Efficient Mapping of Quantum Circuits to the IBM QX Architectures

Alwin Zulehner[1] Alexandru Paler[1,2] Robert Wille[1]
[1]Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
[2]Linz Institute of Technology, Johannes Kepler University Linz, Austria
alwin.zulehner@jku.at alexandru.paler@jku.at robert.wille@jku.at

***Abstract*—In March 2017, IBM launched the project *IBM Q* 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). In order to use these, the desired quantum functionality (e.g. provided in terms of a quantum circuit) has to properly be mapped so that the underlying physical constraints are satisfied – a complex task. This demands for solutions to automatically and efficiently conduct this mapping process. In this paper, we propose such an approach which satisfies all constraints given by the architecture and, at the same time, aims to keep the overhead in terms of additionally required quantum gates minimal. The proposed approach is generic and can easily be configured for future architectures. 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 less than five minutes (and within a fraction of a second in most cases), while IBM's solution suffers from long runtimes and runs into a timeout of 1 hour in several cases. As an additional benefit, the proposed approach yields mapped circuits with smaller costs (i.e. fewer additional gates are required).**

I. Introduction

In the past there has been a lot of research on quantum algorithms that allow 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 the theoretical algorithms have already been developed in the last century (e.g. [2]–[4]), physical realizations have not been *publicly* available for researchers.

This changed in March 2017 when IBM launched its project *IBM Q* with the goal to provide access to a quantum computer to the broad audience [5]. 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.

This rapid progress in the number of available qubits as well as the predictions for providing a quantum computer with 50 qubits by the end of 2017 (also Google announced to manufacture a quantum chip with 49 qubits by the end of 2017 to show quantum supremacy [7]) demand for design automation in order to allow for an efficient use of these quantum computers. In fact, mapping a quantum circuit to a real quantum computer constitutes a non-trivial task.

This work has partially been supported by the European Union through the COST Action IC1405 and the Linz Institute of Technology (CHARON).

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 be applied to selected physical qubits of the *IBM QX* architectures only. 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 potential for errors during the quantum computation) as well as the execution time of the quantum algorithm. Hence, their number should be kept as small as possible.

While there exist several methods to address the first issue, i.e. how to efficiently map higher level components to elementary operations (see [8]–[10]), there is hardly any work on how to efficiently satisfy the additional constraints for these new and real architectures. Although there are similarities with recent work on nearest neighbor optimization of quantum circuits as proposed in [11]–[15], they are not applicable since simplistic architectures with 1-dimensional or 2-dimensional layouts are assumed there which have significantly less restrictions. Even IBM's own solution, which is provided by means of a Python SDK [16] fails in many cases since the random search employed there does not cope with the underlying complexity and cannot generate a result in acceptable time.

All that motivates an approach that is as efficient as circuit designers e.g. in the classical domain take for granted today. In this work, we propose such an approach based on a depth-based partitioning, an A* search algorithm, a look-ahead scheme, as well as a dedicated initialization of the mapping. Experimental evaluations show that the proposed approach is capable to cope with the complexity of satisfying the constraints discussed above. Thus, it can determine a mapping for many quantum circuits within less than five minutes (and within a fraction of a second for most cases), while IBM's solution suffers form long runtimes and runs into a timeout of 1 hour in several cases. Additionally equipped with a decomposition method to transform arbitrary quantum functionality to elementary operations, this results in a comprehensive mapping scheme for the QX architectures provided by IBM. The implementation of this mapping scheme is made publicly available at [17].

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 compared to the performance of the solution provided by IBM. Section VI concludes the paper.

II. Background

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

A. Quantum Circuits

While classical computations and circuits use bits as information unit, quantum circuits perform their computations on qubits [1]. These qubits can not only be in one of the two basis states $|0\rangle$ or $|1\rangle$, but also in a superposition of both – allowing to represent 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 multi-qubit 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 [8], which is composed of the single-qubit gates *H* (Hadamard gate) and *T* (Phase shift by $\pi/4$), as well as the two-qubit gate *CNOT* (controlled not), 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 [18] or Quipper [19]), quantum assembly languages (e.g. OpenQASM 2.0 developed by IBM [20]), or circuit diagrams are employed. In the following, we use the latter to describe quantum circuits (but the proposed approach has accordingly 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.

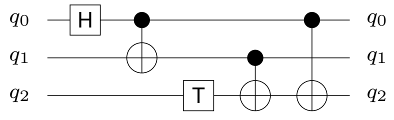


Fig. 1: Circuit diagram of a quantum circuit

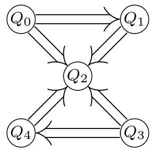
Example 1. *Fig. 1 shows the circuit diagram of a quantum circuit. The quantum circuit is composed of three qubits and five gates. The single-qubit gates* H *and* T *are represented by boxes labeled with H and T, respectively, while the control and target qubit of the CNOT gate are represented by* $\bullet$ *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

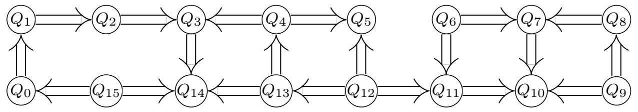
In this work, we consider how to efficiently map a quantum circuit to the *IBM QX* architectures provided by the project *IBM Q* [5]. IBM provides a Python SDK [16] that allows to describe quantum circuits, to simulate them, and to execute them on the real device (a so-called *backend*) in their cloud. The first backend composed of 5 qubits and called *IBM QX2* was launched in March 2017. In June 2017, IBM launched a second one called *IBM QX3* which is composed of 16 physical qubits that are connected with coplanar waveguide bus resonators [6]. Quantum operations are conducted by applying microwave pulses to the qubits.

The IBM QX architectures support the elementary single qubit operation $U(\theta, \phi, \lambda) = R_z(\phi)R_y(\theta)R_z(\lambda)$ (i.e. an Euler decomposition) that is composed by two rotations around the z-axis and one rotations around the y-axis, as well as the CNOT operation. By adjusting the parameters θ, ϕ, and λ, single-qubit operations of other gate libraries like the H or the T gate (cf. Section II-A) can be realized (among others like rotations).

However, there are significant restrictions which have to be satisfied when running quantum algorithms on these architectures. In fact, the user first has to decompose all non-elementary quantum operations (e.g. Toffoli gate, SWAP gate, or Fredkin gate) to the elementary operations $U(\theta, \phi, \lambda)$ and $CNOT$. Moreover, two-qubit gates, i.e. CNOT gates, cannot arbitrarily be placed in the architecture but are restricted to dedicated pairs of qubits only. Even within these pairs, it is firmly defined what qubit is supposed to work as target and what qubit is supposed to work as control. These restrictions are given by the so-called *coupling-map* illustrated in Fig. 2, which sketches the layout of the currently available *IBM QX* architectures. The circles indicate physical qubits (denoted by Q_i) and arrows indicate the possible CNOT applications, i.e. an arrow pointing from physical qubit Q_i to qubit Q_j defines that a CNOT with control qubit Q_i and target qubit Q_j can be applied. In the following, these restrictions are called *CNOT-constraints* and need to be satisfied in order to execute a quantum circuit on an QX architecture.



(a) *IBM QX2*



(b) *IBM QX3*

Fig. 2: Coupling map of the *IBM QX* architectures [6]

III. Mapping of Quantum Circuits to the IBM QX Architectures

Mapping quantum circuits to the IBM QX architectures requires to consider 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 $m = 16$ for QX3) 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 [20] that allows to specify quantum circuits. Besides elementary gates, the language allows to define 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 proposed in [8]–[10] can be applied which determine (e.g. depth optimal) realizations of quantum functionality for specific libraries like *Clifford+T* [8] or *NCV* [21]. They typically use search algorithms or a matrix representation of the quantum functionality. For the *Clifford+T* library, Matsumoto and Amano 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 ϵ) [10]. Several such automated methods are available in Quipper (a functional programming language for quantum computing [19]), the ScaffCC compiler for the Scaffold language [18], [22], and RevKit [23]. Since IBM provides the decomposition for commonly used gates like the Clifford+T gates, (controlled) rotations, or Toffoli gates to their gate library, also these approaches can be utilized.

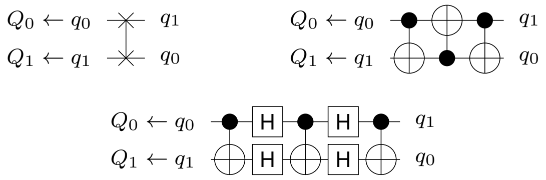


Fig. 3: Decomposition of SWAP gates

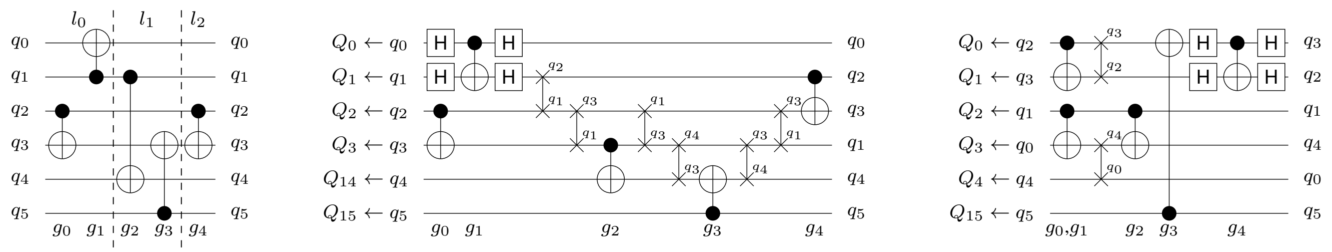
Example 2. *One commonly used operation is the SWAP operation, which exchanges the states of two qubits. Since the SWAP operation is not part of the gate library of IBM's QX architectures, it has to be decomposed into single-qubit gates and CNOTs as shown in Fig. 3. Assume that logical qubits q_0 and q_1 are initially mapped to the physical qubits Q_0 and Q_1, and that their values shall be swapped. As 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 $m = 16$ for QX3) of the IBM QX architecture. Usually, there exists no mapping solution that satisfies all CNOT-constrains 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 5 CNOT gates shown in Fig. 4a 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_{14}, and Q_{15} of the*



(a) Original circuit (b) Naive strategy (c) Proposed strategy

Fig. 4: Mapping of a quantum circuit to the *IBM QX3* architecture

IBM QX3 *architecture shown in Fig. 2b. 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. 4b. For the third gate, we have to change the mapping. To this end, we insert SWAP operations $SWAP(Q_1, Q_2)$ and $SWAP(Q_2, Q_3)$ to move logical qubit q_1 towards logical qubit q_4 (see Fig. 4b). Afterwards, q_1 and q_4 are mapped to the physical qubits Q_3 and Q_{14}, respectively, which allows to apply the desired CNOT gate. Following this procedure for the remaining qubits eventually results in the circuit shown in Fig. 4b.*

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 7 elementary gates (cf. 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. 4a as well as its mapping derived in Example 3 and shown in Fig. 4b. This circuit is composed of 51 elementary operations and has a depth of 36. In contrast, the same quantum circuit can be realized with only 23 elementary operations and depth of 10 as shown in Fig. 4c (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 [11]–[15]. Also here, SWAP gates have been applied to move qubits together in order to satisfy a physical constraint. However, these works consider more simplistic architectures with 1-dimensional or 2-dimensional 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, also the parallel execution of gates (which is possible in the QX architectures) is not considered by these approaches. As a consequence, none of these approaches is applicable for the problem considered here.

As a further alternative, also IBM itself provides a solution within its SDK [16]. 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 to determine a CNOT-constraint-compliant mapping within 1 hour (cf. Section V) – an issue which will become more serious when further architectures with more qubits will be introduced (which are already announced for the end of 2017).

Overall, automatically and efficiently mapping quantum circuits to the IBM QX architectures particularily 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 addresses is covered in the next section.

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 respectively compliant mapping is determined which requires as few additional gates as possible. In the following subsections, 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 l_i contains only gates that act on distinct sets of qubits. Furthermore, this allows to determine a mapping such that the CNOT-constraints for all gates $g_j \in l_i$ are satisfied at the same time. We form the layers in a greedy fashion, i.e. we add a gate to the layer l_i where i 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. 4a. The gates of the circuit can be partitioned into three layers $l_0 = \{g_0, g_1\}$, $l_1 = \{g_2, g_3\}$, and $l_2 = \{g_4\}$ (indicated by the dashed lines in Fig. 4a).*

To satisfy all CNOT constraints, we have to map the logical qubits of each layer l_i to physical ones. Since the resulting mapping for layer l_i does not necessarily have to be equal to the mapping determined for the previous layer l_{i-1}, we additionally need to insert SWAP operations that permute the logical qubits from the mapping for layer l_{i-1} to the desired mapping for layer l_i. In the following, we call this sequence of SWAP operations *permutation layer* π_i. The mapped circuit is then an interleaved sequence of the layers l_i of the original circuit, and the according permutation layers π_i, i.e. $l_0\pi_1 l_1 \pi_2 l_2 \ldots$.

B. Determining Compliant Mappings for the Layers

For each layer l_i, we now determine all mappings $\sigma_j^i : \{q_0, q_1, \ldots q_{n-1}\} \rightarrow \{Q_0, Q_1, \ldots Q_{m-1}\}$ describing to which physical qubit a logical qubit is mapped. The starting point is an initial mapping which is denoted by σ_0^i and obtained from the previous layer l_{i-1}, i.e. $\sigma_0^i = \hat{\sigma}^{i-1}$ (for l_0, a randomly generated initial mapping that satisfies all CNOT constraints for the gates $g \in l_0$ is used). Now, this initial mapping σ_0^i should be changed to the desired mapping which is denoted by $\hat{\sigma}^i$, is CNOT-constraint-compliant for all gates $g \in l_i$, and can be established from σ_0^i with minimum costs, i.e. the minimum number of additionally required elementary operations. In the worst case, determining $\hat{\sigma}^i$ requires the consideration of $m!/(m-n)!$ 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.

A^* is a family of search algorithm that can be used (with an appropriate heuristic) to determine $\hat{\sigma}^i$. The general idea of the algorithm is to expand the mapping σ_j^i with the smallest estimated costs $c(\sigma_j^i)$ (i.e. all its successor mappings are determined) until a $\hat{\sigma}^i$ is reached. The costs $c(\sigma_j^i) = f(\sigma_j^i) + h(\sigma_j^i)$ of a mapping is thereby composed of the *fixed cost* (i.e. the costs for reaching σ_j^i from the initial mapping σ_0^i) and the *heuristic cost* (i.e. the estimated costs for reaching $\hat{\sigma}^i$ from σ_j^i). An optimal solution is guaranteed if the estimated costs are always less than or equal to the true costs (but as close as possible to the true costs in order to determine the optimal solution quickly).

Given a mapping σ_j^i, we can determine all possible successor mappings σ_h^i by employing all possible combinations of SWAP gates that can be applied concurrently.[1] The fixed costs of all these successor states σ_h^i is then $f(\sigma_h^i) = f(\sigma_j^i) + 7 \cdot \#SWAPS$ since each SWAP gate is composed of 7 elementary operations (3 CNOTs and 4 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 6. *Consider again the quantum circuit shown in Fig. 4a and assume we are searching for a mapping for layer $l_1 = \{q_2, q_3\}$. 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. $\hat{\sigma}^0$). This initial mapping $\sigma_0^1 = \hat{\sigma}^0$ does not satisfy the CNOT-constraints for the gates in l_1. Since we only consider four qubits in the CNOTs of l_1, σ_0^i has only 51 successors σ_j^i.*

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 that does not overestimate the real cost (i.e. the minimum number of additionally inserted elementary operations) for reaching $\hat{\sigma}^i$ from σ_j^i.

The real minimum costs for an individual CNOT gate $g \in l_i$ can easily be determined given σ_j^i. 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 σ_j^i). Using the coupling map of the architecture (cf. Fig. 2), we then determine the shortest path (following the arrows in the coupling map[2]) $\hat{p}$ from Q_s to Q_t. The costs of the CNOT gate $h(g, \sigma_j^i) = (|\hat{p}| - 1) \cdot 7$ 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 towards 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 2 Hadamard operations are required before and after the CNOT to change its direction.

The heuristic costs of a mapping σ_j^i can be determined from the real costs of each CNOT gate $g \in l_i$ in layer l_i. Simply summing them up might overestimate the true cost, because one SWAP operations might reduce the distance of the control and target qubits for more than one CNOTs of layer l_i. Since this would prevent us from determining the optimal solution

[1]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).

[2]The direction of the arrow does not matter since a SWAP can be applied beween two physical qubits iff a CNOT can be applied.

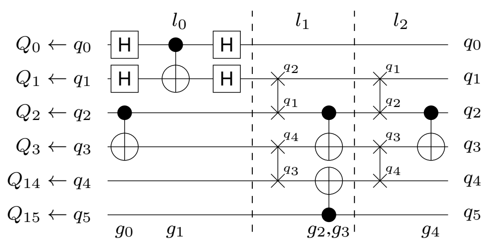


Fig. 5: Circuit resulting from locally optimal mappings

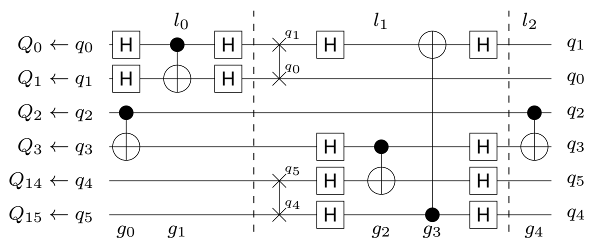


Fig. 6: Circuit generated when using the look-ahead scheme

$\hat{\sigma}^i$, we instead determine the heuristic costs of a state σ_j^i as $h(\sigma_j^i) = \max_{g \in l_i} h(g, \sigma_j^i)$, i.e. the maximum of the true costs of the CNOTS in layer l_i.

Example 6 (continued)**.** *The logical qubits q_1 and q_4 are mapped to the physical qubits $\sigma_0^1(q_1) = Q_1$ and $\sigma_0^1(q_4) = Q_{14}$, respectively. Since the shortest path on the coupling map is $\hat{p} = Q_1 \rightarrow Q_2 \rightarrow Q_3 \rightarrow Q_{14}$ (cf. Fig. 2), the true minimum costs for g_2 is $h(g_2, \sigma_0^1) = 2 \cdot 7 = 14$. Analogously, the costs of g_3 can be determined to be $h(g_3, \sigma_0^1) = 7$ – resulting in overall heuristic costs of $h(\sigma_0^1) = \max(14, 7) = 14$ for the initial mapping. Following the A* algorithm outlined above, we eventually determine a mapping $\hat{\sigma}^1$ 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. 5). Applying the algorithm also for mapping layer l_2, the circuit shown in Fig. 5 results. This circuit is composed of 37 elementary operations and has depth 15.*

C. Optimizations

A^* allows 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 $\hat{\sigma}^i$ for layer l_i.

One way to optimize the proposed solution is to employ a look-ahead scheme which incorporates information of 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 estimation for the layer l_{i+1}. The overall heuristic that guides the search algorithm towards 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. $h(\sigma_j^i) = \sum_{g \in l_i \cup l_{i+1}} h(g, \sigma_j^i)$. 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 sub-optimal solutions in order to find cheaper mappings for the following layers – resulting in smaller overall circuit.

Example 7. *Consider again the quantum circuit shown in Fig. 4a 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 6), but a mapping $\hat{\sigma}^1$ that satisfies all CNOT-constraints with costs of 22 (as show in Fig. 6). The additional costs of 8 result since, after applying two SWAP gates (cf. Fig. 6), 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. 6; gates g_2 and g_3 can be applied concurrently). Consequently, the look-ahead scheme results in a cheaper mapping than the "pure" algorithm proposed in Section IV-B and yielding the circuit shown in Fig. 5.*[3]

Besides the look-ahead scheme, we can further improve the algorithm by not starting with a random mapping for layer l_0. Instead, we propose to use partial mappings σ_j^i and to start with an empty mapping σ_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 by 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 in a way, such that the costs for finding $\hat{\sigma}^i$ is as small as possible.

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

Example 8. *Optimizing the algorithm with a partial mapping that is initially empty results in the circuit already shown before in Fig. 4c. This circuit is composed of 23 elementary operations and has depth 10 (gates g_2 and g_3 can be applied concurrently).*

[3]Note that the graphical representation seems to be larger in Fig. 6. However, this is caused by the fact that the SWAP operations are not decomposed (cf. Fig 3) due to space limitations.

V. Experimental Evaluation

Taking all considerations and methods discussed above into account led to the development of a mapping scheme 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. This resulted in a comprehensive mapping scheme for the QX architectures which has been implemented in C++ on top of RevKit [23] and made publicly available at [17]. In this section, we compare the efficiency of the resulting scheme to the solution provided by IBM [16]. To this end, several functions taken from RevLib [24] as well as quantum algorithms written in the Scaffold language [18] (and pre-compiled by the ScaffoldCC compiler [22]) have been considered as benchmarks. All evaluations have been conducted on a 4.2 GHz machine with 32 GB RAM.

Table I lists the respectively obtained results. For each benchmark, we list the name, the number of logical qubits n, and the number of gates g of the quantum circuit before mapping it to the IBM QX3 architecture. In the remaining columns, we list the number of gates and the runtime t (in CPU seconds) for IBM's solution as well as for the solution proposed in this work.[4] Since IBM's solution randomly searches for mappings that satisfy all CNOT-constrains, we ran this algorithm several times and list only the obtained best results (minimum time and minimum gate count). The timeout for searching a single mapping was set to 1 hour.

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 hour in 6 out of 45 cases, the proposed algorithm determines a mapping for each circuit within 5 minutes or less – in most cases, only a fraction of a second is needed. Besides that, the approach is frequently magnitudes faster compared to IBM's solution.

Besides efficiency, the proposed method for mapping a quantum circuit to the IBM QX architectures also yields circuits with significantly fewer gates than the solution determined by IBM's solution. In fact, the solution proposed in Section IV results on average in circuits with 23% fewer gates compared to the minimum observed when runnings IBM's algorithm several times.

VI. Conclusions

In this paper, we proposed an approach 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 non-trivial 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 to efficiently map quantum circuits to real quantum hardware. This 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 one hour 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.

[4]We do not list circuit depth due to space limitations. However, we can report that similar improvements as for the number of gates have been observed in all evaluations. All results can be reproduced since the implementation is available at [17].

TABLE I Mapping to the IBM QX3 architecture

			IBM's solution		Proposed approach	
Name	n	g	g_{min}	t_{min}	g	t
hwb9	10	207 775	–	>3600.00	749 975	28.21
ising_model10	10	480	682	2.05	674	0.01
max46	10	27 126	125 157	1516.32	99 398	29.06
mini_alu	10	173	805	7.61	591	0.01
qft10	10	200	881	8.20	624	0.01
rd73	10	230	1 107	13.04	760	0.01
sqn	10	10 223	46 381	565.86	36 392	4.23
urf3	10	423 488	–	>3600.00	1 452 222	130.32
9symml	11	34 881	167 150	2049.95	131 271	14.24
dc1	11	1 914	8 687	100.16	6 814	0.06
life	11	22 445	108 137	1292.65	85 804	36.19
urf4	11	512 064	–	>3600.00	1 847 780	29.88
wim	11	986	4 401	51.13	3 401	0.01
z4	11	3 073	14 311	170.48	11 302	0.19
cm152a	12	1 221	5 371	62.58	4 352	0.02
cycle10	12	6 050	28 800	342.62	22 474	17.29
rd84	12	13 658	66 381	790.16	51 095	3.97
sqrt8	12	3 009	14 624	181.89	11 505	2.83
sym9	12	328	1 631	16.95	1 099	0.01
adr4	13	3 439	16 122	191.19	12 667	0.12
radd	13	3 213	15 433	177.20	11 678	0.27
rd53	13	275	1 422	15.37	1 133	0.02
root	13	17 159	83 999	1002.43	65 158	43.83
squar5	13	1 993	9 547	114.01	7 364	0.04
clip	14	33 827	167 488	2022.30	130 216	93.12
cm42a	14	1 776	7 841	91.08	6 274	0.02
cm85a	14	11 414	55 513	663.99	43 248	1.97
plus63mod8192	14	187 112	–	>3600.00	723 610	268.60
pm1	14	1 776	8 112	97.38	6 274	0.02
sao2	14	38 577	193 496	2302.52	148 558	258.01
sym6	14	270	1 396	14.80	932	0.01
dc2	15	9 462	46 479	566.36	35 153	46.74
ham15	15	8 763	40 988	492.87	31 503	0.75
misex1	15	4 813	22 738	273.83	18 369	0.06
rd84	15	343	1 895	17.71	1 252	0.04
square_root7	15	7 630	35 431	412.86	28 417	62.97
urf6	15	171 840	–	>3600.00	660 792	176.72
alu2	16	28 492	146 721	1749.62	113 995	66.07
cnt3-5	16	485	2 192	22.90	1 617	0.14
example2	16	28 492	147 149	1723.58	113 995	65.88
ground_state10	16	390 180	–	>3600.00	878 735	1.10
inc	16	10 619	50 326	619.13	38 853	0.77
ising_model16	16	786	1 246	5.65	1 170	2.52
mlp4	16	18 852	95 005	1140.70	73 664	29.55
qft16	16	512	2 539	26.15	1 635	23.29

References

[1] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.

[2] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[3] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Symposium on the Theory of Computing*, pages 212–219, 1996.

[4] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 439, pages 553–558. The Royal Society, 1992.

[5] IBM Q. https://www.research.ibm.com/ibm-q/.

[6] IBM QX backend information. https://github.com/QISKit/ibmqx-backend-information.

[7] Rachel Courtland. Google aims for quantum computing supremacy. *IEEE Spectrum June 2017*.

[8] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 32(6):818–830, 2013.

[9] D. M. Miller, R. Wille, and Z. Sasanian. Elementary quantum gate realizations for multiple-control Toffolli gates. In *International Symposium on Multi-Valued Logic*, pages 288–293, 2011.

[10] Ken Matsumoto and Kazuyuki Amano. Representation of quantum circuits with clifford and $\pi/8$ gates. *arXiv preprint arXiv:0806.3834*, 2008.

[11] Robert Wille, Aaron Lye, and Rolf Drechsler. Exact reordering of circuit lines for nearest neighbor quantum architectures. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 33(12):1818–1831, 2014.

[12] M. Saeedi, R. Wille, and R. Drechsler. Synthesis of quantum circuits for linear nearest neighbor architectures. *Quantum Information Processing*, 2010.

[13] Robert Wille, Oliver Keszocze, Marcel Walter, Patrick Rohrs, Anupam Chattopadhyay, and Rolf Drechsler. Look-ahead schemes for nearest neighbor optimization of 1d and 2d quantum circuits. In *Asia and South Pacific Design Automation Conference*, pages 292–297, 2016.

[14] A. Shafaei, M. Saeedi, and M. Pedram. Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In *Design Automation Conf.*, pages 41–46, 2013.

[15] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. Qubit placement to minimize communication overhead in 2d quantum architectures. In *19th Asia and South Pacific Design Automation Conference, ASP-DAC 2014, Singapore, January 20-23, 2014*, pages 495–500, 2014.

[16] QISKIT Python SDK. https://github.com/QISKit/qiskit-sdk-py.

[17] http://www.jku.at/iic/eda/ibm_qx_mapping.

[18] Ali J Abhari, Arvin Faruque, Mohammad J Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, and Fred Chong. Scaffold: Quantum programming language. Technical report, Princeton univ nj dept of computer science, 2012.

[19] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. In *Conference on Programming Language Design and Implementation*, pages 333–342, 2013.

[20] Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. Open quantum assembly language. *arXiv preprint arXiv:1707.03429*, 2017.

[21] A. Barenco, C. H. Bennett, R. Cleve, D.P. DiVinchenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *The American Physical Society*, 52:3457–3467, 1995.

[22] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. Scaffcc: a framework for compilation and analysis of quantum computing programs. In *Computing Frontiers Conference, CF'14, Cagliari, Italy - May 20 - 22, 2014*, pages 1:1–1:10, 2014.

[23] Mathias Soeken, Stefan Frehse, Robert Wille, and Rolf Drechsler. RevKit: A toolkit for reversible circuit design. In *Workshop on Reversible Computation*, pages 69–72, 2010. RevKit is available at http://www.revkit.org.

[24] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: an online resource for reversible functions and reversible circuits. In *International Symposium on Multi-Valued Logic*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.