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 with respect to runtime as well as resulting costs.**

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 ones [8, 10, 14, 18]. 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. [8, 10, 18]), physical realizations have not *publicly* been 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 [1]. 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 [2] 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 [6]) 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.

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. This affects 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, the number of SWAP gates 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 [5, 12, 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 [15, 16, 17, 21, 22], 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 [3] 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 – significantly outperforming IBM's own mapping solution. 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 http://www.jku.at/iic/eda/ibm_qx_mapping.

II. Background

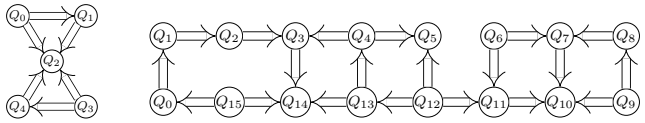
A. Quantum Circuits

While classical computations and circuits use bits as information unit, quantum circuits perform their computations on qubits [14] that 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.

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 (e.g. a Hadamard gate), or on multiple ones (e.g. a CNOT gate or a SWAP gate). 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 $|1\rangle$.

To describe quantum circuits, high level quantum languages (e.g. Scaffold [4] or Quipper [9]), quantum assembly languages (e.g. OpenQASM 2.0 developed by IBM [7]), or circuit diagrams are employed. In the following, we use the latter to describe quantum circuits. Here, qubits are represented by horizontal lines, which are passed through quantum gates. In contrast to classical circuits, this however does not describe

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



(a) *IBM QX2* (b) *IBM QX3*

Fig. 1: Coupling map of the *IBM QX* architectures [2]

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. 3 shows several quantum circuit diagrams. A Hadamard gates is represented by a box labeled with H, while the control and target qubit of the CNOT gate are represented by $\bullet$ and $\oplus$, respectively. Each of the two target qubits of a swap gate is represented by $\times$.*

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* [1]. IBM provides a Python SDK [3] 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 [2].

The IBM QX architectures support the elementary single qubit operation $U(\theta, \phi, \lambda) = R_z(\phi)R_y(\theta)R_z(\lambda)$ that is composed by two rotations around the z-axis and one rotation 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 (e.g. H; cf. Section II-A) can be realized.

However, there are significant restrictions which have to be satisfied when running quantum algorithms on these architectures. Besides decomposing all non-elementary quantum operations (e.g. Toffoli gate or SWAP gate) to the elementary operations $U(\theta, \phi, \lambda)$ and $CNOT$, further constraints have to be satisfied. These restrictions apply to CNOT gates and are given by the so-called *coupling-map* illustrated in Fig. 1, 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 to consider two major issues:

First, 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. This has already intensely been considered in the past (e.g. by methods which allow to describe quantum functionality in terms of elementary operations or building blocks derived from that [7] as well as decomposition and synthesis methods as proposed in [5, 12, 13] and provided by [9, 11, 19]).

Second, 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 such that all CNOT-constraints are satisfied. Since there usually does not exist a mapping solution that satisfies all CNOT-constrains throughout the whole circuit, the mapping might change during the execution of a quantum circuit. 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.[1] In other words, these gates can be used to "move" around the logical qubits on the actual hardware until the CNOT-constraints are satisfied.

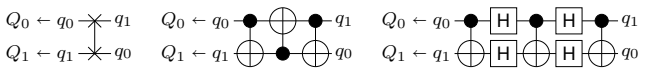


Fig. 2: Decomposition a SWAP operation

Example 2. *Consider the quantum circuit composed of 5 CNOT gates shown in Fig. 3a 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. 1b. 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. 3b. 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. 3b). 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 gates eventually results in the circuit shown in Fig. 3b.*

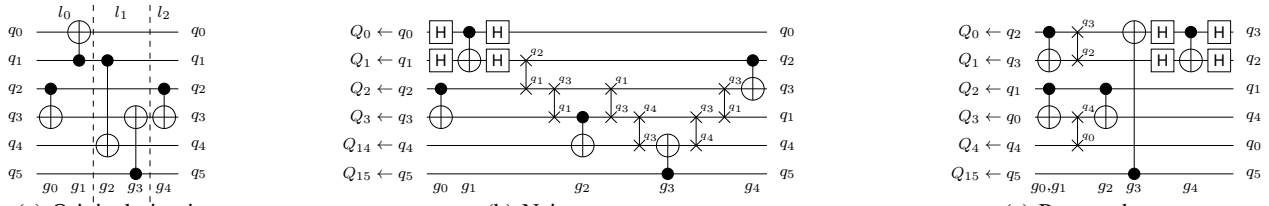
However, inserting the additional gates to satisfy the CNOT-constraints drastically increases the number of gates, which which significantly affects the reliability of the quantum circuit since each gate has a certain However, inserting the additional gates to satisfy the CNOT-constraints drastically increases the number of gates, which significantly 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. 2), particularly their number shall be kept as small as possible.

Example 3. *Consider again the given quantum circuit from Fig. 3a as well as its mapping derived in Example 2 and shown in Fig. 3b. 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. 3c (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 [15, 16, 17, 21, 22]. Also here, SWAP gates have been applied to move qubits together in order to satisfy a physical constraint. However, these works are not applicable here since they consider more simplistic architectures where a two-qubit gate can be applied to any adjacent qubits. The CNOT-constraints to be satisfied for the IBM QX architectures are much stricter with respect to that and also what physical qubit may act as control and as target qubit.

As a further alternative, also IBM itself provides a solution within its SDK [3] that randomly searches for mappings of the qubits at a certain point of time. However, 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 (also evaluated in Section V).

[1] Fig. 2 shows the IBM QX-compatible realization of a SWAP operation. The Hadamard operations $H = U(\pi/2, 0, \pi)$ are required to flip the direction of the middle CNOT gate in order to satisfy the CNOT-constraints.



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

Fig. 3: Mapping of a quantum circuit to the *IBM QX3* architecture

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) 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.

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. 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.[2] 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.

Example 4. *Consider again the quantum circuit shown in Fig. 3a. The dashed lines indicate a partition of the circuit into the layers* $l_0 = \{g_0, g_1\}$, $l_1 = \{g_2, g_3\}$, *and* $l_2 = \{g_4\}$.

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 a Layer

For a layer l_i, we have to determine a mapping $\sigma^i : \{q_0, q_1, \ldots q_{n-1}\} \rightarrow \{Q_0, Q_1, \ldots Q_{m-1}\}$ describing to which physical qubit a logical qubit is mapped such that all CNOT-constraints are satisfied. Furthermore, this mapping shall have the minimum distance from the mapping σ^{i-1} of the previous layer, i.e. the mapping that requires the permutation layer π_i with the fewest elementary operations. In the worst case, this 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.

[2]Note that the depth of a circuit is equal to the number of layers of a circuit.

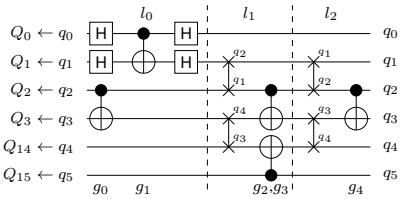


Fig. 4: Circuit resulting from locally optimal mappings

A^* is a family of search algorithm that can be used (guided by an appropriate heuristic) to determine σ^i. We start with the mapping of the previous layer σ^{i-1} and generate subsequent mappings by inserting SWAP operations – eventually resulting in a mapping that satisfies all CNOT-constraints. By having a heuristic that does not overestimate the real distance to such a mapping, A^* yields an optimal solution. The distance can be estimated with the coupling map of the architecture (cf. Fig. 1), since the shortest path (following the arrows in the coupling map) between the target and the control qubits of each CNOT gate serves as lower bound for the distance to a mapping that satisfies all CNOT-constraints.

Example 5. *Consider again the quantum circuit shown in Fig. 3a 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.* σ^0*). This mapping does not satisfy the CNOT-constraints for the gates in* l_1. *Following the A* approach sketched above, we eventually determine a mapping* σ^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. 4). Applying the algorithm also for mapping layer* l_2, *the circuit shown in Fig. 4 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, these local optima may not lead to a (or close to a) globally optimal solution.

To overcome this issue, we propose to employ a look-ahead scheme which incorporates information of the following layer to the cost function of the A^* search algorithm. To this end, we only have to change the heuristics to estimate the costs for π_i (i.e. for reaching a mapping that satisfies all CNOT-constraints from the current one). To incorporate the look-ahead scheme, we additionally add a term that estimates the cost for π_{i+1} to these costs. This way, 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 determine cheaper mappings for the following layers – resulting in smaller overall circuits.

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 start with an empty mapping σ^0 (i.e. none of the logical qubits is mapped to a physical one). Then, before we start searching a mapping for layer l_i, we check whether the qubits that occur in the CNOTs $g \in l_i$ have already been mapped to a physical qubit. If not, we can 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 σ^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 6. *Optimizing the algorithm with a look-ahead scheme and a partial mapping that is initially empty results in the circuit already shown before in Fig. 3c. 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

In this section, we compare the efficiency of the proposed mapping scheme (publicly available at http://www.jku.at/iic/eda/ibm_qx_mapping) to the solution provided by IBM [3]. Several circuits taken from RevLib [20] as well as quantum algorithms written in the Scaffold language [4] (and pre-compiled by the ScaffoldCC compiler [11]) have been considered as benchmarks.

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. 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.

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 several 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 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 (on average by 23%).

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
max46	10	27 126	125 157	1516.32	99 398	29.06
qft10	10	200	881	8.20	624	0.01
rd73	10	230	1 107	13.04	760	0.01
urf3	10	423 488	–	>3600.00	1 452 222	130.32
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
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
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
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

VI. Conclusions

In this paper, we have 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 – clearly outperforming IBM's solution regarding the runtime as well as regarding the number of additionally required gates.

References

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

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

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

[4] 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. Technical report, Princeton univ nj dept of computer science, 2012.

[5] 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*, 32(6):818–830, 2013.

[6] R. Courtland. Google aims for quantum computing supremacy. *IEEE Spectrum June 2017*.

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

[8] D. Deutsch and R. 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.

[9] 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*, pages 333–342, 2013.

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

[11] 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*, pages 1:1–1:10, 2014.

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

[13] 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.

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

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

[16] 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.

[17] A. Shafaei, M. Saeedi, and M. 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.

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

[19] M. Soeken, S. Frehse, R. Wille, and R. 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.

[20] 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.

[21] 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 Conference*, pages 292–297, 2016.

[22] 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*, 33(12):1818–1831, 2014.