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*
alwin.zulehner@jku.at alexandru.paler@jku.at robert.wille@jku.at

***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-qubit 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 hour 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 is publicly available at http://iic.jku.at/eda/research/ibm_qx_mapping.

I. Introduction

Quantum 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-qubit 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 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 [10]–[13]), 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 [14]–[20], they are not applicable since simplistic architectures with 1-dimensional or 2-dimensional layouts are assumed in that work which have significantly less restrictions. Even IBM's own solution, which is provided by means of the Python SDK *QISKit* [21] 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.

The above motivates a solution that is as efficient as circuit designers e.g. in the classical domain, take for granted today. In this work[1], we propose a corresponding methodology. To this end, a multi-step approach is introduced which utilizes a depth-based partitioning and A^* 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 hour 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 the paper.

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 can not 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.

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

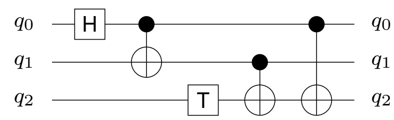


Fig. 1: Circuit diagram of a quantum circuit

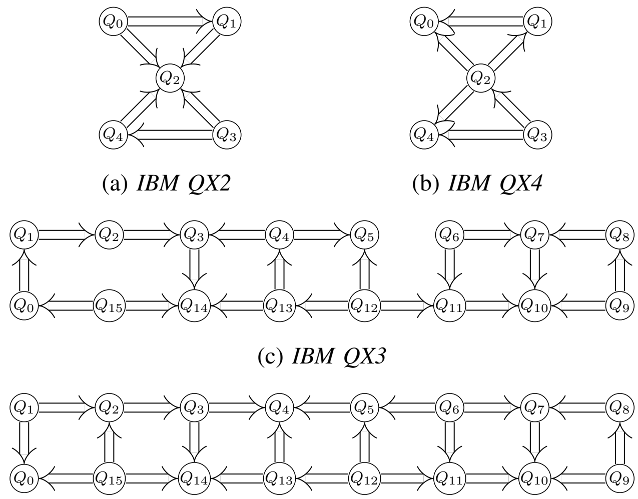
To this end, the qubits of a quantum circuit are manipulated by quantum operations represented by so-called quantum gates. These operations can either operate on a single qubit, or on multiple ones. For multi-qubit gates, we distinguish target qubits and control qubits. The value of the target qubits is modified in the case that the control qubits are set to basis state $|1\rangle$. The *Clifford+T* library [10], 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 [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 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 named *QISKit* [21] that allows a designer 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. In September 2017, IBM launched revised versions of their 5-qubit and 16-qubit backends named *IBM QX4* and *IBM QX5*, respectively.



(a) *IBM QX2* (b) *IBM QX4*

(c) *IBM QX3*

(d) *IBM QX5*

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

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 which qubit is supposed to work as target and which 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.

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 developed a normal form for single qubit operations [12], 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). 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.

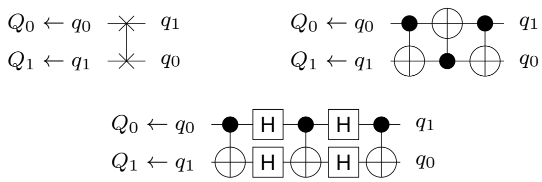


Fig. 3: Decomposition of SWAP gates

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-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* IBM QX3 *architecture shown in Fig. 2c. 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 us 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 [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-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, 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* (QAOA [33]) circuits for solving the MaxCut problem and target the architectures proposed by Rigetti (cf. [34]). As a consequence, none of the approaches discussed above is directly applicable for the problem considered here.

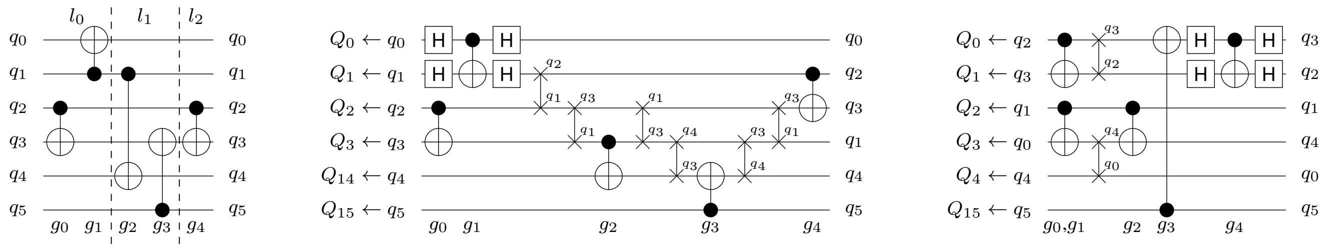
As a further alternative, IBM provides a solution within its SDK [21]. This algorithm randomly searches (guided by heuristics) for mappings of the qubits at a certain point of time. These mappings are then realized by adding SWAP gates to the circuit. But this random search is hardly feasible for many 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 hour (cf. 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 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 addressed 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.

[2]These approaches utilize satisfiability solvers, search algorithms, or dedicated data structures to tackle the underlying complexity.



(a) Original circuit

(b) Naive strategy

(c) Proposed strategy

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

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 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 us 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.

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 towards 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 $f(x) = g(x) + h(x)$ of a node x. The first part ($g(x)$) describes the cost of the current sub-solution (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 $h(x)$. 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^*

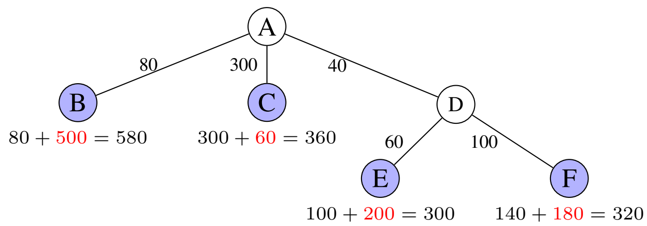


Fig. 5: A^* search algorithm

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 σ_j^i) as well as an admissible heuristic function $h(x)$ to estimate the distance of a state to a goal state (i.e. the mapping $\hat{\sigma}^i$) are required. 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.[3] 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 7. *Consider again the quantum circuit shown in Fig. 4a 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.* $\hat{\sigma}^0$*). 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 function 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[4]) $\hat{p}$ 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 operation might reduce the distance of the control and target qubits for more than one CNOT of layer l_i. Since this would prevent us from determining the optimal solution $\hat{\sigma}^i$, 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 7 (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. 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 beween two physical qubits iff a CNOT can be applied.

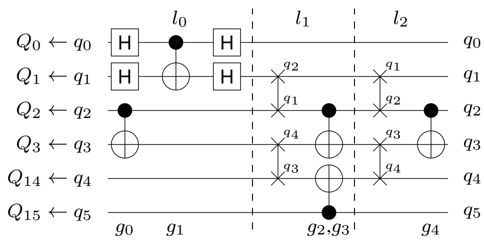


Fig. 6: Circuit resulting from locally optimal mappings

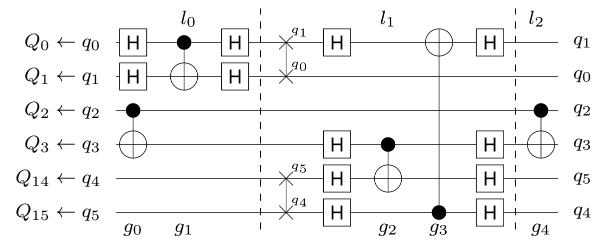


Fig. 7: Circuit generated when using the look-ahead scheme

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 $\hat{\sigma}^i$ 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 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 circuits.

Example 8. *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 7), but a mapping $\hat{\sigma}^1$ 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 (cf. 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 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 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 $\hat{\sigma}^i$ 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 non-beneficial 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. 4c. 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 pre-compiled

[5]Note that the graphical representation seems to be larger in Fig. 7. However, this is caused by the fact that the SWAP operations are not decomposed (cf. Fig 3) in order to maintain readability.

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 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 4 cores (2 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^* algorithm to determine locally optimal mappings for each layer of the circuit (denoted *Baseline* 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 one hour.

Table I clearly shows the improvements that can be gained by applying the optimizations discussed in Section IV-C. On average, the number of gates of the mapped circuit decreases by 16.1% when applying a look-ahead scheme as discussed in Section IV-C. For the depth of the circuit, we obtain similar improvements. Here, the number of layers reduces on average by 13.4%. However, using the look-ahead scheme causes the mapping algorithm to time out in nine cases (instead of five cases for *baseline*) – leading to a less scalable solution. If we additionally allow to evolve the initial mapping of logical qubits to physical qubits throughout the mapping process instead of starting with a random mapping, we can overcome this scalability issue while obtaining mappings of similar quality. In fact, the average improvement regarding the number of 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.

[6] We used all benchmarks that required at most 16 qubits since only these can be mapped to QX5.

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 work. Since IBM's mapping algorithm searches for mappings that satisfy all CNOT-constraints randomly (guided by certain heuristics), we conducted the mapping procedure 5 times for each benchmark and list the obtained minimum, the average (denoted by subscripts $_{min}$ and $_{avg}$, respectively), as well as the standard deviation σ for each of the listed metrics. The timeout for searching a single mapping was again set to one 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 one hour 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 runnings 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. Conclusions

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 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 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 one hour to determine a

TABLE I Effect of the Optimizations

Name	n	g	d	*Baseline* g	d	t	*Look-Ahead* g	d	t	*Fully-Optimized* g	d	t
adder_10	10	142	99	444	251	1.22	355	209	1.08	292	172	1.20
hwb9	10	207 775	116 199	743 973	403 199	1 662.82	653 249	374 954	1 437.33	655 220	375 105	1 422.33
ising_model_10	10	480	70	235	41	4.63	235	41	4.58	251	47	4.14
max46	10	27 126	14 257	–	–	TO	86 049	46 692	185.19	84 914	46 270	185.82
mini_alu	10	173	69	710	301	2.38	587	261	1.32	474	225	1.25
qft_10	10	200	63	685	227	1.23	445	135	1.54	447	170	1.25
rd73	10	230	92	952	405	1.83	916	374	1.57	656	301	1.52
sqn	10	10 223	5 458	37 781	19 461	80.15	32 099	17 785	72.25	32 095	17 801	68.97
sym9	10	21 504	12 087	78 388	43 269	172.95	67 290	38 982	147.99	66 637	38 849	145.37
sys6-v0	10	215	75	962	383	1.96	794	301	1.50	613	250	1.36
urf3	10	125 362	70 702	517 104	271 754	1 045.85	439 268	239 099	888.77	440 509	239 702	873.84
9symml	11	34 881	19 235	133 813	70 088	296.80	114 179	63 659	255.02	116 508	64 279	254.25
dc1	11	1 914	1 038	8 310	4 277	16.22	6 024	3 359	13.07	5 946	3 378	12.38
life	11	22 445	12 511	86 075	45 499	358.56	73 020	41 137	161.90	74 632	41 767	166.95
shor_11	11	49 295	30 520	125 825	70 115	325.13	109 574	60 721	317.81	106 322	58 943	322.78
sym9	11	34 881	19 235	133 813	70 088	292.40	114 179	63 659	249.49	116 508	64 279	251.42
urf4	11	512 064	264 330	1 926 128	980 191	4 257.19	1 653 689	888 594	3 481.55	1 650 845	878 249	3 534.79
wim	11	986	514	3 632	1 914	7.60	3 176	1 712	6.54	2 985	1 711	6.30
z4	11	3 073	1 644	12 041	6 332	24.91	10 002	5 486	20.71	9 717	5 335	20.92
adder_12	12	177	123	631	336	1.76	483	249	1.64	372	226	1.32
cm152a	12	1 221	684	4 254	2 226	9.13	4 039	2 271	8.23	3 738	2 155	8.02
cycle10_2	12	6 050	3 386	23 991	12 405	49.62	19 513	10 950	45.82	19 857	11 141	42.26
rd84	12	13 658	7 261	52 508	26 668	157.72	45 509	24 421	107.69	45 497	24 473	99.89
sqrt8	12	3 009	1 659	11 921	6 224	26.64	10 166	5 642	21.35	9 744	5 501	19.66
sym10	12	64 283	35 572	251 731	130 657	535.84	214 881	118 780	500.05	215 569	118 753	501.02
sym9	12	328	127	1 436	608	2.54	1 240	532	2.27	955	425	2.08
adr4	13	3 439	1 839	13 475	6 829	29.53	11 245	6 120	23.84	11 301	6 205	23.17
dist	13	38 046	19 694	147 115	72 929	323.20	125 342	66 590	334.67	125 867	66 318	291.90
gse_10	13	390 180	245 614	863 511	533 279	2 441.38	576 399	401 121	2 263.71	520 010	376 695	2 237.10
ising_model_13	13	633	71	313	41	6.08	313	41	6.09	329	47	5.11
plus63mod4096	13	128 744	72 246	529 896	270 734	1 203.45	434 900	242 815	1 006.16	439 981	243 861	1 086.48
radd	13	3 213	1 781	11 790	6 387	25.35	10 868	6 088	23.67	10 441	5 872	22.00
rd53	13	275	124	1 367	619	2.53	1 044	457	1.97	942	469	1.93
root	13	17 159	8 835	67 941	32 854	327.20	56 654	29 846	120.01	57 874	30 068	120.82
shor_13	13	98 109	59 350	259 511	140 923	656.43	229 752	121 093	783.74	224 556	118 536	640.55
squar5	13	1 993	1 049	7 948	4 069	16.35	6 453	3 470	13.29	6 267	3 448	12.96
410184	14	211	104	914	441	1.82	708	337	1.42	758	366	1.48
adder_14	14	212	147	–	–	TO	–	–	TO	437	268	1.47
clip	14	33 827	17 879	135 455	67 312	322.36	–	–	TO	114 336	60 882	327.55
cm42a	14	1 776	940	6 473	3 394	13.93	5 572	3 076	11.16	5 431	3 013	11.95
cm85a	14	11 414	6 374	46 300	23 662	185.98	37 927	21 215	464.90	37 746	21 189	242.80
plus127mod8192	14	330 777	185 853	–	–	TO	–	–	TO	1 132 251	626 451	2 481.95
plus63mod8192	14	187 112	105 142	773 514	395 379	1 628.19	637 137	355 040	1 364.63	640 204	354 076	1 443.33
pm1	14	1 776	940	6 473	3 394	13.62	5 572	3 076	11.14	5 431	3 013	11.10
sao2	14	38 577	19 563	155 351	74 524	330.62	–	–	TO	131 002	66 975	283.90
sym6	14	270	135	1 101	547	2.33	1 136	526	2.05	852	456	1.84
co14	15	17 936	8 570	80 399	34 658	331.89	62 348	29 831	176.55	63 826	30 366	133.71
dc2	15	9 462	5 242	36 968	19 306	83.96	31 722	17 559	95.81	30 680	17 269	72.53
ham15	15	8 763	4 819	32 175	17 379	79.80	27 861	15 668	61.70	28 310	15 891	68.75
misex1	15	4 813	2 676	17 833	9 621	38.63	15 260	8 810	33.18	15 185	8 729	33.11
rd84	15	343	110	1 593	553	3.30	1 337	441	2.81	971	353	2.23
square_root	15	7 630	3 847	–	–	TO	–	–	TO	25 212	13 205	55.35
urf6	15	171 840	93 645	684 701	353 581	1 456.37	–	–	TO	580 295	313 011	1 436.16
adder_16	16	247	171	–	–	TO	–	–	TO	515	319	1.72
alu2	16	28 492	15 176	118 919	58 105	244.83	–	–	TO	98 166	51 817	454.93
cnt3-5	16	485	209	1 957	887	3.89	1 488	725	2.98	1 376	669	3.00
example2	16	28 492	15 176	118 919	58 105	246.00	–	–	TO	98 166	51 817	449.08
inc	16	10 619	5 863	41 042	21 614	86.91	34 742	19 431	74.13	34 375	19 176	72.85
ising_model_16	16	786	71	391	41	6.88	391	41	6.86	426	48	6.47
qft_16	16	512	105	2 193	589	69.04	1 299	281	8.62	1 341	404	16.43

n: the number of qubits g: the number of quantum gates (elementary operations) d: depth of the quantum circuits t: runtime of the algorithm
Baseline: the approach described in Sec. IV-B *Look-Ahead*: the approach described in Sec. IV-B enriched with the look-ahead scheme discussed in Sec. IV-C
Fully-Optimized: the approach described in Sec. IV-B enriched with all optimizations discussed in Sec. IV-C

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.

Acknowledgments

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

References

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

[2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997.

[3] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Symposium on the Theory of Computing*, pp. 212–219, 1996.

[4] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," vol. 439, pp. 553–558, The Royal Society, 1992.

[5] "IBM Q." https://www.research.ibm.com/ibm-q/. Accessed: 2017-09-15.

[6] "IBM QX backend information." https://github.com/QISKit/ibmqx-backend-information. Accessed: 2017-09-15.

[7] C. Neill, P. Roushan, K. Kechedzhi, S. Boixo, S. Isakov, V. Smelyanskiy, R. Barends, B. Burkett, Y. Chen, Z. Chen, *et al.*, "A blueprint for

TABLE II Mapping to the *IBM QX5* architecture

Name	n	g	d	IBM's solution g_{min}	g_{avg}	σ_g	d_{min}	d_{avg}	σ_d	t_{min}	t_{avg}	σ_t	Proposed approach g	d	t
adder_10	10	142	99	382	421.00	29.37	203	223.80	16.85	5.75	6.71	0.72	292	172	1.20
hwb9	10	207 775	116 199	–	–	–	–	–	–	TO	–	–	655 220	375 105	1 422.33
ising_model_10	10	480	70	347	401.40	79.37	73	89.60	21.46	6.23	7.07	0.99	251	47	4.14
max46	10	27 126	14 257	105 651	106 357.80	453.99	53 397	53 664.20	138.67	1 652.31	1 672.40	16.15	84 914	46 270	185.82
mini_alu	10	173	69	707	736.00	19.37	290	303.40	10.78	9.82	10.32	0.42	474	225	1.25
qft_10	10	200	63	670	755.00	98.89	210	241.40	30.16	9.49	10.90	2.35	447	170	1.25
rd73	10	230	92	930	1 035.00	70.69	393	424.20	26.18	12.74	14.88	1.46	656	301	1.52
sqn	10	10 223	5 458	39 175	39 563.60	301.56	20 329	20 415.40	75.50	627.32	633.67	5.51	32 095	17 801	68.97
sym9	10	21 504	12 087	80 867	81 893.00	638.00	43 707	44 128.40	342.23	1 314.22	1 321.35	5.48	66 637	38 849	145.37
sys6-v0	10	215	75	853	945.80	51.52	329	346.20	12.22	11.29	12.80	1.17	613	250	1.36
urf3	10	125 362	70 702	–	–	–	–	–	–	TO	–	–	440 509	239 702	873.84
9symml	11	34 881	19 235	143 042	144 299.20	798.04	73 363	73 821.00	496.98	2 233.67	2 257.31	18.51	116 508	64 279	254.25
dc1	11	1 914	1 038	7 283	7 356.40	73.48	3 859	3 890.60	28.10	113.8	116.75	2.83	5 946	3 378	12.38
life	11	22 445	12 511	91 724	92 470.20	599.36	47 471	47 860.80	382.84	1 446.67	1 454.98	5.38	74 632	41 767	166.95
shor_11	11	49 295	30 520	124 160	125 245.00	775.01	67 962	68 367.40	317.02	2 149.13	2 160.12	6.96	106 322	58 943	322.78
sym9	11	34 881	19 235	142 431	143 685.00	751.22	72 959	73 595.40	450.98	2 237.2	2 261.74	13.68	116 508	64 279	251.42
urf4	11	512 064	264 330	–	–	–	–	–	–	TO	–	–	1 650 845	878 249	3 534.79
wim	11	986	514	3 834	3 897.20	46.70	1 947	1 988.60	26.04	59.01	59.88	1.04	2 985	1 711	6.30
z4	11	3 073	1 644	11 905	12 148.20	134.51	6 024	6 190.80	87.10	188.53	192.41	3.74	9 717	5 335	20.92
adder_12	12	177	123	579	636.00	68.67	279	307.00	26.10	8.81	10.28	1.14	372	226	1.32
cm152a	12	1 221	684	4 761	4 928.20	100.23	2 501	2 581.20	64.31	74.61	76.53	1.18	3 738	2 155	8.02
cycle10_2	12	6 050	3 386	25 362	25 666.60	301.28	13 125	13 224.00	80.01	392.12	394.36	2.43	19 857	11 141	42.26
rd84	12	13 658	7 261	56 134	56 865.60	425.85	28 172	28 393.40	126.79	860.9	874.21	10.25	45 497	24 473	99.89
sqrt8	12	3 009	1 659	12 541	12 678.20	127.15	6 398	6 457.60	52.37	194.8	197.02	1.36	9 744	5 501	19.66
sym10	12	64 283	35 572	–	–	–	–	–	–	TO	–	–	215 569	118 753	501.02
sym9	12	328	127	1 411	1 512.20	83.10	582	598.60	18.13	20.09	21.59	1.98	955	425	2.08
adr4	13	3 439	1 839	13 638	13 958.80	172.30	6 991	7 075.40	57.82	210.34	217.16	3.95	11 301	6 205	23.17
dist	13	38 046	19 694	158 516	159 655.00	1 268.06	77 027	77 739.40	631.57	2 412.78	2 445.98	18.58	125 867	66 318	291.90
gse_10	13	390 180	245 614	–	–	–	–	–	–	TO	–	–	520 010	376 695	2 237.10
ising_model_13	13	633	71	439	573.80	101.73	82	138.20	44.62	7.87	9.53	1.37	329	47	5.11
plus63mod4096	13	128 744	72 246	–	–	–	–	–	–	TO	–	–	439 981	243 861	1 086.48
radd	13	3 213	1 781	12 674	13 263.00	331.68	6 716	6 907.00	124.21	206.15	211.40	4.83	10 441	5 872	22.00
rd53	13	275	124	1 223	1 295.20	53.56	518	543.80	20.07	16.65	17.89	0.75	942	469	1.93
root	13	17 159	8 835	71 721	72 252.80	304.63	34 798	35 009.00	205.73	1 094.63	1 099.08	5.15	57 874	30 068	120.82
shor_13	13	98 109	59 350	–	–	–	–	–	–	TO	–	–	224 556	118 536	640.55
squar5	13	1 993	1 049	8 111	8 300.00	201.58	4 073	4 132.20	59.48	124.09	125.77	1.47	6 267	3 448	12.96
410184	14	211	104	864	928.20	40.30	393	408.80	17.65	13.36	13.91	0.43	758	366	1.48
adder_14	14	212	147	659	745.20	114.49	332	373.20	51.39	9.76	11.55	1.49	437	268	1.47
clip	14	33 827	17 879	144 737	145 459.00	496.53	70 732	71 177.00	291.36	2 197.11	2 212.02	12.55	114 336	60 882	327.55
cm42a	14	1 776	940	6 623	6 830.20	170.16	3 480	3 538.20	52.18	104.23	109.00	2.47	5 431	3 013	11.95
cm85a	14	11 414	6 374	47 908	48 885.40	673.07	24 798	25 101.40	253.09	742.46	757.59	10.50	37 746	21 189	242.80
plus127mod8192	14	330 777	185 853	–	–	–	–	–	–	TO	–	–	1 132 251	626 451	2 481.95
plus63mod8192	14	187 112	105 142	–	–	–	–	–	–	TO	–	–	640 204	354 076	1 443.33
pm1	14	1 776	940	6 488	6 809.60	190.40	3 444	3 525.80	53.01	104.21	106.31	1.69	5 431	3 013	11.10
sao2	14	38 577	19 563	163 679	164 561.40	803.87	77 525	77 771.40	228.58	2 495.54	2 509.16	15.15	131 002	66 975	283.90
sym6	14	270	135	1 092	1 246.60	96.11	514	568.00	35.64	16.05	18.61	1.81	852	456	1.84
co14	15	17 936	8 570	83 301	83 649.40	234.76	35 926	36 046.20	98.35	1 177.71	1 192.44	8.36	63 826	30 366	133.71
dc2	15	9 462	5 242	38 807	39 694.60	602.91	20 155	20 359.20	148.41	601.92	625.13	16.72	30 680	17 269	72.53
ham15	15	8 763	4 819	35 150	35 402.20	273.69	18 293	18 453.20	116.27	546.05	552.47	5.29	28 310	15 891	68.75
misex1	15	4 813	2 676	19 090	19 316.80	169.21	10 172	10 235.80	60.85	299.9	304.52	3.18	15 185	8 729	33.11
rd84	15	343	110	1 579	1 807.60	159.89	529	599.20	46.33	19.5	23.18	2.47	971	353	2.23
square_root	15	7 630	3 847	30 349	30 760.80	421.20	14 828	15 029.40	213.97	461.14	468.59	5.71	25 212	13 205	55.35
urf6	15	171 840	93 645	–	–	–	–	–	–	TO	–	–	580 295	313 011	1 436.16
adder_16	16	247	171	968	1 039.40	51.79	437	473.60	29.18	13.88	14.94	1.01	515	319	1.72
alu2	16	28 492	15 176	125 601	126 758.20	906.96	60 839	61 383.20	386.58	1 905.14	1 922.29	15.33	98 166	51 817	454.93
cnt3-5	16	485	209	1 899	2 023.00	102.11	825	863.40	34.92	27.08	29.71	2.78	1 376	669	3.00
example2	16	28 492	15 176	125 022	127 074.60	1 176.85	60 543	61 347.20	440.11	1 900.17	1 920.29	13.42	98 166	51 817	449.08
inc	16	10 619	5 863	43 097	43 561.20	442.97	22 413	22 577.00	127.65	672.39	679.82	6.28	34 375	19 176	72.85
ising_model_16	16	786	71	785	858.00	51.98	149	172.40	16.84	12.44	13.64	0.79	426	48	6.47
qft_16	16	512	105	2 056	2 219.00	128.72	521	542.60	16.27	25.6	27.10	1.76	1 341	404	16.43

n: the number of qubits g: the number of quantum gates (elementary operations) d: depth of the quantum circuits t: runtime of the algorithm
For IBM's solution, we list the obtained minimum, the average, and the standard deviation of 5 runs (denoted by $_{min}$, $_{avg}$, and σ, respectively).

demonstrating quantum supremacy with superconducting qubits," *arXiv preprint arXiv:1709.06678*, 2017.

[8] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, J. M. Martinis, and H. Neven, "Characterizing quantum supremacy in near-term devices," *arXiv preprint arXiv:1608.00263*, 2016.

[9] R. Courtland, "Google aims for quantum computing supremacy [news]," *IEEE Spectrum*, vol. 54, no. 6, pp. 9–10, 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. on CAD of Integrated Circuits and Systems*, vol. 32, no. 6, pp. 818–830, 2013.

[11] D. M. Miller, R. Wille, and Z. Sasanian, "Elementary quantum gate realizations for multiple-control Toffolli gates," in *Int'l Symp. on Multi-Valued Logic*, pp. 288–293, 2011.

[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 *Asia and South Pacific Design Automation Conf.*, pp. 85–92, 2013.

[14] R. Wille, A. Lye, and R. Drechsler, "Exact reordering of circuit lines for nearest neighbor quantum architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1818–1831, 2014.

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

[16] R. Wille, O. Keszocze, M. Walter, P. Rohrs, A. Chattopadhyay, and R. Drechsler, "Look-ahead schemes for nearest neighbor optimization of 1d and 2d quantum circuits," in *Asia and South Pacific Design

Automation Conf., pp. 292–297, 2016.

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

[18] A. Shafaei, M. Saeedi, and M. Pedram, "Qubit placement to minimize communication overhead in 2d quantum architectures," in *Asia and South Pacific Design Automation Conf.*, pp. 495–500, 2014.

[19] R. Wille, N. Quetschlich, Y. Inoue, N. Yasuda, and S. Minato, "Using \pi dds for nearest neighbor optimization of quantum circuits," in *Int'l Conf. of Reversible Computation*, pp. 181–196, 2016.

[20] A. Zulehner, S. Gasser, and R. Wille, "Exact global reordering for nearest neighbor quantum circuits using A^*," in *International Conference on Reversible Computation*, pp. 185–201, Springer, 2017.

[21] "QISKIT Python SDK." https://github.com/QISKit/qiskit-sdk-py. Accessed: 2017-09-15.

[22] A. Zulehner, A. Paler, and R. Wille, "Efficient mapping of quantum circuits to the IBM QX architectures," in *Design, Automation and Test in Europe*, pp. 1135–1138, 2018.

[23] A. J. Abhari, A. Faruque, M. J. Dousti, L. Svec, O. Catu, A. Chakrabati, C.-F. Chiang, S. Vanderwilt, J. Black, and F. Chong, "Scaffold: Quantum programming language," tech. rep., Princeton univ nj dept of computer science, 2012.

[24] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, "Quipper: a scalable quantum programming language," in *Conference on Programming Language Design and Implementation*, pp. 333–342, 2013.

[25] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," *arXiv preprint arXiv:1707.03429*, 2017.

[26] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Design Automation Conf.*, pp. 318–323, 2003.

[27] A. Zulehner and R. Wille, "One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 37, no. 5, pp. 996–1008, 2018.

[28] P. Niemann, R. Wille, and R. Drechsler, "Improved synthesis of clifford+t quantum functionality," in *Design, Automation and Test in Europe*, pp. 597–600, 2018.

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

[30] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi, "Scaffcc: a framework for compilation and analysis of quantum computing programs," in *Computing Frontiers Conference, CF'14, Cagliari, Italy - May 20 - 22, 2014*, pp. 1:1–1:10, 2014.

[31] M. Soeken, S. Frehse, R. Wille, and R. Drechsler, "RevKit: A toolkit for reversible circuit design," in *Workshop on Reversible Computation*, pp. 69–72, 2010. RevKit is available at http://www.revkit.org.

[32] D. Venturelli, M. Do, E. G. Rieffel, and J. Frank, "Compiling quantum circuits to realistic hardware architectures using temporal planners," *CoRR*, vol. abs/1705.08927, 2017.

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

[34] E. A. Sete, W. J. Zeng, and C. T. Rigetti, "A functional architecture for scalable quantum computing," in *IEEE International Conference on Rebooting Computing, ICRC 2016, San Diego, CA, USA, October 17-19, 2016*, pp. 1–6, 2016.

[35] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

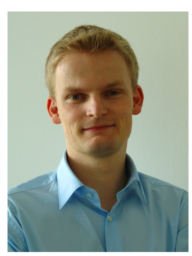
[36] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: an online resource for reversible functions and reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, pp. 220–225, 2008. RevLib is available at http://www.revlib.org.

[37] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," in *Workshop on Quantum Information Processing*, 2005.

[38] S. Beauregard, "Circuit for Shor's algorithm using 2n+3 qubits," *Quantum Information & Computation*, vol. 3, no. 2, pp. 175–185, 2003.

Alwin Zulehner Alwin Zulehner (S'17) received his BSc and MSc degree in computer science from the Johannes Kepler University Linz, Austria in 2012 and 2015, respectively. He is currently a Ph.D. student at the Institute for Integrated Circuits at the Johannes Kepler University Linz, Austria. His 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 (TCAD), Asia and South Pacific Design Automation Conference (ASP-DAC), Design, Automation and Test in Europe (DATE) and International Conference on Reversible Computation.

Alexandru Paler Alexandru Paler is a postdoc at the Johannes Kepler University Linz. His research focuses on algorithms for compiling and assembling large scale error corrected quantum circuits.



Robert Wille 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 of Bremen, Bremen, Germany, and a Visiting Professor with the University of Potsdam, Potsdam, Germany, and Technical University Dresden, Dresden, Germany. Since 2015, he is a Full Professor with Johannes Kepler University Linz, Linz, Austria. His current research interests include the design of circuits and systems for both conventional and emerging technologies. In these areas, he has published over 200 papers in journals and conferences. Dr. Wille has served in Editorial Boards and Program Committees of numerous journals/conferences, such as the IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD), Asia and South Pacific Design Automation Conference (ASP-DAC), Design Automation Conference (DAC), Design, Automation and Test in Europe (DATE) and International Conference on Computer Aided Design (ICCAD).