Qubit Placement to Minimize Communication Overhead in 2D Quantum Architectures

Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram

Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089
{shafaeib,msaeedi,pedram}@usc.edu

Abstract— Regular, local-neighbor topologies of quantum architectures restrict interactions to adjacent qubits, which in turn increases the latency of quantum circuits mapped to these architectures. To alleviate this effect, optimization methods that consider qubit-to-qubit interactions in 2D grid architectures are presented in this paper. The proposed approaches benefit from Mixed Integer Programming (MIP) formulation for the qubit placement problem. Simulation results on various benchmarks show 27% on average reduction in communication overhead between qubits compared to best results of previous work.

I. Introduction

Quantum computing can offer significantly higher performance for a set of problems compared to what we have now, commonly-called classical computing. Quantum algorithms with superpolynomial speedup on a quantum computer include algorithms for number factoring, solving discrete-log and Pell's equation, and walk on a binary welded tree [1].

A well-known technique to implement a quantum algorithm on a quantum computer is to run a quantum physics experiment under the control of a classical computer. The experimental apparatus consists of physical qubits such as ions or photons where the quantum-mechanical properties of qubits are used to perform the required computation. A real-time classical computer directs the experiment by issuing instructions and reading out the quantum states. Final result may require post-process computation and answer checking.

A non-ideal quantum computer, however, is subject to noise and faces numerous limitations and constraints. Environmental disturbances and errors in the control systems are two common examples, which if ignored, can result in computational error. The error rate limits the computation length. In addition, current quantum technologies are subject to constraints on parallelism, *connectivity*, and bandwidth, which further limit the implementation of quantum algorithms.

Various proposals for quantum technologies with 1D, 2D and 3D interactions have been introduced. In general, 1D architectures with only two neighbors per qubit are highly restrictive, and 3D architectures with six neighbors per qubit are difficult to control. Hence, the most promising architecture for a quantum computing system is to arrange qubits in a 2D structure with four neighbors per qubit. Quantum technologies with 2D architectures include neutral atoms [2], superconductors [3], photonics [4], and quantum dots [5].

A physical realization of a quantum program can couple any distant two qubits with some communication overhead. However, this can result in a long sequence of operations, which in turn increases circuit latency and error rate. For instance, the nearest-neighbor communication overhead results in 175x reduction in error threshold for fault-tolerant error correction with a concatenated 7-qubit CSS code [6]. Improving error threshold is costly — it may require a more sophisticated control protocol to construct gates with higher fidelities or a more robust error correction code. Accordingly, optimization of quantum circuits are crucial in order to reduce the communication overhead.

During recent years, several techniques have been proposed to map arbitrary circuits to 1D quantum architectures [7]–[9], which as mentioned earlier have limited number of neighboring qubits. On the other hand, the few works on 2D architectures are hand-optimized techniques designed for special type of quantum circuits. Our focus, however, is to develop a design automation method to optimize qubit interactions considering the connectivity constraint in quantum technologies that use a 2D grid architecture.

Although conventional placement algorithms for VLSI physical design can be used for placement of qubits, the performance of such approaches is limited (Section IV). In this paper, we first characterize similarities and differences between the conventional placement algorithm and the one used in quantum technologies. Then, a Mixed Integer Programming (MIP) formulation is proposed as a standard grid placement algorithm to optimize qubit-to-qubit interaction. The proposed MIP formulation results in a valid placement for qubits. However, direct application of this formulation ignores specific properties of quantum technologies. Accordingly, the MIP formulation is improved by some heuristic techniques to properly capture the effects of quantum architectures.

The rest of this paper is organized as follows. We introduce basic concepts in Section II. Previous work is reviewed in Section III. Section IV discusses the proposed approach followed by experiments in Section V. Finally, paper is concluded in Section VI.

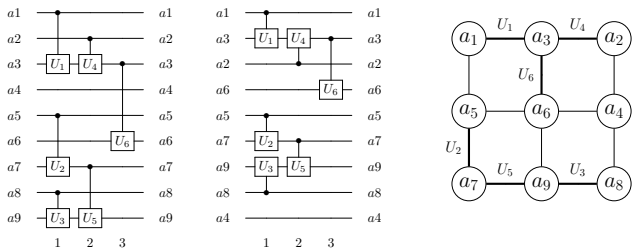

Fig. 1. A sample quantum circuit (left) and its implementation in 1D (middle) and 2D grid (right) architectures. The gate in time step 3 has a non-adjacent interaction in 1D architecture. However, all interactions in the 2D grid involve neighboring qubits.

II. Basic Concepts

In quantum computation, a quantum bit (qubit) is a unit of information which takes a linear superposition of the basis states $|0\rangle$ and $|1\rangle$. An n-qubit *quantum gate* performs a specific $2^n \times 2^n$ unitary operation on the selected n qubits. We do not use particular properties of any 1- or 2-qubit gates, except for 2-qubit SWAP gate. Therefore, we omit definitions. More information can be found in standard quantum computing textbooks and surveys [10], [11].

A quantum algorithm is described by a *quantum circuit* where a set of quantum gates is applied to transform the initial state of the quantum system into a final state. Each gate can involve an arbitrary number of qubits. The resulting circuit is then 'compiled' into another quantum circuit based on a library of primitive one- and two-qubit gates. This quantum circuit is an input to our problem.

Given a quantum circuit with one- and two-qubit gates, one should *map* the circuit into a quantum apparatus, which is a physical experiment that realizes the quantum circuit. The underlying quantum experiment is usually modeled as a *connectivity graph* with pre-defined connectivity patterns between graph nodes where nodes represent physical qubits. Therefore, a complete graph is an ideal quantum architecture with no limit on qubit interactions; and a path is a 1D architecture where only neighboring qubits on a line can interact (see Fig. 1 for an example). On the other hand, the quantum circuit is modeled with another graph, called *interaction graph*, where nodes denote qubits of the circuit. In this case, for each 2-qubit gate working on qubits i and j, an edge is added between nodes i and j in the graph.

Given an interaction graph and a connectivity graph, the mapping problem is a standard graph embedding problem with connectivity and interaction graphs as the *host* and *guest* graphs, respectively. The objective is then to minimize the total distance between adjacent nodes of the interaction graph. For a 2D grid connectivity graph, the mapping problem is NP-complete. Also, determining whether a given interaction graph can be embedded into a 2D grid is NP-complete [12].

If a solution for the grid-embedding problem is known, all circuit qubits have corresponding physical qubits. The next step is to apply quantum gates, which requires gates to be adjacent. This means that all connected nodes in the interaction graph should be placed on adjacent grid nodes. For a qubit located at (i, j) in the grid, all qubits in locations $(i, j-1)$ (left), $(i-1, j)$ (up), $(i, j+1)$ (right), $(i+1, j)$ (down) are neighbors. Therefore for non-adjacent qubits (i, j) and (m, n) a connection should be made, which is achieved by applying a sequence of either MOVE or SWAP operations.

If MOVE operation is not supported by the quantum experiment, adjacent qubits should be exchanged step by step to transform a qubit from (i, j) to one of the four neighbors of (m, n). Since exchanging two neighboring qubits requires one SWAP gate, the total number of SWAP gates by this process is $|m-i|-1$ if $n = j$ (same column), $|n-j|-1$ if $m = i$ (same row), and $|m-i|+|n-j|-1$ otherwise.

The added MOVE or SWAP operations are considered as communication overhead since such gates are not imposed by the original algorithm or circuit. Optimizations should be applied to reduce this overhead. In this paper, we focus on the quantum experiments that only support SWAP gates and not MOVE operations. Quantum technologies based on superconducting [3] and quantum dots [5] are examples of SWAP-based technologies.

III. Previous Work

Certain circuits are amenable for specific interaction-cost optimizations. Examples include circuits for quantum Fourier transform [13], quantum adders [14], [15], modular exponentiation [16], [17], and error correction codes [6], [18]. A more general approach is developed in [19] where particular operations spanning n wires, e.g., rotation of n wires, were analyzed to be optimized for depth.

Optimization of arbitrary quantum circuits for 1D architectures is the topic of recent papers. Minimal number of SWAP gates required to transform one permutation of qubits in a line into another permutation was explored in [8]. Minimizing the number of SWAP gates by changing qubit locations dynamically was investigated in [7]. Minimum linear arrangement problem was employed in [9] to find (near-) optimal solutions, with respect to the number of SWAP gates, in different parts of an interaction graph. All these methods are based on 1D architectures. In [20], the authors considered qubit-to-qubit interaction optimization to map a circuit to a physical device where the underlying quantum device is a general graph (not a grid).

IV. The Proposed Method

The conventional circuit placement problem in VLSI design starts with a (weighted) hypergraph where nodes represent standard cells and hyperedges denote connections among these cells. Each node of the hypergraph has a pre-defined size. Circuit placement determines center positions for nodes such that a specific objective function is optimized and some constraints are met (e.g., no overlap between cells). This is followed by a routing step that connects the placed cells via

wires. Total wirelength, circuit delay, and power consumption are typical objectives in the VLSI physical design algorithms.

The qubit placement problem is similar to the conventional circuit placement problem, with some differences. In general, VLSI placement algorithms can be used for embedding a weighted undirected interaction graph. Also, similar to the minimized total wirelength in the conventional placement problem, we are interested in qubit placements with minimal total distance between connected nodes. However, in qubit placement, positions of instructions are not fixed, whereas in the conventional VLSI circuit placement gates (or instructions) are fixed. This time-variant nature of qubit placement imposes dynamic placement. Additionally, nodes (qubits) have no width and height in the qubit placement problem.

Dynamic placement of qubits can be used to reduce communication overhead. More precisely, after placing qubits in specific grid nodes in SWAP-based quantum technologies, one needs to exchange qubits step by step to 'route' two distant qubits towards each other in order to apply a gate. Location of other qubits on the path will change accordingly. To follow the placement solution, all moved qubits should return to their initial location by reversely applying the same sequence of SWAP gates. As used in e.g., [7] for 1D architectures, instead of returning qubits to their initial location, one may keep the current (updated) placement, and then apply the remaining gates based on the new locations of qubits.

Since VLSI designs include numerous gates, the most successful VLSI placement tools [21] apply several heuristics to avoid unbearable runtime. However, current quantum technologies are limited to a small number of qubits. Hence, we used an MIP-based grid-placement algorithm. Any other placement technique can also be used to solve the grid-embedding problem.

A. MIP-based Formulation

The MIP-based grid-embedding problem assigns each qubit to a unique location on the 2D grid such that frequently interacting qubits are placed as close together as possible. As a consequence, less number of SWAP gates is required in order to route qubits.

To mathematically formulate the problem, a binary variable x_{ij} is used which represents the assignment of $qubit_i$ (node i in the interaction graph) to $location_j$ in the grid. Moreover, w_{ik} denotes the weight between $qubit_i$ and $qubit_k$ in the interaction graph (i.e., the number of gates between them in the circuit), and $dist_{jl}$ represents the Manhattan distance between $location_j$ and $location_l$ in the grid. Hence, the cost of assigning $qubit_i$ to $location_j$ (i.e., x_{ij}) and $qubit_k$ to $location_l$ (i.e., x_{kl}) can be expressed as $c_{ijkl} = w_{ik} \times dist_{jl}$. The problem is then formulated as follows:

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} c_{ijkl}x_{ij}x_{kl} \tag{1}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1,\ldots,n, \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1,\ldots,n, \tag{3}$$

$$x_{ij} \in \{0,1\}, \qquad i,j = 1,\ldots,n. \tag{4}$$

In this formulation, n is the number of grid nodes. More precisely, $n = hw$ for an $h \times w$ grid where h and w denote the number of rows and columns, respectively. Dummy nodes are also added to the interaction graph for the MIP formulation in cases where the number of qubits is less than n.

The objective function (1) is not linear; however, several equivalent formulations that linearize this objective function have been proposed. Among them, Kaufmann and Broeckx's linearization [22] has the smallest number of variables and constraints [23] which is described next. By defining $z_{ij} = x_{ij}\sum_{k=1}^{n}\sum_{l=1}^{n} c_{ijkl}x_{kl}$ for $i,j = 1,\ldots,n$, we can rewrite the objective function as $\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}\sum_{k=1}^{n}\sum_{l=1}^{n} c_{ijkl}x_{kl} = \sum_{i=1}^{n}\sum_{j=1}^{n} z_{ij}$. Authors of [22] then proved that the following MIP formulation is equivalent to Eq. (1) - (4):

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{n} z_{ij} \tag{5}$$

subject to

$$(2),(3),(4),$$

$$\alpha_{ij}x_{ij} + \sum_{k=1}^{n}\sum_{l=1}^{n} c_{ijkl}x_{kl} - z_{ij} \le \alpha_{ij}, \quad i,j = 1,\ldots,n, \tag{6}$$

$$z_{ij} \ge 0, \qquad i,j = 1,\ldots,n, \tag{7}$$

where $\alpha_{ij} = \sum_{k=1}^{n}\sum_{l=1}^{n} c_{ijkl}$ for $i,j = 1,\ldots,n$. This new formulation involves n^2 binary variables (x_{ij}'s), n^2 real variables (z_{ij}'s), and $n^2 + 2n$ constraints.

The above MIP formulation can find optimal placement solution with respect to the aforementioned objective and constraints. However, the resulting qubit placement may not be a valid solution for a SWAP-based quantum technology. In other words, the MIP formulation does not guarantee that all two-qubit gates become local; rather, it tends to place qubits that frequently interact with other as close as possible on the grid. Therefore, a mechanism is required to localize all two-qubit gates. For this purpose, after the MIP problem is solved, two-qubit gates are checked in order until a non-local gate is found. Afterwards, the corresponding control qubit is routed towards the target qubit based on xy routing algorithm (first along x-axis and then along y-axis) by inserting SWAP gates.

B. MIP-based Optimization Framework

The qubit placement discussed in Section A is obtained by applying the grid-embedding formulation on the whole

interaction graph. Basically, the interaction graph has no view on scheduling of instructions. In other words, while $w_{i,k}$ reflects the number of interactions (or gates) between qubits i and k, qubits may interact in very different time steps. In this case, placing qubits i and k close to each other in the whole computation is not useful — one may place highly interacting qubits at different scheduling levels close to each other and move them to other locations when the corresponding qubits will not interact to leave space for other qubits.

In general, a small number of consecutive gates in a given circuit can be executed in parallel due to sharing control or target qubits. Accordingly, a given circuit is (almost) scheduled. Hence, working with gates at one scheduling level results in very few gates. As an alternative approach, we can apply m instances of the grid-embedding formulation on m subsets (subcircuits) of the interaction graph for a circuit with N gates. In this case, the interaction graph for subcircuit j is obtained by only considering consecutive gates between time steps $(j-1)*N/m+1$ and $j*N/m$. Thus, each subcircuit can work on N/m gates simultaneously.

Using several instances of the grid-embedding problem, qubit placements for subcircuits j and $j+1$ can be different. This requires a swapping network to align qubit arrangement of subcircuit j with qubit arrangement of subcircuit $j+1$. For this purpose, we use the snake-like indexing (shown in Fig. 2(a)) with 2D bubble sort algorithm [24, Chapter 9]. While for 1D bubble sort, one can move the maximum element among unsorted items towards its proper location in one way, this is not the case in a 2D grid. If x and y are the row and column differences between an element and its proper location respectively, then the number of paths to move the element towards its proper location is $\frac{(x+y)!}{x!y!}$. Different paths for each element can affect other elements in the grid, which may result in very different number of SWAP gates. Moreover, considering the effect of moving minimum or maximum elements exacerbates the situation. Fig. 2(b) shows one example using two different strategies.

Fig. 3 illustrates the structure of the final circuit which is obtained from the following three steps. (1) MIP-based grid-embedding problem is solved for each subcircuit j in order to find the initial qubit placement of that subcircuit, P_j^i. (2) SWAP gates are inserted before non-local two-qubit gates of each subcircuit j which eventually leads to its final quibit placement, P_j^f. (3) In the end, swapping networks are added to change the final qubit placement of subcircuit j to initial qubit placement of subcircuit $j+1$ ($P_j^f \rightarrow P_{j+1}^i$ for $1 \leq j \leq m-1$).

V. Experimental Results

Proposed methods were implemented in C++ and tested on a server machine with 4 Intel E7-8837 processors and 64GB memory. For MIP solver, we used Gurobi Optimizer Ver. 5.5.0 [25], which uses linear-programming relaxation techniques along with other heuristics in order to quickly solve large-scale MIP problems.

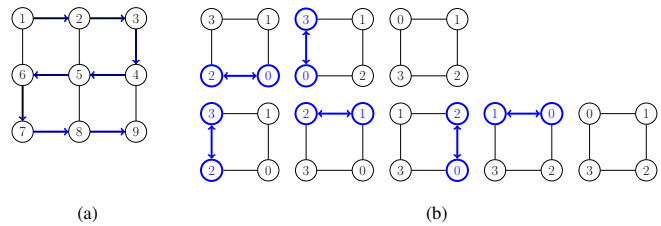

Fig. 2. (a) Snake-like indexing scheme for a 3×3 2D grid. It also shows how a path (traced by arrows) can be mapped on a 2D grid. (b) Sorting on a 2D grid based on different strategies can result in different number of SWAPs. In the first row, minimum element is always moved in XY direction (2 SWAPs). In the second row, maximum element is moved in XY direction (4 SWAPs).

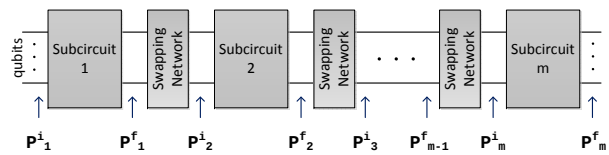

Fig. 3. Dynamic qubit placement via partitioning circuit into m subcircuits. P_j^i and P_j^f denote the initial and final qubit placements of subcircuit j, respectively. The swapping networks align qubit placements.

There are two methods in literature on optimization of communication overhead in 2D architectures for modular exponentiation [17] and adders [14], [15]. The method in [17] added $O(n^4)$ ancilla to reach $O(\log^2 n)$ depth for modular exponentiation. We do not add ancillae and our focus is on circuit size. Applying our techniques on log-size adders does not improve circuits in [15] (which improves [14]) in most cases. In particular, the optimizations in [15] are similar to our Method 2 (described next) while qubit placements for components were hand-optimized.

To evaluate the proposed methods we used reversible benchmarks in [7], [9] along with circuits for Shor's algorithm in [8]. The previous techniques [7]–[9], [16] are based on 1D interactions, which can also be mapped to 2D architectures[1]. However, using 2D interactions adds more flexibility and thus can lower the communication overhead. Accordingly, we do not intend to compare our results with 1D architectures. Instead, 1D results are reported to consider the effect of architectures on reducing overhead.

Runtime for the conventional placement algorithms in VLSI design is important, but as a secondary objective. In quantum computing, runtime for qubit placement is much less important, given that quantum technologies are in preliminary stages — there is no aggressive time to market. Accordingly, the main objective is still circuit quality. Additionally, due to the limitations of current quantum technologies to work with a large number of gates, investing runtime in favor of circuit quality is reasonable. Therefore, we used a time-limit of 30 minutes for each attempted benchmark. For small

[1]For mapping a 1D path onto a 2D grid, please see Fig. 2(a).

benchmarks, runtime varies from a few seconds to 5 minutes. For larger ones, we reported the best result after 30 minutes.

For each circuit, we applied two methods:

- **Method 1:** This method uses a single grid-based MIP formulation on the whole interaction graph. When the *global* qubit placement solution is found, all 2-qubit gates are checked in order and SWAP gates are inserted before each non-adjacent gate. Placement of qubits will change accordingly. This new qubit placement is considered for the remaining gates.
- **Method 2:** Multiple instances of the grid-based placement problem are used in this method. For each instance, we used k consecutive gates. If a circuit includes $< k$ gates, the interaction graph is analyzed at once, same as Method 1. After finding a qubit placement for each instance, SWAP gates are applied before each non-adjacent gate. Swapping networks are also required between any two consecutive placements. We used a 2D bubble sort algorithm that moves (1) the maximum element in XY direction, (2) the maximum element in YX direction, (3) the minimum element in XY direction, (4) the minimum element in YX direction towards its proper location, and then selects the best network.

The results of applying the aforementioned methods are reported in Table I. In this table, for each benchmark we reported the number of qubits and the number of gates in the original circuit, as well as the number of two-qubit gates after decomposing the circuit based on [26] into one- and two-qubit gates. Columns 5-9 report the grid size ($h \times w$) that results in the smallest number of SWAPs along with its corresponding number of SWAP gates after applying the proposed methods. For Method 2, we also report the percentage of SWAPs in the swapping network as compared with total number of SWAPs (column 9). Column 10 shows the minimum number of SWAP gates achieved by our proposed methods (i.e., minimum of columns 6 and 8). Best prior result from different sources are also presented in Columns 11-12.

Comparing the best results for 2D architectures versus the best prior result for 1D architectures shows that the number of SWAP gates can be reduced extensively if one allows interactions in 2D architectures. As can be seen in Table I, our methods improve the best results of 1D architectures 27% on average and up to 67%. A sample circuit mapped to a 2D grid based on Method 1 is also illustrated in Fig. 4.

VI. Conclusion

We optimized qubit-to-qubit interactions in quantum technologies that allow 2D grid architectures. To achieve this, we formulated our problem by mixed integer programming as a grid-embedding problem. To consider scheduling of gates, the interaction graph is partitioned into several instances where the grid-embedding formulation is applied on each instance. To align qubit placement of one instance in the interaction graph with qubit placement of another instance, we applied a 2D bubble sort algorithm. Furthermore, for each benchmark various grid sizes were examined to find the one with smallest number of SWAP gates. The proposed methods result in considerable reduction of communication overhead in 2D architectures. However, further heuristics can be applied to reduce the overhead more. For small circuits the prior methods for 1D architectures result in better circuits.

There are several lines for future researches. For very large circuits, conventional placement algorithms can be adopted to solve the graph-embedding problem. In addition, grid clustering and hierarchical qubit placement may be considered for large circuits, where our proposed method can be applied in each hierarchy level. Another topic for future research is to directly focus on depth of circuits. The method in [19] considered circuit depth for several basic operations in 1D architectures. The problem for 2D architectures is new and indeed interesting.

Acknowledgements

This research was supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center contract number D11PC20165. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

References

[1] D. Bacon and W. van Dam. Recent progress in quantum algorithms. *Commun. ACM*, 53:84–93, Feb. 2010.

[2] M. Saffman, T. Walker, and K. Mølmer. Quantum information with rydberg atoms. *Rev. Mod. Phys.*, 82:2313–2363, Aug 2010.

[3] A. Blais et al. Quantum-information processing with circuit quantum electrodynamics. *Phys. Rev. A*, 75:032329, Mar 2007.

[4] E. Knill, R. Laflamme, G. Milburn. A scheme for efficient quantum computation with linear optics. *Nature*, 409:46–52, 2001.

[5] J. M. Taylor et al. Relaxation, dephasing, and quantum control of electron spins in double quantum dots. *Phys. Rev. B*, 76:035315, 2007.

[6] T. Szkopek et al. Threshold error penalty for fault-tolerant quantum computation with nearest neighbor communication. *IEEE Trans. Nanotechnol.*, 5(1):42 – 49, jan. 2006.

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

[8] Y. Hirata, M. Nakanishi, S. Yamashita, Y. Nakashima. An efficient conversion of quantum circuits to a linear nearest neighbor architecture. *Quant. Inf. Comput.*, 11(1–2):0142–0166, 2011.

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

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

[11] M. Saeedi and I. L. Markov. Synthesis and optimization of reversible circuits - a survey. *ACM Comput. Surv.*, 45(2):21:1–21:34, March 2013.

[12] S. N. Bhatt and S. S. Cosmadakis. The complexity of minimizing wire lengths in VLSI layouts. *Inform. Process. Lett.*, 25(4):263 – 267, 1987.

[13] Y. Takahashi, N. Kunihiro, and K. Ohta. The quantum Fourier transform on a linear nearest neighbor architecture. *Quant. Inf. Comput.*, 7:383–391, 2007.

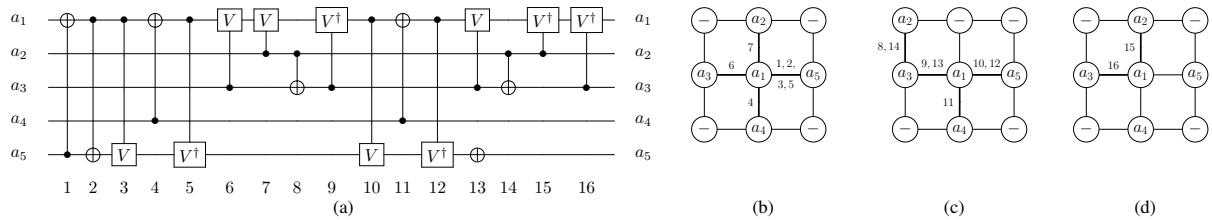

Fig. 4. The result of applying Method 1 on the 4gt13-v1_93 benchmark. Circuit in (a) is the original circuit with one- and two-qubit gates. The initial qubit placement is shown in (b). Since a_1 is adjacent with other qubits, gates in time steps 1-7 can be executed. In time step 8, non-adjacent qubits a_2 and a_3 should interact. A SWAP gate is applied to exchange the locations of qubit a_2 with its neighboring qubit. The result is shown in (c). Then, gates in time steps 8-14 can be executed. To execute the gate in time step 15, we return a_2 to its original location by applying another SWAP gate in (d). The remaining gates can be executed afterwards.

TABLE I

NUMBER OF SWAP GATES AFTER APPLYING PROPOSED METHODS AS WELL AS THE BEST PRIOR RESULT IN 1D ARCHITECTURES. FOR EACH BENCHMARK, THE BEST RESULT IS SHOWN IN BOLD FACE. ON AVERAGE, THE BEST RESULTS OF 1D ARCHITECTURES ARE IMPROVED BY 27%.

Benchmarks				**Method 1**		**Method 2**			Our	Best results in 1D		Imp.
Name	n	# gates	# 2-qubit gates	grid size	#SWAPs	grid size	#SWAPs	SWAP Net	best result	#SWAPs	Ref.	(%)
3_17_13	3	6	13	2x2	6	2x2	6	0%	6	**4**	[9]	-50
4_49_17	4	12	30	2x2	15	2x2	13	0%	13	**12**		-8
4gt10-v1_81	5	6	36	3x2	16	3x2	16	0%	**16**	20		20
4gt11_84	5	3	7	2x3	2	2x3	2	0%	2	**1**		-100
4gt12-v1_89	5	5	52	3x2	21	3x2	19	0%	**19**	35		46
4gt13-v1_93	5	4	16	3x3	2	3x3	2	0%	**2**	6		67
4gt4-v0_80	5	5	43	2x3	17	2x3	17	0%	**17**	34		50
4gt5_75	5	5	22	2x4	9	3x3	8	0%	**8**	12		33
4mod5-v1_23	5	8	24	2x3	11	2x3	11	0%	11	**9**		-22
4mod7-v0_95	5	6	40	3x3	13	3x3	13	0%	**13**	21		38
aj-e11_165	4	13	59	2x3	24	2x3	24	0%	**24**	36		33
alu-v4_36	5	7	31	2x3	10	2x3	10	0%	**10**	18		44
decod24-v3_46	4	9	9	3x2	3	2x2	3	0%	**3**	**3**		0
ham7_104	7	23	87	3x3	48	3x3	50	30%	**48**	68		29
hwb4_52	4	11	23	2x2	9	3x3	9	0%	**9**	10		10
hwb5_55	5	24	106	2x3	48	3x2	45	5%	**45**	63		29
hwb6_58	6	42	146	3x2	94	2x3	79	17%	**79**	118		33
hwb7_62	7	331	2659	3x3	1738	3x3	1688	2%	**1688**	2228		24
hwb8_118	8	633	16608	3x3	11133	3x3	11027	1%	**11027**	14361		23
hwb9_123	9	1959	20405	4x3	15063	4x3	15022	1%	**15022**	21166		29
mod5adder_128	6	15	81	3x2	44	3x2	41	15%	**41**	51		20
mod8-10_177	5	14	108	3x3	45	2x3	48	11%	**45**	72		38
rd32-v0_67	4	2	8	2x3	3	2x3	2	0%	**2**	**2**		0
rd53_135	7	16	78	5x2	39	3x3	42	15%	**39**	66		41
rd73_140	10	20	76	4x3	37	4x4	37	0%	**37**	56		34
sym9_148	10	210	4452	4x4	2363	4x4	2414	2%	**2363**	3415		31
sys6-v0_144	10	15	62	4x4	31	4x4	31	0%	**31**	59		47
urf1_149	9	11554	57770	3x3	38622	3x3	38555	1%	**38555**	44072		13
urf2_152	8	5030	25150	4x2	17175	2x4	16822	1%	**16822**	17670		5
urf5_158	9	10276	51380	3x3	34589	3x3	34406	1%	**34406**	39309		12
QFT5	5	10	10	3x2	5	3x2	7	0%	**5**	6		17
QFT6	6	15	15	5x2	6	2x3	6	0%	**6**	12		50
QFT7	7	21	21	2x4	18	5x2	18	0%	**18**	26		31
QFT8	8	28	28	5x2	25	4x2	18	50%	**18**	33		45
QFT9	9	36	36	3x4	40	3x3	34	0%	**34**	54		37
QFT10	10	45	45	4x3	54	5x3	53	0%	**53**	70		24
cnt3-5_180	16	20	125	3x6	69	4x4	82	30%	**69**	127	[7]	46
cycle10_2_110	12	19	1212	3x4	867	3x4	839	3%	**839**	2304		64
ham15_108	15	70	458	5x3	328	5x3	340	7%	**328**	715		54
plus127mod8192_162	13	910	65455	5x4	53598	5x4	53976	1%	**53598**	151794		65
plus63mod4096_163	12	429	29019	5x3	22118	3x5	22194	1%	**22118**	61556		64
plus63mod8192_164	13	492	37101	5x3	30358	5x3	29835	1%	**29835**	82492		64
rd84_142	15	28	112	5x3	54	5x3	68	28%	**54**	148		64
urf3_155	10	26468	132340	4x3	94017	4x3	94202	1%	**94017**	154672		39
urf6_160	15	10740	53700	5x3	43909	4x4	44394	1%	**43909**	88900		51
Shor3	10	2727	2076	3x5	1737	4x3	1710	1%	**1710**	1816	[8]	6
Shor4	12	6398	5002	3x6	4264	3x4	4339	4%	**4264**	5080		4
Shor5	14	12830	10265	5x4	8456	5x4	10890	2%	**8456**	10760		21
Shor6	16	23139	18885	4x6	20386	4x6	21418	1%	**20386**	20778		2

[14] B.-S. Choi and R. Van Meter. A $\sqrt{n}$-depth quantum adder on the 2D NTC quantum computer architecture. *J. Emerg. Technol. Comput. Syst.*, 8(3):24:1–24:22, August 2012.

[15] M. Saeedi, A. Shafaei, and M. Pedram. Constant-factor optimization of quantum adders on 2D quantum architectures. *Rev. Comp., arXiv:1304.0432*, 2013.

[16] A. G. Fowler, S. J. Devitt, and L. C. L. Hollenberg. Implementation of Shor's algorithm on a linear nearest neighbour qubit array. *Quant. Inf. Comput.*, 4:237–245, 2004.

[17] P. Pham and K. M. Svore. A 2D nearest-neighbor quantum architecture for factoring. *Rev. Comp., arXiv:1207.6655*, 2012.

[18] A. G. Fowler, C. D. Hill, and L. C. L. Hollenberg. Quantum error correction on linear nearest neighbor qubit arrays. *Phys. Rev. A*, 69:042314.1–042314.4, 2004.

[19] S. Kutin, D. Moulton, and L. Smithline. Computation at a distance. *Chicago J. of Theor. Comput. Sci.*, 2007.

[20] D. Maslov, S. M. Falconer, and M. Mosca. Quantum circuit placement. *IEEE Trans. CAD*, 27(4):752–763, Apr 2008.

[21] I. L. Markov, J. Hu, and M.-C. Kim. Progress and challenges in VLSI placement research. *ICCAD'12*, pages 275–282, 2012.

[22] L. Kaufmann and F. Broeckx. An Algorithm for the Quadratic Assignment Problem using Benders Decomposition. *European Journal of Operational Research*, 2(3):204–211, May 1978.

[23] R. E. Burkard, E. ela, P. M. Pardalos, and L. S. Pitsoulis. *The Quadratic Assignment Problem*. Handbook of Combinatorial Optimization, Kluwer Academic Publishers,1998, pp. 241-338..

[24] B. Parhami. *Introduction to Parallel Processing: Algorithms and Architectures*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[25] Gurobi Optimization Inc. Gurobi optimizer reference manual, 2013.

[26] D. Maslov, G. W. Dueck, D. M. Miller, and C. Negrevergne. Quantum circuit simplification and level compaction. *IEEE Trans. CAD*, 27(3):436–444, March 2008.