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

Casey Duckering
University of Chicago
Chicago, Illinois, USA

Jonathan M. Baker
University of Chicago
Chicago, Illinois, USA

Andrew Litteken
University of Chicago
Chicago, Illinois, USA

Frederic T. Chong
University of Chicago
Chicago, Illinois, USA

ABSTRACT

Current quantum computers are especially error prone and require high levels of optimization to reduce operation counts and maximize the probability the compiled program will succeed. These computers only support operations decomposed into one- and two-qubit gates and only two-qubit gates between physically connected pairs of qubits. Typical compilers first decompose operations, then route data to connected qubits. We propose a new compiler structure, Orchestrated Trios, that first decomposes to the three-qubit Toffoli, routes the inputs of the higher-level Toffoli operations to groups of nearby qubits, then finishes decomposition to hardware-supported gates.

This significantly reduces communication overhead by giving the routing pass access to the higher-level structure of the circuit instead of discarding it. A second benefit is the ability to now select an architecture-tuned Toffoli decomposition such as the 8-CNOT Toffoli for the specific hardware qubits now known after the routing pass. We perform real experiments on IBM Johannesburg showing an average 35% decrease in two-qubit gate count and 23% increase in success rate of a single Toffoli over Qiskit. We additionally compile many near-term benchmark algorithms showing an average 344% increase in (or 4.44x) simulated success rate on the Johannesburg architecture and compare with other architecture types.

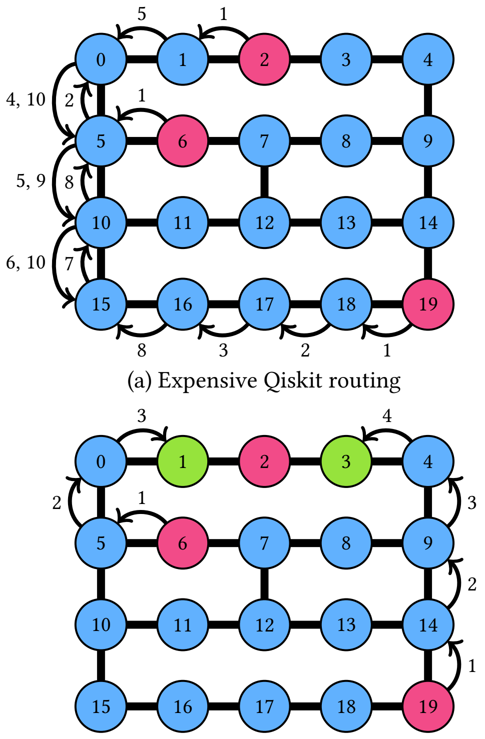
CCS CONCEPTS

• **Computer systems organization** → **Quantum computing**; • **Software and its engineering** → *Compilers.*

KEYWORDS

quantum computing, NISQ, compiler, Toffoli

ACM Reference Format:
Casey Duckering, Jonathan M. Baker, Andrew Litteken, and Frederic T. Chong. 2021. Orchestrated Trios: Compiling for Efficient Communication in Quantum Programs with 3-Qubit Gates. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21), April 19–23, 2021, Virtual, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3445814.3446718



(a) Expensive Qiskit routing

(b) Efficient Trios routing

Figure 1: Example routing from Qiskit (a) vs. Trios (b) for a single Toffoli operation. Circles represent qubits and lines indicate two qubits are connected. Input qubits are highlighted in red. SWAP arrows are labeled by timestep. The routed locations for Trios routing are highlighted in green while Qiskit moves them several times. Qiskit adds 16 SWAPs (=48 CNOTs), some during the Toffoli, while Trios adds only 7 SWAPs (=21 CNOTs) all before the Toffoli. Performing multiple passes of decomposition allows direct routing and enables this huge reduction in communication, increasing the probability of program success.

1 INTRODUCTION

In recent years, quantum hardware has improved dramatically in terms of number of accessible quantum bits (qubits), device error rates, and qubit lifetimes. However, we are still years away from obtaining fully error corrected devices which are required to run important algorithms like Grover's [15] and Shor's [32]. In the current Noisy-Intermediate Scale Quantum (NISQ) [28] era, where despite recent substantial improvements, error rates on current devices are

ASPLOS '21, April 19–23, 2021, Virtual, USA

ACM ISBN 978-1-4503-8317-2/21/04…$15.00
https://doi.org/10.1145/3445814.3446718

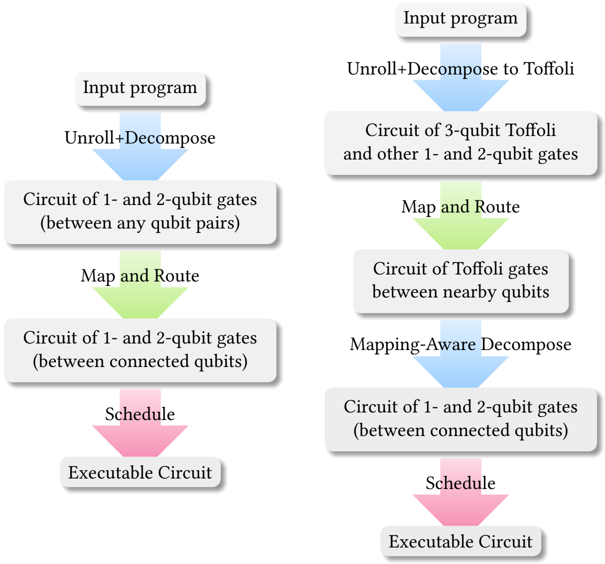
still prohibitive, requiring programs to be highly optimized to have a good chance at succeeding.

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 [26]. 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 $\{u1, u2, u3, cx\}$ [18] (single-qubit gates and the common CNOT gate described later). One critical limitation of many current available architectures is the inability to execute more complex multi-qubit operations, like the Toffoli, directly; instead, these gates must be decomposed into the supported one- and two-qubit gates. Furthermore, many current superconducting architectures only support two 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 [25]. 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 1a. This baseline strategy adds a large number of unnecessary SWAPs as it individually routes each CNOT composing the Toffoli, dramatically reducing the probability of successful execution.

Our approach, Orchestrated Trios (Trios) decomposes and routes qubits in multiple stages, as seen in Figure 2b. For example, first decompose an input program to one- two-, *and* three-qubit gates (e.g. do not decompose Toffoli gates) and route as before except for three-qubits, route all three to a common location with minimal SWAPs. This new program can then undergo a second round of decomposition to produce a circuit containing only hardware permitted one- and two-qubit gates. The second round may use the now known mapping (locations of data qubits on the device) to generate fine-tuned decompositions for the architecture.

This layered approach has a major advantage over current routing techniques: we are better able to capture program structure by inspecting intermediate complex operations for routing. This better informs how qubits should be moved around the device during program execution. In Figure 1, the Trios strategy reduces the total number of SWAPs added to 21: fewer than half compared to Qiskit. This was an extreme example we selected to present the issue, not an average case.


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

(a) Conventional compilation (b) Trios compilation

Figure 2: (a) Typical compilation passes used by Qiskit (simplified). (b) Trios compilation passes.

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 2b. This strategy has a substantial benefit on the overall success rate of programs. We demonstrate these improvements by executing Toffoli gates on a real IBM quantum computer and estimating success probability of a suite of benchmarks via simulation.

Our contributions are as follows:

- A new compiler structure, Trios, with two passes for decomposition with a modified routing pass in between which greatly improves qubit routing.
- A simple method for architecture-tuned Toffoli decompositions during the second decompose pass that allows for a new kind of location-aware optimization.
- On Toffoli-only experiments, Trios reduces the total number of gates by 35% geomean (geometric mean) resulting in 23% geomean increase in success rate when run on real IBM hardware as compared to Qiskit.
- On near-term algorithms shown in Figure 11 (4 to 20 qubit benchmarks), Trios reduces total gate count by 37% geomean resulting in 344% geomean increase in (or 4.44x) simulated success rate on IBM Johannesburg with noise rates of near-future hardware as compared to programs compiled without Trios. A sensitivity analysis over four architecture types shows the benefit range from 133% to 3020% increase in success rate.

2 BACKGROUND

2.1 Quantum Computing Basics

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

While a qubit system can exist in these superpositions during computation, at the end of the computation, the qubits are measured producing a classical binary outcome. The probability of each outcome depends on the amplitude of each basis state (the values of $\alpha, \beta, \gamma, \ldots$). Consequently, since the outcome of a quantum program is a classical bitstring and because quantum systems are inherently noisy, programs are usually run thousands of times to obtain a distribution over possible answers. A comprehensive background can be found in [27].

2.2 Quantum Circuits

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

Quantum circuits have a single line for each qubit, with time flowing from left to right. Gates in a quantum circuits have the same number of inputs and outputs and gates on disjoint sets of lines can be executed in parallel. Single qubit gates are represented as a box labeled with the indicated operation and controlled operations, like the CNOT and Toffoli, have one or two controls respectively indicated by • and target given by ⊕.

Currently available superconducting quantum hardware, like that of IBM and Rigetti, only supports one-qubit gates and two-qubit gates on specific pairs. Therefore, more complex instructions must be decomposed into multiple simpler, supported operations. 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 3, 4. There are two key distinctions in these decompositions illustrating a more general trade off. The first [27] is the most popular decomposition using only 6 CNOT gates but requires CNOTs between all three pairs of qubits. This would require inserted SWAPs or a device connectivity containing a triangle. The second [31] 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.

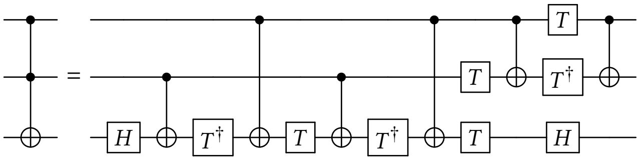


Figure 3: A 6-CNOT decomposition of the Toffoli gate.

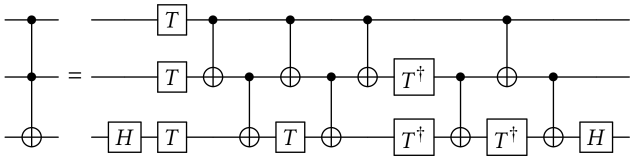


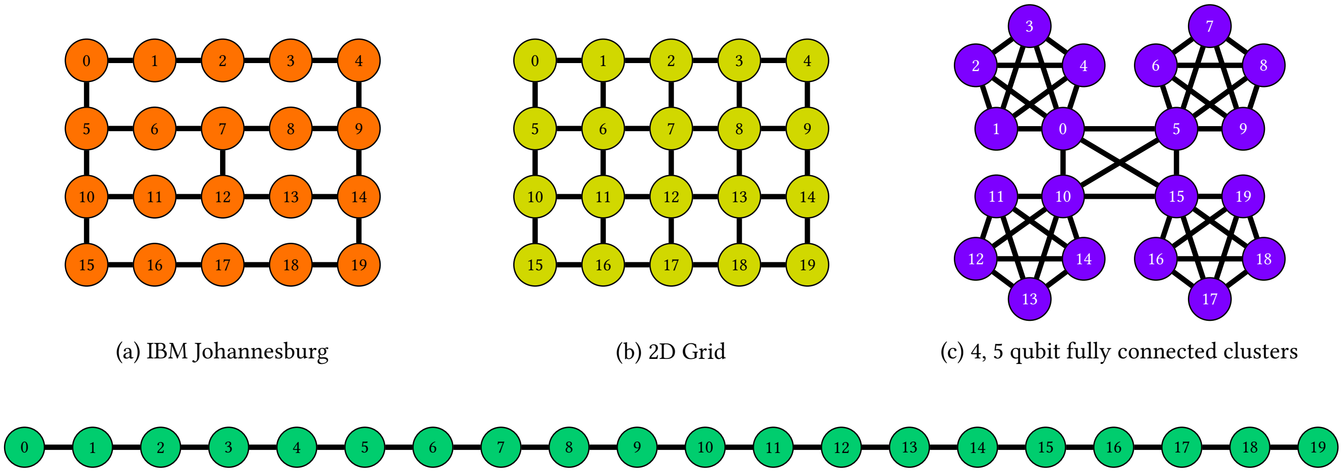
Figure 4: An 8-CNOT decomposition of the Toffoli gate.

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

2.3 Current Quantum Devices

In this paper we focus primarily on currently available superconducting quantum devices. This type of hardware is the primary focus of many industry players like IBM, Rigetti, and Google [1, 18, 34]. We show some representative topologies for superconducting devices in Figure 5abd. For completeness, we include a clustered device shown in Figure 5c representative of a QCCD ion trap device such as [22]. These systems exhibit all of the properties previously discussed. They have a small universal supported gate set which all programs must be transformed into and only support local two-qubit operations. The connectivity of these devices is given as a *coupling graph* specifying which pairs of qubits can execute CNOTs.

Furthermore, these systems are subject to a wide variety of noise which cause programs to fail. Some noise is due to manufacturing imperfections or calibration error. Some is inherent to quantum program execution resulting from the imperfect physical isolation of the qubits from the environment required to manipulate the quantum state [20]. In IBM machines, the experimental devices of this work, single qubit gate errors are small, occurring on average 1 in 2000 operations. CNOT gate errors are more significant occurring on roughly 1 in 100 gates. Measurement error is also significant, with errors on the same order of magnitude as CNOT gates. Finally, qubit lifetimes (coherence times) are relatively short, allowing on the order of 50 CNOT gate durations before the qubit state is lost [2] (but gates can often run in parallel while imposing additional



(a) IBM Johannesburg

(b) 2D Grid

(c) 4, 5 qubit fully connected clusters

(d) Linear

Figure 5: Example topologies of near-term quantum devices. Orange (a): IBM Johannesburg. Yellow (b): 2D Grid. Purple (c): four groups of five fully connected clusters. Green (d) Linear. Our real experiments run on Johannesburg and our simulations explore all of these topologies. Colors correspond with the bars in Figures 9, 10, 11.

crosstalk error). Therefore, quantum compilation is essential to reduce both of these sources of error: add as few extra gates as possible and minimize total execution time.

2.4 The Compilation Problem

In the NISQ era, quantum programs are highly optimized in order to reduce the effect on errors and maximize the probability the correct answer is observed. Similar to many classical programs, compilation uses a pass structure, where a set of transformation and optimizations are applied in a fixed order resulting in the compilation of an input quantum program to an executable for the target hardware [19, 26]. For the most part, these optimizations take place at the circuit-gate level. Some optimizations are hardware independent, for example, reducing total number of gates via commutativity-aware gate cancellation or find and replace with circuit identities. Other passes are focused on decomposing gates into the hardware's ISA [4, 21, 30].

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

Compilation involves three main steps. First, mapping program qubits to hardware qubits in order to minimize the total distance between qubits that will need to be close by in the future [23, 37, 38]. Second, routing pairs of CNOT inputs to be adjacent by inserting SWAPs [10, 17]. Finally, scheduling operations to minimize total execution time [16, 25]. In general, the compilation problem is computationally hard and while some attempts at optimal solutions have been pursued [33, 36, 40] the dominant approach is heuristics. In this work we focus on two pieces of this compilation problem: decomposition and routing.

IBM's Qiskit compiler, the standard for compiling programs to execute on an IBM device, has a default sequence of passes. First, all high level optimization and analysis passes are performed and all gates are unrolled and decomposed to the target gate set. Then single passes of mapping, routing, and scheduling are performed [3].

2.5 Evaluation Metrics

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

Program success rate is highly dependent 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.

2.6 Simulation

Simulating general quantum systems is exponentially expensive in the size of the system and therefore it is difficult to realistically model all of the errors during the execution of a quantum program. We use a simplified model for simulation to predict, specifically 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.

3 MOTIVATION: CONVENTIONAL COMPILATION

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

The default compilation framework in Qiskit used to transform input circuits to be executed on their hardware ensures a fully decomposed circuit before mapping, routing, and scheduling occur. As a simple example, consider three qubits placed fairly distant on IBM's Johannesburg device but for which we need to execute a Toffoli gate on them; as in Figure 1a. Qiskit decomposes this Toffoli as in Figure 3 with 6 CNOTs. Each CNOT acts on distant qubits so the many SWAPs inserted for all 6 CNOTs gets expensive quickly. When routing, we first SWAP the first interacting pair together (usually by adding SWAPs from control to target or the reverse, but a meet-in-the-middle strategy is also possible) and the qubit mapping is updated. The next CNOT is also distant so we add SWAPs to move them together and there is an even chance that the SWAPs for the second CNOT separate the two qubits that were just brought together.

Ideally, we move the third qubit to the already adjacent pair, but Qiskit cannot recognize this situation and could just as well move the other way. This is clearly sub-optimal and could continue on for the other four SWAPs. Even in the case where it makes the correct decision to move the distant third qubit, there are problems. Because pair of qubits needs to interact we may need single additional SWAPs as the qubits compete to be neighbors. This causes the 6-CNOT Toffoli decomposition to use many more than 6 CNOTs when there is not a triangle in the qubit connectivity graph. The core idea is that the routing strategy fails to take advantage of two things. First, it has effectively forgotten the desired operation is a Toffoli which will require all three qubits be adjacent and second that a more efficient Toffoli decomposition could be chosen which is more suitable for the underlying device architecture. In the example, inefficient compilation adds a total of 16 SWAPS or 48 CNOTs in total.

Some approaches in the past have attempted to solve the first of these problems, for example by using lookahead when choosing routing strategies [7, 39] and while this helps to treat the symptoms of pre-decomposing all operations it does not remedy the underlying problem.

4 ORCHESTRATED TRIOS

In this section we describe our proposed compilation structure compared to the conventional one as outlined in Figure 2. 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 3 is best, otherwise use the 8-CNOT Toffoli of Figure 4, ensuring the middle qubit is used for the middle of the decomposition (Any of the three qubits can be the target by simply moving the two H gates to that qubit).

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

This method can be easily extended to be noise-aware like previous work [23, 37] 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 [9], 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 EVALUATION

5.1 Toffoli Only Circuits

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

We study these circuits on a real IBM device, namely IBM Johannesburg, a 20-qubit device with limited connectivity in Figure 5a. We use the default Qiskit compiler which decomposes the Toffoli gates before doing shortest path routing compared to our proposed method where we do shortest path routing first and then decompose the Toffoli. We study the use of two different Toffoli implementations, a 6 CNOT decomposition with full qubit connectivity and an 8 CNOT decomposition with linear qubit connectivity.

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

5.2 NISQ Benchmarks and Quantum Subroutines

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

Table 1: Details about our benchmarks, both NISQ programs and other quantum subroutines

Benchmark	Qubits	Toffolis	CNOTs[1]
`cnx_dirty` [6]	11	16	128
`cnx_halfborrowed` [14]	19	32	256
`cnx_logancilla` [8]	19	17	136
`cnx_inplace` [14]	4	54	490
`cuccaro_adder` [11]	20	18	190
`takahashi_adder` [35]	20	18	188
`incrementer_borrowedbit` [14]	5	50	448
`grovers`[15]	9	84	672
`qft_adder` [29]	16	0	92
`bv` [9]	20	0	19
`qaoa_complete` [13]	10	0	90

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 1 using implementations found in [5]. 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 connectivity on the overall expected success rates and gate counts. We study four different connectivity models, all shown in Figure 5, each with 20 qubits, the topology of IBM's Johannesburg device containing four connected rings, a 2D mesh, a line, and a small clustered architecture representative of a QCCD ion trap.

[1] The total number of CNOT gates is after decomposition with the 8-CNOT Toffoli but does not including any SWAPs for routing.

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 [12]. 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.

6 RESULTS AND DISCUSSION

6.1 Trios Reduces Total Number of Gates

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

In Figure 7 we show 35 different triplets of hardware qubits for each of the four strategies. For each triplet, we note the total distance between the qubits on the hardware, given by the shortest path distance in the underlying topology. Even when the distance is relatively small, Trios outperforms 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 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 10 we note substantial reductions in total gates across all benchmarks containing Toffoli gates across all underlying topologies. The only exception is the two smallest benchmarks (on 4 and 5 qubits) for the clustered topology because they could be compiled with zero SWAPs.

An extreme of the clustered topology is a single cluster with all-to-all connected qubits. On this device, Orchestrated Trios would have no benefit as operations can be performed between any pair of qubits so no SWAPs are needed and routing is trivial. However, as quantum technologies scale to more than a few qubits, fully-connected architectures hits physical limitations and must be re-engineered. As trapped ion qubit chains get longer, for example, gate operations become slower and lower fidelity. [24] 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.

6.2 Trios Improves Overall Success Rate

In general, we expect programs with fewer total two-qubit gates, to succeed with higher probability. In devices with limited connectivity, the addition of routing operations like SWAPs, usually decomposed to 3 CNOTs, can severely reduce the chance an input program 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 6 we show the success rates of our Toffoli-only experiments when the two controls are initialized to $|1\rangle$ and the target is initialized to $|0\rangle$ so we measure the probability of obtaining $|111\rangle$. These results are obtained from Johannesburg on 8/19/2020. The x-axes of both Figures 6 and 7 line up to compare gate counts and resulting success rate. In general, experimentally, fewer gates results in substantial improvements to success rates. For example, a Toffoli on (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 8, we show improvements compiled with Trios normalized to baseline for 99 different triplets of varying total distance on Johannesburg.

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

For our simulated NISQ benchmarks, we see even larger gains. The reduced gate counts in Figure 10 translate to major improvements in simulated success rate in Figure 9 (normalized success rates in Figure 11). For example, in `cnx_logancilla-19`, Trios more than doubles the expected success rates when compiled to each of the architectures. In many cases, the expected success rate of programs compiled with Qiskit is effectively zero while Trios has a realistic chance of obtaining the correct answer. As expected, on programs containing no Toffoli gates, Trios has no effect on success showing that it introduces no excessive overhead. This suggests Trios can easily be added to other quantum compilation toolflows.

6.3 Trios Routes Complex Interactions Better

Trios improves gate counts, and consequently improves success rates, by routing more efficiently and choosing more appropriate Toffoli decompositions based on the underlying architecture's connectivity. Current compilers, like Qiskit, perform routing on fully decomposed and unrolled programs, and while this must eventually be done, it leads to less efficient routing policies and relies on assumptions that a theoretically good decomposition (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

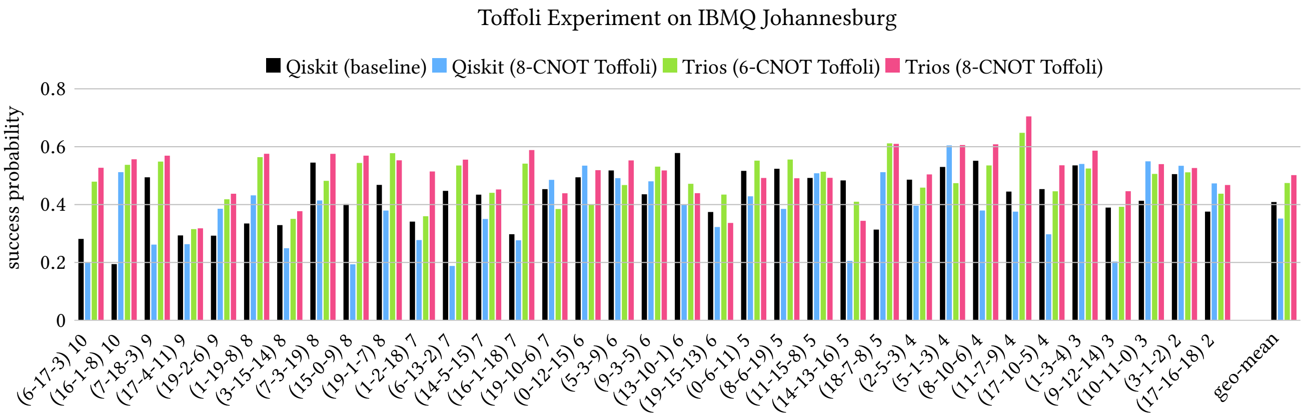


Figure 6: Success probabilities of Toffoli gates between random triplets of qubits. Higher is better. The x labels specify the three qubits and total swap distance. The geometric mean success rates for each compiler are 41%, 35%, 47%, and 50% respectively. Trios (8-CNOT) improves average success rate by 23% vs. the Qiskit baseline.

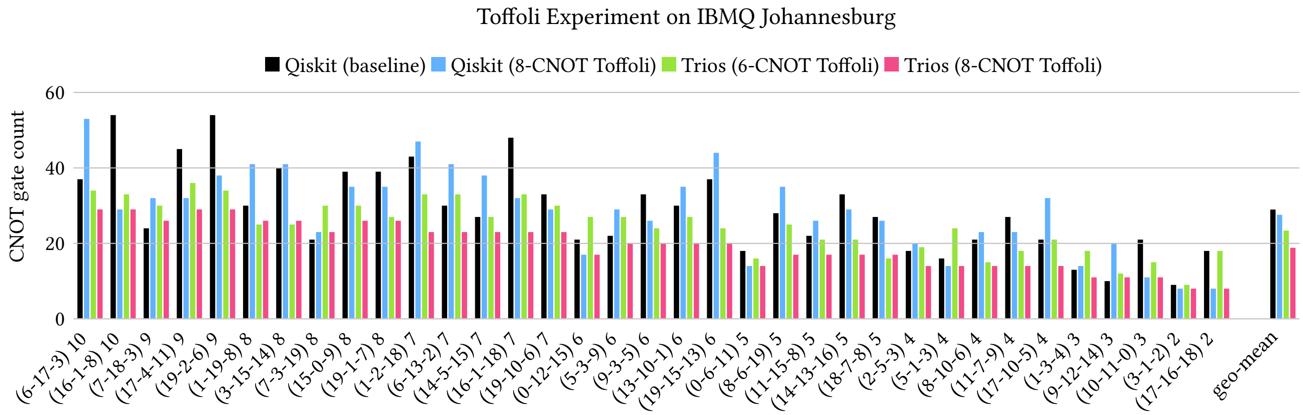


Figure 7: Total number of two-qubit (CNOT) gates required to execute a Toffoli gate between various distant qubits. Lower is better. The x labels specify the three qubits and total swap distance. The geometric mean gate counts for each compiler are 29, 28, 23, and 19 respectively. Trios (8-CNOT) reduces average gate count by 35%.

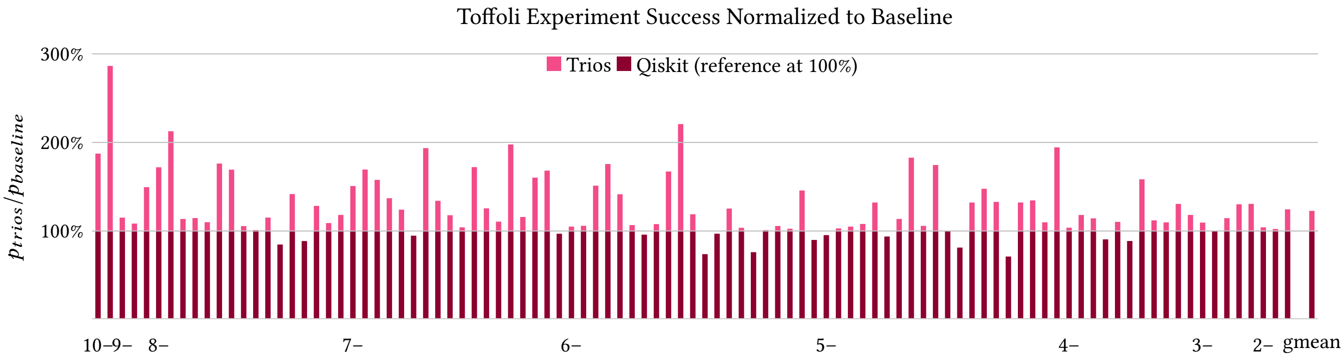


Figure 8: Normalized success probabilities of Toffoli gates between triplets of qubits. Higher is better. Bars below 100% indicate lower success rate for Trios. The geometric mean increase in success rate is 23%. The x labels indicate the qubit distance for a range of bars.

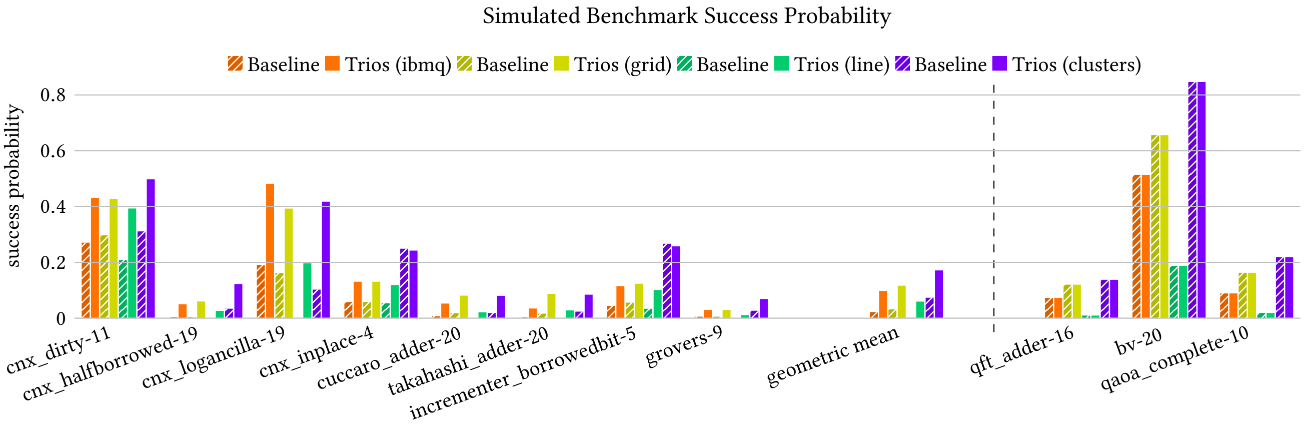


Figure 9: Simulated upper-bounds on the program execution success probability on various hardware (using 20x lower idle and gate errors than Johannesburg). Neighboring pairs of bars compare the baseline with Trios compiled for Johannesburg. Higher is better when comparing pairs of bars with the same color. The geometric mean success rates over the benchmarks that use Toffoli gate for each device type respectively are 2.2%→9.8%, 3.2%→12%, 0.19%→6.0%, 7.3%→17%. The rightmost three benchmarks contain zero Toffoli gates so have no change vs. the baseline.

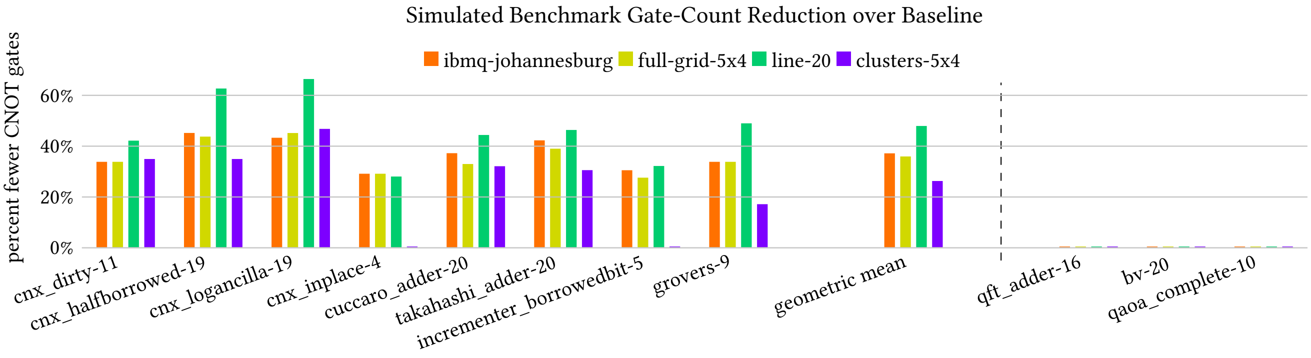


Figure 10: A comparison between the baseline and Trios for various hardware. Above 0% indicates benefit. All two-qubit gates (for communication and computation) are counted. The geometric mean reductions in gate counts are 37%, 36%, 48%, and 26% respectively. The rightmost three benchmarks contain zero Toffoli gates so have no change vs. the baseline.

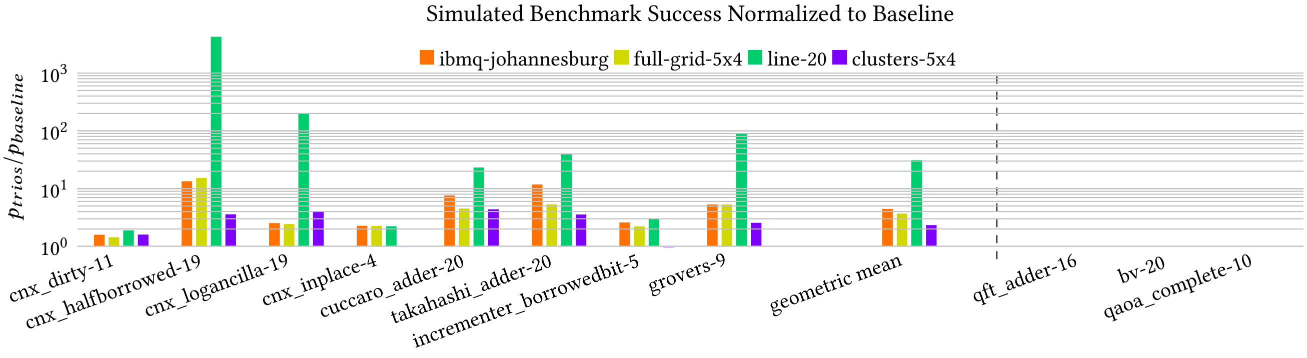


Figure 11: Normalized Figure 9 to show our consistent increase in program success with Trios. Above 10^0 indicates benefit. Some improvement factors are huge due to near-zero baseline success rates. The geometric mean increases in success rate are 4.4x, 3.7x, 31x, and 2.3x respectively. The rightmost three benchmarks contain zero Toffoli gates so have no change vs. the baseline.

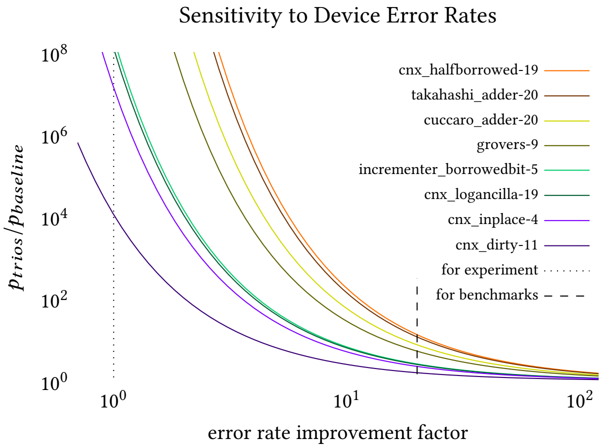


Figure 12: Factor of improvement in success rate in Trios over baseline for scaling gate error rates. The dotted line indicates current error rates on IBM Johannesburg and the dashed line (20x improvement) indicates values of the near future used in simulation. In our approximation of success rate factors of improvement in gate error rates lead to an exponential fall off in success ratios, as expected. In the very near term, we expect Trios to drastically improve the execution of quantum programs.

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.

6.4 Simulation Sensitivity to Error Rates

For our simulations we use an error model (20x better than current errors on Johannesburg) which is forward looking. As errors improve, we expect Trios to have a reduced impact on program success rates since gate errors will contribute less and less to program failure though Trios will never perform worse than the baseline. In Figure 12 we study the sensitivity of simulation results to two qubit error rates beginning with current IBM error rates. For poor error rates, the benefit of Trios is extremely large, owed to the fact that programs compiled with the baseline have probabilities of success very close to 0. In our simplified simulation framework, as error rates improve we expect an exponential drop off in improvement with the most advantage obtained with current error rates.

7 CONCLUSION

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.

ACKNOWLEDGMENTS

This work is funded in part by EPiQC, an NSF Expedition in Computing, under grants CCF-1730449; in part by STAQ under grant NSF Phy-1818914; in part by DOE grants DE-SC0020289 and DE-SC0020331; and in part by NSF OMA-2016136 and the Q-NEXT DOE NQI Center.

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

Disclosure: F. Chong is also Chief Scientist at Super.tech and an advisor to Quantum Circuits, Inc.

REFERENCES

[1] 2018. A Preview of Bristlecone, Google's New Quantum Processor. https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html

[2] 2019. Cramming More Power Into a Quantum Device. https://www.ibm.com/blogs/research/2019/03/power-quantum-device/

[3] Héctor Abraham, AduOffei, Rochisha Agarwal, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Eli Arbel, Abraham Asfaw, Carlos Azaustre, AzizNgoueya, Aman Bansal, Panagiotis Barkoutsos, George Barron, Luciano Bello, Yael Ben-Haim, Daniel Bevenius, Lev S. Bishop, Sorin Bolos, Samuel Bosch, Sergey Bravyi, David Bucher, Artemiy Burov, Fran Cabrera, Padraic Calpin, Lauren Capelluto, Jorge Carballo, Ginés Carrascal, Adrian Chen, Chun-Fu Chen, Edward Chen, Jielun (Chris) Chen, Richard Chen, Jerry M. Chow, Spencer Churchill, Christian Claus, Christian Clauss, Romilly Cocking, Abigail J. Cross, Andrew W. Cross, Simon Cross, Juan Cruz-Benito, Chris Culver, Antonio D. Córcoles-Gonzales, Sean Dague, Tareq El Dandachi, Marcus Daniels, Matthieu Dartiailh, DavideFrr, Abdón Rodríguez Davila, Anton Dekusar, Delton Ding, Jun Doi, Eric Drechsler, Drew, Eugene Dumitrescu, Karel Dumon, Ivan Duran, Kareem EL-Safty, Eric Eastman, Pieter Eendebak, Daniel Egger, Mark Everitt, Paco Martín Fernández, Axel Hernández Ferrera, Romain Fouilland, FranckChevallier, Albert Frisch, Andreas Fuhrer, MELVIN GEORGE, Julien Gacon, Borja Godoy Gago, Claudio Gambella, Jay M. Gambetta, Adhisha Gammanpila, Luis Garcia, Shelly Garion, Austin Gilliam, Aditya Giridharan, Juan Gomez-Mosquera, Salvador de la Puente González, Jesse Gorzinski, Ian Gould, Donny Greenberg, Dmitry Grinko, Wen Guan, John A. Gunnels, Mikael Haglund, Isabel Haide, Ikko Hamamura, Omar Costa Hamido, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Stefan Hillmich, Hiroshi Horii, Connor Howington, Shaohan Hu, Wei Hu, Rolf Huisman, Haruki Imai, Takashi Imamichi, Kazuaki Ishizaki, Raban Iten, Toshinari Itoko, JamesSeaward, Ali Javadi, Ali Javadi-Abhari, Jessica, Madhav Jivrajani, Kiran Johns, Jonathan-Shoemaker, Tal Kachmann, Naoki Kanazawa, Kang-Bae, Anton Karazeev, Paul Kassebaum, Spencer King, Knabberjoe, Yuri Kobayashi, Arseny Kovyrshin, Rajiv Krishnakumar, Vivek Krishnan, Kevin Krsulich, Gawel Kus, Ryan LaRose, Enrique Lacal, Raphaël Lambert, John Lapeyre, Joe Latone, Scott Lawrence, Christina Lee, Gushu Li, Dennis Liu, Peng Liu, Yunho Maeng, Aleksei Malyshev, Joshua Manela, Jakub Marecek, Manoel Marques, Dmitri Maslov, Dolph Mathews, Atsushi Matsuo, Douglas T. McClure, Cameron McGarry, David McKay, Dan McPherson, Srujan Meesala, Thomas Metcalfe, Martin Mevissen, Antonio Mezzacapo, Rohit Midha, Zlatko Minev, Abby Mitchell, Nikolaj Moll, Michael Duane Mooring, Renier Morales, Niall Moran, MrF, Prakash Murali, Jan Müggenburg, David Nadlinger, Ken Nakanishi, Giacomo Nannicini, Paul Nation, Edwin Navarro, Yehuda Naveh, Scott Wyman Neagle, Patrick Neuweiler, Pradeep Niroula, Hassi Norlen, Lee James O'Riordan, Oluwatobi Ogunbayo, Pauline Ollitrault, Steven Oud, Dan Padilha, Hanhee

Paik, Yuchen Pang, Simone Perriello, Anna Phan, Francesco Piro, Marco Pistoia, Christophe Piveteau, Alejandro Pozas-iKerstjens, Viktor Prutyanov, Daniel Puzzuoli, Jesús Pérez, Quintiii, Rafey Iqbal Rahman, Arun Raja, Nipun Ramagiri, Anirudh Rao, Rudy Raymond, Rafael Martín-Cuevas Redondo, Max Reuter, Julia Rice, Diego M. Rodríguez, RohithKarur, Max Rossmannek, Mingi Ryu, Tharrmashastha SAPV, SamFerracin, Martin Sandberg, Ritvik Sapra, Hayk Sargsyan, Aniruddha Sarkar, Ninad Sathaye, Bruno Schmitt, Chris Schnabel, Zachary Schoenfeld, Travis L. Scholten, Eddie Schoute, Joachim Schwarm, Ismael Faro Sertage, Kanav Setia, Nathan Shammah, Yunong Shi, Adenilton Silva, Andrea Simonetto, Nick Singstock, Yukio Siraichi, Iskandar Sitdikov, Seyon Sivarajah, Magnus Berg Sletfjerding, John A. Smolin, Mathias Soeken, Igor Olegovich Sokolov, SooluThomas, Starfish, Dominik Steenken, Matt Stypulkoski, Shaojun Sun, Kevin J. Sung, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Soolu Thomas, Mathieu Tillet, Maddy Tod, Miroslav Tomasik, Enrique de la Torre, Kenso Trabing, Matthew Treinish, TrishaPe, Wes Turner, Yotam Vaknin, Carmen Recio Valcarce, Francois Varchon, Almudena Carrera Vazquez, Victor Villar, Desiree Vogt-Lee, Christophe Vuillot, James Weaver, Rafal Wieczorek, Jonathan A. Wildstrom, Erick Winston, Jack J. Woehr, Stefan Woerner, Ryan Woo, Christopher J. Wood, Ryan Wood, Stephen Wood, Steve Wood, James Wootton, Daniyar Yeralin, David Yonge-Mallo, Richard Young, Jessie Yu, Christopher Zachow, Laura Zdanski, Helena Zhang, Christa Zoufal, Zoufalc, a kapila, a matsuo, bcamorrison, brandhsn, chlorophyll zz, dekel.meirom, dekool, dime10, drholmie, dtrenev, ehchen, elfrocampeador, faisaldebouni, fanizzamarco, gadial, gruu, hhorii, hykavitha, jagunther, jliu45, kanejess, klinvill, kurarrr, lerongil, ma5x, merav aharoni, michelle4654, ordmoj, rmoyard, saswati qiskit, sethmerkel, strickroman, sumitpuri, tigerjack, toural, vvilpas, welien, willhbang, yang.luh, yotamvakninibm, and Mantas Čepulkovskis. 2019. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2562110

[4] Matthew Amy. 2013. *Algorithms for the Optimization of Quantum Circuits.* Master's thesis. University of Waterloo. http://hdl.handle.net/10012/7818

[5] Jonathan M. Baker, Casey Duckering, Pranav Gokhale, and Andrew Litteken. 2020. Quantum Circuit Benchmarks. https://github.com/jmbaker94/quantumcircuitbenchmarks.

[6] Jonathan M. Baker, Casey Duckering, Alexander Hoover, and Frederic T. Chong. 2019. Decomposing Quantum Generalized Toffoli with an Arbitrary Number of Ancilla. *arXiv preprint* (April 2019). arXiv:1904.01671

[7] Jonathan M. Baker, Casey Duckering, Alexander Hoover, and Frederic T. Chong. 2020. Time-Sliced Quantum Circuit Partitioning for Modular Architectures. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*. 98–107. https://doi.org/10.1145/3387902.3392617 arXiv:2005.12259

[8] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. 1995. Elementary gates for quantum computation. *Physical Review A* 52, 5 (Nov. 1995), 3457–3467. https://doi.org/10.1103/PhysRevA.52.3457 arXiv:quant-ph/9503016

[9] Ethan Bernstein and Umesh Vazirani. 1993. Quantum Complexity Theory. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing* (San Diego, California, USA) *(STOC '93)*. 11–20. https://doi.org/10.1145/167088.167097

[10] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. 2019. On the Qubit Routing Problem. 135 (Feb. 2019), 5:1–5:32. https://doi.org/10.4230/LIPIcs.TQC.2019.5 arXiv:1902.08091

[11] Steven A. Cuccaro, Thomas G. Draper, Samuel A. Kutin, and David Petrie Moulton. 2004. A new quantum ripple-carry addition circuit. *arXiv preprint* (Oct. 2004). arXiv:quant-ph/0410184

[12] Casey Duckering, Jonathan M. Baker, and Andrew Litteken. 2021. Source Code for Orchestrated Trios. https://github.com/cduck/orchestrated-trios.

[13] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A Quantum Approximate Optimization Algorithm. *arXiv preprint* (Nov. 2014). arXiv:1411.4028

[14] Craig Gidney. [n.d.]. Constructing Large Controlled Nots. https://algassert.com/circuits/2015/06/05/Constructing-Large-Controlled-Nots.html

[15] Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. 212–219. https://doi.org/10.1145/237814.237866 arXiv:quant-ph/9605043

[16] Gian Giacomo Guerreschi and Jongsoo Park. 2018. Two-step approach to scheduling quantum circuits. *Quantum Science and Technology* 3, 4 (July 2018), 045003. https://doi.org/10.1088/2058-9565/aacf0b arXiv:1708.00023

[17] Yuichi Hirata, Masaki Nakanishi, Shigeru Yamashita, and Yasuhiko Nakashima. 2011. An efficient conversion of quantum circuits to a linear nearest neighbor architecture. *Quantum Information and Computation* 11, 1 (Jan. 2011), 142–166.

[18] ibm0 [n.d.]. IBM Quantum Devices. https://quantumexperience.ng.bluemix.net/qx/devices. Accessed: 2018-05-16.

[19] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. 2015. ScaffCC: Scalable compilation and analysis of quantum programs. *Parallel Comput.* 45 (2015), 2–17. https://doi.org/10.1016/j.parco.2014.12.001 arXiv:1507.01902

[20] Philip Krantz, Morten Kjaergaard, Fei Yan, Terry P. Orlando, Simon Gustavsson, and William D. Oliver. 2019. A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews* 6, 2 (June 2019), 021318. https://doi.org/10.1063/1.5089550 arXiv:1904.06560

[21] Dmitri Maslov, Gerhard W. Dueck, D. Michael Miller, and Camille Negrevergne. 2008. Quantum Circuit Simplification and Level Compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 3 (March 2008), 436–444. https://doi.org/10.1109/TCAD.2007.911334 arXiv:quant-ph/0604001

[22] Steven Moses, Juan Pino, Joan Dreiling, Caroline Figgatt, John Gaebler, Michael Allman, Charles Baldwin, Michael Foss-Feig, David Hayes, Karl Mayer, Ciaran Ryan-Anderson, and Brian Neyenhuis. 2020. Demonstration of the QCCD trapped-ion quantum computer architecture. *Bulletin of the American Physical Society* (June 2020). arXiv:2003.01293

[23] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. 2019. 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*. 1015–1029. https://doi.org/10.1145/3297858.3304075 arXiv:1901.11054

[24] Prakash Murali, Dripto M. Debroy, Kenneth R. Brown, and Margaret Martonosi. 2020. Architecting Noisy Intermediate-Scale Trapped Ion Quantum Computers. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 529–542. https://doi.org/10.1109/ISCA45697.2020.00051 arXiv:2004.04706

[25] Prakash Murali, David C. McKay, Margaret Martonosi, and Ali Javadi-Abhari. 2020. 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*. 1001–1016. https://doi.org/10.1145/3373376.3378477 arXiv:2001.02826

[26] Yunseong Nam, Neil J. Ross, Yuan Su, Andrew M. Childs, and Dmitri Maslov. 2018. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information* 4, 1 (March 2018), 1–12. https://doi.org/10.1038/s41534-018-0072-4 arXiv:1710.07345

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

[28] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (Aug. 2018). https://doi.org/10.22331/q-2018-08-06-79 arXiv:1801.00862

[29] Lidia Ruiz-Perez and Juan Carlos Garcia-Escartin. 2017. Quantum arithmetic with the quantum Fourier transform. *Quantum Information Processing* 16, 6 (April 2017), 152:1–152:14. https://doi.org/10.1007/s11128-017-1603-1 arXiv:1411.5949

[30] Zahra Sasanian and D. Michael Miller. 2012. Reversible and Quantum Circuit Optimization: A Functional Approach. In *International Workshop on Reversible Computation*. Springer Berlin Heidelberg, 112–124. https://doi.org/10.1007/978-3-642-36315-3_9

[31] Norbert Schuch. 2002. Implementation of quantum algorithms with Josephson charge qubits. *Universität Regensburg* (Dec. 2002). https://epub.uni-regensburg.de/1511/

[32] Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (1997), 1484–1509. https://doi.org/10.1137/S0097539795293172 arXiv:quant-ph/9508027

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

[34] Robert S. Smith, Michael J. Curtis, and William J. Zeng. 2016. A Practical Quantum Instruction Set Architecture. *arXiv preprint* (2016). arXiv:1608.03355

[35] Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. 2009. Quantum Addition Circuits and Unbounded Fan-Out. *arXiv preprint* (Oct. 2009). arXiv:0910.2530

[36] Bochen Tan and Jason Cong. 2020. Optimal Layout Synthesis for Quantum Computing. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE. arXiv:2007.15671

[37] Swamit S. Tannu and Moinuddin Qureshi. 2019. 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*. 253–265. https://doi.org/10.1145/3352460.3358257

[38] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. 2019. Mapping Quantum Circuits to IBM QX Architectures Using the Minimal Number of SWAP and H Operations. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE. arXiv:1907.02026

[39] Robert Wille, Oliver Keszocze, Marcel Walter, Patrick Rohrs, Anupam Chattopadhyay, and Rolf Drechsler. 2016. 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)*. IEEE, 292–297. https://doi.org/10.1109/ASPDAC.2016.7428026

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