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.
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 reversing 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.

Thus, 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 are 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:

subject to

\[
\begin{align*}
\sum_{j=1}^{n} x_{ij} &= 1, & i = 1, \ldots, n, & (2) \\
\sum_{i=1}^{n} x_{ij} &= 1, & j = 1, \ldots, n, & (3) \\
x_{ij} &\in \{0, 1\}, & i, j = 1, \ldots, n. & (4)
\end{align*}
\]

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):

\[
\min \sum_{i=1}^{n} \sum_{j=1}^{n} z_{ij} \quad (5)
\]

subject to

\[
\begin{align*}
(2), (3), (4), \\
\alpha_{ij} x_{ij} + \sum_{k=1}^{n} \sum_{l=1}^{n} c_{ijkl} x_{kl} - z_{ij} \leq \alpha_{ij}, & i, j = 1, \ldots, n, & (6) \\
z_{ij} \geq 0, & i, j = 1, \ldots, n, & (7)
\end{align*}
\]

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 \( (x_{ij} \text{'s}) \), \( n^2 \) real variables \( (z_{ij} \text{'s}) \), 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 \( (j - 1) \times N/m + 1 \) and \( j \times N/m \). 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 \( (x+y)^\lfloor x+y \rfloor \). 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 qubit 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.

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


Fig. 4. The result of applying Method 1 on the 4q13-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 \(a_2\) 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Best result</th>
<th>Swap gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method 1</td>
<td>100%</td>
<td>5</td>
</tr>
<tr>
<td>Method 2</td>
<td>95%</td>
<td>5</td>
</tr>
<tr>
<td>Method 3</td>
<td>90%</td>
<td>5</td>
</tr>
</tbody>
</table>

**Fig. 5.** The result of applying Method 2 on the 4q13-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 \(a_2\) 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.

**Fig. 6.** The result of applying Method 3 on the 4q13-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 \(a_2\) 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.

**Fig. 7.** The result of applying Method 4 on the 4q13-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 \(a_2\) 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.

**Fig. 8.** The result of applying Method 5 on the 4q13-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 \(a_2\) 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.

**Fig. 9.** The result of applying Method 6 on the 4q13-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 \(a_2\) 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.