Efficient Implementation of Nearest Neighbor Quantum Circuits Using Clustering with Genetic Algorithm

Anirban Bhattacharjee[1], Chandan Bandyopadhyay[3], Angshu Mukherjee[1], Robert Wille[2], Rolf Drechsler[3], Hafizur Rahaman[1]
[1]Indian Institute of Engineering Science and Technology Shibpur, India-711103
[2]Institute for Integrated Circuits, Johannes Kepler University Linz, A-4040 Linz, Austria
[3]Institute of Computer Science, University of Bremen & Cyber-Physical Systems, DFKI GmbH, 28358 Bremen, Germany
Email: anirbanbhattacharjee330@gmail.com, chandan@uni-bremen.de, angshumukherjee100@gmail.com, robert.wille@jku.at, drechsle@uni-bremen.de, rahaman_h@it.iiests.ac.in

***Abstract*—Although quantum computing has made tremendous progress in last couple of years and technologies like NMR, Ion Trap, superconducting qubits have come out as promising platforms to implement quantum computing devices, such technologies are facing several design constraints. One such constraint is the Nearest Neighbor (NN) property, which demands the adjacency of logical qubits. Aiming to contribute to this cause, here we are proposing an improved design approach for transforming quantum circuits for NN-based architectures using genetic algorithms. In this work, our primary objective remains to form efficient NN structures by restricting the SWAP usage. In the design phase, initially, we use the *k-means* clustering scheme for partitioning the qubits into separate objects and, then, a genetic algorithm is applied that eventually fixes the order of qubits for each individual cluster. In the final phase, all these local solutions are combined and, again, a genetic algorithm is employed to obtain a global solution. We have tested our approach over a large spectrum of benchmarks and improvements are registered over some state-of-the-art design works.**

Keywords— Quantum Circuit,quantum gate, Nearest Neighbour(NN), SWAP gate, Quantum Cost(QC).

I. INTRODUCTION

Limitations of classical computing have resulted in an evolution of an alternative computational technology called quantum computing. The emergence of such technology promises to provide more efficient solutions for certain complex problems like factorization in RSA cryptosystem [1] or database search [2] for which no efficient classical algorithms exist. As a result, the design of quantum algorithms gets immense priority in order to aid the realization of practical applications for quantum computing devices. However, several challenges exist that need to be addressed for this purpose. Amongst these, fault-tolerance is considered an essential factor and it is found that this issue can be addressed by incorporating quantum error correction codes [3], like surface code [4]. But the implementation of such code demands nearest neighbor (NN) restrictions in which the quantum gates need to act on adjacently placed qubits only. Moreover, long interaction distances between the qubits are more susceptible to noise which may lead to computational errors.

Experimentally, it is observed that close qubit interactions reduce such computational errors [5]. Additionally, quantum circuits implementing technologies like ion-trap [6], quantum dots [7], nuclear magnetic resonance [8] and superconducting qubits [9] consider NN interaction as a necessary design constraint. Consequently, the consideration of NN representations has become important to design algorithms for such architectures. The most standard way of achieving NN transformations is to insert a sequence of SWAP gates before quantum gates with non-adjacent qubits. Following such an approach, however, causes an overhead in the resulting circuit with respect to both, depth and gate count. Based on this, efficient synthesis of NN circuits using less SWAP overhead becomes an essential design challenge. This can be fulfilled by rearranging the original positions of the qubits. For this purpose, a wide variety of techniques for an efficient NN representation have been proposed.

For example, a linear nearest neighbor (LNN) realization of quantum circuits using solutions based on templates and reordering strategies (global and local) has been presented in the article [10]. To obtain a better LNN representation, the authors of [11] followed an efficient method which not only reduces the additional circuit overhead but also brings down the time complexity. To improve the linear structure further, a design approach based on circuit partitioning has been developed in [12], where, initially, the input circuit is partitioned into sub-circuits and, then, MINLA approach is executed for obtaining an improved LNN solution. In way to contain the complexity of examining all possible qubit permutation orders, a compact and dedicated data structure representation is employed in the work [13]. To reduce the SWAP count in the overall netlist, look-ahead schemes have been introduced in [14]. Based on the scheme of [14], an advanced version of this look-ahead methodology has been presented in [15]. Exact design solutions have also been introduced in the works [16, 17, 18] that produce an optimal LNN solution with minimal SWAP overhead. However, such approaches are not feasible for larger circuits due to the enormous time consumption. Hence, developing further solutions for the improved realization of NN circuits remains a research topic.

978-1-7281-5406-0/20/$31.00 ©2020 IEEE
DOI 10.1109/ISMVL49045.2020.00-32

Focusing on the need, in this article, we emphasize on developing an intelligent qubit policy based on global reordering of qubits empowered with genetic algorithms. But to determine the best qubit order requires investigation of all possible permutation of qubits. For circuits with n qubits, $n!$ qubit permutations are possible, i.e the complexity grows exponentially. To tackle this complexity, we introduce a heuristic NN design solution which simplifies the search process by exploring only parts of the state space. This approach results in optimal or nearly optimal results for many benchmark circuits.

The rest of the paper is structured as follows. Section II presents an overview of nearest neighbor quantum circuits. Reordering of NN-compliant circuits for the entire netlist is presented in Section III. A description of the proposed reordering solution is discussed in Section IV. An experimental analysis followed by a comparison to some of the earlier works is provided in Section IV. Finally, we conclude the paper in Section V.

II. Nearest Neighbor Based Quantum Circuits

Like bits used in classical computing, qubits represent the information units for quantum computing. A qubit is considered as a two-state quantum system existing in one of the basis states of $|0\rangle$ and $|1\rangle$. In addition to this, qubit states can also exist in a superposition of basis states which can be expressed in the form of state vector ($|\xi\rangle$), $|\xi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β represents the amplitude values that are subject to the condition $\alpha^2 + \beta^2 = 1$. Operations performed on such an n-qubit system can be conducted through the employment of quantum gates which is described as multiplication of $2^n \times 2^n$ unitary transformation matrices. The number of such unitary operators acting together in a specific manner realizes a quantum circuit.

Definition 1: *A network containing a cluster of quantum gates operating over a number of circuit lines forms a quantum circuit.*

Table 1: Symbolic representation of some quantum gates

Gates	Symbol	Gates	Symbol
NOT		Control led -V	
CNOT			
V		Control led -V†	
V†			

The main basic building block for quantum circuits is quantum gate. The schematic representation of commonly used 1-qubit and 2-qubit quantum gates from NCV library [19] are shown in Table 1.

Moreover, physical constraint of certain quantum technologies restricts the qubit interaction distance and only permits the quantum gates to act on qubits placed at adjacent locations. This phenomenon is known as nearest neighbor property. To bring non-adjacently placed interacting qubits close to each other, a special type of gate termed as SWAP is used (the graphical representation of SWAP along with its elementary composition is depicted in Fig. 1).

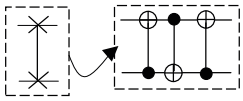

Fig.1: Pictorial representation of the SWAP gate

Definition 2: *The interaction distance occurring between the positions of control and target qubits of any 2-qubit gate is called as Nearest Neighbor Cost (NNC).*

This NN cost [23] calculation can be performed by estimating the difference between the control and target qubit positions as given in Eqn. (1):

$$NNC_g = |c - t| - 1 \quad (1)$$

The above cost expression determines the nearest neighbor cost of a 2-input gate g whose control/target qubits are at positions c and t, respectively. Adding all such individual costs of each gate g results in an overall NN cost for the entire circuit network C and this estimation can be formulated as

$$NNC_C = \sum NNCg = \sum(|c - t| - 1). \quad (2)$$

From this expression, we can infer that a given circuit can be considered nearest neighbor compliant if it contains only gates with adjacently placed interacting qubits – resulting in $\sum \text{NNCg} = 0$. For a better apprehension of the above statements, the transformation of rudimentary quantum circuits to NN architectures is discussed in the next example.

Example 1: *Let's consider the circuit shown in Fig. 2(a) which has $NNC_C = 6$, since the interacting qubits of each gate do not occur at adjacent locations ($NNC_g \neq 0$). Once the SWAP gates are placed before and after each non adjacent gate, a corresponding NN circuit results as shown in Fig. 2(b). In this transformed design, a total of 12 SWAP gates have been embedded to form the NN architecture.*

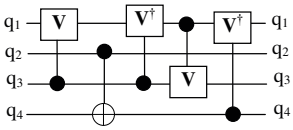

Fig. 2(a): Quantum circuit having $NNC_C = 6$

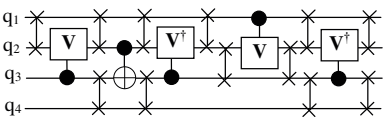

Fig. 2(b): NN complaint design for Fig. 2(a)

III. Global Reordering of NN Circuits

Reducing the SWAP overhead in NN architectures can be achieved through implementations of global reordering schemes, which emphasize on changing the initial qubit order before adding SWAP gates such that the cost overhead in the resulting NN architecture is reduced. Let's take an example to explain it.

Example 2: *Consider again the circuit from Fig. 2(a) and its corresponding NN design from in Fig. 2(b). Now changing the initial qubit order from $q_1q_2q_3q_4$ to $q_2q_3q_1q_4$ leads to the circuit shown in Fig. 3(a) and its NN cost is reduced from 6 to 2. The*

NN representation of this circuit (as depicted in Fig. 3(b)) can be obtained using only 4 SWAPs compared to 12 required in Fig. 2(b).

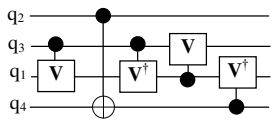

Fig. 3(a): Quantum circuit having $NNC_C = 2$

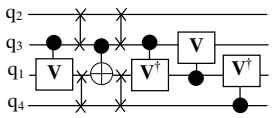

Fig. 3(b): NN circuit of Fig. 3(a)

By investigating the above example, it can be inferred that reordering of the qubit positions has a significant effect on the NN configuration of the circuit. In the next section, an NN synthesis workflow based on global qubit reordering is proposed.

III. Proposed Approach

Here, we introduce an improved design solution for efficient transformation of quantum circuit to NN architectures. This transformation involves five design phases: i) Graph formation, ii) Cluster Formation, iii) Finding local solution, iv) Merging of clusters, v) Finding global solution. For a better apprehension of the entire design flow, we use the circuit shown in Fig. 4 with which all the phases are illustrated.

Phase 1: Graph Formation

In this phase, a complete graph is constructed for a given circuit specification in which the vertices represent the qubits, while the edges interconnect these vertices. Each of the edges is assigned a weight indicating the degree of interaction between the two corresponding qubits of the circuit and it can be computed heuristically using

$$e_{ij} = I_i * I_j / N(\sum 1/T_{ij}), \qquad (3)$$

where the notations I_i, I_j, represent the interaction count of qubits q_i and q_j while notations N and T_{ij} are the total gate count in the given circuit and the timestamp of gates appearing between the qubits q_i and q_j respectively. The significance of considering the term $1/T_{ij}$ is to assign priority to the gates depending on its respective positions and it decreases from left to right of the circuit.

For the circuit shown in Fig. 4, its corresponding complete graph representation is depicted in Fig. 5.

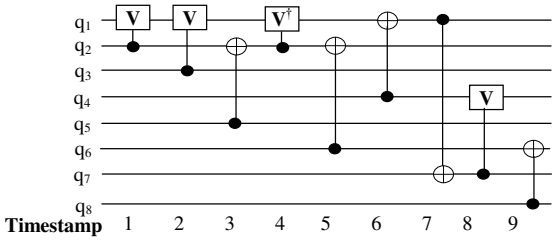

Fig. 4: Input quantum circuit

The edge weight of this complete graph is determined using the expression given in Eqn. (3). There are nodes whose interconnecting edges have been assigned zero weight due to absence of any gate acting on those qubits.

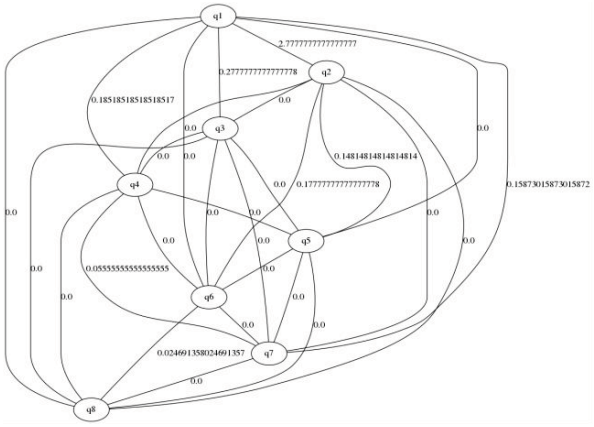

Fig. 5: Graphical representation of circuit in Fig. 4

Phase 2: Cluster Formation

After constructing the complete graph, the purpose of the second phase is to partition the entire graph into sub-graphs obtained in Phase 1. In this process, the *k*-means algorithm has been applied and the qubits representing the vertices are divided into distinct groups of clusters based on the *k*-means policy. The objective behind implementing this clustering technique is to reduce the complexity of finding appropriate solutions for large problem sizes (for circuits with large number of qubits, the number of possible permutations increases). This strategy basically groups the qubits into distinct clusters and, then, finds solution for each cluster separately, which in turn helps to find solutions fast.

The number of clusters (Num_C) required for any given circuit can be determined heuristically using a logarithm function, namely

$$Num_C = \lceil 2.9 * \ln(0.18 * NumQ) \rceil, \qquad (4)$$

where the parameter Num_Q is the number of qubits of a given circuit. This expression computes the number of clusters (Num_C) needed for qubit partitioning and it increases with the number of qubits such that the cluster sizes does not differ much. The constants in the above logarithmic expression are obtained through experimentation. This expression is feasible only when the following condition is satisfied:

$$\begin{aligned} Num_C &\geq 2 \quad \text{iff} \quad Num_Q \geq 8 \text{ otherwise} \\ Num_C &< 2 \quad \text{iff} \quad Num_Q < 8 \end{aligned} \qquad (5)$$

More precisely, qubit partitioning or clustering can only take place when a given circuit contains either 8 or more than 8 qubits; otherwise no partitioning is required, i.e. all qubits are placed in a single cluster (for qubit size of 6 and 7, the expression returns a unit value indicating no cluster formation while for circuits with 3 to 5 qubits, clustering becomes irrelevant since the expression in Eq. (4) returns a negative value). It has been verified through experimentation that, for partitioning purposes, this phase becomes feasible only when the qubit size reaches the threshold value of 8. Otherwise, clustering becomes insignificant for circuits with less than 8 qubits since possible qubit permutations can be determined manually.

The size of each cluster remains almost same and its maximum size (CS_{MAX}) is kept fixed which can be computed by

$$CS_{MAX} = Num_Q / Num_C \quad (6)$$

After determining the number of clusters (Num_C) for any input circuit (of size n qubits), we randomly chose Num_C qubits as the center for each such cluster. In other words, qubits are randomly assigned as cluster centers. The remaining qubits are then assigned to these clusters based on the objective function

$$Obj_{cluster} = \max \sum_{i=1}^{Num_C} \sum_{q \in C_i} E_{weight}(q, c_i) . \quad (7)$$

This expression indicates that qubits (q) are assigned to the clusters C_i, where $i = 1$ to Num_C based on their maximum edge weight (E_{weight}) with respect to cluster centers c_i. Before assigning any qubit to a cluster, we need to verify whether the maximum cluster size has been reached. If this is not the case, we would assign it to the said cluster. Otherwise, we are assigning it to the next best (based on $Obj_{cluster}$) available cluster. In this manner, all the qubits are assigned to distinct clusters.

After the initial assignment of the qubits to different clusters in the first iteration, the following two steps are repeated.

1. From each cluster, a qubit is selected as new cluster center for the next iteration based on roulette wheel.
2. Once all the cluster centers are obtained, the remaining qubits are assigned to these new clusters in a similar manner using the cost function $Obj_{cluster}$. If this computed value is greater than the previous one, then only the clusters will be updated. Otherwise, it remains the same as in the previous iteration.

For a better apprehension, we again consider the input circuit shown in Fig. 4. As this circuit contains eight qubits, clustering becomes suitable according to the conditions stated in Eqn. (5). Using the expression given in Eqn. (4), qubits are partitioned into two clusters based on the objective function $Obj_{cluster}$. Here, the formed clusters are $C_1\{q_5, q_6, q_7, q_8\}$ and $C_2\{q_1, q_2, q_3, q_4\}$ respectively. So, only two clusters will be formed for the given circuit at the end of this phase.

Phase 3: Finding Local Solution

After partitioning the qubits into distinct clusters as discussed in the previous section, we find intra-cluster solution in which a suitable qubit order is determined for each cluster. For this purpose, a nature-inspired meta-heuristic scheme is applied extensively on each cluster separately. Amongst all the possible qubit permutation order within each cluster, we randomly select a subset that has the maximum cluster size. This selection actually forms the population size (qubit sequence order is considered as the genetic chromosome) of the clusters and the random subset orders within each cluster represent the corresponding members of the population. Each of these cluster populations are evaluated using a fitness function.

In our case, we have considered the fitness function equivalent to that of nearest neighbor cost expression given in Eqn. (2). To create new members for the next generations, some random pairs are chosen (as parents with best fitness values from the current population are selected using roulette wheel approach. From these parent members, new members are generated by performing crossover followed by mutation. The crossover operation is carried out by randomly selecting a crossover point across the parents in which the contents appearing before the point of first parent is copied directly in the offspring, while the remaining elements are appended in the order they occur in the second parent. The mutation operation is being executed with a defined probability besides choosing two random mutation points across the parents. This process is repeated iteratively until the termination condition is reached which is set as the iteration count in our approach.

For the circuit shown in Fig. 4, two clusters $C_1\{q_5, q_6, q_7, q_8\}$ and $C_2\{q_1, q_2, q_3, q_4\}$ are formed as discussed in Phase 3. After following all the genetic operations stated above, the resulting cluster representations produced at the end of this phase are $C_1\{q_8, q_6, q_7, q_5\}$ and $C_2\{q_3, q_2, q_1, q_4\}$, respectively. This solution represents the corresponding qubit permutation orders within each cluster.

Phase 4: Merging of Clusters

In this phase, we randomly combine all the formed clusters into a single cluster. At the end of the previous phase, cluster solutions with proper qubit ordering are achieved and these solutions are combined together in a random manner to generate an overall qubit order for the given netlist. Such cluster merging is being conducted in such a way that the order of qubits within each cluster is preserved in the final merged qubit order representation.

For a better apprehension, we have considered the previously formed clusters ($C_1\{q_8, q_6, q_7, q_5\}$, $C_2\{q_3, q_2, q_1, q_4\}$). After randomly merging the elements of these clusters, it yields the resulting qubit order as $C\{q_3, q_8, q_6, q_7, q_2, q_1, q_4, q_5\}$.

Phase 5: Finding Global Solution

Similar to Phase 3, here, we also apply a genetic algorithm for finding a qubit order. But now we consider the resulting qubit order sequence obtained from Phase 4. All the operations associated with this algorithm are applied to obtain an overall qubit ordering for the entire circuit. For circuits with less than eight qubits, this step is applied directly after Phase 1. Otherwise, all the steps (from Phase 1 to 3) are executed sequentially.

To obtain the globally reordered sequence for the initial qubit positions for the design of Fig. 4, again a genetic algorithm is applied. The qubit ordering ($C\{q_3, q_8, q_6, q_7, q_2, q_1, q_4, q_5\}$) obtained from the cluster combination is again permuted into $C\{q_7, q_4, q_1, q_3, q_2, q_5, q_6, q_8\}$ by re-executing the steps of the genetic algorithm. The circuit representation with this resulting qubit ordering ($\pi q_7 q_4 q_1 q_3 q_2 q_5 q_6 q_8$) is shown in Fig. 6. To realize its corresponding NN circuit (shown in Fig. 7), an overall of 8 SWAP gates are required (instead of the 32 gates which are needed with the initial qubit order of $\pi q_1 q_2 q_3 q_4 q_5 q_6 q_7 q_8$). Hence, a significant reduction in the SWAP overhead is achieved using this global reordering scheme.

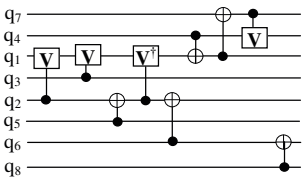

Fig. 6: Globally reordered circuit for Fig. 4

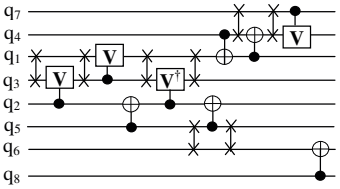

Fig. 7: NN realization of circuit from Fig. 6

IV. Experimental Results

In this section, the performance of the proposed reordering solution is evaluated. To this end, results from our experimental evaluations are reported. Our design workflow has been developed in C++ and processed on an Intel core i5 processor with 3.30 GHz and 4GB RAM configurations.

Experimental analysis has been conducted on a set of benchmark specifications taken from [20] and the resulting outcomes are tabulated in Table 2 and Table 3 respectively.

The best results obtained for each benchmark circuit by running our algorithm repeatedly have been reported, while the remaining cases are omitted due to space limitations. The genetic approach mutation operation is conducted with probability of 0.7, while no crossover probability is being considered. Both, clustering and genetic phases, are executed for 40 iterations.

Table 3: Results from higher qubits synthetic benchmarks

Benchmark names	No. of qubits (n)	Gate Count	Prop. Soln.		
			No. of cluster formed	SWAP count	Time (sec.)
rev_17	17	136	4	1360	0.078
rev_18	18	153	4	1632	0.081
rev_19	19	171	4	1938	0.089
hm_20	20	73	4	270	0.088
hm_21	21	79	4	330	0.091
hm_22	22	85	4	346	0.012
ac_21_1	21	130	4	370	0.168
ac_21_2	21	67	4	408	0.095
ac_21_3	22	42	4	274	0.093
rdom_22	22	55	4	164	0.163
rdom_24	24	47	5	168	0.30
rdom_27	27	53	5	268	0.74
rdom_30	30	59	5	346	0.97
rdom_33	33	65	6	366	1.17
add23_218	23	58	5	222	0.18
add29_218	29	74	5	314	0.52
add8_172	25	96	5	286	0.47

From the investigation of our result set recorded in Table 2, it can be inferred that, by changing the position of qubits using our approach, efficient NN-compliant circuits are produced which significantly reduce the SWAP overhead over works [10] and [13]. The result set from Table 2 shows that an overall cost reduction about 34.5% (69.91% in the best case) is achieved (compared to the work [10]) while a 8.28% reduction (best case value of 21.12%) compared to [13] is recorded. For some benchmark circuits, the column 'No. of cluster formed' contains '0' which indicates the circuits that do not meet the feasibility condition discussed in Eqn. (5). Additionally, the runtime (in seconds) of our method as well as that of work [13] is reported in Table 2. Optimal SWAP count for benchmarks is also mentioned under the column header "*optimal SWAP count*" in Table 2 and it is very promising that our design solution provides either the optimal value (highlighted with yellow color) or nearly optimal results. Further investigations also reveals that our method produces results faster than [13]. The SWAP requirement for large benchmark circuits (synthetic benchmarks) are also recorded in Table 3, which also provides optimal and sub-optimal solutions.

V. Conclusion

This article discussed the problem of NN optimization using the global reordering scheme and also aimed to reduce the SWAP cost of the resulting circuits. In this work, we implemented a qubit reordering policy based on genetic algorithms and the *k-means* technique to obtain better solutions. The entire design flow is implemented in five different phases in which certain random pathways are conducted so as to avoid huge computational runtimes. We also have verified the effectiveness of our methodology by testing our design workflow over a large spectrum of benchmarks and comparing the computed results with some reported linear NN based design solutions. This method can also be extended further for higher dimensional structures (such as considered in [21]) by undertaking some modifications in the workflow. In the future, we will try investigating whether the proposed global reordering scheme to optimize quantum circuits can be extended so that they can be efficiently be used for IBM Q architectures [22].

References

[1] P.W Shor,"Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,"SIAM J. Comput. 26 (5), 1484-1509 (1997).

[2] L.K Grover, "A fast quantum mechanical algorithm for database search," In. Symposium on the Theory of Computing, pp. 212-219 (1996).

[3] 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.

[4] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," Physical Review A, vol. 86, no. 3, p. 032324, Sep. 2012.

[5] H. Haffner et al., "Scalable multiparticle entanglement of trapped ions, " *Nature*, vol. 438, no. 7068, pp. 643-646, Dec 2005.

[6] D. Kielpinski, C. Monroe, D.Wineland. "Architecture for a largescale ion-trap quantum computer",*Nature*, 417(6890):709–711, 2002.

[7] J. Taylor, J. Petta, A. Johnson, A. Yacoby, C. Marcus, and M. Lukin,"Relaxation, dephasing, and quantum control of electron spins in double quantum dots". *Physical Review B*, 76(3):035315, 2007.

[8] B. Criger, G. Passante, D. Park, and R. Laflamme. "Recent advances in nuclear magnetic resonance quantum information processing". *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*,370(1976):4620–4635, 2012.

[9] A. Blais, J. Gambetta, A. Wallraff, D. Schuster, S. Girvin, M. Devoret, and R. Schoelkopf. "Quantum information processing with circuit quantum electrodynamics". *Physical Review A*, 75(3):032329, 2007.

[10] M. Saeedi, R. Wille, R. Drechsler. "Synthesis of quantum circuits for linear nearest neighbor architectures."*Quant. Inf. Proc*., 10(3):355–377, 2011.

[11] Y. Hirata, M. Nakanishi, S. Yamashita and Y. Nakashima, "An efficient conversion of quantum circuits to a linear nearest neighbor architecture", *Quantum Info. Comput.,* vol. 11, no. 1, pp. 142-166, Jan 2011.

[12] A. Shafaei, M. Saeedi, and M. Pedram. Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. *Design Autom. Conf.*, 2013.

[13] R. Wille, N. Quetschlich, Y. Inoue, N. Yasuda, and S. Minato. "Using π-DDs for Nearest Neighbor Optimization of Quantum Circuits" *In Conference on Reversible Computation*, pages 181-196, 2016.

[14] R. Wille, O. Keszocze, M. Walter, P. Rohrs, A. Chattopadhyay, and R. Drechsler, "Look-ahead schemesfor nearest neighbor optimization of 1D and 2D quantum circuits," in *Proc. ASP Design Autom. Conf.*, Jan 2016, pp. 292–297.

[15] A. Bhattacharjee, C. Bandyopadhyay, R. Wille, R. Drechsler, and H. Rahaman, "Improved Look-ahead Approaches for Nearest Neighbor Synthesis of 1D Quantum Circuits, " *In Intl. Conference on VLSI Design*, DOI: 10.1109/VLSID.2019.00054, Jan 2019.

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

[17] R. Wille, A. Lye and R. Drechsler, "Optimal SWAP gate insertion for nearest neighbor quantum circuits," in Proc. ASP Design Automation Conf. Suntec, Singapore: IEEE, 2014, pp. 489-494.

[18] A. Zulehner, S. Gasser, and R. Wille, "Exact Global Reordering for Nearest Neighbor Quantum Circuits Using A*," In Conference on Reversible Computation, 185-201, 2017.

[19] Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H.: Elementary gates for quantum computation. Physical review A, 52(5), p.3457, 1995.

[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 Multiple-Valued Logic (ISMVL), pages 220-225, 2008. RevLib is available at http://www.revlib.org.

[21] A. Lye, R. Wille, and R. Drechsler. Determining the Minimal Number of SWAP Gates for Multi-dimensional Nearest Neighbor Quantum Circuits. In IEEE ASP-DAC, pages 178-183, 2015.

[22] A. Zulehner, A. Paler, and R. Wille, "An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures," IEEE TCAD, 2018.

[23] A. Bhattacharjee, C. Bandyopadhyay, R. Wille, R. Drechsler, H. Rahaman, "A Novel Approach for Nearest Neighbor Realization of 2D Quantum Circuits," IEEE ISVLSI 2018 (pp. 305-310).

Table 2: Comparison with related state-of-the-art NN techniques

Benchmark names	No. of qubits	Gate count	Prop. soln.			[10] No. of (SWAPs)	[13] No. of (SWAPs)	[13] Time (sec.)	**Optimal SWAP count (as mentioned in [13])**	% imprv. over	
			No. of cluster formed	SWAP count	Time (sec.)					[10]	[13]
3_17_13	3	14	0	6	0.0056	6	6	0.1	4	0.0	0.0
4gt4-v0_80	6	34	0	44	0.0192	46	44	0.1	-	4.34	0.0
4gt10-v1_81	5	36	0	32	0.021	76	32	0.1	-	57.89	0.0
4gt5_75	5	22	0	22	0.012	33	22	0.1	-	33.33	0.0
4gt11_84	5	7	0	2	0.0154	6	2	0.1	2	66.66	0.0
rd32-v0_67	4	8	0	4	0.0109	4	4	0.1	4	0.0	0.0
4gt13-v1_93	5	17	0	8	0.014	20	8	0.1	8	60	0.0
4_49_17	4	32	0	32	0.0082	32	32	0.1	-	0.0	0.0
4gt12-v1_89	5	52	0	52	0.018	85	52	0.1	-	38.82	0.0
4mod7-v0_95	5	40	0	44	0.0153	104	44	0.1	-	57.69	0.0
hwb4_52	4	23	0	18	0.009	28	18	0.1	18	35.71	0.0
alu-v4_36	5	32	0	34	0.01362	62	34	0.1	-	45.16	0.0
aj-e11_165	5	59	0	52	0.01369	65	52	0.1	52	20	0.0
mod5adder_128	6	87	0	120	0.0191	196	120	0.2	-	38.77	0.0
rd53_135	7	78	0	136	0.027	208	136	0.3	136	34.61	0.0
ham7_104	7	87	0	140	0.025	190	140	0.3	140	26.31	0.0
mod8-10_177	6	108	0	156	0.0194	166	156	0.1	156	6.02	0.0
hwb5_55	5	106	0	120	0.0143	165	120	0.1	120	27.27	0.0
hwb6_58	6	146	0	290	0.0199	374	294	0.2	290	22.45	1.36
hwb8_118	8	14260	2	49790	0.080	56108	50184	0.6	-	11.26	1
hwb9_123	9	18124	2	71576	0.103	96317	74086	0.8	-	25.68	3.38
rd73_140	10	76	2	150	0.0990	190	178	0.9	150	21.05	15.73
sys6-v0_144	10	62	2	114	0.096	260	118	0.8	114	56.15	3.38
sym9_148	10	4452	2	10984	0.104	20992	12128	0.7	10984	47.67	9.43
urf1_149	9	57770	2	179832	0.0815	200460	203836	0.6	179832	10.29	11.77
urf2_152	8	25150	2	71280	0.0564	90676	73932	0.4	71280	21.39	3.58
Shor3	10	2076	2	4802	0.097	-	4802	0.8	4802	-	0.0
urf6_160	15	53700	3	249952	0.277	427072	257604	10.0	241208	41.47	3
plus63mod4096_163	12	25492	3	114170	0.188	127506	144752	1.9	113104	10.45	21.12
plus63mod8192_164	13	32578	3	149776	0.231	203488	178122	5.6	149708	26.39	15.91
cycle10_2_110	12	1126	3	4104	0.148	6822	4500	1.6	4104	39.84	8.8
rd84_142	15	112	3	308	0.271	528	348	4.7	284	41.66	11.49
ham15_108	15	458	3	1354	0.272	4500	1438	2.5	1340	69.91	6
Shor5	14	10256	3	34680	0.228	-	-	-	34680	-	-
Shor6	16	18885	4	76318	0.308	-	-	-	76318	-	-
% average improvement										**34.5**	**8.28**