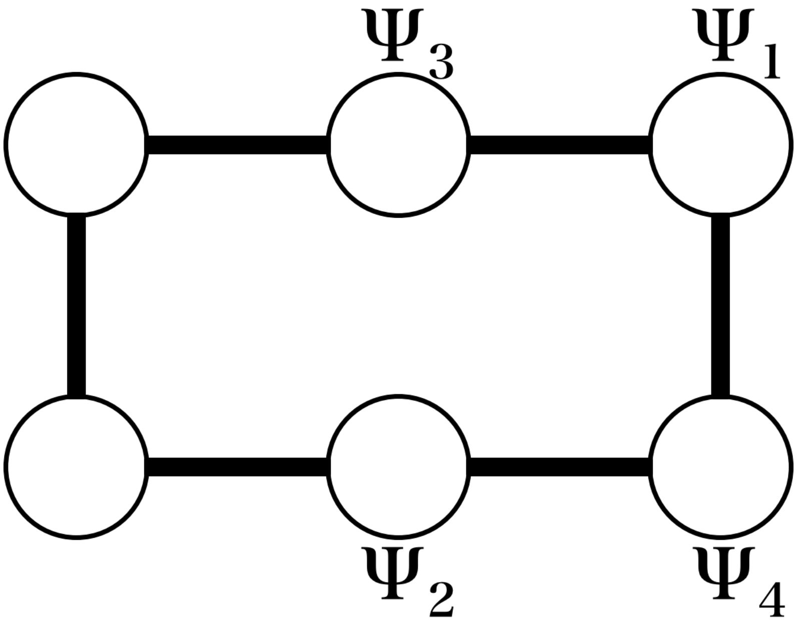


Figure 2.3: An example of an architecture with six qubits, with connections such that it forms a hexagon. A four-qubit Bernstein Vazarani has been mapped onto this architecture.

on a device with limited connectivity. That is, when not all qubits can interact with any other qubit. Currently quantum computers are constructed of up to hundreds of qubits. Even at this scale, it is not possible to interact any one qubit with any other qubit. For example, see Figure 2.3, where we have a hexagon connectivity. However, we can define connections between qubits, and use a special operation called a SWAP gate, that exchanges the quantum data between two physical qubits. This allows us to move qubits across the architecture in a process called *communication*. The process of adapting the circuit in 2.2 to fit this topology can be seen in Figure 2.4 with the initial mapping in 2.3. In a superconducting devices, there is very low connectivity as the resonators between qubits must be defined on a physical chip. On the other hand, ion traps and neutral atoms have longer interaction distances, and somewhat higher connectivity. As circuit sizes grows, so will the need for communication. Adding extra gates incurs error from both the gate success and the coherence of qubits. Ensuring that as little communication is used as possible is important when running actual circuits on quantum hardware.

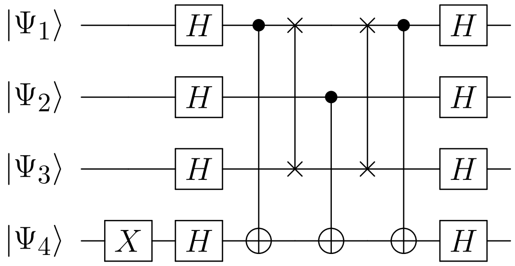


Figure 2.4: A routed Bernstein-Vazarani example circuit.

There are many other issues facing quantum computation such as crosstalk [72] and efficient decompositions of to native gate sets [108]. But, all together, these restrictions demonstrate a need to decrease the use of quantum resources when circuits are run. This includes mainly includes reducing qubit lifetime and the number of gates used.

When we execute quantum circuits, we perform many runs of the same circuit. This is done to build a meaningful distribution of results that can be drawn from. The entire circuit is run, and measured, before being reset. This can be a time intensive process, and can incur significant overhead if the reset process is significant.

2.3 The Quantum Compilation Problem

A compiler for quantum program needs to be able to mold a circuit to fit the constraints of an architecture while reducing the amount of quantum resources required. There have been a number of optimizations developed that take advantage of commutation relationships, gate cancellations and circuit synthesis to reduce gate count and circuit depth. These work well for optimizing the algorithmic level, but do not necessarily work to fit a circuit to an architecture. Instead, this work focuses on changing a circuit to fit the constraints of the architecture, mainly matching connectivity in the most efficient way possible, and choosing the most efficient decompositions of circuits.

The first step is mapping, the program qubits must be assigned to hardware qubits with the goal of minimizing the distance between interacting qubits over the course of the program. As noted, most hardware only supports interactions between a limited set of qubits. Qubits mapped too far from each other must be moved within interaction distance by inserting SWAPs or other communication operations before perfoming the original operation. This communication is often very expensive and every extra gate needed for communication contributes to the overall error rate of the final program. It is common for the mapping and routing steps to occur in tandem as routing changes the mapping of the qubits over the course of the program. Finally, scheduling consists of deciding when to execute which gates and is usually dictated by factors such as run time or crosstalk where we may delay gates to avoid crosstalk effects but possibly increase runtime.

The basic mapping and routing algorithms in this work are based heavily on previous work from [73, 10]. There have been many different explorations into developing efficiently mapping and routing heuristics [112, 104, 113, 27, 50]. Several of these strategies were developed with specific issues in mind. For example, Murali et. al. focused on crosstalk mitigation in [75], and avoiding particularly noisy connections in [73]. The mapping and routing compilation strategies laid out in this section focus on more general strategies that reduce number of SWAPs, which is applicable for any machine, rather than reducing other sources of errors.

2.3.1 Decomposition

An important step in compilation is retargeting the gates used in the circuit to a gate set native to the hardware. Different architectures natively support different sets of gates. As long as the native gate set is universal, it is possible to fully translate any other unitary gate into said native gate set. In many cases, this is sufficiently represented by a single one-qubit gate that can be reparameterized by rotations around each axis and one two-qubit entangler such as the CX gate [79]. This is the strategy used in this work, with some exceptions to use native three-qubit entangling gates. Different native gate sets have varying entangling capabilities, and there have been studies exploring how to best decompose circuits based on the two or greater-qubit gates available within the context of compilation [82, 33].

2.3.2 Mapping for Quantum Architectures

We represent the underlying hardware topology as a graph, where nodes are hardware qubits and edges are between qubits which can interact on the device. In this model, each edge is weighted the same way. That is, a gate performed along this edge has the same probability of success.

For most quantum programs, the entire control flow is known at compile time making optimal solutions for mapping and routing possible but exponentially difficult to find. Lookahead bases mapping and routing on the sum of weighted future interactions with operations further into the future weighted less. The entire circuit is mapped and routed in steps. At each step, we consider the *weighted interaction graph* where nodes are *program qubits* and edges between nodes are weighted by the lookahead function:

$$w(u,v) = \sum_{t \in o(u,v)} f(t - t_c)$$

where $w(u,v)$ is the weight between program qubits u and v, t is a layer of the program, and t_c is the current layer, i.e. the frontier of the program DAG. When considering a multiqubit gate we add this weighting function between all pairs of qubits in the gate. The function $f(t)$ is an arbitrary, usually monotonically decreasing, function. For example, $f(t) = 1/(1+t)$ or $f(t) = e^{-t}$ are usually used as lookahead function.

For the initial mapping, we begin by placing the qubits with the greatest interaction weight in the weighted interaction graph. We place these qubits adjacent in the center of the device based on the hardware graph. For every subsequent logical qubit in this graph we consider all possible assignments to hardware qubits adjacent to the hardware locations of already mapped logical qubits and choose the best based on a score:

$$s(u, h) = \sum_{\text{mapped } v} d(h, \varphi(v)) \times w(u, v)$$

where h is the potential hardware location, and φ is the mapping from program qubits to hardware qubits. d is the physical distance between qubits on the hardware graph. The goal is to place qubits which interact frequently close to each other in order to avoid extra SWAPs during routing. We choose the hardware location h which minimizes this score. We place qubits ordered by their weight to those previously mapped, greatest first. This process continues until all of the qubits are mapped.

2.3.3 Routing for Quantum Architectures

For routing and scheduling, we proceed through the circuit layer by layer, considering operations in the frontier as potential gates to execute. We can only execute gates where the qubits are adjacent together on the hardware interaction graph, we need to have a strategy that can move qubits closer together without ruining the locality of the current mapping of the qubits and future operation.

We use a metric called *disruption*. This balances finding the shortest path between two qubits, without disrupting future operations. This leads to the following scoring function for each neighboring physical qubit to the qubits involved in the operations that is strictly closer to the other qubit in the operation:

$$\begin{aligned} D(u, j) = \sum_{v} & [d(\varphi(u), \varphi(v)) - d(j, \varphi(v))] \times w(u, v) + \\ & [d(j, \varphi(v)) - d(\varphi(u), \varphi(v))] \times w(\varphi^{-1}(j), v) \end{aligned}$$

In this case, j is the candidate swap location u is one logical qubit in the operation, and v is the other. We perform this scoring for all qubits in the operation, and pick the SWAP that maximizes this function. In this function, moving further away from future interactions or displacing the qubit in position j by moving it far from its future interactions is penalized.

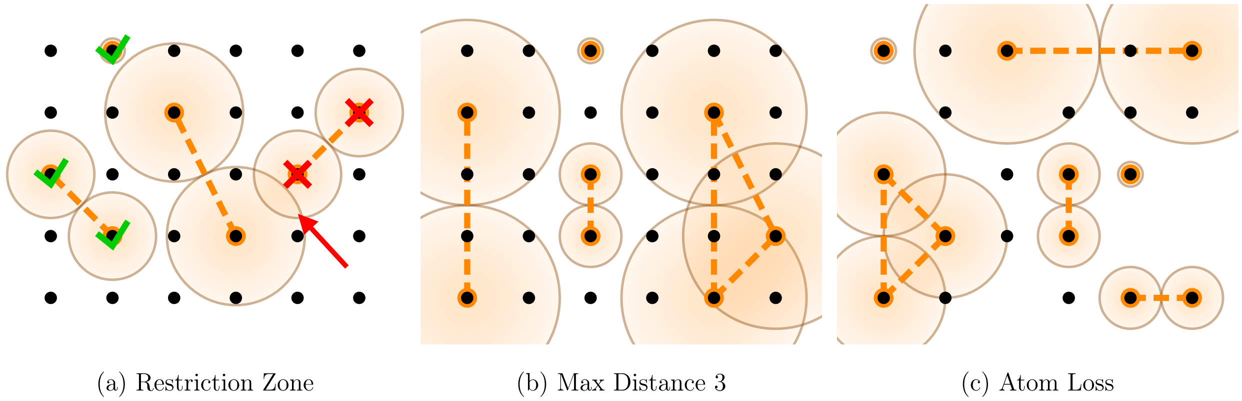
CHAPTER 3

NEUTRAL ATOM ARCHITECTURES: ADAPTING TO FLEXIBLE OPERATIONS AND FRAGILE QUBITS

In the past several years, many leading gate-based quantum computing technologies such as trapped ions and superconducting qubits, have managed to build small-scale systems containing on the order of tens of qubits [114, 86, 55, 101]. However, each of these systems have unique scalability challenges [60, 21]. For example, IBM devices have continued to grow in size while error rates have remained high, larger than what is needed for quantum error correction [44]. Trapped ion machines, despite many promising results, have fundamental challenges in controllability [74]. It is unclear whether any of these platforms in present form will be capable of executing large-scale quantum computation needed for algorithms with quantum speedup like Grover's Search [96] or Shor's Prime Factorization [45].

An alternative approach is to consider emerging quantum technologies and new architectures. In this chapter, we explore hardware composed of arrays of individually-trapped, ultra-cold neutral atoms which have shown great promise and have unique properties which make them appealing from a software standpoint [92]. These properties include: potential for high-fidelity quantum gates, indistinguishable qubits that enable scaling to many qubits, long-range qubit interactions that approximate global (or generally high) connectivity, and the potential to perform multiqubit (≥ 3 operands) operations without expensive decompositions to native gates [66].

Neutral-atom (NA) architectures also face unique challenges. Long-range interactions induce zones of restriction around the operating qubits which prevent simultaneous operations on qubits in these zones. Most importantly, the atoms in a neutral atom device can be lost via random processes during and between computation. In the worst case, the compiled program no longer fits on the now sparser grid of qubits, requiring a reload of the entire array every cycle. This is a costly operation to repeat for thousands of trials. Coping with this loss


(a) Restriction Zone

(b) Max Distance 3

(c) Atom Loss

Figure 3.1: Examples of interactions on a neutral atom device. (a) Interactions of various distances are permitted up to a maximum. Gates can occur in parallel if their zones do not intersect. The interaction marked with green checks can occur in parallel with the middle interaction. (b) The maximum interaction distance specifies which physical qubits can interact. Compiler strategies suited for this variable distance are needed for neutral atom architectures. (c) Neutral atom systems are prone to sporadic atom loss. Efficient adaptation to this loss reduces computation overhead.

in a time-efficient manner is important to minimizing the run time of input programs while not dramatically increasing either gate count or depth, both of which will reduce program success rate. In Figure 3.1 we show a small piece of a neutral atom system with many gates of various sizes and distances being executed in parallel. Restriction zones are highlighted and importantly no pair of gates have intersecting restriction zones. When atoms are lost during computation, rather than a uniform grid we have a much sparser graph and qubits will be further apart on average. A key to the success of a neutral atom system is resilience to loss of atoms, avoiding expensive reloads.

In this chapter, we first explore the trade-offs in neutral atom architectures to assess both current viability and near future prospects. We extend current compilation methods, particularly mapping and routing, to directly account for the unique NA constraints like long-range interactions, areas of restriction, and native implementation of multiqubit gates. We then propose several coping strategies at the hardware and software level to adapt to loss of atoms during program execution and we evaluate the tradeoff of execution time and resilience to atom loss versus program success rate. We then extend the more basic atom loss recovery strategies to make more full use of an architecture and further reducing the execution time while more effectively maintaining program success rate.

3.1 Background: Neutral Atoms

We first briefly introduce some background on the underlying neutral atom technology. A nice introduction can be found in [47]. Rather than being fabricated on a physical chip, atoms in a NA system are trapped via reconfigurable, optical tweezer arrays. These atoms can be arranged in one, two, or even three dimensions [34, 12, 58, 13]. In this sections we consider 2D configurations of these atoms, but arbitrary arrangements of atoms are possible. Historically, one of the major difficulties with scalable neutral atom systems was the probabilistic nature of atom trapping, but this challenge has since been overcome and defect-free arrays of more than 100 atoms [80] have been demonstrated, and more recently, dual species arrays of more than 512 atoms have been demonstrated as well [98]. However, while neutral atom devices are able to trap and interact with several hundreds of qubits, the error rates of demonstrated gates is much lower than that of superconducting architectures. In particular, one-qubit gates have a success rate of 99%, as compared to 99.9%, and two-qubit gates have a success rate of 96.5%, as compared to 99%.

The single atom qubit states can be manipulated using Raman transitions which implement single qubit gates. In order to execute gates between qubits, atoms are optically coupled to highly excited Rydberg states leading to a strong dipole-dipole interaction between the atoms [53]. These Rydberg interactions enable multiple atoms at once to interact strongly and are used to realize multiqubit gates with 87.5% fidelity [66], something that is not very easily achieved in other devices.

This method of physical quantum computation lends itself to unique capabilities beyond arbitrary configurations. The main benefit is that due to the long range of these interactions, gates between qubits which are not directly adjacent in the atom array are feasible. However, these longer distance interactions induce a zone of restriction as a function of the distance. As the distance between qubits in an interaction increases, so does the area of restrictions around both of the qubits in the interaction. This zone of restriction constrains which qubits can be acted on at the same time. Two gates can only occur in parallel if their restriction zones do not overlap.

The strategy has its own unique downsides as well. The weaker optical tweezers can “lose hold” on the atoms, which are subsequently lost from the array. When this occurs, that particular run of the circuit is invalid, and the entire array must be reset via a detection and reloading process. More details of this process will be discuss in Section 3.4 Since the loading of qubits into the array is relatively slow, on the order of one second, compared to program execution which usually takes milliseconds, this is a barrier to scalability for neutral atom devices.

3.2 Compiling for Neutral Atom Architectures

3.2.1 Adapting Mapping and Routing

The general compilation strategies laid out in 2.3 could easily be used on a neutral atom device. If we only used adjacent atom connectivity on the hardware graph, we would likely not run into the issue of conflicting parallel operations. However, this would not be taking full advantage of the long-distance capabilities of a neutral atom device. This section specifically adjust the general mapping and routing algorithms laid out previously.

In a neutral atom device, we are not constricted by physical connections on a fabricated chip as we are on superconducting devices. Instead, we are constrained by a *maximum interaction distance*, or d_{max}, which is the furthest *physical distance*, $d_p(u, v)$ that two qubits u, v can be from each other and still interact with one another. Now, our interaction graph of hardware qubits contains an edge between nodes u, v if $d(u, v) \leq d_{max}$. This breaks

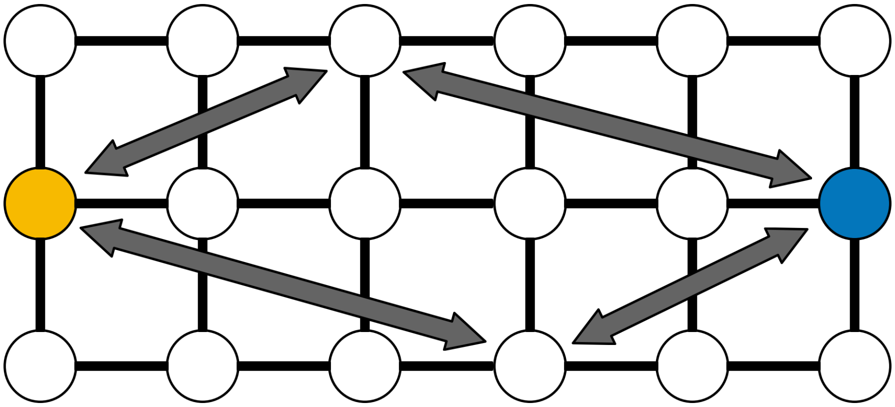


Figure 3.2: An example of how SWAP paths of the same length can result in different interaction distances from other qubits on the architecture and how not all SWAPs will have the same effect on locality as on hardware where the adjacency graph is the interaction graph.

the original assumptions of our interaction graph introduced previously, where a connection implied physical adjacency and intractability. Here, the physical distance could be much longer, affecting the area of restriction with parallel operations, even though the interaction cost is the same. One choice of SWAP can be significantly different to another due to the increased range of operations, as seen in Figure 3.2 where the qubits could be moved further away from neighboring qubits, even if the SWAP paths are the same length.

For mapping, we can simply use the new interaction-graph in place of the original adjacency based interaction graph. The logic in this case remains the same. However, it is not necessarily the case for the router. Here, we must to reconcile the difference of physical distance versus the interaction distance. We still explore the neighbors of the current physical qubit in the interaction graph that move the qubits strictly closer, but we score each candidate SWAP differently. Due to the increased number of paths between qubits, in place of the original distance function, we use a normalized hardware difference: $d_{norm}(p, q) = \lfloor d_h(p, q)/d_{max} \rfloor$ where d_h. This let's estimate the number of SWAPs that will be required to interact two qubits with one another. We also add an additional variable to our maximization function by adding the distance between the current hardware qubit and the candidate SWAP qubit and the disruption function: $S(q,p) = d_{norm}(q,p) + D(q,p)$. This weights the distance moved towards the target, as well as considering the improvement, or detriment, to locality that the change causes.

This consideration does not cover the issue of serialization due to the restriction zones induced by longer interaction distances. Due to the non-local effects of different interactions, this needs to be considered differently than the general algorithm. In general, the compiler attempts to use an "As Soon As Possible" scheduling strategy. This strategy first ensures that any operations that were deferred previously are scheduled first. We then examine the frontier of potentially executable operations and schedule as many in parallel as possible. That is, as many operations as do not overlap with other areas of restriction and the qubits are within range of one another. We then perform a round of SWAP operations that do not conflict with one another that move the qubits closer.

As neutral atom technology is also able to perform native three-qubit gates, we can make a simple change that generalizes these algorithms adaptable to many qubits. When determining which candidate SWAP qubits to check, we only explore those than reduces the sum of the distances to the other qubits. This ensures that we consistently move the locality of the qubits to one another.

3.2.2 Experimental Evaluation

We now explore how these compilations changes affect the resulting circuits, and how well we are able to balance the benefits of Neutral Atom architectures.

Benchmarks

For this work, we have chosen a set of quantum programs which are parametrized, the input size can be specified, to allow us to study how the advantages and disadvantages of a NA system change as the program size increases. Specifically, we study Bernstein-Vazirani [14], a common quantum benchmark, with the all 1s oracle to maximize gates, Cuccaro Adder [28], a ripple carry adder with no parallelism, the CNU gate [11], a logarithmic depth and highly parallel decomposition of a very common subcircuit, QFT Adder [91], a circuit with two QFT components and a highly parallel addition component, and QAOA for MAX-CUT [35], a promising near-term algorithm, on random graphs with a fixed edge density of 0.1.

Experimental Setup

For most experiments, we compile our benchmarks, with sizes up to 100, on a 10×10 NA device. We have a fixed radius of restriction but vary max interaction distance from 1 (emulating superconducting systems) up to the maximum needed for global connectivity (here $hypot(9,9) \approx 13$). In relevant benchmarks we compile with decomposed multiqubit gates and without. All experiments were run using on a machine using Python 3.7 [107], Intel(R) Xeon(R) Silver 4110 2.10GHz, 132 GB of RAM, on Ubuntu 16.04 LTS. All plot error bars show ± 1 standard deviation.

3.2.3 Evaluation Results

In this section, we explore promising architectural advantages provided by the neutral atom technology. We examine long range interactions where atoms distant on the device can interact similar to a device with high connectivity. However, the cost of this longer range interaction is higher serialization due to the proportional increase in restricted area. Second, we explore the native execution of multiqubit gates on the NA platform. Since netural atom technology is still in its early stages, it can be unfair to compare expected program success rates from current gate error rates and coherence times. We analyze common metrics, gate count and depth, which are good predictors of a program's success rate if executed.

Trapped ion and superconducting architectures currently support qubit interaction only between *adjacent* qubits. In SC systems this usually corresponds to a 2D grid or some other sparse connectivity, where each qubit is able to interact with a small number of qubits. One of the important promises of trapped ions is all-to-all connectivity where each qubit can interact freely with any other qubit in the same trap. Each trap however, is currently limited by the number of ions it can support and expensive interactions across different traps.

In NA architectures, the connectivity lies somewhere between these two extremes. The atoms, while often arranged in a 2D grid, have all-to-all connectivity beyond immediate neighbors, i.e. within a fixed radius. This radius is dictated by the capabilities of the hardware and can theoretically reach as large as the device. However, current demonstrations have been more limited, for example up to distance 4. In this work, our experiments analyze the full sweep of interaction distances to understand the importance of long range interactions to optimizing program success rate predictors.

Long range interactions in NA are not free, we define an area of restriction imposed by interacting qubits at a distance d from each other, $f(d)$. Specifically, given this interaction distance between qubits of the set Q all other qubits $q \notin Q$ with distance less than $f(d)$ to *any* of the interacting qubits cannot be operated on in parallel. Furthermore, suppose we have two operations to be performed in parallel. These two operations can only execute in parallel if their areas of restriction do not overlap. For experiments in this work we explore the function $f(d) = d/2$. Intuitively, as this function becomes more restrictive, i.e. the areas surrounding the interacting qubits get larger, fewer total operations will be executable in parallel, affecting the total execution time of the program.

Long range interactions are important for reducing the total number of gates required for execution on devices with relatively limited connectivity. Limited connectivity requires compilers to add in many extra SWAP operations. The lower the connectivity, the greater the average distance between qubits on the device therefore more SWAPs are required to execute multiqubit gates between arbitrary sets of qubits. In Figure 3.3, we explore the gate counts of compiled programs for various sizes over a range of maximum interaction distances

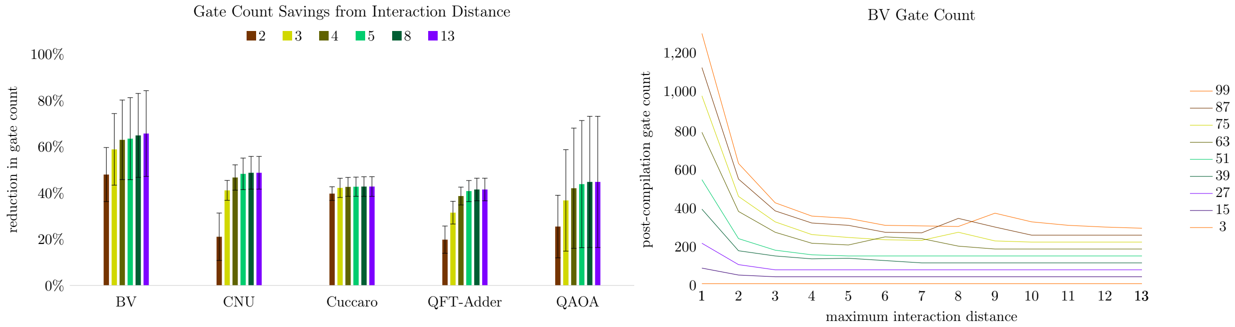


Figure 3.3: Post compilation gate count across benchmarks. On the left are percent savings over the distance 1 baseline averaged over program sizes up to 100 qubits. Each color is a max interaction distance. Noticeably, there is less additional improvement as the MID increases, indicating most benefit is gained for smaller distances. On the right is a sample benchmark (holds in general) with many program sizes compiled for the whole range of MIDs. As the program size increases, larger MID show benefit before flattening off.

up to the largest possible distance supported on the device. In each of these experiments, all programs are compiled to 1 and 2 qubit gates only. Intuitively, we might assume having a larger maximum interaction distance will necessarily be better than a smaller one since it emulates global connectivity therefore not requiring any additional SWAP operations. In general, we find the most benefit in the first few improvements in max interaction distance with more relative gain for larger programs. The reduction in gate count is due solely to a reduction in total SWAPs.

Importantly, the benefit obtained from increasing max interaction distance tapers off with vanishing benefit. The rightmost points in these figures correspond to an interaction distance the full width of the device, providing all-to-all connectivity. At this distance no additional SWAP gates are required, so this is the minimum possible number of gates to execute the input program. This distance is not required to obtain the minimum (or near the minimum). In fact, a smaller interaction distance is sufficient. This is promising in cases where large interaction distances cannot be obtained and hardware engineers can focus on building higher fidelity short to mid range interactions. For larger devices, the curves will be similar, however, requiring increasingly larger interaction distances to obtain the minimum.

The shape of the curve will be more elongated, related directly to the average distance between qubits.

A similar trend exists for circuit depth as seen in Figure 3.4. As interaction distance increases the depth tends to decrease, with the most benefit found in larger programs. Again, the rate of benefit declines quickly. We expected that as the interaction distance increased, the depth would decrease initially, then increase again due to restriction zones proportional to the interaction distance. As the maximum allowed distance increases, the average size of these zones will increase, limiting parallelism. However, there are several important factors diminishing the presence of this effect. First, SWAPs are a dominant cost in *both gate count and depth*, often occurring on the critical path. Therefore, reducing the need for communication typically corresponds to a decrease in depth. Second, many quantum programs are not especially parallel and often do not contain many other gates which need to be executed at the same time limiting the potential for conflicting restriction zones. In our set of benchmarks, the circuits with high initial parallelism like CNU and QFT-Adder (a long stretch of parallel gates in the middle) do show increases in depth with increased interaction size but are not especially dramatic. In cases where gate error is dominant over coherence times, the reduction in gate count far outweighs the induced cost of greater depth or run time.

This isn't to say there is no cost from the presence of a restriction zone. In Figure 3.5 we analyze the relative cost of the restriction zones. In this set of experiments the program is compiled with the same maximum interaction distance. In the ideal case it is compiled with no restriction zones, resembling other architectures which permit simultaneous interactions on any set of mutually disjoint pairs. These two circuits have the same number of gates, including SWAPs. When no parallelism is lost, either from the original circuit or from parallelized communication, these lines are close. A large gap indicates the increased interaction distance causes serialization of gates. One additional side effect, which we do not model here due to complexity of simulation is the effect of crosstalk. By limiting which qubits

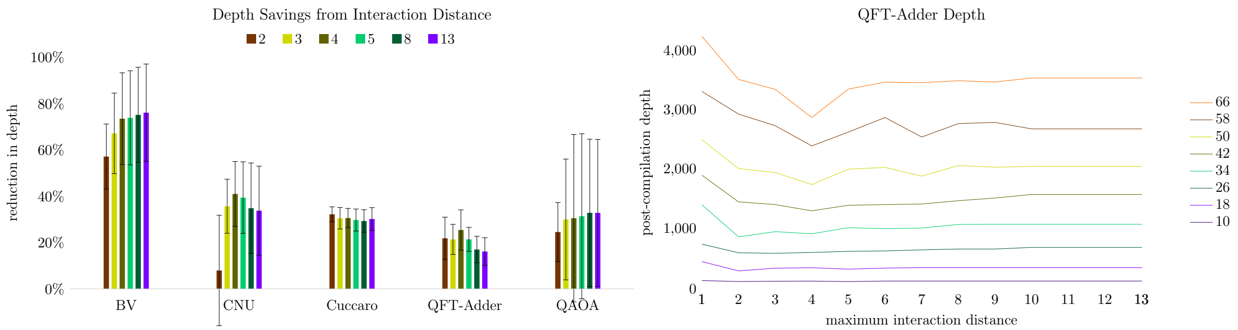


Figure 3.4: Post Compilation depth across all benchmarks. On the left, the reduction in depth over the distance 1 baseline. Each bar is the average over all benchmark sizes. On the right we see a similar drop off in post-compilation depth for the QFT-Adder. We've chosen this specific benchmark to highlight the effect of restriction zones. Here we show a subset of all sizes run. Depth initially drops but for larger interaction distances some of this benefit is lost. We expect this to be more dramatic for even larger programs.

can interact in parallel we can effectively minimize the effects of crosstalk implicitly. This can be made more explicit by artificially extending the restriction zone to reduce crosstalk error by increasing serialization.

Long range interactions are not the only unique property of NA architectures. One of the important promises of NA hardware is the ability to interact multiple qubits and execute complex instructions natively. For example, gates like the three qubit Toffoli could be executed in a single step. This is important for several reasons.

First, it doesn't require expensive decompositions to one- and two-qubit gates. Gates like the generalized Toffoli have expensive decompositions, transforming compact complex instructions into long strings of gates before SWAPs or other communication is added. The base, 3 qubit Toffoli itself requires 6 two qubit gates and interactions between every pair of qubits. If all these Toffoli gates could be executed natively without decomposition this saves up to 6x in gate count alone. Toffoli gates are fairly common in quantum algorithms extended from classical algorithms like arithmetic since they simulate logical ANDs and ORs. If even larger gates are supported, this improvement will be even larger.

Second, efficient decomposition of multiqubit gates often requires large numbers of extra

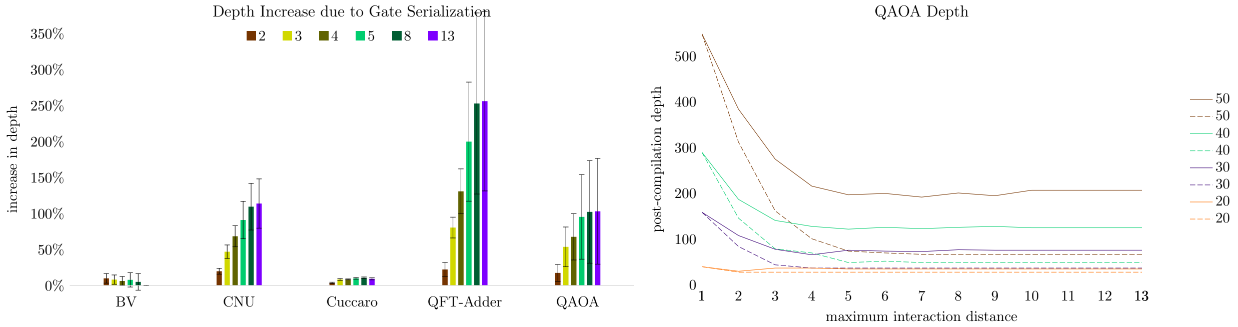


Figure 3.5: The induced restriction zone from interaction distance increases serialization. In the prior results this is hard to discern because compared to low interaction distance the amount of gate savings translates to depth reduction. Here we compare benchmarks compiled with our restriction zone and compare to a program with no restriction zone, to mimic an ideal, highly parallel execution. The existence of a restriction zone most effect on programs which are parallel to begin with. On the right we directly compare this effect on the QAOA benchmark; solid line is compiled with realistic restriction zone and dashed is ideal. The separation between the corresponding lines signifies the effect of the restriction zone.

ancilla qubits. For example, in our CNU benchmark we use the logarithmic depth decomposition which requires $O(n)$ ancilla, where n is the number of controls. When complex gates are executable natively, additional qubits are typically not needed, reducing space requirements for efficient implementation of gates.

In the NA architecture, execution of these gates does come with some constraints. For example, to execute a 3 qubit Toffoli gate, each interacting qubit needs to be less than the maximum interacting distance to every other interacting qubit. Therefore, with only an interaction distance of 1 it is impossible to execute these gates and instead they must be decomposed and for larger gates more qubits will need to be brought into proximity. While not explored explicitly in this work, larger control gates will require increasingly larger interaction distances. In general, the more qubits interacting, the larger the restriction zone, increasing serialization if the qubits are too spread out.

Our set of benchmarks contains two circuit written explicitly in terms of Toffoli gates: CNU and Cuccaro. In Figure 3.6 we analyze the effect of native implementation of these gates rather than decomposition. The benefit is substantial in both cases requiring many fewer

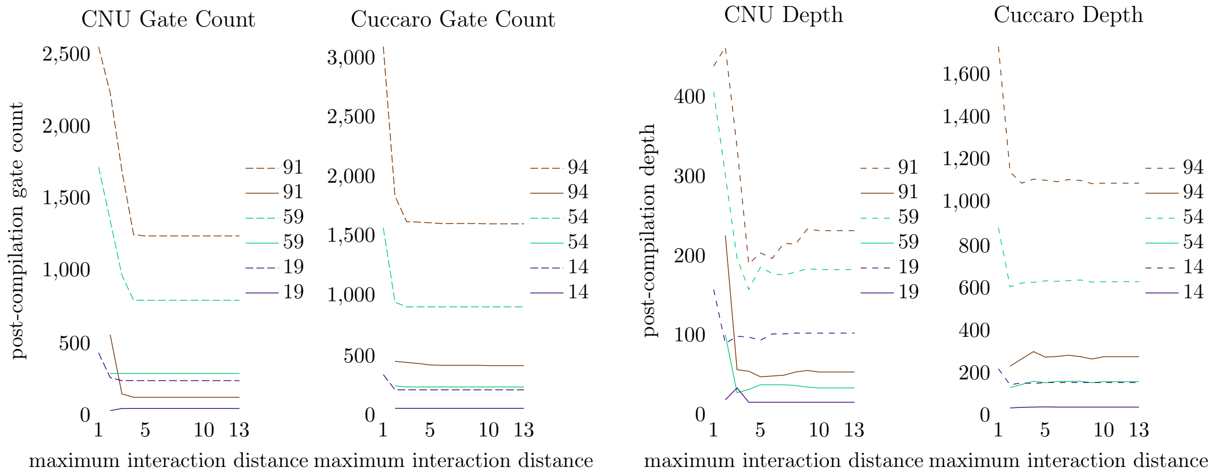


Figure 3.6: Compiling to programs directly to three qubit gates reduces both gate count and depth. Here we highlight a serial and parallel application written to three qubit gates. Here dashed lines are compiled to two qubit gates decomposing all Toffoli gates before mapping and routing. Solid lines compile with native Toffoli gates. With native implementation of three qubit gates we obtain huge reductions in both depth and gate count for both benchmarks.

gates across all maximum interaction distances. While these gates have been demonstrated, their fidelity is much lower than the demonstrated fidelity of two qubit gates. However, a simple estimation given by the product of the gate errors in the decomposition shows the fidelity of the Toffoli gate is greater than that of the decomposition. We give a more precise analysis of this effect in the next section.

Both long range interactions and native implementation of multiqubit gates prove to be very advantageous, though the benefit is tempered by a distance-dependent area of restriction which serializes communication and computation. The importance of these effects is input dependent. Programs written without multiqubit gates cannot take advantage of native implementation. Programs which are inherently serial are less affected by large restriction zones at long interaction distances. One of the most important observations is that excessively long interaction distances are not required and most benefit is obtained in the first few increases. However, as the input program size increases for larger hardware, we expect more benefit to be gained from long interaction distances. This trend is evident here where small programs have almost no benefit from increasing distance 2 to 3 but large programs nearing the device size see much more.

3.3 Error on Neutral Atom Devices

In the previous section we explored the effect on several key circuit parameters like gate count, depth, and parallelism. These metrics are often good indicators for the success rate of programs on near and intermediate term quantum devices where gate error is relatively high and coherence times relatively low. In the case where gate errors and coherence times are uniform across the device and comparable between technologies, these parameters are sufficient for determining advantage of one architecture over another. However, current quantum technology is still in development with some more mature than others. For example, superconducting systems and trapped ion devices have a several year head start over recently emerging neutral atoms.

Consequently, physical properties like gate errors and coherence times are lagging a few years behind their counterparts. It is critical however to evaluate new technologies early and often to determine practical advantages at the systems level and to understand if the unique properties offered by some new technology are able to catapult the co-designed architecture ahead of its competitors. In this section, we evaluate the predicted success rate of programs compiled to a uniform piece of hardware with currently demonstrated error rates and coherence times. It is important to note the gate fidelities and T_1 times used as a starting point are often measured from small systems as no large scale NA architecture has been engineered to date. The average error, or T_1, across the hardware may have variance, as demonstrated in other publicly available technologies, though neutral atoms promises uniformity and indistinguishability in their qubits, similar to trapped ions.

Simulating a general quantum system incurs exponential cost with the size of the system. It is impractical to model all sources of errors during computation and simplified models are typically used to predict program success rate. Here we compute the probability a program will succeed to be the probability that no gate errors happen times the probability that no decoherence errors occur. If $p_{gate,i}$ is the probability an i-qubit gate succeeds and n_i is the number of i-qubit gates then the probability no gate error occurs is given by $\prod_i p_{gate,i}^{n_i}$. Here we consider circuits with up to $i = 3$. For neutral atoms, we consider two different coherence times for the ground state and excited state i.e. $T_{1,g}, T_{1,e}$ and $T_{2,g}, T_{2,e}$ where the ground state coherence times are often much longer than excited state coherence times. Qubits exist in the excited state when they are participating in multiqubit interactions only. The probability coherence errors occur is given as $e^{-\Delta_g/T_{1,g} - \Delta_g/T_{2,g} - \Delta_e/T_{1,e} - \Delta_e/T_{2,e}}$ where Δ_g, Δ_e are the durations spent in the ground and excited states, respectively. Often, gate fidelities already include the effects of T_1 and T_2, i.e. $p_{gate,i}$ includes coherence error. Therefore, we will consider the probability of no coherence error as $e^{-\Delta_g/T_{1,g} - \Delta_g/T_{2,g}}$ only. These simplifications serve as an upper bound approximation on the success rate of a program.

In this section we compare against superconducting systems, specifically, using error values available via IBM for their Rome device, accessed on 11/19/2020. While we directly compare using the same simulation techniques we want to emphasize the purpose of these figures is to indicate the value gained from decreasing gate count and reducing depth relative to other available technology and to suggest that these improvements help overcome current gate error in neutral atom technology. These are not meant to suggest neutral atoms in their current stages are superior to superconducting qubits.

Our simulation results are across a large sweep of error rates from an order of magnitude worse to many orders of magnitude better, where error rates are expected to progress to in order to make error correction feasible. The point is to evaluate different technologies with comparable error rates, how much is saved by architectural differences rather than the current status of hardware error rates, especially when neutral atoms are years behind in development.

In Figure 3.7 we analyze the potential of NA architectures on three representative benchmarks. Here we sweep across various physical error rates and extract the predicted error rate with those parameters; lower is better. In both CNU and Cuccaro we permit 3 qubit gates while the others contain only one- and two- qubit gates. The superconducting curves correspond to similar simulations using error rates and coherence times provided by IBM. At lower physical error rates we expect all architectures to perform well, at virtually no program error rate. At this limit, the hardware is below the threshold for error correction. On the other hand, in the limit of high physical error rates, we expect no program to succeed with any likelihood and to produce random results. For near- and intermediate-term quantum computation, the regions between the limits are most important and the divergence from this all-noise outcome determines how quickly a device becomes viable for practical computation. For comparable error rates between superconducting and NA architectures, we see great advantage obtained via long range interactions and native multiqubit gates, diverging more quickly than the limited connectivity SC devices.

Alternatively, we might ask what physical error rates are needed to run programs of a given size with probability of success above some threshold. In Figure 3.8, we consider this question for a threshold success rate of 2/3. Here we sweep across physical error rates and compute the largest possible program of each benchmark we can successfully execute. There are two interpretations. First, for a fixed physical error rate we can determine what size program is likely to be successfully executed. Alternatively, suppose we have a program of a given size we want to execute, we can then decide what physical error rate do we need to run that program successfully. For a fixed error rate, we find we can execute a larger program or, equivalently, require worse physical error rates than a superconducting system to run a desired program.

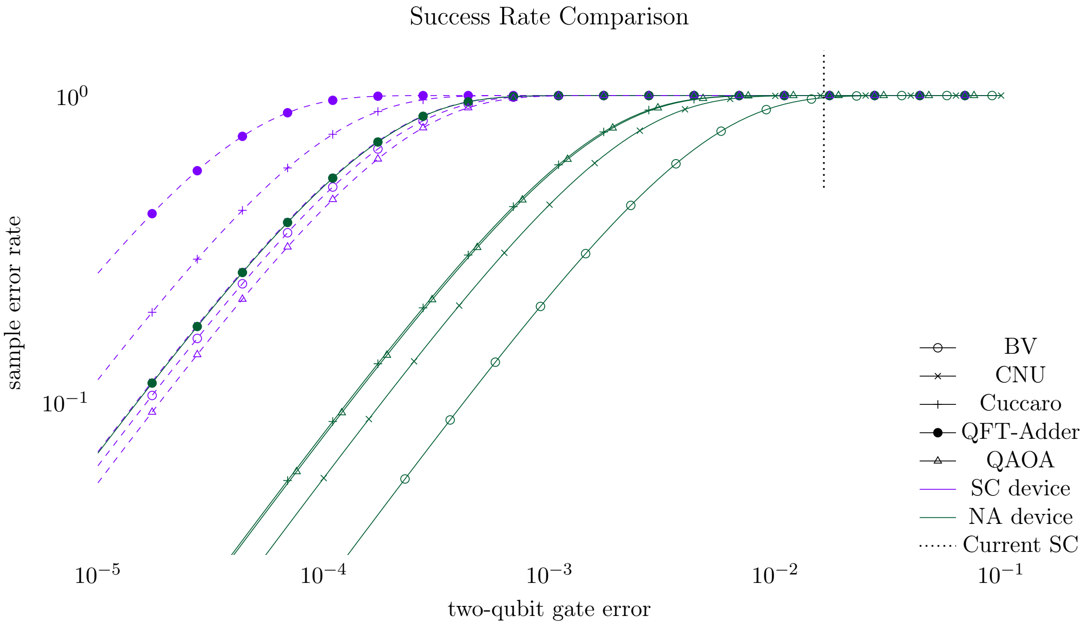


Figure 3.7: Program success rate as a function of two-qubit error rate. Because current NA error rates are lagging behind competitive technologies we scan over a range of two-qubit error rates for each of the benchmarks all on 50 qubit programs (49 for CNU) with max interaction distance of 3. Examining pairs of solid and dashed lines we can compare NA to SC. In the limit of very low two qubit error rate, systems can support error correction. Both SC and NA systems scale at roughly the same rate (slope of the line) but the NA system diverges from the completely random outcome at higher error, allowing us to run programs on the hardware much sooner. The further to the lower right a point is, the higher fidelity circuit it can perform with worse error rate.

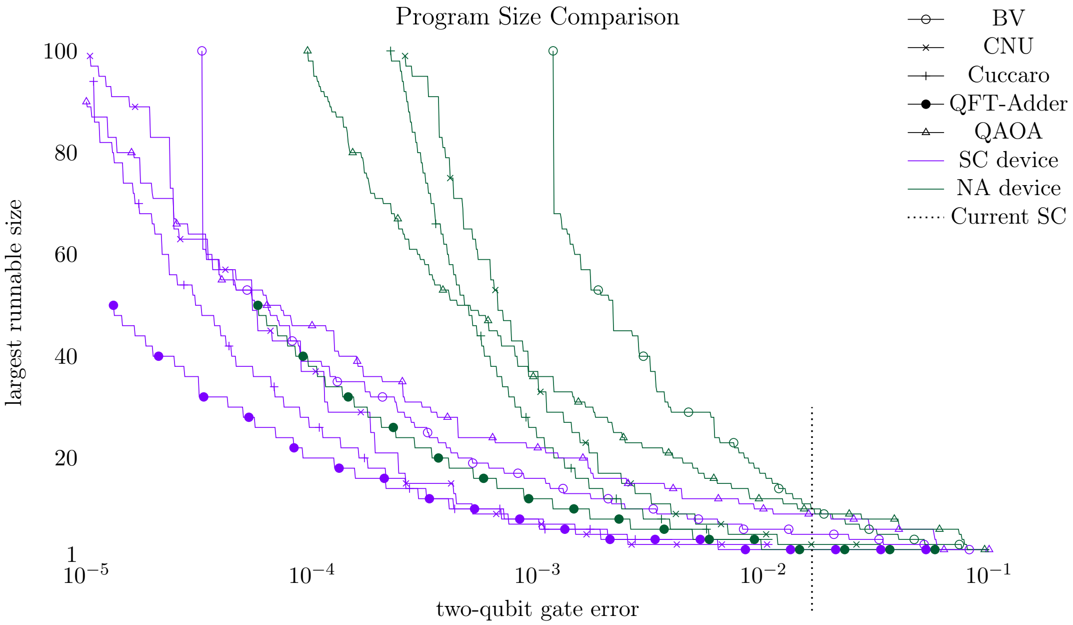


Figure 3.8: Another way to examine the data of Figure 3.7 is to ask, given a desired program success rate, what the required two qubit error rate is. Here we sweep again over two qubit error rates and record the maximum program size to run with success probability greater than 2/3. Again, examining pairs of solid and dashed lines we can compare NA to SC. With the reduced gate counts and depth we expect to be able to run larger programs sooner.

3.4 Adapting to Atom Loss

So far, we've focused primarily on properties of a neutral atom system that are usually advantageous and have analyzed that while there are tradeoffs, the gates and depth saved drastically outweighs any cost. However, neutral atom systems are not without limitation. The atoms in a NA system are trapped using optical dipole traps such as optical tweezers. While this technique offers great flexibility in array geometries and the prospect of scaling to large atom counts, the trapping potential per atom is weak when compared to trapped ion architectures. Consequently, neutral atoms can be lost more easily during or between computations forming a sparser grid. Fortunately, this loss can be detected via fluorescence imaging and remedied with various software and hardware approaches with different overhead costs analyzed here.

Atom loss has critical implications on program execution. For near-term computation, we run programs hundreds or thousands of times to obtain a distribution of answers. Consider running a program, after each trial we evaluate whether atoms have been lost. If an atom used by the computation has been lost, then we have an incomplete answer. Upon a loss, we have no way of knowing if it occurred in computation and must disregard the run from our distribution and perform another shot. Furthermore, a program compiled for the original grid of qubits may no longer be executable. The program must either be recompiled for the now sparser grid of qubits, the array of atoms can be reloaded, or the compiled circuit can be adapted to the atom loss. The first two solutions are costly in terms of overhead time. The third may provide a faster alternative, and provide opportunity to perform more executions in the same time while maintaining a valid program.

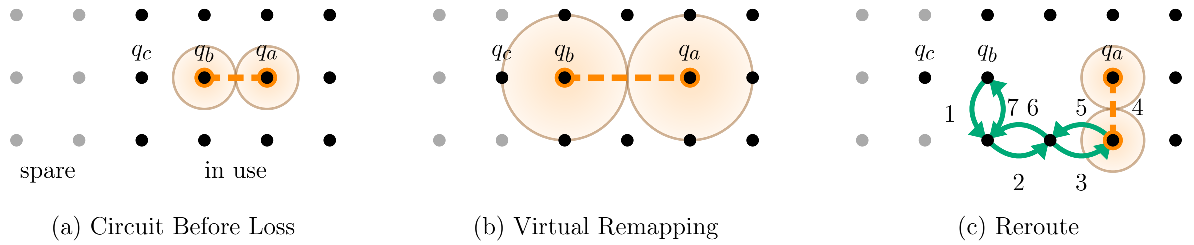
We model atom loss from two processes. The first is based on vacuum limited lifetime where there is a finite chance a background atom collides with the qubit atom held in the optical tweezers displacing the qubit. We approximate this occurs with probability 0.0068 over the course of a program and is uniform across all qubits [26]. Loss during readout is much more likely. In some systems readout occurs by ejecting atoms which are not in a given state, resulting in about 50% atom loss every cycle [38]. This model is extremely destructive and coping strategies are only effective if the program is much smaller than the total size of the hardware. Alternative, potentially lossless techniques, have been proposed for measurement, but are not perfect with loss approximately 2% [64] uniformly across all measured atoms. Detecting atom loss is done via fluorescence which takes on the order of 6ms.

We lay out several coping mechanisms for atom loss and examine their effectiveness in terms of overheads such as the time to perform array loads, qubit fluorescence, and potential recompilation. In each of these experiments we assume the input program is smaller than the total number of hardware qubits in the *original* grid, otherwise any loss of an atom *requires* a reload. These additional unused qubits after initial compilation are considered *spares*, borrowing from classical work on DRAM sparing [77]. Currently, no program that executes with reasonably high success will use the entire grid and these spares will come at no cost. Potentially, as many atom losses can be sustained as number of spares. However, an array reload is always possible there is an unlucky set of holes or no more spares. Below we detail various strategies. These strategies work to adapt the resulting circuit from mapping and routing, using the features neutral atom architectures to adjust the normal runtime strategy of a quantum circuit.

3.4.1 Recovery Strategies

We have several different strategies we can pursue to recover from a incurred atom loss. The first set of strategies are all performed on the originally compiled circuit, it does not try to influence the original compilation.

- *Always Reload.* Every time an atom loss is detected for a qubit used by the compiled program we reload the entire array. This naive strategy is efficient when array reloads



(a) Circuit Before Loss

(b) Virtual Remapping

(c) Reroute

Figure 3.9: Examples of two different atom loss coping strategies. (a) shows the initial configuration of three qubits, with the spare qubits in a light grey, and in use qubits black. (b) Represents how the atoms are shifted into the spare qubits to accommodate a lost atom under the virtual remapping strategy. Notice that the interaction is no longer within interaction distance 1. (c) Demonstrates how rerouting strategies swaps a qubit to a valid interaction configuration, then returns it. Numbers indicate the order of swaps.

are fast since only a single compilation step is needed.

- *Always Full Recompile.* When an interfering atom loss is detected we update the hardware topology accordingly and recompile the input program. This fails when the topology becomes disconnected, requiring a reload.

- *Virtual Remapping.* NA architectures support long range interactions up to a maximum interaction distance. Therefore, shifts in the qubit placement is only detrimental to execution when the required qubit interactions exceed this distance. For this strategy, we start with a virtual mapping of physical qubits to physical qubits where each qubit maps to itself. When an atom is lost we check if it is used in the program. If so, we adjust the virtual mapping by shifting the qubits in a column or row of the qubit in the cardinal direction with the most unused qubits starting from the lost atom to the edge of the device. This process is shown in Figure 3.9b. In this figure, addressing q_b would now point to the original location of q_c, and addressing q_c would point to the qubit to the left of q_c's original location. If there are no spare qubits, we perform a full reload. Otherwise, we execute the gates in order according to the mapping. If two qubits which must interact are now too far apart, we reload. This strategy is efficient in terms of overhead since this virtual remapping can be done on the order of 40 ns in hardware [29] via a lookup table. However, this strategy can be inefficient in number of reloads required since it is easy to exceed the interaction distance. We later explore how many atom losses can be sustained before reload which allows us to estimate how many reloads will be required on average.

- *Minor Rerouting.* Here we perform the same shifting strategy as in *Virtual Remapping* to occupy available spares. However, rather than failing when distance between the remapped qubits exceeds the maximum interaction distance, we attempt to find a path over the usable qubits, and insert SWAP gates along this path. To simplify computation we SWAP the qubits on the found path, execute the desired gate, then reverse the process to maintain the expected mapping. The rerouting is shown in Figure 3.9c. Too many additional SWAPs are detrimental to program success. We may force a reload if the expected success rate drops, for example by half, from the original program's expected success rate.

- *Compile to Smaller Than Max Interaction Distance.* For most programs there are diminishing returns to compiling to larger max interaction distances, but can be sensitive to atom loss. In this strategy we compile to an interaction distance less than the max so when qubits get shifted away from each other it will take more shifts to exceed the true maximum distance. The overhead is the same as *Virtual Remapping* but the compiled program could be less efficient than one compiled to the true max distance.

- *Compile Small and Minor Reroute.* This strategy is based on *Compile to Smaller Than Max Interaction Distance* but performs the same rerouting strategy as *Minor Rerouting* with similar overhead costs.

Each of the above strategies can be augmented by moving beyond the hardware adjacency graphs. Moving along the adjacency graph can maintain locality of nearby qubits, but it perturbs many qubits as we perform different shifting strategies. We can instead shift the

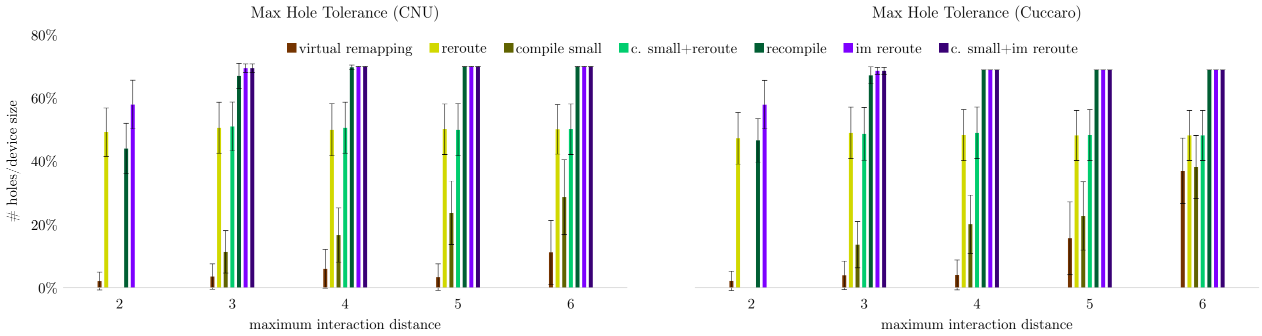


Figure 3.10: Atom loss as a percentage of total device size which can be sustained before a reload of the array is needed. Each program is 30 qubits on a 100 qubit device. As the interaction distance increases most strategies can sustain more atom loss. Strategies like full recompilation can sustain large numbers of atom loss but as we will see are expensive computationally. Fast, hardware solutions or hybrid solutions can sustain fewer numbers of holes but have lower overhead. We show two representative benchmarks parallel vs. serial. "im" represents the interaction graph methods.

qubits along the shortest path to an available qubit on the interaction graph. This moves fewer qubits and can be more flexible. The interaction based strategies are not bounded by the "walls" of the architecture. A demonstration of this is shown in Figure 3.15c.

3.4.2 Experimental Evaluation

Excluding the first approach of reloading with any interfering atom loss, we examine how many losses can be sustained without exceeding the constraints of the architecture in size, dimension, or needed interaction distances. Figure 3.10 shows the maximum number of holes supported by the different strategies for a 30 qubit Cuccaro adder and a 29 qubit CNU. The entries for *compile small* and *compile small + reroute* are compiled to one less than the maximum interaction distance. We do not compile to interaction distance 1, so we do not have entries for these strategies at interaction distance 2.

Maximum Resistance to Loss

As would be expected, *recompile* is able to support the most lost atoms since the only failure cases are: disconnected hardware topology or fewer atoms than required qubits. In fact, since our example circuits use 30% and 29% of the hardware, once the interaction distance overcomes any disconnected pieces, recompiling can sustain 70% atom loss and is the ideal case for how many atoms can be recovered by a given coping mechanism. The non-rerouting strategies, while a fast solution since no paths need be found, offer limited atom loss recovery since it is easy to move outside the maximum interaction distance. The simple virtual remapping is only able to support a small amount of atom loss but does increase as the max distance increases. As predicted, compiling to a smaller interaction distance does enable more resilience to atom loss since more movement can be tolerated before exceeding the maximum interaction distance. Both rerouting strategies have the disconnected topology failure case, but reach the additional failure case of not having space in any direction to shift the qubits in the event of atom loss. This does not occur as often for the non-routing mechanisms since we hit the barrier of exceeding the interaction distance first. Due to the lack of space to move the qubits both of the rerouting strategies for hardware based shifting are only able to sustain 50% atom loss at higher interaction distances. Surprisingly, we find that the interaction graph based shifting and rerouting strategies are comparable to recompiling from the start. But, this is due to the fact that we are not limited by the boundaries of the architecture in these strategies, the shifting strategy moves the qubits where ever needed until either we reach the disconnected graph, or run out of space for the required qubits. These are the same failure conditions of the recompilation case, so is able to recover the same amount of lost atoms.

Success Rate

However, these different strategies add varying numbers of extra SWAPs to handle atom loss, lowering the success rate. As more atoms are lost, more SWAPs are needed, and the rate decreases as seen in Figure 3.11 for Cuccaro and CNU with the rerouting and recompiling strategies. With current error rates, success is very low for 30 qubit circuits. To better demonstrate how the success rate changes, we use lower error rates so that, without atom loss, about 2/3s of shots succeed. For any strategy, as the interaction distance increases, fewer SWAPs are needed so the shot success stays higher. Since the recompilation strategy is able to schedule and map qubits with full knowledge of the current state, including missing atoms, it can achieve the best routing and success rate out of all the atom loss strategies. All of the rerouting strategies have lower rates since, as seen previously, they tend to add more SWAPs per atom loss as compared to recompilation. This is even more pronounced for the interaction graph based solutions, since the qubits are moved further away from their original positions. But, since compiling to a smaller MID before rerouting means the interaction distance is exceeded less often, these strategies require fewer SWAPs, boosting its rate over simply rerouting.

However, when examining the decrease in success rate for interaction graph based shifts, we find that the potential to move qubits much further away from their original positions is detrimental to success rate as well. We are not just increasing the depth of the circuit due to longer range interactions, extra communication is required, increasing the number of gates, and decreasing the probability of success more than the hardware based solutions.

Overhead Time

Taking this into account, we examine the estimated overhead time of each strategy for 500 runs of a given circuit. We use a 2% chance of atom loss for a measured qubit, and a 0.68% chance of atom loss due to atom collision in a vacuum. The overhead times for CNU are seen

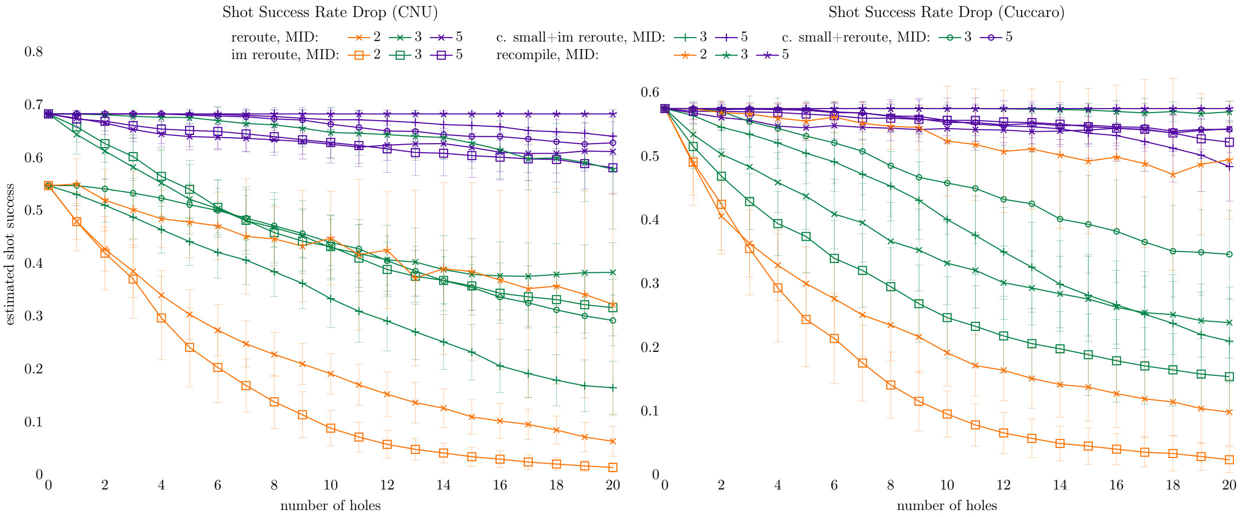


Figure 3.11: For strategies which modify the program such as recompilation or rerouting strategies, additional gates could be added leading to a lower overall success rate. Here we trace the success rate of our three program modifying strategies. The full recompilation strategy (circles) is a rough upper bound which best accounts for holes as they appear being able to move the entire program to a more appropriate location and route best. The gap between strategies on the same MID gets smaller as the MID gets larger. Here we've chosen the two-qubit error rate corresponding to approximate 0.6 success rate to begin with (based on Figure 3.8) in order to best demonstrate the change in shot success probability over a range of atom loss.

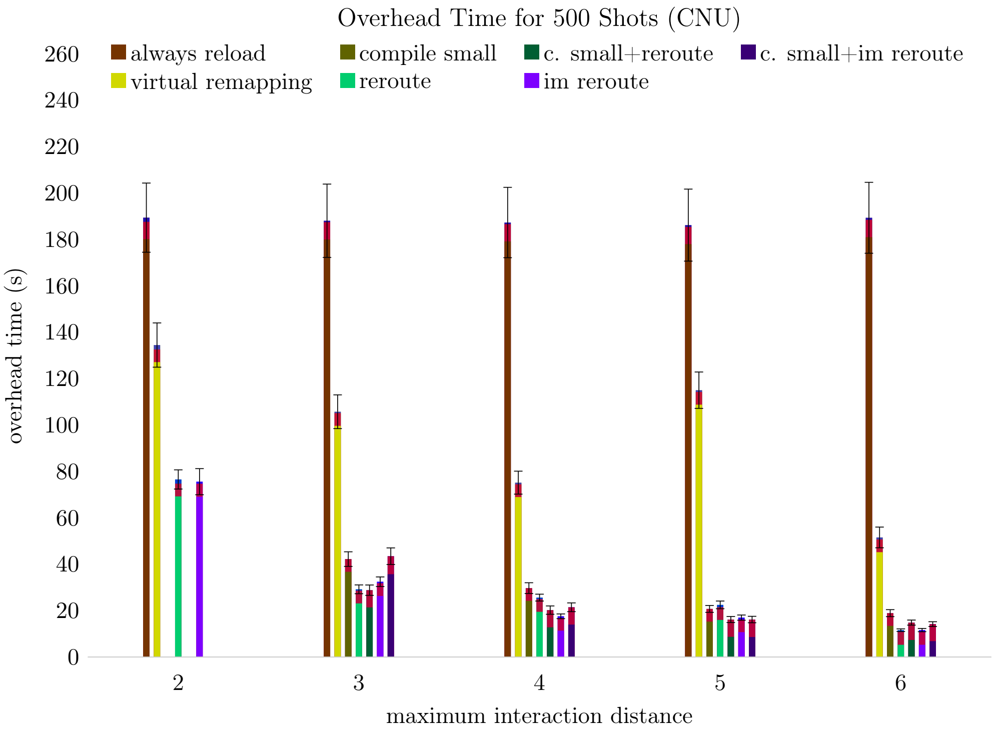


Figure 3.12: Strategies that are able reduce the number of reloads necessary greatly reduce the overhead time when running circuits. Here we show the overhead time for all strategies except recompilation. The proportion of time dedicated to reloading is shown by the dominate color in each bar, followed by fluorescence in red, and recompilation in black. Any strategy whose overhead exceeds that of always reloading, such as full recompilation, should not be considered.

in Figure 3.12. For any rerouting strategy that requires extra SWAPs, a reload is forced once the number of added swaps would decrease the success rate by 50%. For a 96.5% successful two-qubit gate, this would be six SWAPs.

Recompilation is not shown in Figure 3.12 as software compilation exceeds the array reload time, and the overhead time is larger than simply reloading. Other strategies are always more time efficient than reloading all of the atoms. Additionally, since compiling to a small size requires fewer fixes with swaps, the overhead time tends to be smaller at lower interaction distances. Essentially, the fewer times the strategy requires a reload the faster the overhead time is. Subsequently, we see our interaction model rerouting strategies performing less well as compared to the basic hardware based methods as the additional

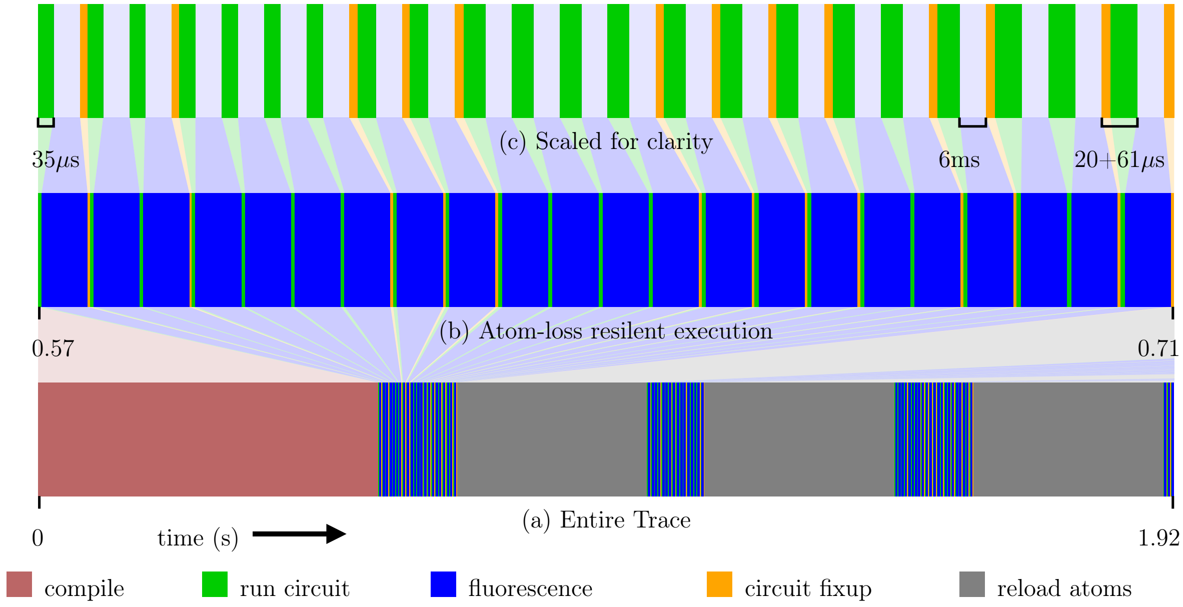


Figure 3.13: A timeline of 20 successful shots for *Compile Small and Reroute* with reload time of 0.3 s and fluorescing time of 6 ms. A majority of the overhead time is contributed by the reload time and fluorescence, indicating, that the duration and count of these actions is crucial to overall runtime.

swaps require more reloading more often. Additionally, as interaction distances increase, the overhead time of each strategy converges. A sample timeline of 20 successful shots using compile small and reroute can be seen in Figure 3.13. After initial compilation, reloading takes a majority of the time, so any ability to reduce the number of reloads vastly reduces the overall run time.

Scalability

Compile small + reroute is an efficient way to improve loss resilience, we next examine the sensitivity of the successful shot count before a reload to the rate of atom loss for this strategy. Figure 3.14 shows how the number of successful shots changes as the rate of atom loss changes. A 10x improvement offers an expected 10x improvement in the number of successful shots before a reload must occur. This is because the rate of atom loss decreases as technology improves, reducing the number of reloads, improving overhead time.

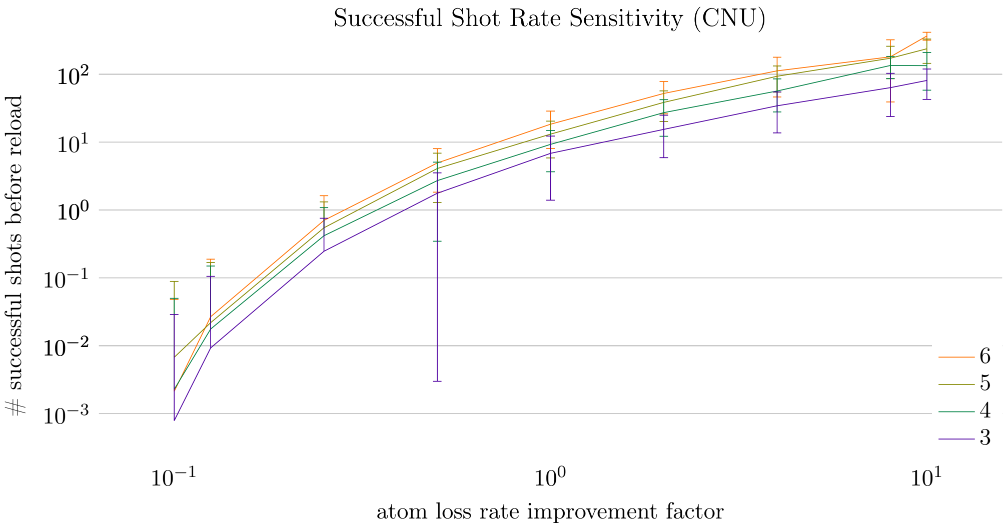


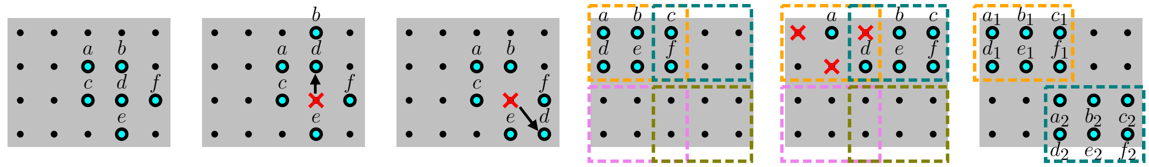
Figure 3.14: Sensitivity to the rate of atom loss for the balanced *Compile Small and Reroute* strategy. In prior experiments we used a fixed rate of 2% atom loss. For larger systems this rate could be worse and in the future we might expect this rate to be much better. For each interaction distance we see as the rate of atom loss gets better we can run many more trials before we must perform a reload and reset. Some error bars don't show on the log axis.

3.5 Designing for Atom Loss

The previous strategies for atom loss only acted on the originally compiled circuit. In the original compilation the mapping and routing strategies had access to the entirety of the architecture. Here, we constrain the mapping and routing problem to specific sections of the architecture in order to further increase the usefulness of the device before performing a reload of the neutral atom array.

3.5.1 Constraining Methods

- *Focused Use and Migration.* In the original compiler, a circuit could be mapped and routed onto any qubit in the architecture. This gives global scope, and finds good mapping and routing solutions, but can lead to less dense mapping across the architecture. Instead, we define a bounding box big enough to hold the circuit, seen in in Figure 3.15d, where for a six qubit circuit, we define a 2 by 3 bounding box. This can be done



(a) Initial Mapping Compilation

(b) Hardware Remapping

(c) Interaction Remapping

(d) Focused Use Bounding Boxes

(e) Focused Use Post Relocation

(f) Full Parallelism Initial Mapping

Figure 3.15: Examples of four different atom loss coping strategies. (a) Shows the initial configuration of qubits using the original compiler that makes use of the entirety of the architecture. (b) shows the result of using the hardware graph to shift the qubits away from a newly lost atom. (c) shows how the interaction graph could make a different decision, taking the shortest interaction path to an open qubit. (d) demonstrates how the architecture can be divided such that we have multiple opportunities to start in a section with fewer lost qubits. (e) shows how a wear levelling approach would appear after one remapping of the circuit. (f) shows how these divided sections can be used to exploit some level of parallelism in the architecture.

in two ways: a loose or tight configuration. The former defines both dimension by the ceiling of the square root of the number of qubits. The latter defines one dimension by the square root, and the second by the number of qubits divided by the square root. This box is then tiled across the architecture. In the event that the bounding box does not neatly fit, some overlap is allowed. The circuit is mapped and routed entirely within one of these sections. When atoms are lost, we continue using any recovery method previously defined. When the recovery method fails, or the estimated probability of success falls below a certain threshold, rather than resetting the array, we directly remap the circuit to a new section of the architecture, accounting for any previously lost atoms. The process after one relocation is shown in Figure 3.15e. Since the atoms in the new section have not been used for computation, fewer will have been lost. We repeat this process until each section has been visited once. Then, the entire array is reloaded before restarting the process.

- *Parallel Executions.* The non-overlapping sections defined for *Focused Use* can be treated as if each is its own architecture. Instead of shifting the mapped circuit from one defined region to the next, we can create a single aggregate circuit [30] that contains multiple instances of the smaller circuit. This is similar to [88] where multiple variational circuits have been mapped onto a single architecture to improve their performance. We map the circuit onto the architecture multiple times, and run multiple shots of the individual circuit in one run of the aggregate circuit. Then any individual circuits that lost no computational atoms can be treated as a successful shot. If an atom is lost, we can use any recovery methods previously explored until it fails. At this point, the array is reloaded.

- *Focused Use + Partial Parallelism.* Rather than filling every section with qubits, we only fill some of the sections. Then, when the recovery strategy fails, we use a different set of the predefined sections. We are exploiting some amount of parallelism, but are giving more resources for the more focused recovery strategies to make use of. This process continues until each section has been used at least once. The entire array is then reloaded.

3.5.2 Evaluation

We use the same two benchmarks used in the previous atom loss section, with the addition of two more near term algorithms variational algorithms. These are the QAOA circuit, described previously and the VQE circuit. VQE attempts to find the minimum eigenvalue of a wave function encoded in parameterized rotation gates and entangled qubits [106]. It is then run several times while tuning the rotation gates. We test one of these iterations. We use linear entanglement, meaning that the first qubit targets the second, the second then targets the third, and so on. This lends itself to very low density, highly serialized circuit. We study these circuits at multiple sizes to understand how our recovery strategies scale as well.

We also use the same atom loss simulation parameters included the 10 by 10 array of qubits, and the same rates of loss of atoms from both an imperfect vacuum and measurement.

3.5.3 Results

We already understand how our original compiling-to-smaller interaction distance with virtual remapping and rerouting fares best among our recovery with respect to recompilation in maintaining fidelity and best in overhead time, and we will use it as our baseline strategy to beat without new relocation and parallelism based strategies.

While shifting along the interaction graph provides benefits for larger circuits, we find that focused use, and relocation to unmapped sections of the architectures can improve probability of success across circuit size, Figures 3.16 and 3.17. Initially, as atoms are lost, this strategies maintains the same probability of success as our baseline. This is expected since we are mapping into a specific section of the architecture, we can still generate a similar mapping and routing. However, after several atoms have been lost, we see an increase in the average probability of success indicating a relocation to a new part of the architecture. Since there have been fewer atoms lost in this new section there is less adjustment, increasing the probability of success. Focused use and relocation is able to more closely adhere to the probabilities of success achieved via recompilation, making it a more effective strategy from this perspective. While we do not achieve strictly better probabilities of success in the 90 qubit circuits, we are able to maintain the same probabilities of success. We do not see the same benefits since the tiles in this circuit will need to be quite large, at least 9 qubits by 10 qubits. This requires significant overlap and relocation will not have as great an effect. Additionally, this strategy does not make use of the interaction graph remapping, and does not benefit from the increased flexibility.

It should also be noted that we do not see a significant difference between the loose and tight bounding boxes. Both follow the same trend line in each of the benchmarks, indicating that the circuit is mapped and routed similarly in each case.

We should also note that remapping along the hardware array often fails for larger circuits. For the 90 qubit circuits, Figure 3.17, we fail to find significant average probabilities of

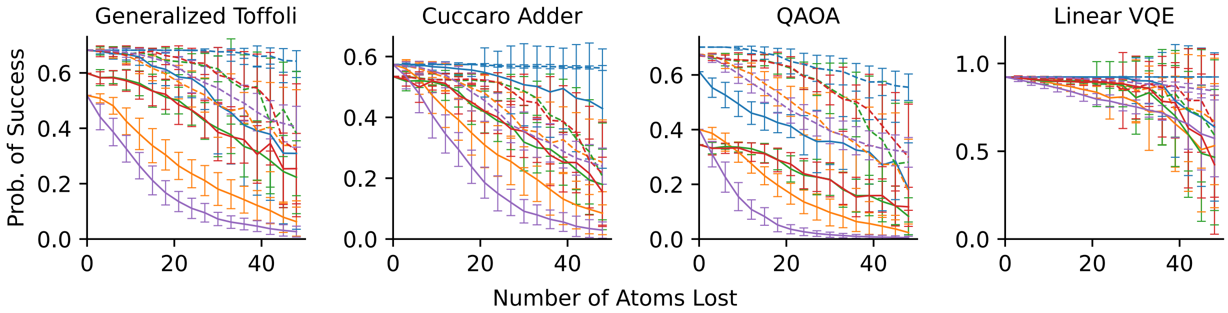


Figure 3.16: Decreases in success rate for different mitigation strategies for 30 qubit circuits. Each color is a different strategy, solid lines are circuits with interaction distance 3, and dash lines are interaction distance 5. Run over 50 trials, each error bar represents one standard deviation from the mean. We see decreases for all strategies, with full recompilation able to maintain the highest probability of success and relocation better approximating recompilation than our baseline strategy.

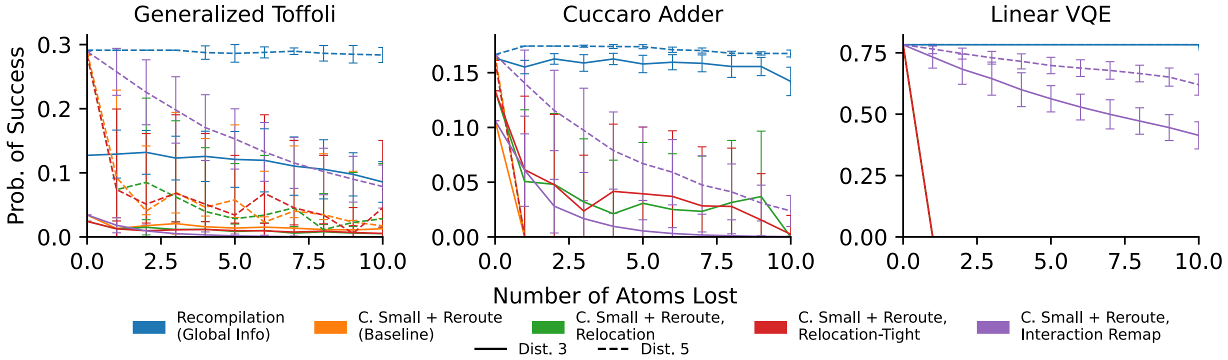


Figure 3.17: Decreases in success rate for different mitigation strategies for 90 qubit circuits. We exclude QAOA from this test as the success rates are not meaningful at current error rate. Each error bar represents one standard deviation from the mean over 50 trials. We see the same patterns, with the exception that interaction remapping is often able to outperform other strategies.

success for several circuits after the loss of more than one or two atoms when using our original strategies. They do not provide enough flexibility to find available atoms at this size, which is why some strategies cannot be seen on the graph. However, when we shift along the interaction graph, we are able to achieve much higher fidelities. Particularly for the Cuccaro Adder, we see that at for a maximum interaction distance of 5, the interaction model based approach achieves much higher probabilities of success. The non-recompilation counterparts are not able to sustain a minimal amount of atom loss at this size of circuit.

Successful Shots before Reload

Another metric to consider is how many successful shots can be completed prior to reloading the entire array. Every shot that can be performed without an additional reload reduces the overall time to run a circuit. We analyze the average successful shots per reload cycle for each mitigation method for each of our benchmarks.

There is a much more pronounced effect on average shots per reload when utilizing the new relocation strategy over the rerouting strategy for both large and small circuits, Figures 3.19 and 3.18. We find a much higher rate of average shots per reload for relocation based strategies. Once the array has been divided into several different sections, when the circuit is relocated to a new section it will be mostly free of atom loss. Any atoms lost will be due to computational atoms being remapped into that space, or the rare event of atoms lost to environmental factors and will be much lower. All of which is conducive to much higher probabilities of success and reduces reloads.

As the circuit size increases, we see diminishing returns of the relocation strategy. As the circuit size increases, the number of distinct sections that can be laid out on the array without overlap, decreases. We do not have as many opportunities to perform a pseudo-reload as the circuit uses more qubits. We can even quantify this relationship. In Figure 3.20 we see the relationship between the increase in the number of shots against the number of times the

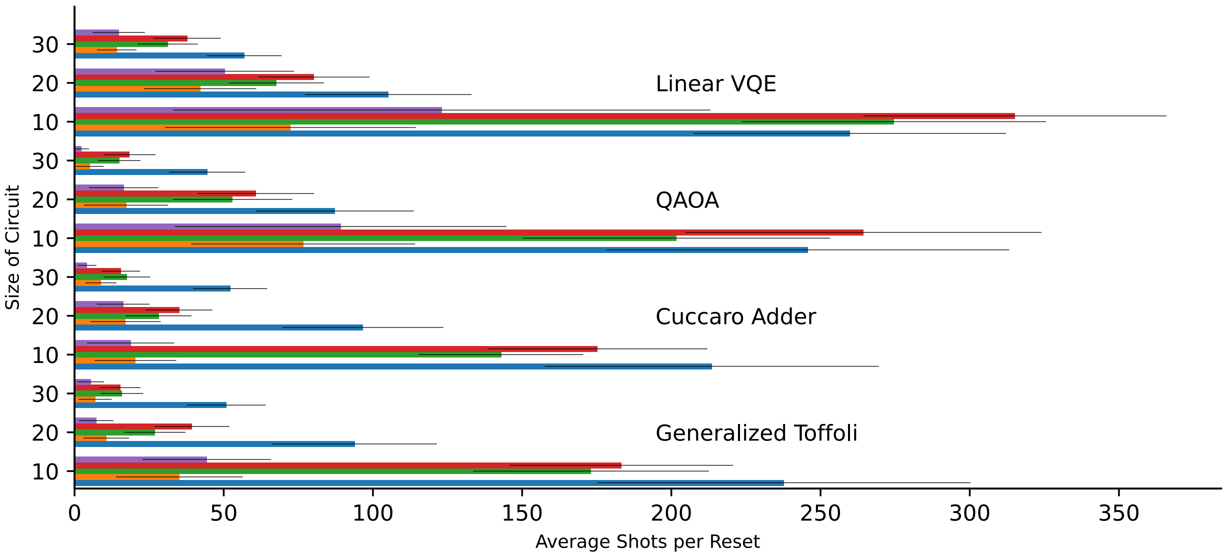


Figure 3.18: The average number of shots per reload 10, 20 and 30 qubit benchmarks at interaction distance four. Error bars indicate one standard deviation from the mean. Each color is a different recovery strategy. We find that there is significant improvement from our relocation strategies (red and green), greatly exceeding the baseline number of shots, more closely matching recompilation.

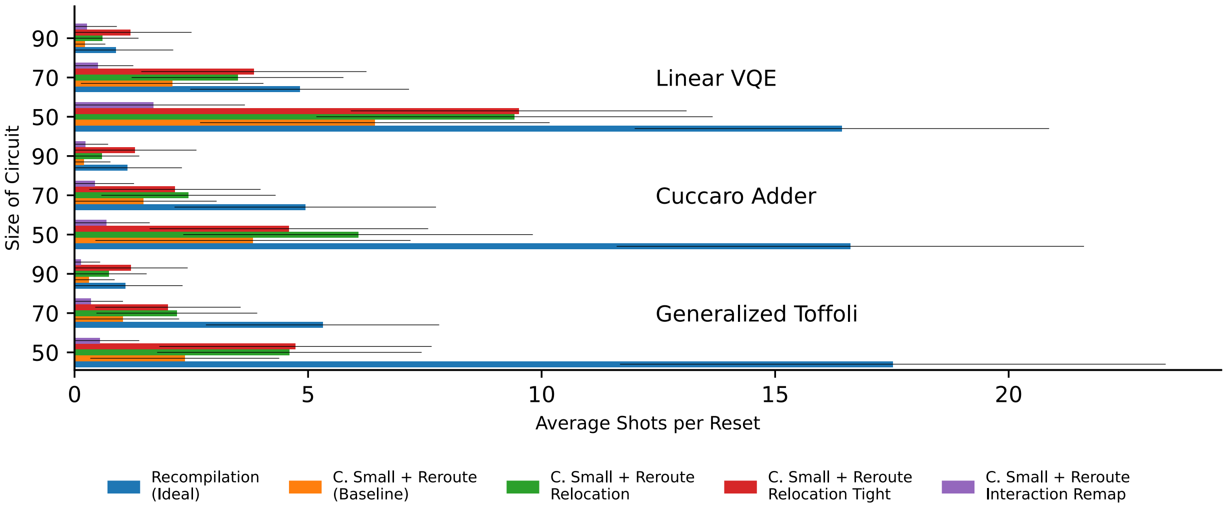


Figure 3.19: The average number of shots per reload 50, 70 and 90 qubit benchmarks at interaction distance four. Error bars indicate one standard deviation from the mean. Each color is a different recovery strategy. We exclude QAOA since it is not a practical circuit at this size.

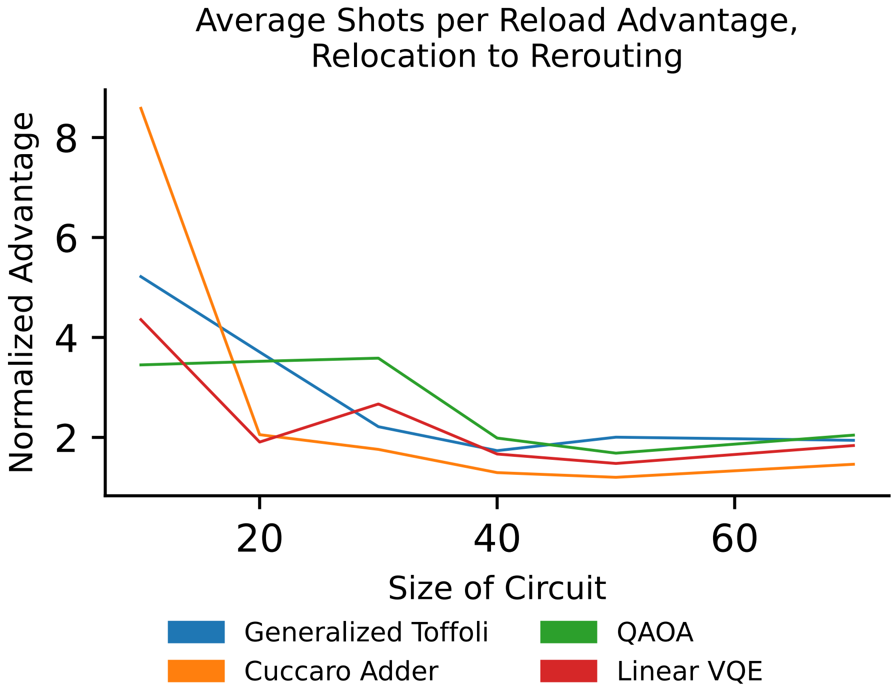


Figure 3.20: The normalized advantage of the relocation strategy to rerouting and interaction distance four. Each color represents a different circuit, and the advantage of relocation decreases at a factor inversely proportional to the size of the circuit.

circuit can be fit onto the architecture. For each circuit, we follow the same pattern. The advantage in the number of shots of relocation to rerouting is roughly proportional to the size of the circuit. For a 30 qubit circuit, this is up to 3.5x improvement. For a 10 qubit circuit, this is up to a 8x improvement in average shots per reload. Empirical results do not match the exact ratio since the bounding box tiles do not fit always fit neatly onto the array. Additionally, these bounding boxes are often larger than the circuit, causing the advantage to be lower.

While average shots per reload cycle is good indicator for a strategy's potential to recover efficiently, it does not take into account the overhead time to determine the best course of action. A strategy is only viable if it is faster than reloading the array. If this cannot be achieved, it is more effective to reload the atoms as it will give us the highest probability of success. Full recompilation cannot be considered for this reason. Previous work found large gains over reloading via the compiling to a smaller interaction distance and rerouting. We will be comparing against this strategy to determine effectiveness.

In Figure 3.21, we examine the effects of different mitigation strategies on the overhead time to execute 500 shots for a 30 qubit Generalized Toffoli circuit. While these strategies are dependent on the length of the circuit, it will scale similarly for each strategy. We see significant reduction in time dedicated to reloading the entire array for our relocation strategies and increases for interaction graph based strategies. However, the increases in calculating the solutions for relocation do not outweigh the overhead time saved by relocation. For interaction distance four, our relocation strategy outperforms the basic rerouting strategy by 55% in reload times, and 45% overall.

As the maximum interaction distance increases, the margin of the overall time for each strategy decreases. As the maximum interaction distance increases, we do not need to add communication to recover from an incompatible circuit. This reduces the number of reloads required for each recovery method. The time dedicated to florescence does not decrease, as each shot, successful or unsuccessful, still requires a fluorescence.

Effects of Parallelism

The benefits of relocation could be improved upon by using multiple non-overlapping sections of the device at the same time. In doing so, we use as much the architecture as possible. For example, using 90 qubits to run three concurrent 30 qubit circuits. Since we are using almost all of the qubits, we will use the interaction graph based remapping as it has proven to be more effective in situations where most of the architecture has been filled. We examine the average probability of success for three concurrent 30 qubit circuits in Figure 3.22. This strategy can only withstand 10 lost qubits at the most since we can only recover as many qubits as those that have been unmapped. Any increased serialization from running circuits in parallel does not substantially impact the decrease in probability of success as compared

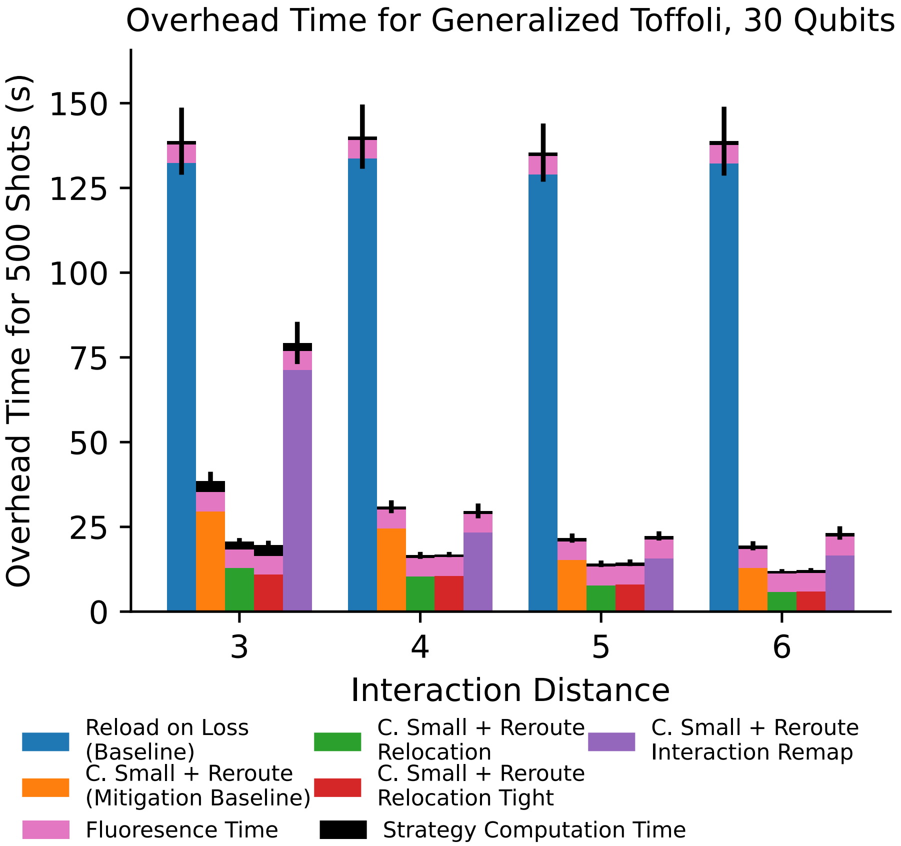


Figure 3.21: Overhead times for different recovery strategies at different maximum interaction distances. The major color in each bar represents time dedicated to reloading. As follows from the significant advantage seen previously, relocation is a very efficient strategy.

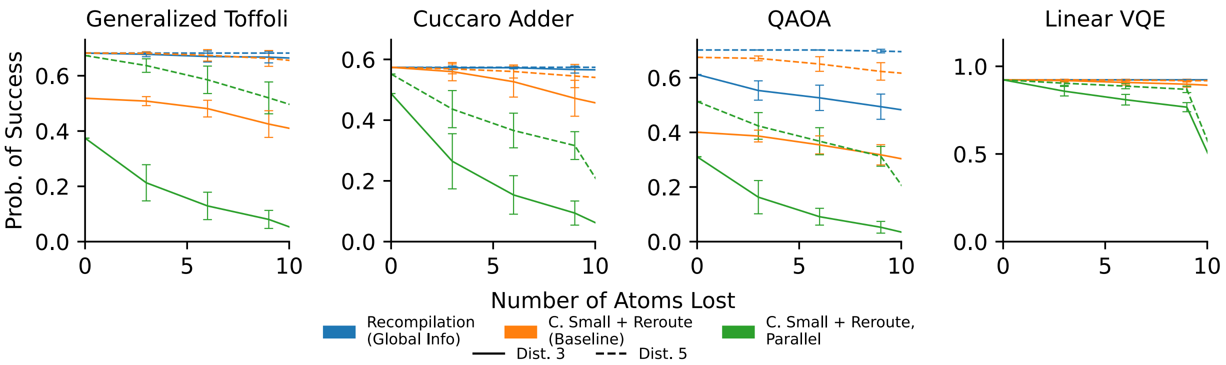


Figure 3.22: Decreases in success rate for different mitigation strategies for parallel 30 qubit circuits. Run over 50 trials, each error bar represents one standard deviation from the mean. In general, we see that for higher interaction distances, increased parallelism does not heavily affect the rate of decrease in probability of success rate.

to previous mitigation strategies.

We also analyze the effects of this strategy on the number of average shots per reload cycle in Figure 3.23. By packing as many instances of circuits as possible into the array, there is no space to shift the qubits causing the recovery strategy to fail quickly and requiring a reload. This frequency will increase the overhead time, indicating that full parallel usage of a neutral atom architecture is not viable at current atom loss rates.

Parallelism with Relocation

Without extra atoms to recover from atom loss, full parallelism is not effective. For smaller circuits, we do not need to fill the entire architecture. By choosing to only use a portion of the previously defined bounding boxes, we can take advantage of partial parallelism. By reducing parallelism, we have the opportunity to make use of relocation as well, potentially adding to the overhead time gains we have already found. In Figure 3.23 we examine multiple levels of parallelism for 10, 20 and 30 qubit circuits. The percentage of parallelism indicates what percentage of the architecture we are using at a time, 30% parallelism means that three instances of a 10 qubit circuit are being run at a time, with 3 different areas to relocate these

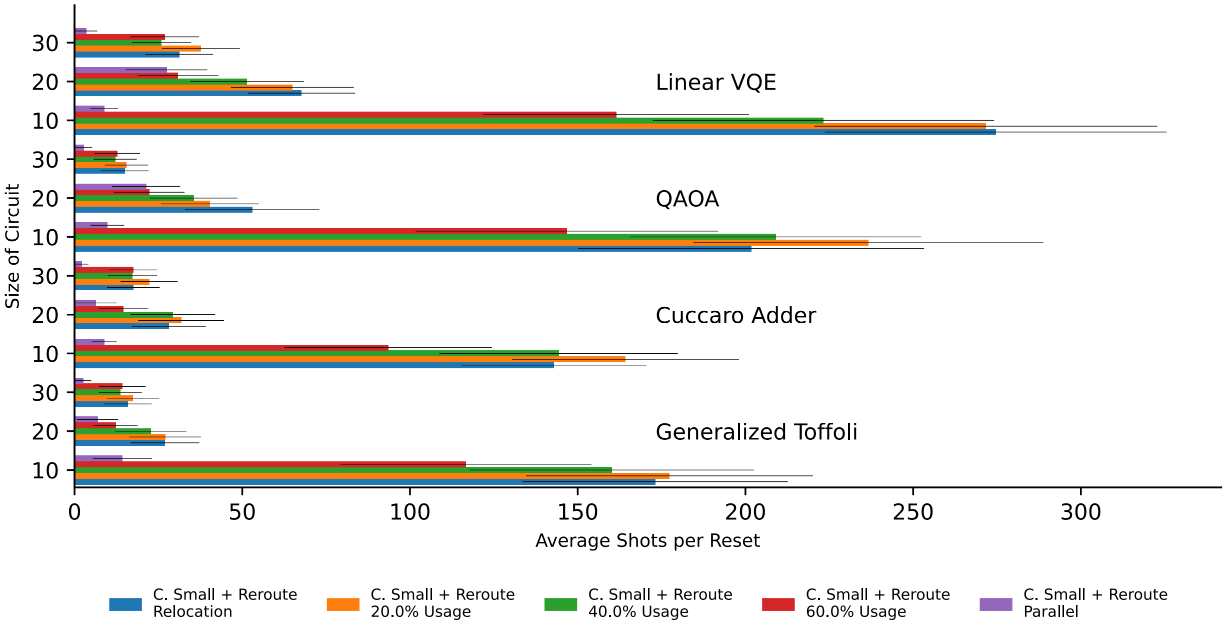


Figure 3.23: The average number of shots per reload for 10, 20 and 30 qubit benchmarks at interaction distance four at different levels of parallelism. Error bars indicate one standard deviation from the mean. Each color is a level of parallelism. Parallel indicates that as much the architecture is used as possible, and relocation means that only on instance is run at a time.

Benchmark	Baseline Time (s)	Relocation Time (s)	2 Parallel Time (s)	% Fluorescence Decrease
Cuccaro-10	10.91	4.67	3.13	51.22
Cuccaro-20	13.82	9.58	8.24	54.02
Cuccaro-30	26.19	17.05	14.80	40.71
CNU-10	7.98	4.59	2.82	50.60
CNU-20	24.09	8.68	10.50	53.12
CNU-30	30.61	16.58	15.15	35.80
QAOA-10	5.83	4.41	2.66	51.05
QAOA-20	14.83	7.54	7.49	52.70
QAOA-30	42.67	16.43	18.68	35.48
Linear VQE-10	6.26	4.16	2.42	50.83
Linear VQE-20	8.77	6.61	5.01	53.75
Linear VQE-30	15.93	9.79	9.56	36.12

Table 3.1: Full Runtimes with Two Instances of Parallelism

three 10 qubit circuits. We can achieve similar shots per reload to the relocation strategy when running up to four instances of a 10 qubit circuit, and up to two instances of a 20 qubit circuit or 30 qubit circuit.

Since setting up parallelism occurs during the initial compilation phase, this strategy will realize the same gains as relocation in terms of reduction of overhead time due to reload time. However, by running multiple circuits at the same time, we are executing several shots per fluorescence cycle. We will have successfully reduced the overhead time from fluorescence by a factor of the number of circuits run at the same time. This can be seen in Table 3.1 for two parallel instances for 10, 20 and 30 qubit circuits. Put together with the gains from relocation, we see total reductions in overhead time for each of our benchmarks of up to 70% for a 10 qubit circuit, 60% for a 20 qubit circuit, and 50% for a 30 qubit circuit, seen in Table 3.1.

3.6 Discussion

Reducing the overhead of running compiled quantum programs is critical to successfully executing useful near- and intermediate term quantum algorithms. Neutral atoms have many attractive properties: long-range interactions, native implementation of multiqubit gates, and ease of scalability. These advantages reduce gate counts and depths dramatically by increasing the relative connectivity of the underlying hardware. While long range interactions induce larger restriction zones which inhibit some parallelism, the amount of gate and depth savings far outweighs this cost.

The dominant cost in NA systems is atom loss. Weaker trapping potential and destructive measurement leads to the loss of atoms as computation is performed. We explore various strategies including the extremes of full recompilation and always reloading. Full recompilation is able to sustain high atom loss but is slow when thousands of trials are needed. But reloading is also slow and is the dominant hardware cost. Our reroute and compile small strategies balance atom loss resilience and shot success rate to save computation time. As resources become more constrained, we require more flexible techniques. By exploiting the unique feature of long distance interactions, we can make use of a small number of atoms to improve how many successful shots can be achieved before reloading the circuit via the interaction graph. This flexibility gives an escape valve for more edge case circuits.

However, we do not only focus on adjusting the initial mapping and routing. By changing the initial mapping and routing while focusing on effectively using all of an architecture and more actively avoiding lost atoms, we can further reduce overhead time. Through the creation of tiled bounding boxes, dividing the array into several small architectures, and fully exhausting each of these sections before moving into a new section without atom loss overhead time due to reloading the array is reduced by a factor of the number of times the circuit fits on the architecture. This can all be done while keeping the probability of success high and without significant increases in the time to adapt the circuit to the lost atoms.

Finally, we also use these different sections to exploit parallelism. While using every available atom on a device is not effective due to space constraints, using less than 100% of the architecture combined with relocation achieves the same number of shots per reload while also reducing the number of runs of the array required. This reduces the overhead time dedicated to florescence, further improving our ability to quickly run repeated shots on the neutral atom device.

Overall, the unique advantages of neutral atom systems, long-distance interactions and multiqubit operations, dramatically reduce communication and depth overheads which translates into lower error rate requirements to obtain successful programs, showing potential for the advancement of neutral atom architectures. Like their competitors, there are fundamental drawbacks of a NA system; here we've highlighted the problem of atom loss. This probabilistic loss is inherent in the trapping process itself. But, we demonstrate that software solutions can effectively mitigate the problems due to atom loss. This is critical for the overall development of the platform: by solving fundamental problems at the systems level, hardware developers can focus on solving and optimizing other problems, such as gate errors, and focus on the process of co-design which can accelerate the advancement of the hardware tremendously.

CHAPTER 4

HIGHER-RADIX ARCHITECTURES: TAKING ADVANTAGE OF VARIABLY COSTED OPERATIONS

4.1 Introduction

As was covered in previous sections, a typical quantum architecture is constructed of binary qu*bits*, which have two distinct states representing 0 and 1. However, this binary abstraction is not the whole picture. Some quantum hardware, such as superconducting qubits [25] and trapped ions[90], have natural access to energy levels beyond the 0 and the 1 states that can be used as extra computation space. And, some algorithms have been designed specifcally to take advantage of higher-radix computing [103, 52].

One potential option is the use of *intermediate qudits* [42], which rely on a mixed radix strategy where a smaller number of qudits are used and are used for a shorter amount of time. While specialized in its use, the expected advantages are strong, for example in the generalized Toffoli decomposition (also the subject of this work) using qutrits (three level quantum systems) temporarily enables a logarithmic depth (approximately circuit duration) decomposition with linear number of two-qudit gates and requires no additional ancilla space, scratch bits. The best known qubit-only circuits can obtain logarithmic depth with linear gate counts [9]. These circuits require linear amounts of additional space which makes efficient implementations infeasible on error-limited devices and bounding the program size on available hardware.

Another uses the concept of four-level ququarts to compress the data of two qubits into a single physical ququart device without the loss of information. This is a more general way to save computational space. However, this strategy has been avoided due to several drawbacks that scale with qudit dimension: quadratically increasing logic gate execution time [67], reduced coherence time, and increased difficulty of experimentally applying gates [51, 15].

These are significant disadvantages for a NISQ device that already has limited connectivity, short qubit lifetimes, and high gate error. However, if these issues can be effectively mitigated, through specialized gates and compilation strategies, this general compression method can effectively *double* the available computational space on a quantum computer for any arbitrary circuit.

This chapter explores both methods of utilizing higher radix quantum devices and how we need to adapt our mapping and routing methods to handle these cases. We will particularly focus on the trade off of gate count for circuit duration, and whether we reduce communication costs through clever mapping and routing to avoid increases in circuit duration such that the error due to decohering qubits outweighs error reduction in other area.

4.2 Background: Higher-Radix Computation

Classically, the basic computation unit is the bit which takes the value of either 0 or 1. In the quantum setting, we often consider the quantum bit (qubit) the most fundamental unit. For many quantum technologies, such as superconducting qubits and trapped ions, the implementation naturally has access to infinitely many discrete levels which can be truncated to any dimension d giving qu*dits* which exist as linear superpositions superpositions of d levels as $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle + ... + \alpha_{d-1} |d-1\rangle$ where if we choose $d = 2$ we recover the qubit. In general, qudit computation is not asymptotically better than qubit computation – both schemes can universally express quantum computation, and full translation from one radix to another affords only constant advantages [42].

Use of additional logical levels for quantum computation is not new. There have been two primary considerations for their use. First, full translations from qubits to qudits, usually of small dimension d, have been proposed. For example, translation has been used to implement arithmetic or Shor's algorithm [18, 17], which require an expanded gate set to implement generalized ternary gates. Second, temporary use of additional logical states has been shown to be useful in specific cases. In these cases, programs begin and end entirely as qubits, but during computation temporarily access additional logical levels. These works focus on a small set of applications such as the generalized Toffoli gate and adders [8, 111] and rely on hand optimization to extract benefit from these states without using too many gates or spending too much time occupying these states. This temporary use strategy has been generalized as *compression* [7] to generate ancilla to speed up specialized subcircuits. These prior works operate in the gate model only, assuming that gates on different dimension qudits are effectively the same - they can be executed with equivalent fidelity and execute in a similar amount of time. They also generally ignore architectural connectivity constraints and the inherent increased cost to communicate qudits at long distances. While gate representations are useful, they omit crucial systems level details, such as gate duration and pulse implementation fidelity, which determine the viability of mixed radix computation in general. Prior work has shown a worst-case quadratic increase in gate *duration* [67] for higher-dimensional gates, meaning in practice we must be extremely careful about how we use qudits.

For a variety of reasons, such as increasing number of error channels which become harder to control, using large numbers of states is often impractical and instead we should carefully choose the computing radix based on our target applications and available hardware. For example, measurement (the process of collapsing a quantum state to a classical value) of high level systems is often challenging for trapped ions and if possible we should try to measure only qubits. Additionally, characterizing higher energy levels is difficult as they are more prone to noise and suffer from lower coherence times, at a rate of around $T_1/(d-1)$, where d is the qudit dimension and T_1 is the coherence time for a qubit [15].

As mentioned in 2.1, to manipulate quantum states we apply *gates*, can be represented as unitary matrices. For the most part, hardware supports at most gates on 1 or 2 inputs and all larger gates must be synthesized directly from smaller ones. For higher radix computation, in 4.4 we consider a basis set which consists of the generalized versions of the X_{+k} gates and the $X_{i,j}$ gates which are classical permutations of the basis elements. The first behaves by shifting every basis element's coefficient $+k$ modulo the dimension of the system. The second behaves by swapping the coefficients of the i-th and j-th states and leaving all other coefficients the same [76]. We also consider the controlled versions of these gates. In some decompositions we will use Toffoli-like gates on higher dimensions, which have constant depth decompositions into 1 and 2 qudit gates.

However, we can expand this gateset to be any set of operations that interact with the $|2\rangle$ and $|3\rangle$ state. In 4.4 we will use these gate sets to develop higher-radix circuits that use these states in an intermediate capacity, expanidng on the work in [8]. In 4.5.1 and 4.6 we will develop a gate set that is functionally the basis of a two-level system, but operates on a higher-radix ququart based architecture.

4.3 Background: Quantum Optimal Control

Gates on a device are implemented by applying hardware-specific control fields $f_k(t)$ to the qudit(s) involved in the operation. In superconducting architectures, control fields are analog microwave pulses. Quantum optimal control searches for a control sequence that best replicates the effect of a target logic gate. Different algorithms and toolboxes have been designed for this purpose [57, 100, 83, 46]. In this work, we use the open-source optimal control software Juqbox [84, 83] to find the shortest-duration control pulse sequence that reaches a specified fidelity for each logic gate of interest. Juqbox optimizes the control fields $f_k(t)$ to minimize an objective function $J[f_k] = 1 - F[f_k] + L[f_k]$ consisting mainly of the gate fidelity

$$F[f_k] = \frac{1}{h^2} \left| \mathrm{Tr}\left\{ U_T^\dagger[f_k]\, V \right\} \right|^2 \tag{4.1}$$

between the target unitary V and the applied transformation $U_T[f_k]$, where h is the Hilbert space dimension of the logical subspace. This is achieved by repeatedly solving the Schrödinger equation and adjusting the control fields in every iteration to minimize J. The full Hilbert space of the optimization typically includes additional guard states to capture the influence of higher energy levels present in superconducting systems [61, 83]. As these guard states are not part of the logical subspace, their populations are penalized with a leakage term $L[f_k]$ in the objective function.

4.4 Routing Intermediate Qudit Circuits

While the advantage conferred from clever circuit design is clear from prior exploration of intermediate qudits, prior works omit two critical considerations from their constructions. First, these circuits are not compiled to any real device. While some devices promise all-to-all connectivity, most devices have limited connections between devices meaning programs must have movement operations inserted during compilation. This introduces large amounts of communication overhead, on average, which must be minimized [50]. Furthermore, swapping states different dimensional qudits requires tailored communication operations which get more expensive as the dimension of the inputs increases. Second, qubit gates and qudit gates are not created equal. From a circuit point of view, they take up one unit of time each, but when implemented on device this may not be the case. To execute gates on hardware, analog pulse sequences must be generated, for example via the optimal control process described earlier. The duration of these pulse sequences determines the length of a given gates. In some cases, converging to a high quality pulse sequence of minimal length is difficult [83, 84].

In this section, we consider a straight-forward compilation of intermediate qudit circuits to connectivity-limited superconducting devices. To do so, we introduce swap gates designed explicitly to communicate qudits of different dimensions, for example qubits with ququarts

(4 level systems). Second, we consider realistic gate times obtained via optimal control for superconducting based quantum architectures to better study expected circuit durations beyond circuit depth. Together, we use these compilation tools to study one representative implementation of the generalized Toffoli using different levels of intermediate qudits. In particular, this section explores

- Detailed efficient decompositions of SWAP gates between qudits of different dimensions
- A simple compilation pipeline transforming intermediate qudit circuits into ones which obey connectivity-limited hardware constraints
- An introduction of intermediate ququart and ququint (5 level system) implementations of the generalized Toffoli, a critical circuit component for many larger algorithms
- An evaluation of the near-term benefits conferred by using intermediate qudits using realistic gate times for superconducting devices.

4.4.1 Higher Radix Communication Circuits

Current hardware has limited connectivity and only qudits which are adjacent on hardware may interact. Communication operations, called SWAPs, are required to move qudit states around the device. Here, we present a generalized decomposition of the SWAP gate taking in qudits of any dimension.

The decomposition of the qudit SWAP gate is straightforward and follows from the qubit-qubit swap. The key idea is to consider size 2 subsets of the basis elements. For qubits, this amounts to the subset $\{0, 1\}$ for which the decomposition is three 1 controlled X_{01} flips. For qutrit SWAPs we now have three possible subsets $\{0, 1\}$, $\{0, 2\}$ and $\{1, 2\}$ and then perform partial SWAPs as if the qudits exist only in the subspace spanned by the elements of the subset. Here, we would do three 1 controlled X_{01} flips followed by three 2 controlled X_{02}

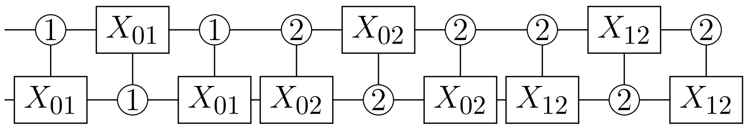


Figure 4.1: The decomposition of a qutrit based SWAP gate.

flips and then three 2 controlled X_{12} flips. The control value can be either of the elements of the subset.

As the dimension increases, a SWAP is decomposed into $O(n^2)$ 2-cycles where n is the dimension of both qudits. For communication between qudits of different dimensions the same strategy applies but the scaling is $O(nm)$ where n, m are the dimensions of the two input qudits. There have been other approaches for generalized SWAP gates by using more contrived basis elements [105, 110], but we find these to be intuitive and match the expected asymptotic scaling that optimal control predicts for gate durations. Additionally, this version of the SWAP gate will handle mixed radix inputs. That is, if one device is in a different radix from the other, the same set of gates for the higher radix SWAP can be used as would be used if they were both higher radix devices. In Figure 4.1 we show a swap gate decomposition for two qutrits.

4.4.2 Synthesizing Gate for Higher Radix Computation with Quantum Optimal Control

By using *optimal control* analog pulse sequences can be produced for a given unitary. Typically, durations are chosen before hand and the goal is to produce a high fidelity (quality) implementation of the gate within the alotted duration. Finding optimal pulse durations is challenging, and is expected to scale quadratically with the dimension of the input unitary. We consider gate times obtained via optimal control to accurately account for communication time costs and total circuit duration. While hardware supports a limited gate set, optimal control procedures can permit us to synthesize any unitary. But, we limit this to a small gate set on bounded numbers of qudits to limit classical overhead. Instead, we can interpolate to

Qudit Levels	**Interaction Time (ns)**	**Swap Time (ns)**
1	30	-
2	50	-
3	50	-
4	50	-
0, 1*	150	600
0, 2	500	1200
0, 3	500	1500
0, 4	600	1800
1, 1	500	900
1, 2	500	1200
1, 3	500	1500
1, 4*	600	1800
2, 2*	675	2950
2, 3	850	5000
2, 4*	1025	7050
3, 3	850	5000
3, 4*	1025	7050
4, 4*	1200	7500

Table 4.1: Times used for various gates across different levels of qudits. An asterisk indicates an interpolated value.

predict other gate durations or produce pulses for gates in the decomposition. As it is not the topic of this work, please see [83, 84] for more information. Gate times for this work are listed in Table 4.1. The times for qubit-qubit interactions and swaps, and single qubit interactions are known from gate times from devices such as IBM's hardware. Times for qubit-quqart, quqart-ququart, and single ququart interactions and swaps were found via optimal control. Gate durations for qutrits and ququints were found via a linear interpolation between these points. The exact times for any gate also depends on the exact Hamiltonian used to model the underlying device or other physical restrictions. The numbers obtained here are from a standard superconducting Hamiltonian, as would be found for IBM's hardware [2].

As seen in table 4.1, as the radix increases, so does the time required to perform an operation on each qudit. As the highest energy level increases, the pulses required to achieved the desired result becomes more complex as it needs to satisfy more constraints and transitions, elongating the necessary pulse duration. There are several examples of how this could be achieved physically [62]. These increased radix devices also have higher rates of decoherence, but if the circuit depth and communication this will be balanced by a shorter circuit duration. While this section does not focus on other architectures, the presented circuits would be valid, but would a new set of control experiments [87]. However, we would expect similar scaling on these architectures due to the higher level of control required.

4.4.3 Adapting Compilation to Higher Radix Devices

Since many qudits will be in higher-level states, we must use qudit SWAP gates to effectively route the circuit on a device. While the same challenges from the generic mapping and routing algorithm still apply, these qudit SWAPs present a new wrinkle. Not all SWAPs have the same time costs, so one SWAP could be more detrimental in terms of circuit duration that two SWAPs between lower radix qudits. Additionally, since higher radix qudits have lower coherence times, longer circuit durations are detrimental to the success rate of circuit execution.

To adapt our generic compiler algorithm to this structure, we develop a dynamic distance function that uses the time to perform a SWAP instead of using simple distance in the edge weight. This is done by maintaining a collection of time values in the routing algorithm and mapping between each qudit and its maximum possible radix at a given time. We can we use these two values to create a weighted connectivity graph, as seen in Figure 4.2. We then perform the disruption calculation as we did in the original algorithm with this new distance function, only examining qubits that move closer together.

4.4.4 Evaluation Benchmarks

We focus on the Generalized Toffoli, or N-Controlled X gate as our benchmark. This circuit is easily generalized to higher level quantum systems with similar constructions at each level.

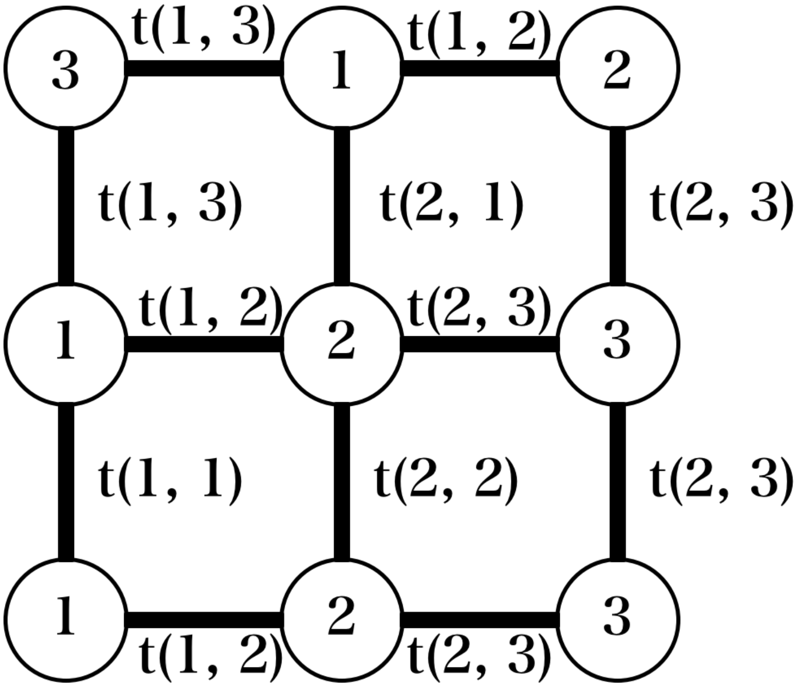


Figure 4.2: A representation of the dynamic map used for mapping and routing when routing quantum devices in higher, mixed radix configurations. Each edge is weighted with the different time it will take to perform a SWAP along that edge.

At the qubit level, this gate can be implemented with any number of ancilla qubits [9]. But, to achieve this circuit in logarithmic depth we need to use $n - 2$ ancilla for n number of controls, significantly reducing the amount of usable space available to execute a quantum circuit. An example of this circuit for 5 controls with 3 ancilla qubits is in Figure 4.3.

An ancilla-free log depth version of the generalized Toffoli has also been developed for qutrit capable quantum systems [42]. By using intermediate qutrits, this circuit uses 1 and 2 controlled $+1$ gates to reduce the number of qubits used and number of gates used. The set of controls are treated as nodes in a tree, recursively stepping down the child nodes in the tree to create the circuit. The child nodes target their parent node with a controlled qudit Toffoli gate which could increment or decrease the state of a qubit rather than performing an X gate. In the case of the leaf nodes, we use a 1 controlled qudit Toffoli gates. Then, any internal nodes use a 2 control since their states will have been increased from 1 to 2 if the leaf qubits were in the 1 state. Finally, the qubit representing the root of the tree performs

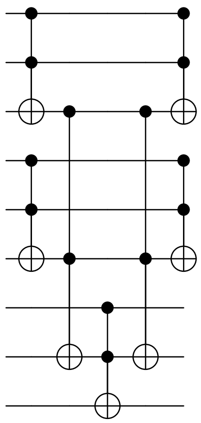


Figure 4.3: A logarithmic depth Generalized Toffoli circuit using qubits with 5 controls and 4 ancilla.

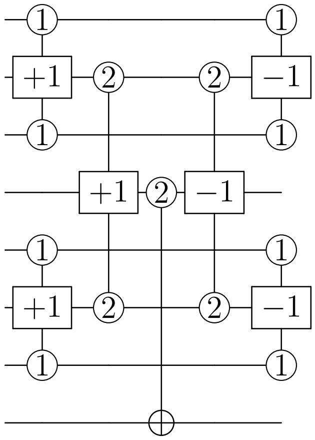


Figure 4.4: A logarithmic depth Generalized Toffoli Circuit using qutrits with 7 controls.

a 2 controlled X gate on the target qubit. We then reverse the controlled qudit Toffoli gates to return the input qutrit to their original, non-elevated state. An example of this circuit with 7 controls is seen in Figure 4.4.

We expand on this idea by constructing generalized Toffoli circuits that take advantage of even higher level quantum systems. We use the same tree-like framework developed for qutrit circuits, but can use the additional computational space afforded to us by these systems. Specifically, we use this extra space to remove the use of the gates involving three qudits, replacing them with two gates that use only two qudits each. While this may seem like an increase in gate count, the three count qudit gates is decomposed into many more one and two qudit gates. For qudit Toffoli gates, 6 two qudit gates and 8 single qudit gates, which is significantly higher. In the ququart case, we use each leaf control to target its parent qudit with a +1 gate. If the controls are in the 1 state, the target will end in the 3 state after being targeted by two controlled +1 gates. Then the internal controls are 3 controlled +1 gates. Similarly, the final controlled X gate becomes a 3 controlled X gate. An example of this can be seen in Figure 4.5.

A similar strategy can be used for any n level system. We split create the control tree into a tree with $n-2$ children rather than the original 2 children in the qutrit and ququart case.

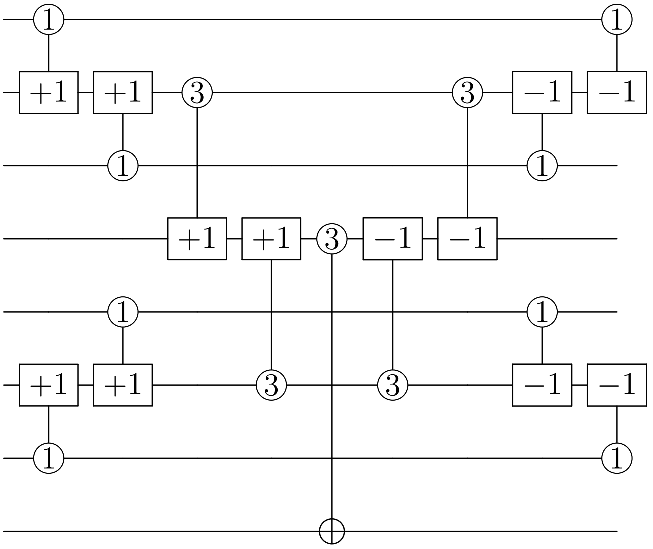


Figure 4.5: A logarithmic depth Generalized Toffoli Circuit using ququarts with 7 controls.

An example of this expansion into a generalized Toffoli gate for 5 level qudits (ququints) can be seen in Figure 4.6.

4.4.5 Results

In this work we observe the changes in gate count, depth, duration, and space time product of a given circuit as we adjust our benchmark to utilize the higher radix levels of a quantum system, up to ququint level devices for the generalized Toffoli gate. Both gate count and depth of a circuit on their own are helpful metrics to compare the runtime of two different quantum circuits. However, as mentioned, different gates have different lengths, and may use more qubits through ancilla. So, it is necessary to examine the duration of the circuit as well. The product of the duration and number of qubits is also used as the space-time product. Minimizing this metric indicates an increase efficiency of the compiled circuit on a given device.

When measuring the time from these circuits, we construct a directed graph with nodes representing the operations, and edges representing dependencies based on qudits used. We are able to label these edges with the time it will take based on the current levels of the

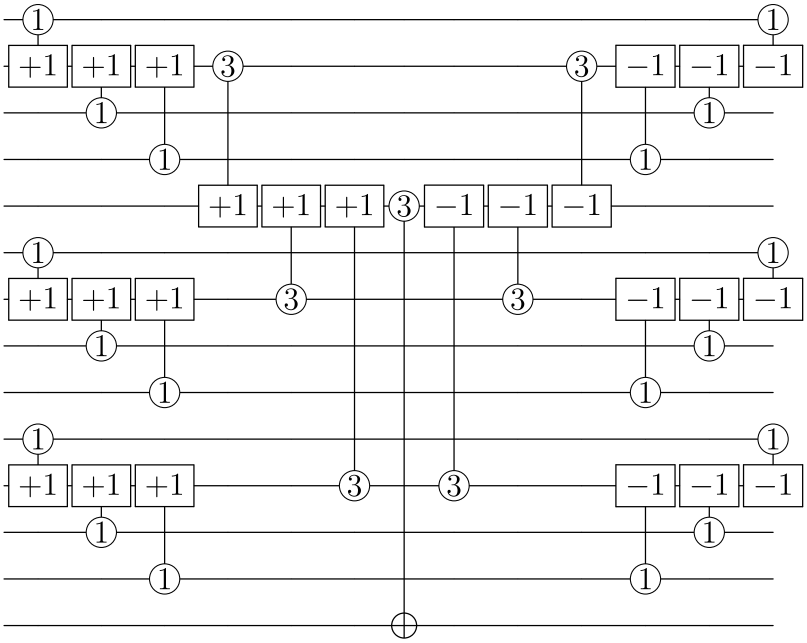


Figure 4.6: A logarithmic depth Generalized Toffoli Circuit using ququints with 10 controls.

qudits at that time. This allows us to find the longest path through the graph, giving us the duration of the critical path, and the duration of the overall circuit. We examine each circuit on a 12 qudit by 12 qudit grid architecture. This provides a middle ground between the structure current quantum architectures, and devices with higher connectivity. It also is a large enough architecture to examine the scalability of a circuit as they grow in size.

Benefits of Qutrits

Previous examinations of the intermediate qutrit generalized Toffoli gate did not examine how it may be affected by routing the circuit on an limited connectivity architecture. In Figure 4.7, we see that the depth of the log-depth qutrit generalized Toffoli gate is greater than the depth of the qubit generalized Toffoli gate, despite the fact that it requires more Toffoli gates to implement the qubit gate. Once SWAP gates are integrated into the circuit via the qudit compilation framework, in Figure 4.8, the depth of the qutrit generalized Toffoli gate is significantly lower than the qubit generalized Toffoli gate. However, referring

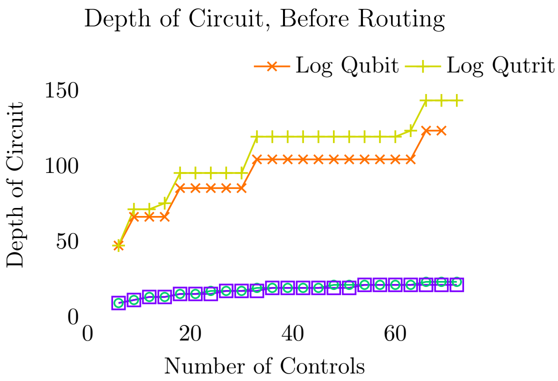


Figure 4.7: The depth of the Generalized Toffoli gates at different levels of qudits before communication gates are added to the circuit.

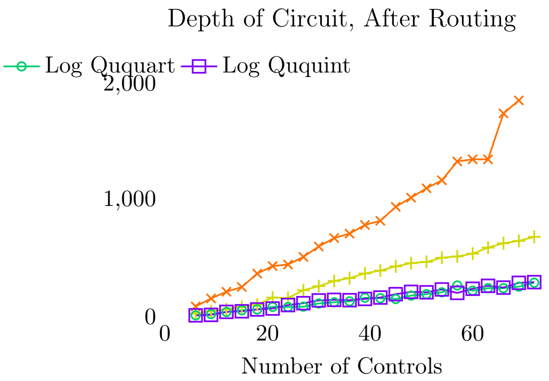


Figure 4.8: The depth of the Generalized Toffoli gates at different levels of qudits after communication gates are added to the circuit.

to Table 4.1, the higher the level of the operations, the longer the times. Upon measuring the duration of a circuit after inserting SWAPs, seen in Figure 4.9 we see that at smaller numbers of controls, qutrit circuit duration is much lower than qubit circuit duration, but as the number of controls increases, these values reverse. Where the qutrit circuit was half the duration of the qubit circuit from 5 to 15 controls, they are 1.5 times the duration from 35 controls onwards. The extra time required by the SWAP gates exceeds the benefits gained from using fewer gates.

However, the construction of the qutrit circuit does not require any ancilla. To take this into account, we examine the change in space time product of the qutrit versus qubit generalized Toffoli gates in Figure 4.10. As the qutrit circuit does not require extra qubits, it can achieve a much lower space time product. This is especially evident when the qutrit circuit has a low duration, starting at a ratio of 3 to 1 from qubit to qutrit space time ratio. As the number of controls increases, this ratio converges to 1.7 to 1 space time ratio from qubits to qutrits, which matches the expected constant increase in computational space from qubits to qutrits, $\log_2(3) = 1.58$.

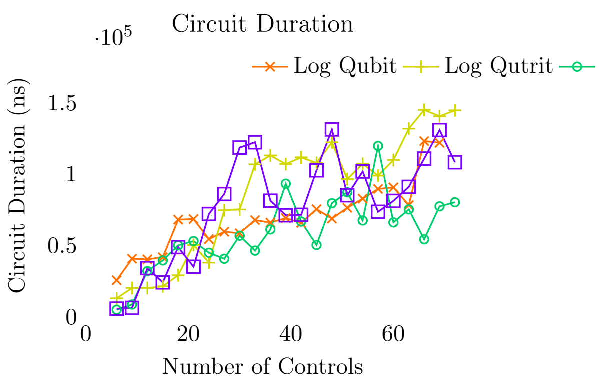


Figure 4.9: The circuit duration of the Generalized Toffoli at different controls for different qudit levels.

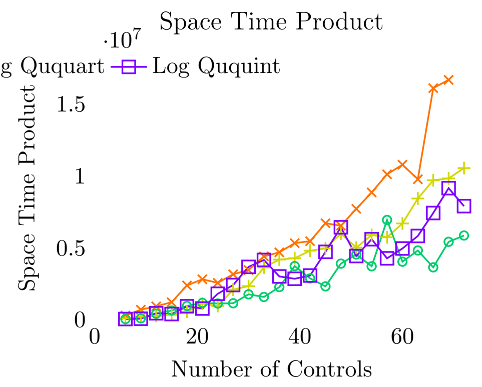


Figure 4.10: The space time product, or number of qudits used by duration, at different controls for different qudit levels.

Benefits of Ququarts over Qutrits

Moving from the qutrit circuit to the ququart circuit does not reduce the number of the qudits. However, since we do not need to decompose any three qudit gates, the circuit requires fewer gates overall. The difference in depth before and after inserting SWAPs can be seen in Figure 4.7 and Figure 4.8. After routing, the depth of the qutrit circuit is 5 to 6 times greater than the ququart circuit, following from the ququart circuit using two two qudit gates, with a depth of two, rather than a Toffoli gate, which uses eight single qudit gates and six two qudit gates and has a depth of 11.

Taking the duration of these ququart gates into account in Figure 4.9, the reduction in gate count and depth generally outweighs the increase in time to perform the qudit interactions. Generally, the ququart circuit time will approach the qutrit duration when the number of controls approaches a power of two. Certain numbers of controls are not conducive to the tree-like structure, and will result in an excess number of gates. When we examine the space time product, ququarts more efficiently utilizes the architecture than both qubits and qutrits. In fact, the qubit to ququart space time product ratio is an average of 2.3 after the initial few increases in controls, which is inline with the expected increase of computational space from qubits to ququarts, $\log_2(4) = 2$. From qutrits to ququarts the expected increase would be $\log_3(4) = 1.9$, and we find the ratio in space time product from qutrits to ququarts to be 1.43, an increase in the efficiency of the use of the device.

Diminishing Benefits of Higher Levels

When we move into higher level systems, specifically levels 5, 6 and 7, we do not see the benefits of using fewer qubits. Since these generalized Toffoli gates have similar constructions to the ququart gate, and do not have the benefits of gate reduction, and do not see significant decreases in the number of gates or depth of the circuit. As seen in Figure 4.7 and Figure 4.8, the ququint depth is the same as the ququart depth before and after insertion SWAPs for routing. Interactions at the ququint level take longer than at the ququart level. In Figure 4.9 we see that the time for a ququint circuit to execute is generally greater than the time for a ququart circuit to execute. This translates to an increased space time product when compared to ququarts. The benefits afforded by the computation space affording by this level of qudit do not outweigh the cost of the complexity and operations required to reach this level over ququart circuits.

4.4.6 Discussion

This section described a strategy for qudit communication through efficient decomposition of SWAPs, and develop a straightforward compilation pipeline for qudit circuits, using realistic gate times for derived from optimal control. Further, we were able to introduce a generalized Toffoli gate that use intermediate ququarts, ququints, following a structure that can be generalized to any number of qudit levels built on strategies developed for intermediate qutrits. Finally, we compiled these circuits for potential architectures to evaluate the benefits at each level of qudit through examination in the change in gate count, depth, execution time, and space time product.

Increasing the highest possible level of a qudit offers opportunities to improve the utilization of the architecture. From qubits to qutrits we see significant decreases in space time product to due reduction in ancilla use, even with increases in time of both interactions and SWAP gates. This trend continues from qutrits to ququarts, where the extra computational space allows us to use two two qudit gates rather than a three qudit gate, reducing duration of the circuit, and the space time product of the circuit as well and improving the use of the architecture. These advantages show that there are gains to be found by implementing these higher dimensional systems and integrating qudit compilation strategies to improve the usage of near term quantum devices.

By adapting the decomposition strategies with more specific communication gates, and adjusting out general purpose algorithm to be time aware, and not just connectivity distance aware, we can efficiently use higher-radix operations significantly reducing opportunities for gate error while maintaining similar circuit duration. Since gate error is a much greater source of error than coherence error in the long term, this is a clear case of how careful compilation for specific architectural features is able to improve circuit success rate.

4.5 Qompress: Compiling Quantum Circuits Using Partial and Mixed-Radix Operations

4.5.1 Introduction

Higher qudit states can be used in a more general way, compared to intermediate qudits circuits, in order to save computational space. The information of two qubits can be fully *encoded* in a single ququart [7], storing two logical units' worth of information in a single computational unit of higher radix. However, this strategy has been avoided due to several drawbacks that scale with qudit dimension: quadratically increasing logic gate execution time [67], reduced coherence time, and increased difficulty of experimentally applying gates [51, 15]. These are significant disadvantages for a NISQ device that already has limited con-

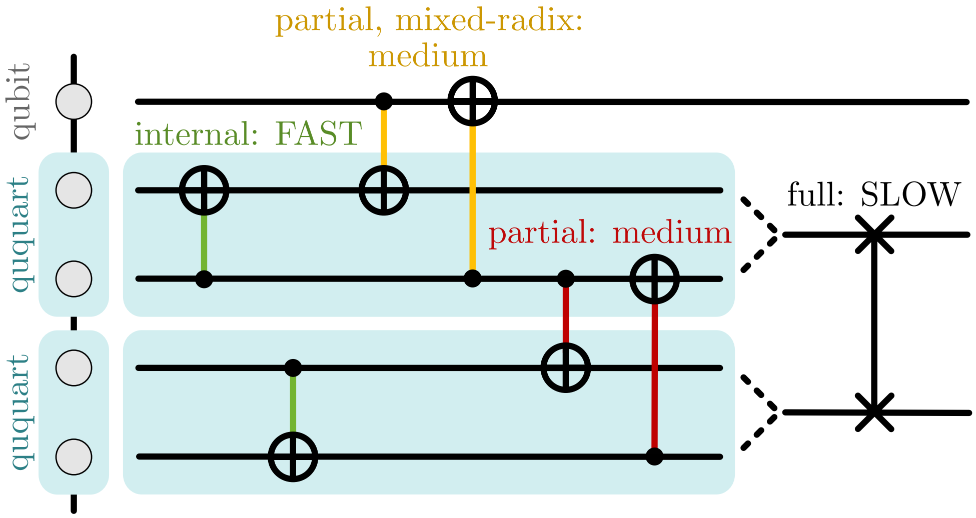


Figure 4.11: Pairs of qubits can be compressed in four-dimensional ququarts and interact with each other internally or through partial operations, enabling novel compilation techniques and space reduction.

nectivity, short qubit lifetimes, and high gate error. However, if these issues can be further mitigated, this general compression method can effectively *double* the available computational space on a quantum computer for any arbitrary circuit.

Previous work has attempted temporary ququart encoding to harness the potential for fast "internal" gates between the two encoded qubits, but had to decode for any operation outside of a specific ququart. In this work, we use quantum optimal control to synthesize high-fidelity, low duration pulse implementations of specific set of gates, that can interact two qubits in the same ququart, a qubit outside of a ququart with a qubit encoded in a ququart, or two different qubits encoded in two different ququarts while leaving others as bare qubits. Each of these gates have varying speeds, and are shown in Figure 4.11.

A mixed-radix paradigm where qubits are directly manipulated within an encoded ququart adds a degree of flexibility not found in the prior work. Inter-ququart operations no longer incur the extra cost of encoding and decoding. Motivated by these new gates, we explore strategies to efficiently generalize partial ququart compression of a circuit to be used with any qubit-based circuit. The partial qubit-ququart and mixed-radix operations inform the design of a compilation pipeline designed to take advantage of the flexibility of our gate set while minimizing the error due to decoherence.

This section proposes several different strategies to find the ideal compression to make the best use of the newly-found ququart compilation methods. Specifically, this section adapts the target gate set, pulse generation, mapping and routing sections of the compiler:

- Develop a library of high-fidelity mixed qubit-ququart operations for partially compressed qubit systems via quantum optimal control for a realistic device Hamiltonian.
- Detail a compiler pipeline that takes advantage of partial ququart operations for mixed-radix computing.
- Evaluate several compression strategies to selectively encode qubits in ququarts using the new compiler pipeline.
- Demonstrate simulated increases in gate fidelity of more than 50% over standard qubit-only compilation across various quantum architectures, as well as up to 2x increased qubit capacity, while accounting for the increased coherence errors associated with operation on ququarts.

4.5.2 Compression and Gate Set

We can redefine the higher-radix gate set to more natively represent traditional qubit operations. To do this, we follow a qubit to ququart compression scheme inspired by [7]. Encoding the state of two qubits $|q_0q_1\rangle$ into a single ququart state promotes one of the qubits q_0 to a ququart while transforming the other qubit q_1 to an ancilla in the ground state $|0\rangle$. While this is referred to as a compression, we do not lose any data. It is the full representation of two qubits, in a single physical unit. We define the encoding gate ENC as

$$\text{ENC} = \begin{cases} |0\rangle\,|0\rangle \rightarrow |0\rangle\,|0\rangle \\ |0\rangle\,|1\rangle \rightarrow |1\rangle\,|0\rangle \\ |1\rangle\,|0\rangle \rightarrow |2\rangle\,|0\rangle \\ |1\rangle\,|1\rangle \rightarrow |3\rangle\,|0\rangle \end{cases} \tag{4.2}$$

This gate defines the encoding scheme; for example, the $|2\rangle$ ququart state represents the $|10\rangle$ qubit-qubit state, and we can perform logical operations on the ququart states as if they are the states of two qubits. We note that, since we do not expect q_0 or q_1 to be in a ququart state prior to the encoding operation, the extension of ENC to a full ququart-ququart unitary gate is arbitrary.

Additionally, to measure a ququart, we simply use the mapping in reverse to determine the state of the two encoded qubits from the state of the ququart. In a physically realized system, we are not able to measure one encoded qubit without simultaneously measuring the other, unless the qubits are first decoded.

Gate Set

Under this encoding scheme, qubit-equivalent operations can be derived for ququarts that encode two qubits, some of which are shown in Figure 4.12.

For example, X gates acting on the encoded qubit state $|q_0 q_1\rangle$ correspond to the ququart operators $\text{X}^0 \sim \text{X} \otimes \mathbb{1}$ (which switches the population of $|0\rangle$ with $|2\rangle$, as well as $|1\rangle$ with $|3\rangle$) and $\text{X}^1 \sim \mathbb{1} \otimes \text{X}$. These gates can also be executed in parallel by applying a ququart gate $\text{X}^{0,1} \sim \text{X} \otimes \text{X}$.

Any two-qubit gate acting on qubits encoded in the same ququart can be expressed as a single-ququart gate. For example, an *internal* SWAP gate (SWAP^{in}) corresponds to a

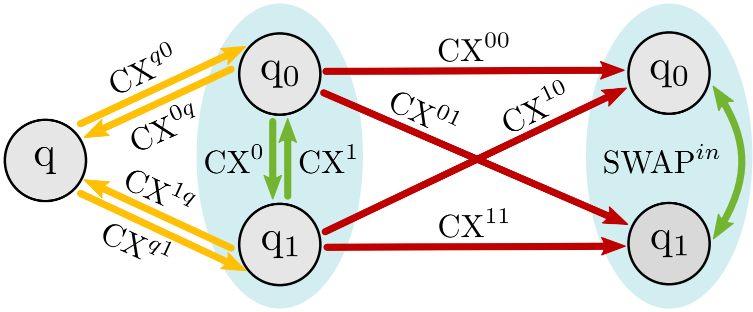


Figure 4.12: Two qubits q_0 and q_1 can be encoded into a ququart (blue oval) and interact with each other internally (green), with a bare qubit q outside (yellow), or with encoded qubits in a different ququart (red). CX arrows point from control qubit to target qubit. Along each CX link, corresponding SWAP gates are defined with the same superscripts/subscripts (not shown).

simple X_{12} operation that exchanges populations of the ququart $|1\rangle$ and $|2\rangle$ states. Similarly, we can define internal CX gates $\{CX^0, CX^1\}$ between the two encoded qubits. On some hardwares, these operations can be done much faster (and with higher fidelity) than the corresponding two-qubit operations. However, in order to compile quantum circuits for a mixed-radix quantum computer using both qubits and ququarts, two-qubit gates between a bare qubit and an encoded qubit as well as between encoded qubits in different ququarts are necessary. To this end we extend our gate set by a variety of *partial* CX- and SWAP-like gates acting on various qubit pairs as shown in Figure 4.12. We define four partial CX gates $\{CX^{q0}, CX^{q1}, CX^{0q}, CX^{1q}\}$ and two partial SWAP gates $\{SWAP^{q0}, SWAP^{q1}\}$ that realize the respective two-qubit gate between a bare qubit and an encoded qubit. Similarly, four partial CX gates $\{CX^{00}, CX^{01}, CX^{10}, CX^{11}\}$ and three partial SWAP gates $\{SWAP^{00}$, $SWAP^{01}$, $SWAP^{11}\}$. We focus only on unique gates, $SWAP^{01}$ and $SWAP^{10}$ are equivalent. can manipulate the states of encoded qubits in different ququarts. We further include the full ququart-ququart $SWAP_4$ gate to enable routing of ququarts.

4.5.3 Synthesizing Mixed-Radix and Full-Ququart Gates

Qudit logic gates are typically assumed to require long durations to implement, due to the increased complexity of the Hilbert spaces involved [67]. This is a problem for several reasons: First, longer-duration gates lead to overall longer-duration circuits; second, higher qudit states have shorter decoherence times [15], amplifying the negative effect of these longer circuits. However, by explicitly synthesizing control pulses for higher-radix gates, we find gate durations that scale efficiently enough to provide overall circuit benefits when combined with a tailored compiler.

We obtain durations of gates in our system by explicitly optimizing high-fidelity control pulses for superconducting hardware. We model two-qudit subsystems with two weakly coupled, anharmonic transmons [61] with total Hamiltonian

$$\begin{aligned} H(t) = & \sum_{k=1}^{2} \left[\omega_k a_k^\dagger a_k + \frac{\xi_k}{2} a_k^\dagger a_k^\dagger a_k a_k \right] + J\Big(a_1^\dagger a_2 + a_2^\dagger a_1 \Big) \\ & + \sum_{k=1}^{2} f_k(t) \Big(a_k + a_k^\dagger \Big), \end{aligned} \tag{4.3}$$

where the first two terms comprise the drift Hamiltonian and the last describes the effect of the control fields $f_k(t)$ applied to the transmons. We choose realistic physical parameters inspired by [94]: The 0-1 transition frequencies of the transmons are $\omega_1/2\pi = 4.914\,\mathrm{GHz}$ and $\omega_2/2\pi = 5.114\,\mathrm{GHz}$, and both transmons have the same anharmonicity $\xi_1/2\pi = \xi_2/2\pi = -330\,\mathrm{MHz}$. They are effectively coupled with $J/2\pi = 3.8\,\mathrm{MHz}$. For single-qudit gates we reduce this model to $k = 1$ and remove the J coupling term. We restrict the maximum amplitude of the control fields to $f_{\mathrm{max}} = 45\,\mathrm{MHz}$ to constrain potential leakage into guard states.

Optimizing Pulses

We aim to find control pulses of shortest *duration* for each gate. As real quantum systems are limited by noise effects such as decoherence, this allows us to investigate the expected cost of using these gates in quantum circuits. Juqbox only allows pulse optimization for a fixed time interval $[0, T]$, therefore we minimize pulse durations T by applying the technique from [93], which involves iterative re-optimization with previous pulse results. This method aims to find the shortest-duration pulse that can execute the desired gate above a minimum fidelity target. In this work, our target fidelity for single-qudit gates is $F = 0.999$ and for two-qudit gates $F = 0.99$.

Figure 4.13 visualizes equivalent state evolutions in two different CX gates: the standard CX_2 between two bare qubits and the partial CX^{0q} controlled by an encoded qubit and targeting an unencoded qubit. In both cases, the control qubit is in state $|1\rangle$ (the ququart state $|3\rangle$ is equivalent to the encoded qubit state $|11\rangle$), causing the target state to flip from $|0\rangle$ to $|1\rangle$. An important observation is the difference in complexity of the state dynamics for the two gates. CX^{0q} operates on twice as many states in the logical subspace as CX_2. This suggests that the difficulty of finding high-fidelity control pulses increases with Hilbert space dimension and explains the variation in gate durations we observe when repeatedly optimizing for complex two-qudit gates.

Qudit Gate Durations

We observe several interesting relationships across gate types. Using an internal CNOT or SWAP instead of its corresponding qubit-qubit operation gives a significant speedup. Additionally, a SWAP between a bare qubit and an encoded qubit (680 or 792 ns) is significantly faster than a SWAP between two encoded qubits (892-964 ns). We emphasize that these relationships are specific to this Hamiltonian, and another system may have different tradeoffs in gate durations. Our goal is to design a compiler that can adapt to different sets of

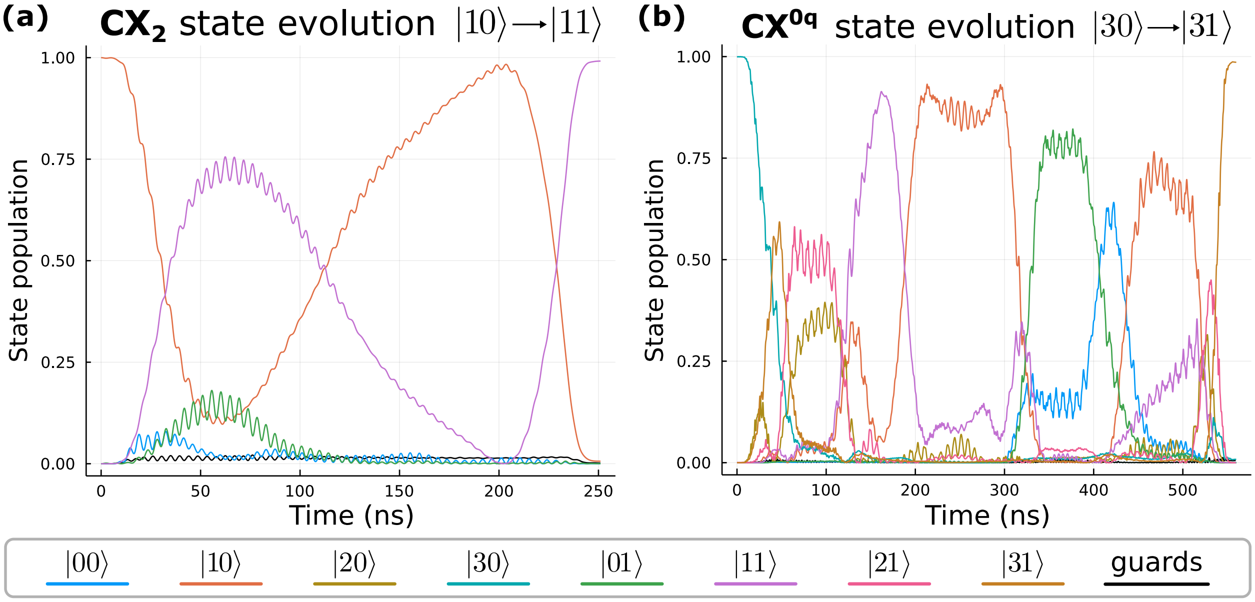


Figure 4.13: Exemplary state evolutions of two CX gates between (a) two bare qubits and (b) a bare qubit and an encoded qubit. (a) The control qubit q_0 is in state $|1\rangle$, hence the state of the target qubit q_1 is flipped. (b) The encoded qubit q_0 inside the ququart controls the CX gate targeting the bare qubit q. The ququart state $|3\rangle$ corresponds to the two-qubit state $|q_0 q_1\rangle = |11\rangle$, so the state of q is flipped.

Table 4.2: Gate durations for one and two-qubit gates synthesized in the qubit-only, mixed-radix and full-ququart environments.

(a) Qudit				**(b) Qubit Only**	
X	35	X^0	87	CX_2	251
X^1	66	$X^{0,1}$	86	CZ_2	236
CX^0	83	CX^1	84	$SWAP_2$	504
$SWAP^{in}$	78				

(c) Mixed-Radix				**(d) Full-Ququart**			
CX^{0q}	560	CX^{q0}	880	CX^{00}	544	CX^{01}	544
CX^{1q}	632	CX^{q1}	812	CX^{10}	700	CX^{11}	700
CZ^{q0}	384	CZ^{q1}	404	CZ^{00}	392	CZ^{01}	488
$SWAP^{q0}$	680	$SWAP^{q1}$	792	CZ^{11}	776	$SWAP^{00}$	916
ENC	608			$SWAP^{01}$	892	$SWAP^{11}$	964

gate durations to determine favorable ququart encodings, without explicitly depending on a specific relationship such as the internal CNOT advantage shown here.

Technical Barriers

The main technical barrier to a physical implementation of this scheme is accurate control of quantum logical units at higher energy levels, not the ability to access them. There are several documented machines that use qutrits [15, 48, 22, 39] and demonstrations of ququart devices [24]; although calibration of these devices is more time-consuming, it is not a fundamental limitation. In superconducting systems, as the energy level increases, increased charge noise causes phase errors and faster state decay, making control more difficult [61, 51, 15].

There is precedent for use of more than two energy levels, e.g. as protection or guard states or to improve measurement and expedite qubit-level gates [61, 63, 41, 69], indicating that it is possible to reliably use higher-energy states. However, at present, we are not aware of experiments using highly-optimized pulses for these energy levels, which would be a potential challenge for the results shown in this work.

As access to calibrated devices with high-dimensional qudits is extremely limited, our methods are currently difficult to physically verify, and the efficiency of controlling these energy states remains an open question. The overhead and difficulty of controlling higher energy states are potential issues and may change the benefits found in the work. We attempt to be as realistic as possible by designing this scheme with a transmon Hamiltonian based on IBM's real hardware, similar to that which has experimentally demonstrated support for qutrit operations [39]. Quantum optimal control has experimentally been proven successful in synthesizing accurate pulses within a certain error threshold for qubit-only gates on similar devices [3], so we are optimistic that the scheme and compilation strategy laid out in this work will be applicable. Additionally, in an ongoing collaboration with the Schuster group [1], we are working to validate our methodology and have to date demonstrated some single ququart gates with high fidelity. While the control pulses that we obtained in this section do not correspond to a real, fully characterized device, we again emphasize that the compilation techniques discussed in the following section are independent of the specific control pulses used, and will adapt to gate durations and error rates different than obtained here. We intend this work to act as a proof-of-concept to demonstrate the potential value of accessing higher-energy qudit states and as a motivating factor to improve ququart characterization and control.

4.5.4 Adapting Compilation for Qubits-on-Ququarts

Compiling a circuit for an architecture which supports full encoding of qubits into ququarts is not fundamentally different from compiling in a qubit-only architecture. However, we must account for higher connectivity and varying efficiency of operations between different pairs of qubits based on their configuration. This includes a new library of communication gates, including the encoding gate, mixed-radix gates, and cross-ququart gates. We extend the original compiler algorithms to account for these new variables and fully utilize the unique set of gates to minimize space requirements while increasing expected circuit success rate and ideally minimizing increased circuit durations.

Representing a Ququart Architecture

Each quantum unit in the system can either be treated as a bare qubit only accessing the lowest two energy levels, or a full ququart which can access the lowest four. We represent this mixed-radix architecture via two graphs. One represents the overall topology of the architecture, where each node represents a single quantum logic unit and edges represent allowed communication between these units. The second represents the logical qubits we will be mapping onto from our original circuit. In this graph, each physical unit is treated as if it is a ququart, and is expanded into two qubit nodes that are connected to one another, called an interaction graph. Both qubit nodes in the expanded ququart are connected to each qubit node contained in adjacent ququarts. For example, if a ququart was connected to n other ququarts, each encoded qubit is connected to $2n + 1$ other encoded qubits. In the secondary qubit graph, there are $2V$ nodes and $4E + V$ edges, where V and E are the number of physical logic units and number of interacting pairs of units, respectively. This expansion is shown in Figure 4.12. Edges are annotated with a set of execution times and error rates for CX gates and SWAP gates, which differ depending on whether either unit is a bare qubit or encoded within a ququart.

Extending Mapping, Routing, and Scheduling

In our proposed mixed-radix system, not all communication is equivalent, or even similar, in cost. A CX gate between a qubit and ququart is lower fidelity and has a longer duration than a CX gate between encoded qubits inside the same ququart. Therefore, our compilation schemes aim to reduce error and circuit duration by taking advantage of gates unique to ququart systems.

The goal of mapping is still to place frequently-interacting qubits close to one another on the device. Since the ququart architecture is modeled as a set of qubits with different connections, we can easily extend mapping strategies designed for qubit-only architectures to ququart-based architectures. First, we find an interaction weight between each pair of qubits in the original circuit. We use the original weighting function to determine the interaciton between all of the logical qubits. We find the center-most ququart in the architecture and place this qubit in the first encoding position. For each unmapped qubit, we compute the greatest sum of weights to the already placed qubits and score each potential placement by how strongly it interacts with mapped qubits and its distance from them.

In our mixed-radix system, we only consider the second encoding location in ququarts if the first encoding location has already been mapped to. In the original mapping function, or when the mixed-radix system is in a qubit only confoguration, distance can be a simple shortest path calculation, with edges weighted by the fidelity of the connection. We do the same for ququarts, but with dynamic weighting based on the current encoding of qubits. We represent the current duration of gate g at connection (i, j) as $T(i, j, g)$ and the fidelity as $F(i, j, g)$. So, for a gate at a given connection, the probability of success is $S(i, j, g) = F(i, j, g)e^{-T(i,j,g)/T_{1,i}}e^{-T(i,j,g)/T_{1,j}}$, where T_1 is the decoherence time for a qubit or a ququart, whichever is being used. This is a common metric of success for a quantum gate as used in [102], and is similar to that used in Section 3.3. The aggregate probability of success for a given path P_n of length n is then modeled as:

$$\begin{aligned} S(P) = &-\log(S(P_{n-1}, P_n, \mathrm{CX})) \\ &+\sum_{i=1}^{n-2} [-\log(S(P_i, P_{i+1})), \mathrm{SWAP}))] \end{aligned} \tag{4.4}$$

This calculation allows us to account for errors introduced by both the decoherence time and the number of gates. We repeat this process iteratively until every program qubit has been mapped to a hardware location represented in the extended ququart graph.

In our system, qubits are still tracked individually even if they are both stored in the same physical object. Therefore, routing for a mixed-radix device is still performed at the qubit level. Any qubit routing strategy, such as lookahead strategies [113, 54, 4], can be translated directly to encoded qubits in ququarts through routing based on the logical qubit architecture graph rather than the ququart-level graph. The main goal is to disrupt the current mapping as little as possible as qubits that need to interact are moved closer to one another. We choose the candidate location which disrupts the current state of the circuit the least and moves the qubits closer together based on the probability of success laid out in Equation (4.4).

However, we do place some constraints on qubit movement. First, we do not encode new ququarts during routing. Second, we avoid swapping "through" ququarts when possible. Ququart operations remain expensive, and pairings determined during mapping are often beneficial to reducing execution costs. Both of these will incur extra costs. Additionally, by placing these restrictions on the compiler, we ensure that there are fewer changes in the maximum energy level of the computational units, we can cache the calculated distances, significantly reducing the amount of classical computation required.

Circuits using fully encoded ququarts require more serialization than an analogous qubit-only circuit. If, at a given time, each qubit in a fully encoded ququart is involved in a CX operation, we can no longer execute both gates in parallel and they must be sequenced one after the other. To break this tie and avoid bottlenecks, we select whichever operation is on the longest execution path of remaining qubits. This can also be an issue for two single-qubit gates targeting two qubits in the same encoded ququart. In this case, we combine both operations into one single-ququart gate, as executing one gate acting on a full ququart is less error prone than executing two single-qubit gates.

4.5.5 Qubit Compression Strategies

Poor initial mapping can incur high communication costs later on. Additionally, ququart gates are inherently more time-intensive and can incur serialization, increasing circuit duration and error due to decoherence. Simply extending qubit mapping strategies to the expanded ququart-based graphs will not take any of these factors into account, or take full advantage of the enhanced connectivity of fully encoded ququarts. In this section, we explore various compression strategies to better account for these features.

The true effect of individual compression circuit duration or fidelity cannot be reliably predicted without recompiling the circuit with a prescribed compression. Further, finding the optimal set of compressions is computationally difficult on its own since there are exponentially many possible subsets. To get a sense for an upper bound on circuit quality, we explore an exhaustive, but iterative and greedy, search of qubit compressions.

At each step, we recompile the input circuit with every pair of qubits compressed and choose the option that maximizes the circuit fidelity. Searching every pair, while more complete, is computationally demanding. Circuit duration is defined by the length of the critical path and therefore it may be more advantageous to explore only compressions which affect the critical path. When choosing compressions we prioritize in order compressions which affect: 1) qubits in non-communication gates on the critical path, 2) qubits off the critical path that insert communication operations along the critical path, and 3) qubits off the critical path. In group order, we choose the best compression, if any, and repeat. We compare this ordered selection to an unordered selection in Figure 4.14.

As can be seen in Figure 4.14, it is not the case that the compression of any two qubits will be advantageous to the combined gate success rate or error due to decoherence. There are two classes of advantageous compression. Some compressions are beneficial due to high interactivity between the two encoded qubits, winning from the fast internal gates. An example is the pairing of qubits 6 and 11 in the circuit shown in Figure 4.14(a). These qubits interact very often, so compression increases the probability of success of CX gate by turning it into a single-qudit gate. The second compression type wins by reducing communication, such as the pairing of qubits 4 and 14 in the cylinder graph based circuit. These qubits rarely interact, but this reduces the *diameter* of the circuit, decreasing required SWAP gates. Critical path prioritization and an unordered search of the entire space result in similar success rate gains, but different compressions but depends on the circuit structure.

Examining all potential compressions is quadratic in the number of qubits in the circuit, on top of the relatively high complexities required for mapping and routing. Using the insights found through this method, we build strategies that are able to approximate these benefits.

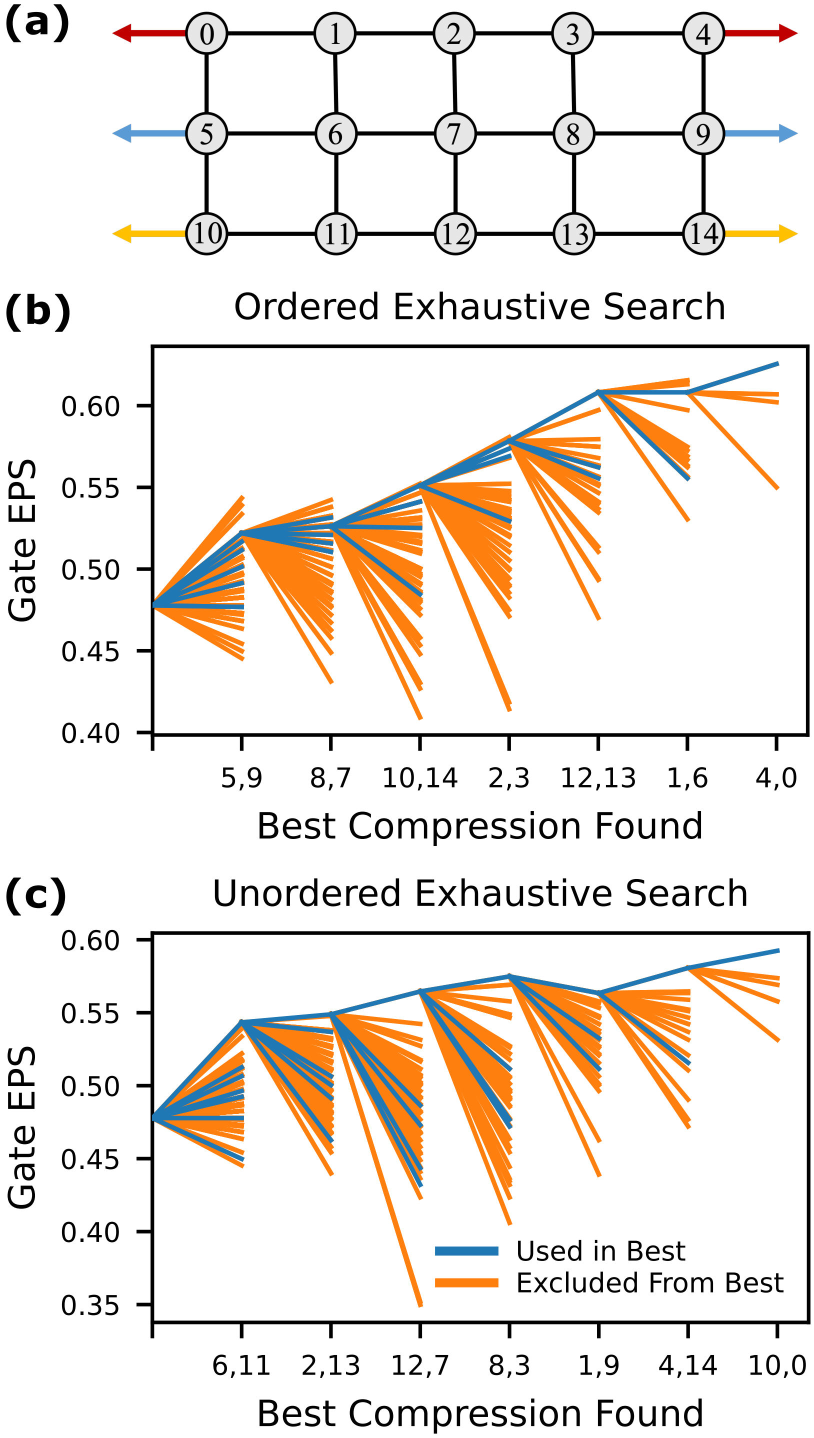


Figure 4.14: An example of an exhaustive search on a cylinder-based interaction graph (a), using a (b) critical path focused selection, and a (c) selection strategy that allows for any pair to be selected.

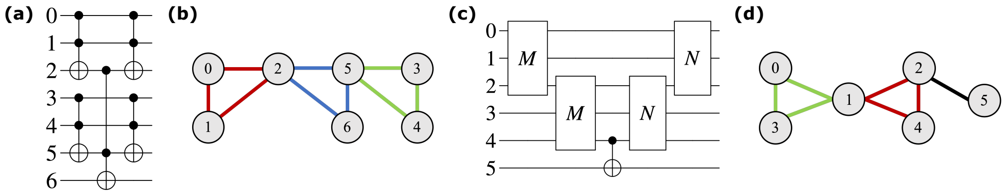


Figure 4.15: (a), (b) Sample generalized Toffoli gate with corresponding interaction graph. (c), (d) Cuccaro adder with interaction graph, M and N are three qubit gate blocks. Some circuits, such as the CNU circuit and Cuccaro adder have clusters of qubits that interact with one another. These clusters can be identified by finding the cycles in the interaction graph. We have identified the cycles here with colored edges.

Extended Qubit Mapping (EQM)

EQM relies entirely on the interactions between the pairs of qubits, and follows the algorithm in Section 4.5.4. There is no explicit pair selection in this method. The qubits are filled greedily, based on the highest weight to the already placed qubits. This strategy clusters qubits that interact often closely together, but will likely preemptively encode two qubits as it sees immediate benefits based on those already placed, focusing on identifying interaction-based compressions. But, it does have classical advantages: EQM is wrapped into the mapping step, and the only additional complexity is due to the expanded qudit graph.

Ring Based (RB)

While frequency of interaction is a good metric for identifying locality, it fails to account for more holistic interaction structures. Figure 4.15 shows that for certain circuits, such as the generalized Toffoli gate and the Cuccaro adder, there is a regular, triangle-based structure in the interaction graph. We can use compressions of these triangles to transform the interaction graph into a line. Looking at Figure 4.15(b), this means compressing pairs 0 and 1, 2 and 6, and 3 and 4. A linear interaction graph is a much more favorable structure to mapping. In fact, for certain configurations, it can be mapped and routed without any SWAPs on any architecture.

We can generalize this triangle compression to any size cycle. If we can find beneficial compressions within a given set of cycles, we may be able to further transform circuit interactions for more effective mappings. It is computationally expensive to find all cycles in an undirected graph, therefore for each qubit, we find the minimum size cycle which includes it. This ensures that each qubit is contained in at least one cycle without finding all cycles. We find the minimum cycle length from this set of cycles, and use it to bound the maximum identifiable cycle size.

After we identify cycles, we find the qubits with the fewest interactions outside their cycle and test compressions with each other qubit in the cycle. For each compression, we collect information such like the number of shared neighbors between qubits, the weight of interaction between the compressed qubits, how often they are interacted on simultaneously, and how many cycles this pair appears in.

We prioritize compressions that maximize the number of internal interactions and the number of connections to other qubits while minimizing the time the encoded qubits are used at the same time, to make full use of both the higher fidelity internal ququart interactions and the new partial ququart operations without introducing serialization. We pick the best compression based on these metrics. The qubit nodes are removed from the interaction graph and replaced with a single node representing the pair, with edges connected to each interacting qubit or pair. Following these adjustments to the interaction graph, we recollect interaction statistics and repeat for other pairings until no other beneficial compressions can be made.

Average Weight per Edge (AWE)

Another method, Average Weight per Edge, makes compressions to maximize the average edge weight of the interaction graph. This method takes advantage of shared interactions to increase potential locality and reducing communication.

We examine each pair of qubits and select the pair that maximizes the average weight per edge once the two qubit node is collapsed into a single node. We repeat until no more compressions can be made, or until there is no pairing that would increase the average weight per edge.

Progressive Pairing (PP)

The final strategy is an extension of EQM, which is more computationally intensive but collects more information. We start by mapping the circuit to a qubit-only architecture. This provides a full picture of how the circuit can be laid out on device, compared to the incremental version provided by EQM. We again examine each potential pairing and compute the estimated fidelity with and without the compression based on changes in distance between interacting pairs without remapping and rerouting. In this scheme there are two choices when compressing two qubits A and B, either A is first or B is first. We select compression adjustment that is expected to increase the fidelity the most.

After selecting the pair that has the greatest increase in estimated overall fidelity, we remap and re-evaluate the circuit with the given pair and repeat the process. We continue this process until we cannot find any pair that will reduce the estimated overall fidelity. While this method scales quadratically, we avoid mapping and routing for each possible pairing by only determining distances. While less scalable than the other methods, Progressive Pairing attempts to replicate the exhaustive search with much less classical overhead.

4.5.6 Evaluation Methods

Architectures

One of the biggest limitations on a quantum architecture is simply the number of qubits available for execution, which constrains many currently feasible programs. Therefore, for different strategies across circuit sizes, we test circuits on regular architectures that are just large enough for the circuit in question. We use a rectangle grid-based mesh as our architecture, with dimensions of $\lceil\sqrt{n}\rceil \times \frac{n}{\lceil\sqrt{n}\rceil}$ where n is the number of qubits in the circuit. Each qubit is connected to four other qubits directly adjacent to it except along the border of the architecture. This construction allows for testing the scalability of our methods and test performance in the edge cases of high architectural usage.

We also examine our methods on the 65-qubit IBM Ithaca heavy-hex topology [2] and a 65-qubit ring architecture. This gives us a sense of how well our methods work on devices with more constrained connectivity.

Gate and Coherence Times Statistics We have shown gate times found via optimal control in Section 4.5.3. We use the target fidelities for these gates as their success rates. Single-qudit gates (regardless of Hilbert dimension) are optimized to a success rate of 99.9% and two-qudit gates to 99% when simulated. The product of the success rate of every gate in the circuit is the Expected Probability of Success (EPS) with respect to gate execution.

To account for increasing noise in higher dimensional systems we consider their coherence times (T_1), after which the qubits are unable to maintain their elevated energy state. For a d-level system, the coherence time is estimated to be $1/d-1$ of the T_1 time for the original two-level system [15]. Critically, because gate durations get longer with corresponding lower T_1 times we capture the increasing susceptibility to error in a mixed-radix system. We use a 163.5 μs qubit T_1 time in our architecture, which gives a 54.5 μs worst case T_1 time when we are in a ququart state. [2]. The product of the probabilities of no decoherence for each qubit, $e^{-t_{qb}/T_{1qb}-t_{qd}/T_{1qd}}$, is the EPS with respect to coherence time. Here t_{qb} is time spent in the qubit state, and t_{qd} is the time spent in the ququart state.

The product of these two statistics gives the overall EPS for the entirety of the circuit, and gives an upper bound for the amount of error found in the circuit. It is the worst case for both gate fidelity and coherence time as it assumes each qubit will be measured and used for the entire duration of the circuit, and uses the worst case coherence time ratio from qubits to qudits. However, it does not take into account the effects of more complicated errors such as crosstalk.

Baselines

We compare against two different baseline strategies, representing the extremes of compilation for qubits on a ququart-supported device.

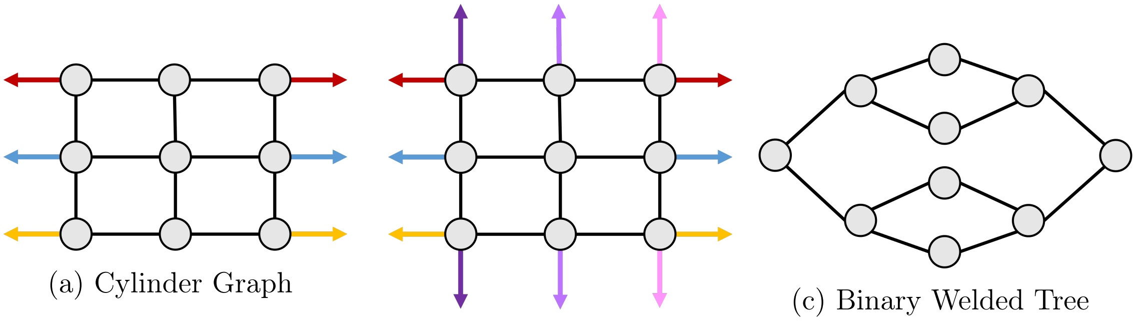
Qubit-Only Compilation One extreme is to never encode any ququarts. In this case, we use EQM but never allow exploration of the second encoded position. These strategies are comparable to other compilation pipelines described in [73].

Full Ququart Pairing with Encoding and Decoding (FQ) There has been discussion of the potential of fast internal ququart operations and use ququarts for general qubit circuit compilation [7]. However, without partial ququart operations, which were assumed to be as expensive as any multi-qudit operation, any external operation required decoding the two qubits, performing an operation, and re-encoding. A drawback of this strategy is that extra space is always required, since a decoding operation requires an ancilla bit to decode into.

We are unaware of an automatic compilation pipeline constructed for this strategy. We use an EQM strategy which allocates extra decoding space next to fully encoded ququarts as the mapping strategy. For routing, we use a similar strategy, but only at the qudit level. This means that we only use full qudit SWAP gates to ensure two qudits are adjacent to on another. Additionally, we must route the empty decoding ancilla next to the neighboring qubits so that we may perform decompression.

Benchmarking

We use a number of different types of circuits to explore many use cases to test the viability of each compression strategy. One set of circuits we explore have localized groups of qubits that



(a) Cylinder Graph

(b) Torus Graph

(c) Binary Welded Tree

Figure 4.16: Examples of graph-based circuit interaction graphs.

interact together. They have no potentially random operations that disrupt the grouping of operations. These include the Cuccaro Adder [28], Generalized Toffoli (CNU) [11], Quantum RAM (QRAM) [43] and Bernstein Vazarani (BV) [14].

We are also interested in certain graph-based interaction structures. We use a QAOA construction [35] that accepts a graph where each node represents a qubit and an edge represents an interaction. For each edge, in a random order, we perform a CX, a Z gate, and another CX gate. These circuits are not necessarily used in practice, but allow for the examination of particular interaction structures which may be relevant in optimization problems. We examine a random graph with edge density 30%, a cylindrical graph shown in Figure 4.16(a), a similar torus structure (Figure 4.16(b)) and a binary welded tree (Figure 4.16(c)).

This set of circuits were selected to be representative of different interaction graphs, and to explore the effects on mostly parallel circuits (Generalized Toffoli) versus mostly serial circuits (Cuccaro Adder and QRAM). Each of these circuits is tested across a range of different sizes to examine how these different strategies scale as the size of the circuit increases.

These compilers were constructed on top of the Qiskit library, version 0.18.3 [4], on Python 3.9 [107]. The benchmarks were run on a machine with Intel(R) Xeon(R) Silver

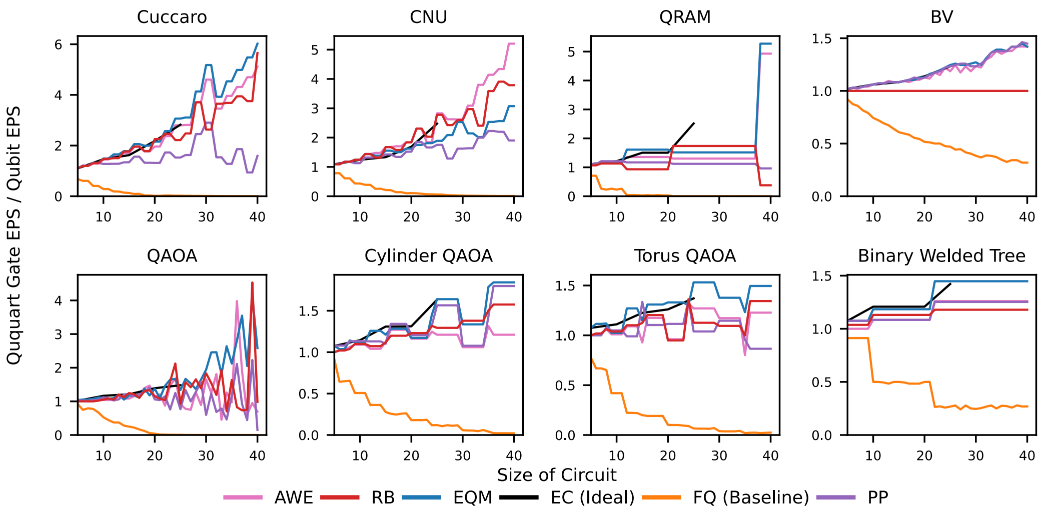


Figure 4.17: Expected Gate Probability of Success for each benchmark. Each color represents a a different compilation strategy, where the black line is the exhaustive solution developed previously, and is the goal. FQ is the previous baseline for generalized ququart computation.

4110 2.10GHz, 132 GB of RAM, on Ubuntu 16.04 LTS.

4.5.7 Results

Two benefits of encoding into ququarts are the decrease in required gates due to reduced communication requirements and the transformation of some two-qubit gates into much faster single-ququart gates. In Figure 4.17, we examine how the gate expected probability of success (EPS) compares to qubit-only compilation for each compression strategy on the same architecture.

Primarily, FQ (orange line, baseline) is consistently worse than our qubit-only baseline. This is expected. Every out-of-pair operation requires more communication, plus the decode and encode steps. As a result, we actually see increases in the number of gates, as well as a decrease in the overall circuit gate fidelity.

Certain compression strategies are more advantageous than others on different circuit constructions. In the more regularly structured circuits, namely CNU and Cuccarro adder, we see the greatest gains in circuit fidelity due to gates from EQM (blue) and RB (red) strategies, with improvements over 50% for both. In fact, these gains match or exceed the EC (black, ideal) case, which requires much more classical computation and is impractical for even moderately sized programs. These circuits have focused interaction on varying sets of qubits over time, as seen in Figure 4.15. The circuit interaction graphs are very easy to flatten into a line by compressing qubits within cycles, eliminating the need for communication. Then only CX operations are needed.

However, the RB strategy is less consistent for BV and QRAM circuits. For BV, this makes sense; there are no cycles to examine in the interaction graph, so no compressions are made. But, QRAM has many cycles. In this case, because the cycles share edges rather than nodes, a compression on one cycle may adversely affect others. Predicting interactions between the cycles is more computationally difficult and was not explored. However, EQM can more closely mimic the interaction graph's connectivity on the architecture, reducing the number of SWAP gates and increasing internal CX interactions.

In the graph-based circuits, where each edge is weighted similarly, no method clearly wins over the other strategies and we only find up to 20% improvements in gate success rate. The most consistent performer is still EQM, which almost never drops below the corresponding qubit compilation success rate. On the other hand, strategies that prioritize communication, such as AWE and PP, are much more inconsistent. These strategies group qubits together that reduce the distance to interact with more qubits. While this might seem to decrease communication, it does not. Instead, qubits are constantly shifted into positions where there is not much locality to exploit. That is, highly-interacting qubits are not necessarily placed close to one another, increasing the chance that communication will be required. These methods significantly reduce the number of internal ququart operations and increase serialization of communication due to both qubits in a pair being required independently for communication. This is shown in Figure 4.18. The overall numbers of gates are similar between EQM and AWE, but EQM uses significantly more internal CX gates (red bars).

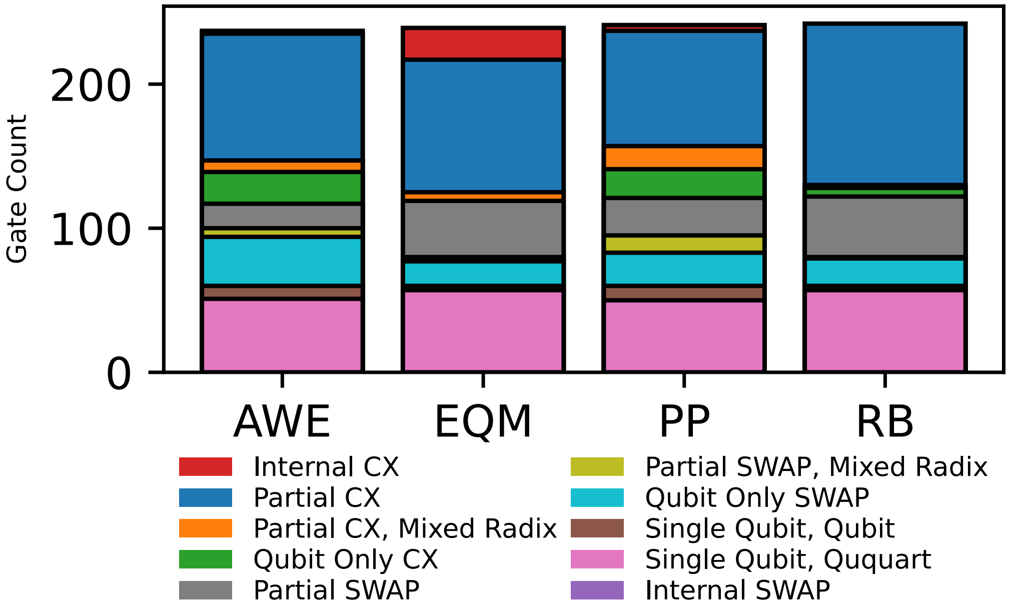


Figure 4.18: The distribution of gate types for different pairing strategies for a 30 qubit Torus QAOA circuit. Each color represents a different "style" of gates. In particular the darker blue represents CX gates between two ququarts, and the red represents an internal CX gate within a single ququart.

AWE and PP tend to use more SWAP gates (grey, green, and cyan bars) and many more partial CX operations (orange and blue bars). EQM enables more success rate improvement by prioritizing higher fidelity multi-qubit operations rather before prioritizing communication reduction.

While communication reduction is important, it is difficult to predict prior to compressions being made and recompilation, and it is more crucial to find the compressions that will increase internal CX count early in the circuit with a general sense of global scope, and leave the router to dynamically adapt to changing communication costs.

Sensitivity to Better Qubit Error It may not be the case that ququart and qubit gates have identical error. As mentioned previously, ququart gates are more difficult to control and may have lower fidelity in a real system. In Figure 4.19, we demonstrate how strategies react

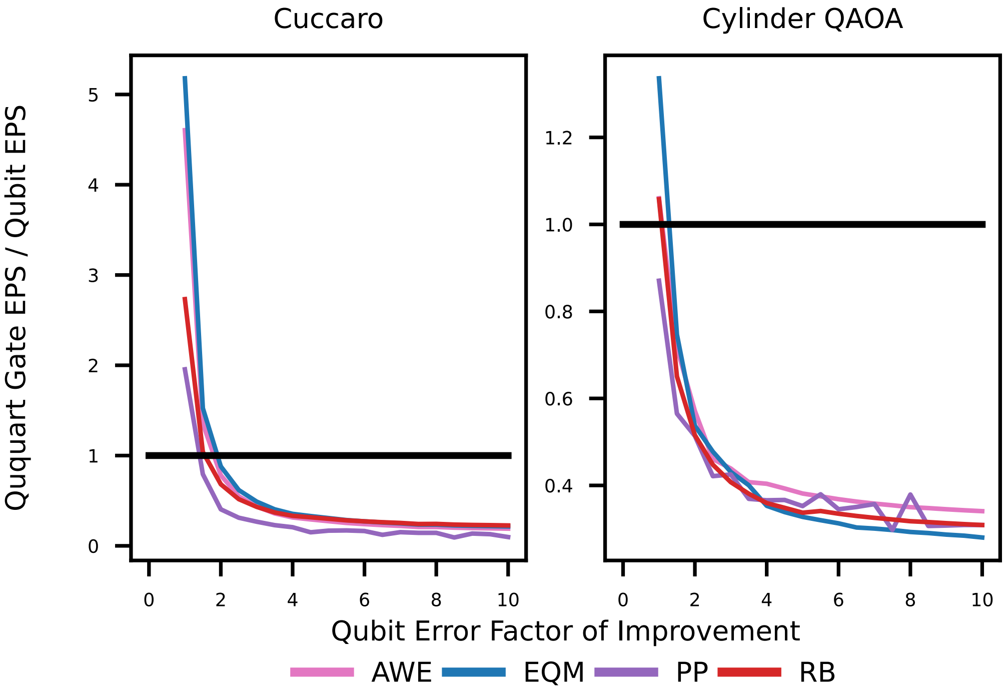


Figure 4.19: Gate Expected Probability of Success as the qubit gate error rate increases and the ququart error gate rate stays constant. The black line represents the crossover point where the error of the qubit only compilation is greater than ququart compilation.

to higher fidelity qubit-only gates for the Cuccaro circuit and for the cylinder QAOA. The strategies maintain their relationship to each other, but, as expected, see diminishing returns as qubit error improves. However, there is some variability in the pairing methods as the qubit error rate improves, as they attempt to preference qubit operations as they become less error prone. Even in these cases, the underlying architecture may be *space limited* meaning we require larger hardware than required by a mixed-radix strategy.

Error Due to Circuit Duration Ququart compressions have a downside. Each compression induces serialization and requires the use of the longer partial ququart gate times. In Figure 4.20, we explore how each compression strategy affects the error due to increasing circuit duration. As the circuit duration increases, a qudit is more likely to decohere due to the significantly worse T_1 time. The probability of an entire system maintaining coherence

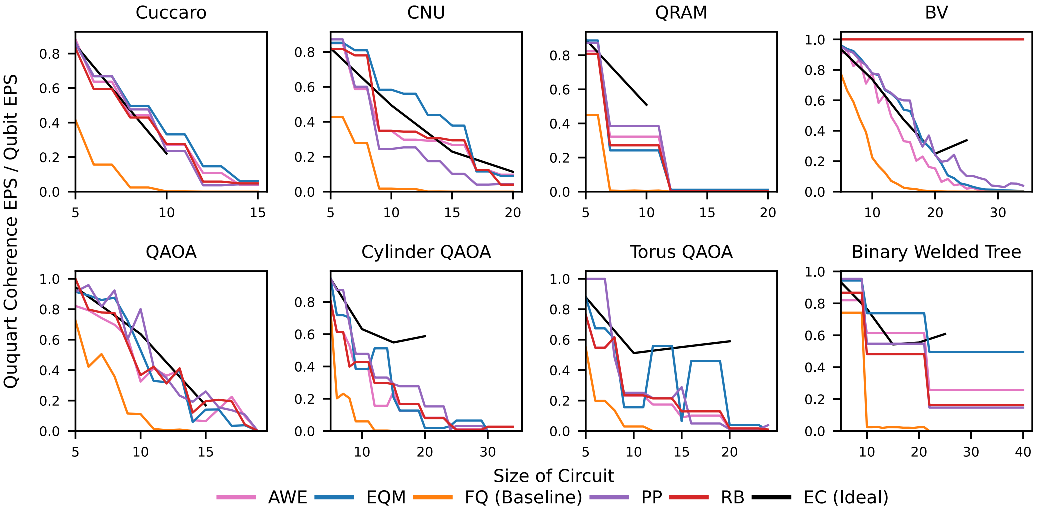


Figure 4.20: Expected Coherence Probability of Success for each benchmark. Each color represents a a different compilation strategy, where the black line is the exhaustive solution, and in theory ideal, developed previously. EC line stops short for computational reasons, requiring many more classical resources. FQ is the previous baseline for generalized ququart computation.

for a mixed-radix circuit is described as the product of $e^{-t_{qb}/T_{1qb}-t_{qd}/T_{1qd}}$ for each qubit. t_{qb} is the time spent in the qubit state and t_{qd} is the time spent in the ququart state, and T_{1x} is the T_1 coherence time for the qudit in the noted state. We first notice that we significantly improve upon the time incurred by FQ; all other compression strategies are able to more effectively mitigate circuit duration increases. Partial SWAP operations are faster than full qudit SWAP operations and prevent extraneous communication. We also find that EQM-based compression generally leads to the best coherence probability of success. Furthermore we observe that, in some cases, the highest gate probability of success does not always match the best coherence probability of success. This is due to the fact that some single-qubit and multi-qubit gates can no longer be performed in parallel due to compression, significantly contributing to the circuit duration. We also notice that we are able to mostly match the duration found through exhaustive search of critical path success rates.

However, a total success rate is the success rate via gate fidelity product times the coherence time success rate, and at current T_1 times (listed in Section 4.5.6) decoherence

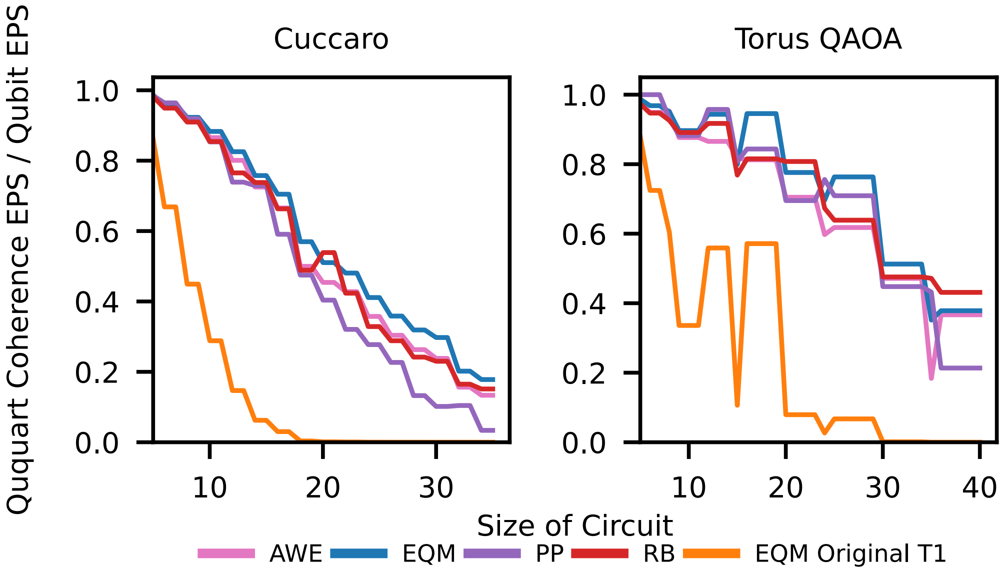


Figure 4.21: Expected Coherence Probability of Success for Cuccaro and Torus QAOA with 10x better ququart and qubit T_1 times.

error outweighs the benefits of success rate increases. We examine the coherence probability of success for a Cuccaro circuit and Torus QAOA in Figure 4.21, with a 10x better T1 time for both qubits and ququarts. While the margin between qubit and ququart circuits improves, it will still outweigh the gate success.

However, the 1 : 3 ratio of T_1 times used in this work is a worst case scenario. Two qubits encoded in a ququart may not be in the maximum $|3\rangle$ state at all times, and will not be subject to the same loss in T_1 time at all points. Additionally, physically realized qutrit devices have seen T_1 relationships that are better than the expected 1/2 reduction [15] and can be specifically designed to enhance the expected decay rates for different energy levels. Using the circuit durations found here, we plot the change in success rate due to circuit duration as the ratio of T_1 time changes in Figure 4.22. As the T_1 for ququarts increases, we find that there is a point, the dashed lines, before the T_1 times are equal where the total success rate is improved with ququarts. In these instances, ququarts would be expected to perform better than qubits. While it is difficult to avoid an increase in circuit duration,

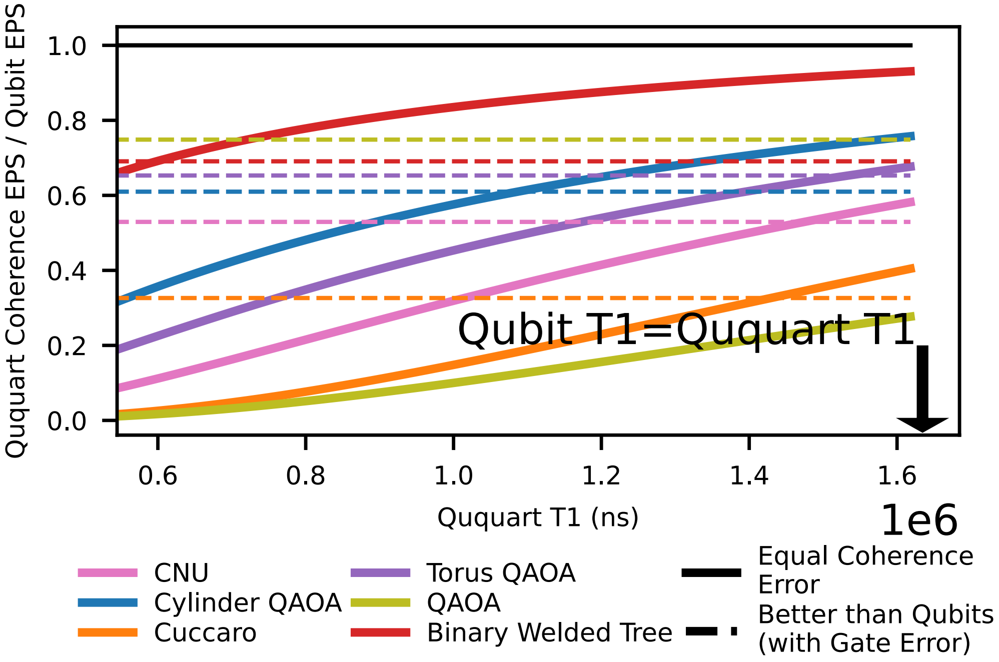


Figure 4.22: Expected Coherence Probability of Success for several 25 qubit benchmarks with 10x better ququart and qubit T_1 times as the ququart time increases T_1 from $1/3$ the qubit T_1 time to the entire qubit T_1 time. Each dashed line represents the point where the coherence success rate no longer outweighs the gate success rate gains for the corresponding circuit.

with enough gains through internal CX gates and SWAP count reduction, encoding qubits into ququarts has the ability to extend what can be computed on a device and perform with better expected probability of success.

Computation on Lower Connectivity Most quantum architectures, especially superconducting devices, have much lower connectivity than the grid architecture we assumed. As described in Section 4.5.4, we also test our methods on the IBM Ithaca topology and a similarly sized ring-based topology. We describe the range of critical path fidelity improvement for some circuits in Figure 4.23. The patterns shown are consistent across all benchmarks. We do not see any significant difference in performance between architectures. This is expected. We use the similar routing steps for both qubits and ququarts, the main difference

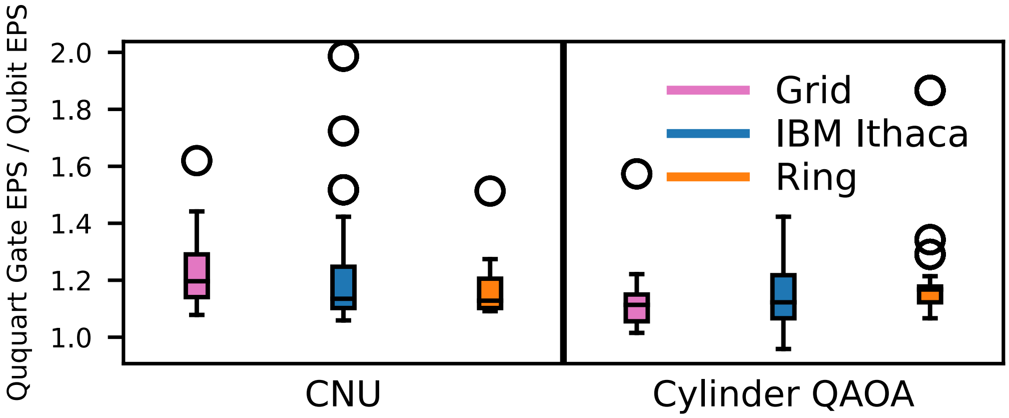


Figure 4.23: Ranges of gate based probability of success for CNU and Cylinder QAOA on three different architectural topologies. This is the combined set of ratios of improvement for circuits sizes 5 to 40.

being that the connectivity is slightly expanded for the ququart routing. Our methods can successfully adapt to different structures with similar effects for each.

4.5.8 Discussion

Architectural size is a huge limitation on useful computation on quantum computers, and many architectural models for quantum architectures have access to higher level states. These higher level states can be used to compress quantum information. In the case of ququarts, two qubits can be fit into a single physical unit in a process we call *compression*. There are difficulties in working with these more complex objects such as shorter coherence times, and longer gate times work against each other to make ququart compilation challenging.

In this section, we adjust several different asepcts of the quanutm compilation pipeline. We explicitly realize a gate set for mixed qubit-ququart systems including partial operations, without the need to encode and decode ququarts. With a gate set designed specifically for ququart computation and communication, we design and evaluate a compiler to mitigate time-intensive ququart-ququart interactions and the increased gate-based communication required to interact two ququarts in past models. Our compiler also exploits higher qubit connectivity and reduces the count of needed physical units to execute programs requiring more qubits than available. By adapting our standard compilation strategies to take enhance the benefits of higher radix computaiton, we are able to mitigate this cost and explore the best way to take advantage of faster operations, such as internal CX gates, enabled by encoding multiple qubits in the same physical unit. We develop several strategies that emphasize the need to prioritize these fast interactions and exploit the locality of certain circuit structures for better gate expected probability of success, and lower increases in circuit duration.

4.6 The Quantum Waltz: Exploiting Mixed-Radix and Full-Ququart Operations for Native Three-Qubit Gates

4.6.1 Introduction

As we've seen, success of quantum algorithms greatly depends on how many error-prone gates are used and the total program duration. In most currently competitive quantum systems, e.g. superconducting systems, trapped ions, and neutral atoms, gates which act on many qubits simultaneously (≥ 3 operands) must be decomposed, contributing to both increased gate counts and increased circuit depth. In this work, we focus primarily on superconducting systems which also suffer from limited connectivity between devices, further exacerbating the decomposition problem.

The significant increase in the number of operations required to perform a calculation strains the device in multiple ways. Many circuits include gates, such as the Toffoli gate, to perform reversible arithmetic calculations; thus, three-qubit operations are common across implementations of quantum algorithms [7, 45, 28, 43]. Finding implementations of these gates without having to reduce them to more elementary gates could be expedient to saving the valuable computational resources of the device.

We've explored several uses of higher-radix computation, intermediate-qudit computa-

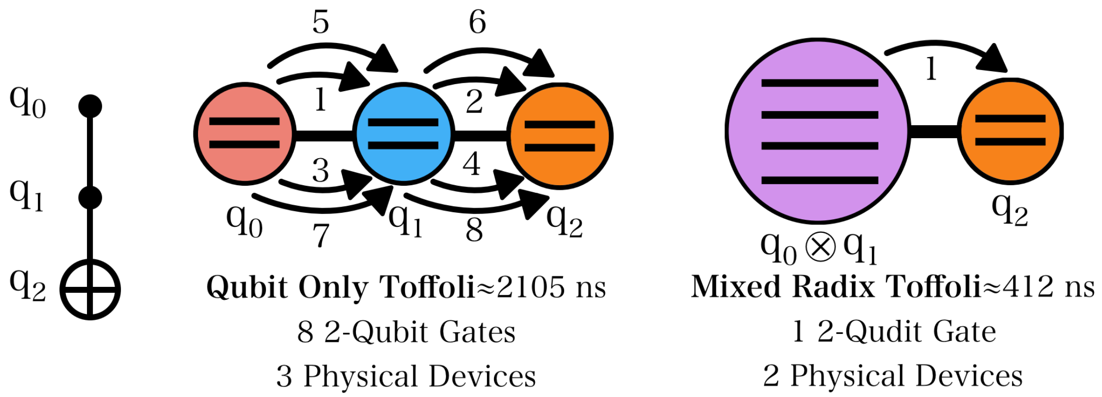


Figure 4.24: A comparison of executing a Toffoli gate on a three-qubit-only system versus a Toffoli gate between a ququart and qubit in a mixed-radix system. In a qubit-only system, we must use a decomposition that uses eight two-qubit gates that can be reduced to one two-qudit gate that has a shorter duration.

tion and qubit compression into ququarts with both the updsides of reduced communication and higher fidelity CX gates on compressed qubits with the downsides of lower coherence times. In this section, we observe the simple fact that one ququart is equivalent to exactly two qubits and therefore the information of two individual qubit devices can instead occupy a single device which has access to four logical states [7]. This has significant advantages on the relative connectivity of the qubit information. By performing this compression, we can access three qubits worth of information by only interacting two physical devices. This type of *mixed-radix* gate (four-level system interacting with an adjacent two-level system) is equivalent to performing a three-qubit gate. Similarly, we can consider two adjacent ququarts which allows us to perform interactions on up to four qubits worth of information by controlling only two physical devices; we call these *full-ququart* gates. This simple observation could remove the need to perform expensive decompositions of three- or four-qubit gates, as visualized in Figure 4.24. We are primarily focused on common three-qubit interactions since they appear more commonly in real applications, unlike four-qubit gates.

We examine using ququarts to dynamically encode and decode gates to perform native three-qubit gates on ququarts on a simulated superconducting device in a compilation pipeline called the Quantum Waltz, a dance done in three-four time. In particular, the major contributions of this section are:

- A library of two- and three-qubit gates performed on mixed-radix and full-ququart schemes, synthesized via quantum optimal control for a realistic Hamiltonian for a superconducting architecture.

- Identifying specific relationships between the controls and targets of three-qubit gates that allow for more efficient execution of mixed-radix and full-ququart gates.

- Detailing a compiler that choreographs three-qubit gates into particular configurations on ququarts for better performance by managing the controls and targets of three qubit gates.

- Simulating a wide variety of benchmarks compiled with different strategies.

- Demonstrating how three-qubit gates on ququarts can achieve a 2x improvements in simulation in a mixed-radix environment and up to 3x fidelity improvements in a full ququart environment, as well as insights into the right situations to implement these gates.

4.6.2 Three Qubit Gates on Ququarts

We use the same compression strategies developed in Section 4.5.2, and use the same pulse generation techniques as well. The gates described here are an extension to a native gate set for ququarts to avoid decomposition of three-qubit gates to longer strings of one- and two-qubit gates.

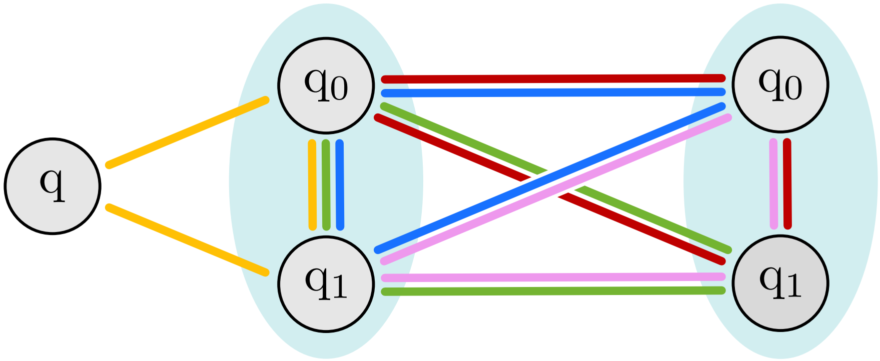


Figure 4.25: Visualization of connectivity advantages in qubit-ququart systems. Encoding qubits in ququarts (lightblue) enables triangle connectivity between triples of qubits, where two of which are encoded in the same ququart and one appears either in a bare qubit or encoded in a neighboring ququart.

Connectivity Advantage

The set of two qubit gates laid out in Section 4.5.2 are enough to universally perform general qubit computation on ququarts [70, 79], but simply compiling to two-qubit gates would not take advantage of the flexibility of this abstraction. When we think about a ququart as two encoded qubits, we virtually increase the connectivity between qubits on our device. An example of this is seen in Figure 4.25. Each of the encoded qubits in a ququart is connected to an adjacent qubit, or both of the encoded qubits in an adjacent ququart. As highlighted by each of the different colors in Figure 4.25, this creates many triangle subgraph relationships between encoded qubits that would not exist otherwise. This is significant, as triangle subgraphs are uncommon in current hardware due to the increased probability of crosstalk as the density of interaction increases and frequencies overlap [72, 32]. But, triangle-based interactions are common in many different circuits that use three-qubit gates to perform arithmetic calculations. Here, we increase the number of virtual connections without increasing number of physical connections to create four virtual triangles, and four interactions between encoded qubits, something that isn't easy on current hardware topologies.

The triangle relationships existing across two physical units gives rise to another important phenomenon. It is not fundamentally harder to interact three or four qubits worth of

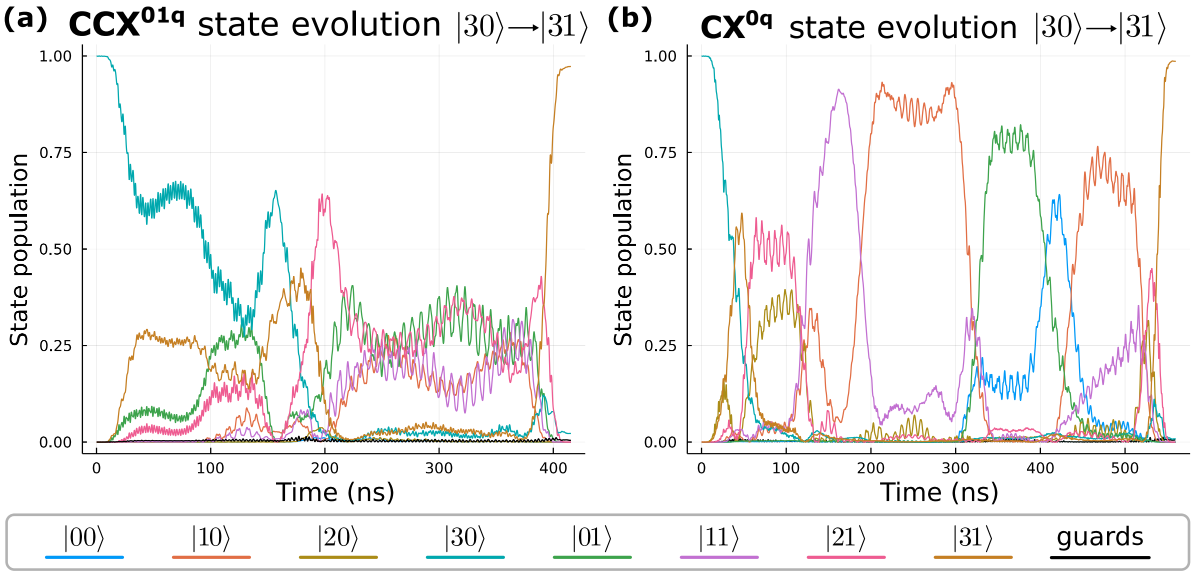


Figure 4.26: Visualization comparing the evolution of a $|3\rangle$-controlled X gate in a mixed-radix environment for a CCX gate in (a) and a CX gate in (b).

information than two qubits worth with a single operation. These gates are equivalent to either mixed-radix or full-ququart gates. For example, if we have a fully encoded ququart next to a bare qubit and perform a Toffoli gate targeting the qubit, it is equivalent to a $|3\rangle$-controlled X gate on the qubit. This is computationally simpler in terms of number of state transitions, than the several $|1\rangle$- and $|3\rangle$-controlled X that would be required in the decomposition and can be seen in the state evolutions in Figure 4.26. This gate implementation gives superconducting qubits more natural access to the native multi-qubit gates.It also avoids decompositions that add a significant number of extra gates.

Pulse Generation

Multi-control Gates Just as a native two-qubit gate on one physical ququart offered significant improvement in both fidelity and execution time as compared to the standard two-qubit gate on two physical qubits, native three-qubit gates on two physical units have the potential to offer a significant improvement in gate fidelity and execution time. In Table 4.3, we show pulse durations for different orientations of the three-qubit Toffoli gate

Table 4.3: Mixed-Radix and Full-Ququart Three-Qubit Gate Durations

(a) Mixed-Radix		**(b) Full-Ququart**			
CCX^{q01}	619	$\text{CCX}^{01,0}$	536	$\text{CCX}^{01,1}$	552
CCX^{1q0}	697	$\text{CCX}^{0,01}$	785	$\text{CCX}^{0,10}$	785
CCX^{01q}	412	$\text{CCX}^{1,10}$	785	$\text{CCX}^{1,01}$	680
CCZ^{01q}	264	$\text{CCZ}^{01,0}$	232	$\text{CCZ}^{01,1}$	310
CSWAP^{01q}	684	$\text{CSWAP}^{01,0}$	680	$\text{CSWAP}^{01,1}$	744
CSWAP^{10q}	762	$\text{CSWAP}^{10,0}$	758	$\text{CSWAP}^{10,1}$	822
CSWAP^{q01}	444	$\text{CSWAP}^{0,01}$	510	$\text{CSWAP}^{1,01}$	432

in both mixed-radix and full-ququart configurations. These gates were synthesized using the same fidelity targets and pulse generation techniques as the two-qubit gates, therefore resulting in an overall higher fidelity than if decomposed with many gates of the same target fidelity. After synthesizing the different configurations of Toffoli gates, we find that there is a substantial difference in the gate duration depending on which qubits are controls and which is the target.

Consider the mixed-radix example where both control qubits are encoded in the same ququart, and the target qubit is in the bare qubit, or CCX^{01q}, seen in Figure 4.27(b). This configuration is about two-thirds the time of the CCX^{0q1}, seen in Figure 4.27(c), where the control qubits are split across the bare qubit and the ququart, while the target is encoded in the ququart as well. The reason for this difference is twofold. The first follows from the two-qubit only gates. Gates which use the ququart as a control and the qubit as a target are generally faster. In this case, the pulse only needs to induce population changes between the $|0\rangle$ and $|1\rangle$ state of the qubit, rather than between the transitions between the pair $|0\rangle$ and $|1\rangle$, and $|2\rangle$ and $|3\rangle$ of the ququart. The second is that the entire ququart acts as the control, only changing the state of the bare qubit if the ququart is in the $|3\rangle$ state. In the split-control case, the ququart must control on both the $|2\rangle$ and $|3\rangle$ state.

The same concept of separation of controls and targets follows for the full-ququart Toffoli gates as well. Regardless of whether the target qubit is in the first or second encoding of the

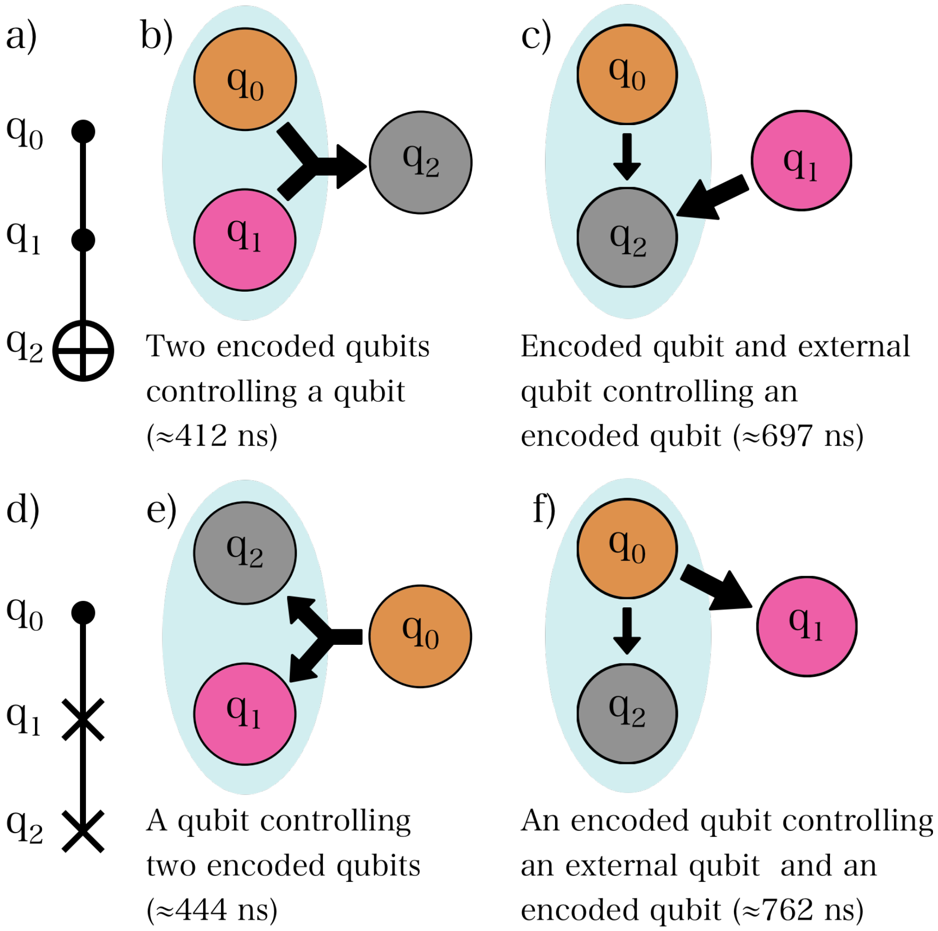

a)
q0
q1
q2
b)
q0
q1
q2
Two encoded qubits controlling a qubit (≈412 ns)
c)
q0
q1
q2
Encoded qubit and external qubit controlling an encoded qubit (≈697 ns)
d)
q0
q1
q2
e)
q2
q0
q1
A qubit controlling two encoded qubits (≈444 ns)
f)
q0
q1
q2
An encoded qubit controlling an external qubit and an encoded qubit (≈762 ns)

Figure 4.27: Examples of two-control and two-target three-qubit gates in a mixed radix environment. a) The Toffoli with two controls q_0 and q_1. b) A configuration where both controls are encoded in the ququart and the target is mapped to a qubit. c) A configuration where the controls are split across the qubit and the ququart and the target is encoded in the ququart. d) The CSWAP gate with two targets q_1 and q_2. e) A configuration where both targets are encoded in the ququart and the control is mapped to the qubit. f) A configuration where the targets are split across the qubit and the ququart and the control is encoded in the ququart.

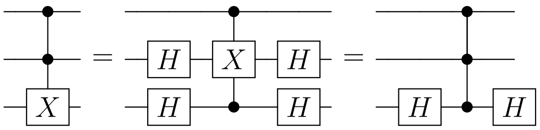


Figure 4.28: Different decompositions for the Toffoli Gate. a) is the base Toffoli circuit. b) is Toffoli circuit with a swapped second control and target from the original. By surrounding the control and the target with Hadamards, we perform the same operation. c) The Toffoli gate constructed from a CCZ gate which can be used as a Toffoli by surrounding the target with Hadamard gates.

ququart, $\mathrm{CCX}^{0,10}$ or $\mathrm{CCX}^{0,11}$, it is substantially faster to keep the controls encoded in the *same* ququart with the target encoded in a separate ququart.

Target-Independent Gates Separating the controls and targets into different devices yields more efficient gate execution; however, compiling circuits to conform to this configuration is unnecessarily constraining. Instead, we consider a situation where all multi-qudit gates are *target-independent* and only affect the global state when all three qubits are in $|1\rangle$. For example, the Toffoli gate, or CCX, is locally equivalent to CCZ which is target-independent, as seen in Figure 4.28.

When pulses are synthesized, CCZ is much more efficient as seen in Table 4.3, remarkably on par with the speed of the qubit only gates. In addition, we only need to define three configurations: $\mathrm{CCZ}^{q,01}, \mathrm{CCZ}^{0,01}, \mathrm{CCZ}^{1,01}$, as opposed to the nine possible CCX configurations, reducing computational overhead. We postulate the short duration of these gates is because CCZ only changes the phase of the entire three-qubit state and does not change the population of any state. A population change for the Toffoli gate is more complicated. This makes the CCZ (and generally target-independent gates) a valuable tool when compiling three-qubit gates. In practice, this removes the need to distinguish targets from controls.

Multi-target Gates We also consider gates that use one control qubit to affect the state of some number of other qubits, for example the CSWAP. With our methods we synthesize gates to the same fidelity targets as before and show their times in Table 4.3. As expected, we find benefits when separating the control qubits from the target qubits as depicted in Figure 4.27(e) versus Figure 4.27(f). When both targets are encoded the same ququart, we limit the possible state changes to be between $|1\rangle$ and $|2\rangle$ in that ququart, rather than needing an expensive mixed-radix or full-ququart based qubit SWAP gate.

In this work we show these gates improve the execution of quantum circuits compared to traditional binary-only strategies. While we are able to perform any configuration of three qubit gates directly in mixed-radix or full-ququarts scenarios, we take care to use best configurations to minimize time in the less stable $|2\rangle$ or $|3\rangle$ states.

4.6.3 Adapting Qubits on Ququarts to Three-Qubit Gates

To execute three-qubit gates (like Toffoli or CCZ), circuit qubits must be routed into a connected subgraph of the interaction graph, e.g. for CCZ(q_0, q_1, q_2) requires $q_0 \sim q_1$ and $q_1 \sim q_2$ but it is not guaranteed that $q_0 \sim q_2$, where $\sim$ defines adjacency. We need to adapt our original strategy to fit within a frameowkr that can best take advantage of routing three-qubit gates together, when they have varying cost based on their configuration between two physical devices.

4.6.4 Using Three-Qubit Gates

Qubit-Only In a qubit-only regime we cannot efficiently execute a native three-qubit gate. In these cases, we use a decomposition into eight CX operations [95]. This decomposition has the flexibility of being target-independent from a compilation standpoint.

This is an expensive compilation, as it requires eight two-qubit gates, and 14 one-qubit gates. But it does not require using the less stable $|2\rangle$ and $|3\rangle$ state. Other, cheaper decompositions of CCX and CCZ require complete connectivity between the operands.

Intermediate Mixed-Radix We also permit *temporary* use the higher energy levels to perform an operation. By performing an encoding gate (ENC) followed by the three-qubit gate and a final decode (ENC†) operation, we get temporary access to the full connectivity needed on a shared physical connection to perform fast three-qubit gates.

As noted, the compiler should opt to encode qubits of similar *type*, i.e. both controls together or both targets together. Let $U(q_0, q_1, q_2)$ be the operation with q_0, q_1 the same (either both controls or both targets). In some cases, encoding is simple because the routing strategy (prior work) results in $A \sim B$ as in 4.27(b). However, it may fail to do this by default and we may have $q_0 \sim q_2$ and $q_1 \sim q_2$ but not $q_0 \sim q_1$ and in 4.27(c).

In this case, we have three options to compile to a favorable configuration. First, we could enforce the ideal relationship through additional gates by adding an additional SWAP(q_0, q_2) (or equivalently SWAP(q_1, q_2)). Second, in the special case where $U =$ Toffoli we can change which pair is the same type with Hadamard gates as in Figure 4.28 to use the most efficient implementation; we call this *re-targeting*. Third, if U permits, we transform U into U' so that q_0, q_1, q_2 are all the same type; for example we transform CCX to CCZ so each operand is a "control." While the additional re-targeting or transformation gates add both error and duration, they enable the shortest duration version of U to be used for an overall net increase in fidelity. In this work, we consider the special cases of $U \in \{CCV | V \in SU(2)\}$, i.e the set of locally equivalent gates to CCX. We leave the generalized case to future work in circuit synthesis. For the multi-target situation there is no easy fix and SWAPs must be inserted.

Full Ququart Many of the compilation strategies laid out for mixed-radix three-qubit gates can be used in full-ququart compilation as well. However, some configurations are extremely unfavorable (minus the target-independent CCZ) for the execution of both two-qubit and three-qubit gates as seen in Table 4.3. To remedy this, we can perform an internal SWAP to exchange the encoding positions first.

The router by default, described previously, does not distinguish control or target. However, when executing three-qubit gates, we ensure only qubits of the same type are encoded if it does not require an extra swap operation.

In a compiler, any of these techniques could be put together to decide the best decomposition for a Toffoli based on the current orientation and placement of the qubits in the mixed-radix or full-ququart system.

Adapting Mapping and Routing

We only need to adapt the mapping and routing algorithms described in Section 4.5.4 where we considered the fidelity differences as a distance metric between different routing paths. In this case, we need to ensure that we are able to generalize to three-qubit gates.

The first step is to decompose the operations in the circuit to native gates supported by the device. As our compiler will need to handle the native execution of three-qubit gates, we decompose to the CX, CCX, CCZ or CSWAP along with a parameterized single-qubit rotation gate based on which gate set and architecture we are studying. This set is enough to support universality, and any gate can be decomposed to some combination of these gates.

Our mapping strategy does not change, and we still only weight each three-qubit gate as if it is one operation. When routing, we track the circuit qubits on the interaction graph and use SWAP gates until the interacting qubits are adjacent. We still attempt to disrupt advantageous qubit layouts as little as possible by using adaptive weights that change as operations are scheduled based on [10]. This strategy attempts to keep qubits interacting in the near future close to one another. To generalize to three-qubit based routing we modify the cost function to a sum over all qubits in the operation, $C(i) = \sum_{j \in Q_o/i} D(i, \varphi^{-1}(n))(d(\varphi(i), \varphi(j)) - d(n, \varphi(j))$ where Q_o is now a set of all operands.

4.6.5 Evaluation

Circuits

We primarily study benchmarks which already include three-qubit gates in their formulation and which can be parameterized by number of qubits to study scalability of our methods

We examine five circuits with different constructions. The first is the Generalized Toffoli (CNU) circuit [9], which flips the state of a target qubit if all the controls are one. This circuit uses exclusively Toffoli gate based decomposition and is highly parallel. The Cuccaro Adder [28] is nearly entirely serialized using $2n + 2$ qubits with a mix of three-, two- and single-qubit gates to add two n-bit numbers. Third is a QRAM circuit which uses primarily CSWAP gates to retrieve data from or move data into a set of qubits [43]. The fourth is a Select circuit, which is a preparation mechanism used in Quantum Phase Estimation (QPE) [68]. It performs a particular Pauli operation on n qubits for each potential 2^m states of m index qubits [6]. For our case, the choice of Pauli string does not affect compilation. To keep the fidelity of circuit simulation within comparable bounds, we only select on two random values rather than all of the potential 2^m values the index qubits could be in. The fifth is a purely synthetic circuit to study relative strength of our architecture on potential distributions of CX versus CCX gates.

Hardware Topology and Error

For this section, we consider the same underlying hardware topology for each comparison point - a 2D mesh. This type of grid architecture has relative density on the upper end of realized superconducting connectivity graphs, reflective of Google's Sycamore chip [5].

This is a rather optimistic connectivity structure, and not fundamentally different from the quantum heavy-hex processors designed by IBM [40]. Here we consider a grid design with dimensions $\lceil\sqrt{n}\rceil \times \frac{n}{\lceil\sqrt{n}\rceil}$ with connections to each qudit adjacent to it on the device.

An important feature of a quantum architecture is the coherence time, which is the limit for how long a qubit is able to maintain its state. We use a realistic T1 time from an IBM device of $163.45\mu s$ [2]. Higher energy levels decohere more quickly. In theory, each state decays at a rate of $1/k$ where k is the energy level. We therefore use $81.73\mu s$ and $54.15\mu s$ as the T1 times for the $|2\rangle$ and $|3\rangle$ states.

Circuit Estimation

In addition to circuit simulation, we use two metrics to estimate the fidelity of a circuit without simulation. These will give us insight into how compiled circuits may perform by comparing direct simulation to our estimated fidelity.

The first is the product of all of the gate success rates in the circuit, called the gate expected probability of success (gate EPS). Since there are multiple classes of multi-qubit gates, some of which have higher fidelity than others, we use the product of these success rates.

The second is more complicated, as it attempts to capture the chance that the state of a qudit will collapse over the course of the circuit. We model this as an exponential decay where the probability of no decoherence is $\prod_{k=1}^{3} \exp(k * t_k/T_1)$ where t_k is the time the qubit spends in state k. When we construct the circuit we keep track of how long each qudit exists in the $|1\rangle$ state as the maximum state, and how long it exists in the maximum $|3\rangle$ state and calculate the probability of not decohering over the course of the execution for each qudit. The product of the expected success of each qudit is the EPS due to coherence for the entire circuit. When multiplied by the gate EPS, we have the EPS for the entire circuit solely through estimation.

Circuit Simulation

We want to go further than estimation. Since access to devices that can use ququarts at this scale are limited, we must use simulation. In general, simulation of quantum systems in the presence of noise with a classical computer is exponentially difficult since for a general pure quantum state containing n qudits requires d^n complex numbers and for non-pure states (which is acquired in noisy quantum systems) requires the representation of a density matrix ρ of size $d^n \times d^n$. In either case, at each step of the simulation we repeatedly perform matrix multiplication with $d^n \times d^n$ matrices for each time step in the circuit. While the more generic simulation method is more precise, it is not feasible outside of high performance computing settings and even in this case it is bounded and requires specific initial conditions [115].

A slightly more scalable alternate approach is the so-called *trajectory method* which requires us to store only a d^n sized state-vector [20]. The trajectory method is a stochastic approach and instead of maintaining the entire density matrix we begin with a random pure state, and apply gates as normal but inject randomly drawn coherent and non-coherent errors into the system at each step and the resulting quantum state remains pure after renormalization. In the limit, the trajectory method converges the same results of full density matrix simulation but has some space savings which allows us to simulate a few extra qudits. This work simulates circuits of up to 24 qubits, which can be represented by 12 ququarts using this method.

For this work, for each circuit of interest, we generate at least 1000 random quantum states and for each we simulate once and compute the average fidelity over all trials and all random states. We emphasize the need for the random *quantum* states rather than beginning with random classical states (e.g. bitstrings) even if many of our circuits are classical in nature because quantum errors do not always have noticeable effects on classical inputs (e.g. a Z-type Pauli error will have no effect on the measurement outcome of a classical bitstring).

In the past, prior work on simulation of qudit systems neglects the realistic duration differences between gates which results in drastically different usage patterns and simply injecting errors on a moment-to-moment basis can skew results. For example, in this work our direct-to-pulse compilation of CCX and CCZ gates have significantly different execution times. In a naive execution of the trajectory method, circuits with each of these perform similarly despite this execution time difference because the total number of *moments* is nearly the same. We modify the trajectory method simulation slightly to account for this difference. Rather than inserting many idle gates during each time step using the longest gate time included in that time step, before each gate, we insert one idle gate using the exact time that qudit has been idle. This is a more accurate representation of how long, when, and from which state these qudits could be decohering.

Noise Model for Qudit Systems

For qubits we consider both symmetric depolarizing and amplitude damping errors. There are four possible single-bit channels: no error (I), bit flip errors (X), phase flip errors (Z) and bit and phase flip errors ($Y = ZX$). In simulation each error channel is drawn with probability $p/3$. Two-qubit errors are given as the product of single-qubit errors, e.g. $X \otimes X$ for a bit flip on both interacting qubits; there are 16 possible channels of this type so each error occurs with probability $p/15$ and no error ($I \otimes I$) occurs with probability $1 - 15p$.

For a general qudit system, we consider a generalized form of these errors. The “bit-flip" type gates become $X_{+1\text{mod }d}$ and the “phase-flip” errors become $Z_d = \text{diag}(1, \exp\{\omega\}, \exp\{\omega^2\}, ... \exp\left\{\omega^{d-1}\right\})$ where ω^j is the $j - th$ root of unity. The product of $\{I, X_{+1\text{mod }d}, ...X^{d-1}_{+1\text{mod }d}\}$ and $\{I, Z_d, Z_d^2, ..., Z_d^{d-1}\}$ is a basis for all $d \times d$ Pauli matrices which allows us to construct a general symmetric qudit depolarizing channel. This explains the expected increase in error for using qudit systems: For a two-qubit gate the chance of *no* error is $1 - 15p$ while for a ququart this chance diminishes to $1 - 255p$ let alone possible differences in p [71].

Amplitude damping for qubits can be described as non-unitary transformations on the quantum state with operators $K_0 = \text{diag}(1, \sqrt{1-\lambda_1})$ and $K_1 = \sqrt{\lambda_1} e_{0,1}$. Here $e_{i,j}$ refers to a matrix with all 0's except for a 1 in the i-th row and j-th column and is of appropriate dimension. In the general qudit case we have $K_0 = \text{diag}(1, \sqrt{1-\lambda_1}, \sqrt{1-\lambda_2}, ... \sqrt{1-\lambda_d})$, $K_1 = \sqrt{\lambda_1} e_{0,1}$, ... $K_d = \sqrt{\lambda_d} e_{0,d-1}$. Since we primarily focus on a superconducting system in this study we take $\lambda_m = 1 - \exp\{-m\Delta t/T_1\}$ where Δt is the idling duration and T_1 is the coherence time of the qubit [56].

In this work we are also concerned with the manipulation of mixed-radix systems.When drawing an error for such a system, for example a qubit-ququart interaction, we consider only relevant errors for the respective participant. For instance, a two-qudit error is drawn from $P_2 \otimes P_4$ and not from $P_4 \otimes P_4$ (where P_d is the set of d-dimensional Paulis, exactly the set of potential errors described above). Similarly, for two-qubit gates on encoded qubits, we consider only single *ququart* errors since gates on encoded systems are equivalent to single-ququart gates. Our model does not account for several real problems - for example crosstalk and population leakage - as these are challenging to incorporate into simulation. These are problems that both qubits and qudits experience, though there is an expectation that leaking in particular is worse for higher dimension d.

4.6.6 Results

When we are able to perform native implementations of three-qubit gates via ququarts, we significantly reduce the number of gates that need to be executed, reducing failure due to the use of many error prone gates. However, we may face the setback that the reduction in time to execute these gates is not enough to overcome the reduced coherence time of higher radix states. In Figure 4.29, we examine the simulation fidelities for three-qubit compilation strategies across different sized circuits using Toffoli gate based decompositions. Each point represents the average fidelity of 1000+ different initial states run once, each

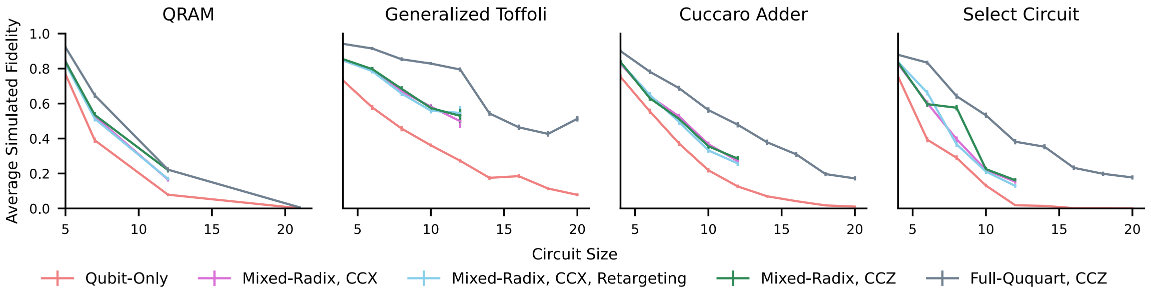


Figure 4.29: Simulated results for QRAM, Generalized Toffoli, Cuccaro Adder and Select Circuit from 5 to 21 qubits with different mixed-radix and full-ququart compilation strategies. The mixed-radix strategies do not have complete error bars due to the requirement to simulate a four-level system for every qubit which would require more than 86 GB of memory per circuit in our simulation framework.

with randomly inserted error. The error bars are the standard error, which is the standard deviation of the all the trials divided by the square root of the number of trials. The mixed-radix compilation schemes stop at 12 qubits due to memory based computational limitations. While mixed-radix circuits start in an all-qubit state, we must model them as if they are entirely on ququarts, since to accurately inject error into the system we must be able to model the higher levels at all times. This restricts the number of physical devices we are able to simulate in this scheme to 12 simulable ququarts.

The first thing we notice is that all of our mixed-radix and full-ququart compilation strategies are able to exceed the fidelity of our baseline qubit-only compilation scheme. From a pure gate error perspective, this should not be unexpected, each of these schemes greatly reduces the number of gates required to execute the same operation. Figure 4.30 demonstrates how the EPS for gate error is substantially improved by using three two-ququart gates gates or one two-ququart gate for full-ququart computation rather than eight two-qubit gates for mixed-radix computation. And, as hoped, the simulation finds that using the $|2\rangle$ and $|3\rangle$ state, and the idle time potentially spent in these states, does not outweigh the benefits of the using fewer gates and the shorter circuit duration. The shorter duration of the gates is counter acts the increased decoherence rate of the ququarts. Figure 4.30 also

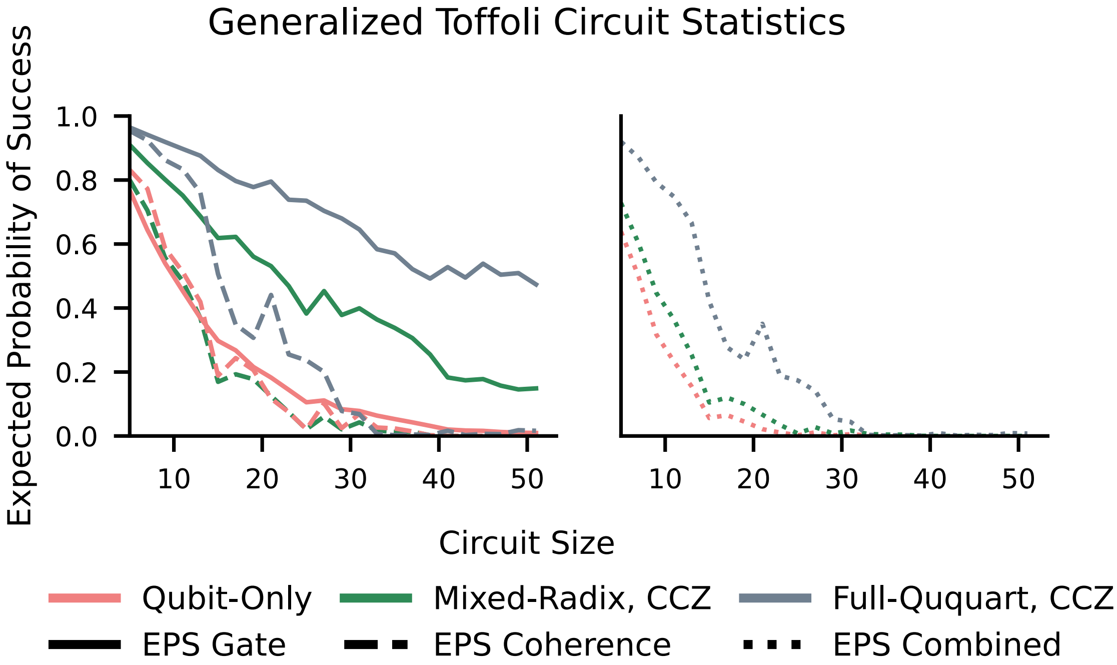


Figure 4.30: EPS statistics for the generalized Toffoli circuit. We show the gate and coherence EPS on the left and the product EPS on the right.

demonstrates the same point. The coherence EPS between all of the mixed-radix strategies and the qubit-only baseline are nearly the same, and is actually improved for full-ququart strategies. The general trend of our simulation results is mirrored in Figure 4.30 where the total EPS of the circuit is shown. As the EPS trends match what we find in our simulated results, we are able to infer that the scaling of simulation will match the scaling of the EPS results, which we can calculate when simulation becomes intractable. While we only show examples of the generalized Toffoli circuit, the results are similar for other circuits.

Digging into the difference between the strategies, we find that the mixed-radix strategies are all relatively similar to one another, with some additional separation as the size of the circuit increases. We first compare the mixed-radix Hadamard corrected CCX gates, shown in light blue, to the mixed-radix gates without this correction, shown in pink. While there is some cancellation between the single qubit gates, the extra serialization and marginal gate error of the correction gates is a drawback to using this correction strategy based on the

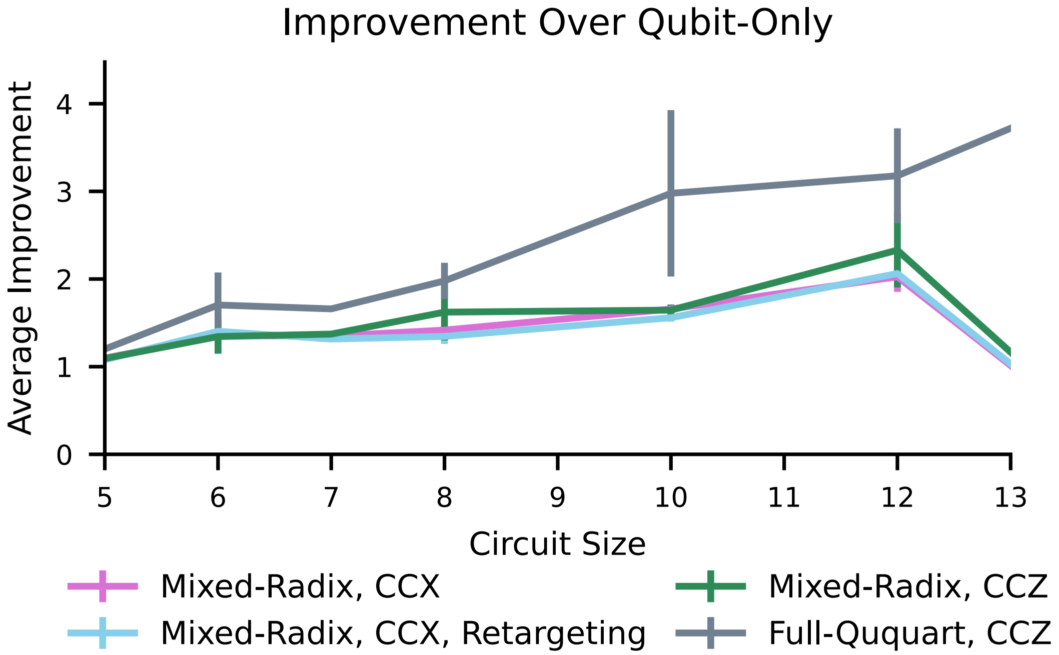


Figure 4.31: The average fidelity improvement for each compilation method over the qubit-only compilation method as the size of the circuit increases.

simulated results. The reduction in time from the better configuration of the CCX gate is not always enough to overcome these additional costs. If we instead use CCZ decomposition, shown in green, we consistently achieve the same, or better, fidelity, especially as the size of the circuit increases and the reduction from CCZ gates is more pronounced. In these cases, the benefits found by using shorter target-independent gates from the start rather than retargeting improves the fidelity of the circuit in this mixed-radix regime. In Figure 4.31 we find that the mixed radix gates achieve 2x better fidelities for circuit size 12, which is a significant improvement over qubit-only computation. This alone would be a important optimization for three-qubit gate based circuits.

We also find that the ququart compilation scheme, shown in grey, has an even higher fidelity improvement, up to 3x reductions as seen in Figure 4.31. The reasoning behind this is two-fold. The first is that we no longer need to encode and decode gates before each three-qubit gate in this scheme. As seen, gate reduction is important, and this reduces the number of gates. The second is the reduction of communication. With the higher connectivity afforded at all times, we further reduce qubit communication required to perform certain gates. Both of these factors add to the reduction in time, keeping the full-ququart based circuits under the coherence limits and maintaining higher circuit fidelity overall. At this level we are able to further reduce the overall circuit time by using faster, target-independent CZ gates in place of CX.

It is worth noting the points where full-ququart compilation does not outperform mixed-radix compilation to the same degree. This is particularly noticeable in the QRAM circuit. There are more than double the CX gates as Toffolis in this circuit. As a result, the serialization induced by ququarts with slower two-qubit gates reduces the effectiveness of ququarts. This will be explored later in a sensitivity study. Additionally, these benchmarks are only kernels of computations that could be used within the context of larger circuits. In such cases, we will not have the benefit of a perfect mapping to start. This would not affect the improvement in fidelity from the qubit only to the mixed-radix strategies, but the effort to encode the qubits into a full-ququart regime before execution may outweigh the benefits of ququart compilation.

Special Gate Case Study: CSWAP

As detailed in Section 4.6.2 we could instead decompose to a different-three qubit gate in the circuit. In the case of QRAM, this is the CSWAP gate. In Figure 4.32, we explore the differences in fidelity when we use CSWAP gate alongside the original results using CCZ gates. A CSWAP can be constructed from two CX gates and one CCX gate, but cannot be re-targeted in the same way. Regardless, in the mixed-radix state, by orienting the CSWAP such that the targets are separate from the controls when possible, we see improvements over the CCZ decomposition. In fact it is able to beat the full-ququart CCZ compilation in some cases because of the reduced number of CX gates. While we can always attempt to

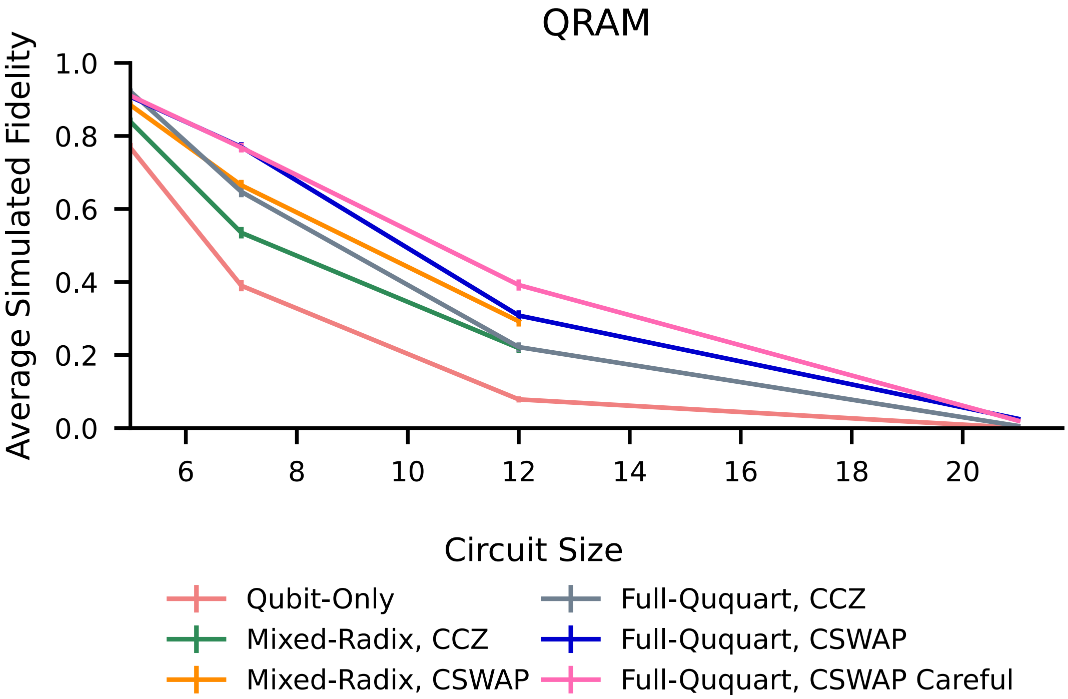


Figure 4.32: Simulated fidelities for the QRAM circuit that included the direct-to-pulse implementation of the CSWAP gate compared to the more general CCZ based methods developed for the other circuits.

encode the qubits favorably in a mixed-radix environment, this is not as natural a change when compiling for ququarts, and could lead to bad configurations if we solely focus on the disruption of qubits on ququarts. If we focus on the CSWAP in a full-ququart regime, the basic version shown in blue, and instead use the strategy that places the targets in the same ququart, shown in bright pink, we find even more improvement to our full-ququart encoding regime. This further indicates the importance using the best decomposition possible by separating the targets and the controls of certain gates.

Sensitivity to Ququart Gate Error Rate

While we synthesized our gates using a realistic hamiltonian, it is still more difficult to physically realize gates that access higher energy levels. In Figure 4.33a we explore how

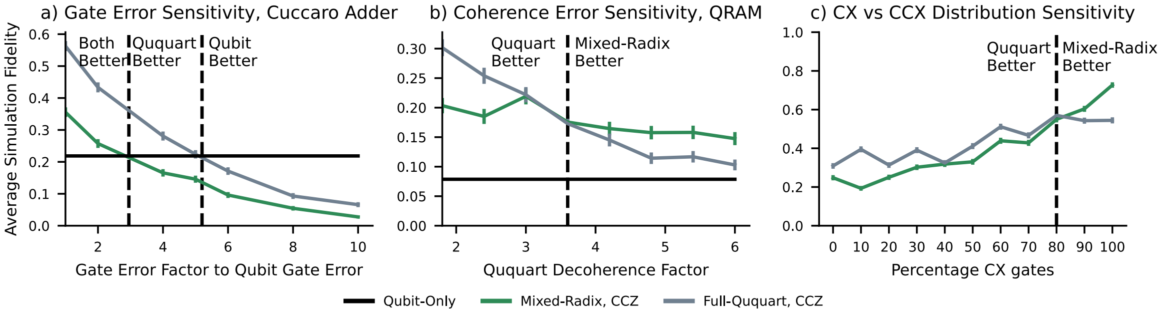


Figure 4.33: The results of several sensitivity studies. (a) Changes in CCZ compilation strategies' fidelities as gate error ququarts increases. (b) Changes in CCZ compilation strategies' fidelities as coherence error for the $|2\rangle$ and $|3\rangle$ level states changes. (c) Differences in fidelities between mixed-radix and full-ququart compilation strategies as the distribution of CX gates to CCX gates in a circuit changes. In all graphs The black line represents the qubit-only compilation method. Below this point mixed-radix or full-ququart methods are more error prone than using only qubits. Please note the different scaling on the y-axis.

the simulated fidelity changes as the error on ququart and mixed-radix gate increases for an 11-qubit Cuccaro Adder. Both strategies see a very fast drop off as the gate error increases, crossing over the qubit-only baseline fidelity when the ququart error rate is between two and four times worse that qubit gates for mixed-radix compilation (97% fidelity), and between four and six times worse for full-ququart compilation (94%). While these are still high fidelity targets, it does indicate that we do not need our three-qubit gates to have the same fidelity as our two-qubit gates for these strategies to be successful.

Sensitivity to Ququart Coherence Error Rate

In this work we selected the expected theoretical decrease in coherence time as the grounding for most of our simulation experiments. However, physical realizations don't always meet reality. Accessing higher energy levels may prove to be more costly in terms of coherence time due to lack of control, or it may be less of an issue as has been found by some testbeds when accessing the qutrit state [22]. In Figure 4.33(b) we demonstrate the effects of changing the rate that the $|2\rangle$ and $|3\rangle$ levels decohere for an 12-qubit QRAM circuit. The main detail to note is that as the rate increases, the distance between mixed-radix and full-ququart fidelities decreases until mixed-radix becomes higher fidelity. Mixed-radix gates do not spend as much time in the higher level states, so as machines are developed and these level are more unstable, it may be better to avoid using full ququart encodings for larger circuits.

Ratio of Three Qubit Gates to Two Qubit Gates

It may not always be the case that the number of three qubit gates greatly exceeds the number of two qubit gates. Future applications may have a higher mix of two-qubit gates to three-qubit gates, or may only require a few three qubit gates to perform the desired operation. In Figure 4.33c, we example how the fidelity of different mixes of two-qubit to three-qubit gates is effected by compilation using a full-ququart strategy versus a mixed-radix strategy for an 11-qubit circuit. As the ratio of two-qubit to three-qubit gates increases, it becomes less and less profitable to use a full ququart encoding. At 80% CX gates it becomes more profitable to remain in the mixed-radix regime. Using CX gates on ququarts requires more serialization, since we cannot perform two separate operations on qubits encoded in the same ququart. This increases the time, and we start seeing the effects of reduced coherence times. This changes the calculus about when mixed-radix is better than full-ququart compilation. In cases where we don't need to use as many three qubit gates, it does not make as much sense to use ququarts for the entirety of the circuit. While this indicates that quantum circuits that only use two-qubit gates do not benefit from this encoding scheme, we can use resynthesis tools [116] to automatically insert three-qubit gates into the circuit, such as in [81]. However, resynthesis can introduce additional error as a perfect direct translation is not always possible and is better explored in a future work.

Random Mappings Most of the benchmarks used in this evaluation are "kernels" of computation, and are intended to be used as smaller portions of larger circuits. As a result,

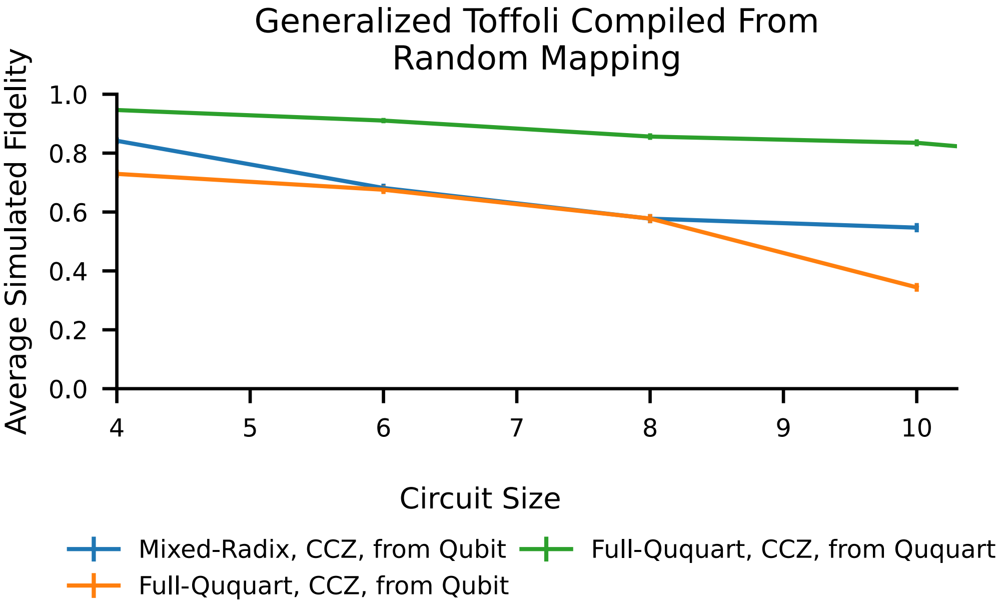


Figure 4.34: The results of the different mixed-radix and full-ququart compilation strategies on random mappings of only qubit-encoding, or only ququart encoding.

the assumption that the compiler will have the opportunity to generate a well mapped circuit from the outset is not necessarily true. We explore the senstivity of the different strategies to random mappings of qubits. In particular, we explore three different setups in 4.34 for the Generalized Toffoli circuit. The first starts with all data qubits in random qubits, without encodings. Then we performing routing, and use mixed-radix CCZ gates to perform the operations. The second starts with the same random mapping, but instead, contracts the mapping such that all qubits are encoded into ququarts. We then perform all computation on this encoding, before returning the qubits to qubit-only encodings. The third starts in a random ququart encoded mapping, before performing the computation solely on ququarts. In this study, we find that mixed-radix and full-ququart compilations are relatively similar to one another. The encoding and decoding in the mixed-radix case, is a slightly less expensive set of operations than encoding to ququarts and performing all communication in this encoding. And, we find that a random mapping on ququarts does not necessarily hinder efficient execution of the circuit. As explored in the previous sensitivity study, it may not always be the case that circuits have high percentages of three-qubit gates, the mixed-radix strategy will be more advantageous in many cases when a good initial mapping cannot be generated.

4.6.7 Alternate Strategies

While this work is the first we are aware of to explore compilation of higher-radix qudits for three qubit gates via encoding specifically for superconducting qubits, there have been studies using existing superconducting qubit technology to execute native three qubit gates. In particular, Kim et al. [59] and Gokhale et al. [43] have explored driving two connections between three qubits in a line to perform three qubit gates in a superconducting architecture. These studies found varying success in doing so. Gokhale developed a technique to execute two CX gates in parallel in a single CXX gate. These gates did not find improved fidelity, but achieved similar goals of faster parallel gate execution than serial execution as described in this work. It should be noted that this was done without explicit calibration for this sort of operation.

On the other hand, Kim developed a very high fidelity, 98.2%, iToffoli gate between three superconducting qubits. In this case, both controls induce a state change in a center qubit by driving the connections between the qubits. This gate is performed very quickly with a gate duration of 392 ns. While a meaningful result, it is difficult to compare this work to our own as the device has significantly different Hamiltonian than this work. Additionally, this took significant calibration between each of the three qubits, which can be an expensive process. Requiring calibration between each set of three qubits on a device is much more expensive than calibrating each pair of connected devices as would be required by our scheme and is already done on current hardware.

There have also been a few physical realizations of the iToffoli gate that use iSWAP gate, a CPHASE gate, and a reverse iSWAP gate using the $|2\rangle$ state to change the state of a qubit in [49, 36]. These are conceptually similar to the encode, mixed-radix Toffoli, and decode scheme that was laid out in this work. But, these gates were not synthesized directly to pulses as was done here. Regardless, they are good motivation to continue working on this problem.

4.6.8 Discussion

This section takes advantage of increased connectivity, and interaction potential when we have encoded qubits into a four-level system. Encoding qubits in this way allows for the interaction of three to four qubits across a single physical connection, and we synthesize a library or efficient three-qubit gates via optimal control that take advantage of this virtual connectivity and are much faster and higher fidelity than performing the decomposition of a three-qubit gate. We then use these gates to develop compilation strategies, the quantum waltz, that use the most efficient configurations of three-qubit gates on mixed-radix and full-ququart systems to produce circuits that achieve 2x and 3x better simulated fidelities in mixed-radix and full-ququart environments, respectively. Despite difficulties of accessing and performing operations on higher level states, this efficient implementation of three-qubit gates provides worthwhile benefits for quantum computation.

Mixed-radix and full-ququart implementations of three-qubit gates makes ququart computation an invaluable piece of the quantum computing repertoire. It is more flexible than previous hand optimized circuits to improve circuit execution via higher radix devices, does not require that the entire circuit use quaternary-based logic, and can be selectively applied to certain sections of larger circuits via the encoding scheme. Realized implementations of these gates provide a massive opportunity to improve near-term execution of quantum circuits and expand the capabilities of quantum computers.

CHAPTER 5

CONCLUSION AND FUTURE WORK

The fundamental aspect of this work is that the core of an algorithm for general purpose quantum computing can be adapted for a specific architectures with considerable improvement to the results of quantum programs. There are some shared errors between quantum architectures, mainly gate errors and decoherence errors and compilation goals such as low gate count and circuit duration are also shared. But these changes allow for the use of the unique features of architectures to further reduce these metrics and in some cases, working around their flaws. A general mapping, routing or scheduling algorithm, such as those included in most quantum frameworks, applied without alteration to a quantum program to be run on a specific device will fail to produce the most successful version of that program.

In the context of Neutral Atom devices, we adapt mapping and routing strategies to continue to honor a more dense interaction graph, but understand that some SWAPs could be much more disruptive to locality that a general communication algorithm. To get around this issue, we adapt our disruption method to take into account a physical distance, not just a topological distance, between qubits. This gave our router a better sense of how a given mapping would be affected by a certain SWAP path. Even further, a traditional routing algorithm would prefer to use the entirety of a machine, and hit the bounds of maximum interaction distances. This is a detriment to Neutral Atom devices. Atom loss can make this infeasible. But by adapting the algorithm to be more constrained, a use a "smaller" device in both size and maximum interaction distance, Neutral Atoms save significant overhead time, increasing utilization of the device.

Higher-radix qudit systems have a unique feature of non-static interaction graphs. A SWAP can be more error prone, or have a longer duration, breaking the initial assumption that all SWAPs have the same error cost to a circuit. Further, this is dependent on the current maximum energy level of the qudit at the time. Adapting to this strategy meant keeping track of the current state of the qudits, and updating the interaction graph accordingly in both the qudit interaction graph, and the expanded qubit interaction graph in the cases where two qubits were encoded into a single ququart. Additionally, with several different sets of generated pulses to achieve the computation of qubits-on-ququarts, we drastically improve we are able to accommodate three-qubit gates which significantly improves both the circuit duration and the error from gate execution. This circumvents the increased error from reduced coherence time of higher-radix qudits, and provides real benefits for using qudits.

The explorations detailed in this work are small portion of the many different architectures in development in quantum computing. IBM has a road map, expecting processors with many thousands of qubits [40]. This will not be achievable on a single chip, and we will likely require changes to our routing algorithms that can account for more error prone, and expensive interconnects [37]. Neutral Atom devices can perform physical movement of qubits, not just data movement as was explored today [16]. In the short term, it may be viable to have tailored compiler toolchains for each type of hardware, but as we move towards larger quantum hardware, the pipeline will need to be able to adapt as well.

With these increases in circuit size on the horizon, there are several avenues of future work to be explored. The first, put simply, is the issue of scalability. Hardware will continue to increases in size in both number of qubits and connectivity, and circuits will make use of these qubits. Many of the compilation algorithms use global information, rather than local decision making. These strategies can scale poorly in the classical sense. While it can be argued that heavy optimization is not an issue for quantum computation since any error reduction can signficantly improve the program, this isn't the most practical approach for cloud providers attempting to efficiently run circuits from many varied users. Other future work involves mapping and routing circuits effectively across different sections of a quantum devices, using the more error prone interconnects mentioned earlier in an effective way and without accumulating error. Ideally, quantum computing and hardware will also be moving into a more fault-tolerant, potentially error corrected era. The role for mapping, routing and pulse control operations is more nebulous in this regime. Exploring compilation for different gate sets tailored to these error correction requirements is a worthwhile exploration, as are interesting ways to use the unique features of a device, such as long distance interaction.

Another direction focuses more on compilation strategy of larger circuits. Many of the benchmarks used in this work are compuational kernels that are used as a smaller piece of a larger circuit. Yet, they are used as demonstrations of mapping of routing algorithms. In the near future, it may be the case that different portions of a circuit will have vastly different interaction patterns. The mapping and routing stratgies listed in this work have some resiliency to this concept through the use of a lookahead pattern, but their effectiveness as a "mid-circuit" strategy has not be explored. It is worth developing heuristics for how a circuit can be subdivided into different interaction patterns, and developing algorithms that perform mid-circuit remapping, or change their routing strategy entirely to fit the the new interaction strategy between qubits.

A third path is developing an understanding of how different optimizations affect one another. Many quantum compilation frameworks tout a suite of different optmizations, often times with a number of different preset list of optimizations that claim to have certain effects on a circuit that use a general intuition of what *should* be best for the circuit in terms of depth and gate count. But, it would valuable to have an automated intuition for which kinds of optimizations should come before others. An exploration into how different optimization orders help or hurt a circuit could be invaluable to developing future optimizations and adding them to existing pipelines. Additionally, a method for information sharing across throughout a compiler pipeline such that optimizations do not conflict with one another could be instrumental as optimizations for quantum architectures increase in number.

As we develop new algorithms and compiler optimizations to execute quantum programs, it is important to reason about how it can be adaptable to specific features, without requiring an upheaval of the entire process. In this work, while the core of each compilation pipeline was relatively similar, it did not natively fit within the bounds of a particular framework, they were rewritten from the ground up to inherit the native features of the architecture itself. Most today frameworks are only able to support the basic assumptions laid out in the original algorithm as it is the most device-independent way to perform an optimization.

As we push into larger, and more varied quantum systems, software developers and quantum compiler engineers need to be cognizant about the fact that not all devices can be assumed to be the same. Although they may have shared features, that is not enough to based on algorithm on that will be applied to several different architectures. We must work to develop systems that can be informed of particular features earlier in compilation so that different architectures can be used to their full potential.

REFERENCES

[1] David Schuster Lab.

[2] IBM Quantum Devices.

[3] M. Abdelhafez, B. Baker, A. Gyenis, P. Mundada, A. A. Houck, D. Schuster, and J. Koch. Universal gates for protected superconducting qubits using optimal control. *Phys. Rev. A*, 101(2):022321, Feb. 2020.

[4] M. S. ANIS, Abby-Mitchell, H. Abraham, AduOffei, R. Agarwal, G. Agliardi, M. Aharoni, V. Ajith, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, M. Amy, S. Anagolum, Anthony-Gandon, E. Arbel, A. Asfaw, A. Athalye, A. Avkhadiev, C. Azaustre, P. BHOLE, A. Banerjee, S. Banerjee, W. Bang, A. Bansal, P. Barkoutsos, A. Barnawal, G. Barron, G. S. Barron, L. Bello, Y. Ben-Haim, M. C. Bennett, D. Bevenius, D. Bhatnagar, P. Bhatnagar, A. Bhobe, P. Bianchini, L. S. Bishop, C. Blank, S. Bolos, S. Bopardikar, S. Bosch, S. Brandhofer, Brandon, S. Bravyi, N. Bronn, Bryce-Fuller, D. Bucher, A. Burov, F. Cabrera, P. Calpin, L. Capelluto, J. Carballo, G. Carrascal, A. Carriker, I. Carvalho, A. Chen, C.-F. Chen, E. Chen, J. C. Chen, R. Chen, F. Chevallier, K. Chinda, R. Cholarajan, J. M. Chow, S. Churchill, CisterMoke, C. Claus, C. Clauss, C. Clothier, R. Cocking, R. Cocuzzo, J. Connor, F. Correa, Z. Crockett, A. J. Cross, A. W. Cross, S. Cross, J. Cruz-Benito, C. Culver, A. D. Córcoles-Gonzales, N. D, S. Dague, T. E. Dandachi, A. N. Dangwal, J. Daniel, M. Daniels, M. Dartiailh, A. R. Davila, F. Debouni, A. Dekusar, A. Deshmukh, M. Deshpande, D. Ding, J. Doi, E. M. Dow, P. Downing, E. Drechsler, E. Dumitrescu, K. Dumon, I. Duran, K. EL-Safty, E. Eastman, G. Eberle, A. Ebrahimi, P. Eendebak, D. Egger, ElePT, Emilio, A. Espiricueta, M. Everitt, D. Facoetti, Farida, P. M. Fernández, S. Ferracin, D. Ferrari, A. H. Ferrera, R. Fouilland, A. Frisch, A. Fuhrer, B. Fuller, M. GEORGE, J. Gacon, B. G. Gago, C. Gambella, J. M. Gam-

betta, A. Gammanpila, L. Garcia, T. Garg, S. Garion, J. R. Garrison, J. Garrison, T. Gates, H. Georgiev, L. Gil, A. Gilliam, A. Giridharan, Glen, J. Gomez-Mosquera, Gonzalo, S. d. l. P. González, J. Gorzinski, I. Gould, D. Greenberg, D. Grinko, W. Guan, D. Guijo, J. A. Gunnels, H. Gupta, N. Gupta, J. M. Günther, M. Haglund, I. Haide, I. Hamamura, O. C. Hamido, F. Harkins, K. Hartman, A. Hasan, V. Havlicek, J. Hellmers, \. Herok, S. Hillmich, H. Horii, C. Howington, S. Hu, W. Hu, C.-H. Huang, J. Huang, R. Huisman, H. Imai, T. Imamichi, K. Ishizaki, Ishwor, R. Iten, T. Itoko, A. Ivrii, A. Javadi, A. Javadi-Abhari, W. Javed, Q. Jianhua, M. Jivrajani, K. Johns, S. Johnstun, Jonathan-Shoemaker, JosDenmark, JoshDumo, J. Judge, T. Kachmann, A. Kale, N. Kanazawa, J. Kane, Kang-Bae, A. Kapila, A. Karazeev, P. Kassebaum, T. Kehrer, J. Kelso, S. Kelso, H. v. Kemenade, V. Khanderao, S. King, Y. Kobayashi, Kovi11Day, A. Kovyrshin, R. Krishnakumar, P. Krishnamurthy, V. Krishnan, K. Krsulich, P. Kumkar, G. Kus, R. LaRose, E. Lacal, R. Lambert, H. Landa, J. Lapeyre, J. Latone, S. Lawrence, C. Lee, G. Li, T. J. Liang, J. Lishman, D. Liu, P. Liu, Lolcroc, A. K. M, L. Madden, Y. Maeng, S. Maheshkar, K. Majmudar, A. Malyshev, M. E. Mandouh, J. Manela, Manjula, J. Marecek, M. Marques, K. Marwaha, D. Maslov, P. Maszota, D. Mathews, A. Matsuo, F. Mazhandu, D. McClure, M. McElaney, C. McGarry, D. McKay, D. McPherson, S. Meesala, D. Meirom, C. Mendell, T. Metcalfe, M. Mevissen, A. Meyer, A. Mezzacapo, R. Midha, D. Miller, H. Miller, Z. Minev, A. Mitchell, N. Moll, A. Montanez, G. Monteiro, M. D. Mooring, R. Morales, N. Moran, D. Morcuende, S. Mostafa, M. Motta, R. Moyard, P. Murali, D. Murata, J. Müggenburg, T. NEMOZ, D. Nadlinger, K. Nakanishi, G. Nannicini, P. Nation, E. Navarro, Y. Naveh, S. W. Neagle, P. Neuweiler, A. Ngoueya, T. Nguyen, J. Nicander, Nick-Singstock, P. Niroula, H. Norlen, NuoWenLei, L. J. O'Riordan, O. Ogunbayo, P. Ollitrault, T. Onodera, R. Otaolea, S. Oud, D. Padilha, H. Paik, S. Pal, Y. Pang, A. Panigrahi, V. R. Pascuzzi, S. Perriello, E. Peterson, A. Phan, K. Pilch,

F. Piro, M. Pistoia, C. Piveteau, J. Plewa, P. Pocreau, A. Pozas-Kerstjens, R. Pracht, M. Prokop, V. Prutyanov, S. Puri, D. Puzzuoli, Pythonix, J. Pérez, Quant02, Quintiii, R. I. Rahman, A. Raja, R. Rajeev, I. Rajput, N. Ramagiri, A. Rao, R. Raymond, O. Reardon-Smith, R. M.-C. Redondo, M. Reuter, J. Rice, M. Riedemann, Rietesh, D. Risinger, P. Rivero, M. L. Rocca, D. M. Rodríguez, RohithKarur, B. Rosand, M. Rossmannek, M. Ryu, T. SAPV, N. R. C. Sa, A. Saha, A. A. Saki, S. Sanand, M. Sandberg, H. Sandesara, R. Sapra, H. Sargsyan, A. Sarkar, N. Sathaye, N. Savola, B. Schmitt, C. Schnabel, Z. Schoenfeld, T. L. Scholten, E. Schoute, M. Schulterbrandt, J. Schwarm, J. Seaward, Sergi, I. F. Sertage, K. Setia, F. Shah, N. Shammah, W. Shanks, R. Sharma, Y. Shi, J. Shoemaker, A. Silva, A. Simonetto, D. Singh, D. Singh, P. Singh, P. Singkanipa, Y. Siraichi, Siri, J. Sistos, I. Sitdikov, S. Sivarajah, Slavikmew, M. B. Sletfjerding, J. A. Smolin, M. Soeken, I. O. Sokolov, I. Sokolov, V. P. Soloviev, SooluThomas, Starfish, D. Steenken, M. Stypulkoski, A. Suau, S. Sun, K. J. Sung, M. Suwama, O. S\textbackslashlowik, H. Takahashi, T. Takawale, I. Tavernelli, C. Taylor, P. Taylour, S. Thomas, K. Tian, M. Tillet, M. Tod, M. Tomasik, C. Tornow, E. d. l. Torre, J. L. S. Toural, K. Trabing, M. Treinish, D. Trenev, TrishaPe, F. Truger, G. Tsilimigkounakis, D. Tulsi, D. Tuna, W. Turner, Y. Vaknin, C. R. Valcarce, F. Varchon, A. Vartak, A. C. Vazquez, P. Vijaywargiya, V. Villar, B. Vishnu, D. Vogt-Lee, C. Vuillot, J. Weaver, J. Weidenfeller, R. Wieczorek, J. A. Wildstrom, J. Wilson, E. Winston, WinterSoldier, J. J. Woehr, S. Woerner, R. Woo, C. J. Wood, R. Wood, S. Wood, J. Wootton, M. Wright, L. Xing, J. YU, B. Yang, U. Yang, J. Yao, D. Yeralin, R. Yonekura, D. Yonge-Mallo, R. Yoshida, R. Young, J. Yu, L. Yu, Yuma-Nakamura, C. Zachow, L. Zdanski, H. Zhang, I. Zidaru, B. Zimmermann, C. Zoufal, aeddins-ibm, alexzhang13, b63, bartek-bartlomiej, bcamorrison, brandhsn, chetmurthy, deeplokhande, dekel.meirom, dime10, dlasecki, ehchen, ewinston, fanizzamarco, fs1132429, gadial, galeinston, georgezhou20, georgiosts, gruu, hhorii, hhyap, hykavitha, itoko, jeppevinkel, jessica-angel7, jezerjojo14, jliu45, johannesgreiner, jscott2, klinvill, krutik2966, ma5x, michelle4654, msuwama, nico-lgrs, nrhawkins, ntgiwsvp, ordmoj, s. pahwa, pritamsinha2304, rithikaadiga, ryancocuzzo, saktar-unr, saswati-qiskit, septembrr, sethmerkel, sg495, shaashwat, smturro2, stern-parky, strickroman, tigerjack, tsura-crisaldo, upsideon, vadebayo49, welien, willhbang, wmurphy-collabstar, yang.luh, and M. Čepulkovskis. Qiskit: An Open-source Framework for Quantum Computing, 2021.

[5] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, Oct. 2019.

[6] R. Babbush, C. Gidney, D. W. Berry, N. Wiebe, J. McClean, A. Paler, A. Fowler, and H. Neven. Encoding Electronic Spectra in Quantum Circuits with Linear T Complexity. *Physical Review X*, 8(4), Oct. 2018.

[7] J. M. Baker, C. Duckering, and F. T. Chong. Efficient quantum circuit decompositions

via intermediate qudits. In *2020 IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 303–308. IEEE, 2020.

[8] J. M. Baker, C. Duckering, P. Gokhale, N. C. Brown, K. R. Brown, and F. T. Chong. Improved quantum circuits via intermediate qutrits. *ACM Transactions on Quantum Computing*, 1(1):1–25, 2020.

[9] J. M. Baker, C. Duckering, A. Hoover, and F. T. Chong. Decomposing Quantum Generalized Toffoli with an Arbitrary Number of Ancilla, 2019. _eprint: 1904.01671.

[10] J. M. Baker, C. Duckering, A. Hoover, and F. T. Chong. Time-sliced quantum circuit partitioning for modular architectures. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*. ACM, May 2020.

[11] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical review A*, 52(5):3457, 1995. Publisher: APS.

[12] D. Barredo, S. De Léséleuc, V. Lienhard, T. Lahaye, and A. Browaeys. An atom-by-atom assembler of defect-free arbitrary two-dimensional atomic arrays. *Science*, 354(6315):1021–1023, 2016. Publisher: American Association for the Advancement of Science.

[13] D. Barredo, V. Lienhard, S. De Leseleuc, T. Lahaye, and A. Browaeys. Synthetic three-dimensional atomic structures assembled atom by atom. *Nature*, 561(7721):79–82, 2018. Publisher: Nature Publishing Group.

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

[15] M. S. Blok, V. V. Ramasesh, T. Schuster, K. O'Brien, J. M. Kreikebaum, D. Dahlen,

A. Morvan, B. Yoshida, N. Y. Yao, and I. Siddiqi. Quantum Information Scrambling in a Superconducting Qutrit Processor. *Physical Review X*, 11(2):021010, Apr. 2021.

[16] D. Bluvstein, H. Levine, G. Semeghini, T. T. Wang, S. Ebadi, M. Kalinowski, A. Keesling, N. Maskara, H. Pichler, M. Greiner, V. Vuletić, and M. D. Lukin. A quantum processor based on coherent transport of entangled atom arrays. *Nature*, 604(7906):451–456, Apr. 2022.

[17] A. Bocharov, S. X. Cui, M. Roetteler, and K. M. Svore. Improved Quantum Ternary Arithmetics. June 2016.

[18] A. Bocharov, M. Roetteler, and K. M. Svore. Factoring with qutrits: Shor's algorithm on ternary and metaplectic quantum architectures. *Physical Review A*, 96(1):012306, July 2017.

[19] N. C. Brown and K. R. Brown. Comparing Zeeman qubits to hyperfine qubits in the context of the surface code: ^{174}Yb^{+} and ^{171}Yb^{+}. *Physical Review A*, 97(5):052301, May 2018. arXiv:1803.02545 [quant-ph].

[20] T. A. Brun. A simple model of quantum trajectories. *American Journal of Physics*, 70(7):719–737, 2002.

[21] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage. Trapped-ion quantum computing: Progress and challenges. *Applied Physics Reviews*, 6(2):021314, 2019. Publisher: AIP Publishing LLC.

[22] A. Cervera-Lierta, M. Krenn, A. n. Aspuru-Guzik, and A. Galda. Experimental High-Dimensional Greenberger-Horne-Zeilinger Entanglement with Superconducting Transmon Qutrits. *Physical Review Applied*, 17(2), Feb. 2022.

[23] P. Chapman. Scaling IonQ's Quantum Computers: The Roadmap, 2020.

[24] Y. Chi, J. Huang, Z. Zhang, J. Mao, Z. Zhou, X. Chen, C. Zhai, J. Bao, T. Dai, H. Yuan, M. Zhang, D. Dai, B. Tang, Y. Yang, Z. Li, Y. Ding, L. K. Oxenløwe, M. G. Thompson, J. L. O'Brien, Y. Li, Q. Gong, and J. Wang. A programmable qudit-based quantum processor. *Nature Communications*, 13(1):1166, Dec. 2022.

[25] S. U. Cho, M.-H. Bae, K. Kang, and N. Kim. Tunable superconducting qudit mediated by microwave photons. *AIP Advances*, 5(8):087186, Aug. 2015.

[26] J. P. Covey, I. S. Madjarov, A. Cooper, and M. Endres. 2000-Times Repeated Imaging of Strontium Atoms in Clock-Magic Tweezer Arrays. *Phys. Rev. Lett.*, 122(17):173201, May 2019. Publisher: American Physical Society.

[27] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah. On the qubit routing problem. *arXiv preprint arXiv:1902.08091*, 2019.

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

[29] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. A performance comparison of contemporary DRAM architectures. In *Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367)*, pages 222–233, 1999.

[30] P. Das, S. S. Tannu, P. J. Nair, and M. Qureshi. A Case for Multi-Programming Quantum Computers. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, pages 291–303, New York, NY, USA, 2019. Association for Computing Machinery. event-place: Columbus, OH, USA.

[31] C. Developers. Cirq, Dec. 2022.

[32] Y. Ding, P. Gokhale, S. F. Lin, R. Rines, T. Propson, and F. T. Chong. Systematic Crosstalk Mitigation for Superconducting Qubits via Frequency-Aware Compilation.

In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, Oct. 2020.

[33] C. Duckering, J. M. Baker, A. Litteken, and F. 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*. ACM, Apr. 2021.

[34] M. Endres, H. Bernien, A. Keesling, H. Levine, E. R. Anschuetz, A. Krajenbrink, C. Senko, V. Vuletic, M. Greiner, and M. D. Lukin. Atom-by-atom assembly of defect-free one-dimensional cold atom arrays. *Science*, 354(6315):1024–1027, 2016. Publisher: American Association for the Advancement of Science.

[35] E. Farhi, J. Goldstone, and S. Gutmann. A Quantum Approximate Optimization Algorithm, 2014.

[36] A. Fedorov, L. Steffen, M. Baur, M. P. d. Silva, and A. Wallraff. Implementation of a Toffoli gate with superconducting circuits. *Nature*, 481(7380):170–172, Dec. 2011.

[37] B. Foxen, J. Y. Mutus, E. Lucero, R. Graff, A. Megrant, Y. Chen, C. Quintana, B. Burkett, J. Kelly, E. Jeffrey, Y. Yang, A. Yu, K. Arya, R. Barends, Z. Chen, B. Chiaro, A. Dunsworth, A. Fowler, C. Gidney, M. Giustina, T. Huang, P. Klimov, M. Neeley, C. Neill, P. Roushan, D. Sank, A. Vainsencher, J. Wenner, T. C. White, and J. M. Martinis. Qubit compatible superconducting interconnects. *Quantum Science and Technology*, 3(1):014005, Nov. 2017. Publisher: IOP Publishing.

[38] A. Fuhrmanek, R. Bourgain, Y. R. P. Sortais, and A. Browaeys. Free-Space Lossless State Detection of a Single Trapped Atom. *Phys. Rev. Lett.*, 106(13):133003, Mar. 2011. Publisher: American Physical Society.

[39] A. Galda, M. Cubeddu, N. Kanazawa, P. Narang, and N. Earnest-Noble. Implementing a Ternary Decomposition of the Toffoli Gate on Fixed-Frequency Transmon Qutrits, Sept. 2021.

[40] J. Gambetta. Expanding the IBM Quantum roadmap to anticipate the future of quantum-centric supercomputing, 2022.

[41] K. Geerlings, Z. Leghtas, I. M. Pop, S. Shankar, L. Frunzio, R. J. Schoelkopf, M. Mirrahimi, and M. H. Devoret. Demonstrating a Driven Reset Protocol for a Superconducting Qubit. *Phys. Rev. Lett.*, 110(12):120501, Mar. 2013.

[42] P. Gokhale, J. M. Baker, C. Duckering, N. C. Brown, K. R. Brown, and F. T. Chong. Asymptotic improvements to quantum circuits via qutrits. In *Proceedings of the 46th International Symposium on Computer Architecture*. ACM, June 2019.

[43] P. Gokhale, S. Koretsky, S. Huang, S. Majumder, A. Drucker, K. R. Brown, and F. T. Chong. Quantum Fan-out: Circuit Optimizations and Technology Modeling, 2020.

[44] D. Gottesman. An introduction to quantum error correction and fault-tolerant quantum computation. In *Quantum information science and its contributions to mathematics, Proceedings of Symposia in Applied Mathematics*, volume 68, pages 13–58, 2010.

[45] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996. _eprint: quant-ph/9605043.

[46] S. Günther, N. A. Petersson, and J. L. DuBois. Quantum Optimal Control for Pure-State Preparation Using One Initial State. *arXiv:2106.09148 [quant-ph]*, Aug. 2021.

[47] L. Henriet, L. Beguin, A. Signoles, T. Lahaye, A. Browaeys, G.-O. Reymond, and C. Jurczak. Quantum computing with neutral atoms. *Quantum*, 4:327, Sept. 2020.

Publisher: Verein zur Forderung des Open Access Publizierens in den Quantenwissenschaften.

[48] A. Hill. Beyond Qubits: Unlocking the Third State in Quantum Processors, Dec. 2021. Publication Title: Rigetti Computing.

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

[50] Y. Hirata, M. Nakanishi, S. Yamashita, and Y. Nakashima. An efficient conversion of quantum circuits to a linear nearest neighbor architecture. *Quantum Information and Computation*, 11(1):142, 2011.

[51] A. A. Houck, J. Koch, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf. Life after charge noise: recent results with transmon qubits. *Quantum Information Processing*, 8(2-3):105–115, June 2009.

[52] S. S. Ivanov, H. S. Tonchev, and N. V. Vitanov. Time-efficient implementation of quantum search with qudits. *Physical Review A*, 85(6):062321, June 2012.

[53] D. Jaksch, J. Cirac, P. Zoller, S. Rolston, R. Côté, and M. Lukin. Fast quantum gates for neutral atoms. *Physical Review Letters*, 85(10):2208, 2000. Publisher: APS.

[54] S. Jandura. Improving a Quantum Compiler, Sept. 2018.

[55] J. Kelly. A Preview of Bristlecone, Google's New Quantum Processor, Mar. 2018. Publication Title: Google AI Blog.

[56] N. Khammassi, I. Ashraf, X. Fu, C. Almudever, and K. Bertels. QX: A high-performance quantum computer simulation platform. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 464–469, 2017.

[57] N. Khaneja, T. Reiss, C. Kehlet, T. Schulte-Herbrüggen, and S. J. Glaser. Optimal control of coupled spin dynamics: design of NMR pulse sequences by gradient ascent algorithms. *Journal of Magnetic Resonance*, 172(2):296–305, Feb. 2005.

[58] H. Kim, W. Lee, H.-g. Lee, H. Jo, Y. Song, and J. Ahn. In situ single-atom array synthesis using dynamic holographic optical tweezers. *Nature communications*, 7(1):1–8, 2016. Publisher: Nature Publishing Group.

[59] Y. Kim, A. Morvan, L. B. Nguyen, R. K. Naik, C. Jünger, L. Chen, J. M. Kreikebaum, D. I. Santiago, and I. Siddiqi. High-fidelity three-qubit iToffoli gate for fixed-frequency superconducting qubits. *Nature Physics*, 18(7):783–788, May 2022.

[60] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver. Superconducting qubits: Current state of play. *Annual Review of Condensed Matter Physics*, 11:369–395, 2020. Publisher: Annual Reviews.

[61] J. Koch, T. M. Yu, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf. Charge-insensitive qubit design derived from the Cooper pair box. *Phys. Rev. A*, 76(4):042319, Oct. 2007.

[62] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver. A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews*, 6(2):021318, June 2019. Publisher: AIP Publishing.

[63] R. Krishna, V. Makwana, and A. P. Suresh. A Generalization of Bernstein-Vazirani Algorithm to Qudit Systems, 2016.

[64] M. Kwon, M. F. Ebert, T. G. Walker, and M. Saffman. Parallel Low-Loss Measurement of Multiple Atomic Qubits. *Phys. Rev. Lett.*, 119(18):180504, Oct. 2017. Publisher: American Physical Society.

[65] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation. In *CGO*, pages 75–88, San Jose, CA, USA, Mar. 2004.

[66] H. Levine, A. Keesling, G. Semeghini, A. Omran, T. T. Wang, S. Ebadi, H. Bernien, M. Greiner, V. Vuletić, H. Pichler, and others. Parallel implementation of high-fidelity multiqubit gates with neutral atoms. *Physical review letters*, 123(17):170503, 2019. Publisher: APS.

[67] S. Lloyd and R. Maity. Efficient implementation of unitary transformations. *arXiv:1901.03431 [quant-ph]*, Jan. 2019.

[68] G. H. Low and I. L. Chuang. Hamiltonian Simulation by Qubitization. *Quantum*, 3:163, July 2019.

[69] E. Lucero, M. Hofheinz, M. Ansmann, R. C. Bialczak, N. Katz, M. Neeley, A. D. O'Connell, H. Wang, A. N. Cleland, and J. M. Martinis. High-Fidelity Gates in a Single Josephson Qubit. *Phys. Rev. Lett.*, 100(24):247001, June 2008.

[70] M. Luo and X. Wang. Universal quantum computation with qudits. *Science China Physics, Mechanics & Astronomy*, 57(9):1712–1717, Sept. 2014.

[71] D. Miller, T. Holz, H. Kampermann, and D. Bruß. Propagation of generalized Pauli errors in qudit Clifford circuits. *Physical Review A*, 98(5), Nov. 2018.

[72] P. Mundada, G. Zhang, T. Hazard, and A. Houck. Suppression of Qubit Crosstalk in a Tunable Coupling Superconducting Circuit. *Physical Review Applied*, 12(5), Nov. 2019. Publisher: American Physical Society (APS).

[73] P. Murali, J. M. Baker, A. J. Abhari, F. T. Chong, and M. Martonosi. Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers, 2019.

[74] P. Murali, D. M. Debroy, K. R. Brown, and M. Martonosi. Architecting Noisy Intermediate-Scale Trapped Ion Quantum Computers. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, pages 529–542. IEEE Press, 2020.

[75] P. Murali, D. C. McKay, M. Martonosi, and A. 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, Mar. 2020. arXiv:2001.02826 [quant-ph].

[76] A. Muthukrishnan and C. R. Stroud. Multivalued logic gates for quantum computation. *Phys. Rev. A*, 62(5):052309, Oct. 2000. Publisher: American Physical Society.

[77] P. J. Nair. Architectural Techniques to Enable Reliable and Scalable Memory Systems. *arXiv preprint arXiv:1704.03991*, 2017.

[78] A. Nersisyan, S. Poletto, N. Alidoust, R. Manenti, R. Renzas, C.-V. Bui, K. Vu, T. Whyland, Y. Mohan, E. A. Sete, S. Stanwyck, A. Bestwick, and M. Reagor. Manufacturing low dissipation superconducting quantum processors, 2019.

[79] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, New York, NY, USA, 10th edition, 2011.

[80] D. Ohl de Mello, D. Schäffner, J. Werkmann, T. Preuschoff, L. Kohfahl, M. Schlosser, and G. Birkl. Defect-Free Assembly of 2D Clusters of More Than 100 Single-Atom Quantum Systems. *Physical Review Letters*, 122(20), May 2019. Publisher: American Physical Society (APS).

[81] T. Patel, D. Silver, and D. Tiwari. Geyser: A Compilation Framework for Quantum Computing with Neutral Atoms. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, pages 383–395, New York, NY, USA, 2022. Association for Computing Machinery.

[82] T. Patel, E. Younis, C. Iancu, W. de Jong, and D. Tiwari. QUEST: Systematically Approximating Quantum Circuits for Higher Output Fidelity. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, pages 514–528, New York, NY, USA, 2022. Association for Computing Machinery. event-place: Lausanne, Switzerland.

[83] N. A. Petersson and F. Garcia. Optimal Control of Closed Quantum Systems via B-Splines with Carrier Waves, 2021. _eprint: 2106.14310.

[84] N. A. Petersson, F. M. Garcia, A. E. Copeland, Y. L. Rydin, and J. L. DuBois. Discrete Adjoints for Accurate Numerical Optimization with Application to Quantum Control, 2020. _eprint: 2001.01013.

[85] J. Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, Aug. 2018.

[86] I. B. M. Quantum. IBM Unveils Breakthrough 127-Qubit Quantum Processor. Publication Title: IBM Newsroom.

[87] 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. *Physical Review A*, 91(1), Jan. 2015. Publisher: American Physical Society (APS).

[88] S. Resch, A. Gutierrez, J. S. Huh, S. Bharadwaj, Y. Eckert, G. Loh, M. Oskin, and

S. Tannu. Accelerating Variational Quantum Algorithms Using Circuit Concurrency, 2021.

[89] Rigetti. Rigetti Computing Announces Commercial Availability of 80-Qubit Aspen-M System and Results of CLOPS Speed Tests, Feb. 2022.

[90] M. Ringbauer, M. Meth, L. Postler, R. Stricker, R. Blatt, P. Schindler, and T. Monz. A universal qudit quantum processor with trapped ions. Technical Report arXiv:2109.06903, arXiv, Sept. 2021.

[91] L. Ruiz-Perez and J. C. Garcia-Escartin. Quantum arithmetic with the quantum Fourier transform. *Quantum Information Processing*, 16(6):152, 2017. Publisher: Springer.

[92] M. Saffman. Quantum computing with atomic qubits and Rydberg interactions: progress and challenges. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 49(20):202001, 2016. Publisher: IOP Publishing.

[93] L. M. Seifert, J. Chadwick, A. Litteken, F. T. Chong, and J. M. Baker. Time-Efficient Qudit Gates through Incremental Pulse Re-seeding, 2022.

[94] S. Sheldon, E. Magesan, J. M. Chow, and J. M. Gambetta. Procedure for systematically tuning up cross-talk in the cross-resonance gate. *Physical Review A*, 93(6):060302, June 2016.

[95] V. V. Shende and I. L. Markov. On the CNOT-cost of TOFFOLI gates. 2008.

[96] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 1995. _eprint: arXiv:quant-ph/9508027 Published: SIAM J.Sci.Statist.Comput. 26 (1997) 1484.

[97] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, Oct. 1997.

[98] K. Singh, S. Anand, A. Pocklington, J. T. Kemp, and H. Bernien. A dual-element, two-dimensional atom array with continuous-mode operation, 2021. _eprint: 2110.05515.

[99] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan. t|ket>: a retargetable compiler for NISQ devices. *Quantum Science and Technology*, 6(1):014003, Jan. 2021.

[100] S. E. Sklarz and D. J. Tannor. Loading a Bose-Einstein condensate onto an optical lattice: An application of optimal control theory to the nonlinear Schr\textbackslash"odinger equation. *Physical Review A*, 66(5):053619, Nov. 2002.

[101] R. S. Smith, M. J. Curtis, and W. J. Zeng. A practical quantum instruction set architecture. *arXiv preprint arXiv:1608.03355*, 2016.

[102] S. Stein, N. Wiebe, Y. Ding, P. Bo, K. Kowalski, N. Baker, J. Ang, and A. Li. EQC: ensembled quantum computing for variational quantum algorithms. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 59–71, New York New York, June 2022. ACM.

[103] A. Taheri Monfared, M. Haghparast, and K. Datta. Quaternary Quantum/Reversible Half-Adder, Full-Adder, Parallel Adder and Parallel Adder/Subtractor Circuits. *International Journal of Theoretical Physics*, 58(7):2184–2199, July 2019.

[104] S. S. Tannu and M. 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, 2019.

[105] S. Terzi and A. Gençten. Development of some two and three qutrit quantum logic gates. *AIP Conference Proceedings*, 2178(1):030027, 2019. _eprint: https://aip.scitation.org/doi/pdf/10.1063/1.5135425.

[106] J. Tilly, H. Chen, S. Cao, D. Picozzi, K. Setia, Y. Li, E. Grant, L. Wossnig, I. Rungger, G. H. Booth, and J. Tennyson. The Variational Quantum Eigensolver: a review of methods and best practices, 2021.

[107] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[108] J. J. Vartiainen, M. Möttönen, and M. M. Salomaa. Efficient Decomposition of Quantum Gates. *Physical Review Letters*, 92(17), Apr. 2004. Publisher: American Physical Society (APS).

[109] P. Wang, C.-Y. Luan, M. Qiao, M. Um, J. Zhang, Y. Wang, X. Yuan, M. Gu, J. Zhang, and K. Kim. Single ion qubit with estimated coherence time exceeding one hour. *Nature Communications*, 12(1):233, Jan. 2021.

[110] Y. Wang, Z. Hu, B. C. Sanders, and S. Kais. Qudits and High-Dimensional Quantum Computing. *Frontiers in Physics*, 8:479, 2020.

[111] 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. IEEE, 2011.

[112] R. Wille, L. Burgholzer, and A. 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)*, pages 1–6. IEEE, 2019.

[113] R. Wille, O. Keszocze, M. Walter, P. Rohrs, A. Chattopadhyay, and R. Drechsler. Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits.

In *2016 21st Asia and South Pacific design automation conference (ASP-DAC)*, pages 292–297. IEEE, 2016.

[114] K. Wright, K. Beck, S. Debnath, J. Amini, Y. Nam, N. Grzesiak, J.-S. Chen, N. Pisenti, M. Chmielewski, C. Collins, and others. Benchmarking an 11-qubit quantum computer. *Nature communications*, 10(1):1–6, 2019. Publisher: Nature Publishing Group.

[115] X.-C. Wu, S. Di, E. M. Dasgupta, F. Cappello, H. Finkel, Y. Alexeev, and F. T. Chong. Full-State Quantum Circuit Simulation by Using Data Compression. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[116] E. Younis, C. C. Iancu, W. Lavrijsen, M. Davis, E. Smith, and USDOE. Berkeley Quantum Synthesis Toolkit (BQSKit) v1, Apr. 2021.