Reoptimization of Quantum Circuits via Hierarchical Synthesis

Xin-Chuan Wu
Intel Labs
Intel Corporation
Santa Clara, CA 95054, USA
xin-chuan.wu@intel.com

Marc Grau Davis
Department of Computer Science
MIT
Cambridge, MA 02139, USA
mgd@mit.edu

Frederic T. Chong
Department of Computer Science
University of Chicago
Chicago, IL 60637, USA
chong@cs.uchicago.edu

Costin Iancu
Computational Research Division
Lawrence Berkeley National Laboratory
Berkeley, CA 94720, USA
cciancu@lbl.gov

***Abstract*—The current phase of quantum computing is in the Noisy Intermediate-Scale Quantum (NISQ) era. On NISQ devices, two-qubit gates such as CNOTs are much noisier than single-qubit gates, so it is essential to minimize their count. Quantum circuit synthesis is a process of decomposing an arbitrary unitary into a sequence of quantum gates, and can be used as an optimization tool to produce shorter circuits to improve overall circuit fidelity. However, the time-to-solution of synthesis grows exponentially with the number of qubits. As a result, synthesis is intractable for circuits on a large qubit scale.**

In this paper, we propose a hierarchical, block-by-block optimization framework, QGo, for quantum circuit optimization. Our approach allows an exponential cost optimization to scale to large circuits. QGo uses a combination of partitioning and synthesis: 1) partition the circuit into a sequence of independent circuit blocks; 2) re-generate and optimize each block using quantum synthesis; and 3) re-compose the final circuit by stitching all the blocks together. We perform our analysis and show the fidelity improvements in three different regimes: small-size circuits on real devices, medium-size circuits on noisy simulations, and large-size circuits on analytical models. Our technique can be applied after existing optimizations to achieve higher circuit fidelity. Using a set of NISQ benchmarks, we show that QGo can reduce the number of CNOT gates by 29.9% on average and up to 50% when compared with industrial compiler optimizations such as *t*|*ket*⟩. When executed on the IBM Athens system, shorter depth leads to higher circuit fidelity. We also demonstrate the scalability of our QGo technique to optimize circuits of 60+ qubits. Our technique is the first demonstration of successfully employing and scaling synthesis in the compilation toolchain for large circuits. Overall, our approach is robust for direct incorporation in production compiler toolchains to further improve the circuit fidelity.

***Index Terms*—Quantum Computing, Optimization, Synthesis, Quantum Compiler**

I. Introduction

Quantum Computing (QC) is expected to solve certain computational problems that even the largest classical supercomputers cannot solve. QC algorithms might have significant impact on areas such as quantum chemistry [44], [57], cryptography [63], and machine learning [12]. Recently, QC devices up to 72 quantum bits (qubits) have been demonstrated by IBM and Google [36], [39]. The current phase of QC is in the Noisy Intermediate-Scale Quantum (NISQ) era [58]. On NISQ devices, quantum gates are noisy and their count must be minimized to produce low-error circuits. In particular, two-qubit gates such as CNOTs are much noisier than single-qubit gates. For NISQ devices, shorter depth translates directly into higher circuit fidelity. Most of the existing compilers focus on optimizing the qubit mapping and swap insertion to reduce circuit depth [41], [47], [73].

Quantum circuit synthesis provides an orthogonal circuit optimization method, which generates a circuit from its high level mathematical description such as unitary matrix. There are several existing studies in this field [1], [6], [13], [23], [24], [26], [27], [32], [45], [50], [53], [60], [67], [72]. Given an arbitrary quantum circuit, we can compute the corresponding unitary, and then use synthesis to generate a new circuit, which has a different sequence of gates from the original circuit, but will be equivalent in terms of unitary operator performed. This approach can work as a reopotimization process to further improve the circuit fidelity.

However, synthesis algorithms face an "exponential" scalability challenge. For a n-qubit circuit, the unitary size is $2^n \times 2^n$. The solving time of synthesis scales exponentially with the number of qubits [1], [23], [27]. Thus, synthesis is intractable for circuits beyond a handful of qubits.

To perform quantum synthesis for optimizing large scale circuits, we propose our hierarchical, block-by-block, optimization framework, called QGo. QGo uses a combination of partitioning and synthesis: 1) partition the circuit into independent sub-blocks whose size can be successfully handled by synthesis; 2) re-generate each block using synthesis; and 3) re-assemble the circuit. Figure 1 shows an example of QGo circuit optimization that partitions a 5-qubit circuit into 3-qubit blocks. In general, partitioning a n-qubit circuit into multiple

k-qubit blocks ($k < n$) leads to replacing an algorithm with $O(exp(n))$ complexity with a sequence of calls to $O(exp(k))$ algorithms. The time to synthesized solution grows linearly with the number of k-qubit blocks, which is a useful tradeoff when compared to the exponential scaling with the number of qubits. After all blocks are synthesized, QGo then puts the blocks back together to produce the final optimized circuit.

QGo allows us to compile a quantum circuit to an optimized executable circuit for a target hardware. QGo is a topology-aware compilation framework. The overall flow is described in Figure 2. QGo includes four core modules: physical qubit mapping, circuit partitioning, quantum synthesis, and circuit composition. The input to QGo is a quantum circuit in a high-level quantum program, together with a description of the topology of the target processor. The first step in QGo is topology mapping of the circuit to the target hardware. We leverage third party mappers and call into "traditional" compilers to perform mapping. This approach allows us to select the best mapper available for a given platform. We use the *t|ket⟩* compiler for physical qubit mapping as it is reported to effectively produce short circuits for several applications [20], [66]. We then run our partitioning algorithm to select tunable size blocks of the circuit that are independent of each other. For large-scale circuit partitioning, we use a greedy-based heuristic approach that partitions a circuit by selecting blocks with high CNOT-count. In the synthesis procedure, each block is converted from its circuit format into a unitary matrix and re-synthesized with a synthesis tool. In this work, we use the state-of-the-art synthesis tool proposed in [23]. The last step is circuit composition. If the synthesized block has more or equal number of CNOTs, QGo will select the original block for the final circuit. The final output of QGo is the optimized circuit by stitching all the optimized blocks together.

For evaluation we use a series of NISQ circuits as our benchmarks. To understand the impact of the partitioning strategy we conduct experiments with 3- and 4-qubit blocks. We evaluate the quality of the generated circuits on IBM's Athens [40] device, as well as using noise simulation for medium-size circuits and on our analytical model for large-size circuits.

Compared with existing synthesis approaches, our hierarchical synthesis guarantees to reduce or keep the same CNOT count for arbitrary gate sets in large-scale circuits within a feasible compilation time.

The main contributions of this work are:

- Circuit fidelity improvement is critical. Our approach provides a next-level reoptimization that is robust for direct incorporation in existing compiler tools. Our technique allows an exponential cost optimization tool to scale to large circuits, and is the first demonstration of successfully employing and scaling synthesis in the compilation toolchain for large circuit optimization.
- QGo reduces the CNOT gate count well beyond the ability of existing compilers. Using a set of NISQ benchmarks, we show that QGo can reduce the number of CNOT gates by 29.9% on average and up to 50% compared with circuits optimized by *t|ket⟩* compiler. These translate into direct fidelity improvements when running on the IBM's Athens device [21]. It was not obvious that selecting blocks with high CNOT-count and synthesizing them would yield significant fidelity improvements.
- We evaluate fidelity improvements using 3 metrics for 3 scaling regimes: small-size circuits using fidelity on real devices, medium-size circuits using fidelity using simulations with noise, and large-size circuits using CNOT reduction measured statically by our compiler. To validate the trends shown by our simulations, we show that there is a 98% correlation between our real-device results and our simulation results on small circuits.
- We present the sensitivity analysis by running Qiskit noise simulations to show the circuit fidelity improvements under different levels of gate errors. The results indicate that QGo is important for NISQ devices.
- We demonstrate the scalability of our technique to optimize the circuits of 60+ qubits. This demonstration suggests that QGo provides a viable path towards higher fidelity for circuits on large scale qubits.

II. Background

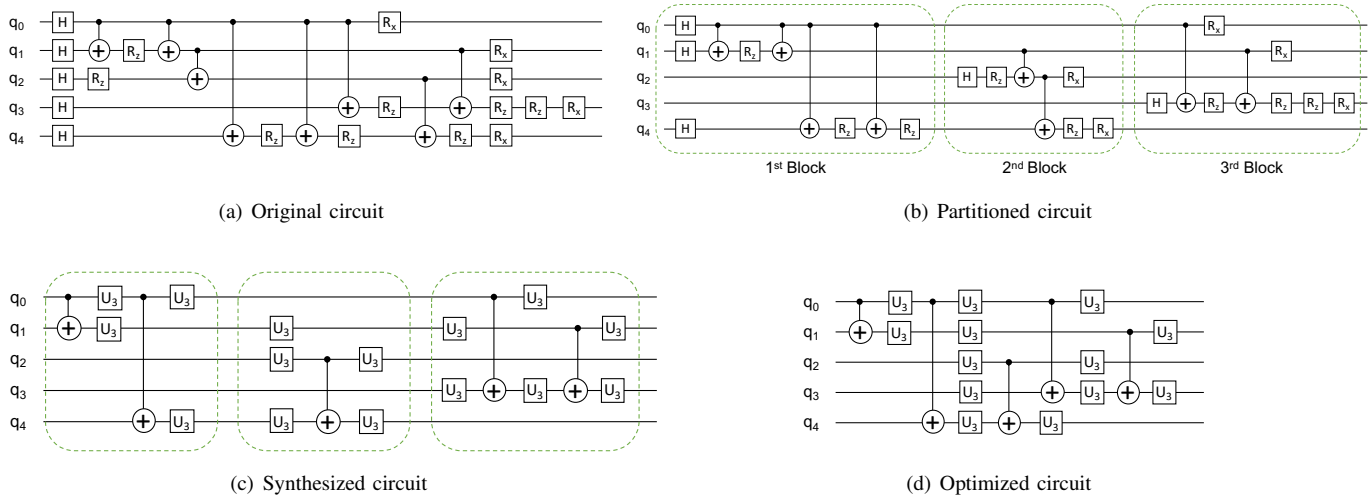
A. Principles of Quantum Computation

A qubit is a two-level quantum system, represented by two orthonormal computational basis states $|0\rangle$ and $|1\rangle$. The quantum state of a qubit can be described by any linear combination of $|0\rangle$ and $|1\rangle$: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers satisfying $|\alpha|^2 + |\beta|^2 = 1$. More generally, the state of a n-qubit quantum system can be represented by using 2^n amplitudes.

A quantum circuit consists of a sequence of quantum gates on qubits, and a quantum gate is a unitary operator, U. The effect of a gate on a quantum state can be expressed as $|\psi'\rangle = U|\psi\rangle$. Gates are applied to one or many qubits simultaneously. Two-qubit controlled gates and arbitrary single-qubit gates are known to be universal [29]. CNOT gates and single-qubit rotation gates (R_x, R_y, R_z) constitute a commonly used universal gate set for quantum programming.

B. Quantum Circuit Synthesis

Quantum circuit synthesis is the process of taking a description of a desired unitary matrix and decomposing it into a sequence of smaller unitaries, representing the gates in a circuit that implements the target unitary. One of the earliest techniques was the Solovay-Kitaev algorithm [1], [24], [53], which combined a recursive matrix decomposition technique with numerical methods required for the base case. This method served as a proof-of-concept for the field of synthesis, but the circuits it produces are very long. One of the main goals of synthesis is to produce "short" circuits, minimizing metrics such as CNOT count, total gate count, and critical path length. CNOT count is of particular importance on NISQ devices, since they contribute significantly more to the overall noise and runtime of circuits than single-qubit gates. Because



(a) Original circuit

(b) Partitioned circuit

(c) Synthesized circuit

(d) Optimized circuit

Fig. 1. An example of QGo circuit optimization. (a) The original 5-qubit circuit. (b)The original circuit is partitioned into three blocks, and each block only contains gates on 3 qubits. The first circuit block is only on q_0, q_1, and q_4. The second block is on q_1, q_2, and q_4. The third block is on q_0, q_1, and q_3. All gates after partitioning still respect the gate dependency in the original circuit. (c) The synthesized circuit. After running quantum synthesis for each block, the CNOT counts in the first and second block are reduced, and the third block still has 2 CNOTs. (d) The single-qubit gates can be combined to produce the final optimized circuit. This circuit is effectively equivalent to the original circuit but with fewer CNOT gates.

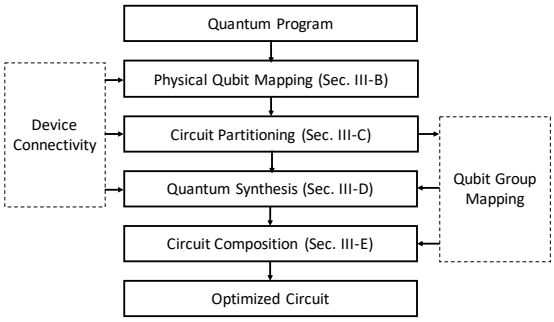


Fig. 2. Our compilation framework for scalable circuit optimization using synthesis (QGo).

of this, CNOT count has been a particular target for synthesis approaches. The approach shown in [6] is able to find minimal length circuits according to a weighting of different gates, and demonstrates a significant advantage over the Solovay-Kitaev approach. The approaches shown in [45] and [27] focus on synthesis runtime, which is another metric needed for synthesis to be practical. All of these techniques mentioned so far work solely with discrete gatesets, where only a finite number of base gates are allowed. However, most current quantum devices can perform continuously varying gates. For example, superconducting qubits use virtual Z rotations, which can be performed with any angle, and trapped ion qubits can perform X and Y rotations with any angle. These continuous gates can be used to perform any single qubit rotation, usually parameterized as $Z(\theta_1)X(\frac{\pi}{2})Z(\theta_2)X(\frac{\pi}{2})Z(\theta_3)$ or $X(\theta_1)Y(\theta_2)Z(\theta_3)$. Leveraging this capability, the KAK decomposition can perform optimal 2-qubit synthesis to perform any 2 qubit unitary with at most 3 CNOTs [67]. The approach shown in [60] uses the cosine-sine decomposition to produce universal gates for any number of qubits, which can perform any unitary simply by changing the parameters. Other approaches use numerical optimization of parameterized gates to find high quality circuits [23], [50], [72].

We employ the search-based numerical synthesis method described in [23]. This technique builds a search tree of CNOT structures filled in with parameterized single-qubit gates, and employs numerical optimization to evaluate the CNOT structure at each node in the search. A* search is used to find the shortest CNOT structure that can implement the desired unitary, and numerical optimization is used to find parameters for the single-qubit gates to produce a fully instantiated circuit. We choose this technique because of its ability to minimize CNOT count.

III. Scalable Circuit Optimization using Synthesis

A. QGo Overview

Figure 2 shows an overview of QGo. It contains four core compiler components: physical qubit mapping, circuit partitioning, quantum synthesis, and the circuit composition procedure. The inputs for QGo consist of a high-level quantum program and a target device connectivity description. First, QGo performs physical qubit mapping to assign the logical qubits to physical qubits and to resolve all unexecutable two-qubit gates by adding swap gates. Since our goal is to produce an optimized topology-aware circuit, if we perform physical qubit mapping after quantum synthesis, there will be swap gates inevitably added into the final circuit. Therefore, we perform physical qubit mapping before quantum synthesis, so that the swap gates added into the circuit become a part of the input circuit for the circuit synthesis, and hence the final circuit would have a reduced CNOT gate count. Second, QGo partitions the circuit into multiple small circuit blocks.

Each block only interacts with a few qubits. The qubit group mapping is generated to indicate what qubits are involved for each circuit block. This qubit group mapping information will be used for the later circuit composition procedure. Since the number of qubits in a block reflects the size of the corresponding unitary matrix, we demonstrate the partitioning of 3-qubit and 4-qubit blocks in this work, but the block size can be further increased. In general, a larger unitary matrix would take exponentially longer time to decompose into a sequence of gates. A circuit partitioned with a larger block size would have fewer blocks, and each block tends to allow the synthesizer to decide circuit elements rather than the mapping algorithm, and hence the final optimized circuit would have fewer CNOTs. Thus, the block size is a trade-off between the time-to-solution and the quality of solution (CNOT reduction). Next, quantum synthesis is applied to the partitioned circuit blocks. We use the state-of-the-art synthesis tool proposed in [23] for our work. Finally, QGo performs the circuit composition by following the qubit group mapping to concatenate the synthesized blocks to produce the final optimized circuit.

B. Physical Qubit Mapping

Given an input quantum circuit and the qubit connectivity graph, physical qubit mapping is to find an initial qubit mapping and insert swap gates to satisfy all two-qubit interactions and try to minimize the total number of swap gates and circuit depth in the final hardware-compliant circuit. Physical qubit mapping problem is NP-Complete [64]. Several approaches for solving this problem have been proposed [9], [16], [21], [41], [47], [49], [51], [56], [66], [70], [71], [73].

We design the physical qubit mapping as the first step of QGo because we want to have the additional swaps gates to be part of the input circuit for the remaining optimization process, so that the optimization by quantum synthesis can reduce the CNOT gate count in the final circuit. In addition, we also find that most quantum applications have certain repeated CNOT patterns, and existing qubit mapping algorithms will perform better by leveraging the structure of these patterns. If the synthesis process runs before physical qubit mapping, the pattern will be destroyed, and the number of additional swap gates will increase in some cases. Thus, performing physical qubit mapping before synthesis can benefit the overall optimization.

We compare industrial compilers such as IBM Qiskit [21] and $t|ket\rangle$ compilers [66]. Since the $t|ket\rangle$ compiler produces shorter circuits in our experiments and is reported to produce shorter circuits for several quantum applications compared with other techniques [20], we adopt the $t|ket\rangle$ compiler for physical qubit mapping in our QGo.

C. Circuit Partitioning

QGo partitions a circuit into multiple small circuit blocks. Each block contains gates only on a small group of qubits. Figure 1 shows an example of partitioning. Each circuit block consists of 3 qubits (Figure 1(b)). Since the CNOT reduction would be higher when there are more CNOTs in a block, the goal of partitioning is to *find a block with the number of qubits and biggest number of CNOTs*. Thus, we propose our algorithm: we use a greedy-based heuristic approach. This is an efficient approach; compared with other partitioning algorithms such as dynamic programming that would be limited by circuit depth, our heuristic algorithm is scalable for large-scale circuit partitioning. We summarize the notations used in our algorithm in Table I.

TABLE I
NOTATIONS USED IN OUR ALGORITHM.

Notation	Definition
n	The total number of qubits in a circuit
k	The total number of qubits in a partitioned block
g_i	A gate operation. i is the index of the gate.
q_i	A qubit. i is the index of the qubit.
Q_i	A qubit group. i is the index of the group.
E_{Q_i}	A set of executable gates on Q_i
G	Gate dependency graph
B	A list of partitioned blocks
M	A list of qubit group mapping

Before our heuristic algorithm, we define a few terms used in our algorithm.

Qubit group. A qubit group is a set of qubits in a circuit block. For a n-qubit circuit, there are $\binom{n}{k}$ different combinations of qubit groups for a k-qubit block. For example, the circuit shown in Figure 1(a) is a 5-qubit circuit. For partitioning the circuit into 3-qubit blocks, $\{q_0, q_1, q_2\}$, $\{q_0, q_1, q_3\}$, ..., and $\{q_2, q_3, q_4\}$ are possible qubit groups for a block. Figure 1(b) shows the circuit after partitioning. The qubit group of the 1st block is $\{q_0, q_1, q_4\}$, and the 2nd block is $\{q_1, q_2, q_4\}$, and the 3rd block is $\{q_0, q_1, q_3\}$. We use Q_i to denote a qubit group in this paper, where $i = 1, 2, ..., \binom{n}{k}$. The qubit group size k is limited to a small number due to the limitation of quantum synthesis. Our qubit group mapping maintains the physical qubit mapping. The qubit mapping is fixed during the partition and synthesis. In this paper, we focus our analysis on the size of 3 and 4 qubits in a qubit group.

Valid qubit group. Since the input circuit for our partitioning algorithm is a physical qubit mapped circuit, all two-qubit gates are applied on the neighbor qubits. We can define *valid qubit groups* by considering device connectivity to reduce the search space in our algorithm. We map the qubit group onto the device connectivity graph. If a qubit group is a connected component, it is a valid qubit group; otherwise, it is an invalid group. Figure 3 shows an example of a valid qubit group. Considering a 3×3 grid connectivity, the qubit group $\{q_0, q_3, q_6\}$ is a valid qubit group. However, the qubit group $\{q_2, q_7, q_8\}$ is an invalid qubit group. When partitioning a circuit, we only need to consider valid qubit groups.

Gate dependency graph. We use a Directed Acyclic Graph (DAG) to represent the gate dependency of a circuit. In a DAG, a node represents a gate, and an edge is the qubit involved for the gate. A gate can be executed only when all the previous

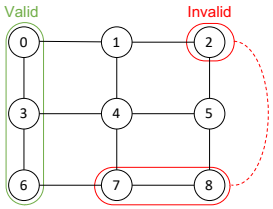


Fig. 3. Example of valid and invalid qubit groups.

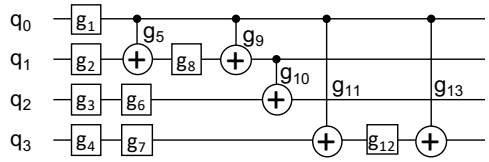
Algorithm 1 Circuit Partitioning

```
Convert the circuit into a gate dependency graph G.
B ← φ
M ← φ
while G is not empty do
    for each valid qubit group Q_i do
        E_{Q_i} ← {Executable Gates on Q_i}
        Compute Score(E_{Q_i})
    end for
    Select the E_{Q_m} with the maximal score;
    B.append(E_{Q_m})
    M.append(Q_m)
    G ← G − E_{Q_m}
end while
Output the sequence of circuit blocks B and the list of qubit group mapping M
```

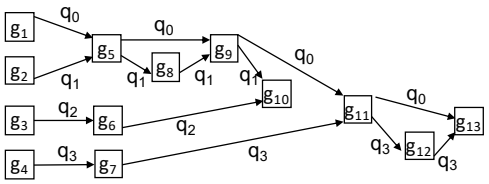
gates have been executed. Figure 4 shows an example of gate dependency graph. The DAG in Figure 4(b) is generated from the original circuit (Figure 4(a)). For example, the gate g_5 depends on g_1 and g_2 because q_0 is used for g_1 and g_5, and q_1 is used for g_2 and g_5. Thus, g_5 can be executed only after both g_1 and g_2 are executed.

Executable gates on a qubit group. A single-qubit gate on q_i can count as an executable gate on a qubit group Q_i only when $q_i \in Q_i$ and all previous gates on q_i are executable gates on Q_i. A two-qubit gate on (q_i, q_j) can count as an executable gate on a qubit group Q_i only when both $q_i, q_j \in Q_i$ and all previous gates on q_i and q_j are executable gates on Q_i. In Figure 4, for example, when we count the executable gates on the qubit group $\{q_0, q_1, q_2\}$, the gate g_9 is an executable gate because q_0 and q_1 are in the qubit group and all the previous gates on q_0 and q_1 are also executable gates. However, g_4 is not an executable gate on this qubit group because q_3 is not in the target qubit group. We use E_{Q_i} to denote the largest set of executable gates on the qubit group Q_i. We can get each E_{Q_i} by traversing the gate dependency graph. Figure 4(c) and Figure 4(d) show $E_{\{q_0,q_1,q_2\}}$ and $E_{\{q_0,q_1,q_3\}}$, respectively.

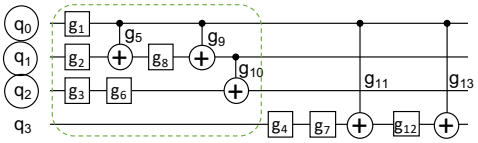
Algorithm 1 shows our partitioning procedure. First, we prepare the gate dependency graph G for the target circuit. Next, we collect the set of executable gates, E_{Q_i}, for each qubit group, and also compute the score for each E_{Q_i}. Since the objective is to find the biggest circuit block, we select the E_{Q_m} with the maximal score as the partitioned block, and save the Q_m as the qubit group mapping for this block. Once a



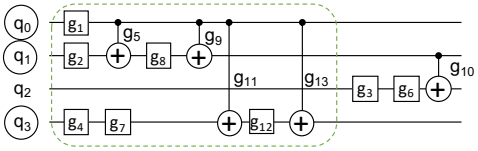
(a) Original circuit



(b) Gate dependency graph



(c) Executable gates on qubit group $\{q_0, q_1, q_2\}$.



(d) Executable gates on qubit group $\{q_0, q_1, q_3\}$.

Fig. 4. Example of gate dependency graph and executable gates on a qubit group.

circuit block is partitioned, we remove the block from the gate dependency graph G, and repeat the procedure to find the next circuit block partition until the gate dependency graph G is empty, which means the entire circuit is partitioned. Finally, a sequence of circuit blocks B and a list of qubit group mapping M are the output of the circuit partitioning process.

The heuristic cost function indicates the number of executable gates for a qubit group. The general form of our heuristic cost function for a qubit group Q_i is shown as follows:

$$Score(E_{Q_i}) = N_{E_{Q_i}}, \tag{1}$$

where $N_{E_{Q_i}}$ is the number of executable CNOT gates on Q_i. Since the objective of partitioning is to find a block for quantum synthesis to minimize the CNOT count, a block with more CNOT gates can potentially achieve higher CNOT reduction. Thus, we use CNOT count as the score function.

The time complexity of our heuristic algorithm for partitioning a circuit into k-qubit blocks is $O(n^k g)$, where n is the

number of qubits in the original circuit, g is the total number of gates, k is the number of qubits in a partitioned block.

D. Quantum Circuit Synthesis

After the circuit is partitioned into multiple blocks, QGo applies quantum synthesis for each circuit block. We integrate the state-of-the-art synthesis tool proposed in [23] into our QGo framework. For each circuit block, QGo computes the corresponding unitary matrix, and this matrix is the input for the synthesis process. As the synthesized circuit should respect the hardware topology, the qubit group for each block and the device connectivity are also the input parameters for the synthesis tool. With our block-based synthesis scheme, the time-to-solution for a n-qubit circuit is reduced from $O(exp(n))$ to $O(exp(k))$.

After the synthesis, there is an undesired synthesis distance due to numerical approximation, leading the final unitary at a distance from the original unitary [23], [54], [72]. As a result, the final unitary distance is in the range of 10^{-10} to 10^{-15}. In the NISQ era, when running a quantum circuit on a real machine, the unitary executed is different from the original intended unitary due to the presence of noise. Since gate errors are multiple orders of magnitudes larger than synthesis distances, these synthesis distances are insignificant. In Section V-B we will show that this distance only causes negligible impact on the overall state fidelity.

E. Circuit Composition

Once all circuit blocks have been synthesized, QGo composes the entire circuit by stitching all circuit blocks. The list of qubit group mappings provide the qubit mapping information to connect blocks correctly. In general, the number of CNOTs in the synthesized block is less than in the original block. If the synthesized circuit block has an equal or greater number of CNOTs compared to the original block, QGo will choose to use the original circuit block to the circuit composition to avoid unnecessary synthesis distance due to floating point errors. Once all blocks are put together, QGo combines the adjacent single-qubit gates to further reduce the gate count, and produces the final optimized circuit.

IV. Experimental Setup

A. Benchmarks

To evaluate QGo against real applications, we select multiple important quantum applications as our benchmarks. Table II lists the applications and brief description in our evaluation. We have three sets of applications according to the different sizes of the circuits. The first set consists of small-size circuits on 4 and 5 qubits; the second set is the medium-size of 9- and 10-qubit circuits; and the third set of benchmarks, large-size circuits, contains circuits beyond 60 qubits. In our evaluation, we run the small-size circuits on a 5-qubit quantum device, Athens, provided by IBM quantum experience platform to demonstrate how much circuit fidelity is improved through our optimization on a real NISQ machine. Next, we run our medium-size benchmarks by using Qiskit noisy simulators to show the fidelity improvement under different levels of gate errors. Finally, we use the large-size benchmarks to demonstrate the scalability of our QGo technique. Since the circuit depth is deep for large scale circuits, we select the important kernel functions to demonstrate the scalability of our approach.

TABLE II
List of benchmarks.

Size	Application	Qubits	Description
Small	QAOA5	5	QAOA on MaxCut problem
	TFIM5	5	Transverse-field Ising model
	MUL5	5	Multiplier arithmetic function
	ADDER4	4	Adder arithmetic function
	QFT5	5	Quantum Fourier transform
	HLF5	5	Hidden linear function
Medium	QAOA10	10	QAOA on MaxCut problem
	TFIM10	10	Transverse-field Ising model
	MUL10	10	Multiplier arithmetic function
	ADDER9	9	Adder arithmetic function
	QFT9	9	Quantum Fourier transform
	HLF9	9	Hidden linear function
Large	MUL60	60	Multiplier arithmetic function
	ADDER63	63	Adder arithmetic function
	QFT64	64	Quantum Fourier transform

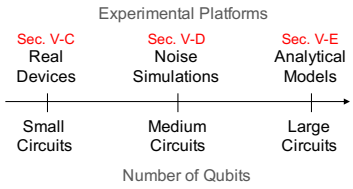


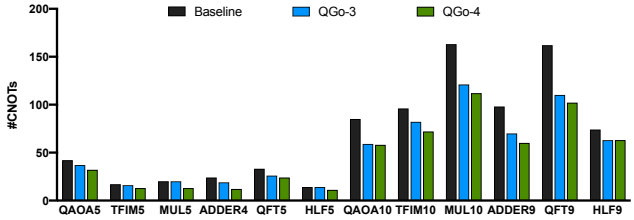
Fig. 5. Experimental platforms used in our evaluation.

Quantum Approximate Optimization Algorithm (QAOA) is a hybrid quantum-classical variational algorithm and it is one of the most important quantum algorithms in the NISQ era [30]. In our study, the QAOA application is a hardware-efficiency ansatz [52] for MaxCut problem. Transverse Field Ising Model (TFIM) is for problems that study the time evolution of chemical systems [38], [62]. Multiplier (MUL) [31] and adder [22], [31] benchmarks are important arithmetic functions in several quantum applications. Quantum Fourier transform (QFT) [42] is a kernel function in many quantum algorithms such as phase estimation algorithm [19], Shor's algorithm [63], and the algorithm for hidden subgroup problem [43]. Hidden Linear Function (HLF) is a quantum circuit solving a problem from a previous study [14].

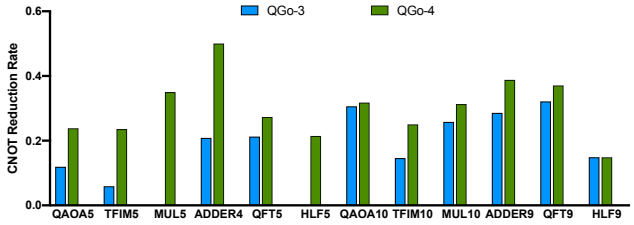
B. Experimental Parameters

We evaluate our QGo with 3-qubit and 4-qubit blocks on different sizes of circuits. All QGo processes and noisy simulations are carried out on a Ubuntu 16.04 system with Intel Xeon Silver 4110 32-core CPU (2.1 GHz) and 128GB RAM.

Figure 5 summarizes our fidelity analysis. For small circuits, we show the experimental results on IBM's Athens device



(a) Total number of CNOTs in the circuits optimized by the baseline compiler and QGo.



(b) CNOT reduction rate compared with the baseline compiler.

Fig. 6. The numbers of CNOTs are reduced in the QGo-optimized circuits.

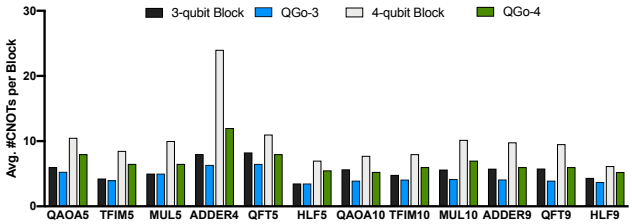


Fig. 7. Average CNOT count in a block.

(Section V-C). For medium circuits, we run the experiments on noise simulations (Section V-D). Finally, we use our analytical model to perform the fidelity results for large circuits (Section V-E). The $t|ket\rangle$ compiler with the highest optimization level is the baseline in our evaluation. The $t|ket\rangle$ compiler is reported to effectively produce shorter circuits for several applications compared with other compilers [20], [66].

V. Evaluation

In our evaluation, we first show the CNOT reduction in the QGo-optimized circuits compared to the baseline ($t|ket\rangle$) optimization, and we compare the compilation time using 3-qubit and 4-qubit blocks. We then discuss the impact of synthesis distance on the quantum state. Next, we present the results running on a real quantum device, IBM's 5-qubit device (Athens), to prove that our technique is critical for the current devices. Then a sensitivity analysis is performed to show how much circuit fidelity is improved under different gate errors. We also demonstrate the scalability of QGo by optimizing large circuits.

A. CNOT Reduction

We use the $t|ket\rangle$ compiler to map 4- and 5-qubit circuits on the IBM's Athens topology, which is a linear connectivity, and

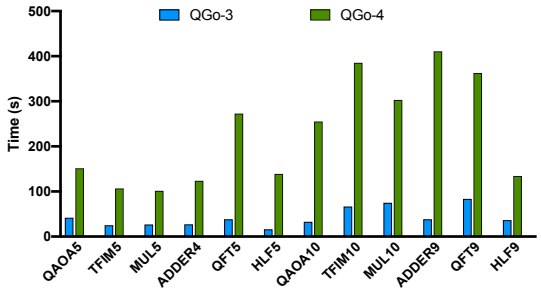


Fig. 8. Compilation time.

map 9- and 10-qubit circuits on a 2D lattice of physical qubits. As discussed in Section III-B, we compare industrial compilers such as IBM Qiskit [21] and $t|ket\rangle$ compilers [66]. Since the $t|ket\rangle$ compiler is shown to generate shorter circuits for several quantum applications [20], we use the circuits optimized by $t|ket\rangle$ compiler as our baseline. The circuits optimized using our QGo with 3-qubit blocks and 4-qubit blocks are denoted as QGo-3 and QGo-4, respectively. Figure 6 shows the total number of CNOTs in the circuits optimized by the baseline and QGo. Our QGo optimization reduces the total number of CNOTs. For MUL5 and HLF5, QGo-3 does not reduce the CNOT count because there is only a small number of CNOTs in a block. QGo-4 can reduce the CNOT counts across all benchmarks. The average CNOT reduction rates of QGo-3 and QGo-4 are 17.2% and 29.9%, respectively. In general, QGo-4 can achieve more CNOT reduction when compared with QGo-3 because a large block will have more CNOTs in a block, and this is easier for the synthesis to find a circuit using less CNOTs than the original circuit block. Figure 7 shows the average CNOT count in a block. If a circuit is partitioned into 4-qubit blocks, the average CNOT count per block is higher when compared with 3-qubit blocks. As a result, QGo-4 can achieve higher CNOT reduction. However, the time-to-solution of QGo-4 is much longer than QGo-3 (Figure 8).

B. Impact of Synthesis Distance

As discussed in Section III-D, since there is a synthesis distance due to numerical approximation, the final state is not exactly the same as the ideal state of the original circuit. In order to analyze the impact of synthesis distance, we use ideal simulation to obtain the final states of the original circuit and synthesized circuit, and we generate multiple random initial states as the input states, and calculate the state infidelity [54]. Figure 9 shows the average state infidelity. Since the unitary matrix distance of 4-qubit circuit synthesis is fundamentally larger than 3-qubit circuit synthesis, QGo-4 has slightly larger state infidelity when compared with QGo-3. The state infidelity of each optimized circuit is less than 10^{-12}. This is insignificant when compared with the gate error in the NISQ era. The current gate infidelity on the available NISQ devices is ranged from 10^{-1} to 10^{-6}. Thus, the impact of synthesis distance is negligible.

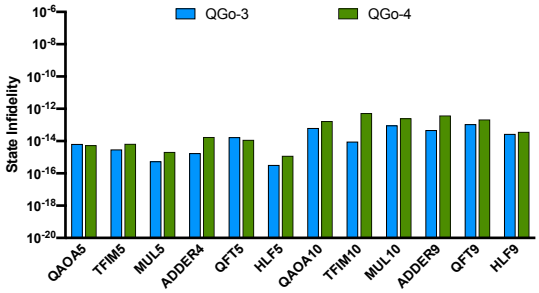


Fig. 9. State infidelity due to the synthesis distance. Compared with the gate error on NISQ devices, the state infidelity due to synthesis distance is negligible.

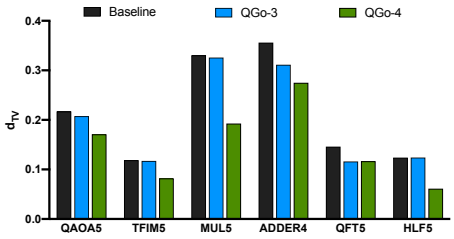


Fig. 10. Running optimized small circuits on IBM's Athens, a 5-qubit device. (Lower d_{TV} is better.)

C. Running on Real Hardware

To measure how much improvement our QGo technique can achieve on the current available NISQ machines, we perform our experiments on quantum devices. Since our medium-size and large-size circuits are too deep for the current quantum devices to generate meaningful results, we choose small-size circuits for this evaluation. We run the small-size benchmarks on IBM's Athens device [40]. We use total variation distance, d_{TV}, to compare measurement samples of the real device with those of the ideal simulation. Total variation distance is commonly used as a metric for QC experiments [7], [8], [28], [48]. Lower d_{TV} means the samples of the real device are closer to the ideal distribution. Figure 10 shows the results on IBM's Athens, a 5-qubit device; each data point is obtained from 8192 shots of the circuit execution. We observe that QGo achieves lower total variation distance for all benchmarks compared with the baseline optimization, and the distance results are correlated to the number of CNOT reduction. Even though the synthesis distance with 4-qubit blocks is larger than with 3-qubit blocks, QGo-4 can achieve lower d_{TV} due to more CNOT reduction. The results suggest that our QGo optimization technique is important in the NISQ era, and is applicable to the current quantum devices.

D. Running on Noise Simulation

To understand the performance of QGo optimization for medium circuits, we run the experiments on IBM Qiskit Aer noise simulation [2] because the medium circuits are too deep to get meaningful results on the current available real devices. To validate the trends shown by our simulations, we run

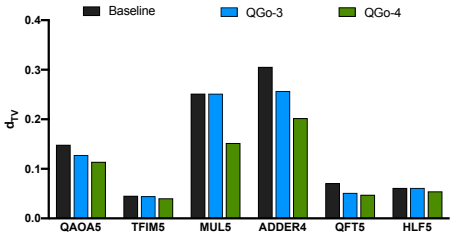


Fig. 11. Running optimized small circuits on Qiskit noise simulation. (Lower d_{TV} is better.)

small circuits on both real devices and simulations to see the correlation between them. Figure 11 shows the results from noise simulation. Compared with the results on IBM's Athens (Figure 10), the d_{TV} improvements are correlated in both experiments. There is a 98% correlation between our real-device results and our simulation results on small circuits.

We perform sensitivity analysis by running noise simulations for medium circuits with different gate errors. Figure 12 shows the noise simulation results. We simulate depolarize_noise for two-qubit gate noises with the gate error probability from 0.1% to 2.5%. We can observe that QGo can reduce more d_{TV} when the gate error is large.

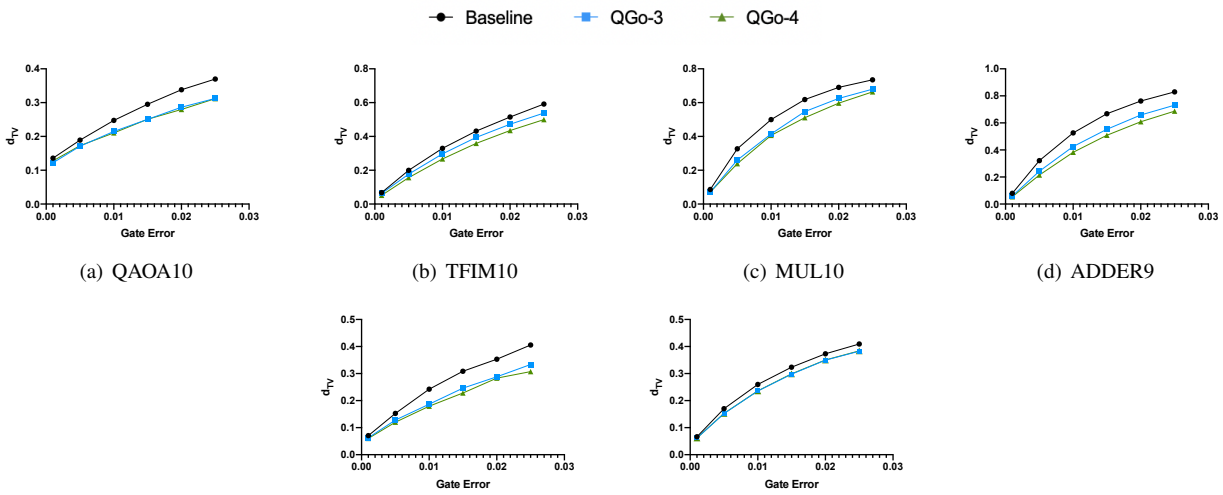
E. Scalability

To demonstrate the scalability of our QGo technique, we optimize the large-scale (60+ qubits) benchmarks using QGo. We map the large-scale circuits on a 2D (8×8) lattice of physical qubits. Figure 13 shows the total number of CNOTs optimized by the baseline compiler and QGo, and Figure 15 shows the CNOT reduction rate. With large-size circuits, QGo-3 and QGo-4 achieves 22.8% and 28.9% CNOT reduction on average. QGo-4 performs better than QGo-3 in terms of CNOT reduction, but it takes longer time to complete the optimization. Figure 14 shows the compilation time of large-scale circuit optimization. QGo-3 can complete the optimization within a few minutes, and QGo-4 can finish the optimization process within a few hours.

Since the scale is too large to perform noise simulation, we use an analytical model to estimate the success rate of each circuit. The success rates in our evaluation are computed by a worst-case analysis using gate success rates. Multiplying the gate success rates, we can obtain the estimated success rate for the whole circuit. Figure 16 shows the results under different gate error models. Since our QGo-4 has the lowest CNOT count, it is projected to achieve the highest success rates for all benchmarks. The success rate improvement is greater when there is a larger gate error.

F. Discussion

The results show the general applicability of our approach. Having access to 3-qubit block synthesis already enables good optimization results on large circuits. In general, larger block size can achieve more CNOT reduction. Running synthesis with 5-qubit blocks is possible, but the solving time is much longer.



(a) QAOA10 (b) TFIM10 (c) MUL10 (d) ADDER9

(e) QFT9 (f) HLF9

Fig. 12. Realistic noise simulation results of d_{TV} under different levels of gate error. (Lower d_{TV} is better.)

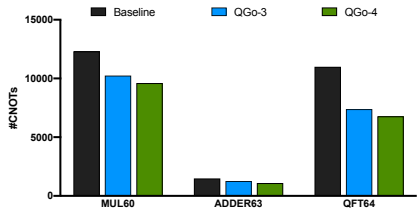


Fig. 13. The number of CNOTs in the optimized large-scale circuits.

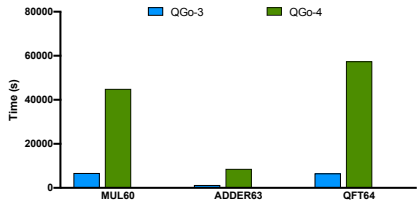


Fig. 14. Compilation time of large-scale circuit optimization.

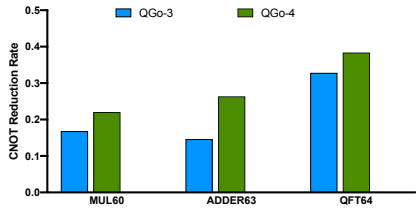


Fig. 15. CNOT reduction rate of large-scale circuits.

QGo allows composability with any mapper available for a given platform and we have experimented with $t|ket\rangle$ compiler. Our preliminary sensitivity analysis of circuit quality to mapping quality indicates there is a direct correlation between QGo efficacy and mapping quality. The best quality mappings have highly interconnected components. In our conjecture, the better mapper provides the higher opportunity of forming large blocks, thus motivating improvements in both mapping and synthesis. Our study offers insights for future compiler design.

VI. Future Work

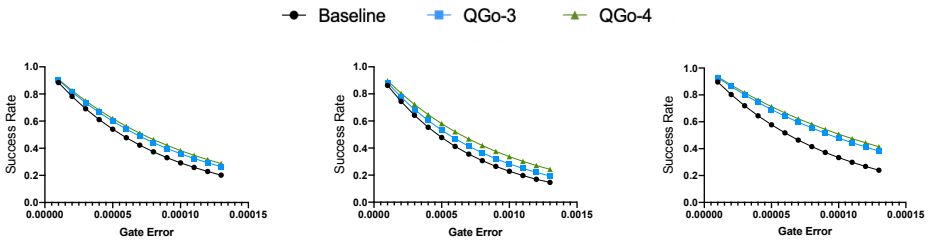
Even though QGo already achieves successful results in optimizing different sizes of circuits, we discuss a few directions that would further extend the line of this research and improve the development of quantum computing.

A. Pulse-Level Optimization

The objective of optimization using synthesis is to reduce the CNOT count. In some cases, it may be desirable to make approximations to reduce the number of CNOTs at the expense of how closely the final circuit approximates the desired unitary. An intuitive way to achieve this is to relax a synthesis threshold. In order to effectively trade accuracy for CNOT count, a threshold-controllable synthesis tool is necessary. Integrating a different synthesis tool or heuristic that is designed for approximate synthesis may further improve the optimization. Since the device-level control of a quantum computer is operated via analog pulses, recent pulse optimization studies aim to generate shorter pulses [17], [18], [33]–[35], [61]. Pulse optimization can also be integrated into our QGo technique as a backend optimization. The current synthesis output is a sequence of quantum gates. To integrate pulse optimization into this work, we can synthesize the block into a sequence of pulses.

B. Partial Optimization

Since our approach already partitions a circuit into multiple blocks, we can simply only perform synthesis on some critical blocks to reduce the compilation time. Also, we can combine different sizes of blocks in the optimization according to the importance of a block.



(a) MUL60 (b) ADDER63 (c) QFT64

Fig. 16. Circuit success rate under different levels of gate errors. (Higher circuit success rate is better.)

C. QGo in Parallel

In this work, we perform the quantum synthesis in serial. However, since we partition a circuit into a sequence of circuit blocks, each block is independent for the synthesis process. As a result, the synthesis of blocks can be executed in parallel. Running the entire QGo on a supercomputing system can reduce the overall compilation time significantly.

D. Crosstalk Mitigation

Our optimization tool that can be applied with any other existing optimization to improve the overall circuit fidelity. For cross talk, we can apply crosstalk mitigation to address this issue, and as we reduce the CNOT count, it would be easier to mitigate crosstalk.

VII. Related Work

Several studies of circuit optimization have been carried out. Most of the existing techniques focus on optimizing the qubit mapping and swap insertion to reduce circuit depth. One common approach is to describe the problem in a mathematical form, such as integer linear programming, and then find the optimal solutions by using solvers [9], [37], [49], [51], [70]. This approach only works for small circuits since the time scaling is exponential. Another approach is to find the optimal solutions by using dynamic programming [41], [65]. However, since the solving time grows exponentially, this method only works for a handful of qubits. Recent studies propose using heuristic search algorithms to find good solutions to avoid long execution times [3], [11], [41], [46], [47], [59], [69], [73]. However, these approaches keep the original CNOT count and only reduce the additional swap count. Our synthesis approach can reduce both swap count and CNOT count used in the circuits.

Previous studies have applied synthesis technique to optimize some specific circuits such as classical reversible circuits [4], [10], [37], [68], a specific gate set [5], or Clifford+T circuits [55]. [25] proposes architecture-aware synthesis for phase polynomials. This manner can be applied to circuits containing only CNOT and R_z gates. In our work, our approach is designed for general circuits. However, since we can easily change the core synthesis tool, these synthesis approaches can be integrated in our compiler framework to improve the optimization for these specific circuits.

Our approach relies on synthesis techniques that are able to produce extremely short circuits. Otherwise, it is unlikely that we will be able to see an improvement when resynthesizing sub-circuits. The KAK decomposition could be used for resynthesizing 2-qubit blocks [67]. We have seen improvement when using larger block sizes, so we use the search-based technique found in [23], which can handle circuits as large as 4 qubits. For scaling further, we are considering the approach found in [72], which produces slightly longer circuits, but offers a better scaling runtime when compared to the search-based approach.

Recent studies such as [13] propose the removal of the constraint of unitary operations by adding ancilla qubits. Additionally, [15] uses ancillas and an approximation technique to produce very short circuits. To achieve greater CNOT reduction, integrating ancillas and approximate synthesis into our QGo is a promising research direction.

VIII. Conclusion

In the NISQ era, since two-qubit gates are much noisier than single-qubit gates, it is essential to minimize their count. Synthesis is a powerful tool for circuit optimization to produce shorter circuits to improve the overall circuit fidelity. However, synthesis is only applicable for small circuits. In this work, we present an automated compilation framework, QGo. It partitions the circuit into blocks, and re-generates each optimized block by using synthesis, and re-composes the circuit by stitching all the blocks together. Our approach to circuit optimization offers a role for quantum synthesis algorithms in large-scale quantum computing scenarios. We evaluate fidelity improvements using 3 metrics for 3 scaling regimes: small-size circuits using fidelity on real devices, medium-size circuits using fidelity using simulations with noise, and large-size circuits using CNOT reduction measured statically by our compiler. The results show that our technique has practical value on current devices and is reliable in the NISQ era. We also discuss using approximate synthesis to further trade for circuit depth and pulse-level optimization. Circuit fidelity improvement is critical. Our approach provides a next-level optimization that is robust for direct incorporation in existing compiler tools. Our study of circuit optimization using synthesis offers insights for future compiler design.

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 work was supported by the U.S. Department of Energy (DOE) under Contract No. DE-AC02-05CH11231, through the Office of Advanced Scientific Computing Research Accelerated Research for Quantum Computing Program.

Disclosure: F. Chong is Chief Scientist at Super.tech and an advisor to QCI.

REFERENCES

[1] O. Al-Ta'ani, "Quantum circuit synthesis using solovay-kitaev algorithm and optimization techniques," Ph.D. dissertation, Kansas State University, 2015.

[2] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C. Chen *et al.*, "Qiskit: An open-source framework for quantum computing," *Accessed on: Mar*, vol. 16, 2019.

[3] M. AlFailakawi, I. Ahmad, and S. Hamdan, "Lnn reversible circuit realization using fast harmony search based heuristic," in *Asia-Pacific Conference on Computer Science and Electrical Engineering*, 2014.

[4] Z. Alwardi, R. Wille, and R. Drechsler, "Synthesis of reversible circuits using conventional hardware description languages," in *2018 IEEE 48th International Symposium on Multiple-Valued Logic (ISMVL)*. IEEE, 2018, pp. 97–102.

[5] M. Amy, P. Azimzadeh, and M. Mosca, "On the controlled-not complexity of controlled-not–phase circuits," *Quantum Science and Technology*, vol. 4, no. 1, p. 015002, 2018.

[6] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 6, pp. 818–830, 2013.

[7] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[8] C. Bădescu, R. O'Donnell, and J. Wright, "Quantum state certification," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 2019, pp. 503–514.

[9] T. Bahreini and N. Mohammadzadeh, "An MINLP model for scheduling and placement of quantum circuits with a heuristic solution approach," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 12, no. 3, pp. 1–20, 2015.

[10] C. Bandyopadhyay, R. Wille, R. Drechsler, and H. Rahaman, "Post synthesis-optimization of reversible circuit using template matching," in *2020 24th International Symposium on VLSI Design and Test (VDAT)*. IEEE, 2020, pp. 1–4.

[11] A. Bhattacharjee, C. Bandyopadhyay, R. Wille, R. Drechsler, and H. Rahaman, "A novel approach for nearest neighbor realization of 2D quantum circuits," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2018, pp. 305–310.

[12] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, p. 195, 2017.

[13] A. Bocharov, M. Roetteler, and K. M. Svore, "Efficient synthesis of universal repeat-until-success quantum circuits," *Physical review letters*, vol. 114, no. 8, p. 080502, 2015.

[14] S. Bravyi, D. Gosset, and R. König, "Quantum advantage with shallow circuits," *Science*, vol. 362, no. 6412, pp. 308–311, 2018.

[15] D. Camps and R. Van Beeumen, "Approximate quantum circuit synthesis using block encodings," *Physical Review A*, vol. 102, no. 5, p. 052411, 2020.

[16] A. Chakrabarti, S. Sur-Kolay, and A. Chaudhury, "Linear nearest neighbor synthesis of reversible circuits by graph partitioning," *arXiv preprint arXiv:1112.0564*, 2011.

[17] J. Cheng, H. Deng, and X. Qia, "Accqoc: Accelerating quantum optimal control based pulse generation," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 543–555.

[18] J. Cheng, H. Deng, and X. Qian, "Accqoc: Accelerating quantum optimal control based pulse generation," in *47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020, Valencia, Spain, May 30 - June 3, 2020*. IEEE, 2020, pp. 543–555. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00052

[19] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, "Quantum algorithms revisited," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 454, no. 1969, pp. 339–354, 1998.

[20] A. Cowtan, S. Dilkes, R. Duncan, W. Simmons, and S. Sivarajah, "Phase gadget synthesis for shallow circuits," *arXiv preprint arXiv:1906.01734*, 2019.

[21] A. Cross, "The IBM Q experience and QISKit open-source quantum computing software," *Bulletin of the American Physical Society*, vol. 63, 2018.

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

[23] M. G. Davis, E. Smith, A. Tudor, K. Sen, I. Siddiqi, and C. Iancu, "Towards optimal topology aware quantum circuit synthesis," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2020, pp. 223–234.

[24] C. M. Dawson and M. A. Nielsen, "The solovay-kitaev algorithm," *arXiv preprint quant-ph/0505030*, 2005.

[25] A. M.-v. de Griend and R. Duncan, "Architecture-aware synthesis of phase polynomials for nisq devices," *arXiv preprint arXiv:2004.06052*, 2020.

[26] A. De Vos and S. De Baerdemacker, "Block-z x z synthesis of an arbitrary quantum circuit," *Physical Review A*, vol. 94, no. 5, p. 052317, 2016.

[27] O. Di Matteo and M. Mosca, "Parallelizing quantum circuit synthesis," *Quantum Science and Technology*, vol. 1, no. 1, p. 015003, 2016.

[28] Y. Ding, X.-C. Wu, A. Holmes, A. Wiseth, D. Franklin, M. Martonosi, and F. T. Chong, "Square: Strategic quantum ancilla reuse for modular quantum programs via cost-effective uncomputation," in *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020.

[29] D. P. DiVincenzo, "Two-bit gates are universal for quantum computation," *Physical Review A*, vol. 51, no. 2, p. 1015, 1995.

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

[31] C. Gidney and D. Bacon, "The cirq developers, quantumlib/-cirq: A python framework for creating, editing, and invoking noisy intermediate scale quantum (nisq) circuits."

[32] B. Giles and P. Selinger, "Exact synthesis of multiqubit clifford+ t circuits," *Physical Review A*, vol. 87, no. 3, p. 032332, 2013.

[33] S. J. Glaser, U. Boscain, T. Calarco, C. P. Koch, W. Köckenberger, R. Kosloff, I. Kuprov, B. Luy, S. Schirmer, T. Schulte-Herbrüggen *et al.*, "Training schrödinger's cat: quantum optimal control," *The European Physical Journal D*, vol. 69, no. 12, pp. 1–24, 2015.

[34] P. Gokhale, Y. Ding, T. Propson, C. Winkler, N. Leung, Y. Shi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Partial compilation of variational algorithms for noisy intermediate-scale quantum machines," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 266–278.

[35] P. Gokhale, A. Javadi-Abhari, N. Earnest, Y. Shi, and F. T. Chong, "Optimized quantum compilation for near-term algorithms with openpulse," *arXiv preprint arXiv:2004.11205*, 2020.

[36] "A Preview of Bristlecone, Google's New Quantum Processor," https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html, accessed: 2020-10-09.

[37] D. Große, R. Wille, G. W. Dueck, and R. Drechsler, "Exact multiple-control toffoli network synthesis with sat techniques," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 5, pp. 703–715, 2009.

[38] M. Heyl, A. Polkovnikov, and S. Kehrein, "Dynamical quantum phase transitions in the transverse-field ising model," *Physical review letters*, vol. 110, no. 13, p. 135704, 2013.

[39] "IBM Announces Advances to IBM Quantum Systems and Ecosystem," https://www-03.ibm.com/press/us/en/pressrelease/53374.wss, accessed: 2020-10-09.

[40] "IBM Quantum Experience," https://quantum-computing.ibm.com/, accessed: 2020-11-15.

[41] T. Itoko, R. Raymond, T. Imamichi, and A. Matsuo, "Optimization of quantum circuit mapping using gate transformation and commutation," *Integration*, vol. 70, pp. 43–50, 2020.

[42] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi, "ScaffCC: Scalable compilation and analysis of quantum programs," *Parallel Computing*, vol. 45, pp. 2–17, 2015.

[43] R. Jozsa, "Quantum factoring, discrete logarithms, and the hidden subgroup problem," *Computing in science & engineering*, vol. 3, no. 2, p. 34, 2001.

[44] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, no. 7671, p. 242, 2017.

[45] V. Kliuchnikov, D. Maslov, and M. Mosca, "Fast and efficient exact synthesis of single qubit unitaries generated by clifford and t gates," *arXiv preprint arXiv:1206.5236*, 2012.

[46] A. Kole, K. Datta, and I. Sengupta, "A heuristic for linear nearest neighbor realization of quantum circuits by SWAP gate insertion using N-gate lookahead," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 1, pp. 62–72, 2016.

[47] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1001–1014.

[48] A. P. Lund, M. J. Bremner, and T. C. Ralph, "Quantum sampling problems, bosonsampling and quantum supremacy," *npj Quantum Information*, vol. 3, no. 1, pp. 1–8, 2017.

[49] A. Lye, R. Wille, and R. Drechsler, "Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits," in *The 20th Asia and South Pacific Design Automation Conference*. IEEE, 2015, pp. 178–183.

[50] E. A. Martinez, T. Monz, D. Nigg, P. Schindler, and R. Blatt, "Compiling quantum algorithms for architectures with multi-qubit gates," *New Journal of Physics*, vol. 18, no. 6, p. 063029, 2016.

[51] D. Maslov, S. M. Falconer, and M. Mosca, "Quantum circuit placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 4, pp. 752–763, 2008.

[52] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn *et al.*, "Quantum optimization using variational algorithms on near-term quantum devices," *Quantum Science and Technology*, vol. 3, no. 3, p. 030503, 2018.

[53] A. B. Nagy, "On an implementation of the solovay-kitaev algorithm," *arXiv preprint quant-ph/0606077*, 2006.

[54] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[61] Y. Shi, N. Leung, P. Gokhale, Z. Rossi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Optimized compilation of aggregated instructions for realistic quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1031–1044.

[55] P. Niemann, R. Wille, and R. Drechsler, "Improved synthesis of clifford+ t quantum functionality," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 597–600.

[56] S. Nishio, Y. Pan, T. Satoh, H. Amano, and R. Van Meter, "Extracting success from ibm's 20-qubit machines using error-aware compilation," *arXiv preprint arXiv:1903.10963*, 2019.

[57] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, p. 4213, 2014.

[58] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[59] M. Saeedi, R. Wille, and R. Drechsler, "Synthesis of quantum circuits for linear nearest neighbor architectures," *Quantum Information Processing*, vol. 10, no. 3, pp. 355–377, 2011.

[60] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum-logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1000–1010, 2006.

[62] D. Shin, H. Hübener, U. De Giovannini, H. Jin, A. Rubio, and N. Park, "Phonon-driven spin-floquet magneto-valleytronics in mos 2," *Nature communications*, vol. 9, no. 1, pp. 1–8, 2018.

[63] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[64] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, 2018, pp. 113–125.

[65] ——, "Qubit allocation," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, 2018, pp. 113–125.

[66] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "t— ket¿: A retargetable compiler for nisq devices," *Quantum Science and Technology*, 2020.

[67] R. R. Tucci, "An introduction to cartan's kak decomposition for qcfon programmers," *arXiv preprint quant-ph/0507171*, 2005.

[68] R. Wille, M. Haghparast, S. Adarsh, and M. Tanmay, "Towards hdl-based synthesis of reversible circuits with no additional lines," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–7.

[69] 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)*. IEEE, 2016, pp. 292–297.

[70] R. Wille, A. Lye, and R. Drechsler, "Optimal SWAP gate insertion for nearest neighbor quantum circuits," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2014, pp. 489–494.

[71] X.-C. Wu, D. M. Debroy, Y. Ding, J. M. Baker, Y. Alexeev, K. R. Brown, and F. T. Chong, "Tilt: Achieving higher fidelity on a trapped-ion linear-tape quantum computing architecture," 2020.

[72] E. Younis, K. Sen, K. Yelick, and C. Iancu, "Qfast: Conflating search and numerical optimization for scalable quantum circuit synthesis," *arXiv preprint arXiv:2103.07093*, 2021.

[73] A. Zulehner, A. Paler, and R. Wille, "Efficient mapping of quantum circuits to the ibm qx architectures," in *2018 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2018, pp. 1135–1138.