An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures

Alwin Zulehner†, Student Member, IEEE, Alexandru Paler, and Robert Wille‡, Senior Member, IEEE

Abstract—In the past years, quantum computers more and more have evolved from an academic idea to an upcoming reality. IBM’s project IBM Q can be seen as evidence of this progress. Launched in March 2017 with the goal to provide access to quantum computers for a broad audience, this allowed users to conduct quantum experiments on a 5-qubit and, since June 2017, also on a 16-qubit quantum computer (called IBM QX2 and IBM QX3, respectively). Revised versions of these 5- and 16-qubit quantum computers (named IBM QX4 and IBM QX5, respectively) are available since September 2017. In order to use these, the desired quantum functionality (e.g., provided in terms of a quantum circuit) has to be properly mapped so that the underlying physical constraints are satisfied—a complex task. This demands solutions to automatically and efficiently conduct this mapping process. In this paper, we propose a methodology which addresses this problem, i.e., maps the given quantum functionality to a realization which satisfies all constraints given by the architecture and, at the same time, keeps the overhead in terms of additionally required quantum gates minimal. The proposed methodology is generic, can easily be configured for similar future architectures, and is fully integrated into IBM’s SDK. Experimental evaluations show that the proposed approach clearly outperforms IBM’s own mapping solution. In fact, for many quantum circuits, the proposed approach determines a mapping to the IBM architecture within minutes, while IBM’s solution suffers from long runtimes and runs into a timeout of 1 h in several cases. As an additional benefit, the proposed approach yields mapped circuits with smaller costs (i.e., fewer additional gates are required). All implementations of the proposed methodology are publicly available at http://iic.jku.at/eda/research/ibm_qx_mapping.

Index Terms—IBM QX, Mapping, Quantum circuits.

I. INTRODUCTION

Q UANTUM computers and quantum algorithms have received lots of interests in the past—of course, mainly motivated by their ability to solve certain tasks significantly faster than classical algorithms [1]–[4]. These quantum algorithms are described by so-called quantum circuits, a sequence of gates that are applied to the qubits of a quantum computer. While theoretical algorithms have already been developed in the last century (e.g., [2]–[4]), physical realizations have been considered “dreams of the future” for a long time. This changed in recent years in which quantum computers more and more evolved from an academic idea to an upcoming reality. IBM’s project IBM Q [5], which launched in March 2017 with the goal to provide access to a quantum computer to the broad audience, can be seen as evidence of this progress. Initially, they started with the 5-qubit quantum processor IBM QX2, on which anyone could run experiments through cloud access. In June 2017, IBM added a 16-qubit quantum processor named IBM QX3 to their cloud [6] and, thus, more than tripled the number of available qubits within a few months. Since then, IBM has been working intensely on improving their quantum computers—leading to 5- and 16-qubit quantum computers (named IBM QX4 and IBM QX5, respectively) which were added to the cloud in September 2017. The rapid progress in the number of available qubits is still going on. While IBM has already manufactured a 20-qubit quantum computer which is available for their partners and members of the IBM Q network, as well as a prototype of a 50-qubit processor, other well-known companies like Google have also announced the intent to manufacture quantum chips with 49 qubits (using architectures as described in [7]) in the near future to show quantum supremacy [8], [9].

However, in order to use these physical realizations, the desired quantum functionality to be executed has to properly be mapped so that the underlying physical constraints are satisfied. This constitutes a complex task. One issue is that the desired functionality (usually described by higher level components) has to be decomposed into elementary operations supported by the IBM QX architectures. Furthermore, there exist physical limitations, namely that certain quantum operations can only be applied to selected physical qubits of the IBM QX architectures. Consequently, the logical qubits of a quantum circuit have to be mapped to the physical qubits of the quantum computer such that all operations can be conducted. Since it is usually not possible to determine a mapping such that all constraints are satisfied throughout the whole circuit, this mapping may change over time. To this end, additional gates, e.g., realizing SWAP operations, are inserted in order to “move” the logical qubits to other physical ones. They affect the reliability of the circuit (each further gate increases the
In this paper, we propose a corresponding methodology. To designers, e.g., in the classical domain, take for granted today, and cannot generate a result in acceptable time. The above motivates a solution that is as efficient as circuit designers, e.g., in the classical domain, take for granted today. In this paper, we propose a corresponding methodology. To this end, a multistep approach is introduced which utilizes a depth-based partitioning and as underlying search algorithm as well as further optimizations such as a look-ahead scheme and the ability to determine the initial mapping of the qubits throughout the mapping process (instead of fixing the initial mapping at the beginning of the algorithm). The resulting methodology is generic, i.e., it can directly be applied to all existing QX architectures as well as similar upcoming architectures which may come in the future (and architectures whose constraints can be formulated in a similar way). Finally, we integrated the methodology into IBM’s Python SDK QISKit—allowing for a more realistic performance evaluation since post-mapping optimizations provided by IBM are additionally considered. Experimental evaluations confirmed the benefits and allowed for an explicit analysis of the effects of the respective optimizations incorporated into the proposed methodology. The results clearly show that the methodology is able to cope with the complexity of satisfying the constraints discussed above. Using this solution, QX-compatible mappings for many quantum circuits can be determined within minutes, while IBM’s own solution suffers from long runtimes and runs into a timeout of 1 h in these cases. Moreover, as an additional benefit, realizations with smaller costs (i.e., fewer additional gates) are obtained. All implementations are publicly available at http://iic.jku.at/eda/research/ibm_qx_mapping and, as mentioned above, have been integrated into IBM’s own SDK—resulting in an advanced and integrated mapping scheme for the QX architectures provided by IBM.

This paper is structured as follows. In Section II, we review quantum circuits as well as the IBM QX architectures. In Section III, we discuss the process to map a given quantum circuit to the IBM QX architectures. How to particularly cope with the problem of satisfying the additional constraints is covered in Section IV. In Section V, the performance of the proposed mapping scheme is analyzed and compared to the performance of the solution provided by IBM. Section VI concludes this paper.

---

1A preliminary version of this paper is available at [22].

---

**II. BACKGROUND**

In this section, we briefly review the basics of quantum circuits and the IBM QX architectures.

### A. Quantum Circuits

Classical computations and circuits use bits as information units. In contrast, quantum circuits perform their computations on qubits [1]. These qubits cannot only be in one of the two basis states \(|0\rangle\) or \(|1\rangle\), but also in a superposition of both—allowing for the representation of all possible \(2^n\) basis states of \(n\) qubits concurrently. This so-called quantum parallelism serves as basis for algorithms that are significantly faster on quantum computers than on classical machines.

To this end, the qubits of a quantum circuit are manipulated by quantum operations represented by so-called quantum gates. These operations can either operate on a single qubit, or on multiple ones. For multiqubit gates, we distinguish target qubits and control qubits. The value of the target qubits is modified in the case that the control qubits are set to basis state \(|1\rangle\). The Clifford+T library [10], which is composed of the single-qubit gates Hadamard gate \((H)\) and phase shift by \(\pi/4\) \((T)\), as well as the two-qubit gate controlled NOT \((CNOT)\), represents a universal set of quantum operations (i.e., all quantum computations can be implemented by a circuit composed of gates from this library).

To describe quantum circuits, high level quantum languages (e.g., Scaffold [23] or Quipper [24]), quantum assembly languages (e.g., OpenQASM 2.0 developed by IBM [25]), or circuit diagrams are employed. In the following, we use the latter to describe quantum circuits (but the proposed approach has also been applied using the other descriptions as well). In a circuit diagram, qubits are represented by horizontal lines, which are passed through quantum gates. In contrast to classical circuits, this, however, does not describe a connection of wires with a physical gate, but defines (from left to right) in which order the quantum gates are applied to the qubits.

**Example 1:** Fig. 1 shows the circuit diagram of a quantum circuit. The quantum circuit is composed of three qubits and five gates. The single-qubit gates \(H\) and \(T\) are represented by boxes labeled with \(H\) and \(T\), respectively, while the control and target qubit of the CNOT gate are represented by \(\bullet\) and \(\oplus\), respectively. First, a Hadamard operation is applied to qubit \(q_0\). Then, a CNOT operation with target \(q_1\) and control qubit \(q_0\) is conducted—followed by a \(T\)-gate that is applied to \(q_2\). Finally, two more CNOTs are applied.

### B. IBM’s QX Architectures

In this paper, we consider how to efficiently map a quantum circuit to the IBM QX architectures provided by the project IBM Q [5]. IBM provides a Python SDK named QISKIT [21]...
III. MAPPING OF QUANTUM CIRCUITS TO THE IBM QX ARCHITECTURES

Mapping quantum circuits to the IBM QX architectures requires the consideration of two major issues. On the one hand, all gates of the given quantum circuit to be mapped have to be decomposed to elementary operations supported by the hardware, i.e., CNOTs and parameterized U gates. On the other hand, the \( n \) logical qubits \( q_0, q_1, \ldots, q_{n-1} \) of that quantum circuit have to be mapped to the \( m \) physical qubits \( Q_0, Q_1, \ldots, Q_{m-1} (m = 5 \) for QX2 and QX4, whereas \( m = 16 \) for QX3 and QX5) of the IBM QX architecture. Each logical qubit has to be represented by a physical one, such that all CNOT-constraints are satisfied. In this section, we describe how these two issues can be handled in an automatic fashion, what problems occur during this process, and how they can be addressed.

A. Decomposing Quantum Circuits to Elementary Operations

Considering the first issue, IBM has developed the quantum assembly language OpenQASM [25] that supports specification of quantum circuits. Besides elementary gates, the language allows the definition of complex gates that are composed from the elementary operations CNOT and \( U \). These gates can then be nested to define even more complex gates. Consequently, as long as a decomposition of the gates used in a description of the desired quantum functionality are provided by the circuit designer, the nested structures are just flattened during the mapping process.

In case the desired quantum functionality is not provided in OpenQASM, decomposition or synthesis approaches such as those proposed in [10]–[13] and [26]–[28], respectively, can be applied which determine (e.g., depth optimal) realizations of quantum functionality for specific libraries like Clifford+T [10] or NCV [29]. They typically use search algorithms or a matrix representation of the quantum functionality. For the Clifford+T library, Matsumoto and Amano [12] developed a normal form for single qubit operations, which allows for a unique and T-depth optimal decomposition (approximation) of arbitrary single qubit gates (e.g., rotations) into a sequence of Clifford+T gates (up to a certain error \( \epsilon \)). Several such automated methods are available in Quipper (a functional programming language for quantum computing [24]), the ScaffCC compiler for the Scaffold language [23], [30], and RevKit [31]. Since IBM provides the decomposition for commonly used gates like the Clifford+T gates, (controlled) rotations, or Toffoli gates to their gate library, these approaches can be utilized.

**Example 2:** One commonly used operation is the SWAP operation, which exchanges the states of two qubits. Since the SWAP operation is not part of the gate library of IBM’s QX architectures, it has to be decomposed into single-qubit gates and CNOTs as shown in Fig. 3. Assume that logical qubits \( q_0 \) and \( q_1 \) are initially mapped to the physical qubits \( Q_0 \) and \( Q_1 \) of QX2, and that their values are to be swapped. As a first decomposition step, we realize the SWAP operation with three CNOTs. If we additionally consider the CNOT-constraints, we have to flip the direction of the CNOT in the middle. To
this end, we apply Hadamard operations before and after this CNOT. These Hadamard operations then have to be realized by the gate \( U(\pi/2, 0, \pi) = H \).

Hence, decomposing the desired quantum functionality to the elementary gate library is already well covered by corresponding related work. Unfortunately, this is not the case for the second issue, which is discussed next.

B. Satisfying CNOT-Constraints

Recall that, in order to satisfy the CNOT-constraints as defined in Section II-B, the \( n \) logical qubits \( q_0, q_1, \ldots, q_{n-1} \) of the quantum circuit to be realized have to be mapped to the \( m \) physical qubits \( Q_0, Q_1, \ldots, Q_{m-1} \) \((m = 5 \) for \( QX2 \) and \( QX4 \), whereas \( m = 16 \) for \( QX3 \) and \( QX5 \)) of the IBM QX architecture. Usually, there exists no mapping solution that satisfies all CNOT-constraints throughout the whole circuit (this is already impossible if CNOT gates are applied to qubit pairs \((q_h, q_i), (q_h, q_j), (q_h, q_k), \) and \((q_h, q_l)\) with \( h \neq i \neq j \neq k \neq l \)). That is, whatever initial mapping might be imposed at the beginning, it may have to be changed during the execution of a quantum circuit (namely exactly when a gate is to be executed which violates a CNOT-constraint). To this end, \( H \) and SWAP gates can be applied to change the direction of a CNOT gate and to change the mapping of the logical qubits, respectively. In other words, these gates can be used to “move” around the logical qubits on the actual hardware until the CNOT-constraints are satisfied. An example illustrates the idea.

Example 3: Consider the quantum circuit composed of five CNOT gates shown in Fig. 4(a) and assume that the logical qubits \( q_0, q_1, q_2, q_3, q_4, \) and \( q_5 \) are, respectively, mapped to the physical qubits \( Q_0, Q_1, Q_2, Q_3, Q_4, \) and \( Q_5 \) of the IBM QX3 architecture shown in Fig. 2(c). The first gate can directly be applied, because the CNOT-constraint is satisfied. For the second gate, the direction has to be changed because a CNOT with control qubit \( Q_0 \) and target \( Q_1 \) is valid, but not vice versa. This can be accomplished by inserting Hadamard gates as shown in Fig. 4(b). For the third gate, we have to change the mapping. To this end, we insert SWAP operations \( \text{SWAP}(Q_1, Q_2) \) and \( \text{SWAP}(Q_2, Q_3) \) to move logical qubit \( q_1 \) toward logical qubit \( q_4 \) (see Fig. 4(b)). Afterwards, \( q_1 \) and \( q_4 \) are mapped to the physical qubits \( Q_3 \) and \( Q_1 \), respectively, which allows us to apply the desired CNOT gate. Following this procedure for the remaining qubits eventually results in the circuit shown in Fig. 4(b).

However, inserting the additional gates in order to satisfy the CNOT-constraints drastically increases the number of operations—a significant drawback which affects the reliability of the quantum circuit since each gate has a certain error rate. Since each SWAP operation is composed of seven elementary gates (see Fig. 3), particularly their number shall be kept as small as possible. Besides that, the circuit depth shall be kept as small as it is related to the time required to execute the quantum circuit. Since a SWAP operation has a depth of 5, this also motivates the search for alternative solutions which realize a CNOT-constraint-compliant mapping with as few SWAP operations as possible.

Example 4: Consider again the given quantum circuit from Fig. 4(a) as well as its mapping derived in Example 3 and shown in Fig. 4(b). This circuit is composed of 51 elementary gates and has a depth of 36. In contrast, the same quantum circuit can be realized with only 23 elementary operations and has a depth of 10 as shown in Fig. 4(c) \((g_2 \) and \( g_3 \) can be applied concurrently)—a significant reduction.

Determining proper mappings has similarities with recent work on nearest neighbor optimization of quantum circuits proposed in [14]–[20].\(^2\) In that work, SWAP gates have also been applied to move qubits together in order to satisfy a physical constraint. However, these works consider simpler and artificial architectures with 1-D or 2-D layouts where any two-qubit gate can be applied to adjacent qubits. The CNOT-constraints to be satisfied for the IBM QX architectures are much stricter with respect to what physical qubits may interact with each other and also what physical qubit may act as control and as target qubit. Furthermore, the parallel execution of gates (which is possible in the QX architectures) is not considered by these approaches. Besides that, there exists a recent approach that utilizes temporal planning to compile quantum circuits to real architectures [32]. However, this approach is rather specialized to Quantum Alternating Operator Ansatz [33] circuits for solving the MaxCut problem and target the architectures proposed by Sete et al. [34]. As a consequence, none of the approaches discussed above is directly applicable for the problem considered here.

As a further alternative, IBM provides a solution within its SDK [21]. This algorithm randomly searches (guided by heuristics) for mappings of the qubits at a certain point of time. These mappings are then realized by adding SWAP gates to the circuit. But this random search is hardly feasible for many quantum circuits and, hence, is not as efficient as circuit designers, e.g., in the conventional domain, take for granted today. In fact, in many cases the provided method is not capable of determining a CNOT-constraint-compliant mapping within 1\( h \) (see Section V)—an issue which will become more serious when further architectures with more qubits are introduced.

Overall, automatically and efficiently mapping quantum circuits to the IBM QX architectures particularly boils down to the question how to efficiently determine a mapping of logical qubits to physical qubits which satisfy the CNOT-constraints. How this problem can be addressed is covered in the next section.

\(^2\) These approaches utilize satisfiability solvers, search algorithms, or dedicated data structures to tackle the underlying complexity.
IV. Efficiently Satisfying CNOT-Constraints

In this section, we propose an efficient method for mapping a given quantum circuit (which has already been decomposed into a sequence of elementary gates as described in Section III-A) to the IBM QX architectures. The main objective is to minimize the number of elementary gates which are added in order to make the mapping CNOT-constraint-compliant. Two main steps are employed: first, the given circuit is partitioned into layers which can be realized in a CNOT-constraint-compliant fashion. Afterwards, for each of these layers, a particular compliant mapping is determined which requires as few additional gates as possible. In the following sections, both steps are described in detail. Afterwards, further optimizations are proposed to reduce the costs of the resulting circuit.

A. Partitioning the Circuit Into Layers

As mentioned above, the mapping from logical qubits to physical ones may change over time in order to satisfy all CNOT-constraints, i.e., the mapping may have to change before a CNOT can be applied. Since each change of the mapping requires additional SWAP operations, we aim for conducting these changes as rarely as possible. To this end, we combine gates that can be applied concurrently into so-called layers (i.e., sets of gates). A layer contains only gates that act on distinct sets of qubits. Furthermore, this allows us to determine a mapping such that the CNOT-constraints for all gates are satisfied at the same time. We form the layers in a greedy fashion, i.e., we add a gate to the layer where it is as small as possible. In the circuit diagram representation, this means to move all gates to the left as far as possible without changing the order of gates that share a common qubit. Note that the depth of a circuit is equal to the number of layers of a circuit.

Example 5: Consider again the quantum circuit shown in Fig. 4(a). The gates of the circuit can be partitioned into three layers which can be realized in a CNOT-constraint-compliant fashion. Afterwards, for each of these layers, a particular compliant mapping is determined which requires as few additional gates as possible.

B. Determining Compliant Mappings for the Layers

For each layer , we now determine all mappings describing to which physical qubit a logical qubit is mapped. The starting point is an initial mapping which is denoted by and obtained from the previous layer (i.e., for a randomly generated initial mapping that satisfies all CNOT constraints for the gates g belonging to .) Now, this initial mapping should be changed to the desired mapping which is denoted by , is CNOT-constraint-compliant for all gates g belonging to , and can be established from with minimum costs, i.e., the minimum number of additionally required elementary operations. In the worst case, determining requires the consideration of possibilities (where m and n are the number of physical qubits and logical qubits, respectively) — an exponential complexity. We cope with this complexity by applying an A* search algorithm.

The A* algorithm [35] is a state-space search algorithm. To this end, (sub)solutions of the considered problem are represented by state nodes. Nodes that represent a solution are called goal nodes (multiple goal nodes may exist). The main idea is to determine the cheapest path (i.e., the path with the lowest cost) from the root node to a goal node. Since the search space is typically exponential, sophisticated mechanisms are employed in order to keep considering as few paths as possible.

All state-space search algorithms are similar in the way they start with a root node (representing an initial partial solution) which is iteratively expanded toward the goal node (i.e., the desired complete solution). How to choose the node to be expanded next depends on the actual search algorithm. For A* search, we determine the cost of each leaf-node of the search space. Then, the node with the lowest cost is chosen to be expanded next. To this end, we determine the cost of a node x. The first part describes the cost of the current subsolution (i.e., the cost of the path from the root to x). The second part describes the remaining cost (i.e., the cost from x to a goal node), which is estimated by a heuristic function . Since the node with the lowest cost is expanded, some parts of the search space (those that lead to expensive solutions) are never expanded.
Example 6: Consider the tree shown in Fig. 5. This tree represents the part of the search space that has already been explored for a certain search problem. The nodes that are candidates to be expanded in the next iteration of the A* algorithm are highlighted in blue. For all these nodes, we determine the cost \( f(x) = g(x) + h(x) \). This sum is composed by the cost of the path from the root to the node \( x \) (i.e., the sum of the cost annotated at the respective edges) and the estimated cost of the path from node \( x \) to a goal node (provided in red). Consider the node labeled \( E \). This node has cost \( f(E) = (40+60)+200 = 300 \). The other candidates labeled \( B \), \( C \), and \( F \) have cost \( f(B) = 580 \), \( f(C) = 360 \), and \( f(F) = 320 \), respectively. Since the node labeled \( E \) has the fewest expected cost, it is expanded next.

Obviously, the heuristic cost should be as accurate as possible, to expand as few nodes as possible. If \( h(x) \) always provides the correct minimal remaining cost, only the nodes along the cheapest path from the root node to a goal node would be expanded. But since the minimal costs are usually not known (otherwise, the search problem would be trivial to solve), estimations are employed. However, to ensure an optimal solution, \( h(x) \) has to be admissible, i.e., \( h(x) \) must not overestimate the cost of the cheapest path from \( x \) to a goal node. This ensures that no goal node is expanded (which terminates the search algorithm) until all nodes that have the potential to lead to a cheaper solution are expanded.

Example 6 (Continued): Consider again the node labeled \( E \). If \( h(x) \) is admissible, the true cost of each path from this node to a goal node is greater than or equal to 200.

To use the A* algorithm for our search problem, an expansion strategy for a state (i.e., a mapping \( \sigma^j \)) as well as an admissible heuristic function \( h(x) \) to estimate the distance of a state to a goal state (i.e., the mapping \( \hat{\sigma}^l \)) are required. Given a mapping \( \sigma^j \), we can determine all possible successor mappings \( \sigma^j_h \) by employing all possible combinations of SWAP gates that can be applied concurrently.\(^3\) The fixed costs of all these successor states \( \sigma^j_h \) is then \( f(\sigma^j_h) = f(\sigma^j) + 7 \cdot \#\text{SWAPs} \) since each SWAP gate is composed of seven elementary operations (three CNOTs and four Hadamard operations). Note that we can restrict the expansion strategy to SWAP operations that affect at least one qubit that occurs in a CNOT gate \( g \in l_i \) on layer \( l_i \). This is justified by the fact that only these qubits influence whether or not the resulting successor mapping is CNOT-constraint-compliant.

Example 7: Consider again the quantum circuit shown in Fig. 4(a) and assume we are searching for a mapping for layer \( l_1 = \{ g_2, g_3 \} \). In the previous layer \( l_0 \), the logical qubits \( q_1, q_3, q_4, \) and \( q_5 \) have been mapped to the physical qubits \( Q_0, Q_3, Q_{14}, \) and \( Q_{15} \), respectively (i.e., \( \sigma^0 \)). This initial mapping \( \sigma^0 \) does not satisfy the CNOT-constraints for the gates in \( l_1 \). Since we only consider four qubits in the CNOTs of \( l_1 \), \( \sigma^0 \) has only 51 successors \( \sigma^1 \).

As mentioned above, to obtain an optimal mapping (i.e., the mapping with the fewest additionally required elementary operations that satisfies all CNOT-constraints), we need a heuristic function that does not overestimate the real cost (i.e., the minimum number of additionally inserted elementary operations) for reaching \( \hat{\sigma}^l \) from \( \sigma^1 \).

The real minimum costs for an individual CNOT gate \( g \in l_1 \) can easily be determined given \( \sigma^1 \). First, we determine the physical qubits \( Q_s \) and \( Q_t \) to which the control and target qubit of \( g \) are mapped (which is given by \( \sigma^1 \)). Using the coupling map of the architecture (see Fig. 2), we then determine the shortest path (following the arrows in the coupling map\(^4\)) \( \hat{p} \) from \( Q_s \) to \( Q_t \). The costs of the CNOT gate \( h(g, \sigma^1) = (|\hat{p}| - 1) \cdot 7 \) are then determined by the length of this shortest path \( |\hat{p}| \). In fact, \( (|\hat{p}| - 1) \) SWAP operations are required to move the control and target qubits of \( g \) toward each other. If none of the arrows of the path \( \hat{p} \) on the coupling map (representing that a CNOT can be applied) points into the desired direction, we have to increase the true minimum costs further by 4, since two Hadamard operations are required before and after the CNOT to change its direction.

The heuristic costs of a mapping \( \sigma^1 \) can be determined from the real costs of each CNOT gate \( g \in l_1 \) in layer \( l_1 \). Simply summing them up might overestimate the true cost, because one SWAP operation might reduce the distance of the control and target qubits for more than one CNOT of layer \( l_1 \). Since this would prevent us from determining the optimal solution \( \hat{\sigma}^l \), we instead determine the heuristic costs of a state \( \sigma^1 \) as \( h(\sigma^1) = \max_{g \in l_1} h(g, \sigma^1) \), i.e., the maximum of the true costs of the CNOTs in layer \( l_1 \).

Example 7 (Continued): The logical qubits \( q_1 \) and \( q_4 \) are mapped to the physical qubits \( \sigma^1_0(q_1) = Q_1 \) and \( \sigma^1_0(q_4) = Q_{14} \), respectively. Since the shortest path on the coupling map is \( \hat{p} = Q_1 \rightarrow Q_2 \rightarrow Q_3 \rightarrow Q_{14} \) (see Fig. 2), the true minimum costs for \( g_2 \) is \( h(g_2, \sigma^1_0) = 2 \cdot 7 = 14 \). Analogously, the costs of \( g_3 \) can be determined to be \( h(g_3, \sigma^1_0) = 7 \) resulting in overall heuristic costs of \( h(\sigma^1_0) = \max(14,7) = 14 \) for the initial mapping. Following the A* algorithm outlined above, we eventually determine a mapping \( \hat{\sigma}^l \) that maps the logical qubits \( q_0, q_1, q_2, q_3, q_4, \) and \( q_5 \) to the physical qubits \( Q_0, Q_2, Q_1, Q_4, Q_3, \) and \( Q_5 \) by inserting two SWAP operations (as depicted in Fig. 6). Applying the algorithm also for mapping layer \( l_2 \), the circuit shown in Fig. 6 results. This circuit is composed of 37 elementary operations and has depth 15.

\(^3\)Note that we apply multiple SWAP gates concurrently in order to minimize the circuit depth as second criterion (if two solutions require the same number of additional operations).

\(^4\)The direction of the arrow does not matter since a SWAP can be applied between two physical qubits iff a CNOT can be applied.
C. Optimizations

$A^*$ allows us to efficiently determine an optimal mapping (by means of additionally required operations) for each layer. However, the algorithm proposed in Section IV-B considers only a single layer when determining $\hat{d}^l$ for layer $l_i$.

One way to optimize the proposed solution is to employ a look-ahead scheme which incorporates information from the following layers to the cost function. To this end, we only have to change the heuristics to estimate the costs for reaching a mapping that satisfies all CNOT-constraints from the current one. In Section IV-B, we used the maximum of the costs for each CNOT gate in layer $l_i$ to estimate the true remaining cost. For the look-ahead scheme, we additionally determine an estimate for layer $l_{i+1}$. The overall heuristic that guides the search algorithm toward a solution is then the sum of both estimates.

To incorporate the look-ahead scheme, we change the heuristics discussed in Section IV-B. Instead of taking the maximum of the CNOTs in the current layer, we sum up the costs of all CNOTs in two layers (the current and the look-ahead layer), i.e., $h(\sigma^j) = \sum_{g \in \sigma^j} h(g, \sigma^j)$. As discussed above, this might lead to an over-estimation of the true remaining costs for reaching a goal state and, thus, the solution is not guaranteed to be locally optimal. However, this is not desired anyways, since we want to allow locally suboptimal solutions in order to find cheaper mappings for the following layers—resulting in smaller overall circuits.

Example 8: Consider again the quantum circuit shown in Fig. 4(a) and assume that the logical qubits $Q_0$, $Q_1$, $Q_2$, $Q_3$, $Q_4$, and $Q_5$ are mapped to the physical qubits $Q_0$, $Q_1$, $Q_2$, $Q_3$, $Q_{14}$, and $Q_{15}$, respectively. Using the look-ahead scheme discussed above will not determine the locally optimal solution with costs of 14 for layer $l_1$ (as discussed in Example 7), but a mapping $\hat{d}^l$ that satisfies all CNOT-constraints with costs of 22 (as show in Fig. 7). The additional costs of 8 result since, after applying two SWAP gates (see Fig. 7), the directions of both CNOTs of layer $l_1$ have to change. However, this mapping also satisfies all CNOT-constraints for layer $l_2$, which means that the remaining CNOT $g_4$ can be applied without adding further SWAPs. The resulting circuit is composed of a total of 31 elementary operations and has depth of 12 (as shown in Fig. 7; gates $g_2$ and $g_3$ can be applied concurrently). Consequently, the look-ahead scheme results in a cheaper mapping than the “pure” methodology proposed in Section IV-B and yielding the circuit shown in Fig. 6.\(^5\)

Besides the look-ahead scheme, we can further improve the methodology by not starting with a random mapping for layer $l_0$. Instead, we propose to use partial mappings $\sigma^j_0$ and to start with an empty mapping $\sigma^0_0$ (i.e., none of the logical qubits is mapped to a physical one). Then, before we start to search a mapping for layer $l_1$, we check whether the qubits that occur in the CNOTs $g \in l_i$ have already been mapped for one of the former layers. If not, we can freely chose one of the “free” physical qubits (i.e., a physical qubit no logical qubit is mapped to). Obviously, we choose the physical qubit so that the cost for finding $\hat{d}^l$ is as small as possible.

This scheme gives us the freedom to evolve the initial mapping throughout the mapping process, rather than starting with an initial mapping that might be nonbeneficial with respect to the overall number of elementary operations.

Example 9: Optimizing the methodology with a partial mapping that is initially empty results in the circuit already shown before in Fig. 4(c). This circuit is composed of 23 elementary operations and has depth 10 (gates $g_2$ and $g_3$ can be applied concurrently).

V. EXPERIMENTAL EVALUATION

Taking all considerations and methods discussed above into account led to the development of a mapping methodology which decomposes arbitrary quantum functionality into elementary quantum gates supported by the QX architectures and, afterwards, maps them so that all CNOT-constraints are satisfied. As mentioned above, IBM’s Python SDK QISKit already implements most of these steps, but lacks an efficient methodology for mapping the circuits such that all CNOT-constraints are satisfied. To overcome this issue, we have implemented the mapping methodology presented in this paper in C++ and integrated it into QISKit. The adapted version of QISKit as well as a standalone version of the methodology are publicly available at http://iic.jku.at/eda/research/ibm_qx_mapping.

In this section, we compare the efficiency of the resulting scheme to the original design flow implemented in QISKit [21]. To this end, several functions taken from RevLib [36] as well as quantum algorithms written in Quipper [24] or the Scaffold language [23] (and precompiled by the ScaffoldCC compiler [30]) have been considered as benchmarks and mapped to the most recent 16-qubit architecture available (i.e., $QX5$).\(^6\) Besides that, benchmarks that are relevant for existing quantum algorithms such as quantum algorithms...
ripple-carry adders (based on the realization proposed in [37] and denoted *adder*) and small versions of Shor’s algorithm (based on the realization proposed in [38] and denoted *shor*) have been considered. All evaluations have been conducted on a 4.2-GHz machine with four cores (two hardware threads each) and 32-GB RAM.

### A. Effect of the Optimizations

In a first series of evaluations, we evaluate the improvements gained by the optimizations discussed in Section IV-C. The corresponding numbers are listed in Table I. For each benchmark, we provide the name, the number of logical qubits $n$, the number of gates $g$, as well as the depth of the circuit $d$, before mapping the circuit to the IBM QX5 architecture. In the remainder of the table, we list the results provided by the proposed methodology, i.e., the number of gates $g$ and the depth of the circuit $d$ after mapping it to the IBM QX5 architecture as well as the time required to determine that mapping (in CPU seconds).

Three different settings of the methodology are thereby considered. As baseline serves the approach proposed in...
Section IV that uses an A\textsuperscript{*} algorithm to determine locally optimal mappings for each layer of the circuit (denoted base-line in the following). Furthermore, we list the numbers when enriching the baseline with a look-ahead scheme as discussed in Section IV-C (denoted look-ahead in the following). Finally, we also list the resulting numbers for the fully optimized methodology that uses a look-ahead scheme and additionally allows for evolving the mapping throughout the mapping process as discussed in Section IV-C (denoted fully optimized in the following). The timeout was set to 1 h.

Table I clearly shows the improvements that can be gained by applying the optimizations discussed in Section IV-C.

| Methodology | n | g | d | \text{avg} | \sigma \text{avg} | \text{avg} | \text{std} | \text{avg} | \text{std} | \text{avg} | \text{std} | \text{avg} | \text{std} | \text{avg} | \text{std} | \text{avg} | \text{std} |
|-------------|---|---|---|----------|---------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| baseline    | 3 | 1 | 1 | 31.0     | 31.0          | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     |
| look-ahead  | 3 | 1 | 1 | 31.0     | 31.0          | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     |
| fully optimized | 3 | 1 | 1 | 31.0     | 31.0          | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     | 62.0     |

For IBM's solution, we list the obtained minimum, the average, and the standard deviation of \text{run} (denoted \text{min}, \text{avg}, and \text{std}, respectively).

Section IV that uses an A\textsuperscript{*} algorithm to determine locally optimal mappings for each layer of the circuit (denoted base-line in the following). Furthermore, we list the numbers when enriching the baseline with a look-ahead scheme as discussed in Section IV-C (denoted look-ahead in the following). Finally, we also list the resulting numbers for the fully optimized methodology that uses a look-ahead scheme and additionally allows for evolving the mapping throughout the mapping process as discussed in Section IV-C (denoted fully optimized in the following). The timeout was set to 1 h.

Table I clearly shows the improvements that can be gained by applying the optimizations discussed in Section IV-C.
gates and the depth of the circuits slightly increase to 19.7% and 14.1%, respectively (compared to Baseline).

Overall, the optimizations discussed in Section IV-C not only increase the scalability of the mapping algorithm outlined in Section IV-B, but—as a positive side effect—also reduce the size of the resulting circuit.

B. Comparison to the State of the Art

In a second series of evaluation, we compare the proposed mapping methodology to the solution provided by IBM via QISKit. A fair comparison of both mapping solution is guaranteed since we incorporated the mapping algorithm discussed in this paper into QISKit. Hence, the same decomposition schemes as well as the same post-mapping optimizations are applied in both cases.

Table II lists the, respectively, obtained results. For each benchmark, we again list the name, the number of logical qubits \( n \), the number of gates \( g \), and the depth \( d \) of the quantum circuit before mapping it to the IBM QX5 architecture. In the remaining columns, we list the number of gates, the depth, and the runtime \( t \) (in CPU seconds) for IBM’s solution as well as for the solution proposed in this paper. Since IBM’s mapping algorithm searches for mappings that satisfy all CNOT-constraints randomly (guided by certain heuristics), we conducted the mapping procedure five times for each benchmark and list the obtained minimum, the average (denoted by subscripts \( \min \) and \( \text{avg} \), respectively), as well as the standard deviation \( \sigma \) for each of the listed metrics. The timeout for searching a single mapping was again set to 1 h.

The results clearly show that the proposed solution can efficiently tackle the considered mapping problem—in particular compared to the method available thus far. While IBM’s solution runs into the timeout of 1 h in 10 out of 60 cases, the proposed algorithm determines a mapping for each circuit within the given time limit. Besides that, the approach is frequently magnitudes faster compared to IBM’s solution.

Besides efficiency, the proposed methodology for mapping a quantum circuit to the IBM QX architectures also yields circuits with significantly fewer gates than the results determined by IBM’s solution. In fact, the solution proposed in Section IV results on average in circuits with 24.0% fewer gates and 18.3% fewer depth on average compared to the minimum observed when running IBM’s algorithm several times. Compared to the average results yield by IBM’s solution, we obtain improvements of 27.5% and 22.0% for gate count and circuit depth, respectively.

VI. Conclusion

In this paper, we proposed an advanced and integrated methodology that efficiently maps a given quantum circuit to IBM’s QX architectures. To this end, the desired quantum functionality is first decomposed into the supported elementary quantum gates. Afterwards, CNOT-constraints imposed by the architecture are satisfied. Particular the later step caused a nontrivial task for which an efficient solution based on a depth-based partitioning, an \( A^* \) search algorithm, a look-ahead scheme, as well as a dedicated initialization of the mapping has been proposed. The resulting approach eventually allows us to efficiently map quantum circuits to real quantum hardware and has been integrated into IBM’s SDK QISKit. The efficiency has been confirmed by experimental evaluations. The proposed approach was able to determine a mapping for quantum circuits within seconds in most cases whereas IBM’s solution requires more than 1 h to determine a solution for several cases. As a further positive side effect, the mapped circuits have significantly fewer gates and smaller circuit depth, which positively influences the reliability and the runtime of the circuit. The resulting methodology is generic, i.e., it can be directly applied to all existing QX architectures as well as similar architectures which may come in the future. All implementations are publicly available at http://iic.jku.at/eda/research/ibm_qx_mapping.

References

[1] M. A. Nielsen and I. L. Chung, Quantum Computation and Quantum Information. Cambridge, U.K.: Cambridge Univ. Press, 2000.
[2] P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” SIAM J. Comput., vol. 26, no. 5, pp. 1484–1509, 1997, doi: 10.1137/s0097539795293172.
[3] L. K. Grover, “A fast quantum mechanical algorithm for database search,” in Proc. Symp. Theory Comput., 1996, pp. 212–219. [Online]. Available: http://doi.acm.org/10.1145/237814.237866.
[4] D. Deutsch and R. Josza, “Rapid solution of problems by quantum computation,” Proc. Roy. Soc. A Math. Phys. Sci., vol. 439, no. 1907, pp. 553–558, 1992.
[5] IBM Q. Accessed: Sep. 15, 2017. [Online]. Available: https://www.research.ibm.com/ibm-q/
[6] IBM Q Backend Information. Accessed: Sep. 15, 2017. [Online]. Available: https://github.com/QISKit/ibmqx-backend-information.
[7] C. Neill et al., “A blueprint for demonstrating quantum supremacy with superconducting qubits,” Science, vol. 360, no. 6385, pp. 195–199, 2018.
[8] S. Boixo et al., “Characterizing quantum supremacy in near-term devices,” Nat. Phys., vol. 14, no. 6, p. 595, 2018.
[9] R. Courtland, “Google aims for quantum computing supremacy [news],” IEEE Spectr., vol. 54, no. 6, pp. 9–10, Jun. 2017.
[10] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, “A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 32, no. 6, pp. 818–830, Jun. 2013, doi: 10.1109/TCAD.2013.2244643.
[11] D. M. Miller, R. Wille, and Z. Sasanian, “Elementary quantum gate realizations for multiple-control Toffoli gates,” in Proc. Int. Symp. Multi Valued Logic, 2011, pp. 288–293.
[12] K. Matsumoto and K. Amano, “Representation of quantum circuits with clifford and \( \pi/8 \) gates,” arXiv preprint arXiv:0806.3834, 2008.
[13] R. Wille, M. Soeken, C. Otterstedt, and R. Drechsler, “Improving the mapping of reversible circuits to quantum circuits using multiple target lines,” in Proc. Asia South Pac. Design Autom. Conf., 2013, pp. 85–92.
[14] R. Wille, A. Lye, and R. Drechsler, “Exact reordering of circuit lines for nearest neighbor quantum architectures,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 33, no. 12, pp. 1818–1831, Dec. 2014, doi: 10.1109/TCAD.2014.2356463.
[15] M. Saeedi, R. Wille, and R. Drechsler, “Synthesis of quantum circuits for linear nearest neighbor architectures,” Quantum Inf. Process., vol. 10, no. 3, pp. 355–377, 2011, doi: 10.1007/s11128-010-0201-2.
[16] R. Wille et al., “Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits,” in Proc. Asia South Pac. Design Autom. Conf., 2016, pp. 292–297, doi: 10.1109/ASPDAC.2016.7428026.
[17] A. Shafaei, M. Saeedi, and M. Pedram, “Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures,” in Proc. Design Autom. Conf., Austin, TX, USA, 2013, pp. 41–46.
[18] A. Shafaei, M. Saeedi, and M. Pedram, “Qubit placement to minimize communication overhead in 2D quantum architectures,” in Proc. Asia South Pac. Design Autom. Conf., 2014, pp. 495–500, doi: 10.1109/ASPDAC.2014.6742940.
[19] R. Wille, N. Quetschlich, Y. Inoue, N. Yasuda, and S.-I. Minato, “Using \( \pi \) DDs for nearest neighbor optimization of quantum circuits,” in Proc. Int. Conf. Reversible Comput., 2016, pp. 181–196, doi: 10.1007/978-3-319-40578-0_14.
Alwin Zulehner (S’17) received the B.Sc. and M.Sc. degrees in computer science from Johannes Kepler University Linz, Linz, Austria, in 2012 and 2015, respectively, where he is currently pursuing the Ph.D. degree with the Institute for Integrated Circuits.

His current research interests include design automation for emerging technologies, currently focusing on reversible circuits and quantum circuits. In these areas, he has published several papers on international conferences and journals, such as the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, the Asia and South Pacific Design Automation Conference, the Design, Automation and Test in Europe, and the International Conference on Reversible Computation.

Alessandra Paler received the Ph.D. degree from the University of Potsdam, Germany.

He is a Post-Doctoral Fellow with Johannes Kepler University Linz, Linz, Austria. His current research interests include algorithms for compiling and assembling large-scale error corrected quantum circuits.

Robert Wille (M’06–SM’09) received the Diploma and Dr.-Ing. degrees in computer science from the University of Bremen, Bremen, Germany, in 2006 and 2009, respectively.

He was with the Group of Computer Architecture, University of Bremen, from 2006 to 2015, and has been with the German Research Center for Artificial Intelligence (DFKI), Bremen, since 2013. He was a Lecturer with the University of Applied Science Bremen, and a Visiting Professor with the University of Potsdam, Potsdam, Germany, and Technical University Dresden, Dresden, Germany. Since 2015, he has been a Full Professor with Johannes Kepler University Linz, Linz, Austria. His current research interests include design of circuits and systems for both conventional and emerging technologies. He has published over 200 papers in journals and conferences in the above areas.

Dr. Wille has served in Editorial Board and Program Committee of numerous journals/conferences, such as the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the Asia and South Pacific Design Automation Conference, the Design Automation Conference, the Design, Automation and Test in Europe, and the International Conference on Computer Aided Design.