Look-ahead Schemes for Nearest Neighbor Optimization of 1D and 2D Quantum Circuits

Robert Wille[1,2] Oliver Keszocze[2,3] Marcel Walter[3] Patrick Rohrs[3] Anupam Chattopadhyay[4] Rolf Drechsler[2,3]
[1]Institute for Integrated Circuits, Johannes Kepler University Linz, A-4040 Linz, Austria
[2]Cyber Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[3]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[4]Nanyang Technological University, Singapore
robert.wille@jku.at {keszocze,mwalter,proehrs,drechsle}@informatik.uni-bremen.de anupam@ntu.edu.sg

***Abstract*—Ensuring nearest neighbor compliance of quantum circuits by inserting SWAP gates has heavily been considered in the past. Here, quantum gates are considered which work on non-adjacent qubits. SWAP gates are applied in order to "move" these qubits onto adjacent positions. However, a decision how exactly the SWAPs are "moved" has mainly been made without considering the effect a "movement" of qubits may have on the remaining circuit. In this work, we propose a methodology for nearest neighbor optimization which addresses this problem by means of a look-ahead scheme. To this end, two representative implementations are presented and discussed in detail. Experimental evaluations show that, in the best case, reductions in the number of SWAP gates of 56% (compared to the state-of-the-art methods) can be achieved following the proposed methodology.**

I. Introduction

Fascinating progress is regularly being reported in the domain of quantum computation. In particular, recent accomplishments such as the realization of a highly coherent qubit [1] (the quantum-equivalent to a conventional bit) or the implementation of a scalable and fault-tolerant qubit architecture [2] received interest. Nevertheless, many obstacles prevent the realization of a practical and affordable quantum computer.

Among those, fault-tolerance stands out to be a prominent one. To address that, error correcting codes, such as Steane or surface code is widely being accepted [2]. But these, however, depend on so called nearest-neighbor interactions, i.e. allow for operations on adjacent qubits only. Furthermore, also recent progresses in two-qubit gate operations [3], quantum simulators [4], quantum error correction [5], or circuit implementations [6], [7] unanimously assume nearest-neighbor interactions. While there is an effort to enable interaction between remote qubits [8], it assumes the underlying architecture to be nearest neighbor-compliant in order to leverage on its fault-tolerance ability. Hence, how to ensure nearest neighbor interaction on qubits is an important issue in the design of quantum circuits.

In order to ease these developments, researchers recently developed various EDA methods for nearest neighbor optimization. To this end, several schemes have been proposed which insert so-called SWAP gates in order to "move" non-adjacent qubits onto adjacent positions. In [9], it has been shown that the underlying problem is computationally very expensive (in fact, exact approaches are applicable to rather small circuits only). As a consequence, researchers focused on heuristics in order to efficiently obtain a nearest neighbor, but also compact circuit (a review on related work is later provided in Section III-A). However, all these works inherit the drawback that the decision how to explicitly place the SWAP gates has mainly been made without considering the effect the resulting "movement" of qubits may have to the remaining circuit.

This work introduces a heuristic which aims for tackling that problem by additionally incorporating a look-ahead methodology. The general idea is to determine all possible options on how to "move" qubits together in order to make them adjacent. Then, the one is chosen which has the best impact to the following gates. In order to evaluate the proposed methodology, two representative implementations are presented and discussed which allow for an efficient but also meaningful estimate about the effect of applied SWAP gates.

We provide experimental validation to claim that the proposed heuristic significantly outperforms the state-of-the-art heuristic algorithms for a wide range of benchmarks. At the same time, we show that the precise implementation of the look-ahead methodology has a significant impact on what improvements can be gained. Overall, following the proposed methodology, reductions in the number of SWAP gates of 56% can be achieved in the best case. Evaluations have been conducted on 1D as well as 2D quantum architectures.

In the following, Section II revisits the basics on nearest neighbor quantum circuits. Afterwards, the considered problem is motivated before the general idea of the proposed methodology is sketched in Section III. Section IV describes the solution including two implemented schemes of the look-ahead methodology. Limitations as well as how to overcome those are discussed in Section V before results of our experimental evaluations are summarized in Section VI. Finally, the paper is concluded in Section VII.

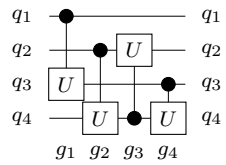
II. Nearest Neighbor Quantum Circuits

Quantum computing relies on manipulating quantum bits (*qubits*) rather than conventional bits. A qubit may assume any state represented by the linear superposition of the basis states $|0\rangle$ and $|1\rangle$. The manipulations are performed by applying specific unitary operations U. Commonly used quantum operations include the Hadamard operation H (setting a qubit into superposition), the phase shift operation S, as well as the NOT operation X. Details on these operations are not relevant in the remainder of this work, but can be found e.g. in [10]. The application of a unitary operation is eventually represented by means of quantum gates, i.e. an *k-qubit quantum gate* applies a $2^k \times 2^k$ unitary matrix to the corresponding qubits. This leads to the following definition of a quantum circuit used in this work.

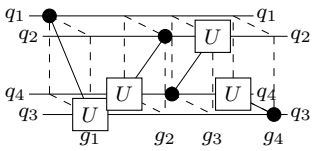
Definition 1. *A* quantum circuit *is a cascade $C = g_1 g_2 \dots g_{|C|}$ of* quantum gates *g_i, where $|C|$ denotes the total number of gates. The number of qubits is denoted by n. Usually, quantum circuits are composed of* unary gates *simply applying the respective unitary operation on a single qubit or* controlled quantum gates *over two qubits. In the latter case, a gate operates on a* control qubit *at position c_i and a* target qubit *at position t_i.*

Example 1. *Consider the circuit depicted in Fig. 1(a) composed of four gates, i.e. $C = g_1 g_2 g_3 g_4$. Control qubits are denoted by a black circle, while target qubits are denoted by U.*

Originally, qubits in a quantum circuit have been arranged in a 1-dimensional (i.e. linear) fashion where each qubit is placed next to each other. Fig. 1(a) shows an example of such a circuit. However, recent technological developments (e.g. [11], [12], [13]) also lead to the consideration of 2-dimensional arrangements where qubits are placed according to a grid-structure. In this case, the circuit from Fig. 1(a) would be realized as sketched in Fig. 1(b). Such arrangements can be extended to higher dimensions eventually leading to *multi-dimensional quantum circuits*.

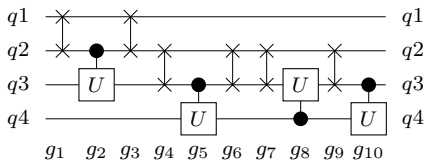


(a) 1D architecture

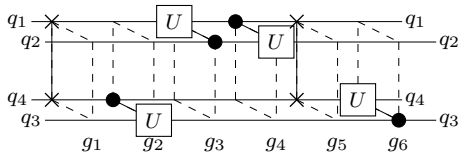


(b) 2D architecture

Fig. 1. Quantum circuits

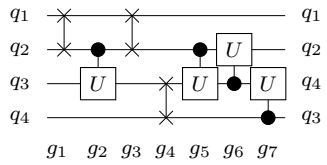


(a) For 1D circuit

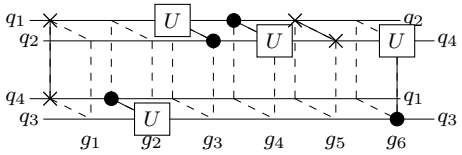


(b) For 2D circuit

Fig. 2. Nearest neighbor quantum circuits



(a) For 1D circuit



(b) For 2D circuit

Fig. 3. Local reordering

Moreover, technological constraints for certain technologies and applications (see e.g. [2], [3]) limit the interaction distance between the qubits and, hence, only allow the application of gates between adjacent (i.e. nearest neighbor) qubits. For the 1D circuit depicted in Fig. 1(a) and the 2D circuit depicted Fig. 1(b), this only holds for gate g_4. All other gates have to be made nearest neighbor-compliant. To this end, so called SWAP gates can be utilized.

Definition 2. *A* SWAP *gate over two qubits q_i, q_j transforms $(q_1, \ldots, q_{i-1}, q_i, q_{i+1} \ldots, q_{j-1}, q_j, q_{j+1}, \ldots, q_n)$ to $(q_1, \ldots, q_{i-1}, q_j, q_{i+1} \ldots, q_{j-1}, q_i, q_{j+1}, \ldots, q_n)$, i.e. simply swaps the value of the two qubits q_i and q_j.*

More precisely, nearest neighbor-compliance of a quantum circuit can be achieved by adding adjacent SWAP gates in front of each gate g_i with non-adjacent control and target qubits to "move" the control (target) qubit of g_i towards the target (control) qubit until they become adjacent. Afterwards, SWAP gates are added to restore the original ordering of the qubits.

Example 2. *Fig. 2(a) exemplary shows the circuit previously considered in Fig. 1(a) which has been made nearest neighbor-compliant by inserting additional SWAP gates (SWAP gate connections are denoted by $\times$). In a similar fashion, this can be applied to the 2D circuit from Fig. 1(b) as shown in Fig. 2(b)*[1].

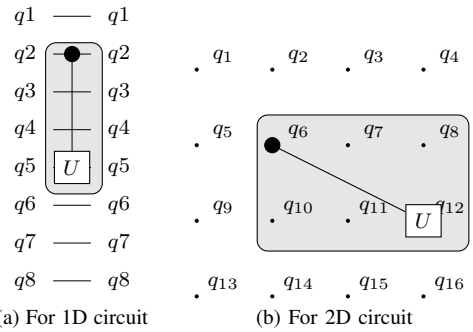
III. Motivation and General Idea

Ensuring nearest neighbor compliance by inserting SWAP gates has heavily been considered in the past and is an established procedure to satisfy the underlying technological constraints. Various solutions have been proposed for this purpose. In this section, we briefly outline the related work and discuss their main drawback to be addressed in this paper. Based on this, we describe the general idea of an alternative nearest neighbor optimization methodology which overcomes this drawback.

A. Related Work and Motivation

According to [9], approaches for nearest neighbor optimization of quantum circuits can roughly be divided into approaches following a global reordering scheme (see e.g. [9], [14]) and approaches following a local reordering scheme (see e.g. [15], [14], [16], [17], [18], [9]). Since the local reordering scheme allows for better results (also confirmed in [9] by means of exact results), we consider this optimization scheme in the following.

Here, SWAP gates may be applied before each non-adjacent gate in order to change the order of the qubits. However, in contrast to the naive scheme sketched before by means of Fig. 2, no SWAP gates are added to restore the original ordering of the qubits. Instead, the respectively resulting qubit order is applied to all remaining gates. As an example, Fig. 3 shows two nearest neighbor quantum circuits (derived from the quantum circuits shown in Fig. 1) which have been determined by following a local reordering scheme. As can be seen, this considerably reduces the number of SWAP gates.



(a) For 1D circuit (b) For 2D circuit

Fig. 4. Options to "move" qubits (gray rectangle highlights SWAP region)

However, a main issue remains: For each non-adjacent gate which is considered, a decision has to be made how exactly the corresponding control/target qubits are "moved" together. In fact, several options exists for this. More precisely:

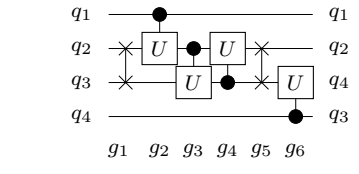
- In 1D architectures, the control qubit can be "moved" to the target qubit, the target qubit can be moved to control qubit, or both qubits can be moved together. This is illustrated in Fig. 4(a).
- In 2D architectures, the control qubit and the target qubit can be "moved" to any adjacent position within a rectangle spanned by the original positions of these qubits. This is illustrated in Fig. 4(b)[2].

In the following, we call the possible nearest neighbor-compliant positions *SWAP regions*. In either case, the precise "movement" influences the qubit/target positions of all following gates and, by this, has a significant effect to the overall costs (i.e. the number of SWAP gates) obtained by nearest neighbor optimization. This is illustrated by the next example:

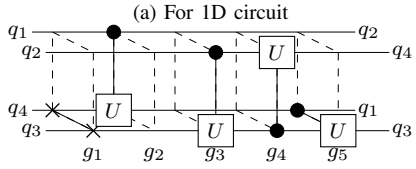
Example 3. *Consider the first gate g_1 in the 1D circuit shown in Fig. 1(a). Following most of the established approaches for nearest neighbor optimization, SWAP gates could have been added as already shown in Fig. 3(a), i.e. the control qubit is "moved" towards the target qubit. However, as shown in Fig. 5(a), "moving" the target qubit towards the control qubit leads to a much better permutation of qubits in which (1) gates g_2 and g_3 become nearest neighbor-compliant as well and (2) where only one further SWAP gate is required to establish nearest neighbor compliance for gate g_4.*

[1] Note that between gates g_2 and g_3 as well as between gates g_3 and g_4, the tuple of identical SWAP gates have been omitted due to space limitations.

[2] Note that it is of course also possible to move the qubits to (adjacent) positions outside the areas sketched in Fig. 4. But this would unnecessarily increase the (local) costs and, hence, is not further considered.



(a) For 1D circuit



(b) For 2D circuit

Fig. 5. Exploiting the options for "moving" qubits together.

Similar issues can be observed in the 2D quantum circuit shown in Fig. 1(b). Established local reordering schemes may motivate swapping qubits q_1 and q_4 in order to make gate g_1 nearest neighbor-compliant (as already discussed using Fig. 3(b)). But all other permutations within the rectangle spanned by these qubits are possible as well. In fact, just swapping qubits q_3 and q_4 is the much better choice as it already makes not only g_1 but the entire circuit nearest neighbor-compliant (as shown in Fig. 5(b)).

Overall, previously proposed solutions for nearest neighbor optimization do not consider the effect a "movement" of qubits may have to the remaining circuit. But since the eventually chosen permutation may have a significant impact on the total number of SWAP gates, in the rest of this paper, we consider the question of how to best exploit the options sketched in Fig. 4 for nearest neighbor optimization.

B. Proposed Solution

In order to determine the best option of how to "move" qubits together, we propose a heuristic methodology which employs a *look-ahead scheme* in order to get an efficient but also meaningful estimate about the impact of an option. The general idea is as follows: Each time a non-adjacent gate g_i is considered, the available options are evaluated by means of the remaining gates $C' = g_{i+1} \dots g_{|C|}$. Then, the option which has the best impact to C' is chosen. In order to measure the impact, *nearest neighbor costs* defined as follows are applied:

Definition 3. *The* nearest neighbor costs *(nnc) for a gate g are defined as*

$$nnc(g) = \begin{cases} 0 & g \text{ is unary} \\ (\sum_{i=1}^{n} |t_i - c_i|) - 1 & \text{otherwise} \end{cases}$$

where c_i and t_i denote the position in the ith dimension of the control and target qubit, respectively. In other words, nearest neighbor costs are either zero or defined by the Manhattan distance between the control and target qubit minus one. In order to determine the nearest neighbor costs with respect to a given permutation π of qubits (e.g. caused by an option to be considered), we apply $nnc_\pi(g) = (\sum_{i=1}^{n} |\pi(t_i) - \pi(g_i)|) - 1$. For a cascade of gates, i.e. a (sub-)circuit C, the nearest neighbor costs are defined as $nnc(C) = \sum_{g \in C} nnc(g)$.

Note that it may be necessary to determine the costs of the composition of two permutations π' and π''. For this, we use the common $\circ$ notation, i.e. $\pi'' \circ \pi'$, to indicate that permutation π'' is applied after applying permutation π'.

Example 4. *Consider the circuit depicted in Fig. 1(a). The first gate has nearest neighbor costs of $nnc(g_1) = 1$, whereas the total circuit has costs of $nnc(C) = 3$. A permutation π that swaps qubits q_1 and q_2 would decrease the costs of the first gate by one while, at the same time, increasing the costs of the next two gates by a total of two. Since the total costs of the resulting circuit would be 4, it might worthwhile to explore other options of how to make g_1 nearest neighbor-compliant.*

Following the proposed look-ahead methodology provides an alternative which may overcome the drawbacks of previously proposed solutions for nearest neighbor optimization. But details on how to apply the scheme – in particular, how to observe the following gates and how to decide for an option based on these observations – remain to be addressed. In fact, several implementations of a look-ahead methodology exists. In the following, we explicitly propose and discuss two representatives of this scheme. As the evaluation later in Section VI shows, the precise implementation will have a significant effect on what improvements can be gained from the methodology.

IV. Look-ahead Schemes for Nearest Neighbor Optimization

The following sections describe two schemes that employ different implementations of the proposed look-ahead methodology. That is, different schemes on (1) how to "look-ahead" (i.e. how to observe the following gates) and (2) how to add SWAP gates into the circuit are proposed. The first scheme follows thereby a joint consideration of the following gates, while the second schemes applies an iterative approach.

A. Joint Consideration Scheme

The main idea of the first scheme is to consider all gates $C' = g_{i+1} \dots g_{|C|}$ following g_i in a joint fashion. From the available options of how to "move" the qubits of g_i together, the one is chosen which has the least negative effect to $|C'|$ (with respect to nearest neighbor costs).

To describe the algorithm in more detail, the following notation of a *circuit-tail* is applied.

Definition 4. *For a given circuit $C = g_1 g_2 \dots g_{|C|}$, a* circuit-tail *of C after gate g_i is defined as $C_i := g_{i+1}, g_{i+2}, \dots, g_{|C|}$.*

The algorithm proceeds as follows: For a given circuit C, iterate over all gates g_i. If the gate is not nearest neighbor-compliant (i.e. $nnc(g_i) > 0$), consider all possible permutations within the SWAP region (see Fig. 4) which would make the gate nearest neighbor-compliant. Next, determine the impact of all these possibilities to the circuit-tail C_i. The permutation that results in the smallest value of $nnc(C_i)$ is chosen and appropriate SWAP gates are added in front of g_i. This procedure is repeated until the end of the circuit is reached.

While doing that, two further issues have to be addressed:

- The permutation leading to the smallest value of $nnc(C_i)$ is not necessarily unique. Hence, if more than one permutation achieving the smallest value is obtained, the directly following gate g_{i+1} is additionally considered.
- When calculating $nnc(C_i)$, the costs of establishing the considered permutation π (i.e. the number of SWAP gates required to generate π) has to additionally be taken into account.

Overall, this leads to a more formal description of the scheme as follows:

1) Initialize the circuit's permutation π_C with the identity function.
2) Iterate over all gates $g_i \in C$. If $nnc_{\pi_C}(g_i) > 0$ then
   a) Store all permutations π with $nnc_{\pi \circ \pi_C}(g_i) = 0$ from g_i's SWAP region in Π.
   b) Determine a permutation $\pi^* \in \Pi$ that minimizes the nearest neighbor costs of the circuit-tail, i.e. $\pi^* = \min_{\pi \in \Pi}\{nnc_{\pi^* \circ \pi_C}(C_i)\}$.
   c) Insert SWAP gates in front of g_i that establishes the permutation π^*.
   d) Update π_C to incorporate π^* (i.e. the new circuit permutation is $\pi^* \circ \pi_C$).
3) Return the nearest neighbor-compliant circuit C.

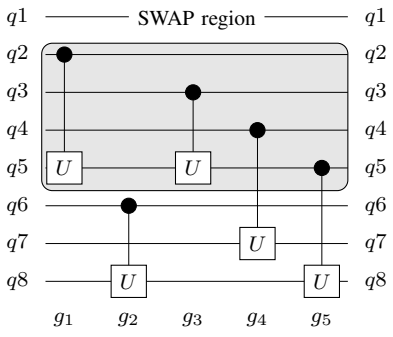


Fig. 6. Cases in the iterative scheme

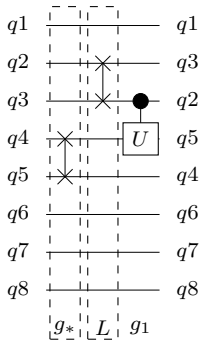


Fig. 7. Res. SWAP gates

Example 5. *Consider again the circuit C shown in Fig. 1(a) with $\pi_C = (q_1, q_2, q_3, q_4)$. The first gate has costs of $nnc(g_1) = 1$. Therefore, a permutation to make this gate nearest neighbor-compliant needs to be found. There are two options within the SWAP region that achieve this: π_1 that swaps q_1 and q_2 and π_2 that swaps q_2 and q_3. As π_1 would lead to costs of 4 (see Example 4) and π_2 would lead to costs of 1, π_2 is chosen. After inserting SWAP gates so that π_2 is established, the algorithm continues until g_4. Here the procedure is applied again eventually leading to the circuit already shown before in Fig. 5.*

B. Iterative Scheme

As an alternative scheme, we propose an approach which does not decide for or against an option by jointly considering the following gates at once, but in an iterative fashion. More precisely, all gates g_j of the circuit-tail C_i following the currently considered gate g_i are considered one after another. For each gate g_j, it is checked whether the positions of the qubits of g_j motivate a particular option to be applied for "moving" the qubits of gate g_i. For this purpose, we can distinguish between three different cases which are sketched in Fig. 6 for 1D circuits[3] (g_1 being the currently considered gate):

1) The qubits of q_j are positioned completely outside of the SWAP region defined by gate g_i (as sketched by gate g_2 in Fig. 6). Then, the gate g_j does not motivate a particular option of how to "move" the qubits of gate g_i and, hence, g_j is ignored.
2) The qubits of q_j are positioned completely inside of the SWAP region defined by gate g_i (as sketched by gate g_3 in Fig. 6). Then, the set of possible options to be applied for gate g_i reduces to only those permutations which reduce the costs of g_j. In the sketch from Fig. 6, "moving" the target qubit of g_1 towards the control qubit is still an option (as it reduces the nearest neighbor costs for both, g_1 *and* g_3), while "moving" the control qubit of g_1 towards the target qubit is not an option anymore (as this would increase the nearest neighbor costs of g_3).
3) The qubits of q_j are positioned partially inside of the SWAP region defined by gate g_i (as sketched by gates g_4 and g_5 in Fig. 6). Here, it depends on the precise configuration of g_j. In the case of gate g_4, "moving" the target qubit of g_1 towards the control qubit would be a preferred option (as, again, it reduces the nearest neighbor costs for both, g_1 *and* g_4). However, this is not the case in the scenario of gate g_5, hence this "movement" would not be used in this case.

Eventually, this leads to the following more formal description of the scheme:

[3]The same cases can analogously be considered for 2D circuits.

1) Initialize the circuit's permutation π_C with the identity function.
2) Iterate over all gates $g_i \in C$. If $nnc_{\pi_C}(g_i) > 0$ then
   a) Set $source = c, target = t, S = \emptyset, j = i + 1$
   b) Retrieve SWAP gates making g_i nearest neighbor-compliant by applying the following sub-algorithm
      i) If either $source$ and $target$ are equal or nearest neighbor-compliant, return $\emptyset$.
      ii) If the end of the circuit is reached, choose a direct permutation from $source$ to $target$, and return its corresponding SWAP gates.
      iii) Create the SWAP region defined by $source$ and $target$.
      iv) If $nnc_{\pi_C}(g_j) = 0$ or if g_j is outside the SWAP region, increase j by one and continue with Step (ii) (Case 1 from above).
      v) Try to determine a SWAP gate g^* within the SWAP region that reduces the costs of g_j. If no such gate exists, increase j by one and continue with Step (ii) (Case 2/3 from above).
      vi) Let π be the permutation established by g^*.
      vii) Let L and R be the SWAP gates determined by applying Steps (i) to (vii) to the paths from the control and target qubit of g_i to the gate g^* (i.e. set $source$ and $target$ accordingly). In the recursion, use $\pi \circ \pi_C$ as the circuit permutation. Return $L \circ g^* \circ R$.
   c) Add the SWAP gates retrieved from Step (b) to C.
   d) Update π_C to incorporate the permutation π that corresponds to the SWAP gates retrieved from step (b) (i.e. the new circuit permutation is $\pi \circ \pi_C$).
3) Return the nearest neighbor-compliant circuit C.

Example 6. *Consider the circuit C shown in Fig. 6. The gate g_1 has costs of 1. The following gate g_2 is not within the SWAP region and, therefore, skipped. Swapping qubits q_4 and q_5 makes gate g_3 nearest neighbor-compliant as it is within the region. Therefore, the permutation π_C is updated to incorporate the corresponding SWAP gate. Now, two new paths are investigated. One is empty (there is no qubit between the former target of g_3 and the end of the SWAP region, i.e. $R = \emptyset$) and the other one is between the updated target of g_3 and the control of g_1. For this path, the SWAP region does not contain any further gates. This allows to directly move g_i's control qubit downwards using a single SWAP gate (i.e. L consists of a single gate only). The resulting SWAP cascade for g_1 is shown in Fig. 7.*

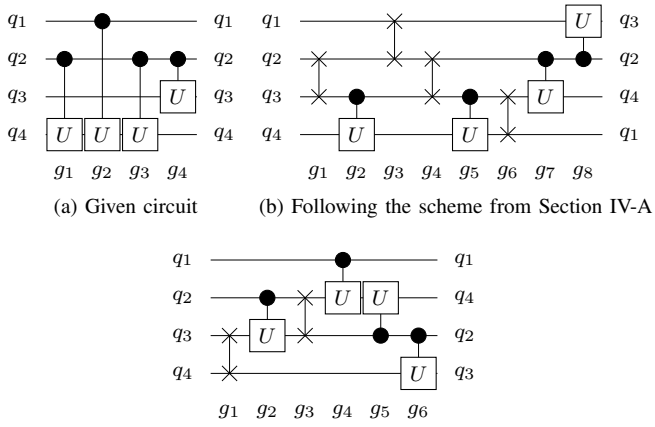
V. Limitations of the Look-ahead Methodology

Following the look-ahead methodology proposed above leads to a nearest neighbor optimization approach which bases its decision how to "move" qubits of a gate g_i together on its circuit-tail C_i. As motivated by the discussions and examples from above, this often proves to be beneficial. However, the general idea that the circuit-tail should be taken into account has its limitations when gates of C_i which follow rather late after g_i suddenly influence how qubits are "moved" together. This becomes particularly crucial when large circuits composed of hundreds or thousands of gates are considered. For example, it is clearly not beneficial anymore to consider e.g. a gate g_{1429} in order to evaluate a decision e.g. for gate g_{14}.

In order to address that, the proposed look-ahead schemes shall only be applied to a restricted circuit-tail. To this end, we apply the concept of a circuit window:

Definition 5. *For a given circuit $C = (g_1, g_2, \ldots, g_{|C|})$, a* circuit window *of size w starting at position i is defined as $C_{i,w} = (g_{i+1}, g_{i+2}, \ldots, g_{i+w})$.*

Applying the proposed look-ahead schemes not on the entire circuit-tail C_i but on a circuit window $C_{i,w}$ restricted by w avoids the negative impacts of gates which follow rather late after g_i.



(a) Given circuit (b) Following the scheme from Section IV-A

(c) Following the same scheme with a window of size 2

Fig. 8. Limitations of the look-ahead methodology

Example 7. *Consider the circuit shown in Fig. 8(a) and let's assume that the look-ahead scheme proposed in Section IV-A is applied in order to evaluate the options how to make gate g_1 nearest neighbor-compliant. Then, Fig. 8(b) shows the resulting circuit when the entire circuit-tail is considered. In contrast, a much cheaper circuit (as shown in Fig. 8(c)) results if only a circuit window $C_{i,2}$ restricted by $w = 2$ is considered.*

Of course, an open question remains to what value to set the size w of the respective circuit windows. Together with a comparison to previously proposed approaches for nearest neighbor optimization, this question is covered in the next section.

VI. Experimental Evaluation

In this section, we report and discuss the most important results obtained by our experimental evaluation of the proposed look-ahead methodology for nearest neighbor optimization. For this purpose, the representative schemes as described in Section IV have been implemented in Java. Evaluations have been performed by means of benchmark circuits (taken from RevLib [19]) which also have been applied in respective related work (e.g. [14], [20]). In a first series of evaluations, we consider the effect of the window sizes to the look-ahead scheme as introduced in Section V. Afterwards, a numerical comparison to results of previously proposed solutions (namely [14], [20]) is provided. In contrast to most of the related work, we consider both, 1D and 2D architectures of quantum circuits.

All evaluations have been conducted on an Xubuntu machine with a 2.6 GHz Intel Core i5 CPU and 8 GB of main memory. All results have been obtained in negligible run-time (i.e. no experiment required significantly more than one CPU-minute) which is why, for sake of clarity, we omitted a detailed run-time discussion in the following[4].

A. Effect of Windows

As discussed in Section V, the window size, which is applied for the look-ahead scheme, may have a significant impact on the final result. Hence, in a first series of experiments we evaluate this effect. To this end, the approaches presented in Section IV have been applied with different window sizes. Afterwards, the respectively obtained number of added SWAP gates has been compared to the results obtained by the approaches with no consideration of windows.

[4]Recently proposed approaches such as [20] often have run-times of up to 30 CPU minutes.

TABLE I. Effect of windows

Benchmark	Gates d	1D Architecture Window size 5	10	20	30	d	Dim.	2D Architecture Window size 5	10	20	30	d
QFT7	21	**-45%**	0%	0%	0%	33	5x2	-20%	**-35%**	0%	0%	20
QFT8	28	**-21%**	-8%	0%	0%	39	4x2	**-39%**	-11%	-21%	0%	28
QFT9	36	**-9%**	0%	0%	0%	54	3x3	**-29%**	**-29%**	3%	-13%	31
QFT10	45	**-32%**	-23%	-19%	0%	94	5x3	-28%	**-30%**	-11%	-13%	53
4gt10-v1_81	36	**-4%**	0%	0%	0%	25	3x2	0%	**-17%**	17%	0%	18
cycle10_2_110	1212	-30%	**-36%**	-42%	-33%	1659	3x4	**-37%**	-29%	-33%	-38%	775
ham15_108	458	-5%	-5%	**-7%**	**-7%**	568	5x3	-12%	-13%	**-17%**	0%	270
ham7_104	87	**-18%**	**-18%**	-16%	**-18%**	88	3x3	0%	**-18%**	-11%	-4%	45
hwb5_55	106	**-15%**	0%	-3%	0%	78	3x2	**-34%**	-9%	-9%	-5%	56
hwb6_58	146	-12%	**-13%**	-5%	3%	128	2x3	**-29%**	-23%	-11%	-2%	83
hwb7_62	2659	**-40%**	-39%	-39%	-31%	3431	3x3	-37%	**-38%**	-36%	-32%	1682
hwb8_118	16608	**-42%**	-40%	-40%	-34%	22838	3x3	-39%	**-41%**	-39%	-33%	10735
hwb9_123	20405	-38%	**-39%**	-37%	-33%	31262	4x3	-37%	**-39%**	-36%	-31%	13909
mod5adder_128	81	**-31%**	-10%	-4%	-9%	67	3x2	**-38%**	-23%	-36%	-26%	53
plus127mod8192_162	65455	-39%	-42%	**-44%**	-40%	113223	5x4	-33%	-38%	**-40%**	-39%	45988
plus63mod4096_163	29019	-39%	-41%	**-44%**	-38%	45918	3x5	-34%	-39%	**-40%**	-37%	19460
plus63mod8192_164	37101	-35%	-39%	**-41%**	-38%	60345	5x3	-28%	-33%	**-35%**	-33%	23935
rd53_135	78	-13%	**-14%**	-13%	-13%	77	3x3	-21%	-27%	**-38%**	-29%	48
sym9_148	4452	**-39%**	-34%	**-39%**	-36%	5111	4x4	-41%	**-43%**	-40%	-37%	2538
urf1_149	57770	**-20%**	**-20%**	**-20%**	-16%	57452	3x3	**-19%**	-15%	-13%	-11%	36281
urf2_152	25150	**-22%**	**-22%**	-20%	-18%	23679	2x4	**-23%**	-18%	-14%	-14%	16697
urf3_155	132340	**-24%**	**-24%**	**-24%**	-23%	143210	4x3	**-20%**	-17%	-15%	-12%	87154
urf5_158	51380	-21%	**-22%**	**-22%**	-19%	51400	3x3	**-20%**	-15%	-14%	-11%	32259
urf6_160	53700	-21%	-25%	**-26%**	**-26%**	73994	4x4	-15%	**-16%**	-14%	-14%	37559
Shor3	2076	**-9%**	-7%	2%	-3%	2312	4x3	**-32%**	-21%	-16%	-14%	1487
Shor4	5002	**-25%**	-15%	0%	1%	7443	3x4	**-28%**	-18%	-12%	-9%	3833
Shor5	10265	**-32%**	-20%	-18%	-11%	18073	5x4	**-23%**	-20%	-13%	-1%	8269
Shor6	18885	**-42%**	-35%	-20%	-14%	39209	4x6	**-30%**	-27%	-20%	-16%	17914

Table I summarizes the results for the scheme proposed in Section IV-A[5]. The first two columns provide the name of the respectively considered benchmark circuits (as given in RevLib [19]) as well as its total number d of gates (note that unary gates are not considered here as they are irrelevant for nearest neighbor optimization). Afterwards, the results obtained by the scheme are reported for window sizes of 5, 10, 20, 30, and d (i.e. without any window). In order to ease the evaluation, we used the configuration without the consideration of window circuits as baseline (i.e. provide absolute values here). All remaining results are given in proportional relation to that baseline. Results are reported for 1D as well as 2D architectures (in case of 2D architectures, the dimension of the respectively considered 2D grid is additionally provided).

The results clearly show that the consideration of window circuits has a significant effect on the number of required SWAP gates. By this, the discussions conducted in Section V are confirmed. In the best cases, improvements of up to 45% can be achieved when not *all* following gates are respectively considered but just a subset defined by the window size. Concerning the best window size, the results slightly differ. But overall, it can be concluded that the windows size should be rather small – in most of the cases, no significant further improvements are observed with window constituted with more than 20 gates[6].

B. Comparison to Related Work

In a second series of experiments, we compared the results obtained by our approach to the state of the art. Since corresponding evaluations have been conducted focusing on 1D architectures or 2D architectures only, we consider the approach proposed in [14] for 1D architectures and proposed in [20] for 2D architectures. Both represent recently proposed methodologies for nearest neighbor architectures for the respective architectures and, hence, guarantee an up-to-date evaluation.

The results of the comparisons are summarized in Table II. Again, the first two columns provide the name of the respectively considered benchmark circuits as well as its total number d of gates. Afterwards, the best results obtained by the approaches presented in Section IV as well as the best results obtained by the approaches from [14], [20] are listed – again for 1D as well as 2D architectures. Additionally, we list the proportional improvement of the proposed approaches.

As can be seen, the look-ahead methodology proposed in this work leads to substantial improvements in the majority of

[5]Similar results have been obtained for the scheme proposed in Section IV-B, but precise numbers are omitted due to space limitations.

[6]In fact, we did not only apply fix window sizes (i.e. 5, 10, 20, ...), but also evaluated window sizes depending on the total number of gates or the total number of gates left to be considered. In all these configurations similar results have been obtained.

TABLE II. COMPARISON TO RELATED WORK

		1D Architecture					2D Architecture				
		Lookahead-joint	Lookahead-iterative	Prev. Work	Improvement		Lookahead-joint	Lookahead-iterative	Prev. Work	Improvement	
Benchmark	Gates d	(Sect. IV-A)	(Sect. IV-B)	[14]	Scheme 1	Scheme 2	(Sect. IV-A)	(Sect. IV-B)	[20]	Scheme 1	Scheme 2
QFT7	21	18	23	29	-38%	-21%	13	22	18	-28%	22%
QFT8	28	31	33	41	-24%	-20%	17	25	18	-6%	39%
QFT9	36	49	53	66	-26%	-20%	22	27	34	-35%	-21%
QFT10	45	64	67	96	-33%	-30%	37	43	53	-30%	-19%
3_17_13	13	6	8	5	20%	60%	5	8	6	-17%	33%
4gt10-v1_81	36	24	38	29	-17%	31%	15	22	16	-6%	38%
aj-e11_165	59	33	42	43	-23%	-2%	16	37	24	-33%	54%
cycle10_2_110	1212	966	2096	2193	-56%	-4%	483	824	839	-42%	-2%
ham15_108	458	531	838	803	-34%	4%	223	355	328	-32%	8%
ham7_104	87	72	84	86	-16%	-2%	37	53	48	-23%	10%
hwb5_55	106	66	101	86	-23%	17%	37	64	45	-18%	42%
hwb6_58	146	111	146	140	-21%	4%	59	85	79	-25%	8%
hwb7_62	2659	2067	3406	3480	-41%	-2%	1050	1703	1688	-38%	1%
hwb8_118	16608	13176	22877	21767	-39%	5%	6316	11096	11027	-43%	1%
hwb9_123	20405	18988	32405	32979	-42%	-2%	8522	14459	15022	-43%	-4%
mod5adder_128	81	46	85	79	-42%	8%	33	45	41	-20%	10%
plus127mod8192_162	65455	63364	128510	136820	-54%	-6%	27549	52333	53598	-49%	-2%
plus63mod4096_163	29019	25617	51824	54999	-53%	-6%	11764	22160	22118	-47%	0%
plus63mod8192_164	37101	35472	73218	77753	-54%	-6%	15484	29939	29835	-48%	0%
rd53_135	78	66	85	96	-31%	-11%	30	47	39	-23%	21%
sym9_148	4452	3103	6297	5353	-42%	18%	1455	2799	2363	-38%	18%
urf1_149	57770	45730	60301	62019	-26%	-3%	29252	41058	38555	-24%	6%
urf2_152	25150	18428	24861	23608	-22%	5%	12872	18101	16822	-23%	8%
urf3_155	132340	108321	145250	140908	-23%	3%	69693	95485	94017	-26%	2%
urf5_158	51380	39852	53202	54038	-26%	-2%	25887	36813	34406	-25%	7%
urf6_160	53700	54815	64600	91563	-40%	-29%	31540	43100	43909	-28%	-2%
Shor3	2076	2112	2529	3353	-37%	-25%	1010	1485	1710	-41%	-13%
Shor4	5002	5616	6128	9510	-41%	-36%	2757	3807	4264	-35%	-11%
Shor5	10265	12221	12358	22846	-47%	-46%	6344	8504	8456	-25%	1%
Shor6	18885	22829	23156	41551	-45%	-44%	12468	15970	20386	-39%	-22%

the cases. At the same time, it can be observed that the precise implementation of this methodology has a significant impact on what improvements can be gained: While the scheme from Section IV-A performs very well on almost all benchmark circuits, limited improvements can be observed using the scheme from Section IV-B. But overall, the look-ahead methodology leads to substantial improvements compared to recently proposed approaches for both 1D and 2D architectures – in the best case, the number of SWAP gates can be reduced by to 56% and 49%, respectively.

VII. CONCLUSIONS

In this work, we proposed a methodology for nearest neighbor optimization of quantum circuits which, in contrast to previous work, incorporates a look-ahead scheme. By this, SWAP gates – added to make a particular gate nearest neighbor-compliant – are chosen with respect to their impact to the following gates. Two different schemes are presented and discussed as possible implementations of the proposed methodology. An experimental evaluation confirmed that the precise implementation of the proposed look-ahead methodology has a significant impact on what improvements can be gained. Overall, improvements of up to 56% have been obtained in the best case. The proposed methodology has been evaluated for 1D and 2D quantum circuits.

ACKNOWLEDGMENTS

This work has partially been supported by the European Union through the COST Action IC1405 and the Nanyang Technological University through grant M4081438.

REFERENCES

[1] M. Veldhorst, J. Hwang, C. Yang, A. Leenstra, B. de Ronde, J. Dehollain, J. Muhonen, F. Hudson, K. Itoh, A. Morello *et al.*, "An addressable quantum dot qubit with fault-tolerant control-fidelity," *Nature nanotechnology*, vol. 9, no. 12, pp. 981–985, 2014.

[2] J. M. Chow, J. M. Gambetta, E. Magesan, D. W. Abraham, A. W. Cross, B. Johnson, N. A. Masluk, C. A. Ryan, J. A. Smolin, S. J. Srinivasan *et al.*, "Implementing a strand of a scalable fault-tolerant quantum computing fabric," *Nature communications*, vol. 5, 2014.

[3] E. H. Lapasar, K. Kasamatsu, S. N. Chormaic, T. Takui, Y. Kondo, M. Nakahara, and T. Ohmi, "Two-qubit gate operation on selected nearest-neighbor neutral atom qubits," *Journal of the Physical Society of Japan*, vol. 83, no. 4, p. 044005, 2014.

[4] Z. Wang, X. Gu, L.-A. Wu, and Y.-x. Liu, "Quantum simulation of pairing hamiltonians with nearest-neighbor interacting qubits," *arXiv preprint arXiv:1411.5097*, 2014.

[5] A. G. Fowler, C. D. Hill, and L. C. Hollenberg, "Quantum-error correction on linear-nearest-neighbor qubit arrays," *Physical Review A*, vol. 69, no. 4, p. 042314, 2004.

[6] P. Pham and K. M. Svore, "A 2d nearest-neighbor quantum architecture for factoring in polylogarithmic depth," *Quantum Information & Computation*, vol. 13, no. 11-12, pp. 937–962, 2013.

[7] M. Wallquist, J. Lantz, V. Shumeiko, and G. Wendin, "Superconducting qubit network with controllable nearest-neighbour coupling," *New Journal of physics*, vol. 7, no. 1, p. 178, 2005.

[8] P. Kumar, "Efficient quantum computing between remote qubits in linear nearest neighbor architectures," *Quantum Information Processing*, vol. 12, no. 4, pp. 1737–1757, 2013. [Online]. Available: http://dx.doi.org/10.1007/s11128-012-0485-5

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

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

[11] L. C. L. Hollenberg, A. D. Greentree, A. G. Fowler, and C. J. Wellard, "Two-dimensional architectures for donor-based quantum computing," *Phys. Rev. B*, vol. 74, p. 045311, 2006. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevB.74.045311

[12] M. Kumph, M. Brownnutt, and R. Blatt, "Two-dimensional arrays of radio-frequency ion traps with addressable interactions," *New Journal of Physics*, vol. 13, no. 7, p. 073043, 2011. [Online]. Available: http://stacks.iop.org/1367-2630/13/i=7/a=073043

[13] N. H. Nickerson, Y. Li, and S. C. Benjamin, "Topological quantum computing with a very noisy network and local error rates approaching one percent," *Nat Commun*, vol. 4, p. 1756, 2013.

[14] M. Saeedi, R. Wille, and R. Drechsler, "Synthesis of quantum circuits for linear nearest neighbor architectures," *Quant. Info. Proc.*, vol. 10, no. 3, pp. 355–377, 2011.

[15] Y. Hirata, M. Nakanishi, S. Yamashita, and Y. Nakashima, "An efficient method to convert arbitrary quantum circuits to ones on a Linear Nearest Neighbor architecture," *Quantum, Nano and Micro Technologies. ICQNM*, pp. 26–33, 2009.

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

[17] A. Matsuo and S. Yamashita, "Changing the gate order for optimal lnn conversion," in *Reversible Computation*, ser. Lecture Notes in Computer Science, A. Vos and R. Wille, Eds. Springer Berlin Heidelberg, 2012, vol. 7165, pp. 89–101. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29517-1_8

[18] R. Wille, A. Lye, and R. Drechsler, "Optimal SWAP gate insertion for nearest neighbor quantum circuits," in *ASP Design Automation Conf.*, 2014, pp. 489–494.

[19] 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*, 2008, pp. 220–225, RevLib is available at http://www.revlib.org.

[20] A. Shafaei, M. Saeedi, and M. Pedram, "Qubit placement to minimize the communication overhead in circuits mapped to 2D quantum architectures," in *ASP Design Automation Conf.*, 2014, pp. 495–500.