

THE UNIVERSITY OF CHICAGO

ARCHITECTURAL DESIGN FOR EMERGING QUANTUM TECHNOLOGIES

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
JONATHAN M. BAKER

CHICAGO, ILLINOIS

DECEMBER 2022

Copyright © 2022 by Jonathan M. Baker
All Rights Reserved

TABLE OF CONTENTS

LIST OF FIGURES	vi
LIST OF TABLES	xxix
ACKNOWLEDGMENTS	xxxii
ABSTRACT	xxxii
1 INTRODUCTION	1
2 DETERMINING THE RIGHT ABSTRACTIONS: MULTIVALUED QUANTUM LOGIC	7
2.1 Relevant Background	9
2.2 Prior Work in Multivalued Quantum Logic	13
2.2.1 Qudits	13
2.2.2 Generalized Toffoli Gate	13
2.3 Circuit Constructions	15
2.3.1 Key Intuition	15
2.3.2 Generalized Toffoli Gate	16
2.3.3 Larger Arithmetic Circuits: The Incrementer	17
2.3.4 Larger Arithmetic Circuits: The A + B Adder	18
2.3.5 Larger Arithmetic Circuits: Constant, +K, Addition	21
2.4 Application to Algorithms	26
2.4.1 Artificial Quantum Neuron	26
2.4.2 Grover’s Algorithm	27
2.4.3 Arithmetic Circuits and Shor’s Algorithm	27
2.4.4 Error Correction and Fault Tolerance	27
2.5 Simulator for Verification of Constructions	28
2.5.1 Noise Simulation	28
2.5.2 Simulator Efficiency	30
2.6 Simulation and Error Models	32
2.6.1 Generic Noise Model	32
2.6.2 Superconducting QC	33
2.6.3 Trapped Ion $^{171}\text{Yb}^+$ QC	34
2.7 Simulation Results	35
2.8 Qubit-Qudit Compression	36
2.8.1 Motivation	36
2.8.2 Compression Schemes	37
2.8.3 Decompositions with Compression	40
2.9 Discussion	45

3	ARCHITECTURAL TRADE-OFFS IN EMERGING TECHNOLOGY: NEUTRAL ATOM ARCHITECTURES	60
3.1	Introduction	60
3.2	Relevant Background	63
3.2.1	Quantum Computation and the Gate Model	63
3.2.2	The Quantum Compilation Problem	65
3.2.3	Neutral Atoms	66
3.3	Neutral Atom Compiler and Methodology	66
3.3.1	Mapping, Routing, and Scheduling	66
3.3.2	Benchmarks	69
3.3.3	Experimental Setup	69
3.4	Unique Advantages of Neutral Atom Architectures	70
3.4.1	Long Range Interactions	70
3.4.2	Native Multiqubit Gates	73
3.5	Error Analysis of Neutral Atom Architectures	75
3.6	Unique Challenge: Sporadic Atom Loss	78
3.7	Remarks	83
4	APPLICATION-GUIDED ARCHITECTURAL DESIGN. VIRTUALIZING ERROR CORRECTED QUBITS	95
4.1	Introduction	95
4.2	Relevant Background	99
4.2.1	Superconducting Qubit Architectures	99
4.2.2	Qubit Memory Technology	99
4.2.3	Quantum Errors	101
4.2.4	Surface Codes, Error Decoding, and Lattice Surgery	103
4.3	Virtualized Logical Qubits	105
4.3.1	Natural Surface Code Embedding	106
4.3.2	Transversal CNOT	109
4.3.3	Compact Surface Code Embedding	110
4.3.4	Architectural Considerations	112
4.4	Evaluation	114
4.4.1	Error Model and Noise Assumptions	114
4.4.2	Experimental Setup	115
4.5	Error Threshold Results	116
4.6	Error Sensitivity Results	117
4.7	Magic State Distillation Resource Estimates	119
4.8	Conclusion	120

5	EVALUATING ARCHITECTURES AT THEIR LIMITS: IMPROVED COMPILATION METHODS	127
5.1	Memory-Equipped Quantum Architectures: The Power of Random Access	128
5.1.1	Relevant Background	131
5.1.2	A Memory-Equipped Quantum Architecture	134
5.1.3	Experimental Setup	144
5.1.4	Results and Discussion	147
5.2	Remarks	156
5.3	Time-Sliced Quantum Circuit Partitioning for Modular Architectures	157
5.3.1	Relevant Background	161
5.3.2	Mapping Qubits to Clusters	163
5.3.3	Lookahead Weights	167
5.3.4	Experimental Setup	171
5.3.5	Results and Discussion	174
5.3.6	Remarks	178
5.4	Noise-Adaptive Compiler Mappings for Noisy Intermediate Scale Quantum Computers	179
5.4.1	Relevant Background	181
5.4.2	Overview of our Compilation Framework	183
5.4.3	Optimal Compilation: Detailed Approach	188
5.4.4	Heuristic Compilation	194
5.4.5	Experimental Setup	196
5.4.6	Optimizing Execution Reliability	198
5.4.7	Remarks	204
5.5	Orchestrated Trios: Compiling for Efficient Communication in Quantum Programs with 3-Qubit Gates	206
5.5.1	Relevant Background	210
5.5.2	Orchestrated Trios	213
5.5.3	Evaluation	216
5.5.4	Results and Discussion	221
5.5.5	Remarks	227
6	DISCUSSION AND CONCLUSION	228
	REFERENCES	232

LIST OF FIGURES

2.1	Reversible AND circuit using a single ancilla bit. The inputs are on the left, and time flows rightward to the outputs. This AND gate is implemented using a Toffoli (CCNOT) gate with inputs q_0 , q_1 and a single ancilla initialized to 0. At the end of the circuit, q_0 and q_1 are preserved, and the ancilla bit is set to 1 if and only if both other inputs are 1.	10
2.2	The five nontrivial permutations on the basis elements for a qutrit. (Left) Each operation here switches two basis elements while leaving the third unchanged. These operations are self-inverses. (Right) These two operations permute the three basis elements by performing a $+1 \pmod 3$ and $-1 \pmod 3$ operation. They are each other's inverses.	11
2.3	A Toffoli decomposition via qutrits. Each input and output is a qubit. The red controls activate on $ 1\rangle$ and the blue controls activate on $ 2\rangle$. The first gate temporarily elevates q_1 to $ 2\rangle$ if both q_0 and q_1 were $ 1\rangle$. We then perform the X operation only if q_1 is $ 2\rangle$. The final gate restores q_0 and q_1 to their original state.	16
2.4	Our circuit decomposition for the Generalized Toffoli gate is shown for 15 controls and 1 target. The inputs and outputs are both qubits, but we allow occupation of the $ 2\rangle$ qutrit state in between. The circuit has a tree structure and maintains the property that the root of each subtree can only be elevated to $ 2\rangle$ if all of its control leaves were $ 1\rangle$. Thus, the U gate is only executed if all controls are $ 1\rangle$. The right half of the circuit performs uncomputation to restore the controls to their original state. This construction applies more generally to any multiply-controlled U gate. Note that the three-qutrit gates are decomposed into 6 two-qutrit and 7 single-qutrit gates in our actual simulation, as based on the decomposition in [53].	48

2.5	Our circuit decomposition for the Incrementer. At each step in the design, multiply-controlled gates using the decomposition in Figure 2.4 are used to efficiently propagate carries over half of the subcircuit. The $ 2\rangle$ control checks for carry generation and the chain of $ 1\rangle$ controls check for carry propagation. The circuit depth is $\log^2 N$, which is only possible because of our log depth multiply-controlled gate primitive.	49
2.6	The Cuccaro adder of [46] with the Toffoli gates replaced by our efficient decomposition. This only reduces the total depth of the circuit by a constant amount, i.e. no asymptotic benefit is obtained. There are several simplifications which can be made to this circuit, most notably the controlled X_{-1} followed immediately by a controlled X_{+1} in several places. For clarity we've kept these gates to see the direct replacement of our Toffoli decomposition into existing circuits.	49
2.7	Decomposing the A+B circuit with intermediate qutrits. We take as input two n qubit registers and output the sum S onto the bits of register B while leaving register A unchanged. In this approach, we first decide if there will be a carry generated on the top half of the inputs. If so, we apply a +1 gate to the bottom half of the inputs (specifically on the bits of B) and then recursively add the first half of A and B and second half of A and B in parallel. The carry circuit outputs an encoded carry status on $a_{n/2}, b_{n/2}$. The controlled +1 can be implemented by just modifying the initial X_{+1} and final X_{02} gates of the incrementer to be controlled by the final output carry status. This decomposition is $O(\log^3 n)$ depth provided the Carry circuit is $O(\log n)$ depth and the incrementer is $O(\log^2 n)$ depth.	50

2.8	On the left is the encoding for generate (g), propagate (p), and kill (k) carry statuses. On the right is the result of combining two input carry statuses C_i and C_j with C_i corresponding to the less significant bits. Notice the order matters, e.g. $k + g = g$ but $g + k = k$	50
2.9	Realization of the truth table of Table 2.2 as the gadget “Combine Carry Status” (CCS). While this gate is expensive in terms of two qutrit gates, it is constant depth.	51
2.10	Realizing the Carry operation in sublinear depth for $n = 4$ inputs using the CCS gadget. The result is a (11) if and only if a carry is generated, while leaving all of the remaining bits as junk, possibly in ternary states. The CCS blocks always take four qubits as inputs, the first and last two bits and output a binary output on the last two inputs. In the context of the $A + B$ adder, we take this output carry status and use it to control an incrementer on the more significant bits. Afterwards, we would apply the inverse of the cascade on the right to return to the original inputs; this step is omitted here.	51
2.11	The carry (C) and UnCompute and Add (UCA) gadgets used for linear $+K$ adder circuit. There are several instances which are a function of the specific k_i and k_{i+1} values.	52
2.12	Linear $+K$ adder, with carry out on a register of size 4. We assume $k_0 = 1$ and we use the gadgets of Figure 2.11. For no carry out, i.e. $+K \bmod 2^{ K }$ simply omit the final C component in the cascade. Note a X_{+1} on an ancilla has the same effect as a X_{01} allowing us to use the same C gadget.	52
2.13	The Decomposition of the $+K$ circuit into a sequence of $+K_i$ circuits, for $i \in [0, M]$, where M is a constant.	53

2.14	The decomposition of the $+K_i$ blocks of Figure 2.13 in $O(\log^3 n)$ depth. Notice again, the controlled incrementer is done by adding two controls, the carry status output from the Carry circuit to the first and final gate of the incrementer. The truth table for this transformation has been omitted for simplicity. . . .	53
2.15	The Prepare Carry (PC) circuit for when $\beta_i = 0$. This takes four bits of input A and the known constant β_i and outputs on the last two bits the carry status for this group of bits on its own, adding $(a_i a_{i+1} a_{i+2} a_{i+4}) + (0000)$. The other two inputs are left in possibly ternary states.	53
2.16	The Prepare Carry (PC) circuit for when $\beta_i = 1$. This takes four bits of input A and the known constant β_i and outputs on the last two bits the carry status for this group of bits on its own, adding $(a_i a_{i+1} a_{i+2} a_{i+4}) + (1000)$. The other two inputs are left in possibly ternary states. The truth table for this transformation has been omitted for simplicity.	53
2.17	Similar to the Combine Carry Circuit (CCS) of Figure 2.9, the CCS_{+K} gadget combines carry statuses of the type found in Table 2.3. This gadget always leaves the final two inputs as the new carry status while leaving the other two inputs possibly in ternary states. The truth table for this transformation has been omitted for simplicity.	54
2.18	Using the PC and CCS_{+K} gadgets, we can produce an $O(\log n)$ depth Carry circuit for the $+K_i$ circuit.	54
2.19	Each iteration of Grover Search has a multiply-controlled Z gate. Our logarithmic depth decomposition, reduces a $\log M$ factor in Grover's algorithm to $\log \log M$	54

2.20	This Moment comprises three gates executed in parallel. To simulate with noise, we first apply the ideal gates, followed by a gate error noise channel on each affected qudit. This gate error noise channel depends on whether the corresponding gate was single- or two- qudit. Finally, we apply an idle error to every qudit. The idle error noise channel depends on the duration of the Moment .	55
2.21	Exact circuit depths for all three benchmarked circuit constructions for the N-controlled Generalized Toffoli up to $N = 200$. Both QUBIT and QUBIT+ANCILLA scale linearly in depth and both are bested by QUTRIT 's logarithmic depth.	55
2.22	Exact two-qudit gate counts for the three benchmarked circuit constructions for the N-controlled Generalized Toffoli. All three plots scale linearly; however the QUTRIT construction has a substantially lower linearity constant.	56
2.23	Circuit simulation results for all possible pairs of circuit constructions and noise models. Each bar represents 1000+ trials, so the error bars are all $2\sigma < 0.1\%$. Our QUTRIT construction significantly outperforms the QUBIT construction. The QUBIT+ANCILLA bars are drawn with dashed lines to emphasize that it has access to an extra ancilla bit, unlike our construction.	56
2.24	The compression of 3 qubits into 2 qutrits and an ancilla, $ 0\rangle$. All +1 gates are done modulo 3. Using a sequence of qutrit gates, we can transform three input qubits into the desired ancilla. When A, B and C are not going to be used for a long time in the circuit, they can be temporarily repurposed as an ancilla bit elsewhere in the circuit. When we want to operate on these stored bits, we run the inverse of this circuit using <i>any</i> ancilla for the third qubit.	57

- 2.25 The compression of 2 qubits into a single ququart and generating an ancilla, $|0\rangle$. The +2 gate here is done modulo 4. This operation takes as input two qubits, A and B, and produces a single ququart and an ancilla $|0\rangle$. To do this, we need only 3 two-ququart gates. Similarly, to retrieve the stored information, we can do the inverse of this operation using *any* ancilla for the second qubit. 57
- 2.26 An adder circuit from [58] on two four-bit registers A and B with a carry-out bit using ancilla. The sum S is computed in-place on register B while A is untouched and the ancilla are restored to $|0\rangle$. We use this as a sub-component of our general decomposition. Each of the ancilla in this circuit can be generated from other input qubits not shown here via our compression circuits. Part a of the circuit computes carry generate and propagate for each bit position. Part b computes the carry-in for every bit position. Part c does the addition, storing the output in register B . Parts d and e uncompute b and a respectively, restoring the ancilla back to $|0\rangle$ 58
- 2.27 Our $A + B$ adder that uses no external ancilla. The variant shown here for $c = 5$ uses 2-3-1 compression to generate one ancilla (marked as $|0\rangle$) for every three unused qubits, storing their values in two qutrits (marked as $d = 3$). A box is drawn around every $(A + B)_2$ and Undo carry gate to indicate that they use all the generated ancilla across the circuit. $c_{out,i}$ or $c_{in,i}$ is included on some of the gates to indicate when the carry-in and carry-out versions are used and on which ancilla the carry-out is stored. The SWAP gates (pairs of \times in the diagram) simply move a carry-out bit to another ancilla where it is used as the next carry-in. The two blocks of gates shown with dashed lines are repeated $c - 2 = 3$ times along the diagonal indicated. If 2-4-1 compression is used, an ancilla is generated for every two unused qubits so only $c = 4$ blocks are needed. The depth of this circuit is $O(\log n)$ 59

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.	61
3.2	A quantum circuit with a 1, 2, and 3 qubit gate translated to interactions on a NA device. These systems allow the execution of multiqubit gates. For 2 and 3 qubit gates the interacting qubits are excited to Rydberg states. Interactions are possible if all interacting qubits are closer than the maximum interaction distance.	64
3.3	Post-compilation gate count across benchmarks. On the top 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 bottom 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.	86

3.4	Post-compilation depth across all benchmarks. On the top, the reduction in depth over the distance 1 baseline. Each bar is the average over all benchmark sizes. On the bottom 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.	87
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 bottom 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.	88
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.	89

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.	90
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.	91
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 the qubits can be swapped to a valid interaction configuration, and returned for rerouting strategies. Numbers indicate the order of swaps.	91

3.13	Sensitivity to the rate of atom loss for the balanced <i>Compile Small and Reroute</i> 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.	93
3.14	A timeline of 20 successful shots for <i>Compile Small and Reroute</i> 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.	94
4.1	Our fault-tolerant architecture with random-access memory local to each transmon. On top is the typical 2D grid of transmon qubits. Attached below each data transmon is a resonant cavity storing error-prone data qubits (shown as black circles). This pattern is tiled in 2D to obtain a 2.5D array of logical qubits. Our key innovation here is storing the qubits that make up each logical qubit (shown as checkerboards) across many cavities to enable efficient computation.	97
4.2	A typical 2D superconducting qubit architecture. The dots are transmon qubits where black are used as data and gray are used as ancilla for error correction. The lines indicate physical connections between qubits that allow operations between them. Four logical qubits, each consisting of 9 error-prone data qubits, are shown here in the rotated surface code with distance 3. Z parity checks are shaded yellow (light) and X parity checks are shaded blue (dark) where checks on only 2 data are drawn as half circles.	100

4.3	A close-up representation of the qubit memory technology we use. On top is a superconducting transmon qubit physically connected to a resonant superconducting cavity. This cavity has many resonant modes each used to store a qubit. These qubits can be loaded and stored (with random access) via the transmon.	101
4.4	The lattice surgery operations to perform a logical CNOT on the standard surface code (and directly supported in our architecture). Given control and target qubits $ C\rangle$ and $ T\rangle$, a CNOT is performed by enabling and disabling the parity checks as shown across 6 timesteps ((e) is two steps). We show this complex process to contrast with the fast transversal CNOT enabled by our architecture (described later in Section 4.3.2).	103
4.5	Circuit showing how to execute our Natural embedding on hardware. Left: The layout of eight data (black) and two ancilla (gray) in hardware. CNOT operations between qubits are drawn between. Right: A circuit diagram of the operations applied over time where each horizontal line corresponds to a qubit and each box or symbol is an operation. The steps are L_z : load from memory mode z , $ 0\rangle$: reset ancilla, CNOTs: compute the Z or X parity, Meter: measure the result, S_z : store back to memory.	106
4.6	The transversal CNOT enabled by our 2.5D architecture. The data qubits for the control logical qubit are loaded into the transmons. Transmon-mediated CNOTs to mode z for every data qubit perform the logical operation. This takes one timestep to perform, 6x better than a lattice surgery CNOT. . . .	122

4.7	Transformation from Natural to Compact. (a) Natural embedding: Only data have attached cavities (not shown). (b) The transformation: Z ancilla (over yellow/light areas) merge with the upper-right data transmon and X ancilla (over blue/dark areas) merge with the lower-left data transmon. The opposite pairings are key to keeping 4-way grid connectivity. (c) Compact embedding: All ancilla transmons without attached cavities have been removed. All remaining transmons have cavities and are used as both data and ancilla.	122
4.8	A 3D view of our Compact embedding. Shown at the top is the 2D grid of transmon qubits. Attached below every transmon is a resonant cavity. Compact surface code patches are shown stored, one in each mode. This deformed patch can be tiled in 2D.	123
4.9	The Compact lattice surgery operations to perform a CNOT. The logical operations performed are identical to Figure 4.4 but the corresponding physical operations are arranged as shown in Figure 4.7. This uses half as many transmons as Natural. As before, it takes 6 timesteps of d error correction cycles each.	123
4.10	The CNOT sequence for parity checks in Compact. Left: A quantum circuit showing the hardware operations over time. Right: The CNOT execution order repeats $A_0D_2, A_1D_3, A_2C_0, A_3C_1, B_0C_2, B_1C_3, B_2D_0, B_3D_1$. The AB and CD sequences run in parallel but offset to ensure ancilla and data use do not conflict. CNOTs for A_0D_2 are marked in red where an isolated circle indicates a transmon-mediated CNOT.	123

4.11	Error thresholds for the baseline 2D architecture and Natural and Compact variants of our 2.5D architecture. The thresholds are comparable to the baseline indicating the space savings obtained in our system does not substantially reduce the error thresholds. The slopes of the lines in this figure indicate, post-threshold, how much improvement in physical error rates improve logical error rate. Except for the baseline, all use a cavity size of 10.	124
4.12	Sensitivity of logical error rate to various error sources in Compact, Interleaved. The logical error rates are most sensitive to physical error of Loads/Stores and SC-SC gates. The logical error rate is less sensitive to the coherence times and mostly insensitive to effects of load-store duration and cavity size.	125
4.13	(a) The T-state generation rates of three different circuits. Higher generation rate is better. (b) The space, in terms of number of patches, required to produce a single $ T\rangle$ per time step. Lower is better. Fast [124] and Small [123] work in the surface code and do not use memory. VQubits is implemented with transversal CNOTs in our 2.5D architecture. All are based on [29].	126
5.1	Average qubit distance on several instances of near-term target architectures. Average qubit distance approximates how many SWAPs are necessary to interact an arbitrary pair of qubits. LNN architectures scale extremely poorly in this metric resulting in a large number of added gates and depth. 2D and MEQC architectures scale much better. We show this translates into reduced number of gates and reduced depth as we scale into the future.	128

5.2	Both current and the proposed MEQC device. On the left is a LNN device where adjacent superconducting transmons are coupled enabling two qubit interactions. In the center is a 2D mesh architecture common among current manufacturers. On the right is the proposed MEQC architecture with transmons arranged in a line. Each transmon has an attached cavity which stores in memory multiple qubits. To operate on the qubits, they must first be loaded into the transmons.	129
5.3	Compiling to near-term devices is a multi-step process. First we map the logical, circuit qubits to the physical hardware qubits. Based on this placement and the input program, we insert SWAPs in order to interact distant qubits. Here we compile a simple quantum program, Bernstein-Vazirani, to a 4 qubit LNN architecture. Quantum programs, like the input program on the left are a sequence of gates specified on qubits. In this example, based on the given mapping, a pair of SWAPs are required to execute a CNOT between $ \Psi_3\rangle$ and $ \Psi_4\rangle$	132
5.4	Compiling a small program to a MEQC device. In this case we map the input qubits $ \Psi_1\rangle$ and $ \Psi_2\rangle$ to one of the two available cavity modes. When executing the gates, we first execute a Load to move the qubit to the connected transmon. The gate is then executed and the qubit is returned to its original mode via a Store. We represent Loads as $L - L$ and Stores as $S - S$	136
5.5	The scaling of depth and gate count in a subset of our benchmarks. LNN-50 is omitted because it adds substantially more gates than both of these architectures and as such is not competitive. In many cases, 2D-5-10 is competitive with the proposed architecture, however, clear separation emerges in all cases.	143

5.6	Post-compilation number of gates and circuit depth for 50 qubit input programs on all benchmarks. In every case, MEQC with transmons arranged with as a chain or a mesh shows improvement over a more standard 2D mesh qubit arrangement. The increase in gate count in MEQC architectures is approximately 60% due to loads and stores and the rest from SWAPs. By requiring fewer gates, we reduce the possibility of gate-induced error.	148
5.7	Output fidelity for full density simulations of the QFT Adder on 2-10 qubits. Even with more gates in these small instances, programs compiled to MEQC devices are competitive, and at 10 qubits we see the start of advantage.	149
5.8	An estimation of if no gate errors occur in small program instances. As noted, this only accounts for errors due to one and two qubit gate errors and is not influenced by decoherence errors. Larger is better and in general programs compiled to our proposed architecture are competitive or better than a 2D architecture. We expect with better T1 times and better gate and depth scaling, our architecture will outperform, by increasing the likelihood of successful execution, by a larger margin as programs scale and gate errors improve. All data points were obtained by running 8000 trials of the input compiled input program.	150
5.9	With 100x better gates, we begin to see the effect of improved compilation. Specifically, by reducing the total number of gates required for execution on the proposed MEQC devices we reduce the probability of a program failing due to gate errors. Furthermore, with substantially longer T1 times in cavity, qubits stored in memory are protected from decoherence errors. All data points are obtained by running 8000 trials of the input program compiled to the two target architectures.	151

5.10	Gates and depth of 20 qubit QFT Adder compiled to MEQC architectures with different transmon connectivity and varying cavity sizes. As the number of qubits per cavity increases, we expect the average qubit distance to be reduced meaning fewer SWAPs necessary. However, in MEQC devices operations on qubits in the same cavity cannot be done in parallel. Therefore, we expect lose some degree of parallelism, hence the increase in depth.	154
5.11	Crossover points for various interconnect error rates of the QFT-Adder benchmark. Interconnect in MEQC devices may not be as good as SWAPs in traditional architectures. We study how much interconnect error we can tolerate in the NISQ target of 100 qubit devices with 10^{-5} two qubit error rates. We find we can tolerate up to 12x worse interconnect errors, provided programs of at least size 52.	155
5.12	Non-local communication overhead in circuits mapped to cluster-based machines. Our new mapping scheme FPG-rOEE reduces the number of operations added for non-local communication on all benchmarks.	159
5.13	An example modular architecture of qubits in individual ion traps connected with optics proposed by Monroe et al [132]. Communication between traps is supported by photon-mediated entanglement. Similar communication for superconducting qubits [40] can facilitate modular architectures for that technology.	160

5.14	(Top) An example of a quantum program with single-qubit gates not shown. The inputs are on the left and time flows to the right toward the outputs. The two-qubit operations here are CNOT (controlled-NOT). (Bottom) The graph representations of the quantum circuit of the above circuit. On the far left is the total interaction graph where each edge is weighted by the total number of interactions for the whole circuit. To the right is the sequence of time slice graphs, where an edge is only present if the qubits interact in the time slice. The sum of all time slice graphs is the total interaction graph.	161
5.15	An example of a time slice graph with lookahead weights based on the circuit in Figure 5.14. We take the graph from the left and add weight to the edges of qubits that interact in the future. In this case, we take the weight equal to the number of times the qubits will interact in the future.	165
5.16	The effect of different lookahead functions with various σ on non-local communication in the Cuccaro adder, a very regular circuit, with 76 data and 24 ancilla qubits using FGP-rOEE. We see the exponential function outperforms the others for a circuit of highly regular structure.	168
5.17	The non-local communication, measured in number of operations between clusters added, for our representative benchmark circuits mapped by each FGP-rOEE using different lookahead functions, each with $\sigma = 1$. The x-axis is the number of input/output qubits. The remainder are used as ancilla for clean multi-control. The exponential function is better on all instances of Clean multi-control and Cuccaro adder, and there is no substantial advantage of one function over the others in the random circuit.	169

5.18	The non-local communication overhead for our benchmark circuits mapped by each mapping algorithm. The x-axis is the number of qubits that are used in the circuit. The y-axis is the number of non-local communication operations inserted to make the circuit executable in our hardware model. In Clean multi-control, Clean multi-target, and Dirty multi-target, the remainder of the 100 qubits are used as ancilla (clean or dirty determined by the circuit name). FGP-rOEE outperforms all other mapping algorithms on all but the multi-target circuits, and shows substantial improvement over the static baseline. As the size of the circuit increases, rOEE tends to outperform by a greater margin, indicating scales better into the future.	176
5.19	Daily variations in qubit coherence time (larger is better) and gate error rates (lower is better) in <i>IBMQ 16 Rueschlikon</i> . The qubits and gates that are most or least reliable are different across days.	180
5.20	Figure (a) shows the intermediate representation of the Bernstein-Vazirani algorithm on 4 qubits (BV4). Each qubit is represented by a line. X and H are single qubit gates. The CNOT gates from each qubit $p_{0,1,2}$ to p_3 are marked by vertical connectors. The measurement or readout operation is indicated by the meter. Figure (b) shows a mapping where qubit movement is required. The numbers on the labelled edges indicate the CNOT gate error ($\times 10^{-2}$). In this mapping, an error-prone CNOT is used. Figure (c) shows an optimized mapping where qubit movement is not required and unreliable hardware CNOTs (crossed) and unreliable qubits (hatched) are avoided. . . .	183

5.21	Optimization Pipeline. Inputs are a QC program, details about the specific hardware configuration, and a set of options, such as routing policy and solver approach. From these, compiler generates a set of appropriate constraints and uses them to map program qubits to hardware qubits and schedule operations. Finally, the compiler generates an executable version of the program, here for <i>IBMQ16</i>	184
5.22	Two routing policies for swap-based architectures.	191
5.23	Measured success rate of R-SMT★ compared to Qiskit and T-SMT★. (Of 8192 trials per execution, success rate is the percentage that achieve the correct answer in real-system execution. R-SMT★ obtains higher success rate than Qiskit because it simultaneously adapts placement according to dynamic error rates and avoids unnecessary qubit movement.	198
5.24	Executions of three benchmarks for 1 week. R-SMT★ is more resilient to errors compared to T-SMT★. Similar trends for other benchmarks.	199
5.25	Measured success rate, execution duration and compile time for three representative benchmarks. T-SMT★ which directly optimizes for execution duration obtains the minimum execution durations, but R-SMT★ with $\omega = 0.5$ is close, and more resilient to errors (higher reliability). All benchmarks compile in less than 1 minute.	200
5.26	For real data/experiment, on <i>IBMQ16</i> , qubit mappings for three optimization objectives, varying the type of noise-awareness. In each figure, the edge labels indicate the CNOT gate error rate ($\times 10^{-2}$), and the numbers inside each node indicate that qubit's readout error rate ($\times 10^{-2}$). (a), T-SMT★ uses an unreliable hardware CNOT between p_3 and p_0 . (b) Program qubits are placed on the best readout qubits, but p_0 and p_3 communicate using swaps. (c) Best CNOTs and readout qubits are used.	201

5.27	Effect of gate durations, routing policy and objective function on execution duration. Although reliability is our primary objective, several variants perform well on run time as well. T-SMT★(either RR or OBP) has the best execution duration, but R-SMT★is very close in run time and offers better success rates. Noise-aware policies, R-SMT★and T-SMT★, are 1.6x better than T-SMT.	202
5.28	Noise-aware Heuristics: GreedyE★ heuristic mapping offers reliability comparable to R-SMT★on most benchmarks.	203
5.29	Scalability of optimal and heuristic methods on synthetic benchmarks. Each line represents a qubit count.	204
5.30	Example routing from Qiskit (a) vs. Trios (b) for a single Toffoli operation. Circles represent qubits and lines indicate two qubits are connected. Input qubits are highlighted in red. SWAP arrows are labeled by timestep. The routed locations for Trios routing are highlighted in green while Qiskit moves them several times. Qiskit adds 16 SWAPs (=48 CNOTs), some during the Toffoli, while Trios adds only 7 SWAPs (=21 CNOTs) all before the Toffoli. Performing multiple passes of decomposition allows direct routing and enables this huge reduction in communication, increasing the probability of program success.	207
5.31	(a) Typical compilation passes used by Qiskit (simplified). (b) Trios compilation passes.	209
5.32	A 6-CNOT decomposition of the Toffoli gate.	211
5.33	An 8-CNOT decomposition of the Toffoli gate.	211

5.34	Example topologies of near-term quantum devices. Orange (a): IBM Johannesburg. Yellow (b): 2D Grid. Purple (c): four groups of five fully connected clusters. Green (d) Linear. Our real experiments run on Johannesburg and our simulations explore all of these topologies. Colors correspond with the bars in Figures 5.38, 5.39, 5.40.	212
5.35	Success probabilities of Toffoli gates between random triplets of qubits. Higher is better. The x labels specify the three qubits and total swap distance. The geometric mean success rates for each compiler are 41%, 35%, 47%, and 50% respectively. Trios (8-CNOT) improves average success rate by 23% vs. the Qiskit baseline.	219
5.36	Total number of two-qubit (CNOT) gates required to execute a Toffoli gate between various distant qubits. Lower is better. The x labels specify the three qubits and total swap distance. The geometric mean gate counts for each compiler are 29, 28, 23, and 19 respectively. Trios (8-CNOT) reduces average gate count by 35%.	219
5.37	Normalized success probabilities of Toffoli gates between triplets of qubits. Higher is better. Bars below 100% indicate lower success rate for Trios. The geometric mean increase in success rate is 23%. The x labels indicate the qubit distance for a range of bars.	220

5.38	Simulated upper-bounds on the program execution success probability on various hardware (using 20x lower idle and gate errors than Johannesburg). Neighboring pairs of bars compare the baseline with Trios compiled for Johannesburg. Higher is better when comparing pairs of bars with the same color. The geometric mean success rates over the benchmarks that use Toffoli gate for each device type respectively are 2.2%→9.8%, 3.2%→12%, 0.19%→6.0%, 7.3%→17%. The rightmost three benchmarks contain zero Toffoli gates so have no change vs. the baseline.	220
5.39	A comparison between the baseline and Trios for various hardware. Above 0% indicates benefit. All two-qubit gates (for communication and computation) are counted. The geometric mean reductions in gate counts are 37%, 36%, 48%, and 26% respectively. The rightmost three benchmarks contain zero Toffoli gates so have no change vs. the baseline.	221
5.40	Normalized Figure 5.38 to show our consistent increase in program success with Trios. Above 10 ⁰ indicates benefit. Some improvement factors are huge due to near-zero baseline success rates. The geometric mean increases in success rate are 4.4x, 3.7x, 31x, and 2.3x respectively. The rightmost three benchmarks contain zero Toffoli gates so have no change vs. the baseline.	222
5.41	Factor of improvement in success rate in Trios over baseline for scaling gate error rates. The dotted line indicates current error rates on IBM Johannesburg and the dashed line (20x improvement) indicates values of the near future used in simulation. In our approximation of success rate factors of improvement in gate error rates lead to an exponential fall off in success ratios, as expected. In the very near term, we expect Trios to drastically improve the execution of quantum programs.	223

LIST OF TABLES

2.1	Asymptotic comparison of N -controlled gate decompositions. The total gate count for all circuits scales linearly (except for Barenco [15], which scales quadratically). Our construction uses qutrits to achieve logarithmic depth without ancilla. We benchmark our circuit construction against Gidney [70], which is the asymptotically best ancilla-free qubit circuit.	14
2.2	Truth table for taking two input carry statuses $C_0 = (c_{0,0}c_{0,1})$ and $C_1 = (c_{1,0}c_{1,1})$, encoded as in the left table of Figure 2.8 and outputting the resulting combined carry status $C'_1 = (c'_{1,0}c'_{1,1})$ while dirtying the bits of C_0 based on the desired behavior of the right table of Figure 2.8. For example, consider the third to last row of the table. The bitstring is (1101) indicating the first carry status is generate (g) and the second carry status is propagate (p). The means we want to output generate $g = (11)$ on the last two bits but we don't care what happens to the first two bits, as we've done here, leaving them in ternary states, specifically $ 2\rangle 2\rangle$	20
2.3	Carry status encoding scheme for the $+K$ adder circuit.	23
2.4	Noise models simulated for superconducting devices.	34
2.5	Noise models simulated for trapped ion devices.	35
2.6	Truth table for 2-3-1 Compression	39
2.7	Truth table for 2-4-1 Compression	40
4.1	Starting point coherence times and constant gate times for the hardware models.	116
4.2	Transmon, depth-10 cavity, and total qubit costs of each T-state generation protocol for $d = 5$	120
5.1	Benchmarks and some of their properties.	139
5.2	Error model details for current systems [92, 141].	139

5.3	Summary of the improvements on 50 qubit benchmarks for MEQC-10-5-2D over 2D-5-10. In all cases, we see strict improvement.	149
5.4	A subset of our benchmarks. Clean multi-control has a maximum size of 87. With more than 87 data qubits and fewer than 13 clean ancilla, the depth of the multi-control decomposition is too large to run on these cluster-based machines with predicted error rates.	170
5.5	Comparing Static-OEE against FGP-rOEE over all benchmarked instances. We obtain improvement across the board with the worst case still reducing non-local communication by 22.6%.	177
5.6	Estimated execution time of the clean multi-control benchmark with 76 data qubits and 24 ancilla. Two-qubit gates take 300ns [92] and the multiplier indicates how many times longer non-local communication operations take. .	177
5.7	List of compiler configurations used in our study. The IBM Qiskit 0.5.7 compiler is used as a the baseline. The use of calibration data is marked by a ★.	185
5.8	Characteristics of benchmark programs.	196
5.9	Details about benchmarks for Trios for reference, both NISQ programs and other quantum subroutines	218

ACKNOWLEDGMENTS

First and foremost I want to thank my advisor Fred Chong for all of his guidance through my PhD. I'd also like my committee Hank Hoffmann, Ken Brown, and Ali Javadi-Abhari all of whom have been valuable collaborators and mentors. I'd like to also thank all of my collaborators with special thanks to several of my frequent collaborators - Prakash Murali, Pranav Gokhale, Gokul Ravi, Kaitlin Smith, Sophia Lin, Andrew Litteken, Alex Hoover, and most of all Casey Duckering. I am grateful for each of my collaborators beyond those here, without which none of this work would have ever been completed.

I'd also like to thank everyone else along the way that's supported me - academically, emotionally, or otherwise. This includes all of the people I've met in EPIQC and the University of Chicago and more, too many to list here. Finally I'd like to thank my family without which I'd never have gotten anywhere close to this point.

ABSTRACT

Despite its relative infancy, there are a number of emerging quantum technologies for quantum computation, and it is unclear which will be the clear winner. Evaluation of these technologies at the architectural level, far beyond the small-scale prototypes of 1 to 2 qubits, is critical to producing viable systems capable of executing both near and long-term applications effectively. In order to fairly evaluate new hardware platforms, it is vital to develop technology-specific optimizations and compilation frameworks to push hardware closer to its fundamental limits rather than being hindered by ineffective software. Ultimately, our goal is develop an evaluation framework to help decipher whether or not proposed technologies are able to scale efficiently into the future. Central questions in platform evaluation can be roughly categorized into three themes: 1. Does this technology support fast and economical program execution. 2. Can this technology produce consistent and high-quality program outputs? 3. Does this technology fit the requirements of practical applications now and/or in the future? This work aims to provide the first steps in the development of this framework.

In this thesis, I explore the design, optimization, and evaluation of a variety of competing quantum technologies which are each vital when deciphering good candidates for scalable quantum computation, even in its very early stages. This manifests as several case studies serving as examples of a much larger, fundamental architectural questions. First, we consider what are the right abstractions for a quantum computing system by exploring the use of multivalued quantum logic to better make use of hardware capabilities. Second, we consider architectural trade-off spaces by considering neutral atom hardware. Third, we consider application-guided architectural design, evaluating how technology specific architectures can be designed to best fit target applications, like error correction codes. We examine the use of localized memory to enable virtualization of error corrected logical qubits. Finally, we explore a variety of technology specific (and agnostic) compilation optimizations to push proposed architectures and hardwares to their limits to best evaluate their ability to scale

beyond prototypes and support large scale applications.

CHAPTER 1

INTRODUCTION

Despite recent booms in quantum information processing research, the field of practical quantum computing is relatively new. As an emerging paradigm, it may seem odd to consider new physical technologies, but the field is already at the horizon of some potentially game-changing capabilities. Although current machines have shown impressive success with devices based upon trapped ions and superconducting transmons, it is unclear what the eventual winning technologies will be and it is imperative to consider how to design and build large-scale architectures based subsets of these physical technologies.

Quantum hardware is still in its relative infancy, boasting tens of qubits as opposed to the thousands to millions needed to execute important algorithms for unordered search and factoring [78, 168]. Most available systems have struggled to scale beyond their prototypes while simultaneously suppressing gate errors and increasing qubit coherence times. This limits the types of programs which can execute effectively, let alone perform error correction. It is unclear whether systems composed of superconducting qubits or trapped ions, the major industry players will take the lead.

Device success is predicated on the increase in the number of qubits, reduction of gate errors below error correction thresholds, and increases in qubit coherence times, or lifetimes. In the near term, this translates into larger (more qubits), deeper programs (more operations per qubit) with improved output distributions (higher likelihood of obtaining the correct answers). In the long term, this translates into qubits which are protected from noise inherent in operating quantum systems which cannot be perfectly isolated from the environment while still having control in the form of encoded logical qubits.

In recent years, there have been a number of improvements throughout the compilation pipeline both close to the hardware and high level circuit optimization. These optimizations aim to reduce gate counts, circuit depth, and communication costs as proxies for increasing

the size and quality of programs executable on currently available hardware. These are valuable optimizations and are often well correlated with improved program success rate or reducing physical requirements as we graduate into error corrected regime.

An alternative approach is to evaluate the viability and trade-offs of new quantum technology and associated architectures. This approach is tightly linked to optimizations along the entire hardware-software stack, where it is much more advantageous to develop technology-specific toolchains. Despite tremendous efforts at both the hardware and software level to minimize the effects of noise, it is unclear if any of the available hardware will be able to scale as needed and no clear winner on underlying hardware has emerged. Even further, it is unknown whether this hardware is best suited to execute the desired programs and while it is often the case that we adapt compilation to the hardware, i.e. transforming applications into the right shape for execution, an alternative approach is explore how to design new architectures which are better suited for the applications we want to run.

Evaluating new hardware technology at the architectural level is decidedly different, though intrinsically coupled to, the development of quantum hardware. Device physicists' goal is often to demonstrate the *existence* of high quality qubits and operations in prototypes, while the architect's goal is to evaluate the systems-level ramifications of this technology, exploring the inherent trade-off spaces to determine viability in the near and long term. Perhaps most critically, this architectural design exploration leads to important insights about what is most important for hardware developers to optimize. For example if some limitations can be effectively mitigated via software, hardware developers can focus on other more fundamental issues. This process of co-design, by evaluating new technology early and often, is central to accelerating scalability.

In this dissertation, we explore several key themes in the design of technology-specific architectural frameworks. While we highlight these at a high level to support a much larger design principle, each chapter that follows aims to be self-contained.

First, we consider how to determine the right abstractions for quantum computation, specifically one of the most fundamental considerations: the computing radix. Most quantum systems, like classical systems, are expressed using a two-level binary abstraction using qubits. However, many available hardware platforms composed of trapped ions or superconducting circuits are not intrinsically binary, instead offering access to a large, sometimes infinite, spectrum of possible logical states. Efficient use of these states could prove extremely useful in expediting quantum computation. For many quantum circuits, we can reduce the total execution time of the program by using additional space in the form of additional qubits or devices. When programs are time-limited it is critical to decompose circuits to minimize depth. This requires that we maintain a delicate balance as using additional space limits the total size of programs which can be executed if a large portion of resources must be reserved. We advocate in particular for the use of *intermediate qudits* (begin and end as qubits) which can be used to obtain the same asymptotic depth as the best known decompositions but can be achieved without the need for any extra space.

The proposed techniques free up more hardware for computation, rather than dedicating device space as ancilla, limiting the maximum size of computation which can be performed. Current hardware is often calibrated for use only with qubits but has higher-level states available. While classically multivalued and mixed-radix computing is niche, it is important to evaluate the architectural ramifications of these strategies for quantum computation. By temporarily accessing already available higher states, these works can accelerate common circuit components and reduce space overhead, extending the frontier of what can be computed.

Second, we consider technology-specific architectural tradeoffs by considering one promising platform candidate: neutral atoms. Current hardware implementations face unique and fundamental scalability challenges. While these devices have been useful as proof of concept demonstrations of small-scale, near-term algorithms, it is unclear whether any of them in present form will be able to execute the large-scale computation needed for quantum speedup.

Evaluating the viability of new hardware is essential. Architectural studies which fully explore their unique trade-off spaces are key for finding the best way to accelerate beyond prototypes. One such alternative to superconducting qubits or trapped ions is neutral atoms.

There are distinct advantages, for example long distance interactions and native complex instructions, provided by neutral atoms by developing a dedicated compilation pipeline for executing quantum programs on the target hardware. Under similar gate error rates, neutral atom architectures will be capable of executing larger programs sooner than other competing technologies. Unfortunately, neutral atom systems suffer a potentially crippling drawback—atoms can be lost both during and between program execution which usually requires the entire array to be reloaded and the output to be discarded. Fortunately, software solutions can effectively mitigate this increased run time overhead. By developing several compiler solutions which are effective at mitigating this loss we are able to demonstrate viability for scalable quantum computation sooner. This work highlights a key design principle: exploring the use of new hardware serves a critical role in the development of the platform itself. By solving fundamental problems at the systems level with software, hardware developers can focus on solving and optimizing other problems. Co-design of quantum systems is key to accelerate the advancement of quantum computing technology.

Third we consider architectural design driven by specific applications. In the long term, error correction is one of the most important applications underlying hardware must support and in the past it has been common to develop error correction codes with already available hardware in mind, for example surface codes and color codes which rely on only nearest-neighbor connectivity between devices to implement. While it has been effective, it may not be ideal. The alternative is to consider designing architectures using new hardware components to better fit codes, or generally applications. As an example, we consider using new quantum memory technology to better match surface codes.

In general, current quantum architectures do not tend to make a distinction between

memory and processing of quantum information. These architectures are viable, however, as more and more qubits are needed the scalability challenges become apparent. To scale to the millions of qubits needed for error correction, a memory-based architecture can be used to decouple qubit-count from transmon count. We explore a proof-of-concept demonstration of its viability by virtualizing surface code tiles in a 2.5D memory-based architecture. Despite the surface code being designed with currently available 2D architectures in mind, this application is better matched with this 2.5D architecture which allows qubits to be virtualized and stored in local memory. This design reduces physical qubit requirements, enabling fast logical CNOTs, and expediting the creation of special resource states. This highlights a central theme for current quantum architecture design—we must explore the uses of new hardware technology, here the use of resonant cavities to store the information of many qubits, at the systems level.

Fourth we consider the broad subject of quantum compilation pipelines, a critical component in the evaluation of any emerging quantum technology. Without architecture-specific compilation frameworks, it is very difficult to effectively compare two technologies, and superficial modifications to preexisting frameworks, while usable, are inefficient. To best understand the limits of new architectures, we must integrate key features and constraints into the mapping, routing, and scheduling stages of our pipelines.

We discuss several compiler optimizations which outperform generic methods and span several different architectural models. For example, for cluster-based architectures, we can use graph partitioning techniques to inform a timed-sliced routing compiler. Further, we can use effective lookahead heuristics to minimize unnecessary qubit displacement which lead to high latency cross-cluster interactions. These lookahead heuristics have proved essential for other compiler designs for both cavity-based superconducting devices as well as neutral atom devices, ensuring low communication overheads. For each of these architectures, we must adapt compiler design to account for device-specific constraints.

The central message is clear: generic and standard compilation flow can often be ineffective, introducing unnecessary communication overhead thereby reducing output quality. This leads to many device-agnostic optimizations. For example, most available devices cannot execute complex instructions like the Toffoli gate and instead decompose these gates early in the compilation pipeline. Routing and scheduling of these gates tends to be more difficult, requiring higher communication costs. However, by modifying this sequence, specifically by decomposing complex instructions based on target hardware topology and only after routing the interacting qubits together, this communication cost can be cut dramatically.

CHAPTER 2

DETERMINING THE RIGHT ABSTRACTIONS: MULTIVALUED QUANTUM LOGIC

¹ Given the severe constraints on quantum resources, it is critical to fully optimize the compilation of a quantum algorithm in order to have successful computation. Prior architectural research has explored techniques such as mapping, scheduling, and parallelism [55, 99, 79] to extend the amount of useful computation possible. In this chapter, we consider another technique: quantum trits (qutrits) and generally the use of *qudits* for any number of logical levels d .

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

Prior work on qutrits (or more generally, d -level *qudits*) identified only constant factor gains from extending beyond qubits. In general, this prior work [154] has emphasized the information compression advantages of qutrits. For example, N qubits can be expressed as $\frac{N}{\log_2(3)}$ qutrits, which leads to $\log_2(3) \approx 1.6$ -constant factor improvements in run times.

Our approach utilizes qutrits in a novel fashion, *intermediate qutrits*, essentially using the third state as temporary storage, but at the cost of higher per-operation error rates. Under this treatment, the run time (i.e. circuit depth or critical path) is *asymptotically* faster, and

1. JMB's contributions in this chapter include the majority of the circuit designs (with equal contribution from CD for many and significant help from PG and CD on others) the introduction of logical temporary ternary, testing and verification.

the reliability of computations is also improved. Moreover, our approach only applies qutrit operations in an intermediary stage: the input and output are still qubits, which is important for initialization and measurement on real devices [158, 157] and means it can easily be substituted for a binary-only version without needing to modify the remainder of the circuit.

The net result of this section is to extend the frontier of what quantum computers can compute. In particular, the frontier is defined by the zone in which every machine qubit is a data qubit, for example a 100-qubit algorithm running on a 100-qubit machine. In this frontier zone, we do not have room for non-data workspace qubits known as ancilla. The lack of ancilla in the frontier zone is a costly constraint that generally leads to inefficient circuits. For this reason, typical circuits instead operate below the frontier zone, with many machine qubits used as ancilla. We demonstrate that ancilla can be substituted with qutrits, enabling us to operate efficiently within the ancilla-free frontier zone.

We can further improve on this result in a more general way by instead having ancilla, specifically clean ancilla, be *generated* local during the decomposition of an algorithm into a quantum circuit. That is, we propose a new circuit which performs qubit-qutrit compression storing the information of many qubits as a small number of qutrits at the cost of some gate overhead. These compression circuits produce clean ancilla in the $|0\rangle$ state. The stored data can be retrieved later when needed since all quantum operations are reversible. Essentially, when certain groups of qubits will be unused for a long period of time, we can repurpose them by compressing them and using the produced ancilla. This “compression” is a rearrangement of the stored binary values into higher states. This allows us to store more information into the same number of physical quantum devices and free up qubits for computation.

In this section we present an application of this technique to give logarithmic depth decompositions of quantum arithmetic circuits - a carry lookahead adder and by extension addition by a constant a direct improvement to the more “clever” version we present at first. We present two compression circuits for qubit-qutrit and qubit-ququart compression and

evaluate advantages of various compression schemes in real applications.

We highlight the primary contributions of this section:

1. A Generalized Toffoli circuit construction based on qutrits that leads to asymptotically faster circuits ($633N \rightarrow 38 \log_2 N$) than equivalent qubit-only constructions. We also reduce total gate counts from $397N$ to $6N$.
2. Larger arithmetic circuit constructions with intermediate qutrits such as the Incrementer, $A + B$ and $+K$ adders which obtain $O(\log^2 n)$, $O(\log^3 n)$, and $O(\log^3 n)$ depth, respectively.
3. Simulation results using the simulator developed in [74], under realistic noise models, which demonstrate our circuit construction outperforms equivalent qubit circuits in terms of error. For completeness, we also benchmark our circuit against a qubit-only construction augmented by an ancilla and find our construction is still more reliable.
4. The introduction of a novel use of intermediate *qudits* based on the compression of information on many quantum devices into a small number but with more logical states to directly enable allocation of devices as ancilla which generalizes the use of intermediate qudits for any circuit.

2.1 Relevant Background

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

Quantum states can be acted on by quantum gates which (a) preserve valid probability distributions that sum to 1 and (b) guarantee reversibility. For example, the X gate transforms

a state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ to $X|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$. The X gate is also an example of a classical reversible operation, equivalent to the NOT operation. In quantum computation, we have a single irreversible operation called measurement that transforms a quantum state into one of the two basis states with a given probability based on α and β .

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

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

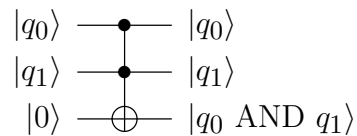


Figure 2.1: Reversible AND circuit using a single ancilla bit. The inputs are on the left, and time flows rightward to the outputs. This AND gate is implemented using a Toffoli (CCNOT) gate with inputs q_0 , q_1 and a single ancilla initialized to 0. At the end of the circuit, q_0 and q_1 are preserved,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

Physical systems in classical hardware are typically binary. However, in common quantum hardware, such as in superconducting and trapped ion computers, there is an infinite spectrum of discrete energy levels. The qubit abstraction is an artificial approximation achieved by

suppressing all but the lowest two energy levels. Instead, the hardware may be configured to manipulate the lowest three energy levels by operating on qutrits. In general, such a computer could be configured to operate on any number of d levels, except as d increases the number of opportunities for error, termed error channels, increases. Here, we focus on $d = 3$ with which we achieve the desired improvements to the Generalized Toffoli gate.

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

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

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

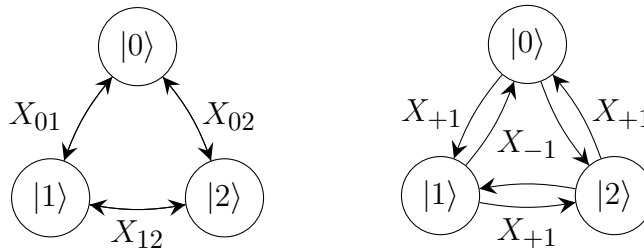


Figure 2.2: The five nontrivial permutations on the basis elements for a qutrit. (Left) Each operation here switches two basis elements while leaving the third unchanged. These operations are self-inverses. (Right) These two operations permute the three basis elements by performing a $+1 \pmod 3$ and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

Other, non-classical, operations may be performed on a single qutrit. For example, the Hadamard gate [147] can be extended to work on qutrits in a similar fashion as the X gate was extended. In fact, all single qubit gates, like rotations, may be extended to operate on qutrits. In order to distinguish qubit and qutrit gates, all qutrit gates will appear with an appropriate subscript.

Just as single qubit gates have qutrit analogs, the same holds for two qutrit gates. For example, consider the CNOT operation, where an X gate is performed conditioned on the control being in the $|1\rangle$ state. For qutrits, any of the X gates presented above may be performed, conditioned on the control being in any of the three possible basis states. Just as qubit gates are extended to take multiple controls, qutrit gates are extended similarly. The set of single qutrit gates, augmented by any entangling two-qutrit gate, is sufficient for universality in ternary quantum computation [35].

One question concerning the feasibility of using higher states beyond the standard two is whether these gates can be implemented and perform the desired manipulations. Qutrit gates have been successfully implemented [53, 140, 111] indicating it is possible to consider higher level systems apart from qubit only systems.

In order to evaluate a decomposition of a quantum circuit, we consider quantum circuit costs. The space cost of a circuit, i.e. the number of qubits (or qutrits), is referred to as circuit *width*. Requiring ancilla increases the circuit width and therefore the space cost of a circuit. The time cost for a circuit is the *depth* of a circuit. The depth is given as the length of the critical path (in terms of gates) from input to output.

2.2 Prior Work in Multivalued Quantum Logic

2.2.1 Qudits

Qutrits, and more generally qudits, have been studied in past work both experimentally and theoretically. Experimentally, d as large as 10 has been achieved (both one- and two-qudit operations) [112], and $d = 3$ qutrits are commonly used internally in many quantum systems [37, 65].

However, in past work, qudits have conferred only an information compression advantage. For example, N qubits can be compressed to $\frac{N}{\log_2(d)}$ qudits, giving only a constant-factor advantage [154] at the cost of greater errors from operating qudits instead of qubits. Under the assumption of linear cost scaling with respect to d , it has been demonstrated that $d = 3$ is optimal [77, 108].

The information compression advantage of qudits has been applied specifically to Grover’s search algorithm [63, 121, 183, 96] and to Shor’s factoring algorithm [26]. Ultimately, the tradeoff between information compression and higher per-qudit errors has not been favorable in past work. As such, the past research towards building practical quantum computers has focused on qubits.

This chapter introduces qutrit-based circuits which are *asymptotically* better than equivalent qubit-only circuits. Unlike prior work, we demonstrate a compelling advantage in both runtime and reliability for smaller constructions, thus justifying the use of qutrits.

2.2.2 Generalized Toffoli Gate

We first focus on the Generalized Toffoli gate, which simply adds more control qubits to the Toffoli circuit in Figure 2.1. The Generalized Toffoli gate is an important primitive used across a wide range of quantum algorithms, and it has been the focus of extensive past optimization work. Table 2.1 compares past circuit constructions for the Generalized Toffoli

	This Work	Gidney [70]	He [84]	Barenco [15]	Wang [183]	Lanyon [115] Ralph [156]
Depth	$\log N$	N	$\log N$	N^2	N	N
Ancilla	0	0	N	0	0	0
Control Type	Qutrits	Qubits	Qubits	Qubits	Qutrits	Qubits
Target Type	Qubit	Qubit	Qubit	Qubit	Qubit	$d = N$ -level qudit
Constants	Small	Large	Small	Small	Small	Small

Table 2.1: Asymptotic comparison of N -controlled gate decompositions. The total gate count for all circuits scales linearly (except for Barenco [15], which scales quadratically). Our construction uses qutrits to achieve logarithmic depth without ancilla. We benchmark our circuit construction against Gidney [70], which is the asymptotically best ancilla-free qubit circuit.

gate to our construction.

Among prior work, the Gidney [70], He [84], and Barenco [15] designs are all qubit-only. The three circuits have varying tradeoffs. While Gidney and Barenco operate at the ancilla-free frontier, they have large circuit depths: linear with a large constant for Gidney and quadratic for Barenco. The Gidney design also requires rotation gates for very small angles, which poses an experimental challenge. While the He circuit achieves logarithmic depth, it requires one ancilla per data qubit, effectively halving the effective potential of any given quantum hardware. Nonetheless, in practice, most circuit implementations use these linear-ancilla constructions due to their small depths and gate counts.

As in our approach, circuit constructions from Lanyon [115], Ralph [156], and Wang [183] have attempted to improve the ancilla-free Generalized Toffoli gate by using qudits. Both the Lanyon [115] and Ralph [156] constructions, which have been demonstrated experimentally, achieve linear circuit depths by operating the target as a $d = N$ -level qudit. Wang [183] also achieves a linear circuit depth but by operating each control as a qutrit.

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

In our simulations, we benchmark our circuit construction against the Gidney construction [70] because it is the asymptotically best qubit circuit in the ancilla-free frontier zone. We label these two benchmarks as QUTRIT and QUBIT. The QUBIT circuit handles the lack of ancilla by using *dirty* ancilla, which unlike *clean* (initialized to $|0\rangle$) ancilla, can have an unknown initial state. Dirty ancilla can therefore be bootstrapped internally from a quantum circuit. However, this technique requires a large number of Toffoli gates which makes the decomposition particularly expensive in gate count.

Augmenting the base Gidney construction with a single ancilla² does reduce the constants for the decomposition significantly, although the asymptotic depth and gate counts are maintained. For completeness, we also benchmark our circuit against this augmented construction, QUBIT+ANCILLA. However, the augmented circuit does not operate at the ancilla-free frontier, and it conflicts with parallelism.

2.3 Circuit Constructions

In order for quantum circuits to be executable on hardware, they are typically decomposed into single- and two- qudit gates. Performing efficient low depth and low gate count decompositions is important in both the NISQ regime and beyond.

2.3.1 Key Intuition

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

2. This ancilla can also be dirty.

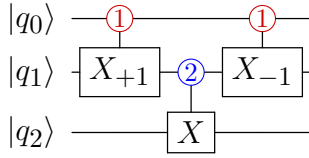


Figure 2.3: A Toffoli decomposition via qutrits. Each input and output is a qubit. The red controls activate on $|1\rangle$ and the blue controls activate on $|2\rangle$. The first gate temporarily elevates q_1 to $|2\rangle$ if both q_0 and q_1 were $|1\rangle$. We then perform the X operation only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

In Figure 2.3, a Toffoli decomposition using qutrits is given. A similar construction for the Toffoli gate is known from past work [115, 156]. The goal is to perform an X operation on the last (target) input qubit q_2 if and only if the two control qubits, q_0 and q_1 , are both $|1\rangle$. First a $|1\rangle$ -controlled X_{+1} is performed on q_0 and q_1 . This elevates q_1 to $|2\rangle$ iff q_0 and q_1 were both $|1\rangle$. Then a $|2\rangle$ -controlled X gate is applied to q_2 . Therefore, X is performed only when both q_0 and q_1 were $|1\rangle$, as desired. The controls are restored to their original states by a $|1\rangle$ -controlled X_{-1} gate, which undoes the effect of the first gate. The key intuition in this decomposition is that the qutrit $|2\rangle$ state can be used instead of ancilla to store temporary information.

2.3.2 Generalized Toffoli Gate

We now present our circuit decomposition for the Generalized Toffoli gate in Figure 2.4. The decomposition is expressed in terms of three-qutrit gates (two controls, one target) instead of single- and two- qutrit gates, because the circuit can be understood purely classically at this granularity. In actual implementation and in our simulation, we used a decomposition [53] that requires 6 two-qutrit and 7 single-qutrit physically implementable quantum gates.

Our circuit decomposition is most intuitively understood by treating the left half of the circuit as a tree. We specifically trace out the construction found in Figure 2.4 here, but is easily generalized. The desired property is that the root of the tree, q_7 , is $|2\rangle$ if and only if each of the 15 controls was originally in the $|1\rangle$ state. To verify this property, we observe the

root q_7 can only become $|2\rangle$ iff q_7 was originally $|1\rangle$ and q_3 and q_{11} were both previously $|2\rangle$. At the next level of the tree, we see q_3 could have only been $|2\rangle$ if q_3 was originally $|1\rangle$ and both q_1 and q_5 were previously $|2\rangle$, and similarly for the other triplets. At the bottom level of the tree, the triplets are controlled on the $|1\rangle$ state, which are only activated when the even-index controls are all $|1\rangle$. Thus, if any of the controls were not $|1\rangle$, the $|2\rangle$ states would fail to propagate to the root of the tree. The right half of the circuit performs *uncomputation* to restore the controls to their original state.

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

Our circuit decomposition still works in a straightforward fashion when the control type of the top qubit, q_0 , activates on $|2\rangle$ or $|0\rangle$ instead of activating on $|1\rangle$. These two constructions are necessary for the Incrementer circuit. We verified our circuits, both formally and via simulation, the code for these constructions is found with our simulation framework [6].

2.3.3 Larger Arithmetic Circuits: The Incrementer

The Incrementer circuit performs the $+1 \pmod{2^N}$ operation to a register of N qubits. While logarithmic circuit depth can be achieved with linear ancilla qubits [57], the best ancilla-free incrementers require either linear depth with large linearity constants [71] or quadratic depth [15]. Using our Generalized Toffoli gate with alternate control activations as a subcircuit, the incrementer circuit is reduced to $O(\log^2 N)$ depth with no ancilla, a significant improvement over past work.

The core operational principle of an incrementer is to flip bits starting with the LSB, stopping after the first 0 is encountered. For example, in the bitstring 11001111, the increment starts at the right and propagates through four 1's before being terminated by the first 0

encountered. As a result, the right five bits are flipped. The key insight in our Incrementer circuit decomposition is that a Generalized Toffoli can be used to propagate a carry bit to the middle of a bitstring by checking if all of the right-half bitstrings were 1. This procedure is then repeated recursively. Figure 2.5 shows an example of our incrementer circuit construction for an $N = 8$ width register. We have verified the general construction, both by formal proof and by explicit circuit simulation for larger N . The critical path of this circuit is the chain of $\log N$ multiply-controlled gates (of width $\frac{N}{2}, \frac{N}{4}, \frac{N}{8}, \dots$) which act on $|a_0\rangle$. Since our multiply-controlled gate decomposition has log-depth, we arrive at a total circuit depth circuit scaling of $\log^2 N$.

2.3.4 Larger Arithmetic Circuits: The $A + B$ Adder

In the $A+B$ adder, we take as input two n -qubit registers A and B and output on the B bits the sum $A + B$ while leaving the A bits unchanged. In all constructions presented here, the least significant bit is found at the top (a_0, b_0) while the most significant bit is at the bottom (a_n, b_n) . One of the most commonly used constructions is the Cuccaro adder [46]. A simple way of improving this construction with intermediate qutrits is to simply replace all of the Toffoli gates used with our more efficient Toffoli decomposition found in Figure 2.3. We demonstrate this straightforward substitution in Figure 2.6. While this does improve the constants, we do not obtain any asymptotic advantage. This suggests an important fact: simple replacement of Toffoli decompositions in other quantum circuits is not a sufficient method for obtaining asymptotic improvements. Instead, we need to be more clever about how we use intermediate qutrits. For simplicity, we will ignore any carry-in or carry-out, though these could be added in a straightforward way.

We will now demonstrate how to obtain sublinear depth with *no additional ancilla* using intermediate qutrits. First, we observe the high level decomposition of Figure 2.7 in which we see that adding $A + B$ can be done by first computing if the first half of input bits

generate a carry and if so, adding 1 to the second half of the sum register. We can then recursively compute $A + B$ on the lower and higher order bits independently. We use the incrementer construction of Figure 2.5, modified only by controlling the initial X_{+1} and final X_{02} gates of the decomposition on the carry status output of the Carry circuit. In order for this construction to have sublinear depth, we need only show the first part, the carry-controlled incrementer, can be done in sublinear depth. This problem reduces further by noting we already have shown an $O(\log^2 n)$ depth incrementer implying we need to show a sublinear carry operation.

Determining Carry Status In-Place in Sublinear Depth

First we introduce an encoding scheme for the different carry statuses, in the traditional language of *generate* (g), *propagate* (p), and *kill* (k). For one bit of an adder, $a_i + b_i$, a carry is generated if and only if $a_i = b_i = 1$ and a carry from a carry-in would be propagated to a carry-out if and only if $a_i \oplus b_i = 1$. Finally, any carry through these bits would be killed if and only if $a_i = b_i = 0$. Therefore, we use the encoding scheme of the left table of Figure 2.8. Next, we need to understand how these different carry statuses combine. For example, if one set of bits generates and the next set propagates, then the net result of the combined group is to *generate*; we summarize these combinations in the right table of Figure 2.8.

With these two pieces of information, we introduce our gadget which will allow us to determine the carry status of a set of input bits in logarithmic depth via the general decomposition of Figure 2.10. This gadget takes four bits as input, where the first pair represents the encoded carry status of the first set and the second pair is the encoded carry status of the second set. The output is the carry status of the combined groups on the second pair. In order to do this we must *dirty* the first pair by having them occupy qutrit states. This gadget guarantees the second pair will be *qubits* meaning this gadget can be used recursively. Therefore, at the end of Figure 2.10, the output on the last two inputs will

$c_{0,0}$	$c_{0,1}$	$c_{1,0}$	$c_{1,1}$	$c'_{0,0}$	$c'_{0,1}$	$c'_{1,0}$	$c'_{1,1}$
0	0	0	0	(0	0)	0	0
0	0	0	1	(2	0)	0	0
0	0	1	0	(0	1)	0	0
0	0	1	1	(0	0)	1	1
0	1	0	0	(0	2)	0	0
0	1	0	1	(1	1)	0	1
0	1	1	0	(0	2)	1	0
0	1	1	1	(0	1)	1	1
1	0	0	0	(1	0)	0	0
1	0	0	1	(2	0)	0	1
1	0	1	0	(1	1)	1	0
1	0	1	1	(1	0)	1	1
1	1	0	0	(2	1)	0	1
1	1	0	1	(2	2)	1	1
1	1	1	0	(1	1)	1	1
1	1	1	1	(1	2)	1	1

Table 2.2: Truth table for taking two input carry statuses $C_0 = (c_{0,0}c_{0,1})$ and $C_1 = (c_{1,0}c_{1,1})$, encoded as in the left table of Figure 2.8 and outputting the resulting combined carry status $C'_1 = (c'_{1,0}c'_{1,1})$ while dirtying the bits of C_0 based on the desired behavior of the right table of Figure 2.8. For example, consider the third to last row of the table. The bitstring is (1101) indicating the first carry status is generate (g) and the second carry status is propagate (p). The means we want to output generate $g = (11)$ on the last two bits but we don't care what happens to the first two bits, as we've done here, leaving them in ternary states, specifically $|2\rangle|2\rangle$.

be (11) if and only if a carry is generated. This carry circuit is logarithmic in depth because at each layer we halve the number of input bits. The realization of this gadget is found in Figure 2.9, and its truth table is found in Table 2.2.

Why is this decomposition $O(\log^3 n)$ depth? We've shown an incrementer can be performed in $O(\log^2 n)$ depth. The depth of the carry circuit is $O(\log n)$ since every CCS circuit is constant depth and the number of inputs is halved at each layer. We use the final carry status output to control an incrementer which is done in $O(\log^2 n)$ depth. We then uncompute the carry to prepare for the recursive step, in which we compute $A + B$ on the two halves of the inputs. Since this recursive step works on inputs half the size of the original input, this requires $O(\log n)$ steps. This results in a total depth of $O(\log^3 n)$. What this means is, with more clever use of intermediate qutrits, beyond simple replacement, we can produce asymptotically better decompositions.

2.3.5 Larger Arithmetic Circuits: Constant, +K, Addition

We will now present how to perform constant addition, that is given an input register A of n qubits we want to add a known constant K to A . We will give two constructions, the first uses a similar cascading technique as the Cuccaro adder to produce a linear depth +K adder and the second will use a technique similar to the new A+B adder to produce a sublinear depth +K adder.

Linear +K Addition

For each of the constructions we give, we want to take advantage of the fact that we know every bit of the constant being added, K and use this to inform the circuits we build. Specifically, the circuit's exact decomposition is of course a function of K itself.

For this construction, we assume $K = (k_0 k_1 \dots k_{n-1})$, with k_0 the least significant bit. Also, we assume $k_0 = 1$. If $k_0 = 0$, we find the first i for which $k_i = 1$ and then add

$K' = (k_i k_{i+1} \dots k_{n-1})$ to $A' = (a_i a_{i+1} \dots a_{n-1})$. There are several components we use given in Figure 2.11 which we choose based on the particular values of the bit string K . Used together, we can quite simply compute $A + K$ for some constant K in linear circuit depth (as well as linear gate count) using intermediate qutrits and no additional ancilla, as found in Figure 2.12.

Can we do better with this technique similar to the $A + B$ adder? Using similar techniques to that construction, as well as a different trick, we obtain a sublinear depth $A + K$ adder with intermediate qutrits.

$O(\log^3 n)$ Depth +K Addition

Our goal of this is to demonstrate sublinearity which we do at the cost of some rather large constants. Some methods for the reduction of these constants are discussed at the end of this section. This construction is slightly more complicated than that of the $A + B$ addition mostly due to fewer available qubits. Specifically, in the $A + B$ adder we could use the qubits of A to store some carry information, giving us roughly twice as many available qubits for the same sized registers. Here, we cannot simply employ the techniques of before.

First, we assume as before that $K = (k_0 k_1 \dots k_{n-1})$ with $k_0 = 1$ and k_0 the least significant bit. In the entire discussion following, the left most bit is the least significant bit. We will make a further assumption for simplicity of future discussion, that is $|K| = 4\ell$ for some $\ell \in \mathbb{Z}^+$. This assumption is not necessary in general. Then, consider groups of bits of K , $B_m = (k_{4m} k_{4m+1} k_{4m+2} k_{4m+3})$ for $m \in [0, \ell - 1]$. For this discussion we say $(B_i)(B_j)$ is the concatenation of the binary number B_i to the front of B_j . In order to add the constant $(0\dots 0)(B_m)(0\dots 0)$ we can add $(0\dots 0)(1000)(0\dots 0)$ exactly B_m -many times, that is

$$A + (0\dots 0)(B_m)(0\dots 0) = A + \sum_{i=1}^{(B_m)} (0\dots 0)(1000)(0\dots)$$

A	B	Carry Status
0	0	k
0	1	k
1	0	p
1	1	g

Table 2.3: Carry status encoding scheme for the $+K$ adder circuit.

Then, taking $M = \max_{j \in [0, \ell-1]} (B_j)$ and since $(B_0)(B_1)\dots(B_{\ell-1}) = K$, we have

$$A + (B_0)(B_1)\dots(B_{n/4}) = A + \sum_{i=0}^M (\beta_{0,i}000)(\beta_{1,i}000)\dots(\beta_{\ell-1,i}000)$$

where $\beta_{j,i} = 1$ if $i \leq (B_j)$ and 0 otherwise. Intuitively, we can understand this expression as saying we can add K by adding a sequence of numbers which add up to K . This sum has at most 15 terms in it since each B_j is exactly 4 bits, that is $M \leq 15$. Taking $K_i = (\beta_{0,i}000)(\beta_{1,i}000)\dots(\beta_{\ell-1,i}000)$ for each $i \in [0, M]$ we have the circuit structure of Figure 2.13.

We now show how to construct $+K_i$ which can be done fairly similar to the construction for $A + B$. A high level scheme for the decomposition of these blocks is given in Figure 2.14 and is very similar to that of Figure 2.7. In this case too we need only show a sublinear depth Carry circuit. Note as before, in this circuit, the Carry outputs a single two-bit encoded carry status and we modify the incrementer circuit by controlling the first and last operation on the status encoding generate.

For this decomposition, we establish a new carry status encoding scheme, given in Table 2.3. This new scheme is necessary. Consider the case of $\beta_{j,i} = 0$, all but 1 of the inputs results in the carry status k , i.e. 15 of the 16 inputs need to output kill. With a simple information theoretic argument we note the original encoding is insufficient. In that encoding, with only a single output encoding kill, (00), at most 9 inputs could result in this output (the other two input bits are used as qutrits for 3^2 states). Therefore, no circuit could exist

to output kill on 15 inputs. If we instead have two outputs encoding kill, here (00) and (01), then we can have $2 * 3^2 = 18$ outputs encode kill. This new encoding is sufficient for both $\beta_{j,i}$ being 0 or 1. We will *combine* carry statuses with the same strategy as before.

In each of the subcircuits presented, we note there is little intuition which can be provided for the specific construction and each was constructed by defining a truth table with a desired set of properties, usually with a certain subset of inputs remaining as qubits while other inputs becoming dirtied. The circuits are then constructed via experimentation, though techniques adapted from reversible logic synthesis could be used to produce similar cascades of gates. These subcircuits all share one important property: they are all constant depth.

We now define two new gadgets which will prepare the groups $(\beta_{j,i}000)$, one for when $\beta_{j,i} = 0$ and one for when $\beta_{j,i} = 1$. In each of these gadgets, we input the group of 4 bits $(a_{4j}a_{4j+1}a_{4j+2}a_{4j+3})$ and output on the last two bits the carry status while leaving the first two bits possibly in qutrit states. These two gadgets are found in Figures 2.15, 2.16. Finally, we need a new combine carry status (CCS_{+K}) gadget because we have a new carry status encoding; this gadget can be found in Figure 2.17. Putting this all together we obtain the Carry decomposition in Figure 2.18. This carry decomposition is done in logarithmic depth. Every group of 4 inputs is prepared with the proper PC subcircuit in parallel in constant depth. Using a cascade of constant depth CCS_{+K} circuits, each with four inputs, we halve the number of carry status inputs at each layer requiring $O(\log n)$ layers to obtain the final carry status.

Each of the $+K_i$ subcircuits has $O(\log^3 n)$ depth. In each, we compute the carry in logarithmic depth and use the result of the carry to control an incrementer in $O(\log^2 n)$ depth. We uncompute the carry to prepare for the recursive step. Since the number of inputs is halved in each step of the recursion, we need $O(\log n)$ recursions. Note the base case for this recursion is to perform an incrementer on the j -th set of bits if $\beta_{j,i} = 1$ and do nothing otherwise. This results in the desired $O(\log^3 n)$ depth. Since there are a constant

(15) number of $+K_i$ blocks added, this implies the total decomposition is $O(\log^3 n)$ depth. This is all done with 0 additional ancilla with all inputs and outputs guaranteed to be qubits. We note there may be more optimized subcircuits which may further reduce our constants.

This demonstrates by using intermediate qutrits we can obtain asymptotically improved circuits beyond just the Toffoli decomposition. In these larger arithmetic circuits, however, the constants will be fairly large due to the large number of two-control gates we use in each of our gadgets. Furthermore, in the $+K$ circuit, we bound the number of K_i blocks by 15. How can we improve on these constants to increase practicality? The number of multiply controlled gates used in the PC, CCS, and other constant depth components could be reduced. This could be done for example by using optimization methods found in reversible logic synthesis. In the same vein, one other optimization could be to increase the number of bits we use in our components from 4 to some higher constant. We chose 4 because it is the smallest size, given our encoding schemes, for which the desired circuits exist. It is possible to use larger components, but this may come at the cost of using more multiply-controlled gates which have extremely poor decompositions.

For the $+K$ circuit design, we note the use of 15 blocks is an intuitive design to bound the number of $+K_i$ blocks. However, there are ways this number can be reduced. One such way to reduce this bound to just 4 blocks is to consider again dividing K into groups $K = B_0B_1\dots B_{\ell-1}$ each containing 4 bits $(b_{i,0}b_{i,1}b_{i,2}b_{i,3})$. Note $K = (b_{0,0}000)\dots(b_{\ell-1,0}000) + (0)(b_{0,1}000)\dots(b_{\ell-1,1}00) + \dots + (000)(b_{0,3}000)\dots(b_{\ell-1,3})$. That is, we can simply shift the starting point. As a simple example, consider adding $4_{10} = (0010)_2$, again the left most digit is the least significant bit, to some number $A = (a_0a_1a_2a_3)$. This can be done by applying a $+1$ circuit to the last two bits of A , i.e. $A + (0010) = (a_0a_1)[(a_2a_3) + (10)]$. So for the above sum, we take the block K_i to be the i -th term, for a total of 4 blocks. Notice this works because the final group of each block never participates in the Carry subcircuit; we never require this group to have 4 bits. We execute the $+K_i$ circuits in the same way as before,

except not including the i least significant bits. By breaking up K in this way, we can reduce the total number of blocks we need to execute by a constant factor.

In this section we've demonstrated the general power of intermediate qutrits, specifically by reducing the depth of many circuits such as the Generalized Toffoli and arithmetic circuits asymptotically without requiring any additional ancilla which is particularly advantageous on devices with limited numbers of qubits and extending what is possible at the ancilla-free frontier.

2.4 Application to Algorithms

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

2.4.1 *Artificial Quantum Neuron*

The artificial quantum neuron [175] is a promising target application for our circuit construction, because the algorithm's circuit implementation is dominated by large Generalized Toffoli gates. The algorithm may exhibit an exponential advantage over classical perceptron encoding and it has already been executed on current quantum hardware. Moreover, the threshold behavior of perceptrons has inherent noise resilience, which makes the artificial quantum neuron particularly promising as a near-term application on noisy systems. The current implementation of the neuron on IBM quantum computers relies on ancilla qubits [174] which constrains the circuit width to $N = 4$ data qubits. Our circuit construction offers a path to larger circuit sizes without waiting for larger hardware.

2.4.2 Grover's Algorithm

Grover's Algorithm for search over M unordered items requires just $O(\sqrt{M})$ oracle queries. However, each oracle query is followed by a post-processing step which requires a multiply-controlled gate with $N = \lceil \log_2 M \rceil$ controls [147]. The explicit circuit diagram is shown in Figure 2.19.

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

2.4.3 Arithmetic Circuits and Shor's Algorithm

The Incrementer circuit is a key subcircuit in many other arithmetic circuits such as constant addition, modular multiplication, and modular exponentiation, as seen before. Further, the modular exponentiation circuit is the bottleneck in the runtime for executing Shor's algorithm for factorization [71, 81]. While a shallower Incrementer circuit alone is not sufficient to reduce the asymptotic cost of modular exponentiation (and therefore Shor's algorithm), our other circuit constructions, specifically the +K adder, will reduce the depth without costing any additional ancilla, that is space.

2.4.4 Error Correction and Fault Tolerance

The Generalized Toffoli gate has applications to circuits for both error correction [42] and fault tolerance [50]. We foresee two paths of applying these circuits. First, our circuit construction can be used to construct error-resilient *logical qubits* more efficiently. This is critical for quantum algorithms like Grover's and Shor's which are expected to require such logical qubits. In the nearer-term, NISQ algorithms are likely to make use of limited error correction. For instance, recent results have demonstrated that error correcting a single qubit at a time for the Variational Quantum Eigensolver algorithm can significantly reduce total

error [152]. Thus, our circuit construction is also relevant for NISQ-era error correction.

2.5 Simulator for Verification of Constructions

To simulate our circuit constructions, we developed a qudit simulation library, built on Google’s Cirq Python library [2]. Cirq is a qubit-based quantum circuit library and includes a number of useful abstractions for quantum states, gates, circuits, and scheduling.

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

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

The software from this section is fully open source [6].

2.5.1 Noise Simulation

Figure 2.20 depicts a schematic view of our noise simulation procedure which accounts for both gate errors and idle errors, described below. To determine when to apply each gate and idle error, we use Cirq’s scheduler which schedules each gate as early as possible, creating a sequence of **Moment**’s of simultaneous gates. During each **Moment**, our noise simulator applies a gate error to every qudit acted on. Finally, the simulator applies an idle error to every

qudit. This noise simulation methodology is consistent with previous simulation techniques which have accounted for either gate errors [129] or idle errors [107].

Gate errors arise from the imperfect application of quantum gates. Two-qudit gates are noisier than single-qudit gates [5], so we apply different noise channels for the two. Our specific gate error probabilities are given in Section 2.6.

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

1. Idle errors depend on duration, which in turn depend on the schedule of simultaneous gates (**Moments**). In particular, two-qudit gates take longer to apply than single-qudit gates. Thus, if a **Moment** contains a two-qudit gate, the idling errors must be scaled appropriately. Our specific scaling factors are given in Section 2.6.
2. For the generic model of gate errors, the error channel is applied with probability independent of the quantum state. This is not true for idle errors such as T_1 amplitude damping, which only applies when the qudit is in an excited state. This is treated in the simulator by computing idle error probabilities during each **Moment**, for each qutrit.

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

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

during intermediate computation. Therefore, the initialization and readout errors for our circuits are identical to those for conventional qubit circuits.

```

 $|\Psi\rangle \leftarrow$  random initial state vector
 $|\Psi\rangle_{\text{ideal}} =$  circuit applied to  $|\Psi\rangle$  without noise
foreach Moment do
    foreach Gate  $\in$  Moment do
         $|\psi\rangle \leftarrow$  Gate applied to  $|\psi\rangle$ 
        GateError  $\leftarrow$  DrawRand(GateError Prob.)
         $|\psi\rangle \leftarrow$  GateError applied to  $|\psi\rangle$ 
    end
    foreach Qutrit do
        if Moment has 2-qudit gate then
            IdleErrors  $\leftarrow$  long-duration idle errors
        else
            IdleErrors  $\leftarrow$  short-duration idle errors
        end
        Prob.  $\leftarrow$  [ $\|M|\Psi\rangle\|^2$  for  $M \in$  IdleErrors]
        IdleError  $\leftarrow$  DrawRand(Prob.)
         $|\psi\rangle \leftarrow$  IdleError applied to  $|\psi\rangle$ 
        Renormalize( $|\psi\rangle$ )
    end
end
return  $\langle \Psi_{\text{ideal}} | \Psi \rangle^2$ , fidelity between ideal & actual output;
Algorithm 1: Pseudocode for each simulation trial, given a particular circuit and noise model.

```

We also do not consider crosstalk errors, which occur when gates are executed in parallel. The effect of crosstalk is very device-dependent and difficult to generalize. Moreover, crosstalk can be mitigated by breaking each **Moment** into a small number of sub-moments and then scheduling two-qutrit operations to reduce crosstalk, as demonstrated in prior work [181, 28].

2.5.2 Simulator Efficiency

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

(`complex128` in NumPy), a state vector occupies 77 megabytes.

A naive circuit simulation implementation would treat every quantum gate or `Moment` as a $d^N \times d^N$ matrix. For 14 qutrits, a single such matrix would occupy 366 terabytes—out of range of simulability. While the exponential nature of simulating our circuits is unavoidable, we mitigate the cost by using a variety of techniques which rely only on state vectors, rather than full square matrices. For example, we maintain Cirq’s approach of applying gates by Einstein Summation [22], which obviates computation of the $d^N \times d^N$ matrix corresponding to every gate or `Moment`.

Our noise simulator only relies on state vectors, by adopting the quantum trajectory methodology [33, 165], which is also used by the Rigetti PyQuil noise simulator [170]. At a high level, the effect of noise channels like gate and idle errors is to turn a coherent quantum state into an incoherent mix of classical probability-weighted quantum states (for example, $|0\rangle$ and $|1\rangle$ with 50% probability each). The most complete description of such an incoherent quantum state is called the density matrix and has dimension $d^N \times d^N$. The quantum trajectory methodology is a stochastic approach—instead of maintaining a density matrix, only a single state is propagated and the error term is drawn randomly at each timestep. Over repeated trials, the quantum trajectory methodology converges to the same results as from full density matrix simulation [170]. Our simulator employs this technique—each simulation in Algorithm 1 constitutes a single quantum trajectory trial. At every step, a specific `GateError` or `IdleError` term is picked, based on a weighted random draw.

Finally, our random state vector generation function was also implemented in $O(d^N)$ space and time. This is an improvement over other open source libraries [102, 103], which perform random state vector generation by generating full $d^N \times d^N$ unitary matrices from a Haar-random distribution and then truncating to a single column. Our simulator directly computes the first column and circumvents the full matrix computation.

With optimizations, our simulator is able to simulate circuits up to 14 qutrits in width.

This is in the range as other state-of-the-art noisy quantum circuit simulations [38] (since 14 qutrits \approx 22 qubits). While each simulation trial took several minutes (depending on the particular circuit and noise model), we were able to run trials in parallel over multiple processes and multiple machines, as described in Section 5.3.5.

2.6 Simulation and Error Models

All simulation results are obtained using the noise simulator of [74]. Here we provide only details about the particular error models for easier reference.

We chose noise models that represent realistic near-term machines. We first present a generic, parametrized noise model that is roughly applicable to all quantum systems. We then present specific parameters, under the generic noise model, that apply to near-term superconducting quantum computers. Finally, we present a specific noise model for $^{171}\text{Yb}^+$ trapped ions.

2.6.1 Generic Noise Model

Gate Errors

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

Idle Errors

We focus on the amplitude damping idle error channel, also referred to as T_1 relaxation. This noise channel irreversibly takes qudits to lower states. For qubits, the only amplitude damping channel is from $|1\rangle$ to $|0\rangle$, and we denote this damping probability as λ_1 . For qutrits, we also model damping from $|2\rangle$, which occurs with probability λ_2 .

2.6.2 Superconducting QC

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

As a starting point for representative near-term noise models, we first consider parameters for *current* superconducting quantum computers. For IBM’s public cloud-accessible superconducting quantum computers, we have $3p_1 \approx 10^{-3}$ and $15p_2 \approx 10^{-2}$. The duration of single- and two- qubit gates is $\Delta t \approx 100ns$ and $\Delta t \approx 300ns$ respectively, and the IBM devices have $T_1 \approx 100\mu s$ [5, 122].

However, simulation for these current parameters indicates that an error is almost certain to occur during execution of a modest size 14-input Generalized Toffoli circuit, our primary simulation benchmark. This motivates us to instead consider noise models for better devices that are a few years away. In particular, we adopt a baseline superconducting noise model, SC, corresponding to a superconducting device which has 10x lower gate errors and 10x longer T_1 duration than the current IBM hardware. We note that this range parameters have already been achieved experimentally in superconducting devices for gate errors [16, 17] and for T_1 duration [159, 61] independently. We also note that faster gates (shorter Δt) are yet another path towards greater noise resilience. To constrain the number of benchmarks, we do not consider noise models with faster gates, though in practice, such improvements could

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

Table 2.4: Noise models simulated for superconducting devices.

also make quantum computers more noise-resilient.

We also consider three additional near-term device noise models, which are indexed to the SC noise model and further improve gate errors, T_1 , or both by a 10x factor. The specific parameters are given in Table 2.4. Our 10x improvements projections are realistic extrapolations of improvements to hardware. In particular, Schoelkopf’s Law—the quantum analogue of Moore’s Law—has observed that T_1 durations have increased 10x every 3 years for the past 20 years [73]. Hence, 100x longer T_1 is a reasonable projection for devices that are ~ 6 years away.

2.6.3 Trapped Ion $^{171}\text{Yb}^+$ QC

We also simulated noise models from trapped ion quantum computing devices. Trapped ion devices are well matched to our qutrit-based circuit constructions because:

- The devices feature all-to-all connectivity [31]
- many ions that are ideal candidates for QC devices are naturally multi-level systems

We focus on the $^{171}\text{Yb}^+$ ion, which has been experimentally demonstrated as both a qubit and qutrit [158, 157]. Trapped ions are often favored in QC schemes due to their long T_1 times. One of the main advantages of using a trapped ion is the ability to take advantage of magnetically insensitive states known as “clock states.” By defining the computational subspace on these clock states, idle errors caused from fluctuations in the magnetic field are minimized. However, compared to superconducting devices, gates are much slower. Thus gate

Noise Model	p_1	p_2
TLQUBIT	6.4×10^{-4}	1.3×10^{-4}
TLBARE_QUTRIT	2.2×10^{-4}	4.3×10^{-4}
TLDRESSED_QUTRIT	1.5×10^{-4}	3.1×10^{-4}

Table 2.5: Noise models simulated for trapped ion devices.

noise is the main source of error for ion trap devices. We modelled a fundamental source of these errors: the spontaneous scattering of photons originating from the lasers used to drive the gates. The duration of single- and two- qubit gates used in this calculation was $\Delta t \approx 1 \mu\text{s}$ and $\Delta t \approx 200 \mu\text{s}$ respectively [32]. The single- and two- qudit gate error probabilities are given in Table 2.5.

2.7 Simulation Results

In order to further evaluate use of intermediate qutrits, we examined in more detail the Generalized Toffoli circuit construction and used the noise simulator of [74]. Figure 2.21 plots the exact circuit depths for all three benchmarked generalized Toffoli circuits. The qubit-based circuit constructions from past work are linear in depth and have a high linearity constant. Augmenting with a single borrowed ancilla reduces the circuit depth by a factor of 8. However, both circuit constructions are surpassed significantly by our qutrit construction, which scales logarithmically in N and has a relatively small leading coefficient.

Figure 2.22 plots the total number of two-qudit gates for all three circuit constructions. As noted before, our circuit construction is not asymptotically better in total gate count—all three plots have linear scaling. However, as emphasized by the logarithmic vertical axis, the linearity constant for our qutrit circuit is 70x smaller than for the equivalent ancilla-free qubit circuit and 8x smaller than for the borrowed-ancilla qubit circuit.

Our simulations under realistic noise models were run in parallel on over 100 n1-standard-4 Google Cloud instances. These simulations represent over 20,000 CPU hours, which was

sufficient to estimate mean fidelity to an error of $2\sigma < 0.1\%$ for each circuit-noise model pair.

The full results of our circuit simulations are shown in Figure 5.7. All simulations are for the 14-input (13 controls, 1 target) Generalized Toffoli gate. We simulated each of the three circuit benchmarks against each of our noise models (when applicable), yielding the 16 bars in the figure.

2.8 Qubit-Qudit Compression

2.8.1 Motivation

As we have seen, many quantum algorithms make use of ancilla, additional free bits used to store temporary information during computation which are typically returned to their original state after use. Ancilla have a variety of use cases such as to reduce the total execution time. In some cases, they can provide asymptotic improvements to the depth of circuit decompositions as with the generalized Toffoli before. This highlights an important space-time tradeoff in quantum programs - we spend extra space in the form of ancilla in order to reduce the depth of an input circuit.

Real quantum machines will have a limited number of qubits so it is important that we make the most of them to enable computation of larger, more useful problems sooner. In the previous examples we saw higher dimensional qudits could be used as a replacement for ancilla in certain circuit components to great effect. Unfortunately, by accessing these states, the computation is subject to a greater variety of errors, in fact the number of error types scale quadratically in the computing radix. However, if qudit states are used properly, the amount gained outweighs this cost. In this section we will make more explicit use of qudit states temporarily during computation while maintaining binary inputs and outputs of a circuit.

In this section we propose ancilla, specifically clean ancilla, be *generated* local during the

decomposition of an algorithm into a quantum circuit. That is, we propose a new circuit which performs qubit-qudit compression storing the information of many qubits as a small number of qudits at the cost of some gate overhead. These compression circuits produce clean ancilla in the $|0\rangle$ state. The stored data can be retrieved later when needed since all quantum operations are reversible. Essentially, when certain groups of qubits will be unused for a long period of time, we can repurpose them by compressing them and using the produced ancilla. This “compression” is a rearrangement of the stored binary values into higher states. This allows us to store more information into the same number of physical quantum devices and free up qubits for computation.

In this work we present an application of this technique to give logarithmic depth decompositions of quantum arithmetic circuits - a carry lookahead adder and by extension addition by a constant. We present two compression circuits for qubit-qutrit and qubit-ququart compression and evaluate advantages of various compression schemes. Then we present our decomposition of the zero-ancilla, in-place $A + B$ adder which takes as input two registers A and B of qubits and possibly carryin and carryout; any fresh $|0\rangle$ states used are generated locally. We then evaluate the costs of this decomposition. Finally, we discuss various extensions to our arithmetic decomposition in Sections 2.8.3 and 2.8.3.

2.8.2 Compression Schemes

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

What does this gain for us? It is known that by simply fully encoding a computation into a higher radix we obtain a constant space and time advantage over binary-only circuits.

However, recently it was shown that use of these higher states can act as temporarily storage, similar to the use of an ancilla, and can convey an asymptotic reduction in circuit depth [74]. This circuit construction, as well as other work, suggests we can obtain better circuits while using fewer qubits by accessing higher states temporarily.

We take this a step further and *generate* ancilla temporarily out of input qubits in order to take advantage of previously known efficient binary circuit decompositions like that of [58]. Using this method, we can reduce the number of external ancilla needed from $O(n)$ to 0 while keeping the same asymptotic circuit depth. To do this, we allow subsets of qubits to temporarily store higher values, becoming qudits, to store the information of many qubits within a few qudits. As a concrete example, consider three qubits. There are $2^3 = 8$ total basis states while for two qudits there are $3^2 = 9$ basis states. Therefore all the information of 3 qubits can be stored in two qudits and the third qubit can be left in a chosen state, $|0\rangle$, a clean ancilla. We refer to this process as *compression*, that is storing the information of many *qubits* in a smaller number of *qudits*.

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

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

A	B	C	A'	B'	C'
0	0	0	0	0	0
0	0	1	2	2	0
0	1	0	0	1	0
0	1	1	0	2	0
1	0	0	1	0	0
1	0	1	2	1	0
1	1	0	1	1	0
1	1	1	1	2	0

Table 2.6: Truth table for 2-3-1 Compression

2-3-1 Compression

In 2-3-1 compression we take as input three qubits and output 2 qutrits and a single ancilla, a qubit guaranteed to be in the $|0\rangle$ state. First, consider the truth table of Table 2.6. We note the partial function represented by this truth table is invertible, implying there exists a reversible circuit that realizes it. The third output, C' , is guaranteed to be in the $|0\rangle$ state, an ancilla. By storing qubit information used infrequently we can generate an extremely useful ancilla to be used elsewhere in the circuit.

Because we ensure all inputs are binary, we do not need to consider the inputs with value 2 to the ternary circuit. An example circuit realizing this truth table is given in Figure 2.24. When a compression circuit of this type is applied, we need to keep track of which pair of qutrits encodes the three qubits, in order. When the compressed data is needed, we can decompress by applying the inverse of this function. The inverse circuit is simply the gates in reverse order with $+1$ replaced with -1 . Notably, this inversion requires an ancilla as input. To retrieve the information, the inverse should be applied taking in any free ancilla and then the stored bits can be computed on as normal.

This circuit, while accomplishing what is desired, can be rather inefficient. For example, in architectures with limited connectivity this circuit requires some number of expensive communication operations since every input qubit must be adjacent at some point. Furthermore,

this circuit requires the use of a two-controlled qutrit gate which is typically decomposed into a sequence of 6 two-qutrit gates and 10 single-qutrit gates [53]. In total this compression requires 22 gates, 12 two-qutrit and 10 single-qutrit gates.

2-4-1 Compression

While 2-3-1 compression required a fairly substantial number of gates, the 2-4-1 compression circuit can convert qubit inputs into ancilla more simply and with few gates. This does not come for free. In quantum computing, we subject our computation to a greater probability of error by using higher radix gates and by persisting for longer durations in higher energy states. In Table 2.7, we show that two qubits can be compressed into a single ququart and one ancilla. 2-4-1 compression is simpler than 2-3-1 compression because 2^2 states fit evenly in a single ququart with 4 states. In Figure 2.25, we show a compression circuit using only 3 two-ququart gates in total, a substantial reduction over the 2-3-1 counterpart. In the next section, we show how compression and decompression can be used to design efficient circuits requiring no ancilla.

A	B	A'	B'
0	0	0	0
0	1	2	0
1	0	1	0
1	1	3	0

Table 2.7: Truth table for 2-4-1 Compression

2.8.3 Decompositions with Compression

We now present our $A + B$ adder. This circuit takes as input two equal-sized registers of qubits, A and B , and optionally carry-in or carry-out bits. This decomposition uses no ancilla and instead generates ancilla locally when needed by sub-components. In prior work,

to achieve a logarithmic depth decomposition, $O(n)$ many ancilla were required where n is the size of the input register. We will demonstrate how this efficient decomposition can be used along with our new compression technique to obtain an $O(\log n)$ depth decomposition of the same adder in-place without the extra use of ancilla.

We first briefly review the work of [58] which gives a qubit-only in-place adder with ancilla which we will refer to as $(A + B)_2$. We give the decomposition for registers of size 4 in Figure 2.26. One of the key contributions of this prior work is to demonstrate how, in logarithmic depth, the carry bits could be computed and used (and subsequently uncomputed to restore input ancilla back to the $|0\rangle$ state). This decomposition requires $2m - w(m) - \lfloor \log m \rfloor$ ancilla, where $w(m)$ is the number of 1's appearing in the binary expansion of the number of inputs, m . We will use this number later to determine how many ancilla to generate via compression. This same prior work demonstrates several variants of this circuit. We require those with either a carry-in bit, a carry-out bit, or both.

We will now present our decomposition shown in Figure 2.27. Let $A = (a_1 a_2 \dots a_n)$ and $B = (b_1 b_2 \dots b_n)$ be the input registers with a_1, b_1 the least significant bits of each register. We divide these registers into c blocks R_1, \dots, R_c each of size $2n/c$. We assume for clarity that n is a multiple of c but our constructions will work for any n , with one additional block containing the remaining $2(n \bmod c)$ qubits. Take $R_i = (a_{(i-1)(c/n)+1} b_{(i-1)(c/n)+1} \dots a_{i(c/n)} b_{i(c/n)})$ then notice for $i > 1$ we can perform an addition circuit $(A + B)_2$ with carry-in and carry-out on block R_i in $O(\log(n/c)) = O(\log n)$ depth by generating the proper number of ancilla out of the other input qubits, specifically $2(n/c) - w(n/c) - \lfloor \log n/c \rfloor$ ancilla. We will assume a worst case scenario of $2n/c$ ancilla to simplify the analysis. Suppose we are performing $(A + B)_2$ on block R_i while every other block is unused. We can perform compression on the currently unused qubits in all other blocks $\{R_j | j \neq i\}$ to obtain generated ancilla which can then be used by the current adder subcircuit.

Recall 2-3-1 compression takes 3 qubits and outputs a single ancilla. Let a 2-3-1 *Compress*

circuit be a circuit which takes any number of qubits m as input and applies 2-3-1 compression to triplets resulting in $\lfloor m/3 \rfloor$ ancilla. Then applying 2-3-1 compression to all qubits in $\{R_j | j \neq i\}$ we obtain $\lfloor (c-1)2n/3c \rfloor$ ancilla. We now have constraints on what the constant c should be for our decomposition to be feasible. That is we must have $\lfloor (c-1)2n/3c \rfloor \geq 2n/c$. Because we must store intermediate carry values between each $(A+B)_2$, we will actually require an additional $c-1$ ancilla, giving us $\lfloor (c-1)2n/3c \rfloor \geq 2n/c + c - 1$. By solving the inequality, this implies our construction is feasible for $c = 5$ and $n \geq 30$. An alternative adder that is ancilla-free but does not scale well asymptotically, like an $O(n)$ -depth adder [46], may be used where our construction is infeasible on small problem sizes with $n < 30$.

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

We have now computed the sum $A+B$ and now must cleanup the intermediate carry bits. This can be done by working in reverse to uncompute each carry-out without undoing the addition. One intuitive way would be to simply apply the inverse of the $(A+B)_2$ circuit we applied to block R_{c-1} which will uncompute the addition and $c_{out,c-1}$ and then re-apply it

without carry-out. Now the ancilla storing $c_{out,c-1}$ is restored to $|0\rangle$. We repeat this process on each of the blocks in reverse order. Finally, after $c_{out,1}$ has been uncomputed and the ancilla restored to $|0\rangle$, we uncompress all of the qubits. The resulting output will be the sum S in register B with register A left unchanged from the input.

Uncomputing the intermediate carry-out bits can be improved dramatically by noticing that by applying the inverse of $(A + B)_2$ with carry-in and carry-out and the subsequently applying $(A + B)_2$ with only carry-in is unnecessary. Instead we can uncompute the carry-out by only applying the inverse of the second half of $(A + B)_2$ with carry-out and then executing the second half of $(A + B)_2$ with a few extra gates in Figure 2.26d to cancel the carry-out.

Earlier, we show our decomposition only works when $c = 5$ using 2-3-1 compression. However, due to page size constraints, we do not show some of the repeated blocks in Figure 2.27. The block of gates surrounded by a dashed line is simply repeated in a block diagonal pattern indicated by the ellipsis. If we instead used 2-4-1 compression, the factor of 3 in the earlier inequality would be replaced with 2 making $c = 4$ feasible with a constraint of $n \geq 12$.

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

Carry-in and Carry-out

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

+K Adder

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

The design of our $+K$ adder will use as subcircuits the $(+K)_2$ circuit derived from $(A + B)_2$ from [58] and described earlier. The design of $(+K)_2$ is the same as $(A + B)_2$ except the qubits of register A are removed and all CNOT gates with a control on a_i are removed and only replaced with X gates if $k_i = 1$. Similarly, the Toffoli gates (controlled-controlled-not gates) are removed and replaced with CNOT gates in the same way. Depending on the value of K , some of the ancilla may also be removed but in the worst case, $(+K)_2$ may still require $2n/c - w(n/c) - \lfloor \log n/c \rfloor - 1$ ancilla for input size n/c which we upper bound by $2n/c$. The circuit still has $O(\log n)$ depth.

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

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

2.9 Discussion

We will first present a more detailed analysis of our Generalized Toffoli decomposition and then give a high level discussion of the use of *intermediate qutrits*, for example using the novel compression schemes, in general.

Figure 5.7 demonstrates that our QUTRIT construction (orange bars) significantly outperforms the ancilla-free QUBIT benchmark (blue bars) in fidelity (success probability) by more than 10,000x. or the SC, SC+T1, and SC+GATES noise models, our qutrit constructions achieve between 57-83% mean fidelity, whereas the ancilla-free qubit constructions all have almost 0% fidelity. Only the lowest-error model, SC+T1+GATES achieves modest fidelity of 26% for the QUBIT circuit, but in this regime, the qutrit circuit is close to 100% fidelity.

The trapped ion noise models achieve similar results—the DRESSED_QUTRIT and the BARE_QUTRIT achieve approximately 95% fidelity via the QUTRIT circuit, whereas the TL-QUBIT noise model has only 45% fidelity. Between the dressed and bare qutrits, the dressed qutrit exhibits higher fidelity than the bare qutrit, as expected. Moreover, as discussed in the appendix of [74], the dressed qutrit is resilient to leakage errors, so the simulation results should be viewed as a lower bound on its advantage over the qubit and bare qutrit. Based on these results, trapped ion qutrits are a particularly strong match to our qutrit circuits. In addition to attaining the highest fidelities, trapped ions generally have all-to-all connectivity [31] within each ion chain, which is critical as our circuit construction requires operations between distant qutrits.

The superconducting noise models also achieve good fidelities. They exhibit a particularly large advantage over ancilla-free qubit constructions because idle errors are significant for superconducting systems, and our qutrit construction significantly reduces idling (circuit depth). However, most superconducting quantum systems only feature nearest-neighbor or short-range connectivity. Accounting for data movement on a nearest-neighbor-connectivity 2D architecture would expand the qutrit circuit depth from $\log N$ to \sqrt{N} (since the distance

between any two qutrits would scale as \sqrt{N}). However, recent work has experimentally demonstrated fully-connected superconducting quantum systems via random access memory [142]. Such systems would also be well matched to our circuit construction.

For completeness, Figure 5.7 also shows fidelities for the QUBIT+ANCILLA circuit benchmark, which augments the ancilla-free QUBIT circuit with a single dirty ancilla. Since QUBIT+ANCILLA has linearity constants $\sim 10x$ better than the ancilla-free qubit circuit, it exhibits significantly better fidelities. While our QUTRIT circuit still outperforms the QUBIT+ANCILLA circuit, we expect a crossing point where augmenting a qubit-only Generalized Toffoli with enough ancilla would eventually outperform QUTRIT. However, we emphasize that the gap between an ancilla-free and constant-ancilla construction for the Generalized Toffoli is actually a fundamental rather than an incremental gap, because:

- Constant-ancilla constructions prevent circuit parallelization. For example, consider the parallel execution of N/k disjoint Generalized Toffoli gates, each of width k for some constant k . An ancilla-free Generalized Toffoli would pose no issues, but an ancilla-augmented Generalized Toffoli would require $\Theta(N/k)$ ancilla. Thus, constant-ancilla constructions can impose a choice between serializing to linear depth or regressing to linear ancilla count. The Incrementer circuit in Figure 2.5 is a concrete example of this scenario—any multiply-controlled gate decomposition requiring a single clean ancilla or more than 1 dirty ancilla would contradict the parallelism and reduce runtime.
- Even if we only consider serial circuits, given the exponential advantage of certain quantum algorithms, there is a significant practical difference between operating at the ancilla-free frontier and operating just a few data qubits below the frontier.

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

circuits. Similarly, for the larger arithmetic circuits, while constants may be prohibitive in the near term, we have shown an improved asymptotic behavior with our technique.

Our circuit construction and simulation results point towards promising directions of future work that we highlight below:

- Relatedly, we see value in a logic synthesis tool that injects qutrit optimizations into qubit circuits, automated in ways inspired by classical reversible logical synthesis tools [172, 130]; we've shown a small example of this in our implementation of the Cuccaro adder with our improved Toffoli used instead of the less efficient qubit only one, however, we also noted simple replacement was not sufficient to take full advantage of intermediate qutrits.
- While $d = 3$ qutrits were sufficient to achieve the desired asymptotic speedups for our circuits of interest, there may be other circuits that are optimized by qudit information carriers for larger d . In particular, we note that increasing d and thereby increasing information compression may be advantageous for hardware with limited connectivity.

Independent of these future directions, the results presented in this chapter are applicable to quantum computing in the near term as is the case of the Toffoli and the generalized version, on machines that are expected within the next five years, as well as showing how in the longer term our technique can be used in larger circuits. The net result of this work is to extend the frontier of what is computable by quantum hardware, and hence to accelerate the timeline for practical quantum computing, rather than waiting for better hardware. Emphatically, our results are driven by the use of qutrits, or qudits in general, for *asymptotically* faster ancilla-free circuits. Moreover, we also improve linearity constants by two orders of magnitudes in the case of the Generalized Toffoli and arithmetic circuits. Our results justify the use of qudits as a path towards scaling quantum computers.

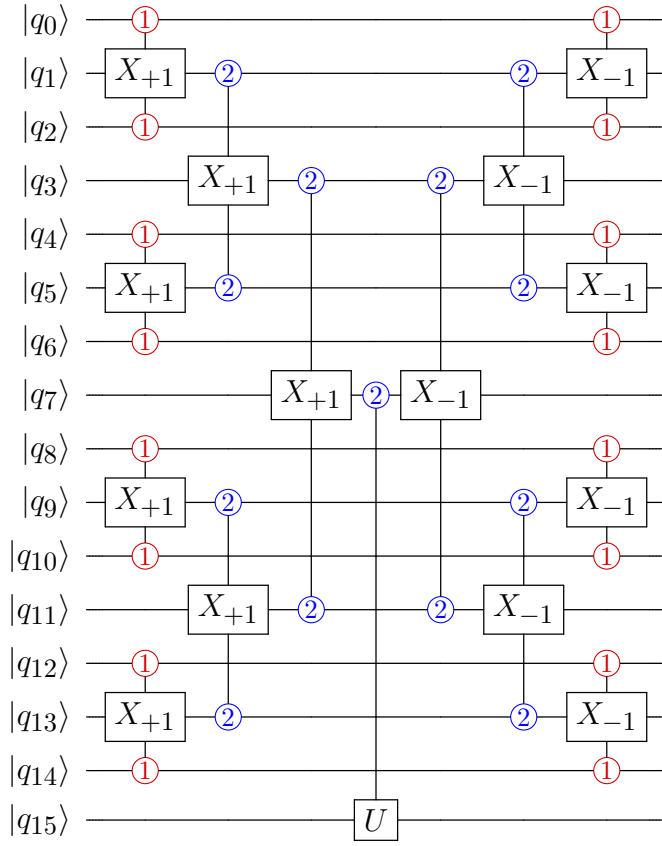


Figure 2.4: Our circuit decomposition for the Generalized Toffoli gate is shown for 15 controls and 1 target. The inputs and outputs are both qubits, but we allow occupation of the $|2\rangle$ qutrit state in between. The circuit has a tree structure and maintains the property that the root of each subtree can only be elevated to $|2\rangle$ if all of its control leaves were $|1\rangle$. Thus, the U gate is only executed if all controls are $|1\rangle$. The right half of the circuit performs uncomputation to restore the controls to their original state. This construction applies more generally to any multiply-controlled U gate. Note that the three-qutrit gates are decomposed into 6 two-qutrit and 7 single-qutrit gates in our actual simulation, as based on the decomposition in [53].

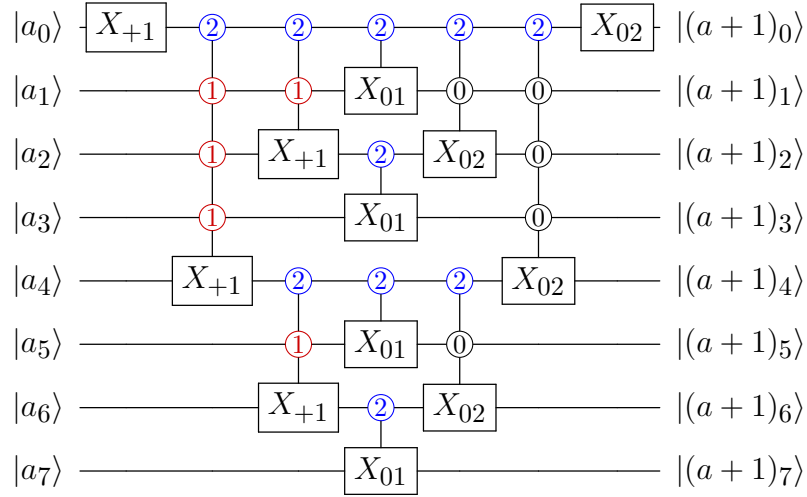


Figure 2.5: Our circuit decomposition for the Incrementer. At each step in the design, multiply-controlled gates using the decomposition in Figure 2.4 are used to efficiently propagate carries over half of the subcircuit. The $|2\rangle$ control checks for carry generation and the chain of $|1\rangle$ controls check for carry propagation. The circuit depth is $\log^2 N$, which is only possible because of our log depth multiply-controlled gate primitive.

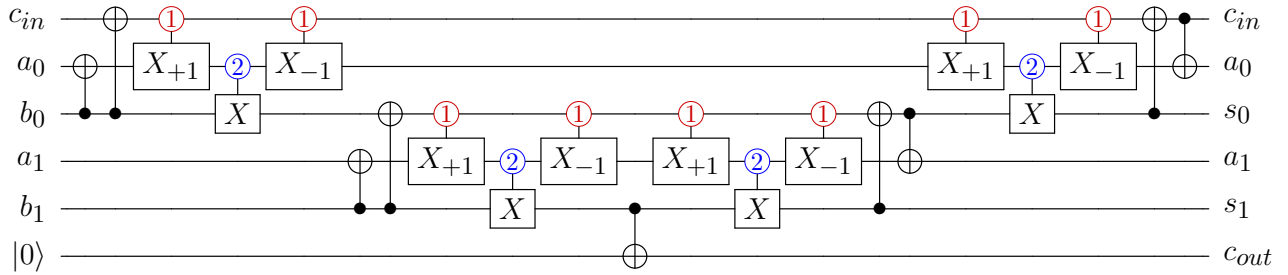


Figure 2.6: The Cuccaro adder of [46] with the Toffoli gates replaced by our efficient decomposition. This only reduces the total depth of the circuit by a constant amount, i.e. no asymptotic benefit is obtained. There are several simplifications which can be made to this circuit, most notably the controlled X_{-1} followed immediately by a controlled X_{+1} in several places. For clarity we've kept these gates to see the direct replacement of our Toffoli decomposition into existing circuits.

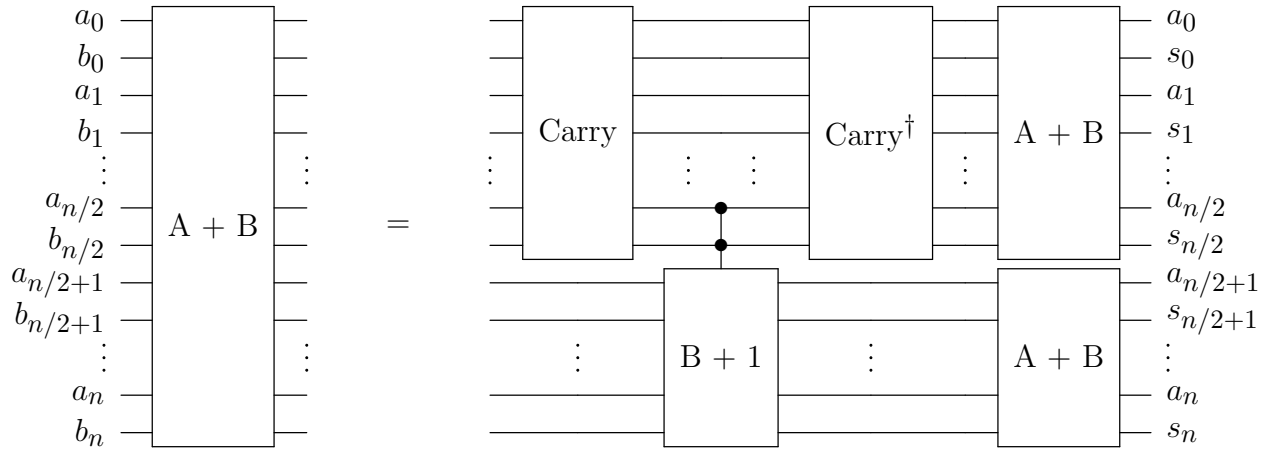


Figure 2.7: Decomposing the $A+B$ circuit with intermediate qutrits. We take as input two n qubit registers and output the sum S onto the bits of register B while leaving register A unchanged. In this approach, we first decide if there will be a carry generated on the top half of the inputs. If so, we apply a $+1$ gate to the bottom half of the inputs (specifically on the bits of B) and then recursively add the first half of A and B and second half of A and B in parallel. The carry circuit outputs an encoded carry status on $a_{n/2}, b_{n/2}$. The controlled $+1$ can be implemented by just modifying the initial X_{+1} and final X_{02} gates of the incrementer to be controlled by the final output carry status. This decomposition is $O(\log^3 n)$ depth provided the Carry circuit is $O(\log n)$ depth and the incrementer is $O(\log^2 n)$ depth.

A	B	Carry Status
0	0	k
0	1	p
1	0	p
1	1	g

C_i	C_j	C_{out}
k	k	k
k	p	k
k	g	g
p	k	k
p	p	p
p	g	g
g	k	k
g	p	g
g	g	g

Figure 2.8: On the left is the encoding for generate (g), propagate (p), and kill (k) carry statuses. On the right is the result of combining two input carry statuses C_i and C_j with C_i corresponding to the less significant bits. Notice the order matters, e.g. $k + g = g$ but $g + k = k$.

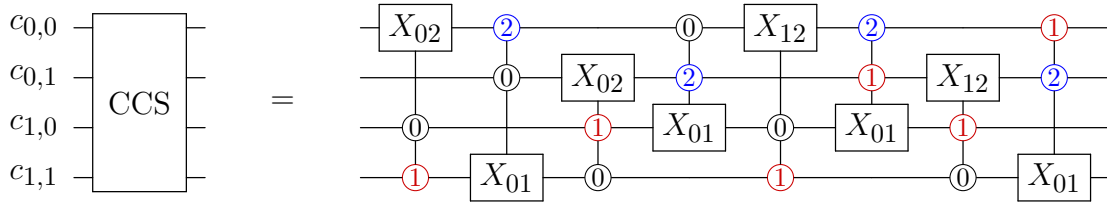


Figure 2.9: Realization of the truth table of Table 2.2 as the gadget “Combine Carry Status” (CCS). While this gate is expensive in terms of two qutrit gates, it is constant depth.

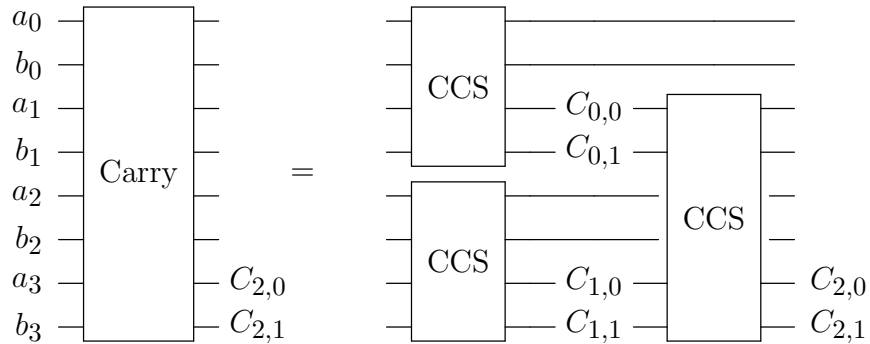


Figure 2.10: Realizing the Carry operation in sublinear depth for $n = 4$ inputs using the CCS gadget. The result is a (11) if and only if a carry is generated, while leaving all of the remaining bits as junk, possibly in ternary states. The CCS blocks always take four qubits as inputs, the first and last two bits and output a binary output on the last two inputs. In the context of the $A + B$ adder, we take this output carry status and use it to control an incrementer on the more significant bits. Afterwards, we would apply the inverse of the cascade on the right to return to the original inputs; this step is omitted here.

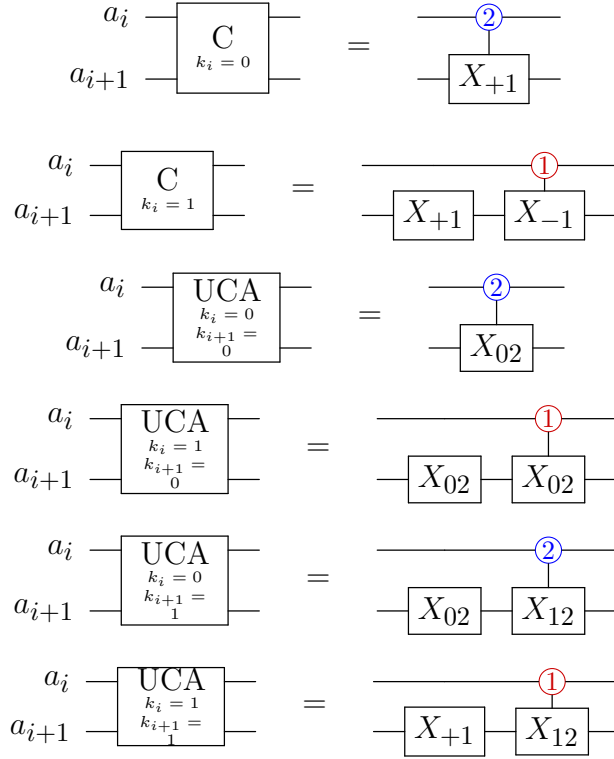


Figure 2.11: The carry (C) and UnCompute and Add (UCA) gadgets used for linear $+K$ adder circuit. There are several instances which are a function of the specific k_i and k_{i+1} values.

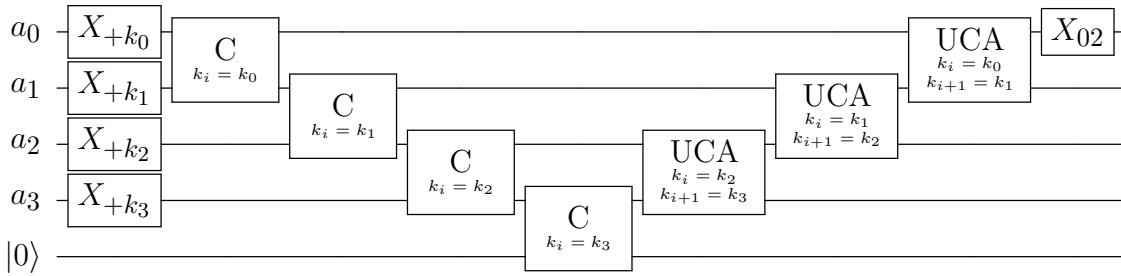


Figure 2.12: Linear $+K$ adder, with carry out on a register of size 4. We assume $k_0 = 1$ and we use the gadgets of Figure 2.11. For no carry out, i.e. $+K \pmod{2^{|K|}}$ simply omit the final C component in the cascade. Note a X_{+1} on an ancilla has the same effect as a X_{01} allowing us to use the same C gadget.

$$A \xrightarrow{/n} \boxed{+K} \xrightarrow{/n} = \xrightarrow{/n} \boxed{+K_0} \boxed{+K_1} \dots \boxed{+K_M} \xrightarrow{/n} A + K$$

Figure 2.13: The Decomposition of the $+K$ circuit into a sequence of $+K_i$ circuits, for $i \in [0, M]$, where M is a constant.

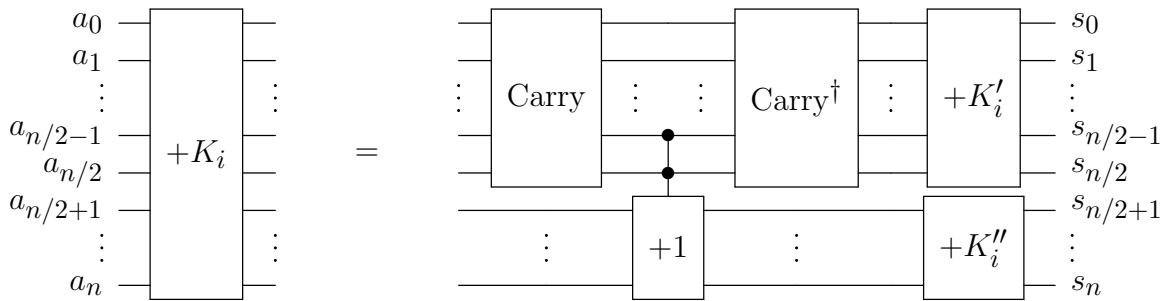


Figure 2.14: The decomposition of the $+K_i$ blocks of Figure 2.13 in $O(\log^3 n)$ depth. Notice again, the controlled incremter is done by adding two controls, the carry status output from the Carry circuit to the first and final gate of the incremter. The truth table for this transformation has been omitted for simplicity.

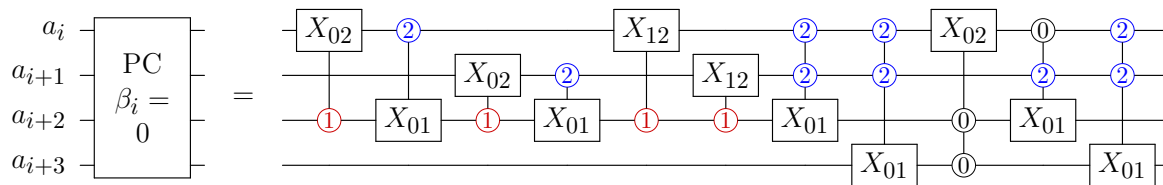


Figure 2.15: The Prepare Carry (PC) circuit for when $\beta_i = 0$. This takes four bits of input A and the known constant β_i and outputs on the last two bits the carry status for this group of bits on its own, adding $(a_i a_{i+1} a_{i+2} a_{i+4}) + (0000)$. The other two inputs are left in possibly ternary states.

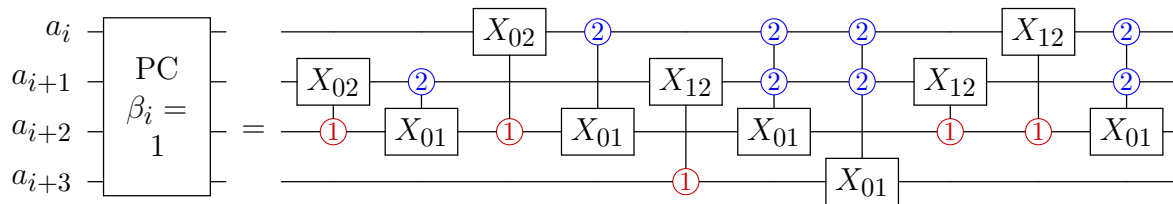


Figure 2.16: The Prepare Carry (PC) circuit for when $\beta_i = 1$. This takes four bits of input A and the known constant β_i and outputs on the last two bits the carry status for this group of bits on its own, adding $(a_i a_{i+1} a_{i+2} a_{i+4}) + (1000)$. The other two inputs are left in possibly ternary states. The truth table for this transformation has been omitted for simplicity.

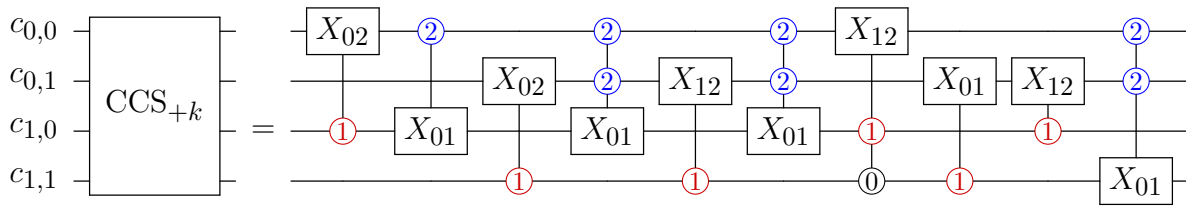


Figure 2.17: Similar to the Combine Carry Circuit (CCS) of Figure 2.9, the CCS_{+K} gadget combines carry statuses of the type found in Table 2.3. This gadget always leaves the final two inputs as the new carry status while leaving the other two inputs possibly in ternary states. The truth table for this transformation has been omitted for simplicity.

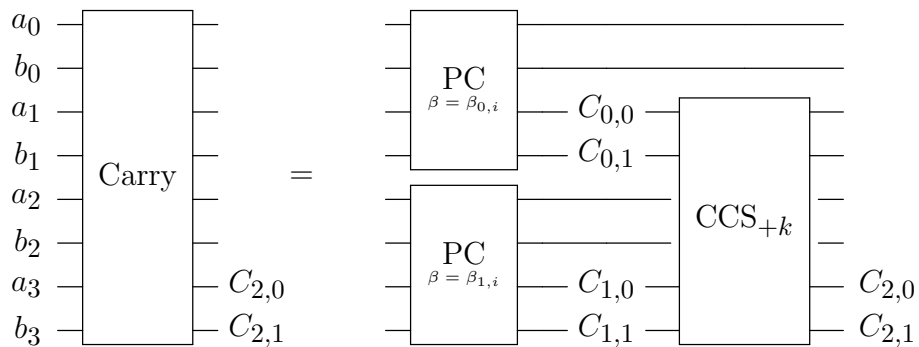


Figure 2.18: Using the PC and CCS_{+K} gadgets, we can produce an $O(\log n)$ depth Carry circuit for the $+K_i$ circuit.

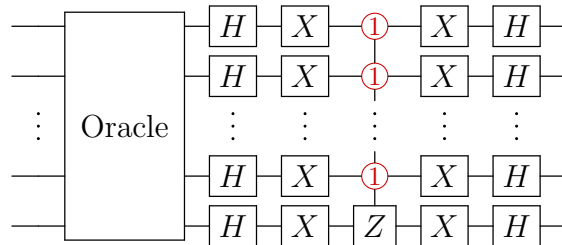


Figure 2.19: Each iteration of Grover Search has a multiply-controlled Z gate. Our logarithmic depth decomposition, reduces a log M factor in Grover's algorithm to $\log \log M$.

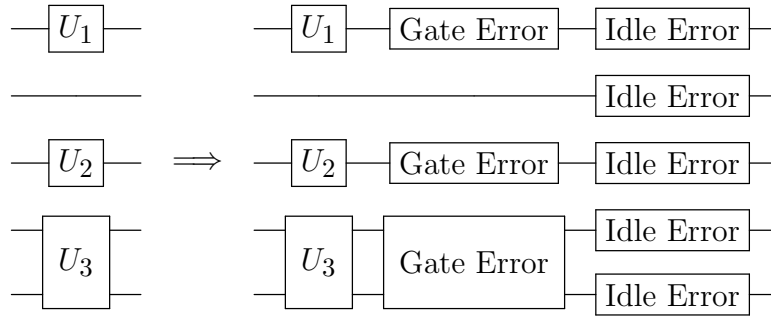


Figure 2.20: This **Moment** comprises three gates executed in parallel. To simulate with noise, we first apply the ideal gates, followed by a gate error noise channel on each affected qudit. This gate error noise channel depends on whether the corresponding gate was single- or two-qudit. Finally, we apply an idle error to every qudit. The idle error noise channel depends on the duration of the **Moment**.

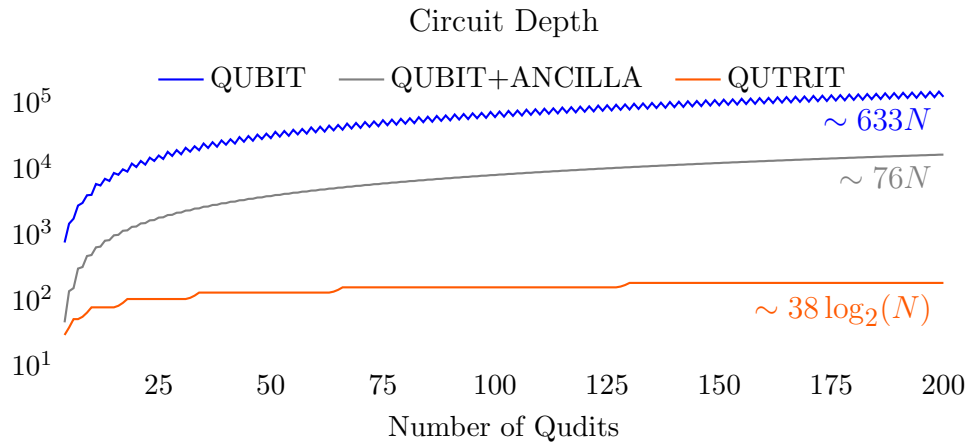


Figure 2.21: Exact circuit depths for all three benchmarked circuit constructions for the N -controlled Generalized Toffoli up to $N = 200$. Both QUBIT and QUBIT+ANCILLA scale linearly in depth and both are bested by QUTRIT's logarithmic depth.

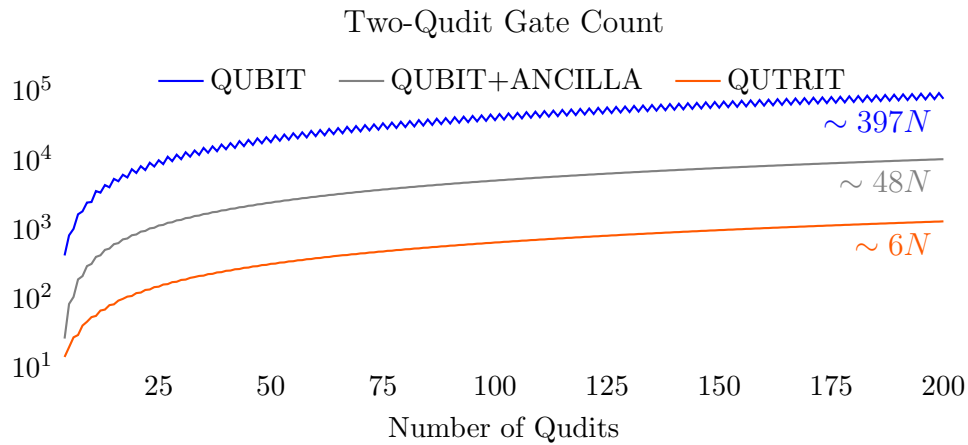


Figure 2.22: Exact two-qudit gate counts for the three benchmarked circuit constructions for the N -controlled Generalized Toffoli. All three plots scale linearly; however the QUTRIT construction has a substantially lower linearity constant.

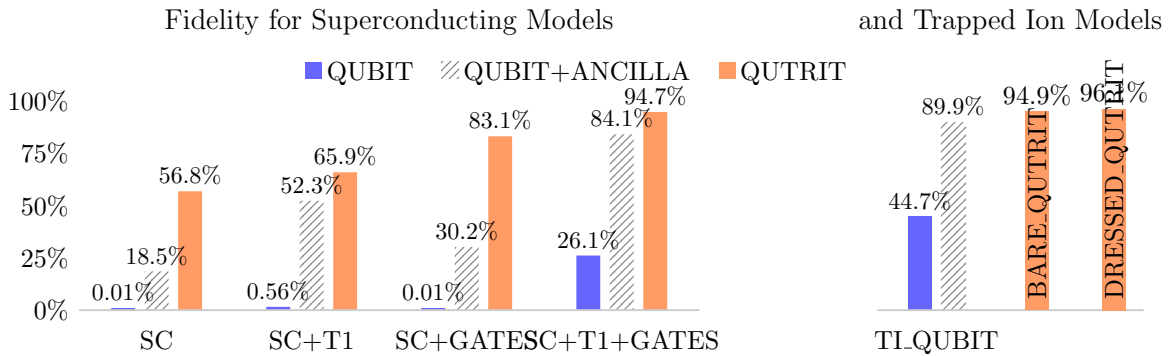


Figure 2.23: Circuit simulation results for all possible pairs of circuit constructions and noise models. Each bar represents 1000+ trials, so the error bars are all $2\sigma < 0.1\%$. Our QUTRIT construction significantly outperforms the QUBIT construction. The QUBIT+ANCILLA bars are drawn with dashed lines to emphasize that it has access to an extra ancilla bit, unlike our construction.

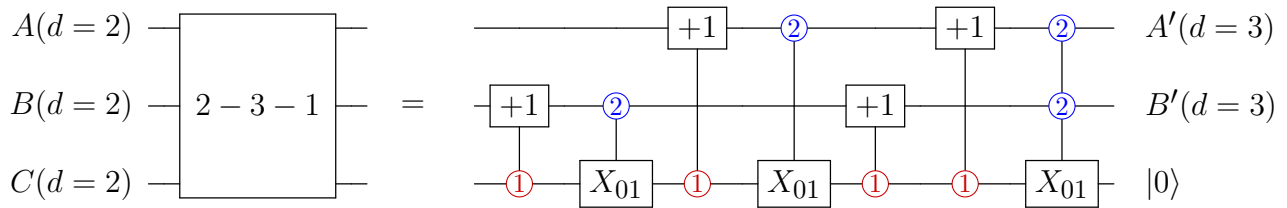


Figure 2.24: The compression of 3 qubits into 2 qutrits and an ancilla, $|0\rangle$. All $+1$ gates are done modulo 3. Using a sequence of qutrit gates, we can transform three input qubits into the desired ancilla. When A, B and C are not going to be used for a long time in the circuit, they can be temporarily repurposed as an ancilla bit elsewhere in the circuit. When we want to operate on these stored bits, we run the inverse of this circuit using *any* ancilla for the third qubit.

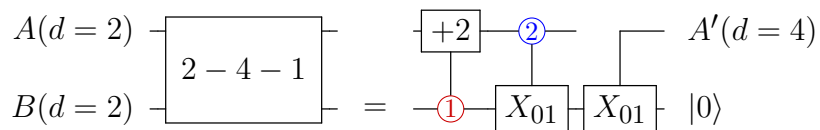


Figure 2.25: The compression of 2 qubits into a single ququart and generating an ancilla, $|0\rangle$. The $+2$ gate here is done modulo 4. This operation takes as input two qubits, A and B, and produces a single ququart and an ancilla $|0\rangle$. To do this, we need only 3 two-ququart gates. Similarly, to retrieve the stored information, we can do the inverse of this operation using *any* ancilla for the second qubit.

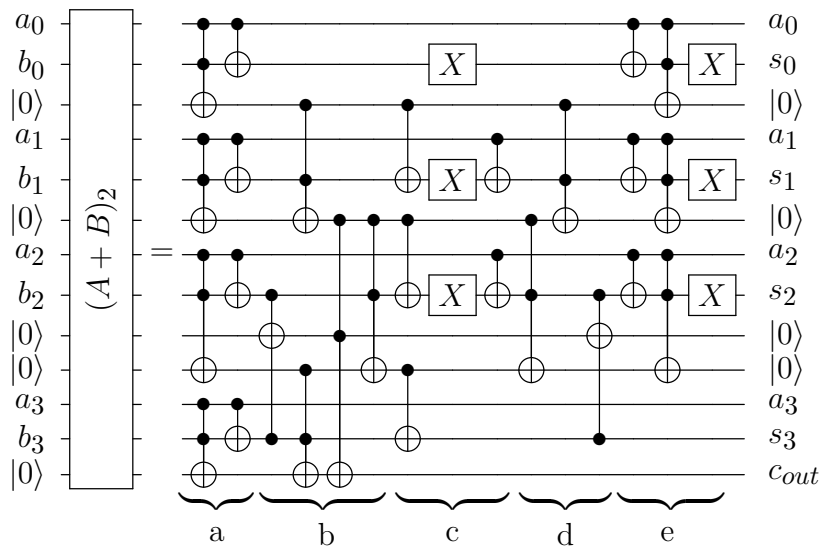


Figure 2.26: An adder circuit from [58] on two four-bit registers A and B with a carry-out bit using ancilla. The sum S is computed in-place on register B while A is untouched and the ancilla are restored to $|0\rangle$. We use this as a sub-component of our general decomposition. Each of the ancilla in this circuit can be generated from other input qubits not shown here via our compression circuits. Part a of the circuit computes carry generate and propagate for each bit position. Part b computes the carry-in for every bit position. Part c does the addition, storing the output in register B . Parts d and e uncompute b and a respectively, restoring the ancilla back to $|0\rangle$.

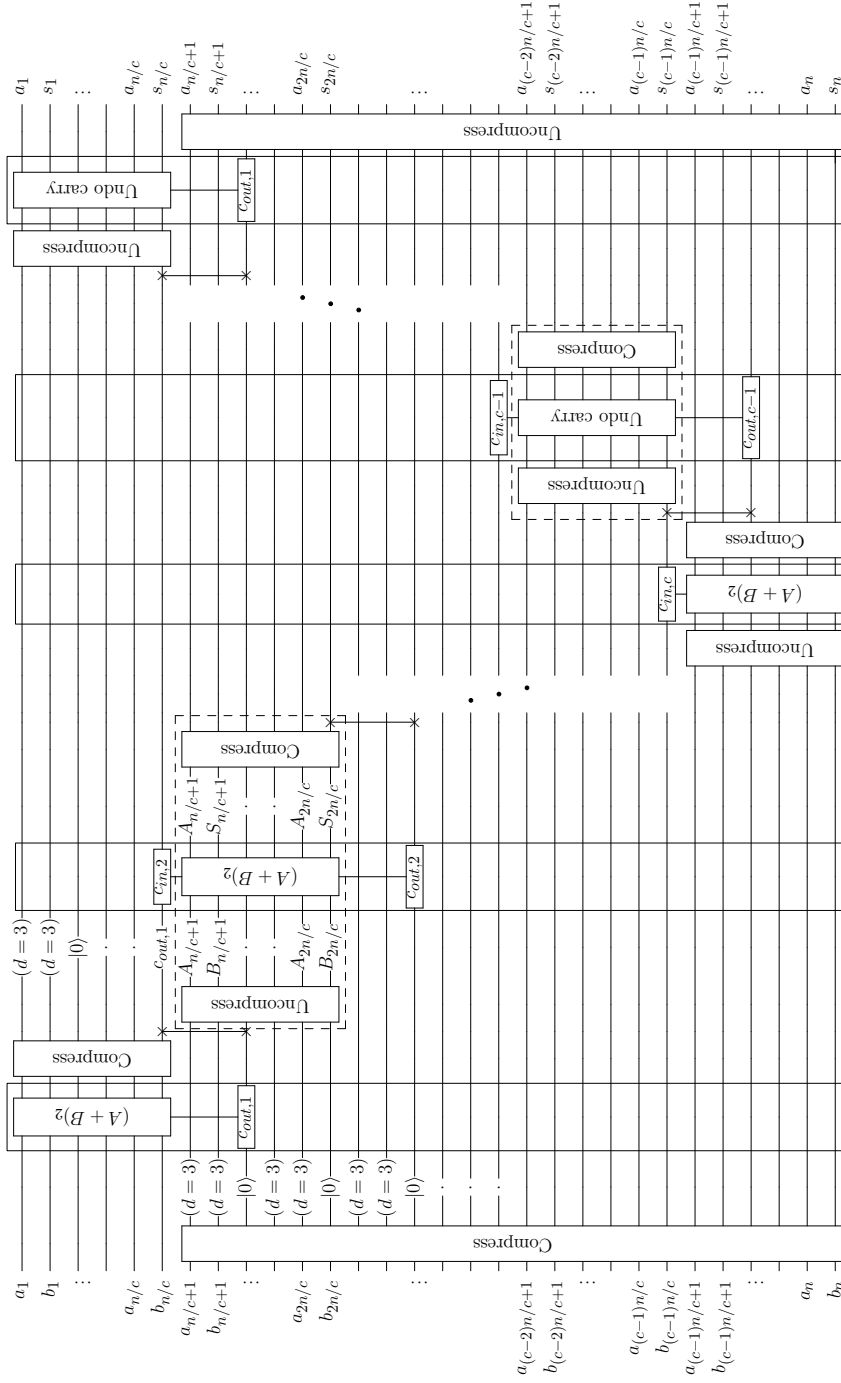


Figure 2.27: Our $A + B$ adder that uses no external ancilla. The variant shown here for $c = 5$ uses 2-3-1 compression to generate one ancilla (marked as $|0\rangle$) for every three unused qubits, storing their values in two qutrits (marked as $d = 3$). A box is drawn around every $(A + B)_2$ and Undo carry gate to indicate that they use all the generated ancilla across the circuit. $c_{out,i}$ or $c_{in,i}$ is included on some of the gates to indicate when the carry-in and carry-out versions are used and on which ancilla the carry-out is stored. The SWAP gates (pairs of \times in the diagram) simply move a carry-out bit to another ancilla where it is used as the next carry-in. The two blocks of gates shown with dashed lines are repeated $c - 2 = 3$ times along the diagonal indicated. If 2-4-1 compression is used, an ancilla is generated for every two unused qubits so only $c = 4$ blocks are needed. The depth of this circuit is $O(\log n)$.

CHAPTER 3

ARCHITECTURAL TRADE-OFFS IN EMERGING TECHNOLOGY: NEUTRAL ATOM ARCHITECTURES

1

3.1 Introduction

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 [186, 93, 3, 171]. However, each of these systems have unique scalability challenges [110, 34]. 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 [76]. Trapped ion machines, despite many promising results, have fundamental challenges in controllability [137]. 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 [168] or Shor’s [78].

These challenges are fundamental to the underlying technology. Consequently, current approaches which aim to reduce error via software and push the limits of current devices are insufficient for long-term scalability. In recent years there has been a number of improvements to the compilation pipeline stretching across the hardware-software stack [136, 178, 184, 44, 89, 80, 144, 139]. Numerous studies have explored reductions in quantum circuit gate counts, depths, and communication costs via heuristics and optimizations, but cannot overcome the fundamental scalability limitations of the technology.

1. JMB’s contributions in this chapter include primary development of the compiler (with significant help from AL), experimentation, and conceptual development, and analysis. The majority of the atom loss section is from AL with help by JMB and CD.

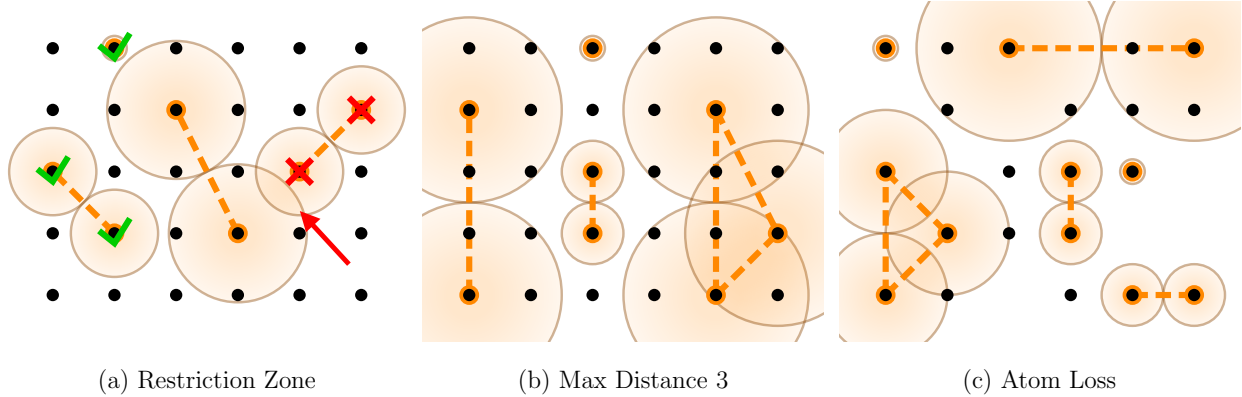


Figure 3.1: Examples of interactions on a neutral atom device. (a) Interactions of various distances are permitted up to a maximum. Gates can occur in parallel if their zones do not intersect. The interaction marked with green checks can occur in parallel with the middle interaction. (b) The maximum 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.

An alternative approach is to consider emerging quantum technologies and new architectures. In this work, 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 [163]. 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 [119].

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 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 NA 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 NA system is resilience to loss of atoms, avoiding expensive reloads.

In this work, we explore the trade-offs in neutral atom architectures to assess both current viability and near future prospects. We extend current compilation methods to directly account for the unique NA constraints like long-range interactions, areas of restriction, and native implementation of multiqubit gates. We propose several coping strategies at the hardware and software level to adapt to loss of atoms during program execution; we evaluate the tradeoff of execution time and resilience to atom loss versus program success rate.

The field of quantum computation is still early in development and no clear winner for underlying hardware has emerged. It is vital to consider new technology and evaluate its potential early and often to determine viability as it scales. In this work, we do just that, considering a neutral atom architecture of size comparable to target hardware sizes for competing technologies. Despite comparably higher gate errors and lack of large scale demonstration, we determine if the unique properties offered by this new technology enable more efficient computation at scale. Perhaps more importantly, this work can serve as a guide for hardware developers. We demonstrate that the fundamental problem of atom loss can be mitigated via software solutions and doesn't need to be highly optimized at the hardware level. This allows hardware engineers to focus on other fundamental problems which cannot easily be mitigated by software, such as gate error rate.

In this section we introduce a scalable neutral atom architecture based on demonstrated physical implementations which permit long range interactions and native multiqubit gates.

The specific major discussions center on the following:

- Adapt current quantum compiler technology by extending prior work to explicitly account for interaction distance, induced restriction zones, and multiqubit gates.
- Evaluate system-wide implications of these properties, specifically reduced gate counts and depth at the cost of increased serialization. Our compiler exploits the gain while mitigating this cost.
- Demonstrate, via simulation based on experimental results and through program error analysis, the ability of NA systems to quickly surpass competitors in the intermediate-term despite currently worse gate errors.
- Model sporadic atom loss in NA systems and propose hardware and compiler solutions to mitigate run time and program error rate overheads. We explore each strategy’s resilience to atom loss, the effect on expected program success rate, and overall run time.

3.2 Relevant Background

3.2.1 *Quantum Computation and the Gate Model*

Most quantum programs for gate-based quantum computation are expressed in the quantum circuit model. On most hardware platforms, only single and two qubit gates are supported, requiring complex operations to be decomposed into smaller pieces. Furthermore, most hardware only supports a small highly calibrated universal set of gates, requiring input programs be rewritten in terms of these gates. A small piece of a circuit is found in Figure 3.2a.

Two important metrics for quantum programs are the depth of the quantum program, given as the length of the longest critical path from inputs to outputs, and the gate count,

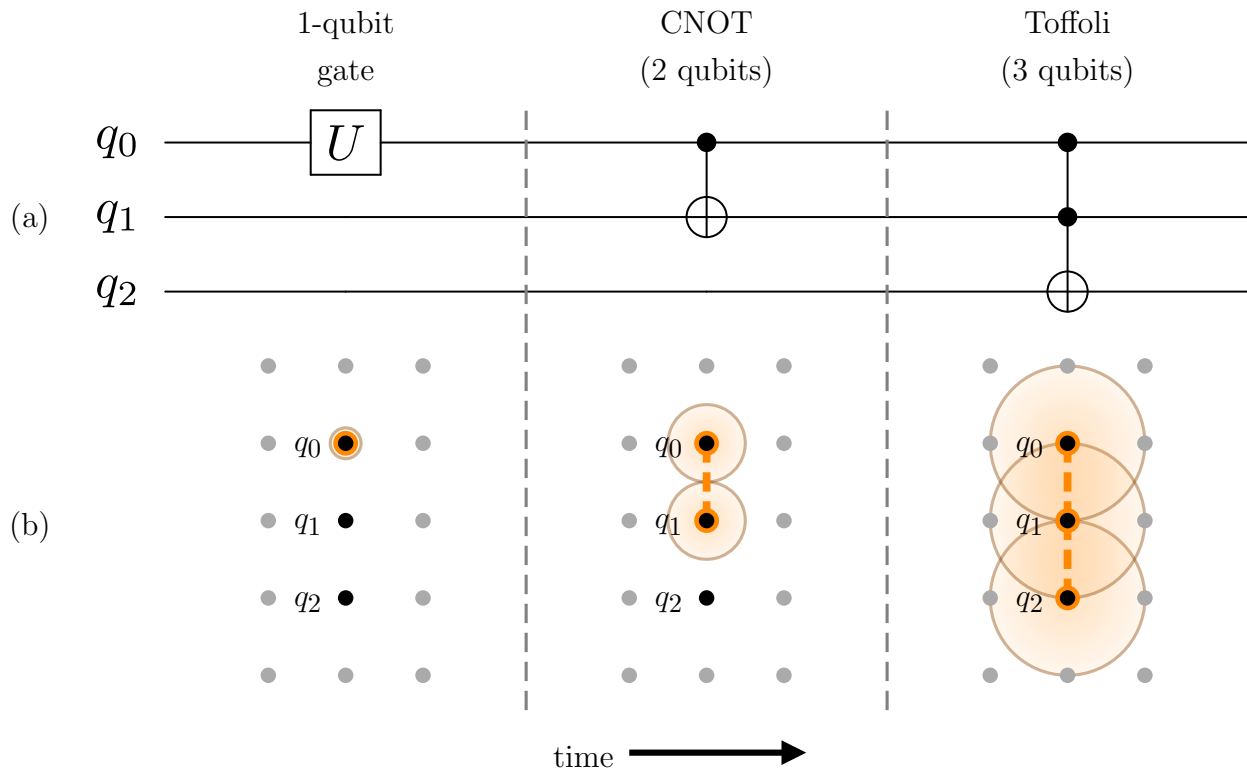


Figure 3.2: A quantum circuit with a 1, 2, and 3 qubit gate translated to interactions on a NA device. These systems allow the execution of multiqubit gates. For 2 and 3 qubit gates the interacting qubits are excited to Rydberg states. Interactions are possible if all interacting qubits are closer than the maximum interaction distance.

that is how many operations it takes to perform the desired algorithm. Both are important for near-term quantum computation which is noise prone. Qubits have limited coherence time, likely erasing a qubit's information by a time limit. Gate error rates are fairly high so computations with many multiqubit gates are less likely to succeed.

3.2.2 The Quantum Compilation Problem

While we will discuss the compilation problem in more detail in the final chapter we give a brief introduction to this problem here. In order to maximize the probability of program success, quantum circuits often undergo extensive compilation and optimization. Compilation generally falls into two main categories: circuit optimization to minimize total number of gates, and translation of input programs to fit the constraints of the target hardware. The latter is often the focus of near-term compilation strategies which break the problem into three main steps: mapping, routing, and scheduling.

In 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 nearby by inserting SWAPs or other communication operations before interacting. 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.

3.2.3 *Neutral Atoms*

We want to briefly introduce some background on the underlying neutral atom technology. A nice introduction can be found in [88]. Atoms in a NA system are trapped via reconfigurable, optical tweezer arrays. These atoms can be arranged in one, two, or even three dimensions [62, 18, 109, 19]. We consider regular square 2D geometries in this work, 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 [151] have been demonstrated. The loading of qubits into the array is relatively slow, on the order of one second, compared to program execution which usually takes milliseconds.

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 [97]. These Rydberg interactions enable multiple atoms at once to interact strongly and are used to realize multiqubit gates [119]. Furthermore, due to the long range of these interactions, gates between qubits which are not directly adjacent in the atom array are feasible. Importantly, these interactions induce a zone of restriction as a function of the distance. Two gates can only occur in parallel if their restriction zones do not overlap.

3.3 **Neutral Atom Compiler and Methodology**

3.3.1 *Mapping, Routing, and Scheduling*

In this work we focus on adapting currently available and effective compilation methods [185, 13, 98, 8] to directly account for the unique properties of neutral atom architectures. We focus on mapping, routing and scheduling of the quantum compilation problem. Other optimizations, such as circuit synthesis, gate optimization, or even pulse optimization, can be

performed as well, but are not the focus of this work. Many of the primary advantages and disadvantages of the neutral atom hardware can be reduced to modifications of the hardware topology or interaction model given to the compiler.

We represent the underlying topology as a graph, where nodes are hardware qubits and edges are between nodes which can interact. We model the underlying hardware as a 2D grid of qubits and, for a given instance, we fix the maximum interaction distance d_{max} . Therefore, there is an edge between nodes u, v if $d(u, v) \leq d_{max}$. We model the restriction zones as circles of radius r centered at each of the interacting qubits. In this work we model this radius as $f(d) = d/2$ where d is the maximum distance between interacting qubits, pairwise. We have ensured this generalizes to any number of qubits in order to support multiqubit gates. In practice, devices may require a different function of d . The larger this radius the fewer possible parallel interactions can occur.

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. Therefore the dominant solutions are heuristics. We have extended prior work on lookahead heuristics. 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:

$$w(u, v) = \sum_{\ell \geq \ell_c} e^{-|\ell_c - \ell|}$$

where $w(u, v)$ is the weight between program qubits u and v , ℓ is a layer of the program, and ℓ_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.

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. For every subsequent qubit in this graph we consider all possible assignments to hardware 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. 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.

For routing and scheduling, we proceed layer by layer, considering operations in the frontier as potential gates to execute. Ideally, we would execute all operations in the frontier in parallel, however if interacting qubits are not close enough or the zones of restriction intersect, this isn't possible. Instead, we first select to execute operations in the frontier which do not have intersecting zones. For any remaining long distance operations in the frontier, we compute the best set of SWAPs to move qubits within the interaction distance. We want to select a path of SWAPs with two goals in mind: the shortest path and the least disruptive to future interactions. This leads to the following scoring function:

$$s(u, h) = \sum_v [d(\varphi(u), \varphi(v)) - d(h, \varphi(v))] \times w(u, v) + [d(h, \varphi(v)) - d(\varphi(u), \varphi(v))] \times w(\varphi^{-1}(h), v)$$

where h is the new location for u after the SWAP. We choose the h which maximizes this function but is also strictly closer to the most immediate interaction for u and v . In this function, moving further away from future interactions or displacing the qubit in position h by moving it far from its future interactions is penalized. This guarantees the qubit always moves

closer to its target. The SWAP is executed if it can run parallel with the other executable operations, otherwise we must wait. We proceed until all operations have been executed. Our compiler and evaluation source code is available at [1].

To scale target programs up to hundreds to thousands of qubits, the heuristics used are fairly simple and fast. One clear advantage of NA is that simpler and faster heuristics will suffice in practice because large interaction distances make the topology densely connected thus saving communication cost.

We validated our compiler by compiling small programs via IBM’s Qiskit compiler [8] with lookahead enabled against our compiler with maximum interaction distance (MID) set to 1 and no restriction zones for two benchmarks, one parallel and one not. In both cases, our compiler closely matched Qiskit in both gate count and depth.

3.3.2 *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 [21], a common quantum benchmark, with the all 1s oracle to maximize gates, Cuccaro Adder [46], a ripple carry adder with no parallelism, the CNU gate [15], a logarithmic depth and highly parallel decomposition of a very common subcircuit, QFT Adder [161], a circuit with two QFT components and a highly parallel addition component, and QAOA for MAX-CUT [64], a promising near-term algorithm, on random graphs with a fixed edge density of 0.1.

3.3.3 *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 $\text{hypot}(9, 9) \approx 13$). In relevant benchmarks we compile with decomposed multiqubit gates and without. All experiments were run using on a machine using Python 3.7 [180], 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.

3.4 Unique Advantages of Neutral Atom Architectures

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 NA 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.

3.4.1 Long Range Interactions

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 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 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.

3.4.2 Native Multiqubit Gates

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 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 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.

3.5 Error Analysis of Neutral Atom Architectures

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.

3.6 Unique Challenge: Sporadic 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 [43]. 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 [69]. 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% [113] uniformly across all measured atoms. Detecting atom loss is done via fluorescence which takes on the order of 6ms.

We propose several coping mechanisms to the loss of atoms 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 [143]. 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.

- *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 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 [47] 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.

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.

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, the ideal case for sustained loss. The non-rerouting strategies, while a fast solution, offer limited atom loss recovery. 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 a disconnected topology failure case, but also the additional failure case of not having the

space in any direction to shift the qubits in the event of atom loss. As a result, both are only able to sustain 50% atom loss at higher interaction distances.

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 about 2/3 of shots succeed without atom loss. 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 is able to achieve the best routing and success rate out of all the atom loss strategies. Both of the rerouting strategies have lower rates since they tend to add more SWAPs per atom loss. But, since compiling to a smaller MID before rerouting means the interaction distance is exceeded less often, it requires fewer SWAPs, boosting its rate over simply rerouting.

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 0.0068% chance of atom loss due to atom collision in a vacuum. The overhead times for CNU are seen 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 less fixes with swaps, the overhead time tends to be smaller at lower interaction distances. 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 3.14. 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.

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 3.13 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.

3.7 Remarks

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 in compiled circuits 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 to adapt to this loss 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.

Popular competitor qubits have a head start on neutral atoms in terms of error rate and device sizes. In simulation, we have demonstrated our large gate count and depth savings

give advantage over superconducting systems. SC systems are often easy to increase in size, but fabrication variability and limited connectivity limit their effectiveness. Trapped-ion systems offer many of the same advantages as neutral atoms such as global interactions and multiqubit gates but at the cost of parallelism. Ions also have stronger trapping potential, mitigating loss. Unfortunately, trapped ion systems will struggle to scale and maintain these properties. These systems have limited parallelism and are held in one dimensional traps limited to around 100 qubits. To scale, systems connect multiple traps with higher inter-trap communication cost [137]. Neutral atom systems are theoretically capable of maintaining their advantages as they scale.

Even at a small scale, the unique properties of the NA systems result in compiled circuits which are lower depth and use fewer communication operations translating to an expected higher probability of success. These advantages will become even clearer as devices scale since the device connectivity per size grows much more favorably than other architectures. Our algorithms for compilation are scalable heuristics and will be able to keep up with increasing hardware size well. For atom loss, some techniques will not be favorable for larger device and program sizes, such as full recompilation, however we've shown other more clever and faster techniques are better suited for the problem and will be able to scale. For example, since the speed to adjusting a hardware mapping is on the order of nanoseconds, rather than the microseconds required to perform reloading and fluorescence, we can expect these techniques to remain viable.

In this work we have focused on software and hardware techniques to demonstrate neutral atoms, with our methods, are a viable and scalable alternative to more established technologies. Long-distance interactions and multiqubit operations dramatically reduce communication and depth overheads which translates into lower error rate requirements to obtain successful programs. 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 and prior hardware studies have focused hardware solutions to reduce this probability of loss. 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 and process of co-design which can accelerate the advancement of the hardware tremendously.

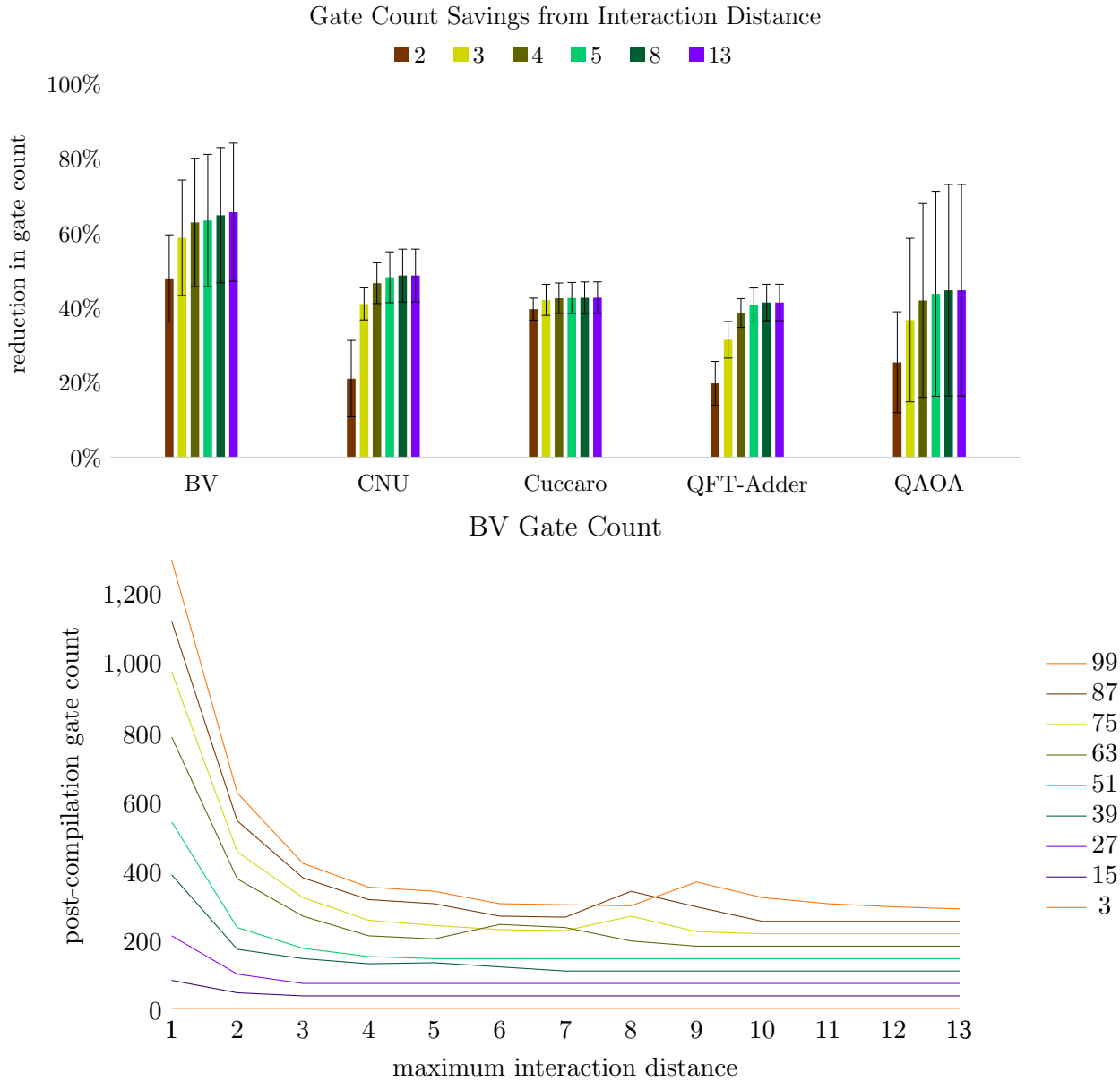


Figure 3.3: Post-compilation gate count across benchmarks. On the top are percent savings over the distance 1 baseline averaged over program sizes up to 100 qubits. Each color is a max interaction distance. Noticeably, there is less additional improvement as the MID increases, indicating most benefit is gained for smaller distances. On the bottom 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.

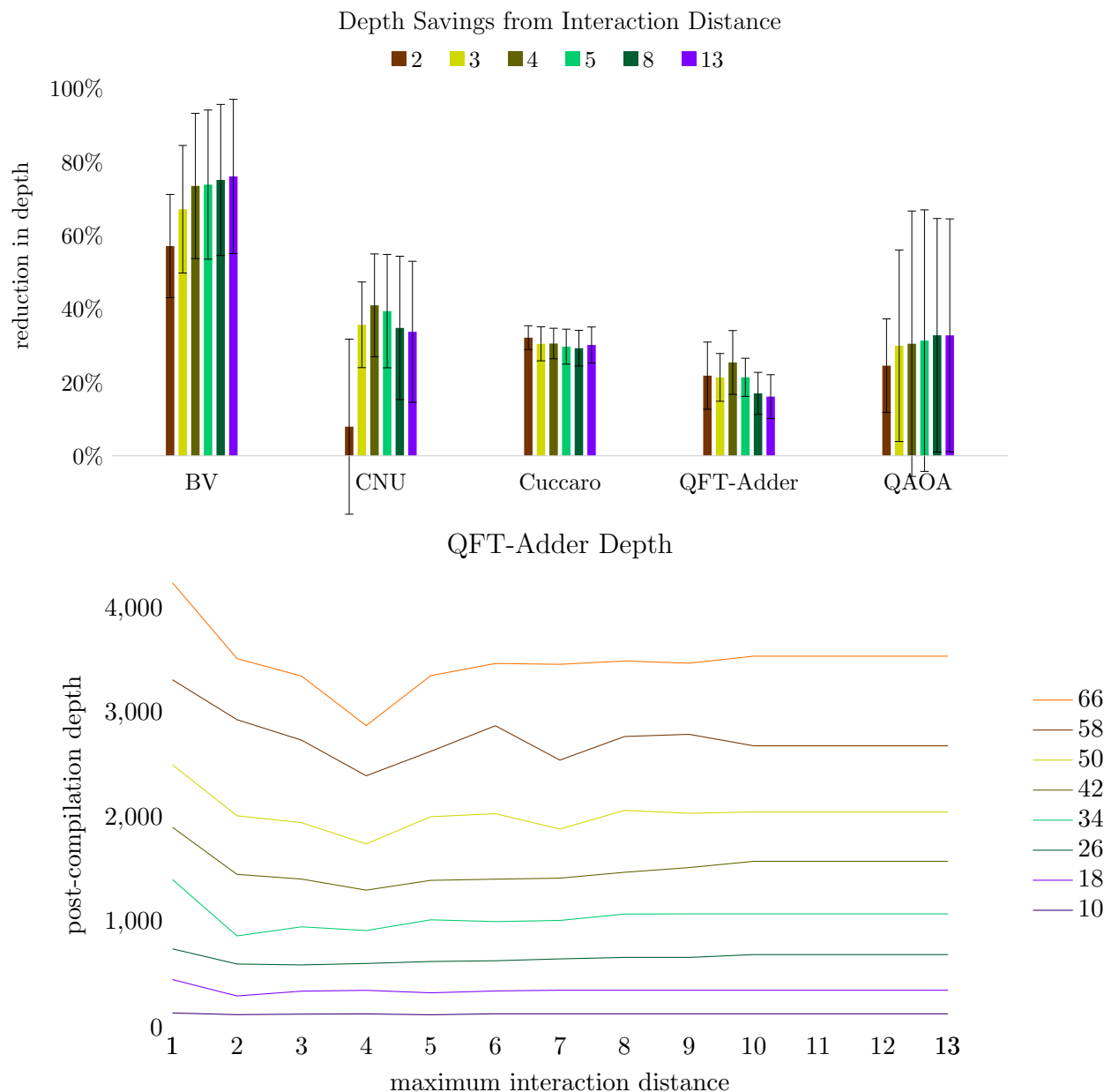


Figure 3.4: Post-compilation depth across all benchmarks. On the top, the reduction in depth over the distance 1 baseline. Each bar is the average over all benchmark sizes. On the bottom we see a similar drop off in post-compilation depth for the QFT-Adder. We’ve chosen this specific benchmark to highlight 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.

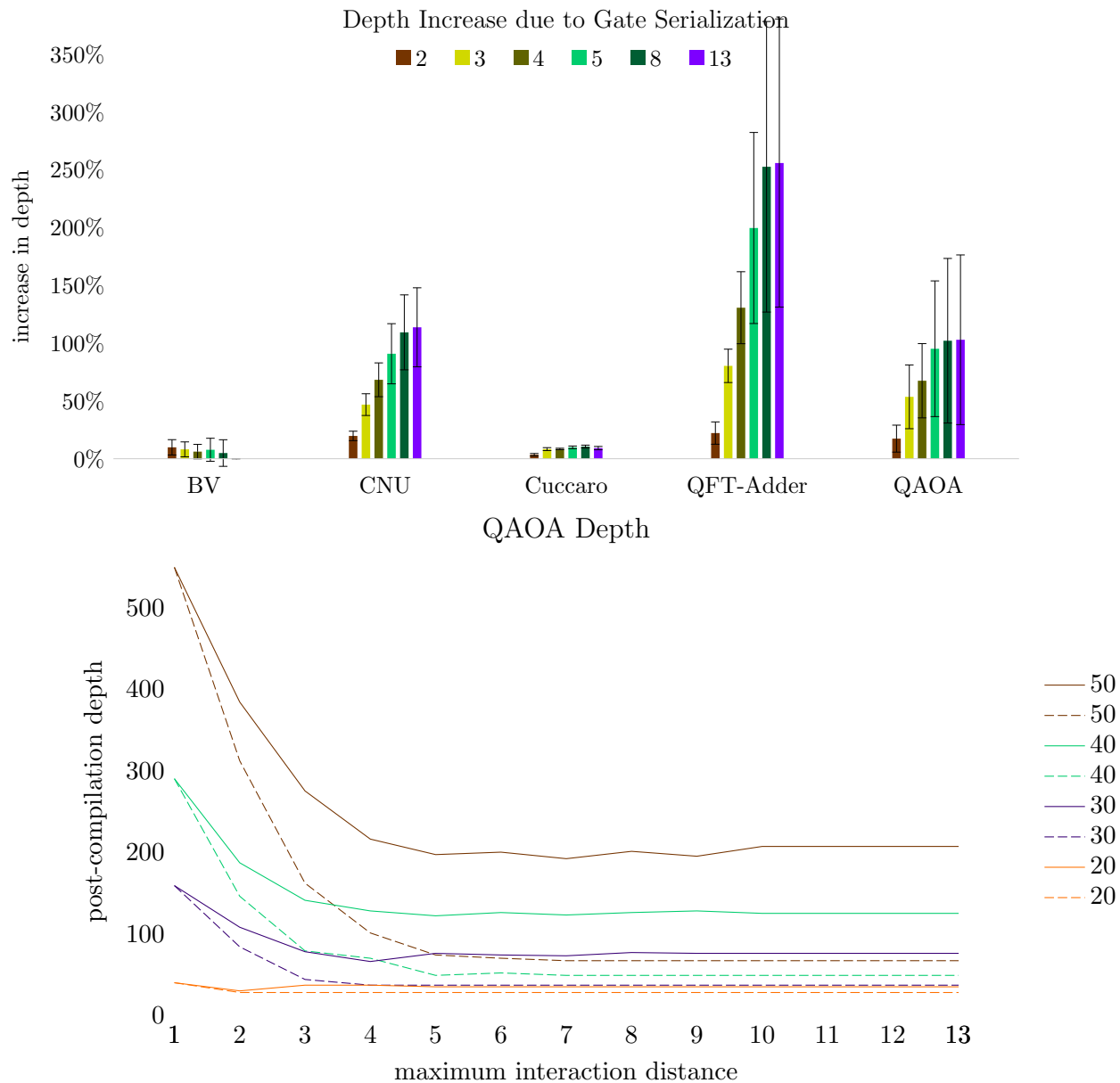


Figure 3.5: The induced restriction zone from interaction distance increases serialization. In the prior results this is hard to discern because compared to low interaction distance the amount of gate savings translates to depth reduction. Here we compare benchmarks compiled with our restriction zone 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 bottom 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.

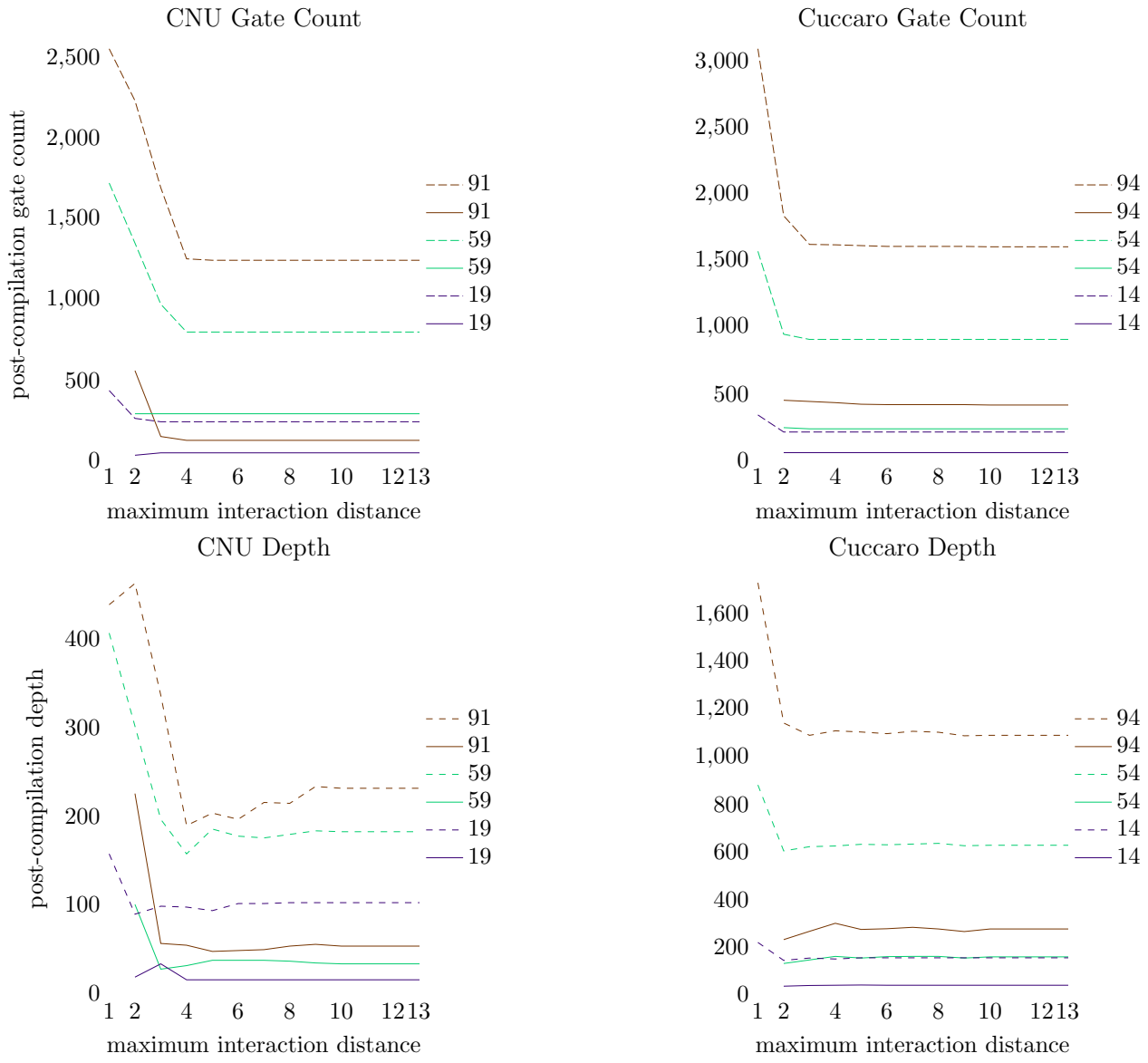


Figure 3.6: Compiling to programs directly to three qubit gates reduces both gate count and depth. Here we highlight a serial and parallel application written to three qubit gates. Here dashed lines are compiled to two qubit gates decomposing all Toffoli gates before mapping and routing. Solid lines 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.

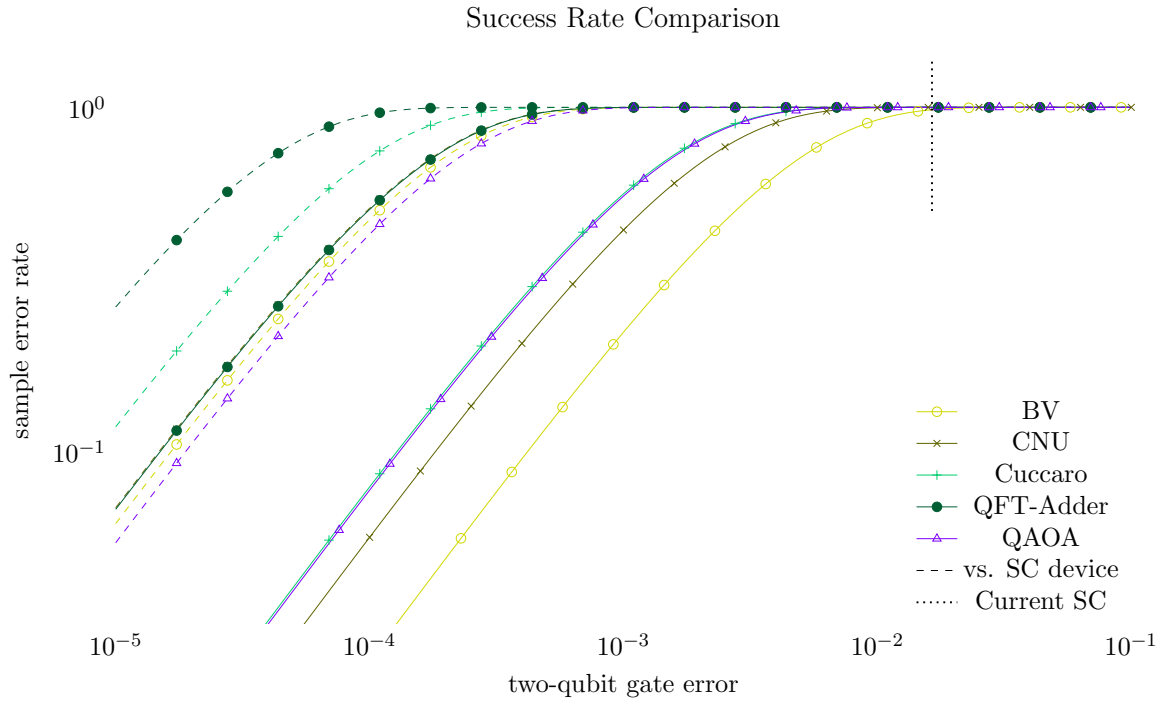


Figure 3.7: Program success rate as a function of two-qubit error rate. Because current NA error rates are lagging behind competitive technologies we scan over a range of two-qubit error rates for each of the benchmarks all on 50 qubit programs (49 for CNU) with max interaction distance of 3. Examining 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.

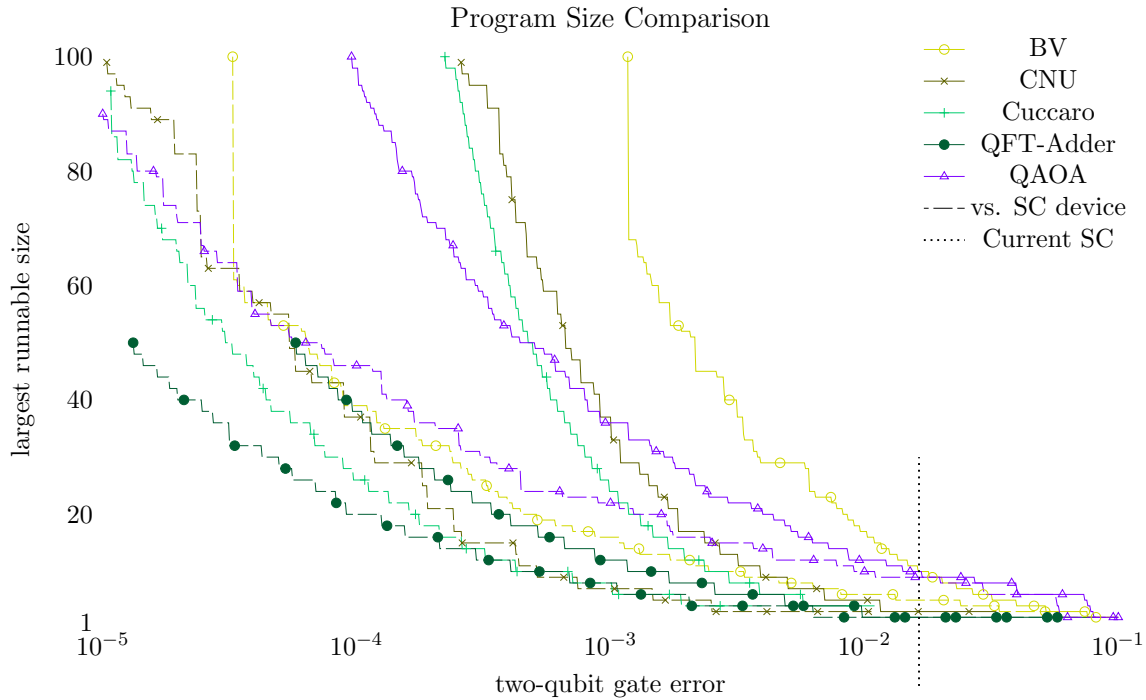


Figure 3.8: Another way to examine the data of Figure 3.7 is to ask, given a desired program success rate, what the required two qubit error rate is. Here we sweep again over two qubit error rates and record the maximum program size to run with success probability greater than $2/3$. Again, examining 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.

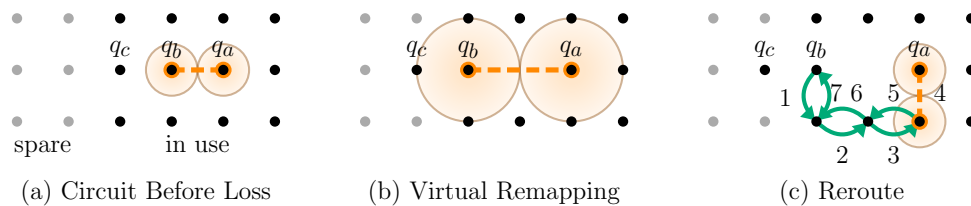


Figure 3.9: Examples of two different atom loss coping strategies. (a) shows the initial configuration of three qubits, with the spare qubits in a light grey, and in use qubits black. (b) Represents how the atoms are shifted into the spare qubits to accommodate a lost atom under the virtual remapping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

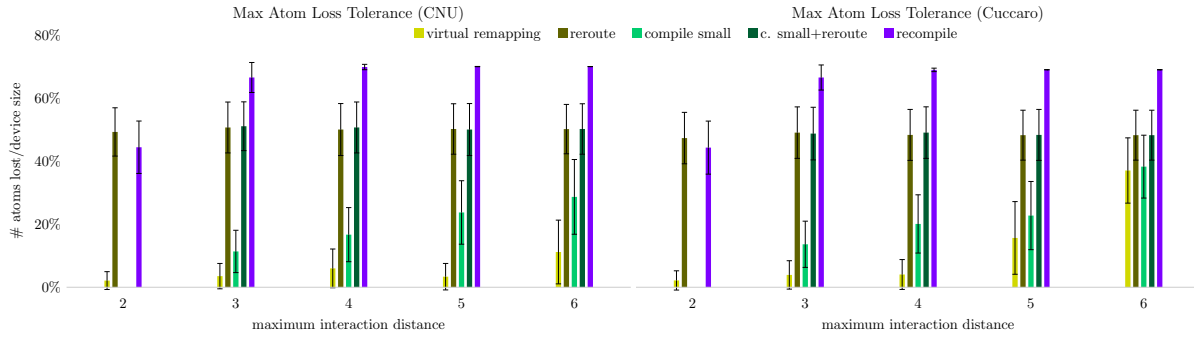


Figure 3.10: Atom loss as a percentage of total device size which can be sustained before a reload of the array is needed. Each program is 30 qubits on a 100 qubit device. As the interaction distance increases most strategies can sustain more atom loss. Strategies like full recompilation can sustain 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.

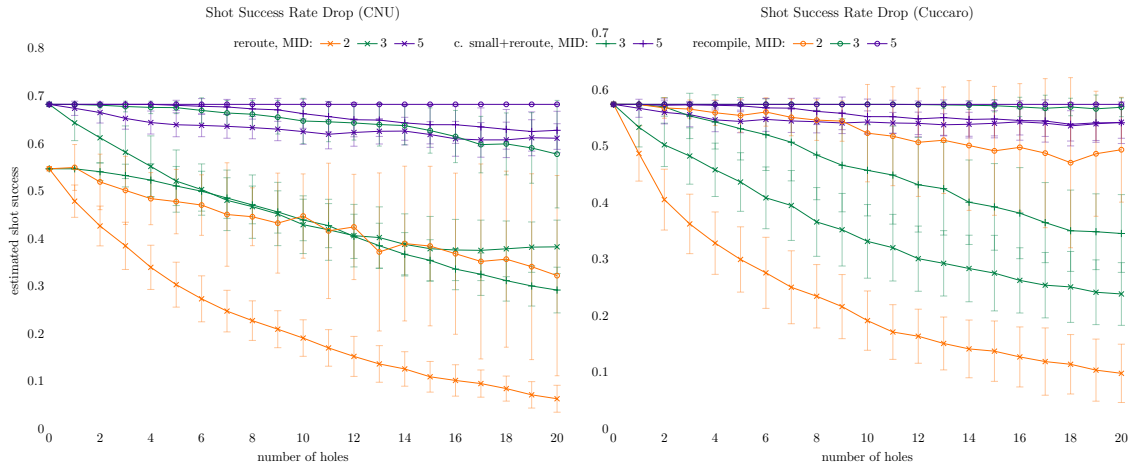


Figure 3.11: For strategies which modify the program such as recompilation or rerouting strategies, additional gates could be added leading to a lower overall success rate. Here we trace the success rate of our three program modifying strategies. The full recompilation strategy (circles) is a rough upper 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.

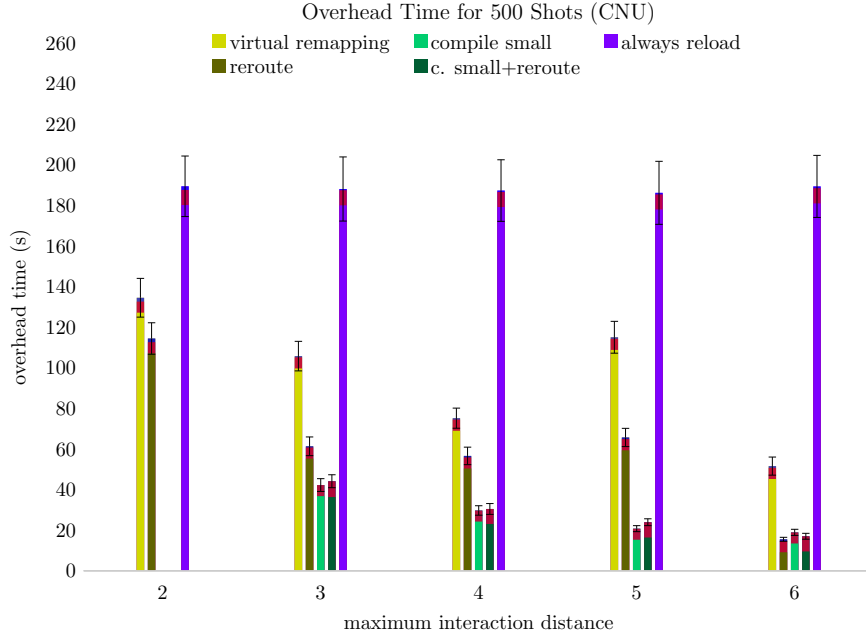


Figure 3.12: Strategies that are able reduce the number of reloads necessary greatly reduce the overhead time when running circuits. Here we show the overhead time for all strategies except recompilation. The proportion of time dedicated to reloading is shown by the dominate color in each bar, followed 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.

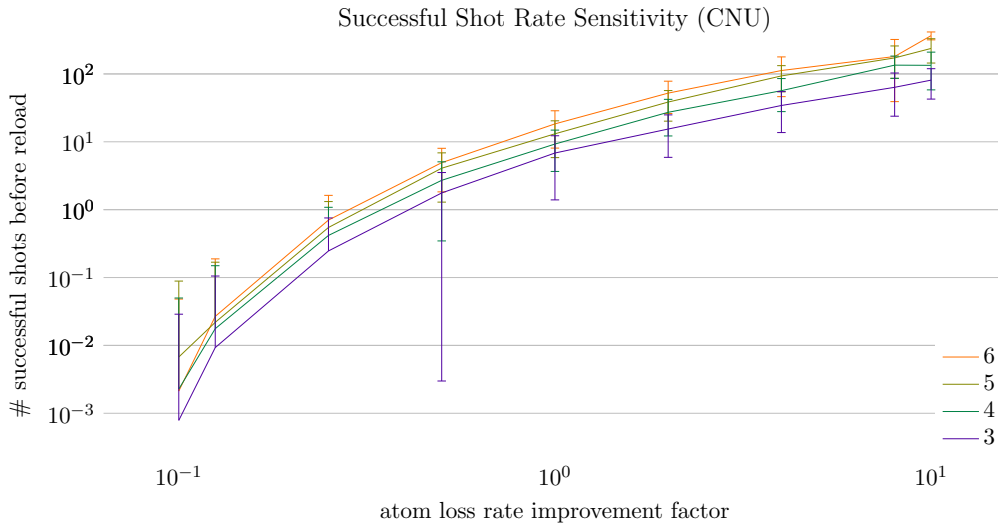


Figure 3.13: Sensitivity to the rate of atom loss for the balanced *Compile Small and Reroute* strategy. In prior experiments we used a fixed rate of 2% atom loss. For larger systems this rate could be worse and in the future we might expect this rate to be much better. For each interaction distance 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.

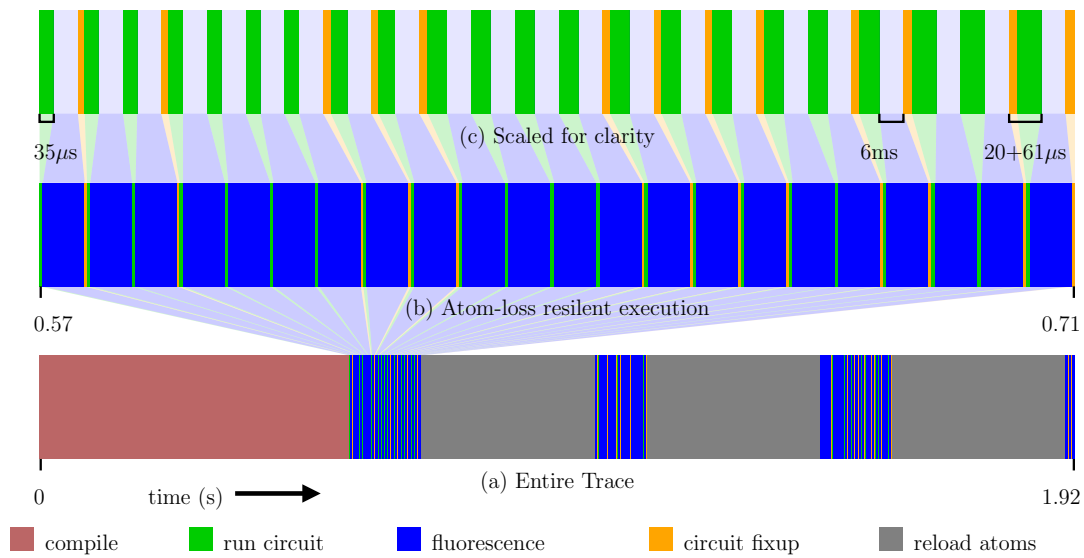


Figure 3.14: A timeline of 20 successful shots for *Compile Small and Reroute* with reload time of 0.3 s and fluorescing time of 6 ms. A majority of the overhead time is contributed by the reload time and fluorescence, indicating, that the duration and count of these actions is crucial to overall runtime.

CHAPTER 4

APPLICATION-GUIDED ARCHITECTURAL DESIGN.

VIRTUALIZING ERROR CORRECTED QUBITS

4.1 Introduction

Quantum devices have improved significantly in the last several years both in terms of physical error rates and number of usable qubits. For example, IBM and others have made accessible via the cloud several devices with 5 to 53 qubits with moderate error rates [92]. Concurrently, great progress has been made at the software level such as improved compilation procedures reducing required overhead for program execution. These efforts are directed at enabling NISQ (Noisy Intermediate-Scale Quantum) [155] algorithms to demonstrate the power of quantum computing. Machines in this era are expected to run some important programs and have recently been used to by Google to demonstrate “quantum supremacy”¹ [10].

Despite this, these machines will be too small for error correction and unable to run large-scale programs due to unreliable qubits. The ultimate goal is to construct fault-tolerant machines capable of executing thousands of gates and in the long-term to execute large-scale algorithms such as Shor’s [168] and Grover’s [78] with speedups over classical algorithms. There are a number of promising error correction schemes which have been proposed such as the color code [114] or the surface code [67, 90, 72]. The surface code is a particularly appealing candidate because of its low overhead, high error threshold, and its reliance on few nearest-neighbor interactions in a 2D array of qubits, a common feature of superconducting transmon qubit hardware. In fact, Google’s next milestone is to demonstrate error corrected qubits [10, 126].

Current architectures for both NISQ and fault-tolerant quantum computers make no

1. JMB’s contributions include the design of the naive and compact mapping of surface code to the 2.5D architecture, syndrome extraction procedure, and original work on the memory-equipped architectural design (see later). This work is equal contributions from both CD and JMB.

distinction between the memory and processing of quantum information (represented in qubits). While currently viable, as larger devices are built, the engineering challenges of scaling up to hundreds of qubits becomes readily apparent. For transmon technology used by Google, IBM, and Rigetti, some of these issues include fabrication consistency and crosstalk during parallel operations. Every qubit needs dedicated control wires and signal generators which fill the refrigerator the device runs in. To scale to the millions of qubits needed for useful fault-tolerant machines [72], we need to a memory-based architecture to decouple qubit-count from transmon-count.

In this work, we use a recently realized qubit memory technology which stores qubits in a superconducting cavity [142]. This technology, while new, is expected to become competitive with existing transmon devices. Stored in cavity, qubits have a significantly longer lifetime (coherence time) but must be loaded into a transmon for computation. Although the basic concept of a compute qubit and associated memory has been demonstrated experimentally, the contribution of our work is to design and evaluate a system-level organization of these components within the context of a novel surface code embedding and fault-tolerant quantum operations. We provide a proof of concept in the form of a practical use case motivating more complex experimental demonstrations of larger systems using this technology.

Our proposed 2.5D memory-based design is a typical 2D grid of transmons with memory added as shown in Figure 4.1. This can be compared with the traditional 2D error correction implementation in Figure 4.2, where the checkerboards represent error-corrected logical qubits. The logical qubits in this system are stored at unique virtual addresses in memory cavities when not in use. They are loaded to a physical address in the transmons and made accessible for computation on request and are periodically loaded to correct errors, similar to DRAM refresh. This design allows for more efficient operations such as the transversal CNOT between logical qubits sharing the same physical address i.e. co-located in the same cavities. This is not possible on the surface code in 2D which requires methods such as braiding or

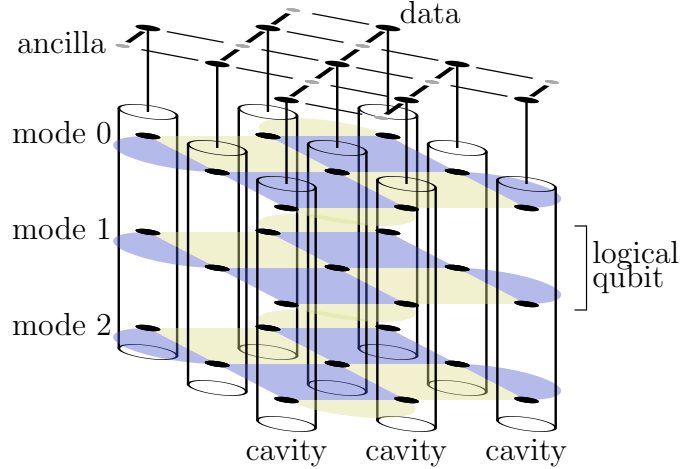


Figure 4.1: Our fault-tolerant architecture with random-access memory local to each transmon. On top is the typical 2D grid of transmon qubits. Attached below each data transmon is a resonant cavity storing error-prone data qubits (shown as black circles). This pattern is tiled in 2D to obtain a 2.5D array of logical qubits. Our key innovation here is storing the qubits that make up each logical qubit (shown as checkerboards) across many cavities to enable efficient computation.

lattice surgery for a CNOT operation.

We introduce two embeddings of the 2D surface code to this new architecture that spread logical qubits across many cavities. Despite serialization due to memory access, we are able to store and error-correct stacks of these logical qubits. Furthermore, we show surface code operations via lattice surgery can be used unchanged in this new architecture while also enabling a more efficient CNOT operation. Similarly, we are able to use standard and architecture-specific magic-state distillation protocols [123] in order to ensure universal computation. Magic-state distillation is a critical component of error-corrected algorithms so any improvement will directly speed up algorithms including Shor’s and Grover’s.

We discuss several important features of any proposed error correction code, such as the threshold error rate (below which the code is able to correct more errors than its execution causes), the code distance, and the number of physical qubits to encode a logical qubit. In many codes, the number of physical qubits can be quite large. We develop an embedding from the standard representation to this new architecture which reduces the required number

of physical transmon qubits by a factor of approximately k , the number of resonant modes per cavity. We also develop a Compact variant saving an additional 2x. This is significant because we can obtain a code distance $\sqrt{2k}$ times greater or use hardware with only $\frac{1}{2k}$ the required physical transmons for a given algorithm. In the near-to-intermediate term, when qubits are a highly constrained resource this will accelerate a path towards fault-tolerant computation. In fact, the smallest instance of Compact requires only 11 transmons and 9 cavities for k logical qubits.

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

At a high level, this chapter serves as a clear example of how applications, such as error correction codes the long term goal of quantum hardware, can guide the design and construction of architectures and help guide towards platforms which are best suited for it.

In summary, we discuss the following contributions:

- We introduce a 2.5D architecture where qubit-local memory is used for random access to error-corrected, logical qubits stored across different memories. This allows a simple virtual and physical address scheme. Error correction is performed continuously by loading each from memory.
- We give two efficient adaptations of the surface code in this architecture, Natural and Compact. Unlike a naive embedding, both support fast transversal CNOTs in addition to lattice surgery operations with improved connectivity between logical qubits.

- We develop an error correction implementation optimized for Compact and designed to maximise parallelism and minimize the spread of errors.
- Via simulation, we determine the surface code adapted to our 2.5D architecture is still an effective error correction code while greatly reducing hardware requirements.

4.2 Relevant Background

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

4.2.1 Superconducting Qubit Architectures

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

4.2.2 Qubit Memory Technology

Recently, studies have demonstrated random access memory for quantum information [142, 83]. Qubit states can be stored in the resonant modes of physical superconducting cavities attached to a transmon qubit as depicted in Figure 4.3. In these devices, transmon-transmon

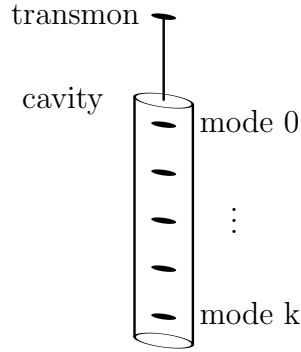


Figure 4.3: A close-up representation of the qubit memory technology we use. On top is a superconducting transmon qubit physically connected to a resonant superconducting cavity. This cavity has many resonant modes each used to store a qubit. These qubits can be loaded and stored (with random access) via the transmon.

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

4.2.3 Quantum Errors

Quantum systems are inherently noisy, subject to a variety of coherent and non-coherent error. For example, when attempting to apply some gate U to a qubit we may actually apply some other gate U' which is close to the desired operation but may include an additional undesired operation. Fortunately, this type of coherent error is fairly easy to model. Since every single-qubit unitary can be expressed as a linear combination of the Pauli matrices I, X, Y, Z we can express this coherent error as a combination of bit flip (X) and phase flip (Z) errors where I is no error and Y is simultaneous bit and phase errors ($Y = iXZ$). For a

quantum error correcting code this will play a part in digitizing errors, meaning we will be able to simply detect and correct X and Z errors.

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

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

4.2.4 Surface Codes, Error Decoding, and

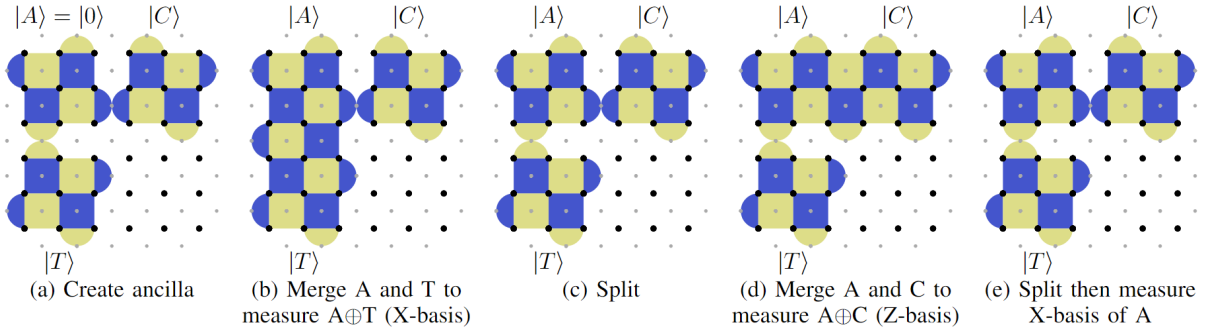


Figure 4.4: The lattice surgery operations to perform a logical CNOT on the standard surface code (and directly supported in our architecture). Given control and target qubits $|C\rangle$ and $|T\rangle$, a CNOT is performed by enabling and disabling the parity checks as shown across 6 timesteps ((e) is two steps). We show this complex process to contrast with the fast transversal CNOT enabled by our architecture (described later in Section 4.3.2).

Lattice Surgery

The surface code [67] is one of the most promising quantum error correction protocols because it requires only nearest neighbor connectivity between physical qubits. The surface code is implemented on a two-dimensional array of physical qubits. These qubits are either data, where the state of the logical qubit is stored, or ancilla used for syndrome extraction (parity checks). These ancilla qubits are measured to stabilize the entangled state of the data. These ancilla fall into two categories, measure-Z and measure-X for Z syndromes and X syndromes designed to detect bit and phase errors respectively. Data qubits not on the boundary are adjacent to two measure-Z and two measure-X qubits.

In Figure 4.2 we show four logical qubits with code distance 3 mapped to a 2D lattice of superconducting qubits. Dark physical qubits are used as data and light qubits are used as measure qubits. In this paper, we opt to explicitly indicate qubits in order to make clear how logical qubits, formed of many square and half-circle plaquettes, are mapped directly to hardware. In our diagrams however, we use customary notation by shading X-plaquettes blue (dark) and Z-plaquettes yellow (light). Half-plaquettes contain only 2 data qubits and are shown as half circles.

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

Errors are decoded by running a classical algorithm on the measured syndromes [68]. In the surface code, when an error occurs on a data qubit, for example a single X bit-flip error, we see this as a change in the measurement outcome of *both* of the Z-syndrome ancilla adjacent to it. If an error occurs on every data qubit in a chain of neighbors, only the two syndromes at the ends will detect a change. The standard way of performing error decoding is to collect all of these changed syndromes into a complete graph with edge weights given by the log-probability of that chain of errors occurring. We perform a maximum likelihood perfect matching of this graph to find the most probable set of error locations which we correct or track in the classical control. If errors are sufficiently low these error chains will be well isolated and this decoding algorithm will be able to determine the correct set of corrections to be made. If errors are less sparse, this matching algorithm may misidentify which error chains have actually occurred and this can result in a logical error, that is a *logical* bit flip or phase flip is applied. These logical errors cannot be detected because they result from misidentifying the physical errors.

There are two primary ways to manipulate the logical qubits of the surface code to perform desired logical operations—braiding and lattice surgery. In this paper we will primarily consider

lattice surgery which has been shown to have some advantages over braiding like using fewer physical qubits. For a more thorough introduction to lattice surgery we refer the reader to [90, 123, 116]. In our proposed scheme, all primitive lattice surgery operations can be used such as split and merge which together perform a logical CNOT as shown in Figure 4.4. For universal quantum computation in surface codes we allow for the creation and use of magic states such as $|T\rangle$ or $|CCZ\rangle$. These states are necessary because the T and CCZ operations cannot be done transversely (using physical gates on the data in parallel to reliably perform the logical gate) in this type of code. However, high fidelity versions of these states can be generated via distillation [29, 123] where many error-prone copies of the state are combined to generate the state with low error probability. Our scheme permits the use of these methods in the same way as other surface code schemes and also allows more efficient implementations.

4.3 Virtualized Logical Qubits

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

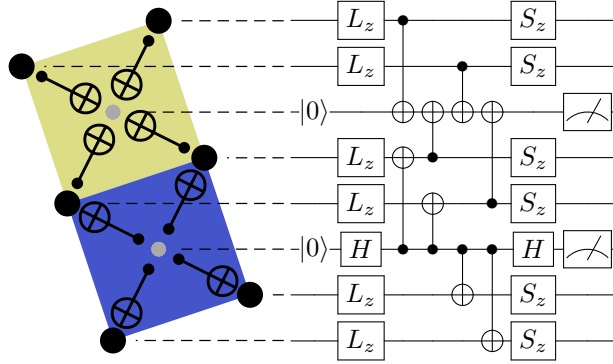


Figure 4.5: Circuit showing how to execute our Natural embedding on hardware. Left: The layout of eight data (black) and two ancilla (gray) in hardware. CNOT operations between qubits are drawn between. Right: A circuit diagram of the operations applied over time where each horizontal line corresponds to a qubit and each box or symbol is an operation. The steps are L_z : load from memory mode z , $|0\rangle$: reset ancilla, CNOTs: compute the Z or X parity, Meter: measure the result, S_z : store back to memory.

4.3.1 Natural Surface Code Embedding

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

Consider the high-level three dimensional view of the quantum memory architecture presented in [142]. For every transmon in this architecture (the compute qubits in the top layer of Figure 4.1) there is a cavity attached with a fixed number of resonant modes, k . Each cavity can store k qubits, one per mode. Each transmon can load and store qubits from its attached cavity by performing a transmon mediated iSWAP. We assume all transmons can be operated on in parallel as is the case in most superconducting hardware (i.e. from IBM or Google). For example, we can load qubit q_{iz} to transmon t_i and load q_{jz} to transmon t_j in parallel, simultaneously execute single qubit operations on each qubit, then store in parallel.

Any other qubits stored in cavities i or j will be unaffected by these operations. We expect this technology to allow cavity size k on the order of 10 to 100 qubits and it will likely not be practical to scale k along with the size of the 2D grid as hardware improves so we cannot implement a true 3D code such as [27]. For our analysis, we conservatively assume $k = 10$ and view this as a 2.5D architecture where we expect the width and height of the grid to scale while the depth, k , remains small.

We demonstrate how our system is sensitive to the length of these cavities in section 4.6 where the amount of time between error correction cycles is directly a function of this cavity size k . As the size of the cavity becomes very large, the physical qubits stored are expected to be subject to more and more decoherence errors which will reduce our ability to properly decode the errors.

Consider the rotated surface code of Figure 4.2 and the high level view of this architecture in Figure 4.1. We imagine mapping each of the physical qubits of this logical qubit $q_{L,1}$ to the same mode z of each cavity in this memory architecture. Another logical qubit $q_{L,2}$ can be mapped to mode $z_2 \neq z$ of the same set of cavities. We view this as stacking the surface code patches, the logical qubits, together under the same set of transmon qubits. The transmons themselves are only used for logical operations and error correction cycles performed on the patches.

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

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

Alternatively, we can Interleave the extraction cycles by loading the logical qubit in index 0, performing one syndrome extraction step, then storing. We execute this same procedure for every logical qubit in the stack and repeat d times. We expect this latter procedure to be less efficient, subjecting the data qubits to d load and store errors per d cycles as opposed to performing exactly one set of loads and stores when collecting all d measurements at once. We study the effect of this choice of syndrome extraction on the error threshold in Section 4.5. We detail these extraction protocols for each syndrome in Figure 4.5. Here we use L_z (S_z) to indicate loading (storing) the data from (to) index z of the attached cavity.

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

In our system, up to k logical qubits share the same set of transmons, more efficiently storing these qubits than on a single large surface. In order to interact logical qubits in different stacks we load them in parallel to the transmons then interact them via lattice

surgery operations like the CNOT shown in Figure 4.4. In these cases, all of the other stacks' transmons between the interacting logical qubits act as a single (possibly large) logical ancilla. In typical planar architectures, we are unable to execute transversal two-qubit operations due to limited connectivity. We can perform physical operations between qubits in the same cavity, mediated by the transmon. Therefore, in our system, we *are* able to perform transversal two-qubit interactions if the logical qubits are co-located in the same stack. We describe this next.

4.3.2 *Transversal CNOT*

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

Figure 4.6 demonstrates this for the transversal CNOT gate which we verified via process tomography [145, 146] to apply the expected CNOT unitary in simulation. This can be performed in a single round of d error correction cycles while the lattice surgery CNOT shown in Figures 4.4 (and later 4.9) takes 6 rounds. This can translate to major savings in runtime for algorithms.

The transversal CNOT is not limited to logical qubits currently stored in the same 2D address. With an extra step it is possible to transversely interact any two logical qubits.

To do this one of the qubits must be *moved* to the same 2D address as the other using a move operation described in [123]. The move operation involves growing the patch toward the move target in one step by adding new plaquettes along the entire path and performing d cycles, one timestep, of error correction. Once grown, the patch can be shrunk from the other end back to its original size. The data qubits freed during the shrink are measured and used to determine any fixup operation. Once the two qubits are in the same 2D address, the transversal CNOT can be applied. It can then be moved back, left where it is, or moved somewhere else as determined during compilation. This process takes 2 timesteps or 3 if including the second move.

4.3.3 Compact Surface Code Embedding

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

In the above memory architecture, because we do not store any logical qubits in the transmon layer, these qubits can act as the measurement ancilla, rather than have separate transmons only there to act as the syndrome measurement ancilla. With this observation, we can pack the data qubits of the surface code patch of Figure 4.7a more efficiently with *every* transmon having a cavity attached. Each plaquette of the rotated surface code has a single ancilla at its center, interacting with each data qubit. For Z plaquette (yellow or light) in this mapping scheme we colocate the upper-right data and the ancilla; the upper-right data is located in the cavity attached to the transmon corresponding to the ancilla. Similarly, for

each X plaquette (blue or dark) we colocate the lower-left data and the ancilla; the lower-left data is located in the cavity attached to the transmon corresponding to the ancilla.

This mapping results in plaquettes which resemble triangles rather than squares, where the center of the hypotenuse of each triangle corresponds to both the ancilla qubit and the data qubit, stored “beneath” in its cavity. Every data qubit is still mapped to the *same* index. Notice in this scheme every data (sans the boundary) is still adjacent to two measure- Z and two measure- X ancilla where adjacent means either in the cavity of the ancilla or in a cavity adjacent to the ancilla. We illustrate this transformation from our undistorted Natural surface code patch to Compact in Figure 4.7 and a diagram of this architecture with a cavity for every transmon in Figure 4.8. If a different ancilla location were chosen, for example all sharing with the upper-right data, some of the syndrome extraction gates in the resulting arrangement would require six-way connectivity, two diagonal to the grid, which would be much more difficult to engineer with low noise. This scheme where X and Z ancilla share with data in opposite directions is the best scheme we found to satisfy the hardware connectivity.

In Natural, we assign square patches to predetermined square patches on the hardware. In Compact, we assign square patches to predetermined rhombus or diamond patches on the hardware. Previously, operations on the virtualized patches closely resembled the original operations because the shape was unchanged, except with the addition of loads and stores to retrieve the logical qubit from memory. The same operations apply here. We can examine the original, unmapped surface code patch and perform the same sequence of operations modulo loads and stores, on the transformed coordinates of the mapped version.

This new mapping also requires a new syndrome extraction procedure because data cannot be loaded while a transmon is in use as an ancilla. A single round of syndrome extraction can be executed by dividing the plaquettes into four groups, with each group containing non-interfering plaquettes. Two plaquettes are non-interfering if they do not share their ancilla with any data qubits of the other plaquette. This process is detailed explicitly in

Figure 4.10. It is imperative this process use both the minimum number of loads and stores and keep data qubits loaded for as short a time as possible as the error incurred during this circuit directly impacts the error threshold for the code. This has a similar cost as Natural, Interleaved where a higher numbers of load and store gates were also required.

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

4.3.4 *Architectural Considerations*

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

move the logical qubit through.

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

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

Finally, many lattice surgery operations require the use of ancilla logical qubits, for example to measure specific stabilizers which are done to execute a particular set of operations in [123]. This restriction requires our architecture and any compiler to guarantee one free mode of every stack be allocated to temporarily obtain large logical qubits. This free mode may be shared with qubit movement or separate if many ancilla logical qubits are used.

4.4 Evaluation

In this section, we outline our error model and experimental setup used to determine error thresholds for our mapping and syndrome extraction schemes. We compare to the surface code on a typical 2D architecture. Our goal is to demonstrate the error thresholds for various error correction schemes, i.e. to determine the necessary *physical* error rate required to begin obtaining exponentially better *logical* error rate as the code distance increases. Currently, neither transmon devices nor transmon-memory devices used for our schemes have consistently achieved physical error rates below this threshold and instead the threshold serves as a goal or checkpoint.

4.4.1 Error Model and Noise Assumptions

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

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

error occurs with probability p then we apply an X , Y , or Z with probability $p/3$ and I (no error) with probability $1 - p$.

- We detect and correct X and Z errors independently. A Y error is both an X and Z error.

For each of our experiments we rely on realistic device data for current superconducting devices, provided by IBM [92]. For the memory hardware, we use experimental data from [142]. These parameters are listed in Table 4.1, where $T_{1,c}$ is the coherence time of the cavity, $T_{1,t}$ is the coherence time of the transmon, Δ_t is the single qubit gate time, Δ_{t-t} is the two-qubit transmon-transmon gate time, Δ_{t-m} is the two-qubit gate time of transmon-mode interactions, and $\Delta_{l/s}$ is the load and store times. In every experiment, the gate durations for one- and two-qubit interactions is fixed. In a first set of experiments, we vary all gate errors and coherence times together, all derived from a single probability of error p given as the probability of an SC-SC (Transmon-Transmon gates) two-qubit gate error. We consider T_1 times of both cavities and transmons to determine the probability of storage error given as $\lambda = 1 - \exp\{-\Delta t/T_1\}$, where Δt is the duration stored. We consider the same potential gate error rates for each of these devices since the underlying technology behaves very similarly. While typical coherence errors are not generally Pauli, we model them as Pauli errors here as a worst-case approximation since correcting Pauli errors is harder than correcting coherence errors in general.

4.4.2 *Experimental Setup*

In every experiment, we run 2,000,000 simulated trials per data point with each trial consisting of a round of error correction. We compute the logical error rate as the number of logical errors (misidentified error chains) over the total number of trials. The large number of trials is required to estimate logical error rates to 10^{-5} . To determine the error threshold values for different surface code schemes, we vary the physical error rate over several different code

Table 4.1: Starting point coherence times and constant gate times for the hardware models.

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

sizes. The goal is to find an intersection point for each of these lines which gives a physical error rate below which we expect our logical error rate to get better as the physical error rate improves. Below the threshold we also expect the logical error rate to get better exponentially in the code distance d .

We study 5 setups to determine initial error thresholds.

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

In our designs, the possible sources of error are more nuanced and we study the thresholds' sensitivity to variation in the parameters. In all threshold experiments, we assume cavity depth of 10 but later study sensitivity to cavity size. The simulation code used to generate our results is available on GitHub [60].

4.5 Error Threshold Results

We detail our threshold results in Figure 4.11. We study 5 different code distances in order to obtain the physical error threshold value. The threshold value indicates at which point increasing the code distance, d , improves the logical error rate instead of hurting it. This threshold is a function of both the physical system model, the chosen syndrome extraction

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

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

4.6 Error Sensitivity Results

In this section, we study the effects of different sources of error on the thresholds obtained in section 4.5. Specifically, we show how different system-level details affect the threshold of the code. Here we focus on Compact, Interleaved as the most efficient physical qubit mapping and subject to a wide variety of errors. In these studies, the physical error rates of all but

a single error source are fixed at a typical operating point below the threshold obtained previously, 2×10^{-3} and the cavity depth is fixed at 10. Gate times are fixed while we vary the physical error rate of SC-SC gates, SC-Cavity gates, Load-Store gates or the coherence times of the cavity and the transmon. We additionally study the duration of load/store, the gates unique to memory technology. We note the effect of the SC-Cavity gate duration will be a similar, smaller effect since it occurs only once per qubit per error correction cycle. Finally, we study the effect of cavity size by varying the number of modes per cavity, causing a proportional delay between error correction cycles.

The results of these sensitivity studies are found in Figure 4.12. The logical error rate is sensitive to a particular error source's probability if the slope of the line is pronounced at the marked reference value. The logical error rate for Compact, Interleaved is sensitive to all changes in system-level details to some degree. The gate error rates show the highest sensitivity, indicating improvement in these will give the greatest benefit. Coherence times are not quite as sensitive but the slightly over 10x offset between the cavity and transmon plots shows that there is no benefit in transmon T_1 being longer than 1/10 cavity T_1 when the cavity size is 10. The lines taper off, indicating other errors sources eventually dominate. Initially, we expected the cavity size to have a large impact on the logical error rate. However, when coherence times are high and gate error rates are fairly low below the threshold, the logical error rate does increase proportional to the length of the cavity but the effect is very minor. This indicates, given cavities with good coherence times, our proposed system will be able to scale smoothly into the future as cavity sizes increase.

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

4.7 Magic State Distillation

Resource Estimates

Now that we have shown error correction is effective in our virtualized qubit architecture, we analyze how the transversal CNOT and memory connectivity can benefit the performance of an algorithm overall. In error-corrected quantum algorithms, the dominating cost (commonly $> 90\%$) in both space and time resources is magic state distillation [177, 56, 72]. For this analysis we consider how T-state distillation, a commonly used magic state, is improved. Any improvements here will translate directly to improvements in important algorithms like Shor's and Grover's.

We take the 15-to-1 distillation circuit of [29] to generate a T magic state using a single patch of transmons with 6 logical qubits stored in the attached cavities. This circuit consists of 16 qubit initializations, 15 measurements, 35 CNOT gates and a few other operations. It takes a total of 110 surface code timesteps to generate a T-state using only a single patch of transmons. If pairs of these circuits are executed in lock-step, they only take 99 timesteps.

In Figure 4.13 we compare the T-state generation rate with memory against two representative extremes designed for

Table 4.2: Transmon, depth-10 cavity, and total qubit costs of each T-state generation protocol for $d = 5$.

Protocol	# transmons	# cavities	total qubits
Fast Lattice [124]	1499	-	1499
Small Lattice [123]	549	-	549
VQubits (natural)	49	25	299
VQubits (compact)	29	25	279

speed or size, Fast Lattice [124] and Small Lattice [123] (also based on [29]). Fast Lattice generates a T-state every 6 timesteps but uses 30 patches of space whereas Small Lattice, generates a T-state every 11 timesteps using only 11 patches of space. We compare these results by computing the T-state generation rate per timestep if we filled 100 patches with copies of the circuit running in parallel. Table 4.2 show the qubit cost of each and chip area will be proportional to the number of transmons. Using our VQubits protocol generates 1.82x as many T-states as Fast Lattice and 1.22x as many as Small Lattice. This improvement allows an algorithm like Shor’s to run roughly 1.22x faster or work on smaller hardware.

4.8 Conclusion

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

By taking advantage of recent advances in quantum memory technology, we present a new architecture to substantially reduce hardware requirements by storing logical qubits distributed in memory. This technology allows memory to be separated but local to computation in a quantum system. We provide two direct mappings of the surface code to this new system with virtual addressing and illustrate how syndrome extraction and error correction procedures

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

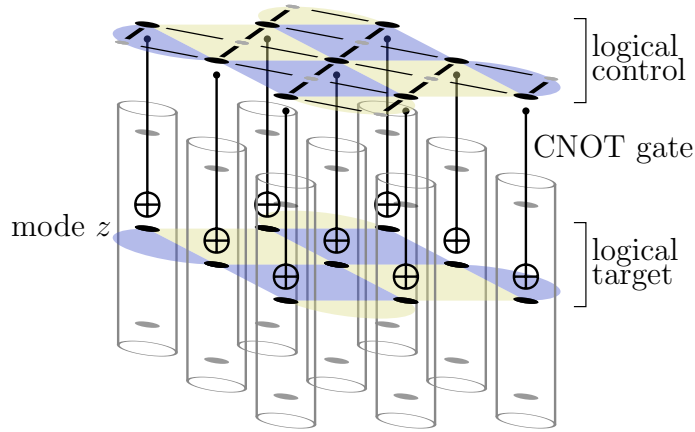


Figure 4.6: The transversal CNOT enabled by our 2.5D architecture. The data qubits for the control logical qubit are loaded into the transmons. Transmon-mediated CNOTs to mode z for every data qubit perform the logical operation. This takes one timestep to perform, 6x better than a lattice surgery CNOT.

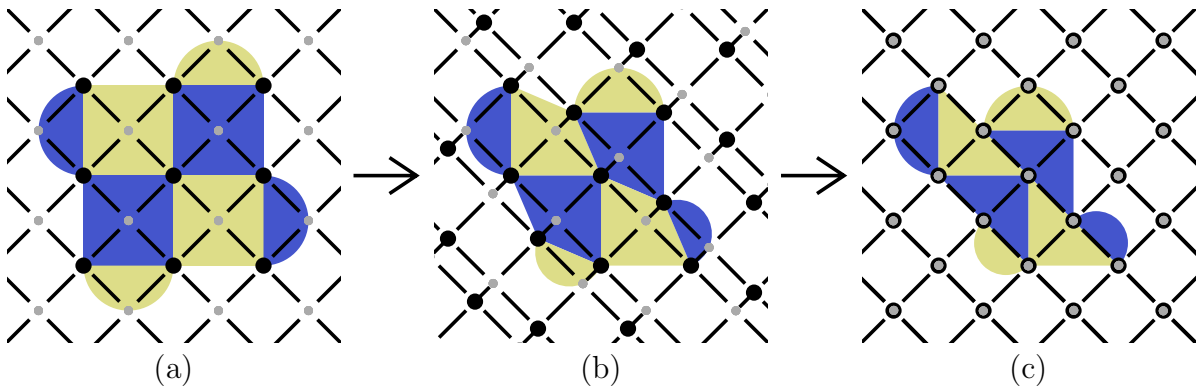


Figure 4.7: Transformation from Natural to Compact. (a) Natural embedding: Only data have attached cavities (not shown). (b) The transformation: Z ancilla (over yellow/light areas) merge with the upper-right data transmon and X ancilla (over blue/dark areas) merge with the lower-left data transmon. The opposite pairings are key to keeping 4-way grid connectivity. (c) Compact embedding: All ancilla transmons without attached cavities have been removed. All remaining transmons have cavities and are used as both data and ancilla.

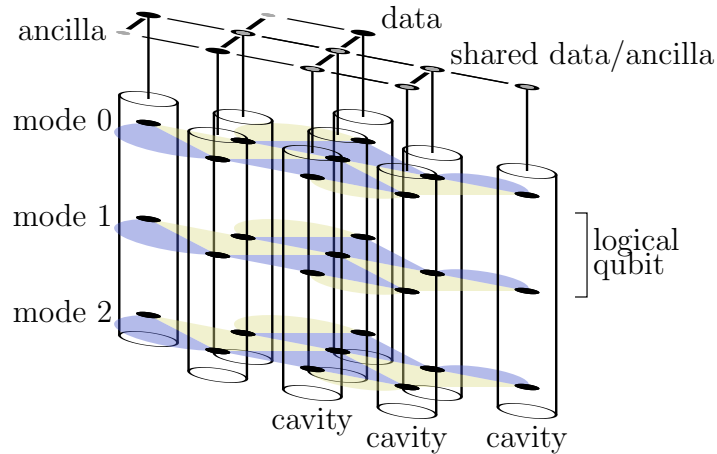


Figure 4.8: A 3D view of our Compact embedding. Shown at the top is the 2D grid of transmon qubits. Attached below every transmon is a resonant cavity. Compact surface code patches are shown stored, one in each mode. This deformed patch can be tiled in 2D.

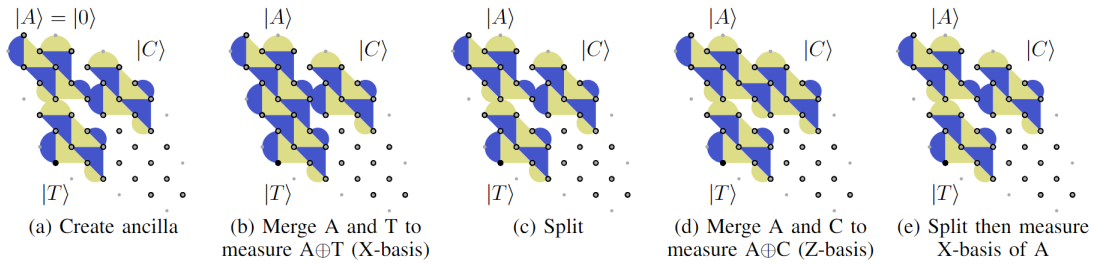


Figure 4.9: The Compact lattice surgery operations to perform a CNOT. The logical operations performed are identical to Figure 4.4 but the corresponding physical operations are arranged as shown in Figure 4.7. This uses half as many transmons as Natural. As before, it takes 6 timesteps of d error correction cycles each.

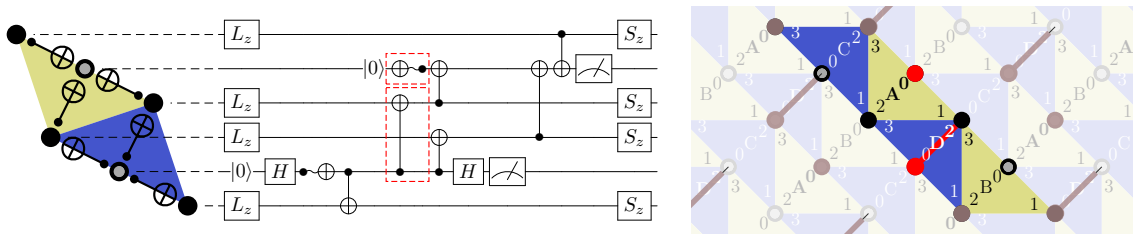


Figure 4.10: The CNOT sequence for parity checks in Compact. Left: A quantum circuit showing the hardware operations over time. Right: The CNOT execution order repeats $A_0D_2, A_1D_3, A_2C_0, A_3C_1, B_0C_2, B_1C_3, B_2D_0, B_3D_1$. The AB and CD sequences run in parallel but offset to ensure ancilla and data use do not conflict. CNOTs for A_0D_2 are marked in red where an isolated circle indicates a transmon-mediated CNOT.

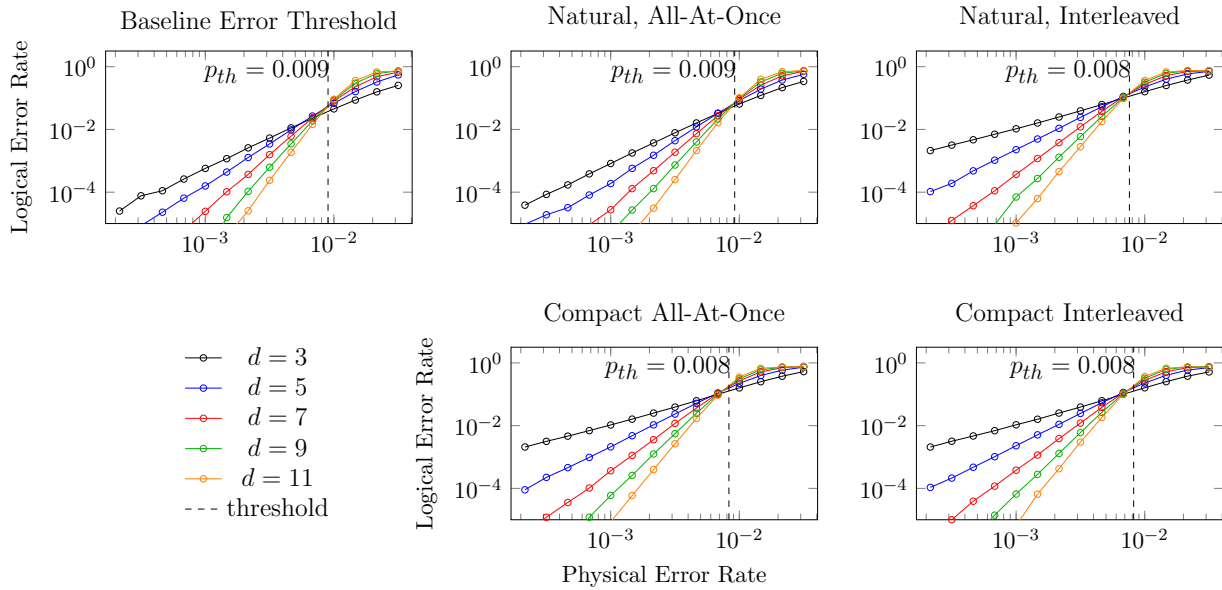


Figure 4.11: Error thresholds for the baseline 2D architecture and Natural and Compact variants of our 2.5D architecture. The thresholds are comparable to the baseline indicating the space savings obtained in our system does not substantially reduce the error thresholds. The slopes of the lines in this figure indicate, post-threshold, how much improvement in physical error rates improve logical error rate. Except for the baseline, all use a cavity size of 10.

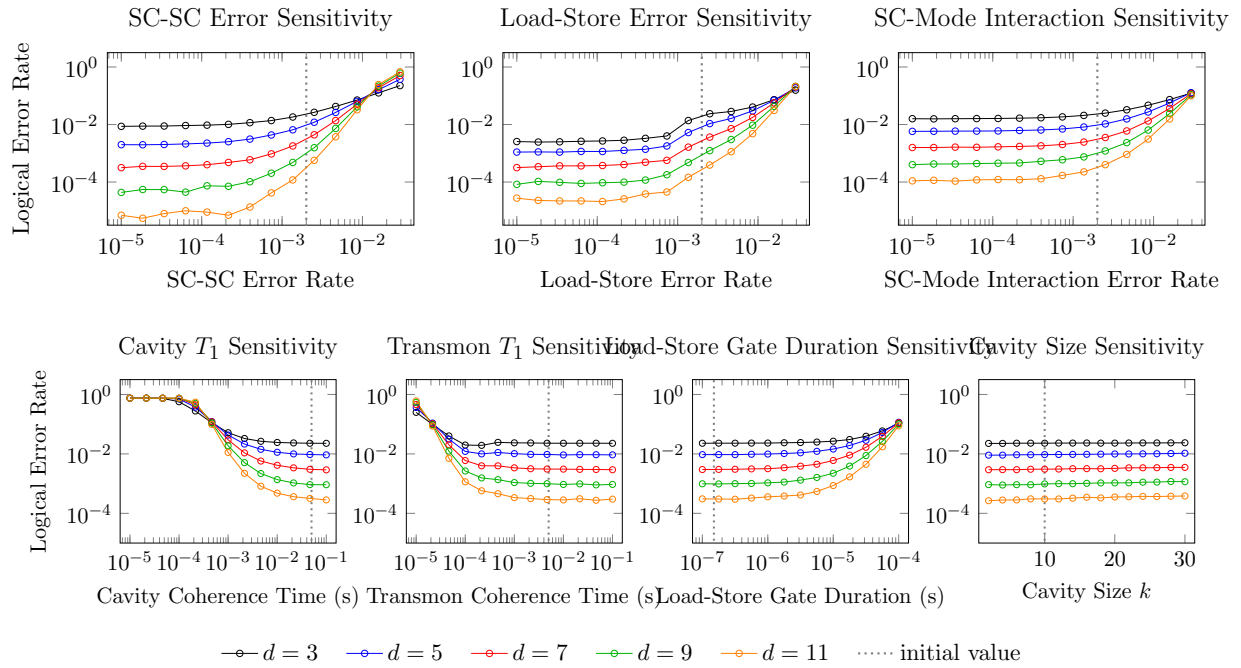


Figure 4.12: Sensitivity of logical error rate to various error sources in Compact, Interleaved. The logical error rates are most sensitive to physical error of Loads/Stores and SC-SC gates. The logical error rate is less sensitive to the coherence times and mostly insensitive to effects of load-store duration and cavity size.

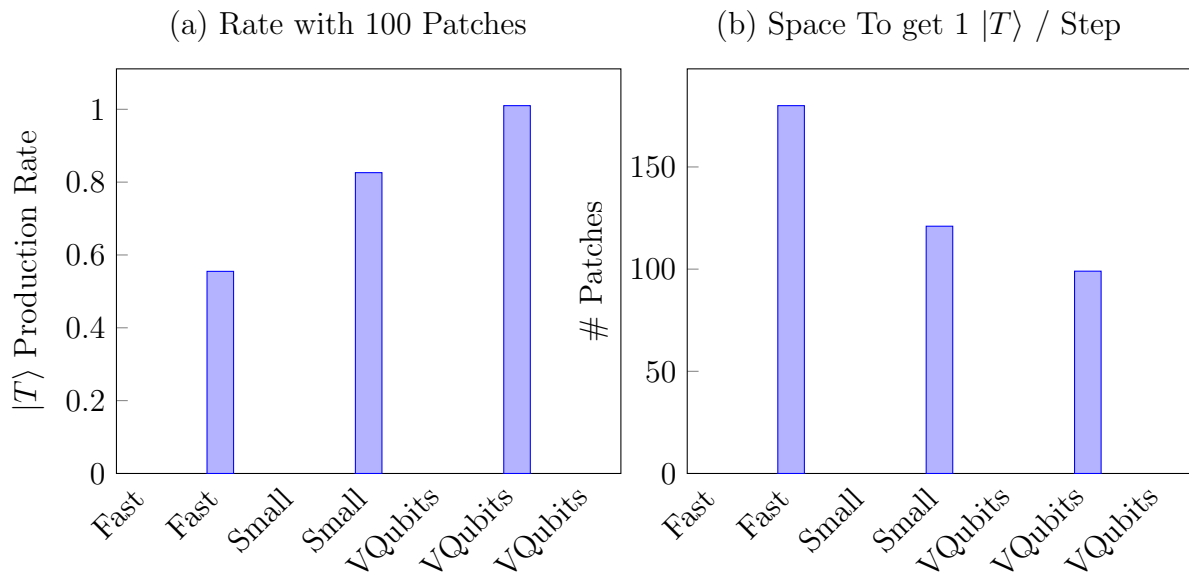


Figure 4.13: (a) The T-state generation rates of three different circuits. Higher generation rate is better. (b) The space, in terms of number of patches, required to produce a single $|T\rangle$ per time step. Lower is better. Fast [124] and Small [123] work in the surface code and do not use memory. VQubits is implemented with transversal CNOTs in our 2.5D architecture. All are based on [29].

CHAPTER 5

EVALUATING ARCHITECTURES AT THEIR LIMITS: IMPROVED COMPILATION METHODS

In the previous chapters, we focused primarily on the evaluation of potential quantum technologies, however, as we saw in the neutral atom case study, it was vital to adapt compilation methods - how we transform an input program from the user into an executable which can be run on directly on hardware - which optimized directly the constraints of the underlying machine. Different technologies come with their own set of benefits and pitfalls which must be accounted for in order to fairly evaluate and compare different implementations of universal gate-based quantum computers. In the neutral atom case, for example, if we use a generic compiler which decomposes every operation into 1 and 2 qubit gates initially then we may miss opportunities to execute complex multiqubit instructions natively using fewer communication operations. On one hand, it is valuable to adapt compilation methods to meet the device specific constraints and in this vein we explore two such examples: compilation to memory-equipped quantum technologies and graph partitioning for distributed, clustered quantum hardware.

While many compilation optimizations can be hardware specific, it is also valuable to consider more generic compilation methods, for example developing mapping and routing strategies which directly account for device errors which vary spatially and temporally. In general, quantum compilation passes rely on heuristics to obtain approximate solutions with optimality infeasible. In some passes, such as in operation routing, this results in using only partial, local information about circuit structure while neglecting more global structures. In the case where operations are decomposed into 1 and 2 qubit gates, using only local information can result in poor decisions about how to move qubits around the device. An alternative is to hierarchically route operations, in the simplest example by routing qubits participating Toffoli gates at once, rather than individually routing pairs.

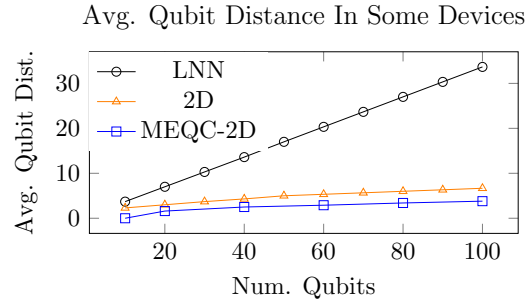


Figure 5.1: Average qubit distance on several instances of near-term target architectures. Average qubit distance approximates how many SWAPs are necessary to interact an arbitrary pair of qubits. LNN architectures scale extremely poorly in this metric resulting in a large number of added gates and depth. 2D and MEQC architectures scale much better. We show this translates into reduced number of gates and reduced depth as we scale into the future.

In either case - hardware specific or agnostic - developing better optimization tools is invaluable for the development of quantum hardware. In the context of this thesis, relying on generic tools developed with one particular backend hardware in mind results in unfair comparison between competing underlying technologies. In this chapter we discuss four examples of compiler optimizations and respective architectures when appropriate: the development of a compiler for memory-equipped quantum architectures which relies on new cavity technology to store multiple qubits worth of information in the same physical location, a compiler for distributed but clustered quantum hardware, noise-aware mapping and routing, and a first step in hierarchical routing.

5.1 Memory-Equipped Quantum Architectures: The Power of Random Access

There are several competing technologies for the underlying implementation of qubits such as trapped ions or superconducting circuits. Each of these technologies present unique challenges to scalability, at least without error correction. For example, in superconducting technology with limited connectivity between hardware qubits, numerous additional operations called SWAPs must be added to an input program in order to execute it. For example, in Linear-

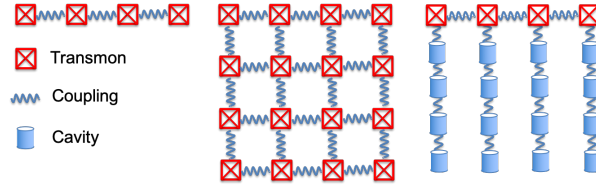


Figure 5.2: Both current and the proposed MEQC device. On the left is a LNN device where adjacent superconducting transmons are coupled enabling two qubit interactions. In the center is a 2D mesh architecture common among current manufacturers. On the right is the proposed MEQC architecture with transmons arranged in a line. Each transmon has an attached cavity which stores in memory multiple qubits. To operate on the qubits, they must first be loaded into the transmons.

Nearest-Neighbor and a 2D grid topologies on the order of thousands of additional operations must be inserted for execution, many of which are derived from increased average distance between qubits, as shown in Figure 5.1. With so many additional operations, the input programs are almost guaranteed to fail. These added operations dramatically increase the execution time of these programs, moving far beyond the coherence limit of current qubits (approximately the lifetime of qubits)¹.

Past efforts have focused primarily on the compilation problem to these small, near-term devices, such as variation aware mappings or SWAP reductions [135, 188, 120, 187]. These techniques have improved the ability of programs to succeed on currently available devices, but do not provide a path to scalability. There are inherent limits in current quantum architectures, such as gate error rates or qubit coherence times. Even with improved error rates or longer qubit lifetimes, the large gate and depth overheads induced by limited connectivity are too cumbersome.

We instead propose a new architecture, Memory-Equipped Quantum Computing (MEQC), the same underlying technology utilized in the prior VLQ work. We use superconducting qubits equipped with resonator cavities which can store qubits in their modes. Recently, small, physical prototypes of this have been realized and experimented with; however, the

1. The work in this section is entirely contributed by JMB with advisorial support from FTC and DS.

architectural implications of this new technology are almost entirely unexplored. For example, in the original experiments, the proposed benefit of this technology was to store less frequently used qubits in long coherent memory. Qubits stored in the modes of the attached cavity are expected to have an order of magnitude longer coherence times than typical transmon coherence times. This directly will reduce the frequency of idle errors on qubits which are unused. Idle errors are due inability to isolate qubits perfectly from the environment while still being able to manipulate them. In other superconducting devices, all qubits are roughly equally subject to these errors. In the proposed architecture, stored qubits are more isolated, resulting in protection from these idle errors.

This benefit is secondary to another hugely important feature for near- and intermediate-term and that is connectivity. In the proposed architecture, the transmon-cavity technology enables a gate to be applied, via transmon mediation, to any pair of qubits stored in the same cavity. This random access to stored qubits greatly improves local connectivity between qubits in these devices. This translates into significant reduction in compilation overhead, reducing the need for large numbers of SWAPs, but only if the compilation procedure properly accounts for these well connected regions. While experimentalists anticipated the coherence times of cavities to be the critical advantage, we find in our proposed MEQC architecture the primary benefit is this random access. For our proposed architecture, we provide a complete compilation framework transforming input quantum circuits to ones executable in hardware utilizing transmon-local memory. Our framework explicitly maximizes the advantages of both of these features to minimize compilation overhead.

Neither of these gains come for free. In the proposed architecture, in order to execute a gate we must first load the qubit from memory into a transmon which can be manipulated. Therefore, operations on this architecture require on average two additional operations in the form of Loads and Stores. For small programs, this architecture will require more operations than others. Furthermore, since there is only a single operational transmon per cavity, this

prevents gates from being executed in parallel on qubits located in the same cavity. The gains of this new architecture in terms of scalability outweigh these downsides, specifically by reducing the number of total operations required to execute input programs.

In the remainder of this section we perform the following:

- We introduce a new scalable quantum architecture, MEQC, which takes advantage of recent quantum memory-like hardware increasing local qubit connectivity and protects infrequently used qubits from idle errors.
- We develop a full compilation framework for MEQC devices which includes heuristics to maximize time spent in long coherent memory when unused and heuristics for mapping and routing of qubits during execution which minimizes the total number of SWAP operations inserted.
- We demonstrate our system reduces required gate and depth overhead substantially over other competitive options and subsequently increases the chance to succeed.
- We analyze different architecture-specific design choices such as number of modes per cavity, and top-level transmon connectivity. Furthermore, our system is able tolerate up to 12x worse transmon-transmon interconnect error which is expected to be the dominant source of error in systems utilizing transmon-cavity technology in near-term. We conclude our architecture presents a path towards scalability in the near and intermediate term.

5.1.1 Relevant Background

Near-Term Quantum Architectures

In the last several years, a number of competing technologies have emerged such as superconducting qubits and trapped ions. These have been physically realized in systems containing

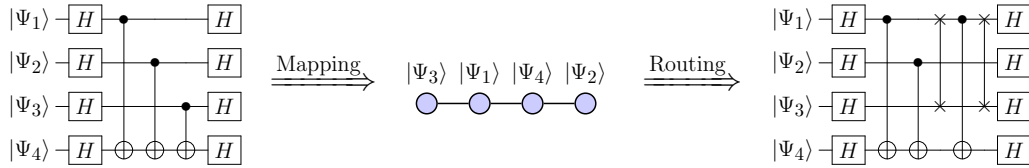


Figure 5.3: Compiling to near-term devices is a multi-step process. First we map the logical, circuit qubits to the physical hardware qubits. Based on this placement and the input program, we insert SWAPs in order to interact distant qubits. Here we compile a simple quantum program, Bernstein-Vazirani, to a 4 qubit LNN architecture. Quantum programs, like the input program on the left are a sequence of gates specified on qubits. In this example, based on the given mapping, a pair of SWAPs are required to execute a CNOT between $|\Psi_3\rangle$ and $|\Psi_4\rangle$.

on the order of 10s of qubits [3, 4, 171]. These devices have limited connectivity with qubit connections specified by an underlying topology. We represent this connectivity as a graph with nodes corresponding to physical qubits and edges indicating valid two-qubit interactions. Some near-term topologies are found in Figure 5.2. For superconducting technology, common underlying topologies are Linear Nearest Neighbor (LNN) and 2D Mesh connectivity. We use these two styles of architectures as baselines to compare the proposed memory-equipped superconducting architecture (MEQC).

Compilation to Quantum Architectures

In the near-term, quantum programs are typically specified as quantum circuits. In order to execute this program, the circuit is decomposed into only single and two qubit gates. Then, the circuit level qubits are mapped to the physical qubits of the device. Because movement operations are expensive, qubits which interact often in the circuit should be placed in close proximity. For each operation specified in the circuit, if the qubits are already adjacent, nothing additional needs to be done. If they are not adjacent, a sequence of SWAP operations are inserted into the circuit to as to make the qubits adjacent on hardware. This process is known as *routing*.

Due to error, it is critical to minimize the number of added operations to circuits, in this

case these SWAP operations. Special care should be taken to map qubits to hardware in an effective manner. It is imperative to compile input programs in a way which requires the least amount of additional overhead in terms of number of gates and in added depth (roughly the length of the critical path from start to end of a program).

Errors on Quantum Devices

Quantum devices in the NISQ era are subjected to fairly significant error rates, somewhere on the order of 1 per 100-1000 operations [186, 92]. As such, the output of a quantum computation may be erroneous and it is common for a program to be run thousands of times to collect a distribution of outputs. In an ideal case, the correct answer appears with substantially higher probability than all of the other wrong answers.

There are a variety of different types of errors which can occur during the execution of a program [147]. The first are coherent errors such as bit-flip or phase-flip errors. These typically occur due to an error in the application of the gate to the qubits. The one and two qubit error rates of the device approximate how often this type of error occurs. These gates are easier to model and we expect this class of errors to be dominant in the near-term. Another type of error is decoherence errors, such as amplitude damping. This type of error is due to interaction with the environment; physical qubits are often kept isolated from the environment to avoid these types of decoherence errors, however, perfect isolation cannot be achieved because in order to do something useful we need to manipulate the qubits. The T1 times, or coherence times, of qubits and the amount of time it takes to execute a gate approximate how frequently these types of errors will occur. A good quantum device has long coherence times and low error rates. Finally, one other common error is crosstalk which occurs when multiple gates (usually two qubit gates) occur in parallel on adjacent sets of qubits. Unfortunately, crosstalk error is hard to approximate in larger systems, but it is suspected crosstalk in our proposed system is no worse than crosstalk in current superconducting

systems.

We are interested in designing a new type of scalable quantum architecture, specifically, one with long coherent memory attached to computational transmon qubits. As such we are unable to execute our benchmarks on realizations of these larger-scale devices. Instead, we model error in this and other competing architecture and use simulation results to determine performance. We use the Kraus operator formalism and density matrix simulation allowing us to inject noise channels into an executable circuit specific for a target architecture and approximate how well, here by measuring fidelity, the circuit performs compared to the ideal no noise version.

Simulating quantum systems is hard, requiring a large amount (exponential) of memory [82], and we are only able to do this for small benchmarks. An alternative approach is to use the probabilities of errors and for every operation in the circuit randomly draw if an error occurs or not. A successful program is one in which no errors are drawn, that is we assume any error causes the program to fail. In practice, this isn't always the case but this method provides a simple way to underestimate the probability of success (overestimate the probability of failure) for larger circuits on the order to 50 to 100 qubits. We use this worst-case estimation for programs with many qubits which even with the largest supercomputers would be unable to simulate.

5.1.2 A Memory-Equipped Quantum Architecture

In this section, we will present the proposed architecture termed “Memory-Equipped Quantum Computing” due to long coherent resonator cavities which resembles memory attached to each of the transmon, computational qubits. Recently, some key hardware components have been physically realized [142]. These initial experiments are significant but limited in scope and their architectural implications remain unexplored. We present an architecture which takes advantage of these memory-like components. These components have several proposed

benefits but also have technology-dependent constraints.

Our proposed architecture, while focused on superconducting qubit technology can be extended readily to other technologies such as ion trap devices. As we will see, our cavity model well approximates the operation of an ion trap device, specifically in connectivity. Furthermore, inter-trap communication technology is being demonstrated and is analogous to the transmon-transmon interactions presented here [25], wallraff2018deterministic. As such, our algorithms extend to these other technologies, though would require some modification regarding Loads, Stores, etc. which are technology specific.

We focus on the specific technology of [141] which realizes a multi-mode quantum “memory,” but in general we naturally support other hardware components which are memory-like. For multi-mode quantum memory, at a high level qubits stored in a mode of a resonator cavity which have long coherence times (T_1), about an order of magnitude longer than traditional superconducting qubits. When qubits are unused or idle they can be well isolated from the environment in these cavities, reducing the frequency of decoherence errors occurring. We consider these cavities as a sort of attached quantum memory to the actual computational qubits, here realized as transmons. These transmons in theory can be strung together or connected to each other in configurations similar to other superconducting technology.

The qubits stored in memory cannot be operated on except by special operations called Load and Store (transmon-mode iSWAPs) which essentially transfer the qubit stored in memory to the parent transmon. Therefore, in order to perform a single qubit operation on a qubit Q_1 it must first be loaded into a transmon, the operation then is enacted, and then Q_1 is replaced back into memory. In order to perform any action on the stored qubits, an additional two gates must be inserted. However, the modes of the resonator can be accessed like random access memory meaning this Load and Store pair is all that is needed. Besides improved resistance to decoherence errors, our architecture has the added benefit of full connectivity between qubits within the same cavity. This means we can interact pairs of

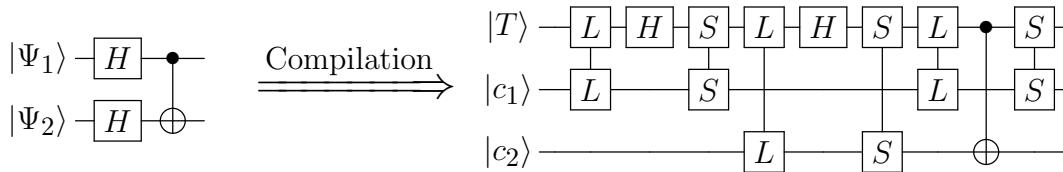


Figure 5.4: Compiling a small program to a MEQC device. In this case we map the input qubits $|\Psi_1\rangle$ and $|\Psi_2\rangle$ to one of the two available cavity modes. When executing the gates, we first execute a Load to move the qubit to the connected transmon. The gate is then executed and the qubit is returned to its original mode via a Store. We represent Loads as $L - L$ and Stores as $S - S$.

qubits in the same cavity without needing to add a large number of SWAPs as we would typically. These transmon-resonator pairs are strung together as in Figure 5.2.

This does not mean SWAPs are unnecessary altogether. SWAPs are still required in order to interact qubits in different cavities. To interact pairs of qubits located far away, we first load both qubits into the top level transmons. Then we can swap them through a path of connecting transmons at the top level. Once they are co-located in the same cavity, we can perform a two qubit interaction as usual. This architecture reduces the average distance between any pair of qubits meaning reduction in SWAP overhead. This of course comes at the cost of having a required a Load and Store for every operation.

As an example, consider an input program as in Figure 5.4. On other near-term devices, this program could be executed as is, with no additional operations (depending on the chosen gate set the CNOT may need to be converted). On this device, we require no SWAPs but several sets of Loads and Stores.

One other notable difference is the scale of the proposed devices. Our architecture would require many cavities, which occupy a much larger volume (though still small) than transmons. This leads to the need for unique ways of connecting the transmon qubits which may result in worse interconnect error rates. We explore our sensitivity to this later.

MEQC Compilation: Input Program to Executable

We now present our compilation framework which takes as input a program specified as a quantum circuit and outputs a new circuit executable on our architecture, specifically maps program qubits to modes of the hardware cavities and inserts the requisite Loads and Stores as well as any SWAPs that may be necessary to move qubits between cavities. For all of the steps presented below we assume the input circuit has been decomposed into a suitable gate set (the set of gates executable on the device) consisting of only one and two qubit interactions.

From this decomposed circuit, we may produce the *interaction graph* of the circuit, where nodes in this graph correspond to the circuit qubits and the edges are weighted by the number of interactions between the pairs of qubits. Specifically, if the nodes of the interaction graph are q_1, \dots, q_n then $w(q_i, q_j)$ is the number of interactions between q_i and q_j in the input program.

Initial Qubit Placement via Graph Partitioning

The first step in the compilation process is to generate a mapping from circuit level qubits to hardware qubits. In this case, we want to map the qubits to modes of the available resonator cavities. In order to avoid needing excessive SWAPs, we want to map qubits which frequently interact to the same cavity. Because there is essentially full connectivity between qubits stored in the same cavity, by placing frequently interacting qubits together we circumvent the need for SWAPs.

Because each cavity is represented as a fully connected graph in the underlying topology, we can perform initial placement by partitioning the interaction graph such that we minimize the number of edge crossings between partitions. Specifically, if we have k many cavities with exactly p resonant modes, we want to partition the interaction graph into k clusters with maximum cluster size of p . We do not require a minimum size; if there are disconnected

components it will usually be worse to force clusters to be full. We allow for clusters to be non-full by adding in dummy nodes which have weight zero to every real qubit. This problem is well-studied, and we adapt a version of a heuristic Overall Extreme Exchange to do this initial partitioning [106].

Alone, this is insufficient. We want to map the clusters to the physical cavities. If these cavities were fully connected (i.e. every transmon was connected to every other transmon), then every assignment of clusters to cavities is the same, that is in order to interact a pair of qubits located in different cavities we would need exactly one SWAP. However, this is not in general the case and the transmon connectivity may be much less well connected and therefore if *clusters* which interact frequently are distant we may need many more SWAPs. To avoid this, we want to locate these cavities nearby in terms of the device topology.

To handle this, we first choose any mapping of clusters to cavities. Let P_1, \dots, P_n be the partitions produced by the partitioning algorithm, which may be empty, and let C_1, \dots, C_n be the cavities of the machine. These cavities have a distance between them $d(C_i, C_j)$ give by the shortest path distance in the device topology. This initial mapping is an assignment of partitions to cavities, a bijective map φ . During the following process, the qubits in the cavities remain fixed and therefore the *weight* between partitions is fixed and is given as

$$W(P_i, P_j) = \sum_{q_i \in P_i} \sum_{q_j \in P_j} w(q_i, q_j)$$

where if W is large then the partitions P_i and P_j should be located nearby each other.

We then repeat the following process. First we compute the *gain*, g , obtained by swapping pairs of adjacent partitions. We want to compute if by swapping two partitions they in general get closer to partitions they have a large weight with while simultaneously not getting too far from partitions they were already close to with high weight. Specifically, for every pair of partitions P_i, P_j with $d(\varphi(P_i), \varphi(P_j)) = 1$ we compute the following “forward gain” and “backward gain”

Num. Qubits	QAOA			QFT-Adder			Generalized-Toffoli			Rand-0.4			Rand-0.6		
	10	20	50	10	20	50	9	19	49	10	20	50	10	20	50
Num. of Gates	290	580	1450	45	165	975	119	289	799	19	72	483	24	127	757
Circuit Depth	125	203	276	14	29	74	52	73	95	6	11	27	7	17	37

Table 5.1: Benchmarks and some of their properties.

Model	p_1	p_2	Δ_1	Δ_2	Δ_ℓ	T_1	$T_{1,t}$	$T_{1,c}$
SC-Current	0.001	0.01	100 ns	300 ns	-	73 μ s	-	-
MEQC-Current	0.001	0.01	100 ns	250 ns	150 ns	-	150 μ s	900 μ s

Table 5.2: Error model details for current systems [92, 141].

$$g_f(P_i, P_j) = \sum_{\substack{P_k \\ d(\ell_j, \ell_k) < d(\ell_i, \ell_k)}} W(P_i, P_k) - W(P_j, P_k)$$

$$g_b(P_i, P_j) = \sum_{\substack{P_k \\ d(\ell_j, \ell_k) > d(\ell_i, \ell_k)}} W(P_j, P_k) - W(P_i, P_k)$$

where $\ell_i = \varphi(P_i)$. Then the net gain is given as

$$g(P_i, P_j) = g_f + g_b$$

We choose the pair P_i, P_j with $g(P_i, P_j) > 0$ maximized and swap them. Specifically, we set $\varphi(P_i), \varphi(P_j) = \varphi(P_j), \varphi(P_i)$ and repeat until no more swaps can be made. This heuristic helps locate strongly weighted partitions together on the topology in order to reduce needed SWAPs to interact qubits between them.

Scheduling and Routing: Inserting Loads, Stores, and SWAPs

In this section, we will briefly describe the baseline method for compilation to other current proposed scalable architectures specifically the LNN and 2D mesh, the benchmarks we test

on, and methods of evaluation.

In other common quantum architectures, we only need to insert SWAPs in order to make interacting qubits adjacent. Here, we must also insert Loads and Stores. For an input circuit, we define a *moment* as the maximal set of operations which can be performed simultaneously, or in parallel. We assume all input programs have operations occurring as soon as possible. In each moment, there are operations which can already be executed, that is all one-qubit operations and all two-qubit operations for which both qubits are co-located in the same cavity. We execute these first, in an arbitrary order.

For each single qubit operation we insert a Load, moving the qubit into the cavity's transmon, execute the gate on the transmon, then Store the qubit in it's original location in the resonator. For a two qubit controlled gate, we insert a Load to move the control into the transmon, we execute the gate, and then Store the control in its original location.

Operations which cannot be executed in this moment are ones in which the two qubits are located in different cavities, call them q_0 and q_1 . SWAPs are inserted which modify the qubit mapping. We determine the sequence of SWAPs based on a heuristic algorithm with two goals. The first is to minimize the total number of SWAPs inserted. The second is to displace qubits in the cavities along the path from q_0 to q_1 which are least attached to their current location, i.e will interact the least in the future with qubits in its current cavity.

We begin by computing an updated interaction graph for the program, considering only future moments, updating edge weights of the original interaction graph to be only the number of future interactions between pairs of qubits, $w(q_i, q_j)$. Let $d(C_i, C_j)$ be the distances between cavities on the device, as before. Let λ be a map from qubits to cavities. The distance between qubits q_0, q_1 is then given as $d(\lambda(q_0), \lambda(q_1))$. Then, for every cavity C_i with $d(\lambda(q_0), C_i) = 1$ and $d(C_i, \lambda(q_1)) < d(\lambda(q_0), \lambda(q_1))$ we compute the following *gain* to swap q_0 with some element $q_t \in C_i$

$$\begin{aligned}
gain(q_0, q_t) = & \sum_{q_i \in C_i \setminus q_t} [w(q_0, q_i) - w(q_t, q_i)] + \\
& \sum_{q_i \in \lambda(q_0) \setminus q_0} w(q_t, q_i)
\end{aligned}$$

In the same way we compute $gain(q_1, q_t)$ for $q_t \in C_i$ for every C_i with $d(\lambda(q_1), C_i) = 1$ and $d(C_i, \lambda(q_0)) < d(\lambda(q_1), \lambda(q_0))$. This captures how much is gained (or lost) by making a particular swap of qubits between cavities while maintaining we only consider cavities strictly closer to the target qubit. This ensures we always get closer to our target. We choose to swap the qubits with the greatest gain, and then repeat. Notice we allow either q_0 or q_1 to be moved at each step, so the qubits may meet in some cavity between them. The SWAPs are executed in order by Loading both qubits to be swapped into their transmons, execute the SWAP between transmons and Store the qubits back into their new cavity. This may introduce redundant Loads and Stores, but they can be eliminated via an optimization procedure (Sec 5.1.2).

The gain value above can be modified to favor swapping with cavities which are closest to the target, however, we choose this method because it allows us to sometimes anticipate future interactions of qubits eliminating the need for some future SWAPs. Once all the operations of a moment have been executed, we move to the next moment and perform the same procedure.

Optimizations

The above algorithms produce a valid executable of an input circuit, however with more gates than necessary due to an invariant which requires every operation to be initiated with a Load and terminated with a Store. For example, if we wanted to execute two single qubit gates on the same qubit in sequence we would need to insert two Loads and two Stores resulting in a

total of 6 gates. We can eliminate redundant Loads and Stores by simply checking if a Store from a transmon to a cavity mode is followed immediately by a Load from the same mode to the same transmon.

Another important optimization is to ensure that qubits are loaded from memory only immediately when they are being used. Because T1 times are much lower in the transmon, we do not want any qubit to idle there and instead want to make sure it persists in a cavity for as long as its unused to protect it from decoherence errors. We perform both of these optimizations in our compilation process.

Fundamental Trade Offs

The proposed architecture has several fundamental trade offs distinct from currently operational architectures. The first is the requirement for qubits to be loaded from memory in order to be operated on. This forces a greater degree of sequentialism since only a single operation per cavity can be performed at once. This also means, for every operation we need approximately 3 times as many gates, the Load, the Store, and the gate itself.

This architecture has several benefits which outweigh these costs as we scale to larger devices. When qubits are not being operated on they are stored in memory with coherence times substantially longer than the computational qubits, the transmons. The cavities are random access for the transmon, providing full connectivity between qubits in the same cavity and reducing the average distance between qubits on the device meaning many fewer SWAPs required. Parallelism is still achievable in this architecture. Each transmon-cavity pair operates independently; a well-partitioned algorithm with some degree of parallelism will still be executed in parallel.

Limitations and Potential

The underlying technology which serves as a basis for our proposed architecture is less developed than currently available commercial hardware like that of IBM or Rigetti. However, the current limits such as greater errors, are not fundamental and the technology is expected to evolve with similar trajectory of other current hardware. Our goal is to demonstrate the power of the unique advantages provided by equipping transmons with localized memory. In order to evaluate this, we experiment with more speculative error rates and coherence times which are believed to reflect the potential of the underlying hardware. One possible important advantage not evaluated here is consistency. Current manufacturers have struggled to scale current designs beyond handfuls of transmon qubits with consistently low error rates. With MEQC, obtaining the same number of logical qubits requires fewer total transmons which may help in reducing overall variability by manufacturing a smaller number but higher quality set of transmons. MEQC, while currently in developmental stages, is not fundamentally limited.

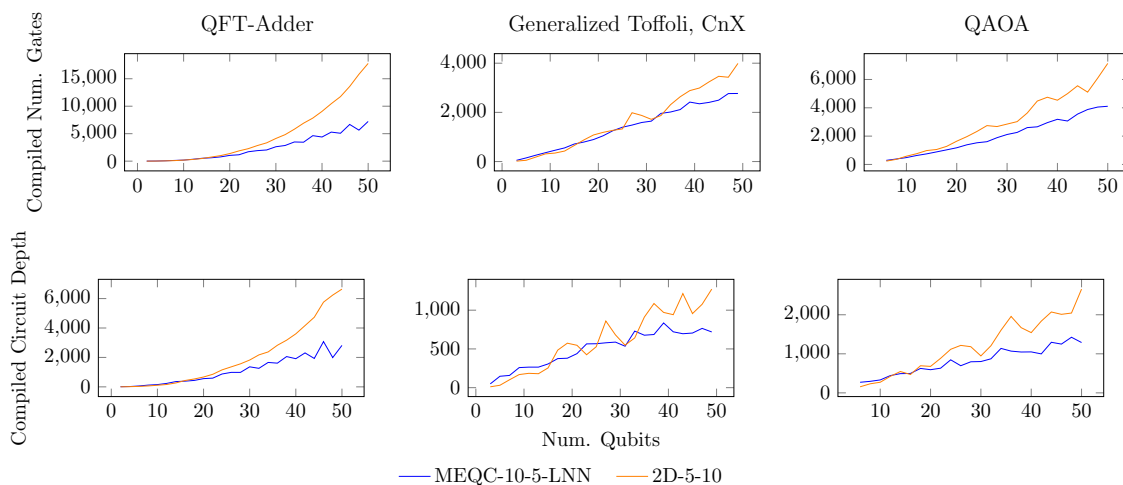


Figure 5.5: The scaling of depth and gate count in a subset of our benchmarks. LNN-50 is omitted because it adds substantially more gates than both of these architectures and as such is not competitive. In many cases, 2D-5-10 is competitive with the proposed architecture, however, clear separation emerges in all cases.

5.1.3 *Experimental Setup*

Compiling to LNN and 2D Mesh Architectures

The compilation procedure for these devices closely reflects the compilation procedure for our proposed MEQC architecture. For each of these, we utilize the heuristic found in [135] to give an initial mapping of circuit qubits to hardware qubits. Operations on these architectures are assumed to be done in parallel if possible and no loading or storing is required. If an operation cannot be done given the initial mapping, the interacting qubits are moved via SWAPs to each other, the operation is performed, and then the qubits are moved back to their original position. This common SWAP strategy assumes the initial qubit mapping is a good one, minimizing the average total distance between qubits which interact frequently and consequently number of required SWAPs.

In order to limit the number of gates applied to specific qubits on the 2D mesh architectures, we look at all shortest paths between interacting qubits given by the bounding rectangle between them. For each path in this rectangle, we choose the path which uses qubits with the fewest number of past and future uses. SWAP paths are done in parallel if possible.

These devices are denoted as LNN- x and 2D- n - m where x is the number of qubits in the chain, and n, m are the dimensions of the mesh. The average distance between qubits in 2D and LNN architectures scales much worse than an MEQC device, as noted in Figure 5.1, meaning on average to interact a pair of qubits many more SWAPs will be required even with MEQC transmons connected poorly.

Benchmarks

We evaluate our proposed architecture compared to other competing designs by compiling a range of both parallel and serial NISQ applications, see Table 5.8.

Arithmetic Circuits

Many important quantum algorithms like Shor's algorithm make use of arithmetic circuits. Many of these circuits like modular exponentiation are beyond the NISQ era. Other smaller arithmetics like addition are much more practical. There are several efficient implementation of these circuits like the Cuccaro Adder [46] and the QFT (Quantum Fourier Transform) Adder [160]. These circuits are highly sequential for a majority of their execution. We focus on the QFT Adder as a representative for this class of circuits. Unlike the Cuccaro adder, the QFT adder has a highly parallel section in the middle of its execution.

The Generalized Toffoli

The Toffoli gate, and its generalized form, is well studied as well as its decompositions. It has many practical uses in a number of applications over a wide range of sizes for example in Grover's search algorithm, larger arithmetic circuits, etc. We consider a generalized decomposition using ancilla given by [15] which allows this operation to be highly parallel, specifically in logarithmic depth.

QAOA - MAX-CUT

One of the most promising algorithms for NISQ era devices is optimization problems. One such problem is QAOA, quantum approximate optimization algorithm [64], which can be used to find approximation solutions to combinatorial optimization problems such as MAX-CUT. The parallelism in this circuit is dependent on the underlying graph in the input problem. We run our QAOA on 4-regular graphs with n nodes, n the number of qubits.

Random Circuits

Finally, we generate a number of random circuits which tend to have very low amounts of parallelism. To generate these circuits, over n trials we select with probability p if an interaction exists between two qubits. If it does, we insert a two qubit interaction between them. We explore $p = 0.4$ and $p = 0.6$ for more sparse and more dense random circuits, respectively.

Evaluation

When evaluating systems, we use the error model details laid out in Table 5.2, where p_1 be the probability of an error occurring on a single qubit operation, p_2 be the probability of an error occurring on a two qubit operation, Δ_1 the duration of a one qubit gate, Δ_2 the duration of a two qubit gate, Δ_ℓ the duration of a Load or Store, T_1 the coherence time of a superconducting qubit in a traditional architecture, $T_{1,c}$ the coherence time of a qubit located in the cavity, and $T_{1,t}$ the coherence time of a qubit located in the transmon of a MEQC device. A dash indicates the value is not relevant to the architecture.

For small circuits and devices, we are able to perform full density matrix simulations and compute the fidelity of circuits compiled for different architectures with noise channels based on various error models. For these simulations, we use typical superconducting error rates for one and two qubit gate errors. Similarly, we use the T1 times provided from [92] for current SC devices and T1 from [141] for the proposed MEQC architecture as well as gate times. For these simulations, we use the Kraus operator formalism [147] for noise simulation in which coherent error channels (bit-flip and phase-flip operators) are symmetric and amplitude damping probability is a function of the T1 times. For our architecture, we have two different T1 times and we apply amplitude damping as a function of the $T_{1,c}$ when qubits are present in the cavity and $T_{1,t}$ when being operated on.

Unfortunately, simulation for even moderate sized programs is extremely hard. For larger

circuits we will use an approximation method to determine an estimate for the probability of success of a circuit. For each gate in the circuit, with given probability we draw if an error occurs. If *any* error occurs during the entire circuit, we consider the program to have failed. This is only a rough approximation and will be directly related to the number of gates in the circuit. In this model, we are unable to quantify the effect of amplitude damping as a result of different T1 times, however, it should be noted in general longer coherence times mean qubits are well isolated from the environment and more protected from decoherence errors.

A more general metric we will use is gate count and depth of the compiled circuit. Usually, as the number of gates increases the probability of success drops and as the depth increases our computation will approach the coherence limit and so it is best to keep both as small as possible.

We explore a variety of different MEQC arrangements, specifically exploring different cavity sizes as well as different arrangements of the transmons. We will abbreviate these machines as MEQC- x - y - z with x the number of transmons, y the number of modes per cavity, and z the arrangement of the transmons, for example LNN for an arrangement in a chain, 2D to refer to a mesh, and Full for full connectivity between transmons. For example, MEQC-2-5-LNN means an MEQC design with 2 transmons connected in a chain each with a single attached cavity containing 5 resonant modes. This device would be able to store 10 qubits in memory. A 2D MEQC can be built by building the chain of cavities in the 3rd dimension [40] and any effect on communication error in the mesh is discussed in Section 5.1.4.

5.1.4 Results and Discussion

Depth and Gate Count Scaling

As noted before, in the proposed MEQC architecture every gate requires approximately two additional operations, the Load and the Store. Furthermore, this approach induces a large degree serialization, specifically when multiple operations are scheduled to be done on qubits

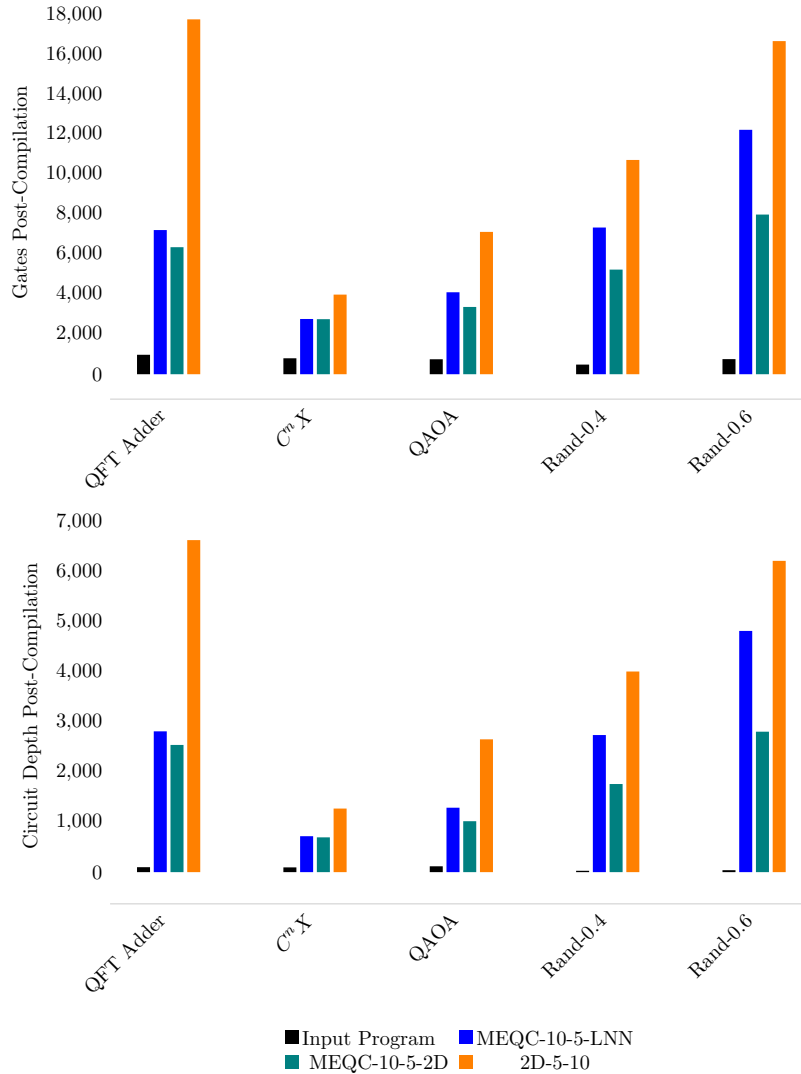


Figure 5.6: Post-compilation number of gates and circuit depth for 50 qubit input programs on all benchmarks. In every case, MEQC with transmons arranged with as a chain or a mesh shows improvement over a more standard 2D mesh qubit arrangement. The increase in gate count in MEQC architectures is approximately 60% due to loads and stores and the rest from SWAPs. By requiring fewer gates, we reduce the possibility of gate-induced error.

Benchmark	Factor Gate Improvement	Factor Depth Improvement
QFT-Adder	2.46x	2.35x
$C^n X$	1.44x	1.76x
QAOA	1.74x	2.06x
Rand-0.4	1.46x	1.46x
Rand-0.6	1.36x	1.29x
Harmonic Mean	1.62x	1.70x

Table 5.3: Summary of the improvements on 50 qubit benchmarks for MEQC-10-5-2D over 2D-5-10. In all cases, we see strict improvement.

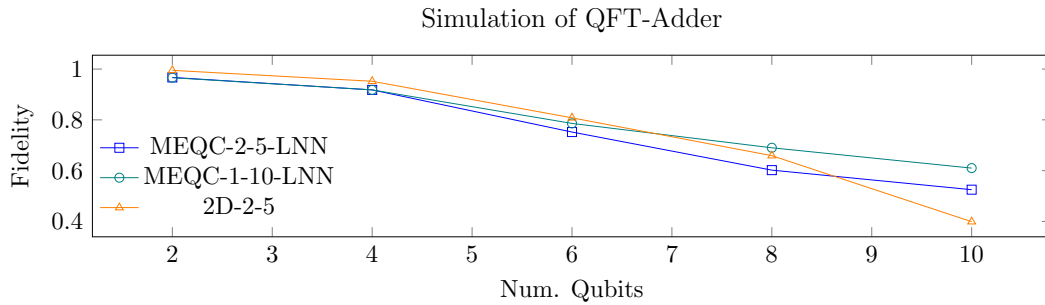


Figure 5.7: Output fidelity for full density simulations of the QFT Adder on 2-10 qubits. Even with more gates in these small instances, programs compiled to MEQC devices are competitive, and at 10 qubits we see the start of advantage.

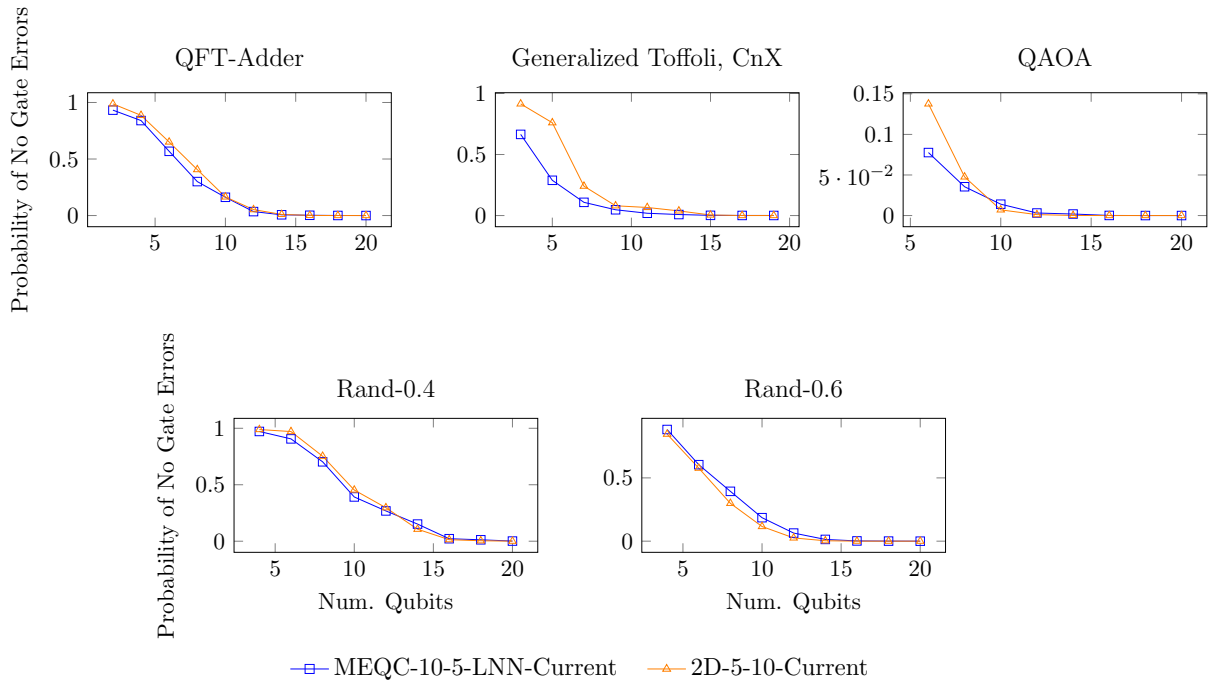


Figure 5.8: An estimation of if no gate errors occur in small program instances. As noted, this only accounts for errors due to one and two qubit gate errors and is not influenced by decoherence errors. Larger is better and in general programs compiled to our proposed architecture are competitive or better than a 2D architecture. We expect with better T1 times and better gate and depth scaling, our architecture will outperform, by increasing the likelihood or successful execution, by a larger margin as programs scale and gate errors improve. All data points were obtained by running 8000 trials of the input compiled input program.

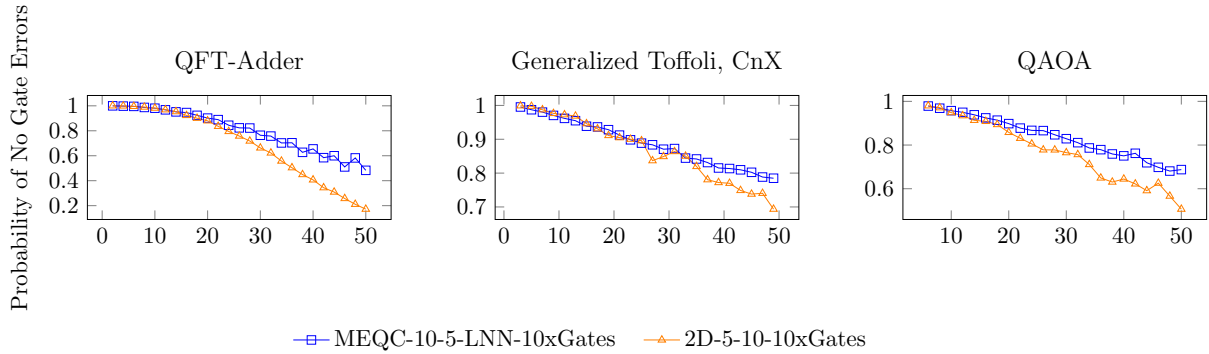


Figure 5.9: With 100x better gates, we begin to see the effect of improved compilation. Specifically, by reducing the total number of gates required for execution on the proposed MEQC devices we reduce the probability of a program failing due to gate errors. Furthermore, with substantially longer T1 times in cavity, qubits stored in memory are protected from decoherence errors. All data points are obtained by running 8000 trials of the input program compiled to the two target architectures.

in the same cavity they cannot be executed in parallel. However, what we lose in some serialization and required extra operations, we make up for in full connectivity within cavity and reduced average distance between qubits.

In Figure 5.5 we compare the scaling of gate counts and depth in a subset of the benchmarks to a 2D and MEQC device. We note all LNN devices with 50 qubits insert dramatically more SWAPs and are completely infeasible, and as such they are omitted. In Figure 5.5, 2D-5-10 and MEQC-5-10-LNN are competitive in both gate count and depth, with a clear separation emerging as we reach the limits of the devices. In Figure 5.6, we compare circuit depth and gate count post compilation for 50 qubit inputs for all benchmarks. In all cases, the number of gates required for execution is much greater than the input program. However, MEQC with transmons connected as either LNN or 2D both improve over a traditional 2D architecture. We expect this separation to be even more pronounced as the devices scale up. For reference, in MEQC architectures approximately 60% of added gates are loads and stores with the remainder due to added SWAPs.

While we might first anticipate architectures which permit every qubit to be operated on in parallel if needed to outperform our proposed architecture which forces only a small

subset of qubits to be operated on at a time, it turns out communication limitations on other near-term devices eliminates this advantage. The number of SWAPs inserted to make a program executable scales extremely poorly. A greater degree of connectivity in near-term devices, and therefore reduced average distance between qubits, is critical to the performance of a program. The insertion of SWAPs itself induces a degree of serialization even in 2D or LNN architectures. Even if a program is written maximizing parallel operations, compilation procedures to transform a program into one satisfying connectivity constraints can evaporate this advantage.

Even though more traditional architectures provide the mechanism for large amounts of parallelism, it may be extraneous. In this case, qubits are still subject to the same low T1 times even when not being operated on. There is no mechanism to protect these qubits while being unused. Initially, MEQC was appealing because it avoided this issue. When qubits are not needed, they can be stored in memory with long T1 times. Random access memory and full connectivity within cavity provides a much more realizable advantage. These architectural details were previously unexplored by physical experimentalists, but MEQC indicates these new memory technologies are a path towards scalability in the near-term.

Effect On Probability of Success: An Estimation

In non-error corrected devices an input program is run thousands of times to obtain a distribution of answers and if run without error we expect the correct answer to appear with the highest probability. These systems have moderate errors and intuitively we expect as the number of gates required for execution increases the probability with which a program succeeds diminishes. Similarly, as the depth of a program increases, perhaps because parallelism is sequentialized or communication operations like SWAPs delay input program's execution, the qubits are more and more likely to be subject to decoherence errors. This also leads to a decreased chance of the correct answer appearing with highest probability in the output

distribution.

In order to evaluate the effect of this system on the probability of success, we simulate some small instances of the QFT-Adder benchmark. We perform full density simulations by injecting into the final compiled circuit both coherent errors, or gate errors, with probability given in Table 5.2 and decoherence errors, specifically depolarizing errors with probability given as a function of the T1 times and gate durations of Table 5.2. Specifically, we used Google’s quantum framework Cirq [2] which contains a full density simulator. We run this simulator on the compiled circuit, which results in the ideal outcome matrix, and the same circuit except with the appropriate error channels which results in a noisy outcome matrix. We compute the pseudo-metric fidelity to evaluate how close the noisy outcome is to the ideal outcome. In Figure 5.7 we show the resulting fidelities for one benchmark. Despite fewer gates and less depth in small circuits compiled to the traditional 2D architecture, MEQC is competitive and begins to edge out the 2D architecture beginning around 10 qubits, indicating the significantly longer T1 times of qubits stored in memory do indeed protect these qubits. MEQC-2D is roughly the same performance as MEQC-LNN in these small programs and is omitted for clarity.

Unfortunately, modeling errors via simulation is difficult to do for even moderately sized programs, requiring exponential space and time. More generally, we rely on an approximation to account for how added gates affect the output of a program. We approximate the probability of no gate errors occurring given the error rates in Table 5.2. This will not account for decoherence errors due to interaction with the environment. However, the T1 times of the proposed device are significantly longer than those of other competing devices. We expect with reduced depth and larger coherence times that programs compiled to MEQC architectures will be less affected by this type of error as we scale, as indicated by the small simulations.

Given current error rates of typical superconducting devices, on all benchmarks up to size

20 we notice competitive if not better probability of no gate errors as in Figure 5.8. In order to better distinguish the effects of Figure 5.5 on this probability, in Figure 5.9 we explore a potential set of futuristic error rates, specifically 100x better gate errors. In this case, we begin to be able to identify the advantage of our proposed architecture with separation emerging on even moderate sized input programs. We expect even greater advantage when all error types are considered.

Sensitivity to Arrangement and Cavity Size

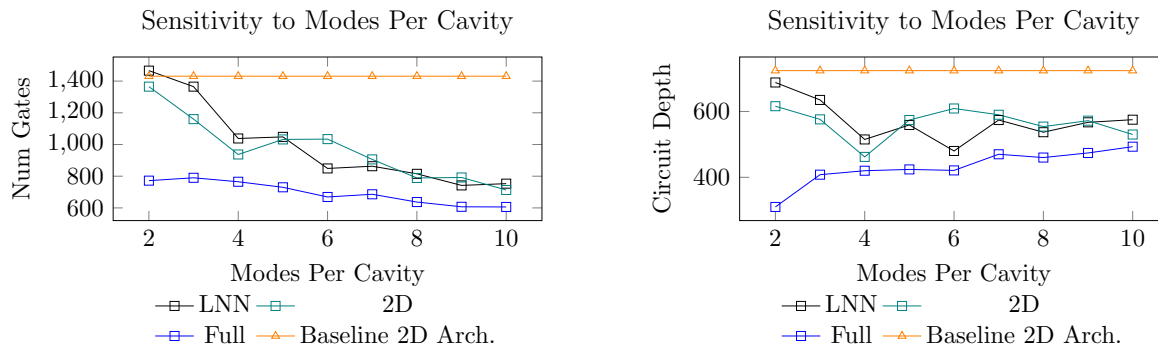


Figure 5.10: Gates and depth of 20 qubit QFT Adder compiled to MEQC architectures with different transmon connectivity and varying cavity sizes. As the number of qubits per cavity increases, we expect the average qubit distance to be reduced meaning fewer SWAPs necessary. However, in MEQC devices operations on qubits in the same cavity cannot be done in parallel. Therefore, we expect lose some degree of parallelism, hence the increase in depth.

One notable feature of Figure 5.6 is that better connected transmon qubits results in fewer gates and less depth. In order to further evaluate this we study two adjustable parameters in the proposed MEQC architecture: the number of modes in resonator cavity and the top-level connectivity of the transmon qubits. Specifically, we study LNN, 2D, and Full connectivity between the transmons as well as cavities of various size.

We expect as the number of modes in the cavity increase, the number of SWAPs, and hence the total gates, required will decrease because of improved average qubit distance. The compilation procedure will prefer to place qubits in these large, well-connected regions of the

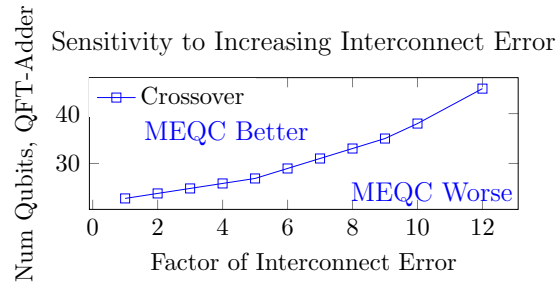


Figure 5.11: Crossover points for various interconnect error rates of the QFT-Adder benchmark. Interconnect in MEQC devices may not be as good as SWAPs in traditional architectures. We study how much interconnect error we can tolerate in the NISQ target of 100 qubit devices with 10^{-5} two qubit error rates. We find we can tolerate up to 12x worse interconnect errors, provided programs of at least size 52.

machine because of this. However, as we noted previously, operations cannot be performed in parallel on qubits co-located in the same cavity. We expect this corresponds to an increase in overall program depth because of reduced parallel operations. We study this tradeoff in Figure 5.10, in which we study three different arrangements of 10 transmons with increasing cavity size for a 20 qubit QFT Adder input program.

While full connectivity shows consistent improvement, it is only very slight advantage. We expect as the size of the input program increases this will become more important but for near-term devices it suggests we do not need as well connected transmons and what matters most is the well connectedness of the cavity itself. The best choice of modes per cavity here is 5, a balance between number of gates and depth, though these curves are a function of the particular input program. For example, for the 50 qubit QFT Adder of Figure 5.6 there is only marginal improvement by moving to a 2D connectivity of the transmons. For other benchmarks, this gain is larger. This also demonstrates if we know gate errors will be much more dominant than idle errors, we can choose to favor designs with larger cavities.

Sensitivity to Interconnect Error Rates

In each of the previous studies, we assume the error rates of SWAPs and communication *between* transmons of MEQC devices is the same as they would be in other more traditional architectures. Demonstrations of this communication protocol have achieved less desirable fidelity, sometimes with error several factors worse than SWAPs in current devices [83, 40]. We call this interconnect error. As we've seen, the scaling of both number of gates and depth is better for the proposed MEQC device and we study the degree to which we can tolerate this greater interconnect error.

In Figure 5.11, we use fixed 100 qubit machines, 2D-10-10 and MEQC-10-10-2D, with two qubit error rates 1000x better than current error rates (e.g. two qubit errors of 10^{-5}), target machines for the NISQ era [23]. We scale the error rate of gates occurring between transmons of the MEQC devices and use our approximation method as before to predict the probability of no gate errors occurring. We locate the crossover points, the program size where it becomes advantageous to use our architecture.

The crossover points are depicted in Figure 5.11 for error factors up to 12x worse interconnect error on the QFT-Adder. We find for NISQ devices up to 100 qubits, a 2D MEQC can tolerate up to 12x worse interconnect error as other 2D architectures. As the number of qubits n grows, the random access advantage of MEQC grows substantially (as long as n does not overly dominate the maximum memory bank size of 10 cavities per transmon).

5.2 Remarks

Current quantum architectures have limited connectivity requiring numerous additional operations in the form of SWAPs to be inserted into the circuit in order to execute them. These added operations often result in not only a lower probability of success but longer execution time. We propose MEQC as an alternative option as we scale into the NISQ era to machines with 50-100 qubits or more.

While initially, the proposed architecture seemed to gain advantage in long coherent qubit memory attached to more traditional computational qubits, the real benefit was in the form of greater connectivity. This reduces the number of SWAPs which must be executed which tend to dominate in other more traditional architectures and subsequently the depth of the compiled program. Fewer gates and lower depth translates directly into a greater chance for success.

While we were only able to demonstrate the effects of long coherence times for cavity qubits for small programs up to 10 qubits with full density simulations, we were able to show that with gate errors expected in the NISQ era the proposed architecture wins out over other proposals. Furthermore, we demonstrated even if interconnect errors in the proposed devices are much worse than SWAP error rates, we can tolerate this as we scale to larger sized programs.

5.3 Time-Sliced Quantum Circuit Partitioning for Modular Architectures

Due developing technology for communicating between different quantum chips [25, 182], we may expect quantum hardware to scale via a modular approach similar to how a classical computer can be scaled increasing the number of processors not just the size of the processors. Two of the leading quantum technologies, ion trap and superconducting physical qubits, are already beginning to explore this avenue and experimentalists project modularity will be the key to moving forward [30, 52, 59, 14, 127, 131, 91]. One such example for ion traps is shown in Figure 5.13 where many trapped ion devices are connected via a single central optical switch. Technology such as resonant busses in superconducting hardware or optical communication techniques in ion trap devices will enable a more distributed approach to quantum computing, having many smaller, well-connected devices with sparser and more expensive non-local connections between them. Optimistically, due to current technology in

the near term, we expect these non-local communication operations to be somewhere between 5-100x higher latency than in-cluster communication².

With cluster-based approaches becoming more prominent, new compiler techniques for mapping and scheduling of quantum programs are needed. As the size of executable computations increase it becomes more and more critical to employ program mappings exhibiting both adaptivity of dynamic techniques and global optimization of static techniques. Key to realizing both advantages is to simplify the problem. Since non-local communication is dominant, we focus on only non-local costs. This simplification, along with static knowledge of all control flow, allows us to map a program in many timeslices with substantial lookahead for future program behavior. This approach would not be computationally tractable on a non-clustered machine.

For devices with many modular components mapping quantum programs translates readily to a graph partitioning problem with a goal of minimizing edge crossings between partitions. This approach is standard in many classical applications such as high performance parallel computing, etc. [101, 169, 166] with the goal of minimizing total latency. Here latency is approximated by the total number of times qubits must be shuttled between different regions of the device. Graph partitioning is known to be hard and heuristics are the dominant approach [66, 153, 106, 86, 54].

While this problem is related to many problems in distributed or parallel computing, there are a few very important distinctions. In a typical quantum program, the control flow is statically known at compile time, meaning all interactions between qubits are known. Furthermore, the no-cloning theorem states we cannot make copies of our data, meaning non-local communication between clusters is *required* to interact data qubits. Finally, any additional non-local operations affect not only latency as they would classically but are directly related to the probability a program will succeed since operations in quantum computing are

2. This work was equally contributed to by JMB, CD and AH with original conceptualization by JMB.

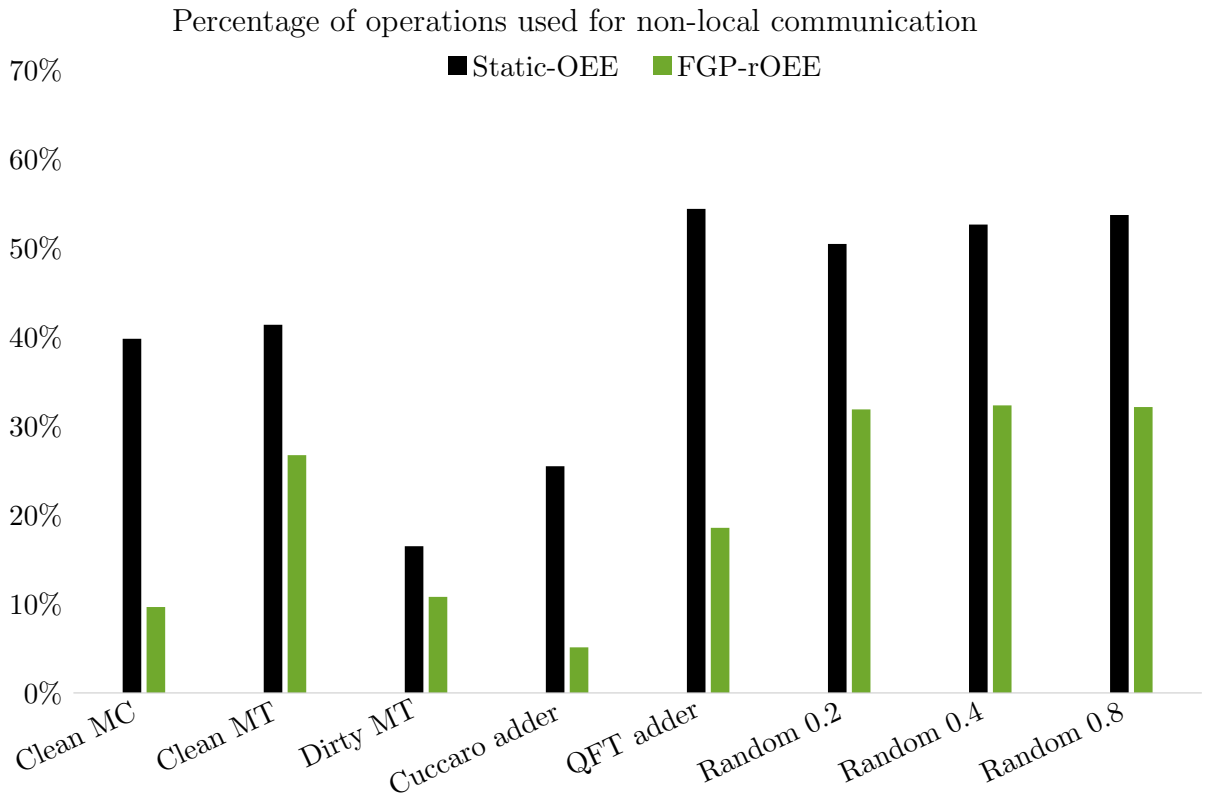


Figure 5.12: Non-local communication overhead in circuits mapped to cluster-based machines. Our new mapping scheme FGP-rOEE reduces the number of operations added for non-local communication on all benchmarks.

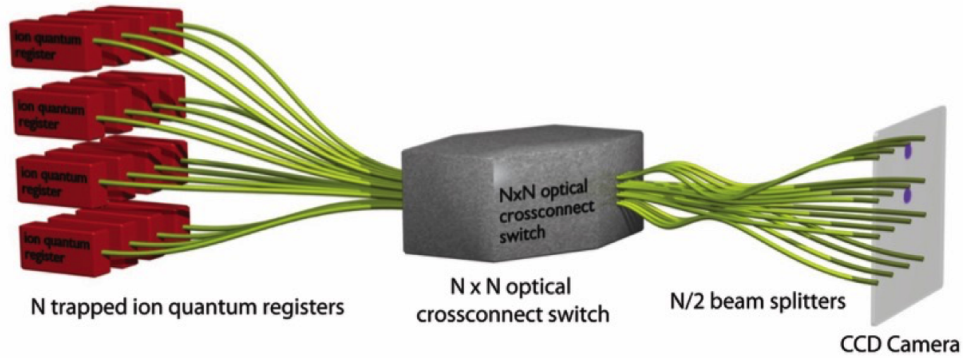


Figure 5.13: An example modular architecture of qubits in individual ion traps connected with optics proposed by Monroe et al [132]. Communication between traps is supported by photon-mediated entanglement. Similar communication for superconducting qubits [40] can facilitate modular architectures for that technology.

error prone and therefore reducing non-local communication is especially critical for successful quantum program execution.

Our primary contribution is the development of a complete system for mapping quantum programs to near-term cluster-based quantum architectures via graph partitioning techniques where qubit interaction in-cluster is relatively free compared to expensive out-of-cluster interaction. Our primary goal is to minimize the communication overhead by reducing the number of low-bandwidth, high-latency operations such as moving qubits which are required in order to execute a given quantum program. Rather than partitioning the circuit once to obtain a generally good global assignment of the qubits to clusters, we find a sequence of assignments, one for each time slice in the circuit. This fine-grained approach is much less studied, especially for this class of architectures. With our techniques, we reduce the total number of non-local communication operations by 89.8% in the best case and 60.9% in the average case; Figure 5.12 shows a few examples of circuits compiled statically versus with our methods.

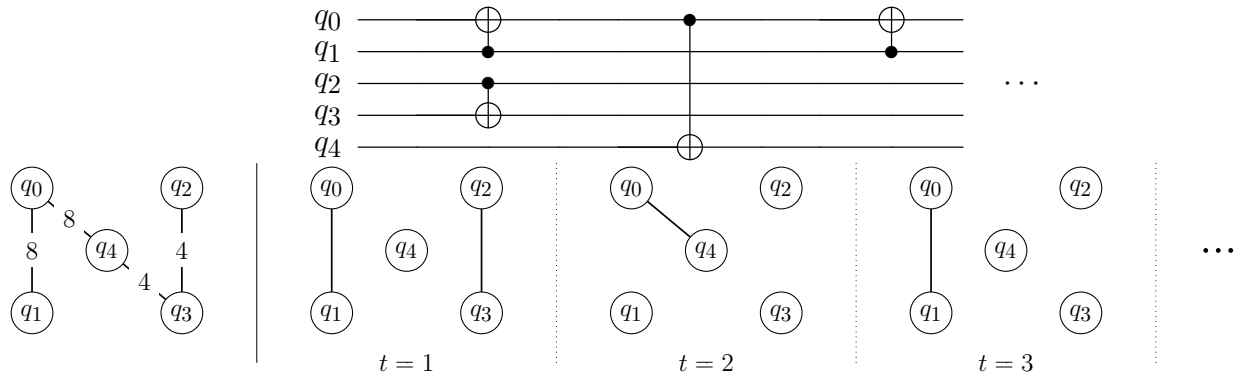


Figure 5.14: (Top) An example of a quantum program with single-qubit gates not shown. The inputs are on the left and time flows to the right toward the outputs. The two-qubit operations here are CNOT (controlled-NOT). (Bottom) The graph representations of the quantum circuit of the above circuit. On the far left is the total interaction graph where each edge is weighted by the total number of interactions for the whole circuit. To the right is the sequence of time slice graphs, where an edge is only present if the qubits interact in the time slice. The sum of all time slice graphs is the total interaction graph.

5.3.1 Relevant Background

In this section, we are motivated by a specific set of architectures or extensions to such architectures, as in [142, 179, 20, 118]. In these devices, qubits are arranged into several regions of high connectivity with expensive communication between the clusters, referred to as non-local communication. These devices naturally lend themselves to mapping techniques which utilize partitioning algorithms.

Quantum programs are often represented as circuit diagrams, for example the one in Figure 5.14a. We define a *time slice* in a quantum program as a set of operations which are parallel in the circuit representation of the program. We express time slices as a function of both the circuit representation and limitations of the specific architecture. We also define a *time slice range* as a set of contiguous time slices; we also refer to them as *slices* and when no length is specified, it will be assumed to be of length 1.

For evaluation, we consider two primary metrics: the *width* and the *depth* of a circuit. The width is the total number of qubits used and the depth, or the run time, is the total number of time slices required to execute the program. Qubit movement operations which

are inserted in order to move interacting qubits into the same partition contribute to the overall depth of the circuit.

We consider two abstract representations of quantum programs: the total interaction graph and a sequence of time slice interaction graphs, examples of which are found in Figure 5.14b. In both representations, each qubit is a vertex and edges between qubits indicate two-qubit operations acting on these qubits. In the total interaction graph, edges are weighted by the total number of interactions between pairs of qubits. In time slice graphs, an edge with weight 1 exists only if the pair of qubits interact at that time slice.

Graph Partitioning

Static Partitioning

Finding graph partitions is a well studied problem [66, 153, 106, 87] and is used frequently in classical architecture. In this paper, we consider a variant of the problem which fixes the total number of partitions and bounds the total number of elements in each partition. Specifically, given a fixed number of partitions k , a maximum partition size p , and an undirected weighted graph G with $|V(G)| \leq k \cdot p$ we want to find a k -way assignment of the vertices to partitions such that the weight of edges between vertices in different partitions is minimized. This can be rephrased in terms of statically mapping a quantum circuit to the aforementioned architectures. Let the total interaction graph be G and let k and p fixed by the topology of the architecture. Minimizing the edge weight between partitions corresponds to minimizing the total number of swaps which must be executed.

Solving for an optimal k -way partition is known to be hard [36], but there exist many algorithms which find approximate solutions [106, 153, 66]. There are several heuristic solvers such as in [105, 104] which can be used to find approximate k -way partition of a graph. However, they often cannot make guarantees about the size of the resulting partitions, preventing us from using them for the fixed size partitioning problem.

Partitioning Over Time

Rather than considering a single graph to be partitioned we instead consider the problem of generating a *sequence* of assignments of qubits to clusters, one for each moment of the circuit. We want to minimize the total number of differences between consecutive assignments, naturally corresponding to minimizing the total number of non-local communications between clusters. This problem is much less explored than the prior approach. Partitioning in this way guarantees interacting qubits will be placed in the same partition making the schedule for the input program immediate. In the case of a static partition, which gives only the initial mapping, a further step is needed to generate a schedule.

Optimal Compilation and Exact Solvers

It is too computationally expensive to find a true optimal solution for even reasonably sized input programs. Use of constraint-based solvers has been used recently to look for optimal and near-optimal solutions [135, 150, 148]. Unfortunately, these approaches will not scale in the near-term let alone to larger, error-corrected devices. We explored the use of these solvers but found them to be too slow. Finding a static mapping with SMT is impractical with more than 30 to 40 qubits, and SMT partitioning over time is impractical when number of qubits times the depth became more than 40.

5.3.2 Mapping Qubits to Clusters

We define an *assignment* as a set of partitions of the qubits, usually at a specific time slice. We present algorithms which take a quantum circuit and output a *path*, defined as a sequence of assignments of the qubits with the condition that every partitioning in the sequence is *valid*. An assignment is valid if each pair of interacting qubits in a time slice are located within the same partition. Finally, we define the *non-local communication* between consecutive assignments as the total number of operations which must be executed to transition the

system from the first assignment to the second assignment. The total communication of a path is the sum over all communication along the path.

Computing Non-local Communication

To compute the non-local communication overhead between consecutive assignments of n qubits, we first construct a directed graph with multiple edges where the nodes in the graph are the partitions and the edges indicate a qubit moving from partition i to partition j . We extract all 2-cycles from this graph and remove those edges from the graph. We proceed extracting all 3-cycles, and so on and record the number of k -cycles extracted as c_k . When there are no cycles remaining, the total number of remaining edges is r , and the total communication overhead C is given by

$$C = r + \sum_{k=2}^n (k-1) \cdot c_k$$

The remaining edges indicate a qubit swapping with an unused qubit. We repeat this process for every pair of consecutive assignments in the path to compute the total non-local communication of the path. These cycles specify where qubits will be moved with non-local communication.

Baseline Non-local Communication

As a baseline we consider using a *Static Mapping* using an owner computes model, which takes into account the full set of qubit interactions for the circuit, providing a generally good assignment of the qubits for the entire duration of the program, called the static assignment. At each time step in the circuit, a good static assignment ensures, on average, qubits are not *too far* from other qubits they will interact with frequently. We find the assignment which requires the fewest number of swaps from the static assignment but has each pair of

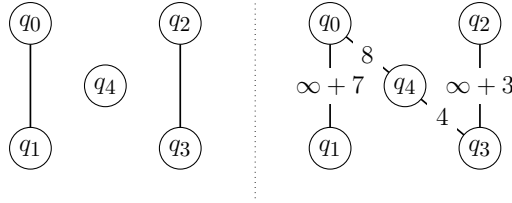


Figure 5.15: An example of a time slice graph with lookahead weights based on the circuit in Figure 5.14. We take the graph from the left and add weight to the edges of qubits that interact in the future. In this case, we take the weight equal to the number of times the qubits will interact in the future.

interacting qubits in a common partition. These assignments form a path for the computation. We refer to this method of path generation in conjunction with a partitioning algorithm, for example Static Mapping with OEE (Overall Extreme Exchange, discussed further later) is referred to as Static-OEE.

Fine Grained Partitioning

The primary approach we developed to dynamically map a circuit to hardware is *Fine Grained Partitioning* (FGP). In this algorithm, we find an assignment at every time slice using the time slice graphs. By default, these time slice graphs give only immediately local information about the circuit but have no knowledge about upcoming interactions. Alone, they only specify the constraints of which qubits interact in that time slice. The key advantage for this method is using *lookahead weights*. The main idea is to construct modified time slice graphs capturing more structure in the circuit than the default time slice graphs. We refer to these graphs as time slice graphs with lookahead weights, or *lookahead graphs*.

To construct the lookahead graph at time t , we begin with the original time slice graph and give the edges present infinite weight. For every pair of qubits we add the weight

$$w_t(q_i, q_j) = \sum_{t < m \leq T} I(m, q_i, q_j) \cdot D(m - t)$$

to their edge, where D is some monotonically decreasing, non-negative function, which we

call the lookahead function, and $I(m, q_i, q_j)$ is an indicator that is 1 if q_i and q_j interact in time slice m and 0 otherwise, and T is the number of time slices in the circuit. The new time slice graphs consider the remainder of the circuit, more heavily weighting sooner interactions. The effectively infinite weight on edges between interacting qubits is present to guarantee any assignment will place interacting qubits into the same partition. An example is shown in Figure 5.15.

The final mapping of the qubits in our model is obtained by partitioning each of these time slices. Iteratively, we find the next assignment with a partitioning algorithm, seeded with the assignment obtained from the previous time slice. The first can choose a seed randomly or use the static assignment (presented in 5.3.2). The new weights in the time slice graphs will force any movement necessary in the partitioning algorithm. Together, these assignments give us a valid path for the circuit to be mapped into our hardware.

Choosing the Partitioning Algorithm

We assume full connectivity within clusters and the ability to move between clusters. These assumptions give us the liberty to tap into well studied partitioning algorithms. The foundation of many partitioning algorithms is largely considered to be the Kernighan-Lin heuristic for partitioning graphs with bounded partition sizes [106, 66, 153]. The KL heuristic selects pairs of vertices in a graph to exchange between partitions based on the weights between the vertices themselves and the total weight between the vertices and the partitions.

We consider a natural extension of the KL algorithm, Overall Extreme Exchange presented by Park and Lee [153]. The OEE algorithm finds a sequence of pairs of vertices to exchange and makes as many exchanges as give it an overall benefit. Using OEE, the Fine Grained Partitioning scheme often over corrects (see Figure 5.18). If a qubit needs to interact in another partition, then it can “drag along” a qubit it is about to interact with because OEE attempts to minimize weight between partitions regardless of its relation to the previous or

next time slice graphs. Choosing an optimal partitioning algorithm would not give better solutions to our non-local communication based mapping problem. Instead, we consider a more relaxed version of a partitioning algorithm using the KL heuristic.

Relaxing the Partitioning Algorithm

We provide relaxed version of the algorithm better suited to generating a path over time, called relaxed-OEE (rOEE). We run OEE until the partition is valid for the time slice (all interacting qubits are in the same partition) and then make no more exchanges. This is similar in approach to finding the time slice partitions in our Static Mapping approaches. It is critically important we make our exchange choices using lookahead weights applied to the time slice graphs. Choosing without information about the upcoming circuit provides no insight into which qubits are beneficial to exchange. As a side benefit, making this change strictly speeds up OEE, an already fast heuristic algorithm. Although a strict asymptotic time bound for OEE is difficult to prove, rOEE never took more than a few seconds on any instance it was given.

With such a significant non-local communication overhead improvement (see Figure 5.18), this relaxed KL partitioning algorithm is much better suited for the problem at hand. It has the ability to take into account local structure in the circuit and avoid over correcting and swapping qubits unnecessarily.

5.3.3 Lookahead Weights

Finding a suitable lookahead weight function to use in Fine Grained Partitioning is necessary to maximize the benefit gained from choosing our swaps appropriately between time slices. We only require the lookahead function to be monotonically decreasing and non-negative. Throughout this section, we denote our lookahead weight function as D .

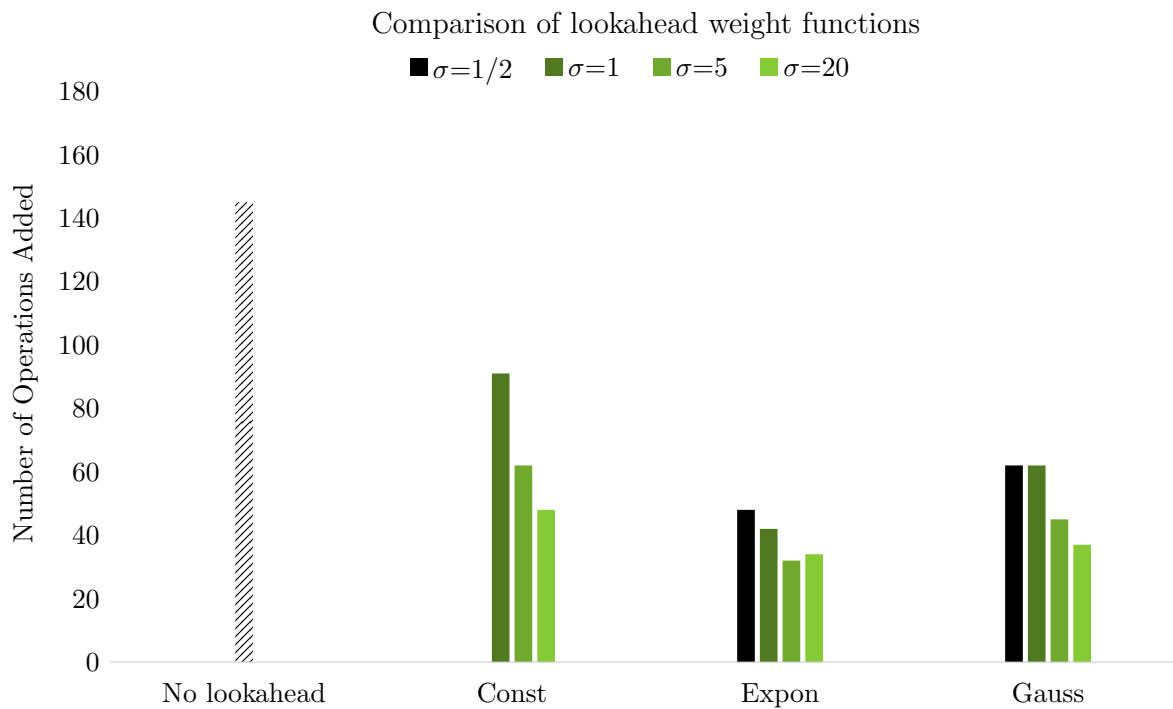


Figure 5.16: The effect of different lookahead functions with various σ on non-local communication in the Cuccaro adder, a very regular circuit, with 76 data and 24 ancilla qubits using FGP-rOEE. We see the exponential function outperforms the others for a circuit of highly regular structure.

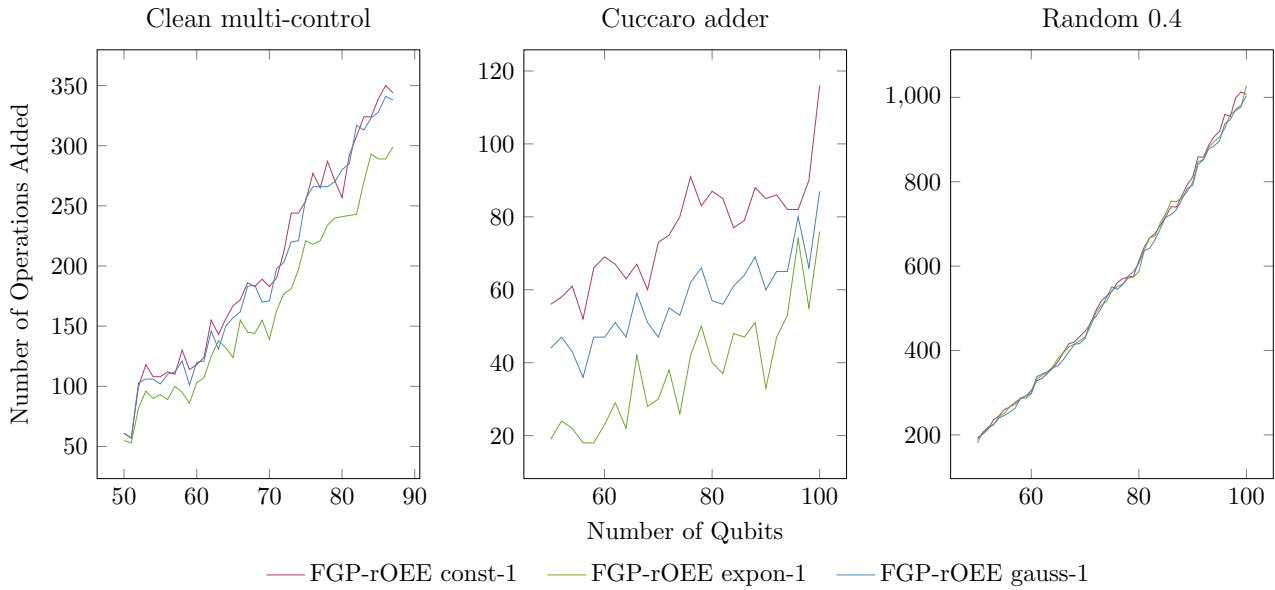


Figure 5.17: The non-local communication, measured in number of operations between clusters added, for our representative benchmark circuits mapped by each FGP-rOEE using different lookahead functions, each with $\sigma = 1$. The x-axis is the number of input/output qubits. The remainder are used as ancilla for clean multi-control. The exponential function is better on all instances of Clean multi-control and Cuccaro adder, and there is no substantial advantage of one function over the others in the random circuit.

	Clean multi-control			Clean multi-target			Dirty multi-target			Cuccaro adder		
Data Qubits	50	76	87	50	76	100	50	76	100	50	76	100
Depth	82	265	846	17	22	99	26	34	99	435	669	885
Two Qubit Op Count (Unmapped)	760	2040	2488	57	85	99	103	157	99	505	778	1030
Non-local Comm. Ops (Static-OEE)	288	1297	1928	35	60	169	34	31	169	159	243	365
Non-local Comm. Ops (FGP-rOEE expon-1)	55	218	299	21	31	72	17	19	72	19	42	76

	QFT adder			Random 0.2			Random 0.4			Random 0.8		
Data Qubits	50	76	100	50	76	100	50	76	100	50	76	100
Depth	72	111	147	15	23	30	28	41	54	46	67	86
Two Qubit Op Count (Unmapped)	625	1444	2500	246	588	995	477	1156	1997	965	2260	3944
Non-local Comm. Ops (Static-OEE)	512	1144	2542	180	486	863	344	993	1795	682	1944	3462
Non-local Comm. Ops (FGP-rOEE expon-1)	131	329	541	96	275	498	181	552	1028	386	1070	1964

Table 5.4: A subset of our benchmarks. Clean multi-control has a maximum size of 87. With more than 87 data qubits and fewer than 13 clean ancilla, the depth of the multi-control decomposition is too large to run on these cluster-based machines with predicted error rates.

Natural Candidates

We explore a few natural candidate weighting functions from the huge space of possible functions. In each of the functions we explore below, we vary a stretching factor or scale σ which can be tuned for the given circuit, providing a trade-off between local and global information.

Constant Function

$$D(n) = \begin{cases} 1 & n \leq \sigma \\ 0 & n > \sigma \end{cases}$$

A constant function captures a fixed amount of local information in the circuit. This is just the number of times the pair of qubits interact in the next σ time slices. For $\sigma = 0$, this function corresponds to no lookahead applied.

Exponential Decay

$$D(n) = 2^{-n/\sigma}$$

An exponential is a natural way to model a decaying precedence. When $\sigma \leq 1$, any interaction will always have a weight at least as high as the sum of interactions after it.

Gaussian Decay

$$D(n) = e^{-n^2/\sigma^2}$$

Similar to an exponential, a Gaussian is natural to model decaying precedence with more weight given to local interactions.

Evaluating Lookahead Functions

To evaluate the choice of lookahead function as well as choice of σ , we study Fine Grained Partitioning using rOEE with all of the above candidate functions with varying σ on benchmarks of various types: those with lots of local structure (a quantum ripple carry adder), those with very little structure (a random circuit), and those which lie somewhere in between (a Generalized Toffoli decomposition).

In Figure 5.16, we show an example of a circuit which benefits from having a large scale σ , the Cuccaro Adder [46]. In contrast, all of the random benchmarks benefit from having small σ values, functions which decay quickly even for small n .

We also compare the different natural lookahead functions we described in the previous section on some representative benchmarks in Figure 5.17. In these figures, we see the exponential decay has a clear benefit over the rest in the structured circuits of the Multi-Control gate and the Cuccaro Adder. In random circuits, there seems to be no clear benefit to any of the lookahead functions, so long as they have some small lookahead scaling factor. So, we use exponential decay with $\sigma = 1$ for our primary benchmarks.

5.3.4 *Experimental Setup*

All experiments were run on an Intel(R) Xeon(R) Silver 4100 CPU at 2.10 GHz with 128 GB of RAM with 32 cores running Ubuntu 16.04.5. Each test was run on a single core. Our framework runs on Python 3.6.5 using Google's Cirq framework for circuit processing and

for implementing our benchmarks [2]. For testing exact solvers, we used the Z3 SMT solver [48], though results could not be obtained for the size of benchmarks tested because Z3 never completes on problems this size.

Benchmarks

We benchmark the performance of our circuit mapping algorithms on some common sub-circuits used in many algorithms (for example Shor’s and Grovers) and, for comparison, on random circuits. Our selection of benchmarks covers a wide variety of internal structure. For every benchmark, we use a representative cluster-based architecture with 100 qubits with 10 clusters each containing 10 qubits but our methods are not limited to any size. We sweep over the number of qubits used from 50 to 100, when in the cases of a few benchmarks the remaining qubits are available for use as either clean or dirty ancilla³.

Generalized Toffoli Gate

The Generalized Toffoli gate ($C^n U$) is an n -controlled U gate for any single qubit unitary U and is well studied [147, 57, 71, 26, 81]. A $C^n X$ gate works by performing an X gate on the target conditioned on all control qubits being in the $|1\rangle$ state. There are many known decompositions [70, 84, 15] both with and without the use of ancilla. A complete description of generating these circuits is given by [12], which provides a method for using clean ancilla.

Multi-Target Gate

The multi-target gate performs a single-qubit gate on many targets conditioned on a single control qubit being in the $|1\rangle$ state. This is useful in several applications such as one quantum adder design [71] and can also be used in the implementation of error correcting codes [51].

3. An ancilla is a temporary quantum bit used often to reduce the depth or gate count of a circuit. “Clean” indicates the initial state of the ancilla is known while “dirty” means the state is unknown.

These circuits can be generated with different numbers of ancilla (both clean and dirty), as given by [12].

Arithmetic Circuits

Arithmetic circuits in quantum computing are typically used as subcircuits of much larger algorithms like Shor's factoring algorithm and are well studied [57, 71, 125]. Many arithmetic circuits, such as modular exponentiation, lie either at the border or beyond the range of NISQ era devices, typically requiring either error correction or large numbers of data ancilla to execute. We examine two types of quantum adders - the Cuccaro Adder and the QFT Adder - as representatives of a class of highly structured and highly regular arithmetic circuits [46, 160].

Random Circuit

The gates presented above have a lot of regular structure when decomposed into circuits. We want to contrast this with circuits with less structure.

We create these random circuits by picking some probability p and some number of samples and generate an interaction between two qubits with probability p for each sample. These circuits have the same structure as QAOA solving a min-cut problem on a random graph with edge probability p , so these circuits are a realistic benchmark.

Circuit to Hardware

We begin with a quantum program which is specified at the gate level, consisting of one and two qubit gates. We then generate the total interaction and time slice graphs, where we assume gates are inserted at the earliest possible time. Any further optimization, such as via commutivity or template matching, should be done prior to mapping the program to

hardware. We also take the specifications of the hardware, such as number of clusters and the maximum size of the clusters, which constrain possible mappings.

We use our rOEE as our algorithm for Fine Grained Partitioning. Therefore, we pass the total interaction graph to a static partitioning algorithm to obtain a good starting assignment. This serves as a seed to rOEE rather than starting with a random assignment which may introduce unnecessary starting communication. To the time slice graphs, we apply the lookahead function to obtain the lookahead graphs. We run rOEE on this set of graphs to obtain an assignment sequence such that at every time slice qubits which interact appear in the same bucket. This assignment describes what non-local communication is added before each slice. Finally, we compute the cost and insert the necessary movement operations into the circuit to move interacting qubits into the same partition, this is a path. As a byproduct, by generating a partitioning over time, we obtain a schedule of operations to be performed.

5.3.5 *Results and Discussion*

We run our mapping algorithms on each of our benchmark circuits. The results are shown in Figure 5.18.

Baseline mapping and the original version of OEE perform worse than our best scheme on any benchmark tested. Baseline mapping uses global structure of the graph, but often maintains this structure too much throughout the execution of the circuit. This lack of local awareness and rigid nature of the Static Mapping limits its usefulness. Most out of the box graph partitioning algorithms are designed to only minimize the edge weight between partitions; this will tend to over correct for local structure in the circuit. FGP can overcome this limitation with its choice of partitioning algorithm. By relaxing the partitioning algorithm and not requiring local optimality, we only move qubits until all interacting pairs are together, we require far fewer non-local operations.

The most noticeable changes between FGP-OEE and FGP-rOEE are on the clean multi-

control gate with many controls and on the Cuccaro adder. Here, there are often consecutive, overlapping operations with little parallelism. With this structure, after the first operation is performed, the original OEE algorithm will exchange qubits to comply with the next time slice for the next operation. OEE is required to separate qubits which will later interact. To minimize the total crossing weight between partitions, more qubits are shuffled around, usually towards this displaced qubit. In rOEE, this reshuffle optimization never takes place because we terminate once each pair of interacting qubits in a time slice is placed in a common partition. The reshuffling detracts the overall non-local communication when running the circuit because of how often qubits will be displaced from their common interaction partners. In rOEE, not reshuffling keeps the majority of the qubits in sufficiently good spots and the displaced qubit has the opportunity to immediately move back with its interaction partners later.

We include the algorithm Fixed Length Slicing as an alternative not presented in this paper. It is a method with slower computation which explores grouping time slices at fixed intervals. Fixed Length Slicing was consistently the best performing time slice range based mapping algorithm, so we present it in our results. FLS-OEE only beats FGP-rOEE on some instances of the multi-target benchmarks and consistently performs worse on all other benchmarks.

In Figure 5.12, we show the percentage of operations used for non-local communication for each of the benchmark circuits, and in Table 5.5 we show the percent improvement of our algorithm over the baseline. On average, we save over 60% of the non-local communication operations added. When each non-local communication operation is implemented in hardware, the amount of time each takes is significantly longer than the operations between the qubits in the clusters [134]. Based on current communication technology, we expect these non-local communication operations to take anywhere from 5x to 100x longer than local in-cluster operations. Furthermore, the choice in technology limits how many of these expensive

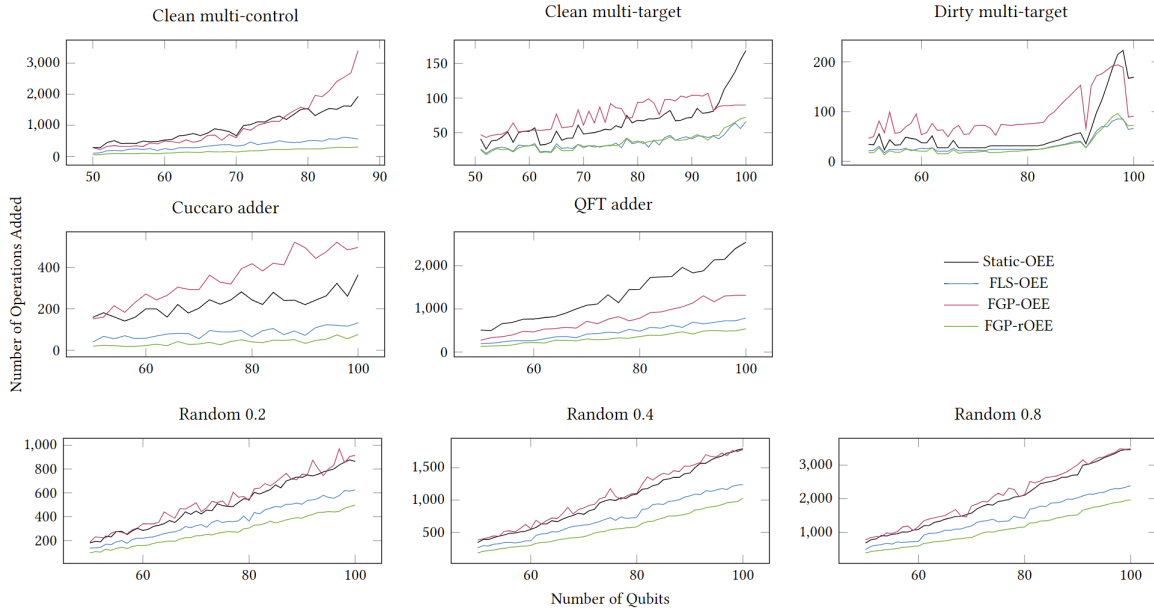


Figure 5.18: The non-local communication overhead for our benchmark circuits mapped by each mapping algorithm. The x-axis is the number of qubits that are used in the circuit. The y-axis is the number of non-local communication operations inserted to make the circuit executable in our hardware model. In Clean multi-control, Clean multi-target, and Dirty multi-target, the remainder of the 100 qubits are used as ancilla (clean or dirty determined by the circuit name). FGP-rOEE outperforms all other mapping algorithms on all but the multi-target circuits, and shows substantial improvement over the static baseline. As the size of the circuit increases, rOEE tends to outperform by a greater margin, indicating scales better into the future.

operations can be performed in parallel.

In Table 5.6 we compute the estimated running time based on this ratio of costs and show that by substantially reducing the non-local communication via FGP-rOEE, we can drastically reduce the expected run time. We compare our algorithm to the baseline when non-local communication can be performed in parallel (such as in optically connected ion trap devices) and when it is forced to occur sequentially (as when using a resonant bus in superconducting devices). Based on current technology, a 5-10x multiplier is optimistic while 100x is realistic in the near term.

Table 5.5: Comparing Static-OEE against FGP-rOEE over all benchmarked instances. We obtain improvement across the board with the worst case still reducing non-local communication by 22.6%.

% Reduction	min	max	gmean
Clean multi-control	78.1	84.9	81.9
Clean multi-target	30.8	59.6	44.7
Dirty multi-target	22.6	65.1	39.9
Cuccaro adder	79.1	89.8	85.0
QFT adder	76.6	84.5	81.5
Random 0.2	52.4	57.8	55.3
Random 0.4	53.6	59.0	57.0
Random 0.8	57.0	60.4	59.1
Aggregate	22.6	89.8	60.9

Table 5.6: Estimated execution time of the clean multi-control benchmark with 76 data qubits and 24 ancilla. Two-qubit gates take 300ns [92] and the multiplier indicates how many times longer non-local communication operations take.

Multiplier	Sequential Comm.		Parallel Comm.	
	Static-OEE	FGP-rOEE	Static-OEE	FGP-rOEE
$5x$	2.0 ms	0.41 ms	0.67 ms	0.26 ms
$10x$	4.0 ms	0.73 ms	1.3 ms	0.43 ms
$100x$	39 ms	6.6 ms	12 ms	3.6 ms

5.3.6 *Remarks*

Alternative to using near-optimal graph partitioning algorithms to find a single static assignment for an entire circuit, we show considering the locality in a circuit during a mapping gives a reduction in the total non-local communication required when running a quantum circuit. There is a natural restriction in using static mappings suggesting the problem of mapping qubits to cluster-based architectures has a different structure than partitioning a single graph for minimum weight between the partitions. Our modification to OEE no longer attempts to optimize the weights at every time slice. It is much more effective in practice to guide the partitioning based on heuristics and not to find the optimal value for every time slice. Optimality at every time slice does not correspond to a global reduction in non-local communication overhead.

We propose to use similar schemes for other cluster-based quantum hardware, especially those based on internally connected clusters. In our model, the different clusters of the architecture are also very well connected, but is not limited to only this specific instance of a clustered architecture.

Our proposed algorithm produces partitions based on a simplifying assumption about the connectivity of the clusters because the cost of non-local communication is substantially more expensive than any in-cluster operations. Our method can be adapted to other cluster-based architectures by first applying our partitioning algorithm to obtain good clusters of operations and then adding a device-specific scheduling algorithm for scheduling much cheaper in-cluster operations.

A relaxed version with well chosen lookahead functions of a heuristic outperforms a well selected initial static mapping. Using lookahead weights has been explored previously, as in [188], and more can be done to better choose the lookahead function, for example based on a metric of circuit regularity. Techniques for mapping which attempt to solve for near optimal mappings will not scale and instead heuristics will be the dominant approach. Our

approach is computationally tractable and adaptable to changes in machine architecture, such as additional or varied size clusters.

Non-local communication overhead in quantum programs makes up a large portion of all operations performed, therefore, minimizing non-local communication is critical. In recent hardware [134], the cost of moving between clusters makes non-trivial computation impossible with current standards for mapping qubits to hardware. Reducing this hardware bottleneck or finding algorithms to reduce the non-local communication are critical for quantum computation. We reduce this cost substantially in cluster-based architectures.

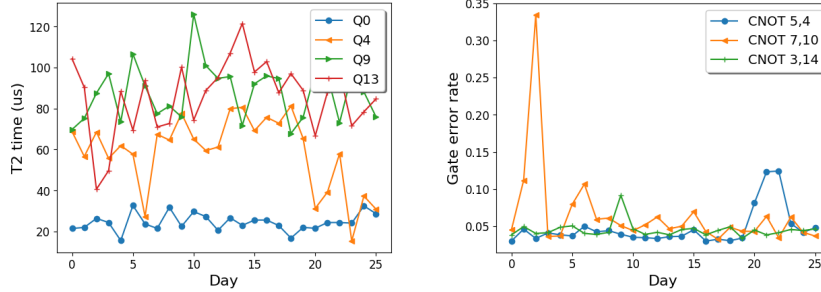
5.4 Noise-Adaptive Compiler Mappings for Noisy Intermediate Scale Quantum Computers

4

The term Noisy Intermediate-Scale Quantum (NISQ) computers refer to the current and near-term QC systems which have roughly 1000 qubits or fewer—typically too small to employ error correction codes (ECC) [155]. While resource constrained, NISQ machines offer an important step forward: if used properly, they can demonstrate QC applications generating useful results. Making good use of NISQ hardware, however, requires very efficient, near-optimal mappings of algorithms onto them. This section proposes a suite of optimization- and heuristic-based approaches for mapping applications onto NISQ hardware, and evaluates them by running the mapped executables on a public 16-qubit IBM QC, *IBMQ 16 Rueschlikon* referred to usually as *IBMQ16* but has since been retired [93].

A good mapping of a quantum algorithm onto NISQ hardware requires first an intelligent initial placement of the program qubits onto the hardware qubits in order to reduce communication requirements. Second, it requires efficient orchestration of operations both for the

4. JMB’s contributions include performance testing, refinement of the solver model, and design of the scalable heuristics.



(a) Coherence time (T2) (b) CNOT gate error rate

Figure 5.19: Daily variations in qubit coherence time (larger is better) and gate error rates (lower is better) in *IBMQ 16 Rueschlikon*. The qubits and gates that are most or least reliable are different across days.

computation itself, and also for the additional SWAP operations which communicate state between hardware qubits. Third and most importantly, mapping decisions must reduce the likelihood of operational or decoherence errors which cause the program run to fail to achieve a useful answer. Our work performs mappings using the daily calibration data provided by IBM in order to avoid using unreliable qubits and to prioritize qubit positioning which reduces the likelihood of communication (SWAP) errors. For example, Fig. 5.19 shows large daily variations in the gate error rates and coherence times of the qubits of the *IBMQ16* instance on which we experiment. Our contributions are:

- We develop an LLVM [117] compiler which optimally or near-optimally maps quantum programs to OpenQASM assembly code [45] and then to the web-accessible *IBMQ16* machine for real-system evaluation. For 12 QC programs written in the Scaffold quantum programming language [7], we use this framework to explore how optimal and heuristic mapping methods, qubit movement policies, and the intelligent adaptation to machine calibration data can affect the quality of the compiled code.
- In particular, our compiler provides up to 1.68X gain in execution time and 9X gain in success rate over an optimal but calibration-unaware baseline. Our compiler obtains an average 2.9X improvement (up to 18X) in success rate, and an average 2.7X improvement in execution time (up to 6X), compared to the IBM Qiskit compiler [8], which is the

industry standard for *IBMQ16*.

- Although compile-time is not a first-order design goal, QC compilers must scale well enough for intelligent compilation to be tractable throughout NISQ-range machines. We show that our methods based on Satisfiability Modulo Theory (SMT) scale well up to 32 qubits. Further, we have developed calibration-aware heuristic methods which produce executables with similar reliability and execution time as the SMT approaches, but with more scalable compile-times.
- Across the 12 benchmarks, we study the influence of application instruction mix and time varying qubit error characteristics on compiled programs. For example, applications for which our compiler can identify zero-qubit-movement mappings have substantially higher likelihood of success (up to 2.8x), compared to programs which require even a single qubit movement operation.

Overall, NISQ systems are important to QC progress because their success in demonstrating quantum supremacy and running small but useful QC programs is an important stepping-stone in the maturation of this technology. In its leveraging of intelligent and calibration-aware mapping techniques to significantly improve execution time and success rate of quantum executions, our tool makes an important contribution in helping close the gap to quantum supremacy and advancing toward practical QC.

5.4.1 *Relevant Background*

NISQ Systems

NISQ systems are near-term experimental quantum systems expected to scale to a few hundred qubits, paving the way towards large-scale QC [155]. Qubits in NISQ systems have short coherence time, high gate error rates and limited qubit connectivity. They are typically too resource-constrained to implement error-correcting codes (ECC).

Underlying systems implement a set of 1- and 2-qubit operations, akin to an instruction set. For 2-qubit operations, many machines only support hardware CNOT gates being performed between *adjacent* qubits, based on the topology. To perform CNOT gates between non-adjacent qubits, we should use SWAP operations between adjacent qubits until the two of interest for a given CNOT computation are in adjacent locations. Each SWAP operation between two adjacent qubits itself requires 3 CNOT gates. Our compiler aims to reduce the *time cost* of these operations. More importantly, each one of these operations incurs some error, so a key goal of our optimization is to reduce operation counts and error rates in order to increase the likelihood of an overall successful run. We refer to this as *reliability* and it is the primary design goal of this work.

In addition to compiler optimization based on attributes like gate counts, our approach also adapts based on publicly-available experimental data. In particular, the IBM Q machines are calibrated twice a day. Once a day there are public postings of experimental measurements of key properties: qubit relaxation time (T1), coherence time (T2), gate errors and readout errors [94]. From daily calibration logs, we observe that qubit coherence time is 70 microseconds on average, but varies up to 9.2x spatially and temporally across qubits and daily calibrations. The average error rate for CNOTs is 0.04, readouts is 0.07 and single qubit gates is 0.002. CNOT and readout error rates exhibit up to 9.0x and 5.9x variation across qubits and calibration cycles, respectively. CNOT gate durations vary up to 1.8 across qubits.

These error rates imply only very short programs can execute reliably on the machine. A program with more than 16 CNOT operations, has less than 50% chance of executing correctly. A key goal of our compiler optimizations is to use this calibration data to boost the success rate of individual program runs, by avoiding portions of the machine with poor coherence, operation, or readout errors.

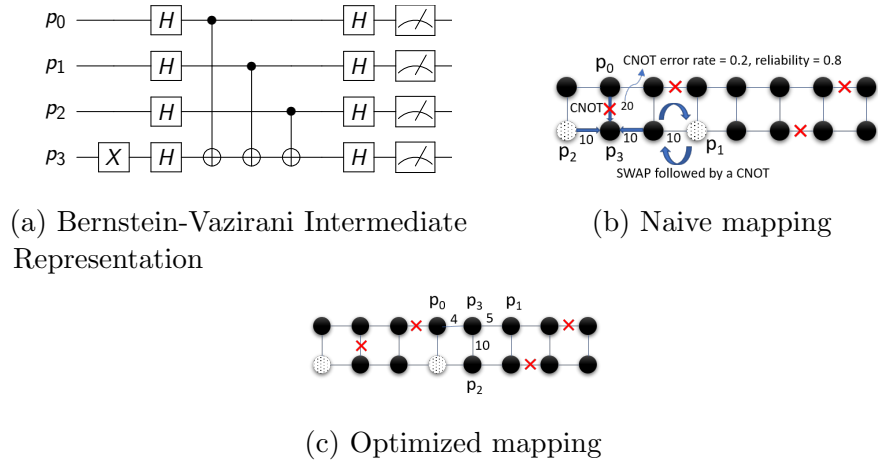


Figure 5.20: Figure (a) shows the intermediate representation of the Bernstein-Vazirani algorithm on 4 qubits (BV4). Each qubit is represented by a line. X and H are single qubit gates. The CNOT gates from each qubit $p_{0,1,2}$ to p_3 are marked by vertical connectors. The measurement or readout operation is indicated by the meter. Figure (b) shows a mapping where qubit movement is required. The numbers on the labelled edges indicate the CNOT gate error ($\times 10^{-2}$). In this mapping, an error-prone CNOT is used. Figure (c) shows an optimized mapping where qubit movement is not required and unreliable hardware CNOTs (crossed) and unreliable qubits (hatched) are avoided.

5.4.2 Overview of our Compilation Framework

Our framework takes a Scaffold program [7] as input, and produces compiled OpenQASM code [45]. The Scaffold quantum programming language extends C with quantum gates. Scaffold programs are independent of the machine topology, size and qubit properties. The ScaffCC compiler [100, 164] performs automatic gate and rotation decomposition, implements high level operations like the Toffoli gate and produces an LLVM Intermediate Representation (IR) [117] of the program. The IR version of the program includes the qubits required for each operation and the data dependencies between operations. For example, Figure 5.20a shows the IR for the simple 4-qubit Bernstein-Vazirani algorithm which is chosen because it fits on machines of this size and has an answer which can be calculated to check our results [21]. We use the program IR as a starting point for the noise-aware backend described here.

Starting from the LLVM IR, the noise-aware backend has three primary tasks. First, qubits in the program must be **mapped** to distinct qubits in the hardware implementation,

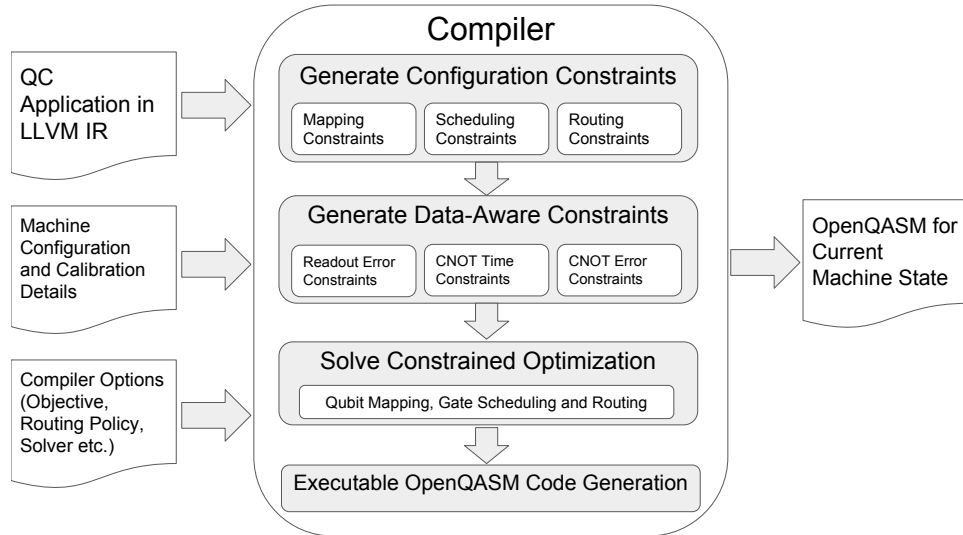


Figure 5.21: Optimization Pipeline. Inputs are a QC program, details about the specific hardware configuration, and a set of options, such as routing policy and solver approach. From these, compiler generates a set of appropriate constraints and uses them to map program qubits to hardware qubits and schedule operations. Finally, the compiler generates an executable version of the program, here for *IBMQ16*.

preferably in a way that reduces qubit state movement required as the program executes. Second, the compiler performs **operation scheduling** while respecting data dependencies between gates. To accomplish this, each operation is assigned a start time constraint, and the scheduler emits control code that enforces this. Third, to perform 2-qubit operations on non-adjacent qubits, the compiler should orchestrate **communication through SWAPs**. That is, it automatically inserts the required SWAP operations to bring the qubits adjacent to each other before the operation is performed.

Consider a simple compilation method where program qubits are assigned to random qubits on the hardware. Figure 5.20b shows such a mapping for the BV4 IR. In this mapping, the compiler must insert qubit movement or swap operations to perform the CNOT gates between p_1 and p_3 . In contrast, the mapping shown in Figure 5.20c requires no qubit movement because the qubits required for the CNOTs are adjacent. In addition, this mapping is noise-aware; namely, it uses the calibration data to select a mapping that avoids using qubits with low coherence time and gates with high error rates. Our compiler uses machine

Algorithm	Objective	Parameters	Constraints
Qiskit	Heuristic, minimize duration	-	-
T-SMT	SMT solver, minimize duration	Routing policy: RR, 1BP	1-4, 7-9
T-SMT★	SMT solver, minimize duration	Routing policy: RR, 1BP	1-3, 5-9
R-SMT★	SMT solver, maximize reliability	Routing policy: 1BP Readout weight $\omega \in [0, 1]$	1-3, 5-6, 9, 10-11
GreedyV★	Heuristic, maximize reliability	Routing Policy: Best Path	-
GreedyE★	Heuristic, maximize reliability	Routing Policy: Best Path	-

Table 5.7: List of compiler configurations used in our study. The IBM Qiskit 0.5.7 compiler is used as a the baseline. The use of calibration data is marked by a ★.

topology and calibration data to automatically generate such mappings for a given program.

Our primary goal is to maximize the likelihood that the program runs successfully. To accomplish this, we have three main strategies. First, the compiler places program qubits on hardware locations with high reliability, based on the calibration data. The compiler considers the effect of errors due to CNOTs and readouts; for this machine, single-qubit error rates are considerably smaller so our formulation chooses to ignore them. Second, to mitigate errors due to decoherence, the compiler should schedule all gates to finish before the coherence time of the hardware qubits (intuitively analogous to making use of data within the refresh interval of a DRAM). Third, the compiler optimizes for the qubit topology to avoid unnecessary qubit movement. Qubit movement not only increases execution duration, but more importantly leads to high error rates since each qubit SWAP operation includes three error-prone CNOTs. We have designed a set of optimal and heuristic compilation variants to accomplish these goals.

Table 5.7 enumerates the full set of compiler variants we consider in this paper. In addition to the publicly-available IBM Qiskit compiler we use as a comparative baseline, we also develop several approaches which are either truly optimization-based or heuristic. We give an overview of these approaches here, before offering details in the following section.

Optimization-Based Mappings

In the optimization-based variants of our compiler, we implement the above goals by posing the compilation problem as a constrained optimization problem to be solved by a satisfiability modulo theory (SMT) solver. The optimization problem has variables and constraints which express program information, machine topology constraints, and machine error information. The variables include program qubit locations, gate start times and routing paths. The constraints specify qubit mappings should be distinct, gates should start in the program dependency order, and routing paths should be non-overlapping. Fig. 5.21 summarizes the general compilation pipeline for the solver-based approach, beginning with an IR of a quantum program and resulting in execution-ready code.

The optimization objective is to maximize the reliability or success rate of program runs. We express the reliability of the program as the product of the reliability of all gates in the program. (Because of the degree of entanglement in QC programs, this serves as a useful measure of overall correctness.) For a given qubit mapping, the solver determines the reliability of each program CNOT, readout operation and single qubit gate. It then computes an overall reliability score for the mapping. For the optimization variants which are noise-aware, the solver can maximize the reliability score over all mappings by tracking and adapting to the error rates, coherence limits, and qubit movement based on program qubit locations.

Given a target machine, our framework converts the program IR into an optimization problem by expressing an objective and constraints that can be solved using an Satisfiability Modulo Theory (SMT) solver [48, 24]. For classical programs, these solvers have been used to obtain optimal hardware mapping and scheduling for spatial architectures [149], but to our knowledge, ours is the first use of them for QC systems. SMT solvers take as input a set of linear constraints, and an objective function and search for an optimal solution. Although the reliability objective is a product of individual gate reliability scores (and therefore non-linear),

we linearize the objective by instead optimizing for the additive logarithms of the reliability scores. An SMT solver can then be invoked to find a mapping which maximizes the log reliability.

Does maximizing the reliability score achieve our goal of increasing program success rate?

Optimizing for the reliability score induces the compiler to place qubits at locations where CNOT and readout errors are low. It also indirectly minimizes qubit movement because CNOTs between far away qubits are error-prone. For example, for the BV4 IR, consider mapping shown in Figure 5.20b. Here, the reliability of the CNOT between p_0 and p_3 is 0.8 (80% chance of executing correctly), while the reliability of the CNOT between p_1 and p_3 is only 0.65^5 . Thus, the compiler will choose mappings where communicating qubits are close together, minimizing unnecessary qubit movement and allowing gates to be scheduled to finish within the coherence window.

Heuristic Mappings

We also determine whether heuristic techniques can approach the optimization-based results, but with better scalability. For this, we develop two comparative algorithms based on greedy heuristics. The greedy heuristics analyze the CNOTs in the program IR, and determine a gate frequency for each qubit and program CNOT.

We explore two policies. In the first policy, GreedyV \star , we place program qubits on hardware qubits in the heaviest qubit first order. In the second policy, GreedyE \star , we place program CNOTs and their control and target qubits in a heaviest edge first order. Intuitively, the first policy aims to place qubits which use more CNOTs in locations which have good

5. p_1 has to swap once to move to a location adjacent to p_3 . The net reliability of the 3 CNOTs required to perform the SWAP is $0.9^3 = 0.729$. Then the actual CNOT operation can be performed with reliability 0.9. Hence, the overall CNOT reliability is 0.65.

CNOT and readout error rates. The second policy places pairs of qubits which have the most frequent CNOTs first.

5.4.3 *Optimal Compilation: Detailed Approach*

Notations and Assumptions

Let Q be the set of program qubits. Let H be the set of hardware qubits. In this work, we assume hardware qubits are arranged as a 2-D grid of dimensions $M_x \times M_y$. Likewise, due to the connectivity characteristics of *IBMQ16*, we assume only hardware qubits which are adjacent in the grid are permitted to participate in two qubit operations. More elaborate topology and routing assumptions can be handled in future work. For $q \in Q$, the ordered pair $(q.x, q.y)$ corresponds to the location of the hardware qubit assigned to the program qubit q . Let G be the set of operations in the program. This includes single-qubit gates such as H , and the 2-qubit *CNOT* gate and qubit measurement or *Readout* operations. *CNOT* and *Readout* operations dominate the reliability outcomes, so the reliability score focuses on them. The subset of *CNOT* gates is denoted by G_{CNOT} , and the subset of readout (qubit measurement) operations is $G_{Readout}$. For each gate g in the program, the start time is denoted by $(g.\tau)$, duration by $(g.\delta)$, and reliability by $(g.\epsilon)$. To denote data dependencies between the operations, we use a binary relation $>$ on the gates, so that for two operations $g_2 > g_1$ if g_2 depends on g_1 . Although the reliability objective focuses on a subset of operations, we map and schedule all operations (including single-qubit operations) to provide a valid real-system executable.

Constraints

Qubit Mapping Constraints

Constraint 5.1, guarantees all program qubits are mapped to actual hardware qubits. Constraint 5.2 guarantees each program qubit is assigned a unique location.

$$\forall q \in Q : 0 \leq q.x < M_x \wedge 0 \leq q.y < M_y \quad (5.1)$$

$$\forall q_1, q_2 \in Q : q_1.x \neq q_2.x \vee q_1.y \neq q_2.y \quad (5.2)$$

Gate Scheduling Constraints

For each gate g in the program, the compiler determines the start time and execution duration and emits classical control code to hold to this schedule. If two gates g_1 and g_2 both operate on the same qubit, and g_2 uses the output of g_1 , g_2 should start only after g_1 finishes. For every such edge in the dependency graph, Constraint 5.3 shows the form we use to enforce such data dependencies.

$$\forall g_1, g_2 \in G : g_2 > g_1 \Rightarrow g_2.\tau \geq g_1.\tau + g_1.\delta \quad (5.3)$$

The durations, δ , for single qubit operations are set using the documented durations in timeslots of the corresponding hardware operations. For CNOTs, the duration includes both the operation itself as well as the time to bring the relevant program qubit states into adjacent hardware qubits; this depends on the routing policy and is discussed below.

CNOT Duration based on Grid Distance

The duration of a CNOT gate accounts for both CNOT time and the duration of the swap paths before and after the CNOT. For a CNOT $g \in G_C$, let the control and target qubits be

q_c and q_t . Then the duration of the CNOT is: $g.\delta = 2 * (\|q_c - q_t\|_1 - 1) * \tau_{SWAP} + \tau_{CNOT}$ where $\|q_c - q_t\|_1 = |q_c.x - q_t.x| + |q_c.y - q_t.y|$ and τ_{SWAP}, τ_{CNOT} are the times to complete a SWAP or CNOT operation, respectively.

The compiler must schedule operations before the individual qubits decohere. For T-SMT (noise-unaware) we simply use an assumption of M_T as 1000 timeslots of coherence time, which is the long-term average for the machine:

$$\forall g \in G : g.\tau + g.\delta < M_T \quad (5.4)$$

CNOT Duration based on Calibration Data

For T-SMT \star and R-SMT \star , we set durations based on calibration data. In particular, since qubit coherence time changes daily (Figure 5.19a) and CNOT gate durations vary across qubits, these approaches use the calibration-based data in the optimization constraint. To set durations based on calibration data, we assume a routing policy and compute the CNOT durations for each hardware qubit pair. Let Δ be an $|H| \times |H|$ matrix where $\Delta_{h_i, h_j}, i \neq j$, specifies the duration of a CNOT between hardware qubits $h_i, h_j \in H$. The duration of a program CNOT can be set as: for all $g \in G_{CNOT}$ and for all $h_1, h_2 \in H$:

$$g_c = h_1 \wedge g_t = h_2 \Rightarrow g.\delta = \Delta_{h_1, h_2} \quad (5.5)$$

For the calibration-aware coherence time bound, constraint 5.6 ensures every gate finishes before the coherence time of the qubits it acts on i.e., if a gate uses a hardware qubit h , it should complete before h decoheres, with $h.\tau$ as the coherence time of a hardware qubit

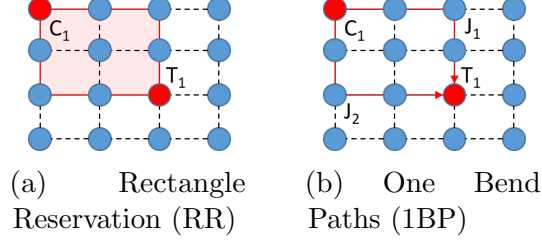


Figure 5.22: Two routing policies for swap-based architectures.

$h \in H$. We have for all $g \in G$ and for all $h_1, h_2 \in H$:

$$g_c = h_1 \wedge g_t = h_2 \Rightarrow g.\tau + g.\delta \leq \min(h_1.\tau, h_2.\tau) \quad (5.6)$$

Routing for CNOT Gates

To route multiple CNOTs in parallel, the compiler uses two routing policies based on policies in VLSI routing [75, 44].

Rectangle Reservation

In this policy, for every CNOT in the program, the compiler blocks a 2D region bounded by the control and target qubit, during the CNOT execution. For example, in Figure 5.22a, the highlighted rectangle is reserved for the duration of the CNOT.

Consider a CNOT gate $g_i \in G_{CNOT}$. Let (l_x^i, l_y^i) and (r_x^i, r_y^i) denote the top left and bottom right corners, respectively, of the bounding rectangle of g_i . These variables are defined using min and max relations on the qubit mapping variables of the CNOT. For two CNOTs g_i and g_j , the routing constraint is:

$$S(R_i, R_j) = \neg(l_x^i > r_x^j \vee r_x^i < l_x^j \vee l_y^i > r_y^j \vee r_y^i < l_y^j) \quad (5.7)$$

$$T(g_i, g_j) = \neg(g_i.\tau > g_j.\tau + g_j.\delta \vee g_j.\tau > g_i.\tau + g_i.\delta) \quad (5.8)$$

Constraint S checks if the two rectangles overlap in space. Constraint T checks whether CNOTs overlap in time. For any pair of CNOTs g_i and g_j , they cannot overlap in time if they overlap in space: $S(g_i, g_j) \implies \neg T(g_i, g_j)$.

One Bend Paths

In this policy, CNOT routes are restricted to the two paths along the bounding rectangle of the control and target qubit. For example, in Figure 5.22b, the CNOT is allowed to use one of the two highlighted paths. To implement this policy, the solver selects one of the two routes for every CNOT in the program.

To express constraints for this policy, we use variables to record the junction through which the CNOT is routed. The one bend path is composed of two segments: control to junction and junction to target. For generality, we can consider these segments as rectangles, and apply the same overlap check as in rectangle reservation. Denote the control to junction path for CNOT i as R_i^{cj} . Then, we can check if two CNOTs g_i and g_j overlap using:

$$\begin{aligned} \text{Overlap}(i, j) = & S(R_i^{cj}, R_j^{cj}) \vee S(R_i^{cj}, R_j^{jt}) \vee \\ & S(R_i^{jt}, R_j^{cj}) \vee S(R_i^{jt}, R_j^{jt}) \end{aligned} \quad (5.9)$$

Similar to rectangle reservation, we impose the condition that CNOTs do not overlap in time if they overlap in space.

Reliability Constraints

To optimize the reliability of program executions, we use a set of constraints to track the reliability scores of CNOT and readout operations in the program. Let $g.\epsilon$ denote the

reliability score for the operation g . For readout operations, we set the reliability as

$$\forall g \in G_{Readout} : \forall h \in H : g.q = h \Rightarrow g.\epsilon = E_h^R \quad (5.10)$$

where E_h^R is the reliability score for readout operations on hardware qubit h , and $G_R \subseteq G$ is the set of readout operations.

For CNOT gates, we set reliability tracking variables based on the routing policy. For each pair of hardware qubits, we pre-compute a matrix E^C , where E_{h_1, h_2}^C is the reliability of the CNOT gate between qubit h_i and h_j . This reliability factors in the reliability of the swap paths and the actual CNOT operation. Let $g.j$ be the junction for gate $g \in G_{CNOT}$. The constraints to track CNOT error are given for all $g \in G_{CNOT}$ and for all $h_1, h_2, h_j \in H$:

$$g.c = h_1 \wedge g.t = h_2 \wedge g.j = h_j \Rightarrow g.\epsilon = E_{h_1, h_2}^C \quad (5.11)$$

In our experiments, considering the error rates of single qubit gates such as H, X, Y etc. is not required for *IBMQ16*, because their error rates are much smaller than CNOTs and readouts. For systems where such errors matter, they can be easily incorporated into the optimization using similar constraints.

Optimal Compilation: Objective Function

The different optimization variants use different objective functions. For the time-oriented variants T-SMT and T-SMT \star , the objective function is based on the execution time for the program. Using the gate scheduling and duration constraints from before, the objective is to minimize the finish time of the last gate in the dependency order.

For the reliability-oriented variant, R-SMT \star , the objective function is based on the reliability of a program execution. We define the reliability of a program execution as the product of the reliability of each of its gates. Since single qubit gates have low error,

we define the reliability using CNOT and readout operations only. Ideally, the reliability objective would be the product across all gates of the readout and CNOT errors for the whole program: $\max \prod_{g \in G_{Readout} \cup G_{CNOT}} (g.\epsilon)$. Because the SMT solver requires linear operations, we convert this to an additive linear objective function by considering the logarithm of the operation reliabilities, instead of their product. Finally, to allow for different emphases on readout error versus CNOT error, we convert the above objective into a weighted objective using a weight ω which is applied to the readout error rates:

$$\omega \sum_{g \in G_{Readout}} \log(g.\epsilon) + (1 - \omega) \sum_{g \in G_{CNOT}} \log(g.\epsilon). \quad (5.12)$$

We use this objective to study the relative importance of CNOT and readout error rates.

Optimizing reliability places qubits at hardware locations with high CNOT and readout reliability. It indirectly optimizes qubit movement because CNOT gates between non-adjacent qubits have low reliability. This objective is used by R-SMT[★] in our experiments. To compute a qubit mapping and gate schedule which maximizes this objective, we set up an optimization problem using this along with the mapping and scheduling constraints, gate durations using calibration data, routing approaches, and reliability constraints discussed before. The reliability constraints make the $g.\epsilon$ variables dependent on the qubit mapping variables.

5.4.4 *Heuristic Compilation*

Where tractable, the SMT-based compilation approach offers the best chance at successful application runs on real hardware. However, effective heuristic approaches may offer similar reliability but scale better to future NISQ systems with hundreds of qubits. Here we propose and evaluate heuristic mapping/scheduling alternatives as comparators to the optimization-based approaches.

Our heuristic techniques are also based on a program graph constructed from the program IR. The program graph has a node for every qubit, and an edge between every pair of qubits which is involved in a CNOT. For example, the program graph of BV4 has 4 nodes for $p_{0,1,2,3}$ and 3 edges, one from each of $p_{0,1,2}$ to p_3 . For each heuristic, we first compute the most reliable path between every pair of hardware qubits using Dijkstra’s algorithm, where edge weights are given as the negative log of the CNOT errors from the calibration data. For both heuristics, once we map the qubits, we schedule gates using an earliest ready gate first policy [85] and route based on the precomputed paths.

Greatest Vertex Degree First

The GreedyV \star heuristic seeks to minimize communication distance (and therefore reduce the number of error-prone SWAP operations) by considering qubits in descending order of degree. The degree of the qubit is the number of CNOTs in which the qubit is used. First, place the highest degree program qubit at the hardware location which has highest readout reliability among high degree hardware qubits. Next, for each program qubit which shares a CNOT with an already placed qubit, place this qubit in order to maximize the total reliability of paths between it and each of its placed neighbors, where the total reliability is given by the sum of the path lengths computed from before between it and its neighbors.

Greatest Weighted Edge First

In GreedyE \star , we map edges in the descending order of weight. The weight of an edge between two nodes is the number of times a CNOT gate is invoked between them. Therefore, placing edges with high weight first allows qubits which interact highly to be close together. Such placement reduces qubit movement and increases reliability. The algorithm starts by placing the highest weighted edge at on hardware location with maximum CNOT and readout reliability. Next, for each edge which has one mapped one unmapped endpoint, we map the









Benchmark	Qubits	Gates	CNOTs	CNOT Graph	Benchmark	Qubits	Gates	CNOTs	CNOT Graph
BV4	4	12	3		Fredkin	3	19	8	
BV6	6	12	3		Or	3	17	6	
BV8	8	18	3		Peres	3	16	5	
HS2	2	16	2		Toffoli	3	18	6	
HS4	4	28	4		Adder	4	23	10	
HS6	6	42	6		QFT	2	13	5	

Table 5.8: Characteristics of benchmark programs.

unmapped qubit to the position which maximizes the total reliability of CNOTs with already mapped qubits, where the total reliability is given by the sum of the path lengths computed from before between it and its neighbors. The process is repeated for each unmapped edge in weight order.

5.4.5 Experimental Setup

Benchmarks

Table 5.8 lists 12 quantum programs derived from prior work on compilation and system benchmarking [9, 122, 173]. These benchmarks include the Bernstein-Vazirani algorithm [21], Hidden Shift Algorithm [39], Quantum Fourier Transform [147], a one bit adder and important quantum kernels such as the Toffoli gate [128]. We used or created Scaffold programs for each benchmark and obtained LLVM IR using the Scaffold compiler [100]. To be runnable on real-system QC hardware, the benchmarks must be relatively small in qubit counts and short in execution time steps. Nonetheless, our ability to show order-of-magnitude improvements in success rate for these programs is a promising indicator of the value of such compilation techniques for future larger systems and programs. Furthermore, several of these programs—such as QFT and Toffoli—are important kernels for larger programs.

Beyond these, to study scalability trends across different qubit and gate counts, we generate a synthetic benchmark where we can specify the number of qubits and gates and from this, we experiment with randomly generated quantum programs with 4-128 qubits and 128-2048 gates. We generate these circuits by uniformly sampling gates from the universal gate set of H,X,Y,Z,S,S[†], T,T[†], CNOT.

Compiler Configurations

In order to study various compilation schemes, our framework includes various options for the solver, routing policy, use of calibration data and other parameters. We evaluate these options one factor at a time using the set of configurations listed in Table 5.7.

Experimental Setup

Our compilation experiments use an Intel Skylake processor (2.6GHz, 12GB RAM) using Python3.5 and gcc version 5.4. Our optimization approach uses the Z3 SMT solver [48]. To perform experiments on *IBMQ16*, we use the IBM Quantum Experience APIs [93, 94]. The daily machine calibration data is available through the Quantum Experience APIs. The calibration data includes time data such as single qubit gate time, qubit coherence time (T2 time), durations for CNOT gates, and error rates such as single qubit gate error, CNOT gate error, and read out (measurement) error. We use IBM’s Qiskit compiler/mapper as our baseline for comparison, version 0.5.7.

Metrics

Before each run, we obtain the latest calibration data, and recompile the benchmark. We execute each benchmark on *IBMQ16*, using 8192 trials in each run. We measure the success rate as the fraction of trials which gave the correct answer. For example, success rate of 0.6 means the execution produced the correct answer in 60% of the trials. The ideal success rate

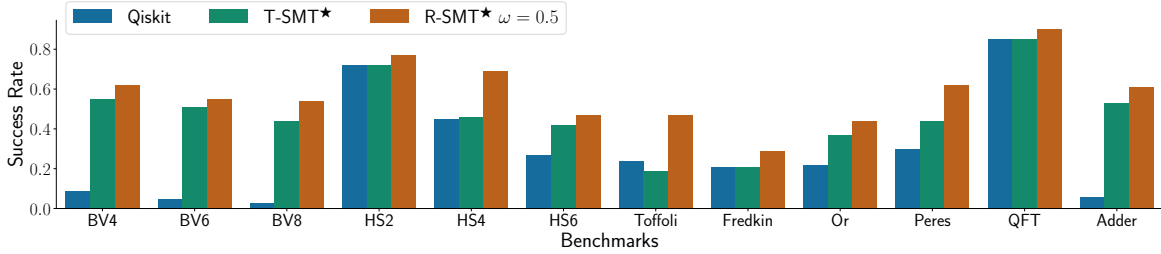


Figure 5.23: Measured success rate of R-SMT \star compared to Qiskit and T-SMT \star . (Of 8192 trials per execution, success rate is the percentage that achieve the correct answer in real-system execution. R-SMT \star obtains higher success rate than Qiskit because it simultaneously adapts placement according to dynamic error rates and avoids unnecessary qubit movement.

is 1, where all trials succeed. Results within a single graph are performed closely in time so are comparable. Results from different graphs may not be comparable because the machine error characteristics can be different across runs. We also study quantum execution time and compilation time. Because timing granularity is so coarse, execution time is estimated using real gate duration data from the *IBMQ16* system. We report durations in terms of timeslots on *IBMQ16*, where each timeslot is 80ns.

5.4.6 Optimizing Execution Reliability

Baseline Comparison to IBM Qiskit

Here we compare the success rate of program runs from our compiler versus the IBM Qiskit compiler for real-system runs on *IBMQ16*. Figure 5.23 shows the success rate of the IBM Qiskit compiler, T-SMT \star and R-SMT \star with $\omega = 0.5$ on all the benchmarks. In all benchmarks, R-SMT \star has higher success rate than Qiskit, indicating that its reliability-oriented objective function is effective. In fact, R-SMT \star obtains geomean 2.9x improvement over Qiskit, with up to 18x gain on BV8. For BV8, the compiled code produced by Qiskit uses 15 CNOT operations to move qubits, while R-SMT \star obtains mappings which require no qubit movement. Each extra CNOT gate increases both the error rate and the execution duration of the code and leads to poor success rate. Benchmarks which require no qubit movement

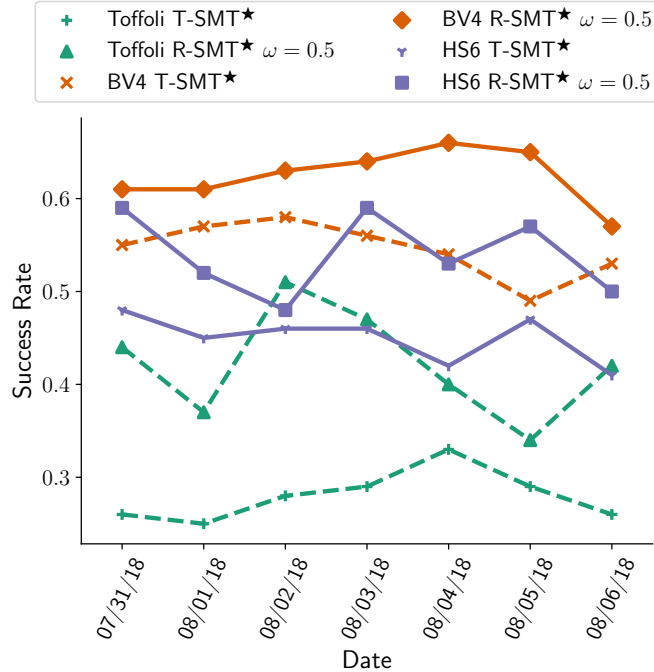


Figure 5.24: Executions of three benchmarks for 1 week. R-SMT \star is more resilient to errors compared to T-SMT \star . Similar trends for other benchmarks.

such as BV, HS, QFT and Adder have higher reliability than benchmarks such as Toffoli, Fredkin, Or, and Peres, which require at least one qubit swap.

In all benchmarks, R-SMT \star outperforms T-SMT \star , even though they use the same number of qubit movement operations. Hence, while optimizing qubit communication is important, it is essential to optimize for gate error rates to improve success rate. In fact, in our experiments, when the machine state has high variability, R-SMT \star can obtain up to 9.2x improvement in success rate over T-SMT \star (see Figure 5.25).

Resilience to Daily Variations

Since IBM limits the executions researchers may perform per day, we perform detailed experiments on three benchmarks, BV4, HS6 and Toffoli. These benchmarks are chosen as examples of different CNOT patterns (see Table 5.8). Figure 5.24 compares the success rate of R-SMT \star and T-SMT \star over a week for the three benchmarks. The success rate of the

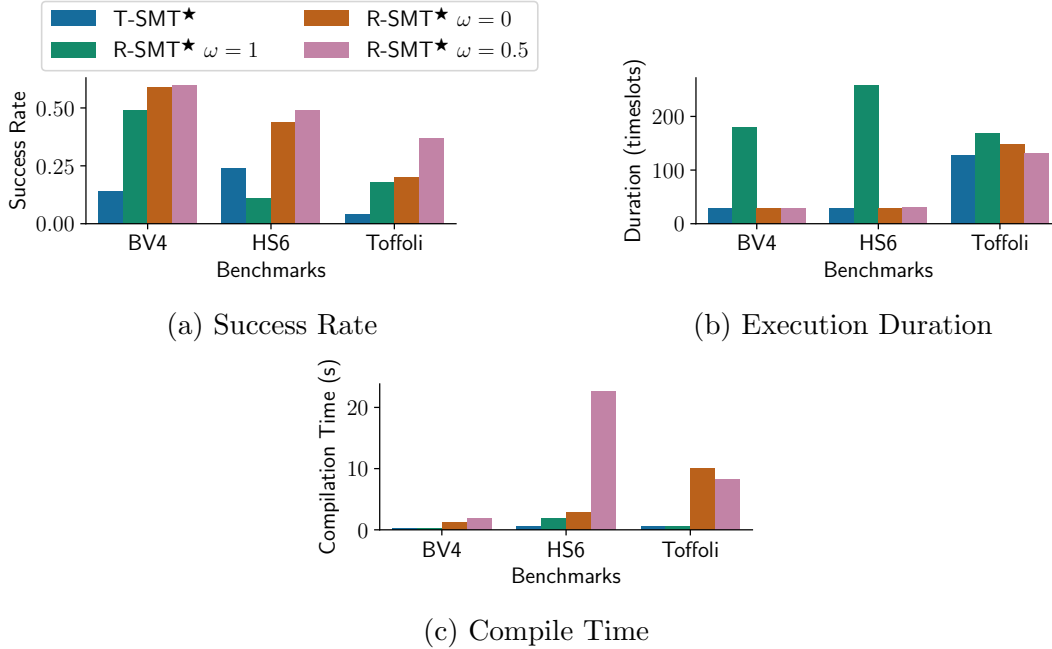


Figure 5.25: Measured success rate, execution duration and compile time for three representative benchmarks. T-SMT \star which directly optimizes for execution duration obtains the minimum execution durations, but R-SMT \star with $\omega = 0.5$ is close, and more resilient to errors (higher reliability). All benchmarks compile in less than 1 minute.

programs change every day because error rates of the hardware CNOT and readout units change daily. (We recompile each day before running.) For all three benchmarks, R-SMT \star is more resilient to error than T-SMT \star , since it adapts the qubit mappings to account for daily variations in operation error rates. Since T-SMT \star compiles based on static information (qubit topology and gate duration), it uses the same qubits and hardware gates every day, irrespective of their dynamic error characteristics.

Choice of Optimization Objective

Figure 5.25 compares R-SMT \star with $\omega = \{0, 0.5, 1\}$ and T-SMT \star on the three benchmarks. R-SMT \star with $\omega = 0.5$ achieves the highest success rate among the methods, with up to 9.25x gain over T-SMT \star . For BV4, we illustrate the mappings obtained by these methods in Figure 5.26. T-SMT \star obtains a mapping which requires no qubit movement, but it uses

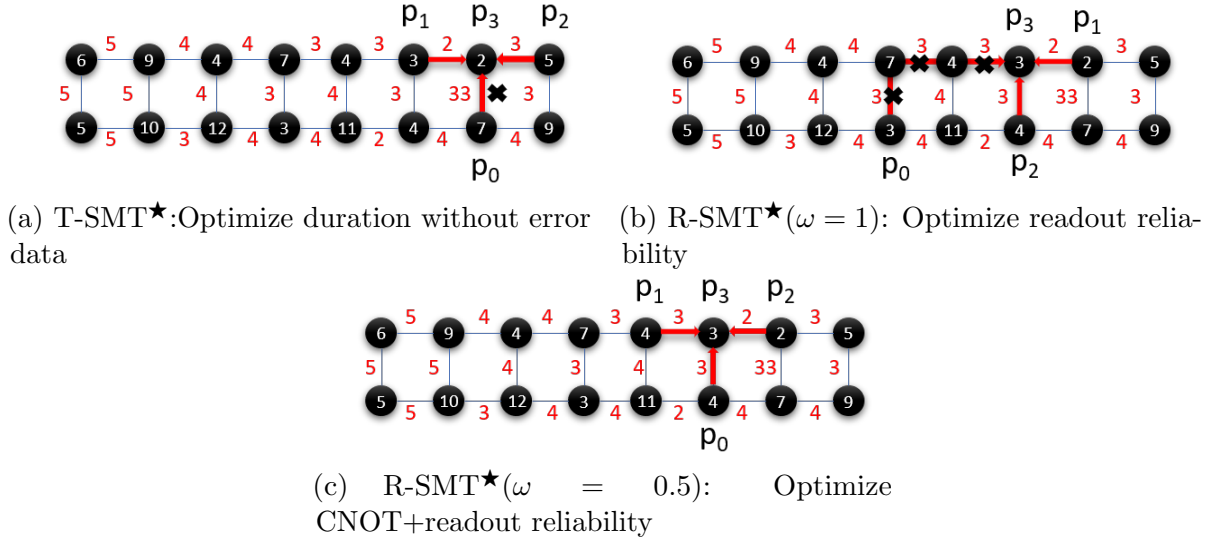


Figure 5.26: For real data/experiment, on *IBMQ16*, qubit mappings for three optimization objectives, varying the type of noise-awareness. In each figure, the edge labels indicate the CNOT gate error rate ($\times 10^{-2}$), and the numbers inside each node indicate that qubit’s readout error rate ($\times 10^{-2}$). (a), T-SMT \star uses an unreliable hardware CNOT between p_3 and p_0 . (b) Program qubits are placed on the best readout qubits, but p_0 and p_3 communicate using swaps. (c) Best CNOTs and readout qubits are used.

a hardware CNOT with very high error rate. With $\omega = 1$, R-SMT \star optimizes only for readouts and uses long swap paths which reduce success rate. With $\omega = 0.5$, R-SMT \star maps qubits to simultaneously optimize CNOT gate error, readout error and qubit movement.

R-SMT \star with $\omega = 0.5$ also achieves near-optimal execution durations, comparable to T-SMT \star , which directly optimizes for duration. From the perspective of compilation time, optimizing for reliability is harder than optimizing execution duration. However, each method finds optimal mappings in under a minute, for each benchmarks.

R-SMT \star was executed with $\omega \in [0, 1]$ to determine the relative importance of optimizing for read out error and CNOT error. In general, choosing an ω roughly near 0.5 is appropriate to obtain good success rates. On the *IBMQ16* machine, readout and CNOT error rates are fairly balanced, and hence we see that an equal weighted combination of both is suitable for optimization.

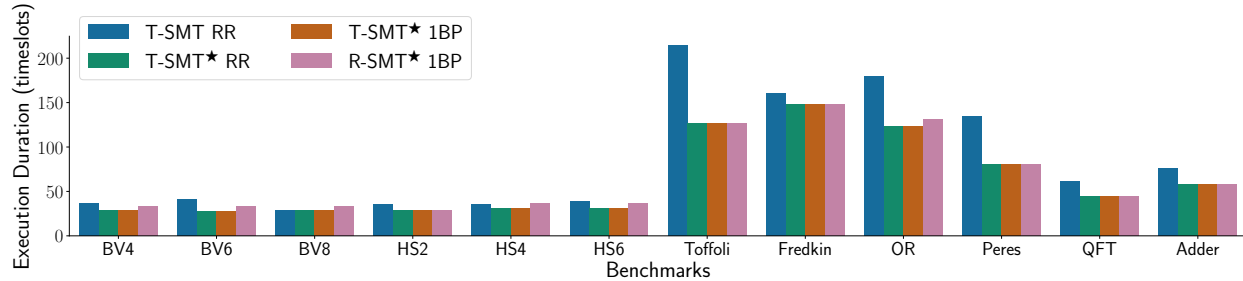


Figure 5.27: Effect of gate durations, routing policy and objective function on execution duration. Although reliability is our primary objective, several variants perform well on run time as well. T-SMT★(either RR or OBP) has the best execution duration, but R-SMT★ is very close in run time and offers better success rates. Noise-aware policies, R-SMT★ and T-SMT★, are 1.6x better than T-SMT.

Sensitivity to Gate Durations and Coherence Time

We test whether the use of real gate time data significantly affects the execution duration of NISQ benchmarks. Our compiler is run on three settings: T-SMT(RR) which assumes all hardware CNOTs have the same gate duration and T-SMT★ (RR) and R-SMT★ (1BP) which use real gate durations. We restrict R-SMT★ to the 1BP policy to reduce the number of experimental configurations; we show in Section 5.4.6 the choice of routing policy doesn’t affect execution duration for NISQ benchmarks.

Gate Durations

Figure 5.27 shows execution duration, computed using the gate time data, for the three methods. Considering real gate durations can improve the execution duration for each benchmark, with up to 1.68x gain on Toffoli. Considering real durations increases the number of constraints in the optimization problem and increases the compilation time by up to 3x (not shown). However, even with real durations, each benchmark requires only a few seconds of compilation time.

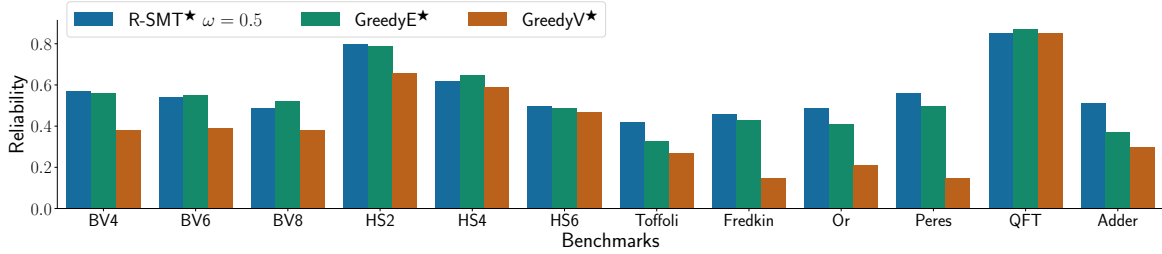


Figure 5.28: Noise-aware Heuristics: GreedyE* heuristic mapping offers reliability comparable to R-SMT* on most benchmarks.

Coherence Time

Each benchmark finishes in less than 150 timeslots using the R-SMT* method. Since the coherence time of the worst qubit on the machine is more than 300 timeslots, considering fine grained variations in coherence time is not necessary for our benchmarks.

Effect of Routing Policy

Figure 5.27 compares the execution duration and compilation time of T-SMT* with two routing policies (RR and 1BP) and R-SMT* (1BP). The three policies produce executables with similar execution duration since NISQ benchmarks are small, and have only few parallel CNOTs. Hence, most CNOTs execute without swapping or blocking qubits. Although R-SMT* optimizes reliability, it obtains execution durations close to T-SMT* on all benchmarks.

Success Rate and Scalability of Heuristics

In this section, we compare the success rate of heuristics to the optimal methods and evaluate the scalability of all methods.

Figure 5.28 compares the success rate of the heuristics and R-SMT*. Greedy methods are comparable to R-SMT* in success rate and in some cases, they outperform R-SMT* marginally because $\omega = 0.5$ may not be the optimal value for every benchmark and machine

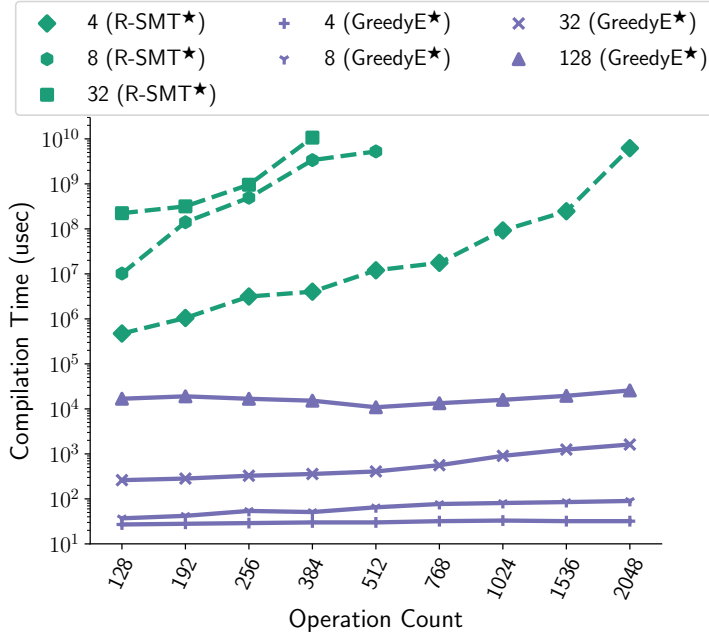


Figure 5.29: Scalability of optimal and heuristic methods on synthetic benchmarks. Each line represents a qubit count.

state. GreedyE[★] is as successful as R-SMT[★] in all cases. Our study reveals the edge based heuristic GreedyE[★], is more successful than the vertex based heuristic GreedyV[★]. Considering edges instead of vertices allows the heuristic to prioritize the reliability of the most frequent CNOTs.

To study the scalability of optimal and heuristic methods, we used a benchmark of randomly generated quantum programs. Figure 5.29 shows the compilation time on the benchmark. R-SMT[★] requires up to 3 hours to compile a program with 32 qubits and 384 gates. On the other hand, the greedy methods compile programs in under one second in all cases.

5.4.7 Remarks

This paper proposed and evaluated calibration-aware compiler techniques for NISQ systems. We considered optimal and heuristic compilation methods, the use of calibration data, different

objective functions and routing policies. Our evaluations show it is crucial to adapt quantum program compilation to dynamic operation error characteristics of the machine. It is most important to consider CNOT and readout error rates, since these operations are more noisy than single qubit gates. Optimization based on qubit coherence time is also useful, but less critical here because gate errors severely limit useful computation time. Our compiler, which optimizes program placement using these choices, obtained up to 2.9X geometric and up to 18X improvement in reliability over the widely-used IBM Qiskit compiler. We plan to open-source our code and facilitate its integration into Qiskit releases for broader use. Although we demonstrate our real-system results on a machine with 16 superconducting qubits, our approaches can easily support other NISQ systems and technologies. These include fully-connected trapped-ion QCs, of which a 5-qubit prototype is available [49]. We can also support other routing approaches such as teleportation-based communication [41], by choosing the appropriate technology-specific durations in the CNOT constraints. Our compiler can be extended to include local, qubit-specific optimizations to improve reliability: for example, if a hardware qubit has poor readout reliability, and a neighboring qubit has a reliable readout unit, a swap operation can be used to transport and then output the hardware qubit reliably. Such resource multiplexing may be important for executing large quantum programs that use almost all of the machine qubits.

This section's results offer important insights on QC based on real-system measurements. In particular, our work shows the importance of initial qubit placement, namely benchmarks which require more qubit movement are hard to reliably execute on systems with 2D grid topologies. Our results show proper placement could result in over 10X improvements in run success rate. Mapping and scheduling based on calibration data such as gate error rates and coherence times offer further benefits. Ultimately the best-performing approach offered up to 18X improvement (2.9X average) in success rate and up to 6X (2.7X average) improvement in runtime over the current IBM Qiskit baseline. Our results also give insights to future system

designers. Developing richer qubit topologies, such as the diagonal CNOT connections in the 50-qubit system announced by IBM [95] will reduce the need for SWAP operations and will be helpful in reliably executing important quantum primitives such as the Toffoli gate.

This research has shown that SMT approaches are very effective for current and near-term systems, but may not scale well to the far-NISQ machines of 500 qubits or more. For those, we have developed heuristic approaches, GreedyV[★] and GreedyE[★], which offer nearly as good results but with much more tractable compile times.

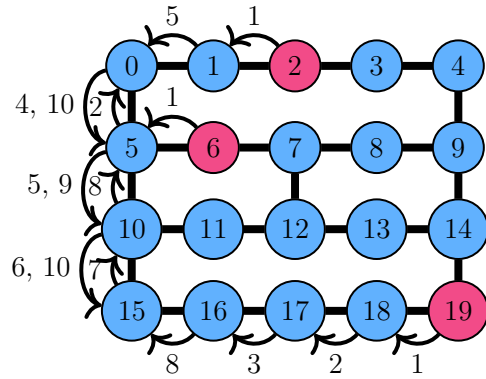
Overall, given the challenges of building reliable and scalable QC hardware, the key for the next five years or more will lie in ultra-efficient use of the resources available in NISQ systems. Our tool offers important leverage in stewarding runtime resource usage and optimizing reliability.

5.5 Orchestrated Trios: Compiling for Efficient Communication in Quantum Programs with 3-Qubit Gates

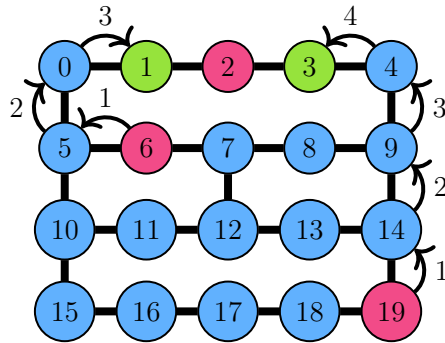
6

As we have already seen, quantum program compilation involves many passes of transformations and optimizations similar in many ways to classical compilers. Some optimizations occur at the abstract circuit level, independent of the underlying hardware, such as gate cancellation [144]. One of the first steps usually taken is to convert an input program into a gate set (ISA) supported by the target hardware. For example, on IBM devices, gates are typically rewritten using only gates in the set $\{u1, u2, u3, cx\}$ [93] (single-qubit gates and the common CNOT gate described later). One critical limitation of many current available architectures is the inability to execute more complex multi-qubit operations, like the Toffoli, directly; instead, these gates must be decomposed into the supported one- and two-qubit gates.

6. JMB's contributions in this section include benchmarking, routing pass design, and experimentation (with support from AL). Original conceptualization and split circuit decomposition due to CD.



(a) Expensive Qiskit routing



(b) Efficient Trios routing

Figure 5.30: Example routing from Qiskit (a) vs. Trios (b) for a single Toffoli operation. Circles represent qubits and lines indicate two qubits are connected. Input qubits are highlighted in red. SWAP arrows are labeled by timestep. The routed locations for Trios routing are highlighted in green while Qiskit moves them several times. Qiskit adds 16 SWAPs (=48 CNOTs), some during the Toffoli, while Trios adds only 7 SWAPs (=21 CNOTs) all before the Toffoli. Performing multiple passes of decomposition allows direct routing and enables this huge reduction in communication, increasing the probability of program success.

Furthermore, many current superconducting architectures only support two qubit operations on adjacent hardware qubits wired together with a coupler. This requires the insertion of additional operations called SWAPs to move the data onto adjacent (and connected) qubits.

The process of transforming an optimized and decomposed program to the desired target is typically broken down into three distinct steps: decomposing the program into basic gates, mapping the logical qubits of a program to hardware qubits and routing interacting qubits so that they are adjacent on hardware when they interact, and scheduling operations in order to minimize total program run time (depth) or to minimize errors due to crosstalk [139]. Each of these steps is critical to the success of the input program. A well-mapped and well-routed program will reduce the total number of communication operations added and subsequently reduce the compiled program's depth, both of which will increase the chance of success. Conventionally, these three steps occurs sequentially. By doing so, current strategies are unable to account for structure in the input program, resulting in inefficient routing of qubits. An optimal compiler could find the best routing despite the lack of structure but at the cost of much slower compilation. Consider the SWAP paths inserted by IBM's Qiskit compiler for a single Toffoli compiled to IBM's Johannesburg device in Figure 5.30a. This baseline strategy adds a large number of unnecessary SWAPs as it individually routes each CNOT composing the Toffoli, dramatically reducing the probability of successful execution.

The approach of this section, Orchestrated Trios (Trios) decomposes and routes qubits in multiple stages, as seen in Figure 5.31b. For example, first decompose an input program to one- two-, *and* three-qubit gates (e.g. do not decompose Toffoli gates) and route as before except for three-qubits, route all three to a common location with minimal SWAPs. This new program can then undergo a second round of decomposition to produce a circuit containing only hardware permitted one- and two-qubit gates. The second round may use the now known mapping (locations of data qubits on the device) to generate fine-tuned decompositions for the architecture.

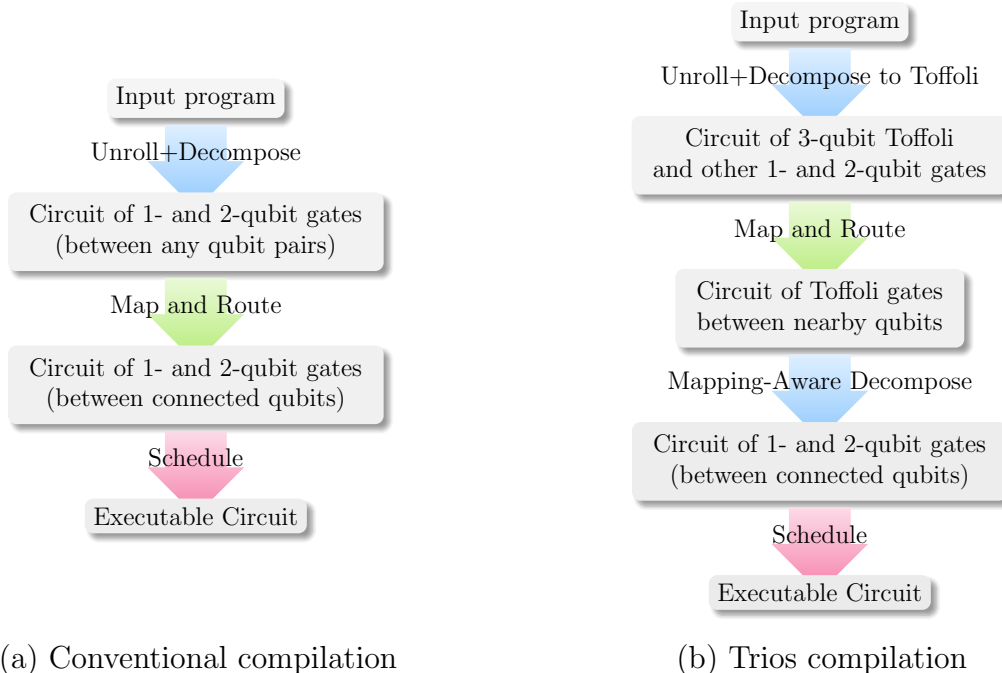


Figure 5.31: (a) Typical compilation passes used by Qiskit (simplified). (b) Trios compilation passes.

This layered approach has a major advantage over current routing techniques: we are better able to capture program structure by inspecting intermediate complex operations for routing. This better informs how qubits should be moved around the device during program execution. In Figure 5.30, the Trios strategy reduces the total number of SWAPs added to 21: fewer than half compared to Qiskit. This was an extreme example we selected to present the issue, not an average case.

We specifically propose a two-pass approach to circuit decomposition. We will focus on superconducting hardware systems like IBM’s cloud accessible devices, but our strategy can easily be adapted to other systems. An overview of our compilation structure is found in Figure 5.31b. This strategy has a substantial benefit on the overall success rate of programs. We demonstrate these improvements by executing Toffoli gates on a real IBM quantum computer and estimating success probability of a suite of benchmarks via simulation.

This section focuses its discussion on the following:

- A new compiler structure, Trios, with two passes for decomposition with a modified routing pass in between which greatly improves qubit routing.
- A simple method for architecture-tuned Toffoli decompositions during the second decompose pass that allows for a new kind of location-aware optimization.
- On Toffoli-only experiments, Trios reduces the total number of gates by 35% geomean (geometric mean) resulting in 23% geomean increase in success rate when run on real IBM hardware as compared to Qiskit.
- On near-term algorithms shown in Figure 5.40 (4 to 20 qubit benchmarks), Trios reduces total gate count by 37% geomean resulting in 344% geomean increase in (or 4.44x) simulated success rate on IBM Johannesburg with noise rates of near-future hardware as compared to programs compiled without Trios. A sensitivity analysis over four architecture types shows the benefit range from 133% to 3020% increase in success rate.

5.5.1 *Relevant Background*

Quantum Circuits in the Context of Trios

As usual, complex instructions must be decomposed into multiple simpler, supported operations. For example, many quantum algorithms and subroutines make use of the Toffoli gate, a three-input gate which performs the logical AND between two controls bits and writes the output onto the target bit. This gate cannot be executed directly on available hardware and instead is decomposed into an equivalent sequence of one- and two-qubit operations. Two such popular decompositions are given in Figures 5.32, 5.33. There are two key distinctions in these decompositions illustrating a more general trade off. The first [147] is the most popular decomposition using only 6 CNOT gates but requires CNOTs between all three pairs

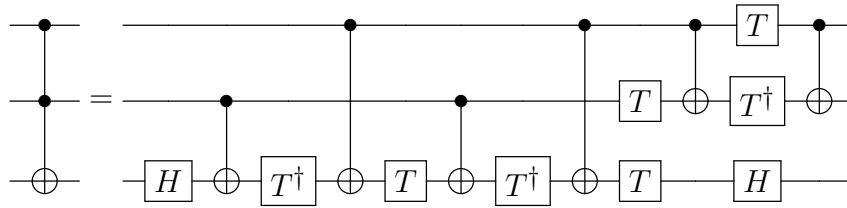


Figure 5.32: A 6-CNOT decomposition of the Toffoli gate.

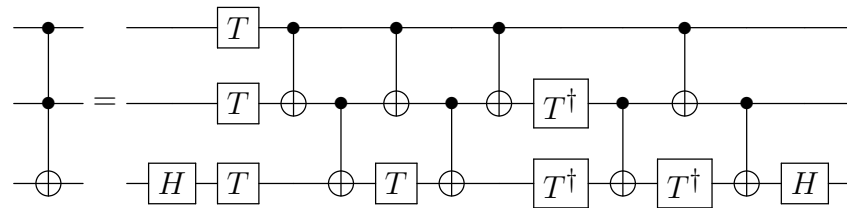
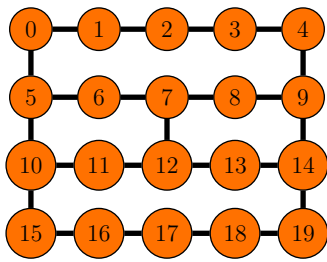


Figure 5.33: An 8-CNOT decomposition of the Toffoli gate.

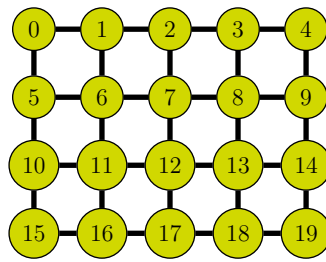
of qubits. This would require inserted SWAPs or a device connectivity containing a triangle. The second [167] uses a total of 8 CNOT gates and requires all three inputs be only linearly connected (only two of the three qubit pairs are required to be connected). While the first is apparently more efficient, this is not true if the connectivity of the underlying hardware does not directly support it. It is more efficient to use the 8-CNOT version than use the 6-CNOT version with added SWAPs.

Current Quantum Devices

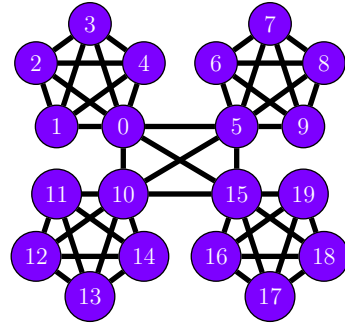
In this paper we focus primarily on currently available superconducting quantum devices. This type of hardware is the primary focus of many industry players like IBM, Rigetti, and Google [171, 93, 3]. We show some representative topologies for superconducting devices in Figure 5.34abd. For completeness, we include a clustered device shown in Figure 5.34c representative of a QCCD ion trap device such as [133]. These systems exhibit all of the properties previously discussed. They have a small universal supported gate set which all programs must be transformed into and only support local two-qubit operations. The connectivity of these devices is given as a *coupling graph* specifying which pairs of qubits can



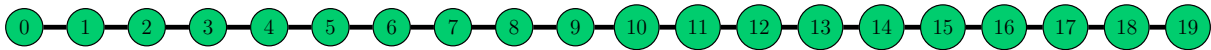
(a) IBM Johannesburg



(b) 2D Grid



(c) 4, 5 qubit fully connected clusters



(d) Linear

Figure 5.34: Example topologies of near-term quantum devices. Orange (a): IBM Johannesburg. Yellow (b): 2D Grid. Purple (c): four groups of five fully connected clusters. Green (d) Linear. Our real experiments run on Johannesburg and our simulations explore all of these topologies. Colors correspond with the bars in Figures 5.38, 5.39, 5.40.

execute CNOTs.

The Baseline Compilation Scheme

IBM's Qiskit compiler, the standard for compiling programs to execute on an IBM device, has a default sequence of passes. First, all high level optimization and analysis passes are performed and all gates are unrolled and decomposed to the target gate set. Then single passes of mapping, routing, and scheduling are performed [8].

Evaluation Metrics

As in the previous sections, our primary metric is program success rate, the fraction of circuit executions that result in the correct output. Others use fidelity which can stand-in for success rate when evaluating sub-circuits where the output is not measured. When executing a quantum algorithm, the corresponding quantum circuit is typically executed thousands of

times to gather output statistics or identify the error-free result.

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

Simulation, Reiterated

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

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

5.5.2 *Orchestrated Trios*

In this section we describe our proposed compilation structure compared to the conventional one as outlined in Figure 5.31. Specifically, we focus on improving the routing and

decomposition stages of compilation. Previously, we identified a key problem in current methods: decomposing the program to one- and two-qubit gates up front hinders the ability of heuristic-based compilers to effectively minimize the communication cost, i.e. the number of SWAPs added, and eliminates the possibility of location-aware decompositions.

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

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

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

Finally, the second decomposition pass is run. This is different from normal decomposition

as there are only Toffoli gates to decompose and they are already mapped to neighboring qubits. We could use the default 6-CNOT decomposition and still get the above benefit of improved routing but now that we have more information, this can be exploited to further reduce SWAPs due to a mismatch between the decomposition and the hardware connectivity. If all three pairs of qubits are connected, then the 6-CNOT Toffoli of Figure 5.32 is best, otherwise use the 8-CNOT Toffoli of Figure 5.33, ensuring the middle qubit is used for the middle of the decomposition (Any of the three qubits can be the target by simply moving the two H gates to that qubit).

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

This method can be easily extended to be noise-aware like previous work [136, 178] by using a noise-aware mapper with the simple modification described earlier where the path-finding graph has weighed edges with the $-\log$ value of the CNOT success rate. The path distance represents the $-\log$ probability of success of that particular path where lower values indicate a higher success rate and the shortest path can be found just as before and the routing steps are unchanged. Any routing strategy designed for one and two-qubit gates can be modified to work for one, two, and three-qubit gates and used as the first routing step of Trios.

In programs where there are no three qubit gates as in the typical NISQ benchmark, Bernstein-Vazirani [21], which is specified directly as CNOT gates, our strategy will have no effect. Many benchmarks, however, are written using Toffoli gates because they are the quantum analog the AND gate ubiquitous in arithmetics and other common subroutines.

Trios can naturally be extended to any multi-qubit operation of three or more qubits

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

5.5.3 Evaluation

Toffoli Only Circuits

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

We study these circuits on a real IBM device, namely IBM Johannesburg, a 20-qubit device with limited connectivity in Figure 5.34a. We use the default Qiskit compiler which decomposes the Toffoli gates before doing shortest path routing compared to our proposed method where we do shortest path routing first and then decompose the Toffoli. We study the use of two different Toffoli implementations, a 6 CNOT decomposition with full qubit connectivity and an 8 CNOT decomposition with linear qubit connectivity.

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

NISQ Benchmarks and Quantum Subroutines

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

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

A summary of our benchmarks is found in Table 5.9 using implementations found in [11]. The last three benchmarks use no Toffoli gates where we expect advantage only for circuits containing Toffoli gates. For BV, we assume the all 1 bit string oracle. The different CnX (many-controlled-NOT) benchmarks use various numbers of ancilla.

As noted previously, the connectivity of the underlying hardware has a significant impact on the number of required SWAPs. For example, on a completely connected set of qubits, no SWAPs are ever needed. In architectures with greater connectivity, we may opt for a more efficient Toffoli decomposition using 6 CNOTs. With simulation we study the effect of

7. The total number of CNOT gates is after decomposition with the 8-CNOT Toffoli but does not include any SWAPs for routing.

Table 5.9: Details about benchmarks for Trios for reference, both NISQ programs and other quantum subroutines

Benchmark	Qubits	Toffolis	CNOTs ⁷
<code>cnx_dirty</code> [12]	11	16	128
<code>cnx_halfborrowed</code> [70]	19	32	256
<code>cnx_logancilla</code> [15]	19	17	136
<code>cnx_inplace</code> [70]	4	54	490
<code>cuccaro_adder</code> [46]	20	18	190
<code>takahashi_adder</code> [176]	20	18	188
<code>incrementer_borrowedbit</code> [70]	5	50	448
<code>grovers</code> [78]	9	84	672
<code>qft_adder</code> [162]	16	0	92
<code>bv</code> [21]	20	0	19
<code>qaoa_complete</code> [64]	10	0	90

connectivity on the overall expected success rates and gate counts. We study four different connectivity models, all shown in Figure 5.34, each with 20 qubits, the topology of IBM’s Johannesburg device containing four connected rings, a 2D mesh, a line, and a small clustered architecture representative of a QCCD ion trap.

We use error rates reported by IBM obtained via randomized benchmarking on a daily basis; for simulations we use error numbers obtained from Johannesburg obtained on 8/19/2020 with an average T1 time of $70.87\mu s$, T2 time of $72.72\mu s$, two qubit gate time of $0.559\mu s$, a one qubit gate time of $0.07\mu s$, two qubit gate error of 0.0147, one qubit gate error of 0.0004. Source code for all experiments is available at [60]. Experiments using IBM are tested with version 0.14.0 through their Python API. When compiling with Qiskit for the single Toffoli experiments, we use the default settings for the `transpile` function while specifying the Johannesburg backend. This means light optimization is performed: a stochastic routing policy is chosen, and some simple optimizations such as single qubit gate consolidation is performed. We fix the initial mapping to force routing to occur.

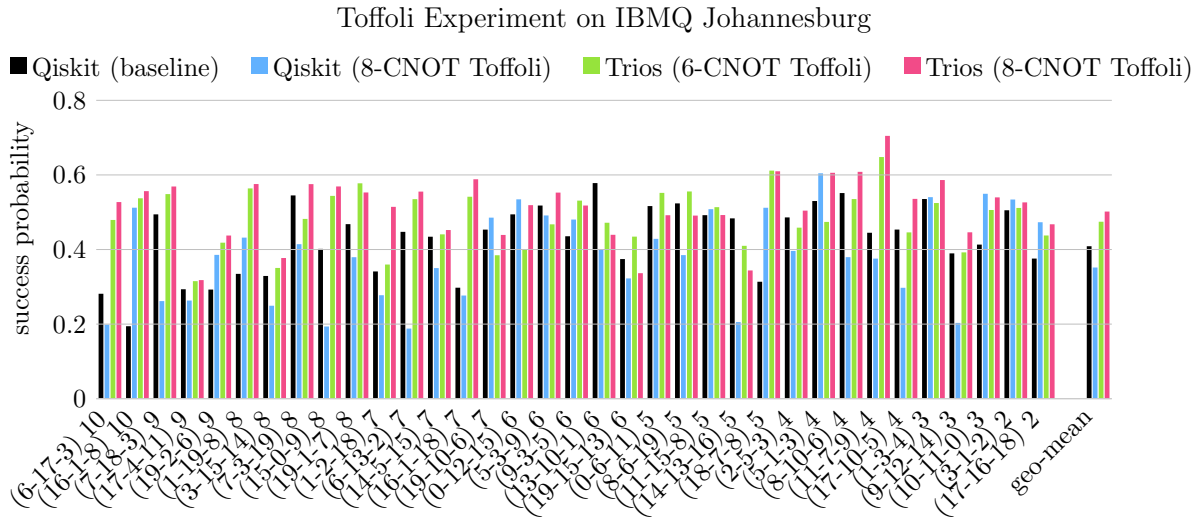


Figure 5.35: Success probabilities of Toffoli gates between random triplets of qubits. Higher is better. The x labels specify the three qubits and total swap distance. The geometric mean success rates for each compiler are 41%, 35%, 47%, and 50% respectively. Trios (8-CNOT) improves average success rate by 23% vs. the Qiskit baseline.

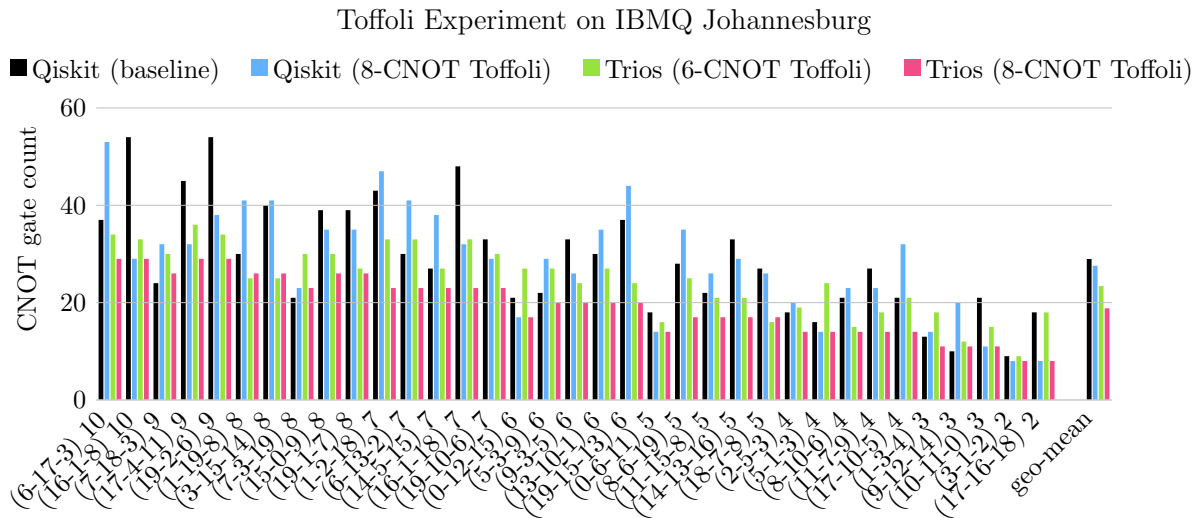


Figure 5.36: Total number of two-qubit (CNOT) gates required to execute a Toffoli gate between various distant qubits. Lower is better. The x labels specify the three qubits and total swap distance. The geometric mean gate counts for each compiler are 29, 28, 23, and 19 respectively. Trios (8-CNOT) reduces average gate count by 35%.

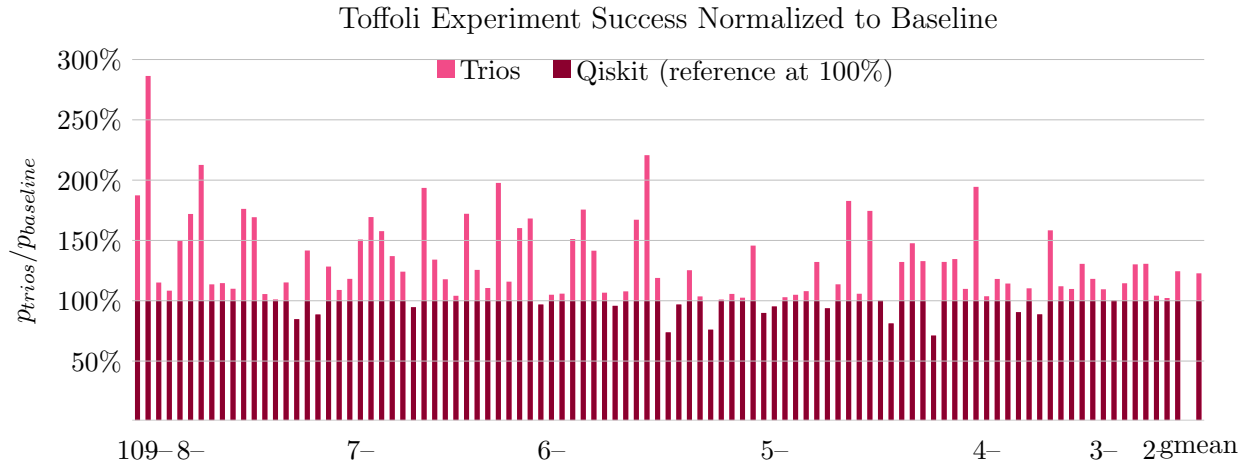


Figure 5.37: Normalized success probabilities of Toffoli gates between triplets of qubits. Higher is better. Bars below 100% indicate lower success rate for Trios. The geometric mean increase in success rate is 23%. The x labels indicate the qubit distance for a range of bars.

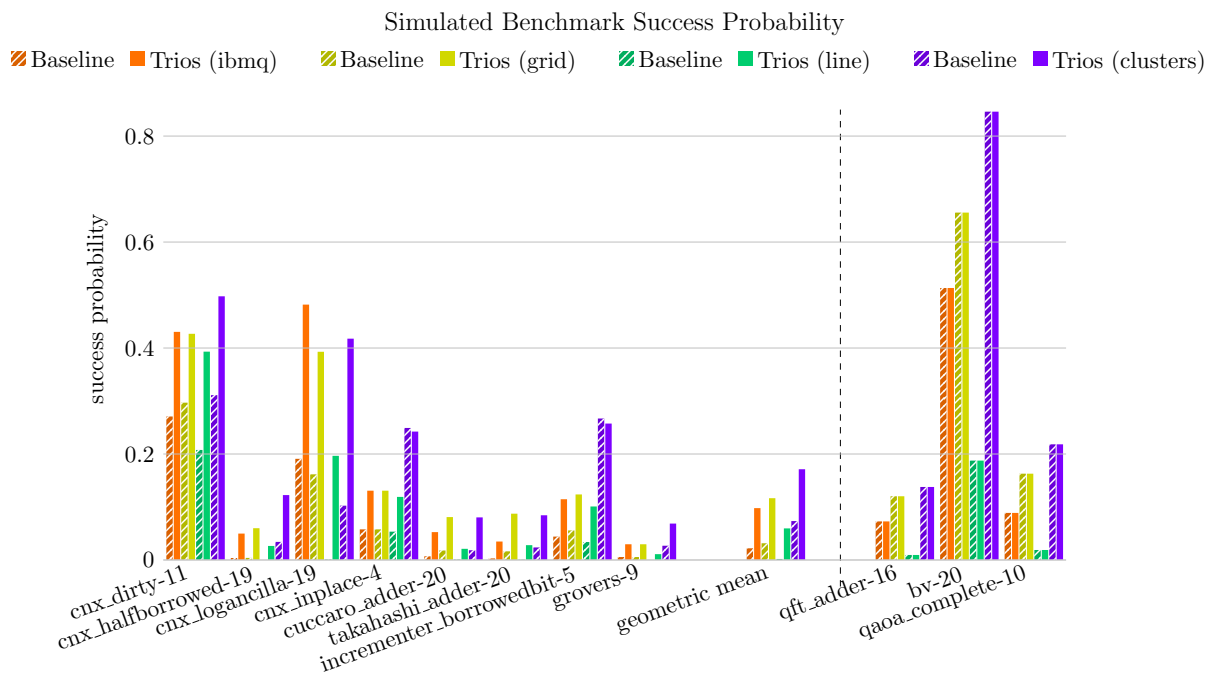


Figure 5.38: Simulated upper-bounds on the program execution success probability on various hardware (using 20x lower idle and gate errors than Johannesburg). Neighboring pairs of bars compare the baseline with Trios compiled for Johannesburg. Higher is better when comparing pairs of bars with the same color. The geometric mean success rates over the benchmarks that use Toffoli gate for each device type respectively are 2.2%→9.8%, 3.2%→12%, 0.19%→6.0%, 7.3%→17%. The rightmost three benchmarks contain zero Toffoli gates so have no change vs. the baseline.

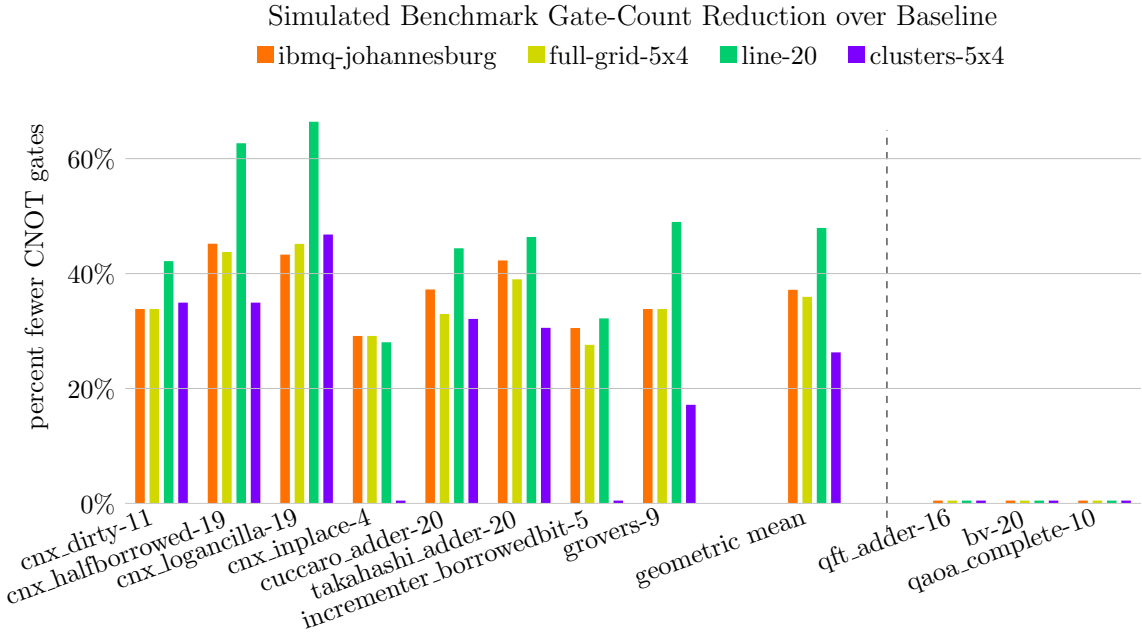


Figure 5.39: A comparison between the baseline and Trios for various hardware. Above 0% indicates benefit. All two-qubit gates (for communication and computation) are counted. The geometric mean reductions in gate counts are 37%, 36%, 48%, and 26% respectively. The rightmost three benchmarks contain zero Toffoli gates so have no change vs. the baseline.

5.5.4 Results and Discussion

Trios Reduces Total Number of Gates

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

In Figure 5.36 we show 35 different triplets of hardware qubits for each of the four strategies. For each triplet, we note the total distance between the qubits on the hardware, given by the shortest path distance in the underlying topology. Even when the distance is relatively small, Trios outperforms reducing overall gate count and as the distance increases, the margin tends to increase. In the small distance cases, this can be attributed to Trios choosing the better Toffoli decomposition for a linearly connected topology. This is significant

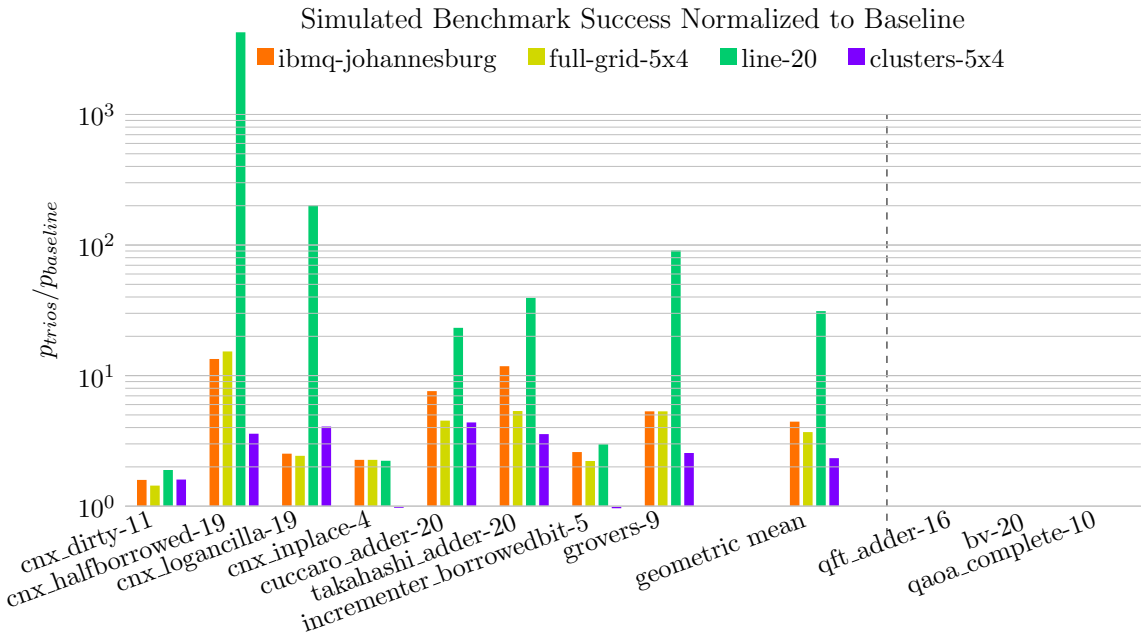


Figure 5.40: Normalized Figure 5.38 to show our consistent increase in program success with Trios. Above 10^0 indicates benefit. Some improvement factors are huge due to near-zero baseline success rates. The geometric mean increases in success rate are 4.4x, 3.7x, 31x, and 2.3x respectively. The rightmost three benchmarks contain zero Toffoli gates so have no change vs. the baseline.

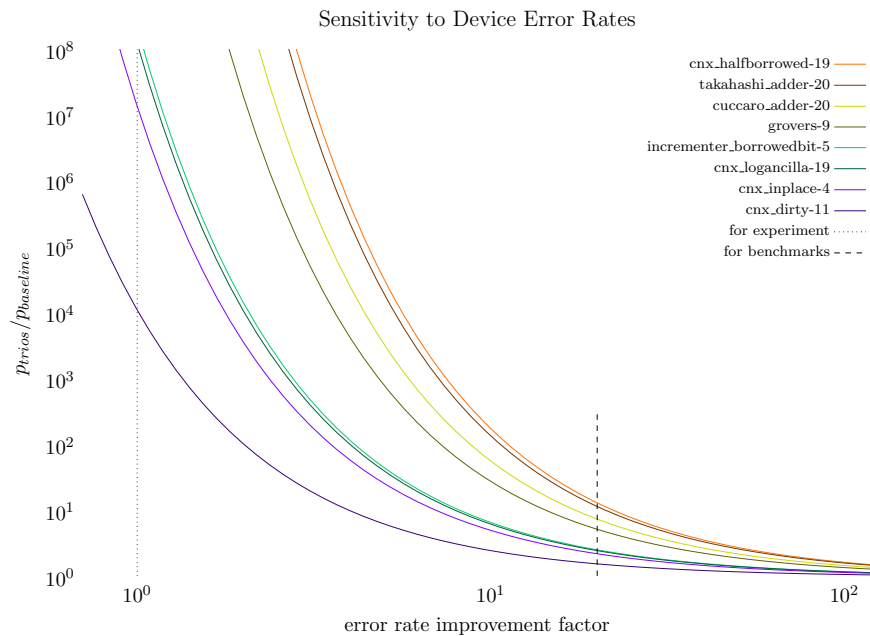


Figure 5.41: Factor of improvement in success rate in Trios over baseline for scaling gate error rates. The dotted line indicates current error rates on IBM Johannesburg and the dashed line (20x improvement) indicates values of the near future used in simulation. In our approximation of success rate factors of improvement in gate error rates lead to an exponential fall off in success ratios, as expected. In the very near term, we expect Trios to drastically improve the execution of quantum programs.

for two reasons. First, the fewer the gates, the less likely an error occurs due to qubit manipulation. Second, fewer gates, especially long sequential chains of SWAPs, often means lower circuit depth, meaning fewer chances for decoherence errors. Together this translates into faster and more successful programs.

This advantage extends to our NISQ benchmarks which contain various numbers of Toffoli gates. In Figure 5.39 we note substantial reductions in total gates across all benchmarks containing Toffoli gates across all underlying topologies. The only exception is the two smallest benchmarks (on 4 and 5 qubits) for the clustered topology because they could be compiled with zero SWAPs.

An extreme of the clustered topology is a single cluster with all-to-all connected qubits. On this device, Orchestrated Trios would have no benefit as operations can be performed between any pair of qubits so no SWAPs are needed and routing is trivial. However, as quantum technologies scale to more than a few qubits, fully-connected architectures hits physical limitations and must be re-engineered. As trapped ion qubit chains get longer, for example, gate operations become slower and lower fidelity. [138] showed that the optimal trap size is 15-25 ions interconnected similar to our cluster model with cluster sizes of 15-25 where Trios does benefit.

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

Trios Improves Overall Success Rate

In general, we expect programs with fewer total two-qubit gates, to succeed with higher probability. In devices with limited connectivity, the addition of routing operations like SWAPs, usually decomposed to 3 CNOTs, can severely reduce the chance an input program can succeed. While success rate is inversely correlated with number of gates, gate error is not

the only reason a program can fail and reducing gate counts does not *guarantee* improved success rates.

In Figure 5.35 we show the success rates of our Toffoli-only experiments when the two controls are initialized to $|1\rangle$ and the target is initialized to $|0\rangle$ so we measure the probability of obtaining $|111\rangle$. These results are obtained from Johannesburg on 8/19/2020. The x-axes of both Figures 5.35 and 5.36 line up to compare gate counts and resulting success rate. In general, experimentally, fewer gates results in substantial improvements to success rates. For example, a Toffoli on (6-17-3) compiled with Trios improves success rate from around 30% to over 50%. On average, we improve success rates by 23 % geomean with max of 286%. In Figure 5.37, we show improvements compiled with Trios normalized to baseline for 99 different triplets of varying total distance on Johannesburg.

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

For our simulated NISQ benchmarks, we see even larger gains. The reduced gate counts in Figure 5.39 translate to major improvements in simulated success rate in Figure 5.38 (normalized success rates in Figure 5.40). For example, in `cnx_logancilla-19`, Trios more than doubles the expected success rates when compiled to each of the architectures. In many cases, the expected success rate of programs compiled with Qiskit is effectively zero while Trios has a realistic chance of obtaining the correct answer. As expected, on programs containing no Toffoli gates, Trios has no effect on success showing that it introduces no excessive overhead. This suggests Trios can easily be added to other quantum compilation toolflows.

Trios Routes Complex Interactions Better

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

Trios greatly improves effectiveness compared to a *heuristic-based* compiler by applying similar heuristics to the higher abstraction level Toffoli gates. An optimal routing of the decomposed circuit would be better except it cannot select the best architecture location-specific decomposition. This makes a huge difference specifically with Toffolis on any square-grid-based device. One might choose to improve the solution found by an optimal compiler by always decomposing Toffolis to the 8-CNOT version before optimally routing, but this will still limit the solution. There are multiple possible qubit orders for the decomposition and the best can only be selected after the routing pass.

Simulation Sensitivity to Error Rates

For our simulations we use an error model (20x better than current errors on Johannesburg) which is forward looking. As errors improve, we expect Trios to have a reduced impact on program success rates since gate errors will contribute less and less to program failure though Trios will never perform worse than the baseline. In Figure 5.41 we study the sensitivity of simulation results to two qubit error rates beginning with current IBM error rates. For poor error rates, the benefit of Trios is extremely large, owed to the fact that programs compiled with the baseline have probabilities of success very close to 0. In our simplified simulation framework, as error rates improve we expect an exponential drop off in improvement with

the most advantage obtained with current error rates.

5.5.5 *Remarks*

We present a new quantum compilation structure, Trios, with a split decomposition pass to greatly reduce compiled communication cost and enable architecture-tuned decompositions. We specifically target the three-qubit Toffoli operation to capture program structure enabling more optimal compiled circuits. Because current quantum computers are especially error prone, they require high levels of optimization to reduce gate counts and maximize the probability the compiled program will succeed.

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

CHAPTER 6

DISCUSSION AND CONCLUSION

The high level objective of this work is to open a discussion on how to approach new hardware technologies - what are the fundamental tradeoff spaces as we move from small numbers of error-prone devices to large numbers of error corrected devices? This work introduces and adapts software optimization frameworks, both technology-dependent and technology-agnostic, to bridge the gap between what can be computed on available hardware and long-term application targets with a secondary objective in opening a discussion on how to fairly evaluate competing quantum technologies.

Recently, hardware developers have begun to develop more reliable, more consistent, and much larger devices composed of increasing numbers of qubits each typically corresponding to a single physical device which can be connected together to give a larger, monolithic machine. We are moving quickly beyond small-scale proof-of-concept machines composed of 1-5 qubits and providers are able to fabricate machines on the orders of 10s and 100s of qubits. Major industry players like IBM, IonQ and ColdQuanta have demonstrated machines with 127 superconducting qubits, 32 trapped ion qubits, and 64 neutral atom qubits each different underlying physical technologies used to realize qubits. While increasing qubit counts is a major accomplishment as it allows us to think about executing programs of non-trivial sizes, but alone is not sufficient. Most of these machines are missing consistent reliability which is needed to get these qubits and machines to actually work. Evaluation of these devices at scale is currently necessary to determine their long-term viability, and therefore it is vital to develop corresponding optimization frameworks to get them there.

Quantum systems are noisy, and current ones prohibitively so, meaning as we execute programs there is a chance that desired operations don't execute as expected or qubit states decohere; meaning that the qubits can only be reliably usable for a limited amount time. For success in the long run, we need to simultaneously increase qubit counts while suppressing

these various error rates. There are a variety of competing technologies, some examples mentioned above. Each has their own promising advantages but come with fundamental limitations to scalability; for example different technologies have specific types of errors or drawbacks which are more dominant than others. These require tailored optimizations to mitigate. It is currently unclear which technologies will be the eventual winners which able to scale efficiently to support large-scale error corrected applications.

This uncertainty makes it imperative to evaluate each technology to determine its viability as we scale - and highlights the primary objective of this work. Physicists and hardware developers often focus on improvement to fundamental properties in small physical prototypes, while this work proposes an alternative which is to examine the effects of advantages and limitations in systems with thousands to millions of physical devices. It is often the case that the expected appeals of these systems are overshadowed by more important features at scale.

The ultimate goal of quantum computation may be large scale error corrected applications with millions of physical qubits, but there is certainly a long trajectory from where we are now, where we can execute near-term algorithm on hardware with modest numbers of noisy qubits, and where we're going, the far right where we have huge numbers of qubits which are protected from error. This work compiles several points along this this spectrum and across many different technologies. It's important to note, that even work that targets near-term improvements such as hardware error mitigation, can accelerate progress towards error correction.

At the highest level, this work is motivated by how to fairly evaluate new quantum technologies as they are developed by designing appropriate architectures, adapting existing optimization frameworks, and building complete and tailored compilation pipelines. This evaluation framework can be viewed as a pyramid; that is suppose we're presented with some new piece of quantum technology - for example a new way to implement a qubit or a new way to communicate qubits at long distances - we want to think about many different

factors affecting whether or not it will be compatible with our top -level question “will this technology scale efficiently into the future?” to begin to answer this, the goal is to ask the right set of questions and apply the right set of optimization tools to maximize proposed advantages while mitigating fundamental hardware limitations; we need to both optimize over trade-off spaces but identify the most important first. In many cases, the key limitations may not be immediate and similarly what might be thought to be prohibitive could be effectively mitigated in software.

Here, this top level question is supported by three primary legs or factors that will help guide these questions. The first of these categories of questions is “does this technology support fast and economical program execution?” Fast meaning how long does the actual quantum circuit take to execute on hardware which will be directly related to output quality or meaning how long does it take in terms of wall-clock time which is relevant for say variational algorithms where we want to avoid the quantum computer taking too long that machine parameters change. Second, we want to consider “can this technology produce consistent and high quality program outputs?” Limited coherence times mean the time from initialization to measurement to obtain our classical bitstring answer is bounded and gates on those qubits may not execute exactly as planned, for example we might over-rotate or rotate around the wrong axis. These types of errors can vary over time and from qubit to qubit and result in incorrect or non-optimal answers. While errors can be very complex, it is useful to discern which are most dominant. Finally, we want to ensure that whichever technology we choose is going to be suitable for both near-term and long-term applications - “will it’s properties match what’s needed for programs we are and are going to run?” In one sense, there are a limited set of known algorithms to date and while it’s continuing to grow it may be more appealing to design architectures and choose technologies based on our ultimate application targets as we saw in the VLQ work.

Each technology that emerges will have its own diverse set of unique advantages and

disadvantages - asking the right sets of questions will guide our development of hardware-specific optimizations and enable a holistic evaluation of its ability to scale into the future. There is clearly an interplay between each of these high-level lines of questioning. For example: clearly support for applications will require a certain degree of reliability which will be dependent on say the underlying hardware topology, how the qubits are connected. Full evaluation relies on full-stack, cross-cutting examination of these tradeoff spaces.

In this work we have discussed several case studies based on emerging technologies and their corresponding optimization frameworks which more deeply analyze some fundamental questions about emerging quantum technologies at scale. For example, multivalued logic for quantum computation extends the computational space of each physical device which when used properly, e.g. to mitigate the increased possibility for error, can lead to some major gains in both physical space requirements and execution time. However, there is still many open questions about our temporary ternary (or higher states) paradigm - such as will it be useful beyond hand optimizations? Can we develop compilation and optimization frameworks which apply this strategy to any input?

Similarly, we've shown certain applications, like the Surface Code quantum error correction code, despite being designed for 2D architectures, are actually better suited for a 2.5D or 3D architecture resulting in lower physical qubit requirements and faster gate executions and under our realistic error model have equivalently good or slightly worse error correction thresholds as a baseline 2D implementation. Again, there are many open questions remaining - such as "is this architecture the best match for the application? Is this application the best match for this architecture? What about for some new architecture based on a completely different technology?"

Ultimately, each of the presented projects are examples just scratching the surface of a much larger set of questions centered towards the end-goal of efficient scalability. There are so many potential technologies, this work only exploring one subset, with unique sets

of advantages and constraints both physically and at scale, and its entirely unclear which technologies are going to be ultimately viable or best suited for quantum applications.

What's clear, however, is that any evaluation of these technologies requires a complete exploration of systems-level tradeoffs. Clearly some of the most important advantages of a new technology can only be studied at scale and may not even be obvious from small-scale prototypes. Furthermore, any evaluation will require tailored technology-specific, vertically integrated, optimization frameworks to push the technology to its true limitations. This is a full-hardware-software-stack interdisciplinary-effort - improving on and optimizing qubit implementations, their control, software mitigation of errors, a architecture specific compilation framework, and applications like quantum error correction tailored to the architecture. Our ultimate objective is to develop these technology-specific frameworks so we can best determine which are well positioned to support near and intermediate-term algorithms as we transition to large-scale quantum computation in the future.

REFERENCES

- [1] Neutral atom compilation.
- [2] Cirq: A python framework for creating, editing, and invoking noisy intermediate scale quantum (NISQ) circuits. <https://github.com/quantumlib/Cirq>, 2018.
- [3] A preview of bristlecone, google’s new quantum processor, Mar 2018.
- [4] Quantum devices simulators, Jun 2018.
- [5] Quantum devices and simulators. <https://www.research.ibm.com/ibm-q/technology/devices/>, 2018.
- [6] Code for asymptotic improvements to quantum circuits via qutrits. <https://github.com/epiqc/qutrits>, 2019.
- [7] A. J. Abhari, A. Faruque, M. J. Dousti, L. Svec, O. Catu, A. Chakrabati, C.-F. Chiang, S. Vanderwilt, J. Black, F. Chong, M. Martonosi, M. Suchara, K. Brown, M. Pedram, and T. Brun. Scaffold: Quantum programming language. Report TR-934-12, Princeton University, 2012.
- [8] H. Abraham, AduOfiei, R. Agarwal, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, M. Amy, E. Arbel, Arijit02, A. Asfaw, A. Avkhadiiev, C. Azaustre, AzizNgoueya, A. Banerjee, A. Bansal, P. Barkoutsos, A. Barnawal, G. Barron, G. S. Barron, L. Bello, Y. Ben-Haim, D. Bevenius, A. Bhobe, L. S. Bishop, C. Blank, S. Bolos, S. Bosch, Brandon, S. Bravyi, Bryce-Fuller, D. Bucher, A. Burov, F. Cabrera, P. Calpin, L. Capelluto, J. Carballo, G. Carrascal, A. Chen, C.-F. Chen, E. Chen, J. C. Chen, R. Chen, J. M. Chow, S. Churchill, C. Claus, C. Clauss, R. Cocking, F. Correa, A. J. Cross, A. W. Cross, S. Cross, J. Cruz-Benito, C. Culver, A. D. Córcoles-Gonzales, S. Dague, T. E. Dandachi, M. Daniels, M. Dartiailh, DavideFrr, A. R. Davila, A. Dekusar, D. Ding,

J. Doi, E. Drechsler, Drew, E. Dumitrescu, K. Dumon, I. Duran, K. EL-Safty, E. Eastman, G. Eberle, P. Eendebak, D. Egger, M. Everitt, P. M. Fernández, A. H. Ferrera, R. Fouilland, FranckChevallier, A. Frisch, A. Fuhrer, B. Fuller, M. GEORGE, J. Gacon, B. G. Gago, C. Gambella, J. M. Gambetta, A. Gammanpila, L. Garcia, T. Garg, S. Garton, A. Gilliam, A. Giridharan, J. Gomez-Mosquera, Gonzalo, S. de la Puente González, J. Gorzinski, I. Gould, D. Greenberg, D. Grinko, W. Guan, J. A. Gunnels, M. Haglund, I. Haide, I. Hamamura, O. C. Hamido, F. Harkins, V. Havlicek, J. Hellmers, L. Herok, S. Hillmich, H. Horii, C. Howington, S. Hu, W. Hu, J. Huang, R. Huisman, H. Imai, T. Imamichi, K. Ishizaki, R. Iten, T. Itoko, JamesSeaward, A. Javadi, A. Javadi-Abhari, W. Javed, Jessica, M. Jivrajani, K. Johns, S. Johnstun, Jonathan-Shoemaker, V. K, T. Kachmann, A. Kale, N. Kanazawa, Kang-Bae, A. Karazeev, P. Kassebaum, J. Kelso, S. King, Knabberjoe, Y. Kobayashi, A. Kovyrrshin, R. Krishnakumar, V. Krishnan, K. Krsulich, P. Kumkar, G. Kus, R. LaRose, E. Lacal, R. Lambert, J. Lapeyre, J. Latone, S. Lawrence, C. Lee, G. Li, D. Liu, P. Liu, Y. Maeng, K. Majmudar, A. Malyshev, J. Manela, J. Marecek, M. Marques, D. Maslov, D. Mathews, A. Matsuo, D. T. McClure, C. McGarry, D. McKay, D. McPherson, S. Meesala, T. Metcalfe, M. Mevissen, A. Meyer, A. Mezzacapo, R. Midha, Z. Minev, A. Mitchell, N. Moll, J. Montanez, G. Monteiro, M. D. Mooring, R. Morales, N. Moran, M. Motta, MrF, P. Murali, J. Müggenburg, D. Nadlinger, K. Nakanishi, G. Nannicini, P. Nation, E. Navarro, Y. Naveh, S. W. Neagle, P. Neuweiler, J. Nicander, P. Niroula, H. Norlen, NuoWenLei, L. J. O’Riordan, O. Ogunbayo, P. Ollitrault, R. Otaolea, S. Oud, D. Padilha, H. Paik, S. Pal, Y. Pang, V. R. Pascuzzi, S. Perriello, A. Phan, F. Piro, M. Pistoia, C. Piveteau, P. Pocreau, A. Pozas-iKerstjens, M. Prokop, V. Prutyranov, D. Puzzuoli, J. Pérez, Quintiii, R. I. Rahman, A. Raja, N. Ramagiri, A. Rao, R. Raymond, R. M.-C. Redondo, M. Reuter, J. Rice, M. Riedemann, M. L. Rocca, D. M. Rodríguez, RohithKarur, M. Rossmannek, M. Ryu, T. SAPV, SamFerracin, M. Sandberg, H. Sandesara, R. Sapra,

H. Sargsyan, A. Sarkar, N. Sathaye, B. Schmitt, C. Schnabel, Z. Schoenfeld, T. L. Scholten, E. Schoute, J. Schwarm, I. F. Sertage, K. Setia, N. Shammah, Y. Shi, A. Silva, A. Simonetto, N. Singstock, Y. Siraichi, I. Sitdikov, S. Sivarajah, M. B. Sletfjording, J. A. Smolin, M. Soeken, I. O. Sokolov, I. Sokolov, SooluThomas, Starfish, D. Steenken, M. Stypulkoski, S. Sun, K. J. Sung, H. Takahashi, T. Takawale, I. Tavernelli, C. Taylor, P. Taylour, S. Thomas, M. Tillet, M. Tod, M. Tomasik, E. de la Torre, K. Trabing, M. Treinish, TrishaPe, D. Tulsi, W. Turner, Y. Vaknin, C. R. Valcarce, F. Varchon, A. C. Vazquez, V. Villar, D. Vogt-Lee, C. Vuillot, J. Weaver, J. Weidenfeller, R. Wieczorek, J. A. Wildstrom, E. Winston, J. J. Woehr, S. Woerner, R. Woo, C. J. Wood, R. Wood, S. Wood, S. Wood, J. Wootton, D. Yeralin, D. Yonge-Mallo, R. Young, J. Yu, C. Zachow, L. Zdanski, H. Zhang, C. Zoufal, Zoufal, a kapila, a matsuo, beammorrison, brandhsn, nick bronn, brosand, chlorophyll zz, csseifms, dekel.meirom, dekelmeirom, dekol, dime10, drholmie, dtrenev, ehchen, elfrocampeador, faisaldebouni, fanizza-marco, gabrieleagl, gadiel, galeinston, georgios ts, gruu, hhorii, hykavitha, jagunther, jliu45, jscott2, kanejess, klinvill, krutik2966, kurarr, lerongil, ma5x, merav aharoni, michelle4654, ordmoj, sagar pahwa, rmoyard, saswati qiskit, scottkelso, sethmerkel, shaashwat, sternparky, strickroman, sumitpuri, tigerjack, toural, tsura crisaldo, vvilpas, welien, willhbang, yang.luh, yotamvakninibm, and M. Čepulkovskis. Qiskit: An open-source framework for quantum computing, 2019.

- [9] M. Amy, D. Maslov, M. Mosca, and M. Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. 2012.
- [10] 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, 2019.
- [11] J. M. Baker, C. Duckering, P. Gokhale, and A. Litteken. Quantum circuit benchmarks. <https://github.com/jmbaker94/quantumcircuitbenchmarks>, 2020.
- [12] J. M. Baker, C. Duckering, A. Hoover, and F. T. Chong. Decomposing quantum generalized toffoli with an arbitrary number of ancilla. *arXiv preprint*, Apr. 2019.
- [13] 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*, pages 98–107, 2020.
- [14] A. Bapat, Z. Eldredge, J. R. Garrison, A. Deshpande, F. T. Chong, and A. V. Gorshkov. Unitary entanglement construction in hierarchical networks. *Physical Review A*, 98(6):062328, 2018.
- [15] 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. *Phys. Rev. A*, 52:3457–3467, Nov 1995.
- [16] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O’Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov, A. N. Cleland,

- and J. M. Martinis. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508:500 EP –, 04 2014.
- [17] E. Barnes, C. Arenz, A. Pitchford, and S. E. Economou. Fast microwave-driven three-qubit gates for cavity-coupled superconducting qubits. *Phys. Rev. B*, 96:024504, Jul 2017.
- [18] 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.
- [19] 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.
- [20] A. Bermudez, X. Xu, R. Nigmatullin, J. O’Gorman, V. Negnevitsky, P. Schindler, T. Monz, U. G. Poschinger, C. Hempel, J. Home, F. Schmidt-Kaler, M. Biercuk, R. Blatt, S. Benjamin, and M. Müller. Assessing the progress of trapped-ion processors towards fault-tolerant quantum computation. 2017.
- [21] E. Bernstein and U. Vazirani. Quantum complexity theory. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC ’93, pages 11–20, June 1993.
- [22] J. Biamonte and V. Bergholm. Tensor networks in a nutshell. *arXiv preprint arXiv:1708.00006*, 2017.
- [23] L. S. Bishop, S. Bravyi, A. Cross, J. M. Gambetta, and J. Smolin. Quantum volume. *Quantum Volume. Technical Report*, 2017.

- [24] N. Bjørner, A.-D. Phan, and L. Fleckenstein. *νz* - an optimizing smt solver. In C. Baier and C. Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 194–199, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- [25] B. R. Blakestad, A. Vandevender, C. Ospelkaus, J. Amini, J. W. Britton, D. G. Leibfried, and D. J. Wineland. High fidelity transport of trapped-ion qubits through an x-junction trap array—nist. *Nature Physics*, 102(Nature Physics), 2009.
- [26] A. Bocharov, M. Roetteler, and K. M. Svore. Factoring with qutrits: Shor’s algorithm on ternary and metaplectic quantum architectures. *Phys. Rev. A*, 96:012306, Jul 2017.
- [27] H. Bombín. Gauge color codes: optimal transversal gates and gauge fixing in topological stabilizer codes. *New Journal of Physics*, 17(8):083002, 2015.
- [28] K. E. Booth, M. Do, J. C. Beck, E. Rieffel, D. Venturelli, and J. Frank. Comparing and integrating constraint programming and temporal planning for quantum circuit compilation. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 2018.
- [29] S. Bravyi and J. Haah. Magic-state distillation with low overhead. *Physical Review A*, 86(5):052329, 2012.
- [30] T. Brecht, W. Pfaff, C. Wang, Y. Chu, L. Frunzio, M. H. Devoret, and R. J. Schoelkopf. Multilayer microwave integrated quantum circuits for scalable quantum computing. *npj Quantum Information*, 2:16002, 2016.
- [31] K. R. Brown, J. Kim, and C. Monroe. Co-designing a scalable quantum computer with trapped atomic ions. *npj Quantum Information*, 2:16034, 2016.
- [32] N. C. Brown and K. R. Brown. Comparing zeeman qubits to hyperfine qubits in the context of the surface code: $^{174}\text{Yb}^+$ and $^{171}\text{Yb}^+$. *Phys. Rev. A*, 97:052301, May 2018.

- [33] T. A. Brun. A simple model of quantum trajectories. *American Journal of Physics*, 70(7):719–737, 2002.
- [34] 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.
- [35] J.-L. Brylinski and R. Brylinski. Universal quantum gates. In *Mathematics of quantum computation*, pages 117–134. Chapman and Hall/CRC, 2002.
- [36] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. 42(3):153 – 159.
- [37] T. Bækkegaard, L. B. Kristensen, N. J. S. Loft, C. K. Andersen, D. Petrosyan, and N. T. Zinner. Superconducting qutrit-qubit circuit: A toolbox for efficient quantum gates. *arXiv preprint arXiv:1802.04299*, 2018.
- [38] A. Y. Chernyavskiy, V. V. Voevodin, and V. V. Voevodin. Parallel computational structure of noisy quantum circuits simulation. *Lobachevskii Journal of Mathematics*, 39(4):494–502, May 2018.
- [39] A. M. Childs and W. van Dam. Quantum algorithm for a generalized hidden shift problem. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1225–1232, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.
- [40] K. S. Chou, J. Z. Blumoff, C. S. Wang, P. C. Reinhold, C. J. Axline, Y. Y. Gao, L. Frunzio, M. H. Devoret, L. Jiang, and R. J. Schoelkopf. Deterministic teleportation of a quantum gate between two logical qubits. *Nature*, 561(7723):368–373, 2018.
- [41] K. S. Chou, J. Z. Blumoff, C. S. Wang, P. C. Reinhold, C. J. Axline, Y. Y. Gao, L. Frunzio, M. H. Devoret, L. Jiang, and R. J. Schoelkopf. Deterministic teleportation of a quantum gate between two logical qubits, 2018.

- [42] D. G. Cory, M. D. Price, W. Maas, E. Knill, R. Laflamme, W. H. Zurek, T. F. Havel, and S. S. Somaroo. Experimental quantum error correction. *Phys. Rev. Lett.*, 81:2152–2155, Sep 1998.
- [43] 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.*, 122:173201, May 2019.
- [44] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah. On the qubit routing problem. *arXiv preprint arXiv:1902.08091*, 2019.
- [45] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta. Open quantum assembly language, 2017.
- [46] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton. A new quantum ripple-carry addition circuit, 2004.
- [47] 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.
- [48] L. de Moura and N. Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [49] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536:63 EP –, Aug 2016.
- [50] E. Dennis. Toward fault-tolerant quantum computation without concatenation. *Phys. Rev. A*, 63:052314, Apr 2001.

- [51] S. J. Devitt, K. Nemoto, and W. J. Munro. Quantum error correction for beginners. 2009.
- [52] M. H. Devoret and R. J. Schoelkopf. Superconducting circuits for quantum information: an outlook. *Science*, 339(6124):1169–1174, 2013.
- [53] Y.-M. Di and H.-R. Wei. Elementary gates for ternary quantum logic circuit. *arXiv preprint arXiv:1105.5485*, 2011.
- [54] C. H. Q. Ding and H. D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 107–114, Nov 2001.
- [55] Y. Ding, A. Holmes, A. Javadi-Abhari, D. Franklin, M. Martonosi, and F. Chong. Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 828–840. IEEE, 2018.
- [56] Y. Ding, A. Holmes, A. Javadi-Abhari, D. Franklin, M. Martonosi, and F. Chong. Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 828–840. IEEE, 2018.
- [57] T. G. Draper. Addition on a quantum computer. *arXiv preprint quant-ph/0008033*, 2000.
- [58] T. G. Draper, S. A. Kutin, E. M. Rains, and K. M. Svore. A logarithmic-depth quantum carry-lookahead adder. *Quantum Information & Computation*, 6(4):351–369, 2006.
- [59] L.-M. Duan and C. Monroe. Colloquium: Quantum networks with trapped ions. *Reviews of Modern Physics*, 82(2):1209, 2010.

- [60] C. Duckering and J. M. Baker. Simulation source code for virtualized logical qubits. <https://github.com/cduck/VLQ>, 2020.
- [61] N. Earnest, S. Chakram, Y. Lu, N. Irons, R. K. Naik, N. Leung, L. Ocola, D. A. Czaplewski, B. Baker, J. Lawrence, J. Koch, and D. I. Schuster. Realization of a Λ system with metastable states of a capacitively shunted fluxonium. *Phys. Rev. Lett.*, 120:150504, Apr 2018.
- [62] 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.
- [63] Y. Fan. Applications of multi-valued quantum algorithms. *arXiv preprint arXiv:0809.0932*, 2008.
- [64] E. Farhi, J. Goldstone, and S. Gutmann. A quantum approximate optimization algorithm. *arXiv preprint*, Nov. 2014.
- [65] A. Fedorov, L. Steffen, M. Baur, M. P. da Silva, and A. Wallraff. Implementation of a toffoli gate with superconducting circuits. *Nature*, 481:170 EP –, Dec 2011.
- [66] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181. IEEE, 1982.
- [67] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.
- [68] A. G. Fowler, A. C. Whiteside, and L. C. Hollenberg. Towards practical classical processing for the surface code. *Physical review letters*, 108(18):180501, 2012.
- [69] A. Fuhrmanek, R. Bourgain, Y. R. P. Sortais, and A. Browaeys. Free-space lossless state detection of a single trapped atom. *Phys. Rev. Lett.*, 106:133003, Mar 2011.

- [70] C. Gidney. Constructing large controlled nots, 2015.
- [71] C. Gidney. Factoring with $n+2$ clean qubits and $n-1$ dirty qubits. *arXiv preprint arXiv:1706.07884*, 2017.
- [72] C. Gidney and M. Ekerå. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *arXiv preprint arXiv:1905.09749*, 2019.
- [73] S. M. Girvin. Circuit qed: superconducting qubits coupled to microwave photons. *Quantum Machines: Measurement and Control of Engineered Quantum Systems*, page 113, 2011.
- [74] 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*, pages 554–566. ACM, 2019.
- [75] T. F. Gonzalez and D. Serena. Complexity of pairwise shortest path routing in the grid. *Theoretical Computer Science*, 326(1):155 – 185, 2004.
- [76] 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.
- [77] A. D. Greentree, S. G. Schirmer, F. Green, L. C. L. Hollenberg, A. R. Hamilton, and R. G. Clark. Maximizing the hilbert space for a finite number of distinguishable quantum states. *Phys. Rev. Lett.*, 92:097901, Mar 2004.
- [78] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Annual ACM Symposium on Theory of Computing*, pages 212–219. ACM, 1996.
- [79] G. G. Guerreschi and J. Park. Two-step approach to scheduling quantum circuits. *Quantum Science and Technology*, 3(4):045003, jul 2018.

- [80] G. G. Guerreschi and J. Park. Two-step approach to scheduling quantum circuits. *Quantum Science and Technology*, 3(4):045003, 2018.
- [81] T. Häner, M. Roetteler, and K. M. Svore. Factoring using $2n + 2$ qubits with toffoli based modular multiplication. *Quantum Info. Comput.*, 17(7-8):673–684, June 2017.
- [82] T. Häner and D. S. Steiger. 0.5 petabyte simulation of a 45-qubit quantum circuit. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 33. ACM, 2017.
- [83] C. T. Hann, C.-L. Zou, Y. Zhang, Y. Chu, R. J. Schoelkopf, S. M. Girvin, and L. Jiang. Hardware-efficient quantum random access memory with hybrid quantum acoustic systems. *arXiv preprint arXiv:1906.11340*, 2019.
- [84] Y. He, M.-X. Luo, E. Zhang, H.-K. Wang, and X.-F. Wang. Decompositions of n-qubit Toffoli Gates with Linear Circuit Complexity. *International Journal of Theoretical Physics*, 56:2350–2361, July 2017.
- [85] J. Heckey, S. Patil, A. JavadiAbhari, A. Holmes, D. Kudrow, K. R. Brown, D. Franklin, F. T. Chong, and M. Martonosi. Compiler management of communication and parallelism for quantum computation. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 445–456, New York, NY, USA, 2015. ACM.
- [86] B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995.
- [87] B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. 1995.
- [88] L. Henriët, L. Beguin, A. Signoles, T. Lahaye, A. Browaeys, G.-O. Raymond, and C. Jurczak. Quantum computing with neutral atoms. *Quantum*, 4:327, Sep 2020.

- [89] 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.
- [90] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.
- [91] D. Hucul, J. E. Christensen, E. R. Hudson, and W. C. Campbell. Spectroscopy of a synthetic trapped ion qubit. *Physical review letters*, 119(10):100501, 2017.
- [92] IBM Quantum Devices. <https://quantumexperience.ng.bluemix.net/qx/devices>. Accessed: 2019-03-16.
- [93] IBM Quantum Devices. <https://quantumexperience.ng.bluemix.net/qx/devices>. Accessed: 2018-05-16.
- [94] IBM Quantum Experience API. <https://github.com/Qiskit/qiskit-api-py/tree/master/IBMQQuantumExperience>. Accessed: 2018-05-16.
- [95] IBM Announces Advances to IBM Quantum Systems and Ecosystem. <https://www-03.ibm.com/press/us/en/pressrelease/53374.wss>. Accessed: 2018-08-05.
- [96] S. S. Ivanov, H. S. Tonchev, and N. V. Vitanov. Time-efficient implementation of quantum search with qudits. *Phys. Rev. A*, 85:062321, Jun 2012.
- [97] 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.
- [98] S. Jandura. Improving a quantum compiler, Sep 2018.
- [99] A. Javadi-Abhari, P. Gokhale, A. Holmes, D. Franklin, K. R. Brown, M. Martonosi, and F. T. Chong. Optimized surface code communication in superconducting quantum

- computers. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pages 692–705, New York, NY, USA, 2017. ACM.
- [100] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi. Scaffcc: A framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, CF '14, pages 1:1–1:10, New York, NY, USA, 2014. ACM.
- [101] F. M. Johannes. Partitioning of vlsi circuits and systems. In *33rd Design Automation Conference Proceedings, 1996*, pages 83–87, June 1996.
- [102] J. Johansson, P. Nation, and F. Nori. Qutip: An open-source python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 183(8):1760–1772, 8 2012.
- [103] J. Johansson, P. Nation, and F. Nori. Qutip 2: A python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 184(4):1234–1240, 4 2013.
- [104] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96 – 129, 1998.
- [105] G. Karypis and V. Kumar. MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. <http://www.cs.umn.edu/~metis>, 2009.
- [106] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2):291–307, 1970.
- [107] N. Khammassi, I. Ashraf, X. Fu, C. G. Almudever, and K. Bertels. Qx: A high-performance quantum computer simulation platform. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '17, pages 464–469, 3001 Leuven, Belgium, Belgium, 2017. European Design and Automation Association.

- [108] M. H. A. Khan and M. A. Perkowski. Quantum ternary parallel adder/subtractor with partially-look-ahead carry. *J. Syst. Archit.*, 53(7):453–464, July 2007.
- [109] 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.
- [110] 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.
- [111] A. B. Klimov, R. Guzmán, J. C. Retamal, and C. Saavedra. Qutrit quantum computer with trapped ions. *Phys. Rev. A*, 67:062313, Jun 2003.
- [112] M. Kues, C. Reimer, P. Roztocky, L. R. Cortés, S. Sciara, B. Wetzell, Y. Zhang, A. Cino, S. T. Chu, B. E. Little, D. J. Moss, L. Caspani, J. Azaña, and R. Morandotti. On-chip generation of high-dimensional entangled quantum states and their coherent control. *Nature*, 546:622 EP –, 06 2017.
- [113] M. Kwon, M. F. Ebert, T. G. Walker, and M. Saffman. Parallel low-loss measurement of multiple atomic qubits. *Phys. Rev. Lett.*, 119:180504, Oct 2017.
- [114] A. J. Landahl, J. T. Anderson, and P. R. Rice. Fault-tolerant quantum computing with color codes. *arXiv preprint arXiv:1108.5738*, 2011.
- [115] B. P. Lanyon, M. Barbieri, M. P. Almeida, T. Jennewein, T. C. Ralph, K. J. Resch, G. J. Pryde, J. L. O’Brien, A. Gilchrist, and A. G. White. Simplifying quantum logic using higher-dimensional hilbert spaces. *Nature Physics*, 5:134 EP –, 12 2008.
- [116] L. Lao, B. van Wee, I. Ashraf, J. van Someren, N. Khammassi, K. Bertels, and C. G. Almudever. Mapping of lattice surgery-based quantum circuits on surface code architectures. *Quantum Science and Technology*, 4(1):015005, 2018.

- [117] C. Lattner and V. Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04, pages 75–, Washington, DC, USA, 2004. IEEE Computer Society.
- [118] B. Lekitsch, S. Weidt, A. G. Fowler, K. Mølmer, S. J. Devitt, C. Wunderlich, and W. K. Hensinger. Blueprint for a microwave trapped-ion quantum computer. 2015.
- [119] H. Levine, A. Keesling, G. Semeghini, A. Omran, T. T. Wang, S. Ebadi, H. Bernien, M. Greiner, V. Vuletić, H. Pichler, et al. Parallel implementation of high-fidelity multiqubit gates with neutral atoms. *Physical review letters*, 123(17):170503, 2019.
- [120] G. Li, Y. Ding, and Y. Xie. Tackling the qubit mapping problem for nisq-era quantum devices, 2018.
- [121] H. Y. Li, C. W. Wu, W. T. Liu, P. X. Chen, and C. Z. Li. Fast quantum search algorithm for databases of arbitrary size and its implementation in a cavity QED system. *Physics Letters A*, 375:4249–4254, Nov. 2011.
- [122] N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, 114(13):3305–3310, 2017.
- [123] D. Litinski. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum*, 3:128, 2019.
- [124] D. Litinski. Magic state distillation: Not as costly as you think. *arXiv preprint arXiv:1905.06903*, 2019.
- [125] I. L. Markov and M. Saeedi. Constant-optimized quantum circuits for modular multiplication and exponentiation. 2012.

- [126] J. Martinis. Quantum supremacy using a programmable superconducting processor, 11 2019. Institute for Quantum Information and Matter Seminar at the California Institute of Technology.
- [127] D. Maslov, Y. Nam, and J. Kim. An outlook for quantum computing [point of view]. *Proceedings of the IEEE*, 107(1):5–10, 2018.
- [128] N. D. Mermin. *Quantum Computer Science: An Introduction*. Cambridge University Press, New York, NY, USA, 2007.
- [129] D. Miller, T. Holz, H. Kampermann, and D. Bruß. Propagation of generalized pauli errors in qudit clifford circuits. *Physical Review A*, 98(5):052316, 2018.
- [130] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, pages 318–323, June 2003.
- [131] C. Monroe and J. Kim. Scaling the ion trap quantum processor. *Science*, 339(6124):1164–1169, 2013.
- [132] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim. Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects. *Phys. Rev. A*, 89:022317, Feb 2014.
- [133] S. Moses, J. Pino, J. Dreiling, C. Figgatt, J. Gaebler, M. Allman, C. Baldwin, M. Foss-Feig, D. Hayes, K. Mayer, C. Ryan-Anderson, and B. Neyenhuis. Demonstration of the QCCD trapped-ion quantum computer architecture. *Bulletin of the American Physical Society*, June 2020.
- [134] E. Mount, D. Gaultney, G. Vrijsen, M. Adams, S.-Y. Baek, K. Hudek, L. Isabella, S. Crain, A. van Rynbach, P. Maunz, et al. Scalable digital hardware for a trapped ion quantum computer. *Quantum Information Processing*, 15(12):5281–5298, 2016.

- [135] 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.
- [136] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1015–1029, Apr. 2019.
- [137] P. Murali, D. M. Debroy, K. R. Brown, and M. Martonosi. Architecting noisy intermediate-scale trapped ion quantum computers. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 529–542. IEEE, 2020.
- [138] P. Murali, D. M. Debroy, K. R. Brown, and M. Martonosi. Architecting noisy intermediate-scale trapped ion quantum computers. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 529–542, June 2020.
- [139] 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, 2020.
- [140] A. Muthukrishnan and C. R. Stroud. Multivalued logic gates for quantum computation. *Phys. Rev. A*, 62:052309, Oct 2000.
- [141] R. Naik, N. Leung, S. Chakram, P. Groszkowski, Y. Lu, N. Earnest, D. McKay, J. Koch, and D. Schuster. Random access quantum information processors using multimode circuit quantum electrodynamics. *Nature communications*, 8(1):1904, 2017.

- [142] R. Naik, N. Leung, S. Chakram, P. Groszkowski, Y. Lu, N. Earnest, D. McKay, J. Koch, and D. Schuster. Random access quantum information processors using multimode circuit quantum electrodynamics. *Nature communications*, 8(1):1904, 2017.
- [143] P. J. Nair. Architectural techniques to enable reliable and scalable memory systems. *arXiv preprint arXiv:1704.03991*, 2017.
- [144] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information*, 4(1):1–12, 2018.
- [145] M. G. Neeley. *Generation of three-qubit entanglement using Josephson phase qubits*. PhD thesis, University of California, Santa Barbara, 2010.
- [146] M. A. Nielsen and I. L. Chuang. Quantum information and quantum computation. *Cambridge: Cambridge University Press*, 2(8):23, 2000.
- [147] 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.
- [148] T. Nowatzki, N. Ardalani, K. Sankaralingam, and J. Weng. Hybrid optimization/heuristic instruction scheduling for programmable accelerator codesign. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques, PACT '18*, pages 36:1–36:15, New York, NY, USA, 2018. ACM.
- [149] T. Nowatzki, M. Sartin-Tarm, L. De Carli, K. Sankaralingam, C. Estan, and B. Robatmili. A general constraint-centric scheduling framework for spatial architectures. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13*, pages 495–506, New York, NY, USA, 2013. ACM.

- [150] T. Nowatzki, M. Sartin-Tarm, L. De Carli, K. Sankaralingam, C. Estan, and B. Robatmili. A scheduling framework for spatial architectures across multiple constraint-solving theories. *ACM Trans. Program. Lang. Syst.*, 37(1):2:1–2:30, Nov. 2014.
- [151] 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.
- [152] M. Otten and S. K. Gray. Accounting for errors in quantum algorithms via individual error reduction. *npj Quantum Information*, 5(1):11, 2019.
- [153] T. Park and C. Y. Lee. Algorithms for partitioning a graph. *Computers & Industrial Engineering*, 28(4):899–909, 1995.
- [154] A. Pavlidis and E. Floratos. Arithmetic circuits for multilevel qudits based on quantum fourier transform. *arXiv preprint arXiv:1707.08834*, 2017.
- [155] J. Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, Aug. 2018.
- [156] T. C. Ralph, K. J. Resch, and A. Gilchrist. Efficient toffoli gates using qudits. *Phys. Rev. A*, 75:022313, Feb 2007.
- [157] J. Randall, A. M. Lawrence, S. C. Webster, S. Weidt, N. V. Vitanov, and W. K. Hensinger. Generation of high-fidelity quantum control methods for multilevel systems. *Phys. Rev. A*, 98:043414, 10 2018.
- [158] J. Randall, S. Weidt, E. D. Standing, K. Lake, S. C. Webster, D. F. Murgia, T. Navickas, K. Roth, and W. K. Hensinger. Efficient preparation and detection of microwave dressed-state qubits and qutrits with trapped ions. *Phys. Rev. A*, 91:012322, 01 2015.

- [159] M. Reagor, W. Pfaff, C. Axline, R. W. Heeres, N. Ofek, K. Sliwa, E. Holland, C. Wang, J. Blumoff, K. Chou, M. J. Hatridge, L. Frunzio, M. H. Devoret, L. Jiang, and R. J. Schoelkopf. Quantum memory with millisecond coherence in circuit qed. *Phys. Rev. B*, 94:014506, Jul 2016.
- [160] L. Ruiz-Perez and J. C. Garcia-Escartin. Quantum arithmetic with the quantum fourier transform. 2014.
- [161] L. Ruiz-Perez and J. C. Garcia-Escartin. Quantum arithmetic with the quantum fourier transform. *Quantum Information Processing*, 16(6):152, 2017.
- [162] L. Ruiz-Perez and J. C. Garcia-Escartin. Quantum arithmetic with the quantum fourier transform. *Quantum Information Processing*, 16(6):152, 2017.
- [163] 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.
- [164] ScaffCC Compiler. <https://github.com/epiqc/ScaffCC>. Accessed: 2018-05-16.
- [165] R. Schack and T. A. Brun. A C++ library using quantum trajectories to solve quantum master equations. *Computer Physics Communications*, 102(1-3):210–228, 1997.
- [166] K. Schloegel, G. Karypis, and V. Kumar. Sourcebook of parallel computing. chapter Graph Partitioning for High-performance Scientific Simulations, pages 491–541. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
- [167] N. Schuch. Implementation of quantum algorithms with Josephson charge qubits. *Universität Regensburg*, Dec. 2002.
- [168] 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.

- [169] H. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(2):135 – 148, 1991. Parallel Methods on Large-scale Structural Analysis and Physics Applications.
- [170] R. S. Smith, M. J. Curtis, and W. J. Zeng. A practical quantum instruction set architecture. *arXiv preprint arXiv:1608.03355*, 2016.
- [171] R. S. Smith, M. J. Curtis, and W. J. Zeng. A practical quantum instruction set architecture. *arXiv preprint arXiv:1608.03355*, 2016.
- [172] M. Soeken, S. Frehse, R. Wille, and R. Drechsler. Revkit: An open source toolkit for the design of reversible circuits. In A. De Vos and R. Wille, editors, *Reversible Computation*, pages 64–76. Springer Berlin Heidelberg, 2012.
- [173] M. Soeken, T. Häner, and M. Roetteler. Programming quantum computers using design automation, 2018.
- [174] F. Tacchino. Personal Communication.
- [175] F. Tacchino, C. Macchiavello, D. Gerace, and D. Bajoni. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, 5(1):26, 2019.
- [176] Y. Takahashi, S. Tani, and N. Kunihiro. Quantum addition circuits and unbounded fan-out. *arXiv preprint*, Oct. 2009.
- [177] S. S. Tannu, Z. A. Myers, P. J. Nair, D. M. Carmean, and M. K. Qureshi. Taming the instruction bandwidth of quantum computers via hardware-managed error correction. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 679–691. IEEE, 2017.
- [178] 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.
- [179] C. J. Trout, M. Li, M. Gutierrez, Y. Wu, S.-T. Wang, L. Duan, and K. R. Brown. Simulating the performance of a distance-3 surface code in a linear ion trap. 2017.
- [180] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.
- [181] D. Venturelli, M. Do, E. Rieffel, and J. Frank. Compiling quantum circuits to realistic hardware architectures using temporal planners. *Quantum Science and Technology*, 3(2):025004, feb 2018.
- [182] A. Wallraff. Deterministic quantum state transfer and generation of remote entanglement using microwave photons. In *APS Meeting Abstracts*, 2018.
- [183] Y. Wang and M. Perkowski. Improved complexity of quantum oracles for ternary grover algorithm for graph coloring. In *2011 41st IEEE International Symposium on Multiple-Valued Logic*, pages 294–301, May 2011.
- [184] 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.
- [185] 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.
- [186] K. Wright, K. Beck, S. Debnath, J. Amini, Y. Nam, N. Grzesiak, J.-S. Chen, N. Pimenti, M. Chmielewski, C. Collins, et al. Benchmarking an 11-qubit quantum computer. *Nature communications*, 10(1):1–6, 2019.

- [187] X. Zhang, H. Xiang, T. Xiang, L. Fu, and J. Sang. An efficient quantum circuits optimizing scheme compared with qiskit, 2018.
- [188] A. Zulehner, A. Paler, and R. Wille. An efficient methodology for mapping quantum circuits to the ibm qx architectures, 2017.