Verifying Results of the IBM Qiskit Quantum Circuit Compilation Flow

Lukas Burgholzer*   Rudy Raymond†   Robert Wille*‡

*Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
†IBM Research – Tokyo, Japan
‡Software Competence Center Hagenberg GmbH (SCCH), Hagenberg, Austria
lukas.burgholzer@jku.at   rudyhar@jp.ibm.com   robert.wille@jku.at
https://iic.jku.at/eda/research/quantum/

Abstract—**Realizing a conceptual quantum algorithm on an actual physical device necessitates the algorithm's quantum circuit description to undergo certain transformations in order to adhere to all constraints imposed by the hardware. In this regard, the individual high-level circuit components are first synthesized to the supported low-level gate-set of the quantum computer, before being mapped to the target's architecture—utilizing several optimizations in order to improve the compilation result. Specialized tools for this complex task exist, e.g., IBM's Qiskit, Google's Cirq, Microsoft's QDK, or Rigetti's Forest. However, to date, the circuits resulting from these tools are hardly verified, which is mainly due to the immense complexity of checking if two quantum circuits indeed realize the same functionality. In this paper, we propose an efficient scheme for *quantum circuit equivalence checking*—specialized for verifying results of the IBM Qiskit quantum circuit compilation flow. To this end, we combine characteristics unique to quantum computing, e.g., its inherent reversibility, and certain knowledge about the compilation flow into a dedicated equivalence checking strategy. Experimental evaluations confirm that the proposed scheme allows to verify even large circuit instances with tens of thousands of operations within seconds or even less, whereas state-of-the-art techniques frequently time-out or require substantially more runtime. A corresponding open source implementation of the proposed method is publicly available at https://github.com/iic-jku/qcec.**

I. Introduction

Quantum computing has gained considerable momentum over the past years, as actual quantum computers are reaching feasibility and more and more algorithms for potential applications are discovered. Similar to the conventional realm, a conceptual quantum algorithm needs to be *compiled* to a representation that conforms to all restrictions imposed by the device it shall be executed on. To this end, fast-evolving compilation flows such as those of IBM's Qiskit [1], Google's Cirq [2], Microsoft's QDK [3], or Rigetti's Forest [4] are available.

Naturally, it is of utmost importance that the results of such compilation flows are correct, i.e., that the compiled quantum circuit still realizes the originally intended functionality. This motivates the development of methods for verification or, more precisely, equivalence checking. Conceptually, equivalence checking is simple, since each quantum circuit G realizes a unitary transformation U and comparing two quantum circuits G and G' boils down to comparing the corresponding unitary matrices U and U', respectively. However, since these matrices U and U' are exponentially large, many existing approaches to this problem (e.g., [5]–[9]) remain unsatisfactory and can hardly be employed on a large scale.

Recently, a promising solution to address this problem has been proposed in [10]. There, the use of dedicated data-structures, in particular decision diagrams [9], [11], [12], has been proposed which allow for a non-exponential representation of matrices in some (albeit not all) cases. To further improve upon that, it is exploited that $G'^{-1} \cdot G = \mathbb{I}$ holds if two circuits G and G' are indeed equivalent, i.e., the cascade composed of one circuit with the inverse of the other should eventually yield the identity function. Because of that, checking whether G and G' are in fact equivalent can be conducted by starting with the identity matrix (which can be represented in linear rather than exponential space) and, then, applying operations of G and G' in a particular order until all operations have been applied. If all those operations can be applied so that the respective intermediate computations remain as close to the identity as possible, substantial improvements can be achieved [10]. However, how to determine the "perfect" order of applications, i.e., whether to apply operations from G or from G' in order to keep the respective intermediate representation close to the identity, remains an open problem.[1]

In this paper, we address this problem. We show that, by utilizing knowledge about the compilation flow, a verification methodology can be obtained which keeps the respectively occurring intermediate representations close to the identity in an almost perfect fashion. By this, the exponential complexity is frequently reduced to a linear or close-to-linear complexity—substantially reducing the runtime of the verification process. In order to showcase the possible improvements, we consider the compilation flow as it is currently conducted by IBM's Qiskit.[2] Experimental evaluations show that, using

---

[1]This is discussed and illustrated in more detail later in Section III.
[2]However, the methods proposed in this work can be tailored to any other compilation flow as well.

the proposed method, circuits composed of tens of thousands of operations can be verified within seconds or even less, whereas state-of-the-art techniques frequently time-out or require substantially more runtime. A corresponding open source implementation of the proposed method is publicly available at https://github.com/iic-jku/qcec.

The rest of this work is structured as follows: In Section II, we review the necessary basics of quantum circuits and the IBM Q systems needed to keep this work self-contained. Then, Section III reviews the compilation flow as it is conducted by IBM's Qiskit and elaborates on verifying results of this flow. Based on that, Section IV illustrates how all different aspects are exploited and consequently orchestrated to form a dedicated verification strategy. The strategy's performance is then evaluated in Section V, before Section VI concludes the paper.

II. Quantum Computing and the IBM Q Systems

In this section, we briefly review the concepts of quantum computing [13] and introduce the notation used in this work. Besides that, we also review the considered platform, i.e., IBM Q systems [14]. For more detailed information, we refer the interested reader to the provided references.

A. Quantum Circuits

In quantum computing, the main computational unit is the *qubit*. In contrast to classical bits, a single qubit q can be in an arbitrary superposition of the basis states $|0\rangle$ and $|1\rangle$, i.e.,

$$|q\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$$

with $\alpha_0, \alpha_1 \in \mathbb{C}$ and $|\alpha_0|^2 + |\alpha_1|^2 = 1$. An n-qubit quantum system can be in an arbitrary superposition of the 2^n basis states

$$|b_{n-1}\rangle \otimes \cdots \otimes |b_0\rangle = |b_{n-1} \ldots b_0\rangle = \left| \sum_{i=0}^{n-1} b_i 2^i \right\rangle$$

with $b_i \in \{0, 1\}$, i.e.,

$$|q\rangle_n = \sum_{i=0}^{2^n - 1} \alpha_i |i\rangle \text{ with } \alpha_i \in \mathbb{C} \text{ and } \sum_{i=0}^{2^n - 1} |\alpha_i|^2 = 1.$$

In the circuit model of quantum computation, qubits are represented by *wires* and are manipulated by quantum operations (*quantum gates*). Specifically, a quantum circuit G with m gates, operating on n qubits, is denoted by $G = g_0 \ldots g_{m-1}$, where each g_i represents a quantum gate acting on (a subset of) n qubits. This is usually visualized through *quantum circuit diagrams*, where the qubit wires are drawn as horizontal lines, gates are drawn using a variety of symbols, and progression of time is assumed to happen from left to right.

Example 1. *An example of a quantum circuit G with 16 gates acting on three qubits is shown in Fig. 1a. This sequence of operations describes a small instance of the famous* Grover

search algorithm [15]. The small boxes with identifiers correspond to operations applied to single qubits such as X gates (the quantum analogue to the NOT gate) and H(adamard) gates (which can be used to set a qubit into superposition). Moreover, there are multiple-controlled X operations, where an X operation is only applied to a target qubit (denoted by $\oplus$) if all of its control qubits (denoted by $\bullet$) are in state $|1\rangle$. In case there is only one control qubit, such a gate is also called CNOT or controlled-NOT, while in case of two control qubits it is also called a Toffoli gate.*

Initially, quantum algorithms are described in a way which is agnostic of the device they are planned to be executed on. However, physical devices today only support very low-level quantum operations. In this work, we focus on devices provided by IBM, which are briefly reviewed next.

B. The IBM Q Systems

In 2016, IBM launched the *IBM Quantum Experience* cloud platform which, for the first time, provides public access to a quantum computer. Today, eight devices with either one, five, or 15 qubits are freely available for development. All quantum computers developed at IBM provide the same **limited gate-set** consisting of arbitrary single-qubit gates U and the two-qubit $CNOT$ operation.[3] Although seemingly small, this already constitutes a universal gate-set, i.e., every possible quantum computation can be realized using only gates from this set [13]. In addition, the IBM Q systems (or in general quantum computers based on superconducting qubits) feature a severely **limited connectivity** of their qubits. This is usually described by a *coupling graph*, where the graph's nodes represent the qubits and an edge between two nodes indicates that a $CNOT$ operation may be applied to those qubits.[4]

Example 2. *The coupling graph of the IBM Q London system is shown in Fig. 1c. It consists of five (physical) qubits Q_i and a $CNOT$ operation may only be applied to the qubits (Q_0, Q_1), (Q_1, Q_2), (Q_1, Q_3), or (Q_3, Q_4).*

A device's physical qubits are inherently affected by noise—leading to rather **short coherence times** and **limited fidelity** of the individual operations. Until a certain threshold concerning the number of available qubits is reached, error correction is not yet an option. Throughout this work, we will refer to the *physical* qubits of a device using upper case Q_i and denote the *logical* qubits of a quantum algorithm with lower case q_i. Additionally, $Q_i : q_j$ denotes that the logical qubit q_j is assigned to the physical qubit Q_i.

---

[3]Specifically, $U = U_3(\theta, \phi, \lambda)$ with $0 \leq \theta \leq \pi$ and $0 \leq \phi < 2\pi$ are supported. Special cases are $U_2(\phi, \lambda) = U_3(\frac{\pi}{2}, \phi, \lambda)$ and $U_1(\lambda) = U_3(0, 0, \lambda)$.

[4]Nowadays, these edges are typically undirected, i.e., it does not matter which qubit acts as control or target—whereas past architectures actually prescribed the direction.
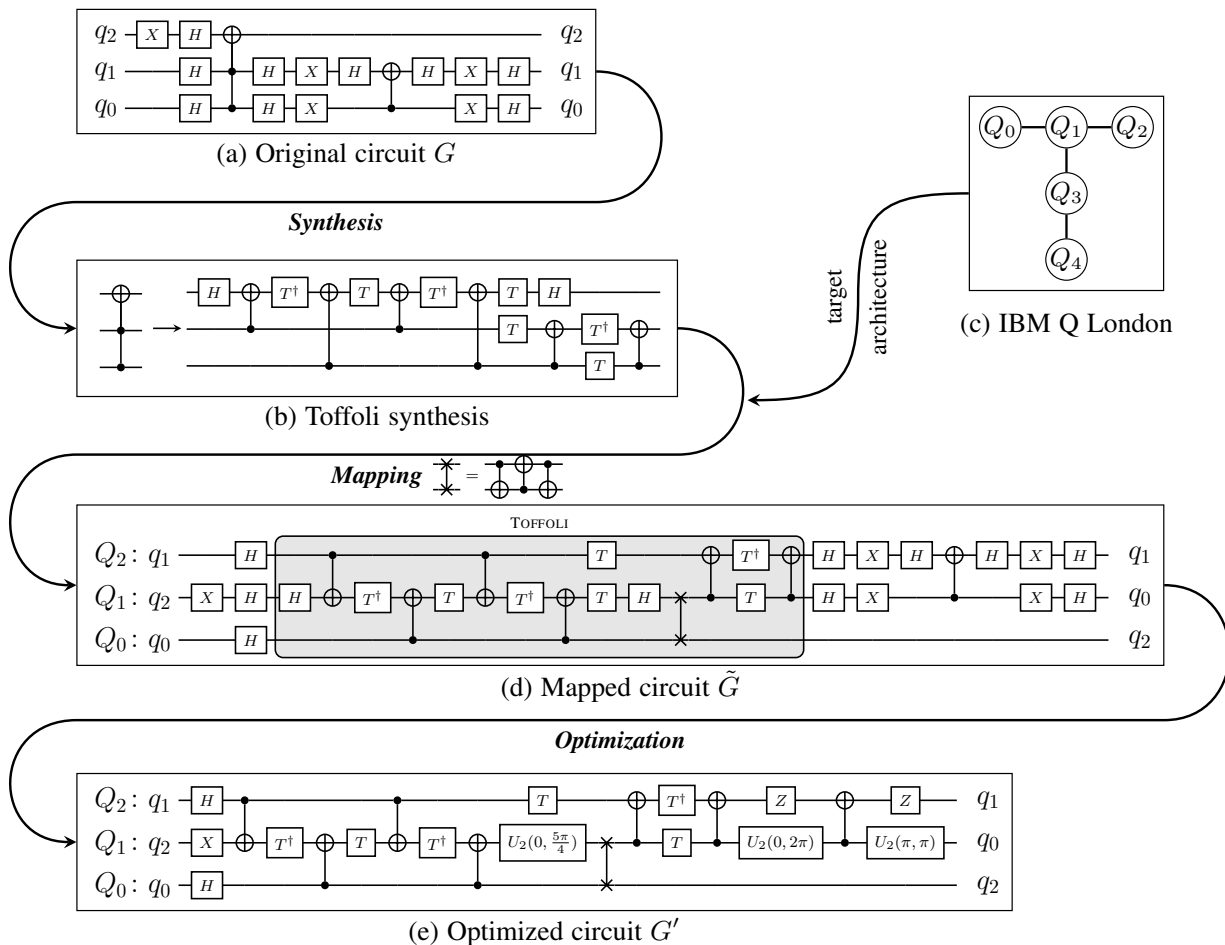
Figure 1: Exemplary illustration of the IBM compilation flow

III. MOTIVATION AND CONSIDERED PROBLEM

In order to execute a conceptual quantum algorithm on an actual device such as provided by the IBM Q systems, several transformations have to be applied to the original circuit in order to conform to all the restrictions imposed by the targeted device. This transformation process is frequently called *compilation*[5] due to its similarity to a classical compiler transforming high-level code into a (machine-executable) assembly. Many specialized tools for this complex task exist, e.g., IBM's Qiskit [1], Google's Cirq [2], Microsoft's QDK [3], or Rigetti's Forest [4]. However, to date, the circuits resulting from these tools are hardly verified, i.e., it is hardly checked whether the resulting quantum circuit description still realizes the same functionality as the originally given circuit. In the following, we review this compilation flow using IBM's Qiskit compilation flow as a representative. Afterwards, we elaborate on the problem of verifying results of this flow, which will eventually motivate this work.

---

[5]Other terms (interchangeably) referring or relating to (steps of) this process are synthesis, mapping, qubit allocation, qubit routing, or transpilation. In this work, we consider the complete process as *compilation*—split into the three main tasks *synthesis*, *mapping*, and *optimization*.

A. *The IBM Qiskit Compilation Flow*

Compilation of quantum circuits addresses the three kinds of restrictions which limit the usability of a quantum computer and have been reviewed above: The first two, i.e., the limited gate-set and connectivity, constitute *hard* constraints— a computation not conforming to these restrictions may not be executed on the device. In contrast, the short coherence time and limited gate fidelity represent a *soft* constraint—a quantum circuit may be executed on a device, but it is not guaranteed to produce meaningful results if the circuit, e.g., is too large for the state to stay coherent. The Qiskit compilation flow is structured as a collection of individual passes, each of which is responsible for dealing with a certain constraint (or an aspect thereof). Just as traditional compilers, there are different optimization levels offering a trade-off between compilation runtime and quality of the compilation result. More precisely:

First, the gates of the original quantum circuit are **synthesized** to the gate-set supported by the targeted device. Most importantly, since devices typically only support up to two-qubit gates, any gate acting on more than two qubits is broken down into "elementary" gates. This process may require the use of additional *ancillary* qubits for realizing the desired

operation. In this regard, Qiskit provides several modes, e.g., for the decomposition of multi-controlled gates—offering a trade-off between circuit size and number of required ancillary qubits [16]–[18].

Example 3. *Consider again the circuit G from Ex. 1 as shown in Fig. 1a. If this circuit shall be executed on an IBM Q system, the Toffoli gate (the two-controlled NOT) first has to be realized using only arbitrary single-qubit gates and CNOTs. One possible synthesized version is shown in Fig. 1b. It takes six CNOTs, nine single qubit gates, and no additional ancillaries to realize the desired gate.*

Now, the circuit just contains elementary gates supported by the device, but it may not yet conform to the device's limited connectivity. Thus, the quantum circuit is **mapped** to the target architecture, i.e., a mapping between the circuit's *logical* and the device's *physical* qubits is established. Qiskit provides several heuristics for determining an *initial mapping*—from a trivial one-to-one mapping ($Q_i : q_i$) to explicitly considering calibration data and picking the most reliable set of qubits for the computation [19]. However, in most cases, it is not possible to globally define a mapping which conforms to *all* connectivity limitations. As a consequence, the logical-to-physical qubit mapping usually is changed dynamically throughout the circuit. Typically, this is accomplished by inserting $SWAP$ gates into the circuit—effectively allowing to change the mapping of logical qubits to physical qubits so that all operations can be executed while, at the same time, all connectivity constraints are satisfied. To this end, Qiskit per default uses a very fast, stochastic solution (based on Bravyi's algorithm). Several other approaches have been proposed for tackling this immensely complex task[6] [7], [19]–[25] and some of them have even been integrated into Qiskit.

Example 4. *Consider again the circuit G from Ex. 1 and assume that the Toffoli gate has been synthesized as shown in Fig. 1b. Further, assume that the circuit is to be executed on the IBM Q London architecture shown in Fig. 1c. Then, Fig. 1d shows one possible circuit $\tilde{G}$ resulting from this mapping process. The physical qubits Q_0, Q_1, and Q_2 were chosen and initially assigned logical qubits q_0, q_2, and q_1, respectively. Just one SWAP operation applied to Q_0 and Q_1 (indicated by ×) was added in the middle of the circuit in order to conform to the target's connectivity constraints[7].*

After this step of the compilation flow, circuits are ready to be executed on the targeted devices (corresponding to the most basic optimization level O0). However, the previous steps significantly increased the size of these circuits—impacting the achievable performance due to the limited coherence time and gate fidelity. Thus, several **optimizations** may be employed to

reduce the circuit's size and, hence, improve the actual performance on the quantum computer. Since the IBM Q systems natively support arbitrary single-qubit gates, any number of subsequent single-qubit gates may be *fused* into one single gate. Additionally, *adjacent-gate-cancellations* can be used to eliminate instances where a gate is directly followed by its inverse, e.g., two consecutive $CNOT$ operations with the same control and target qubits can be cancelled. These are the most basic optimizations that constitute the standard optimization level O1 of Qiskit. Naturally, more sophisticated optimization techniques have been developed, e.g., gate transformation and commutation [26] (which is included in optimization level O2) or re-synthesis of two-qubit unitary blocks [27] (which is part of the top optimization level O3).

Example 5. *Consider again the circuit $\tilde{G}$ from Ex. 4 shown in Fig. 1d that has been mapped to the IBM Q London architecture. Applying one-qubit-fusion and adjacent-gate-cancellation eventually allows to eliminate nine single-qubit gates and results in the* optimized *circuit G' shown in Fig. 1e.*

B. Verifying Results of the Compilation Flow

Naturally, it is of utmost importance that the originally intended functionality of a quantum algorithm is preserved throughout the whole compilation flow. This can be guaranteed by *verifying* the results of the compilation flow. To this end, two possible approaches can be employed: (1) systematically verifying the compilation methods themselves, e.g., by using formal verification techniques [28], [29], or (2) checking the functional equivalence of the original circuit to the respectively compiled circuit [5]–[10]. In this work, we consider the second approach, because, although the first approach allows to guarantee the validity of results for arbitrary circuit inputs, their applicability is severely limited by the effort required to adapt and extend these methods for new developments, such as new optimizations or mapping strategies.

Checking the equivalence between two circuits boils down to checking whether they indeed realize the same functionality. Mathematically, the quantum gates g_i of an n-qubit quantum circuit G are defined by $2^n \times 2^n$ unitary matrices U_i. Consequently, the functionality of a quantum circuit G with gates $g_0, \ldots, g_{m-1}$ is described by a unitary matrix U, which is obtained by consecutively multiplying the unitary matrix representations U_i of each gate g_i in reverse order, i.e., $U = U_{m-1} \cdots U_0$. Thus, checking the equivalence of two circuits G and G' amounts to building and comparing the circuits' system matrices U and U'. While simple in its concept, the exponential size of the involved matrices quickly renders many direct approaches infeasible[8].

In order to cope with this complexity, decision diagrams have been proposed as an efficient data-structure for canoni-

---

[6]In fact, the mapping task has been shown to be NP-complete [20].

[7]A $SWAP$ operation is eventually realized using three $CNOT$ operations as indicated in the middle of Fig. 1.

[8]Equivalence checking of quantum circuits has even been proven to be QMA-complete in the general case [30].

cally representing and manipulating quantum functionality in the recent past [9], [11], [12][9]. While decision diagrams indeed frequently allow to represent quantum functionality in a very compact fashion, the decision diagrams corresponding to the circuits G and G' may still grow exponentially in the worst case.

Hence, to further improve upon that, a promising approach was recently proposed in [10] utilizing the following observation: Consider two equivalent quantum circuits $G = g_0 \ldots g_{m-1}$ and $G' = g'_0, \ldots, g'_{m'-1}$. Then, due to the inherent reversibility of quantum circuits, this certainly allows for the conclusion that $G'^{-1} \cdot G = \mathbb{I}$, where G'^{-1} denotes the inverse of G' and $\mathbb{I}$ denotes the identity function. Moreover, it holds that

$$
\begin{aligned}
\mathbb{I} = G'^{-1} \cdot G &= (g'^{-1}_{m'-1} \ldots g'^{-1}_0) \cdot (g_0 \ldots g_{m-1}) \\
&\equiv (U_{m-1} \cdots U_0) \cdot (U'^{\dagger}_0 \cdots U'^{\dagger}_{m'-1}) \\
&= U_{m-1} \cdots U_0 \cdot \mathbb{I} \cdot U'^{\dagger}_0 \cdots U'^{\dagger}_{m'-1} \\
&=: G \to \mathbb{I} \leftarrow G'.
\end{aligned}
$$

As a consequence, checking whether G and G' are in fact equivalent can be conducted by starting with the identity and, then, either applying operations of G "from the left" (denoted by $G \to \mathbb{I}$) or (inverted) operations of G' "from the right" (denoted by $\mathbb{I} \leftarrow G'$) until all operations have been applied. If, afterwards, the identity still remains, the circuits G and G' have been proven to be equivalent.

The intention of this idea is to keep the intermediate computations as close to the identity as possible, since the identity constitutes the best case for most representations of quantum functionality (e.g., linear in the number of nodes with respect to the number of qubits for decision diagrams).

Example 6. *Assume, w.l.o.g, that $m \leq m'$, i.e., G' has at least as many gates as G. Further assume an* oracle $\omega \colon G \to (G')^*$ *exists that, given a gate $g_i \in G$, returns a consecutive sequence of gates $g'_k \ldots g'_l \in G'$ such that $g_i \equiv g'_k \ldots g'_l$. Then, subsequently applying one gate $g \in G$ and $|\omega(g)|$ inverted gates from G' constitutes a "perfect" strategy—yielding the identity after each pair of applications. As a result, only matrices representing, or staying close to, the identity occur. Since these can usually be represented very efficiently using, e.g., decision diagrams, the process of equivalence checking is substantially improved.*

However, a major problem remains in how to obtain the "perfect" oracle $\omega(\cdot)$, i.e., in deciding when to apply operations of G ("from the left") and when to apply operations of G' ("from the right"). In [10], several strategies for this purpose have been proposed and, indeed, substantial speed-ups in checking the equivalence of two quantum circuits have been achieved with that (cf. Table 1 in [10]). But after all, those

[9]Canonicity implies that a comparison of the root pointers of two decision diagrams allows to decide their equivalence.

strategies remain rather simple (e.g., they employ a one-to-one or size-proportional application of gates from G and G') and certainly do not resemble a "perfect" strategy which can indeed keep the computation of $G \to \mathbb{I} \leftarrow G'$ close to the identity.

In contrast, the compilation flow as reviewed in Section III-A provides detailed insights how a circuit G is eventually compiled to a circuit G'—providing ideal knowledge about how to derive the "perfect" oracle $\omega(\cdot)$. In this work, we propose a verification scheme which uses the idea of applying $G \to \mathbb{I} \leftarrow G'$ and, at the same time, utilizes the knowledge about an actual compilation flow (namely the Qiskit flow). As the experimental evaluations (summarized later in Section V) confirm, this allows for drastic speed-ups and, eventually, makes equivalence checking feasible on a large scale.

IV. PROPOSED VERIFICATION SCHEME

In this section, we propose a verification scheme which rests on the ideas discussed above, but additionally utilizes knowledge about the Qiskit quantum circuit compilation flow to derive a much better oracle $\omega(\cdot)$. For each step in the compilation flow (i.e., for synthesis, mapping, and optimization), a corresponding strategy for $\omega(\cdot)$ is derived which keeps applying $G \to \mathbb{I} \leftarrow G'$ close to the identity. Those strategies are described in the following subsections. Afterwards, they are combined to an overall scheme.

A. Utilizing Knowledge about the Synthesis Step

Considering the first step of the compilation flow, two issues become relevant for determining the "perfect" $G \to \mathbb{I} \leftarrow G'$ strategy: (1) each gate $g \in G$ is compiled to a sequence of gates $g'_k \ldots g'_l \in G'$ and (2) the circuits G and G' may operate on different numbers of qubits due to the addition of ancillary qubits required for the synthesis.

For the first issue, it can be exploited that the actual decomposition scheme, i.e., into how many elementary gates each of the original circuit's gates is decomposed, is known a priori. Thus, an *oracle* $\omega(\cdot)$ which, given a gate $g \in G$, returns the corresponding sequence of gates $g'_k \ldots g'_l \in G'$, is explicitly known in this case. Assuming that G' resulted from the synthesis of a given quantum circuit G, applying *one* gate from G and $|\omega(g)|$ inverted gates from G' constitutes an optimal strategy for conducting $G \to \mathbb{I} \leftarrow G'$—yielding the identity after each step.

Example 7. *Consider the original circuit G shown in Fig. 1a. As indicated by Fig. 1b, the Toffoli gate of G needs to be decomposed into elementary gates supported by the architecture, while all other gates of G are already supported. Thus, $|\omega(g)| = 1$ holds for all $g \in G$ except for the Toffoli gate, where $|\omega(g)| = 15$ holds.*

In case both circuits do not operate on the same number of qubits, the corresponding unitaries have different dimensions

$$U: \begin{array}{c} \\ {\scriptstyle q_{n-1}} \\ {\scriptstyle \text{to}} \end{array} \begin{array}{cc} & \text{from} \\ & |0\rangle \quad |1\rangle \\ \begin{array}{c} |0\rangle \\ |1\rangle \end{array} & \left[\begin{array}{c|c} U_{00} & U_{10} \\ \hline U_{01} & U_{11} \end{array} \right] \end{array}$$

(a) Original matrix U

$$\tilde{U}: \begin{array}{c} \\ {\scriptstyle q_{n-1}} \\ {\scriptstyle \text{to}} \end{array} \begin{array}{cc} & \text{from} \\ & |0\rangle \quad |1\rangle \\ \begin{array}{c} |0\rangle \\ |1\rangle \end{array} & \left[\begin{array}{c|c} U_{00} & 0 \\ \hline U_{01} & 0 \end{array} \right] \end{array}$$

(b) Modified matrix $\tilde{U}$

Figure 2: Handling of ancillary qubits

and cannot be applied directly. Unfortunately, it is not sufficient to match the qubit count of G' by just augmenting the original circuit with idle qubits. Since ancillary qubits are always initialized in a particular state (typically $|0\rangle$), this leaves some degree of freedom in the overall unitary representation U'. In order to compensate for this degree of freedom, the eventually resulting matrix U' has to be modified as shown in the following example.

Example 8. *Consider a unitary $2^n \times 2^n$ matrix U and assume that, w.l.o.g., the last qubit q_{n-1} acts as an ancillary qubit initialized to $|0\rangle$. In general, the action of U depending on the state of q_{n-1} is described by the four $2^{n-1} \times 2^{n-1}$ sub-matrices U_{ij} as illustrated in Fig. 2a. Since the ancillary is initialized to $|0\rangle$, the sub-matrices corresponding to the transformation from $|1\rangle$ can be ignored—resulting in the modified matrix $\tilde{U}$ shown in Fig. 2b.*

B. Utilizing Knowledge about the Mapping Step

Mapping to the targeted architecture establishes a connection between the circuit's logical and the device's physical qubits. Consequently, while the description of G is expressed in terms of logical qubits $q_0, \ldots, q_{n-1}$, the circuit G' operates on (a subset of) the device's physical qubits $Q_0, \ldots, Q_{N-1}$. If a non-trivial initial mapping (i.e., anything but $Q_i \colon q_i$) is employed, this leads to the situation that gates from G', although functionally equivalent, are applied to different qubits than the gates of G. Thus, concluding the equivalence of both circuits is not possible by straight-forwardly using the oracle function $\omega(\cdot)$. Instead, a *qubit map $m(\cdot)$* is employed, which stores the mapping between the physical qubits of the circuit G' and the logical qubits of the original circuit G, i.e., $m(Q_i) = q_j$ if physical qubit Q_i is initially assigned logical qubit q_j. Whenever a gate from G' is to be applied to a certain physical qubit Q_i, this is translated to the corresponding logical qubit $m(Q_i) = q_j$—again allowing to stay close to the identity.

Example 9. *Consider the original circuit G and the mapped circuit $\tilde{G}$ shown in Fig. 1a and Fig. 1d, respectively. While the X gate at the beginning of G is applied to the logical qubit q_2, it is applied to the physical qubit Q_1 in the circuit $\tilde{G}$. In order to fix this mismatch, the qubit map $m(\cdot)$—mapping $Q_0 \mapsto q_0$, $Q_1 \mapsto q_2$, and $Q_2 \mapsto q_1$—is employed. Consequently, the X*

gate of $\tilde{G}$ is applied to $m(Q_1) = q_2$ which now matches the original gate from G perfectly.

However, as discussed in Section III-A, the logical-to-physical qubit mapping of a compiled circuit in general changes dynamically throughout the circuit in order to satisfy all constraints imposed by the device's coupling map. As a consequence, the potential of using the (static) qubit map $m(\cdot)$ in combination with the oracle function $\omega(\cdot)$ to stay close to the identity is significantly diminished. That is, because the dynamically changed mapping again results in a scenario where gates from G' are applied to different qubits than in the circuit G. Therefore, a perfect verification strategy needs to keep track of the changes in the logical-to-physical qubit mapping caused by $SWAP$ operations[10] and, accordingly, needs to *update the qubit map $m(\cdot)$* throughout the verification procedure.

Example 10. *Consider again the scenario of Ex. 9. If the $G \to \mathbb{I} \leftarrow G'$ scheme is carried out using the qubit map $m(\cdot)$ defined there, the result would not represent the identity. That is, because the logical-to-physical qubit mapping is changed in the middle of $\tilde{G}$ by a $SWAP$ operation applied to Q_0 and Q_1. Thus, at that specific point, the qubit map $m(\cdot)$ has to be updated accordingly, i.e., it then has to map $Q_0 \mapsto q_2$, $Q_1 \mapsto q_0$, and $Q_2 \mapsto q_1$. Through this dynamic change, the computation of $G \to \mathbb{I} \leftarrow G'$ remains close to the identity and, eventually, proves the equivalence of both circuits.*

C. Utilizing Knowledge about the Optimization Step

If no optimizations were to be applied to the circuit resulting from the synthesis and mapping step (which is equivalent to applying Qiskit's O0 optimization level), the strategies proposed above allow to conduct $G \to \mathbb{I} \leftarrow G'$ in a perfect fashion—yielding the identity after each step. However, optimizations as discussed in Section III-A further alter the circuit—making it harder to verify the resulting circuit. In the following, we cover how to anticipate the effects of the two most common optimizations employed in Qiskit—*single-qubit gate fusion* and *adjacent gate cancellation* (see Section III-A).

Example 11. *Consider again the circuit $\tilde{G}$ shown in Fig. 1d. There, the grey box indicates the gates of $\tilde{G}$ realizing the Toffoli gate of the original circuit G shown in Fig. 1a. The middle qubit thereby contains a T gate, which is directly followed by an H gate. Accordingly, in the optimized circuit shown in Fig. 1e, these have been merged into a single $U_2(0, \frac{5\pi}{4})$ gate. Thus, $|\omega(g)| = 15$ does no longer hold, but has to be modified to $|\omega(g)| = 14$ instead in case of the Toffoli gate (see Ex. 7).*

In addition to anticipating fusions within individual gate realizations through adaptations of the oracle function $\omega(\cdot)$, a *pre-processing* pass is conducted which fuses consecutive

---

[10]$SWAP$s can be reconstructed from consecutive sequences of three $CNOT$s in G' as indicated in the middle of Fig. 1.
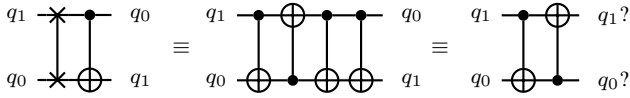
Figure 3: $SWAP$ and $CNOT$ cancellation

single-qubit gates where they are present in the original circuit G (e.g., fusing the $H - X - H$ cascade at the end of the circuit G shown in Fig. 1a to a single Z gate). However, reductions across multiple gates that were decomposed during synthesis cannot be accounted for in this fashion. Thus, the formerly constructed *perfect* oracle function $\omega(\cdot)$ becomes *approximate*.

Example 12. *Consider again the circuit $\tilde{G}$ shown in Fig. 1d and its optimized variant G' shown in Fig. 1e. Then, the cancellation of the two consecutive H gates in the beginning of $\tilde{G}$ cannot be anticipated through a straightforward adaptation of $\omega(\cdot)$. However, as also confirmed by the experimental evaluations in Section V, $\omega(\cdot)$ remains a suitable approximation for staying close to the identity.*

The second optimization employed per default—adjacent gate cancellation—introduces a peculiar issue for the verification strategy as shown in the following example.

Example 13. *Consider a $SWAP$ operation directly followed by a $CNOT$ operation. Since the $SWAP$ operation itself is realized by three consecutive $CNOTs$, this sequence of operations may be simplified by cancelling two of them as shown in Fig. 3. While the qubit map $m(\cdot)$ can be easily adapted in the first two cases, the optimized circuit shows no sign of an applied $SWAP$ and, furthermore, introduces an additional $CNOT$ gate to the compiled circuit which previously did not exist in G. This makes it hard for the proposed strategies to still identify the $SWAP$ and, hence, update the qubit map $m(\cdot)$ as described in the previous section.*

As a solution, any occurrence of two consecutive $CNOT$ operations in G' as shown on the right of Fig. 3 that is not followed by a third matching $CNOT$ is *substituted* by the sequence shown on the left of Fig. 3. Overall, this again allows to accurately track the qubit mapping $m(\cdot)$ and conduct the equivalence check in an optimal fashion.

D. Resulting Verification Scheme

All of the considerations above finally result in a dedicated verification scheme that is tailored for verifying results of the Qiskit compilation flow. First of all, a pre-processing step fuses subsequent single-qubit gates in the circuit G and substitutes $SWAP$ (and possibly a $CNOT$) gates where applicable in the circuit G'. Afterwards, if necessary, the circuit G is augmented with idle ancillary qubits. Then, the general $G \rightarrow \mathbb{I} \leftarrow G'$

scheme is employed—utilizing the oracle function $\omega(\cdot)$ to determine which gates from G' are to be applied for each application of a gate from G.

The actual application of gates from G' happens with regard to the qubit map $m(\cdot)$ which establishes the connection between the circuit G's logical qubits and G''s physical qubits. During these steps, this qubit map is dynamically updated to account for the insertion of $SWAP$ operations during the mapping. After applying all gates from both circuits, the result of this scheme is modified as illustrated in Ex. 8. Eventually, the two circuits are shown to be equivalent if the modified result resembles the identity for the non-ancillary qubits.

As confirmed by the experimental evaluations, which are summarized next, this scheme allows to efficiently verify even large instances consisting of tens of thousands of gates within seconds. Additionally, in contrast to formally verifying the individual compilation steps (see Section III-B), this approach remains generic enough to work well out of the box, even when optimizations are employed that have not been directly accounted for, e.g., commutation rules.

V. EXPERIMENTAL EVALUATIONS

The proposed verification scheme has been implemented on top of the tool proposed in [10] (which is available at https://iic.jku.at/eda/research/quantum_verification). More precisely, we took the recent version (revision 1.2) and extended this tool with the strategies described in Section IV. Afterwards, we conducted extensive experiments to evaluate the performance of the resulting approach. In this section, we summarize our evaluations. To this end, we first briefly review the setup and, afterwards, present as well as discuss the obtained results.

A. Setup

In our evaluations, we considered circuits that are frequently used to benchmark compilers. Using IBM Qiskit [1] (specifically, Qiskit Terra 0.12.0), each original circuit G has been compiled for a specific target device—resulting in an alternative circuit G'. During this process, multi-controlled Toffoli gates have been synthesized using the "*basic*" mode (yielding the smallest circuits, with the highest number of additionally needed qubits) and the target device has been chosen as the smallest possible one capable of accommodating the resulting number of qubits. Specifically, *IBM Q Boeblingen* has been chosen for circuits with up to 20 qubits, while *IBM Q Rochester* has been used for circuits with up to 53 qubits.

As discussed in Section III-A, Qiskit offers several optimization levels for compiling circuits. In our evaluations, we considered Qiskit's default optimization level (i.e., O1) and—in order to show the proposed approach's applicability to scenarios it has not been explicitly tailored towards—the more advanced O2 level. All evaluations have been performed on a 4 GHz *Amazon EC2 z1d* instance running Ubuntu 18.04

with at least 32 GB per job using GNU Parallel [31]. A hard timeout of 1 h (i.e., 3 600 s) was set for each run.

B. Obtained Results

In a first series of evaluations, we considered Qiskit's default optimization level O1. A subset of the respectively obtained results is shown in Table I[11]. Here, the first columns describe the original circuit G (its name, number of qubits, and number of gates) as well as the device the circuit was mapped to. Then, the size of the resulting circuit G' is listed, along with the runtimes of (1) the equivalence checking routine from [11], (2) its advanced improvement from [10] (utilizing $G \to \mathbb{I} \leftarrow G'$)[12], and (3) the strategy proposed in this work.

The results clearly show the superiority of the proposed method. While the first approach using $G \to \mathbb{I} \leftarrow G'$ (as proposed in [10]) allows to reduce the equivalence checking runtime down to a third or a half in most cases (sometimes even more), substantial runtimes (or even timeouts) are still reported. On the contrary, additionally exploiting explicit knowledge about the compilation flow as proposed in this work allows for drastic further improvements. In fact, *all* considered instances are successfully verified within (fractions of) seconds, whereas state-of-the-art equivalence checking methods and even the recently proposed advanced techniques frequently time-out or require substantial runtime.

In a second series of evaluations, we considered Qiskit's more advanced optimization level O2. In this regard, Table II shows a representative subset of the obtained results[13]. While the proposed methodology was explicitly tailored for the default optimization level of Qiskit, i.e., O1, its performance remains almost on an equally high level in case of verifying circuits compiled with optimization level O2—where several more advanced optimization techniques, such as gate commutation rules, are employed, which are not directly accounted for in the proposed scheme. This shows that even utilization of partial knowledge about the underlying compilation flow is sufficient to drastically improve the verification of the correctness of its result.

VI. Conclusion

In this work, we proposed a dedicated scheme for verifying results of the IBM Qiskit compilation flow. To this end, we exploit characteristics unique to quantum computing and explicitly incorporate knowledge about the compilation flow in order to design a strategy that allows to keep the overhead of verifying compilation results minimal. Experimental evaluations confirm that the proposed strategy consistently allows to verify instances with more than ten-thousand gates within seconds—even if optimizations are employed which are not directly accounted for. Compared to the state of the art, which often requires substantial runtimes or even time-outs in these tasks, this is a drastic improvement. The resulting tool is publicly available at https://github.com/iic-jku/qcec and can easily be adapted to different compilation flows or additional optimizations in the future.

References

[1] G. Aleksandrowicz *et al.*, "Qiskit: An open-source framework for quantum computing", 2019.

[2] *Cirq: A python framework for creating, editing, and invoking Noisy Intermediate Scale Quantum (NISQ) circuits.* [Online]. Available: https://github.com/quantumlib/Cirq (visited on 04/10/2020).

[3] *Quantum Development Kit*, Microsoft. [Online]. Available: https://microsoft.com/en-us/quantum/development-kit (visited on 03/25/2020).

[4] *Forest SDK*, Rigetti. [Online]. Available: https://rigetti.com/forest (visited on 04/10/2020).

[5] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Checking equivalence of quantum circuits and states", in *Int'l Conf. on CAD*, 2007.

[6] S. Yamashita and I. L. Markov, "Fast equivalence-checking for quantum circuits", in *Int'l Symp. on Nanoscale Architectures*, 2010.

[7] K. N. Smith and M. A. Thornton, "A quantum computational compiler and design tool for technology-specific targets", in *Int'l Symp. on Computer Architecture*, 2019, pp. 579–588.

[8] P. Niemann, R. Wille, and R. Drechsler, "Equivalence checking in multi-level quantum systems", in *Int'l Conf. of Reversible Computation*, 2014.

[9] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, and S.-Y. Kuo, "An XQDD-based verification method for quantum circuits", in *IEICE Trans. Fundamentals*, 2008, pp. 584–594.

[10] L. Burgholzer and R. Wille, "Advanced equivalence checking of quantum circuits", *ArXiv: 2004.08420 [quant-ph]*, 2020.

[11] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler, "QMDDs: Efficient quantum function representation and manipulation", *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 86–99, 2016.

[12] A. Zulehner, S. Hillmich, and R. Wille, "How to efficiently handle complex values? Implementing decision diagrams for quantum computing", in *Int'l Conf. on CAD*, 2019.

[13] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[14] IBM Q, *IBM Q*. [Online]. Available: https://www.research.ibm.com/ibm-q/ (visited on 04/10/2020).

---

[11]Due to space limitations, only a small subset of benchmarks is listed here. However, the proposed scheme is publicly available at https://github.com/iic-jku/qcec to conduct further evaluations.

[12]The "*Proportional*" strategy was used, as it produced the best results in [10].

[13]In some instances, the circuit resulting from optimization level O2 is actually larger than the circuit resulting from O1. This is due to the fact that the default $SWAP$-insertion technique employed in Qiskit is stochastic. However, this does not influence the general observations gained from this evaluation, since the actual number of $SWAP$s actually does not influence the performance of the proposed strategy.

Table I: Optimization Level O1

Benchmark				Results			
Name	n	$|G|$	Architecture	$|G'|$	t_{sota} [s]	t_{adv} [s]	t_{prop} [s]
sym9_317	27	64	Rochester	1 540	>3 600.00	63.61	**0.03**
c2_182	35	305	Rochester	2 545	>3 600.00	2 500.79	**0.10**
cm163a_213	29	39	Rochester	3 539	>3 600.00	183.93	**0.02**
apla_203	22	80	Rochester	13 548	>3 600.00	>3 600.00	**0.10**
cm151a_211	28	33	Rochester	3 889	>3 600.00	>3 600.00	**0.03**
rd84_313	34	113	Rochester	2 295	>3 600.00	>3 600.00	**2.64**
mod5adder_306	32	110	Rochester	2 348	>3 600.00	>3 600.00	**0.22**
add6_196	19	229	Rochester	25 460	1 513.95	514.19	**0.51**
cycle17_3_112	20	48	Rochester	14 987	1 111.38	1 284.86	**0.34**
rd73_312	25	76	Rochester	1 426	1 108.62	196.10	**0.20**
cm150a_210	22	53	Rochester	3 788	761.82	577.95	**0.05**
ryy6_256	17	44	Rochester	13 103	504.42	316.13	**0.08**
c2_181	35	116	Rochester	2 708	403.80	4.06	**0.03**
alu3_200	18	94	Rochester	10 285	255.25	151.17	**0.07**
decod_217	21	80	Rochester	6 197	192.30	121.17	**0.05**
example2_231	16	157	Rochester	19 411	187.63	79.62	**0.21**
cu_219	25	40	Rochester	5 031	177.15	72.20	**0.02**
plus63mod8192_164	13	492	Rochester	111 720	166.02	112.40	**1.85**
C7552_205	21	80	Rochester	6 384	158.30	80.38	**0.05**
alu2_199	16	157	Rochester	19 007	139.00	68.86	**0.17**
dk17_224	21	49	Rochester	6 650	105.19	36.56	**0.04**
mlp4_245	16	131	Rochester	14 513	79.06	22.69	**0.09**
mux_246	22	35	Rochester	3 679	71.17	39.75	**0.04**
urf1_150	9	1 517	Boeblingen	134 216	68.45	43.96	**9.52**
clip_206	14	174	Rochester	20 872	62.91	43.62	**0.20**
plus63mod4096_163	12	429	Rochester	82 195	58.47	44.01	**1.46**
dist_223	13	185	Boeblingen	19 458	19.93	12.08	**0.13**
inc_237	16	93	Rochester	7 523	18.02	8.46	**0.04**
urf5_159	9	499	Boeblingen	57 099	15.99	9.17	**0.90**
alu1_198	20	32	Rochester	1 377	13.19	4.02	**0.01**
cmb_214	20	18	Rochester	2 589	11.47	10.76	**0.01**
sqr6_259	18	81	Rochester	4 436	10.89	4.50	**0.02**
5xp1_194	17	85	Rochester	5 487	10.78	4.59	**0.04**
rd84_253	12	111	Boeblingen	7 857	9.99	6.78	**0.05**
root_255	13	99	Boeblingen	8 594	7.97	5.15	**0.05**

n: Number of qubits $|G|$: Gate count of G *Architecture*: Boeblingen (20 qubits) or Rochester (53 qubits)

$|G'|$: Gate count of G' t_{sota}: Runtime of state-of-the-art EC routine [11]

t_{adv}: Runtime of advanced methodology EC routine [10] t_{prop}: Runtime of proposed, dedicated EC scheme

[15] L. K. Grover, "A fast quantum mechanical algorithm for database search", *Proc. of the ACM*, pp. 212–219, 1996.

[16] A. Barenco *et al.*, "Elementary gates for quantum computation", *Phys. Rev. A*, vol. 52, no. 5, pp. 3457–3467, 1995.

[17] D. Maslov, "On the advantages of using relative phase Toffolis with an application to multiple control Toffoli optimization", *Phys. Rev. A*, vol. 93, no. 2, p. 022 311, 2016.

[18] R. Wille, M. Soeken, C. Otterstedt, and R. Drechsler, "Improving the mapping of reversible circuits to quantum circuits using multiple target lines", in *Asia and South Pacific Design Automation Conf.*, 2013.

[19] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for Noisy Intermediate-Scale Quantum computers", in *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1015–1029.

[20] M. Y. Siraichi, V. F. dos Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation", in *Proc. Int'l Symp. on Code Generation and Optimization*, 2018, pp. 113–125.

[21] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the IBM QX architectures", *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2019.

[22] R. Wille, L. Burgholzer, and A. Zulehner, "Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations", in *Design Automation Conf.*, 2019.

[23] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for NISQ-era quantum devices", in *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2019.

[24] A. Matsuo, W. Hattori, and S. Yamashita, "Reducing the overhead of mapping quantum circuits to IBM Q system", in *IEEE International Symposium on Circuits and Systems*, 2019.

[25] M. Amy and V. Gheorghiu, "Staq – A full-stack quantum processing toolkit", *ArXiv:1912.06070*, 2019.

[26] 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.

Table II: Optimization Level O2

Benchmark				Results			
Name	n	$\|G\|$	Architecture	$\|G'\|$	t_{sota} [s]	t_{adv} [s]	t_{prop} [s]
sym9_317	27	64	Rochester	1 566	>3 600.00	81.40	**0.03**
c2_182	35	305	Rochester	2 692	963.18	99.00	**0.22**
cm163a_213	29	39	Rochester	3 704	2 004.05	515.69	**0.03**
apla_203	22	80	Rochester	13 212	>3 600.00	>3 600.00	**0.11**
cm151a_211	28	33	Rochester	3 798	>3 600.00	>3 600.00	**0.04**
rd84_313	34	113	Rochester	2 158	>3 600.00	>3 600.00	**12.47**
mod5adder_306	32	110	Rochester	2 380	>3 600.00	>3 600.00	**0.37**
add6_196	19	229	Rochester	26 201	455.70	138.65	**1.29**
cycle17_3_112	20	48	Rochester	13 876	777.82	667.16	**1.80**
rd73_312	25	76	Rochester	1 498	199.84	226.25	**0.65**
cm150a_210	22	53	Rochester	3 678	795.15	583.42	**0.07**
ryy6_256	17	44	Rochester	13 100	22.17	23.40	**0.20**
c2_181	35	116	Rochester	3 271	>3 600.00	>3 600.00	**0.04**
alu3_200	18	94	Rochester	10 463	44.81	27.40	**0.11**
decod_217	21	80	Rochester	6 289	35.55	23.60	**0.05**
example2_231	16	157	Rochester	19 207	38.57	16.77	**0.62**
cu_219	25	40	Rochester	4 863	136.76	48.83	**0.03**
plus63mod8192_164	13	492	Rochester	111 258	110.21	78.31	**8.61**
C7552_205	21	80	Rochester	6 291	60.18	32.07	**0.04**
alu2_199	16	157	Rochester	19 080	23.56	12.42	**0.27**
dk17_224	21	49	Rochester	6 589	7.70	2.99	**0.05**
mlp4_245	16	131	Rochester	14 394	12.41	4.59	**0.17**
mux_246	22	35	Rochester	3 829	156.01	91.49	**0.06**
urf1_150	9	1 517	Boeblingen	132 898	75.19	44.03	**21.89**
clip_206	14	174	Rochester	21 407	44.16	23.82	**0.26**
plus63mod4096_163	12	429	Rochester	83 835	67.01	45.24	**4.71**
dist_223	13	185	Boeblingen	18 721	28.09	16.70	**0.24**
inc_237	16	93	Rochester	7 624	55.03	42.83	**0.05**
urf5_159	9	499	Boeblingen	56 089	15.52	8.77	**3.00**
alu1_198	20	32	Rochester	1 424	14.56	1.42	**0.01**
cmb_214	20	18	Rochester	2 669	15.03	15.31	**0.02**
sqr6_259	18	81	Rochester	4 536	5.31	2.03	**0.03**
5xp1_194	17	85	Rochester	5 420	6.89	3.52	**0.04**
rd84_253	12	111	Boeblingen	7 715	4.48	2.77	**0.06**
root_255	13	99	Boeblingen	8 729	6.10	3.82	**0.06**

n: Number of qubits $\quad$ $|G|$: Gate count of G $\quad$ *Architecture*: Boeblingen (20 qubits) or Rochester (53 qubits)

$|G'|$: Gate count of G' $\quad$ t_{sota}: Runtime of state-of-the-art EC routine [11]

t_{adv}: Runtime of advanced methodology EC routine [10] $\quad$ t_{prop}: Runtime of proposed, dedicated EC scheme

[28] K. Hietala, R. Rand, S.-H. Hung, X. Wu, and M. Hicks, "A verified optimizer for quantum circuits", *ArXiv:1912.02250*, 2019.

[29] Y. Shi *et al.*, "Contract-based verification of a realistic quantum compiler", *ArXiv:1908.08963*, 2019.

[30] D. Janzing, P. Wocjan, and T. Beth, ""Non-identity check" is QMA-complete", *Int. J. Quantum Inform.*, vol. 03, no. 03, pp. 463–473, 2005.

[31] O. Tange, *GNU Parallel 2018*. Ole Tange, 2018.

[27] G. Vidal and C. M. Dawson, "Universal quantum circuit for two-qubit transformations with three controlled-NOT gates", *Phys. Rev. A*, vol. 69, no. 1, p. 010301, 2004.