An Efficient Quantum Circuits Optimizing Scheme Compared with QISKit (Short Paper)

Xin Zhang[1], Hong Xiang[1,2(✉)], and Tao Xiang[3]

[1] School of Big Data and Software Engineering, Chongqing University, Chongqing, China
xianghong@cqu.edu.cn
[2] Key Laboratory of Dependable Service Computing in Cyber Physical Society, Chongqing University, Ministry of Education, Chongqing, China
[3] School of Computer Science, Chongqing University, Chongqing, China

Abstract. Recently, the development of quantum chips has made great progress – the number of qubits is increasing and the fidelity is getting higher. However, qubits of these chips are not always fully connected, which sets additional barriers for implementing quantum algorithms and programming quantum programs. In this paper, we introduce a general circuit optimizing scheme, which can efficiently adjust and optimize quantum circuits according to arbitrary given qubits' layout by adding additional quantum gates, exchanging qubits and merging single-qubit gates. Compared with the optimizing algorithm of IBM's QISKit, the quantum gates consumed by our scheme is 74.7%, and the execution time is only 12.9% on average.

Keywords: Quantum computing · Quantum circuit · Circuit optimizing

1 Introduction

Quantum computing has attracted increasing attention because of its tremendous computing power [7,11,12] in recent years. There are more and more companies and scientific research institutions who devote themselves to developing quantum chips with more qubits and higher fidelity. While most theoretical studies assume that interactions between arbitrary pairs of qubits are available, almost all these realistic chips have certain constraints on qubit connectivity [6,8]. For example, IBM's 5-qubit superconducting chips *Tenerife* and *Yorktown* [1] adopt neighboring connectivity. [14] uses a 4-qubit superconducting chip, in which four qubits are not directly connected, but are connected by a central resonator. That is, the layout of this chip is central. In addition, CAS-Alibaba Quantum Laboratory's 11-qubit superconducting chip [4] and Tsinghua University's 4-qubit NMR chip [13] both reduce the fully connectivity to the linear

H. Gao et al. (Eds.): CollaborateCom 2018, LNICST 268, pp. 467–476, 2019.
https://doi.org/10.1007/978-3-030-12981-1_32

nearest-neighbor connectivity. Distinctly, this non-fully connected connection sets additional barriers for implementing quantum algorithms and programming quantum programs.

As early as 2007, Cheung et al. made a discussion about the non-fully connected physical layout [6]. By adding SWAP gates, they turned illegal CNOT operations into legitimate operations and proved that the star-shaped or the linear nearest-neighbor connectivity could be able to utilize additional $O(n)$ quantum gates to complete the adjustment, where n stands for the number of qubits. In 2017, IBM developed a quantum information science kit, namely QISKit [3], which contains an algorithm that can adjust and optimize quantum programs according to any layout. Recently, in order to find more efficient solutions, IBM organized the QISKit Developer Challenge [2].

The paper is organized as follows: Sect. 2 briefly introduce the necessary conceptions. In Sect. 3, the design concept of our optimizing scheme is presented in detail. We next compare the cost and efficiency of our scheme with QISKit's optimizing method in Sect. 4. The conclusion and future research can be found in Sect. 5.

2 Common Solutions

Before introducing the common solutions, we need to point out the main obstacles for hindering the execution of quantum programs:

- Obstacle-1: the direction of CNOT gate is illegal, as shown in the red line in Fig. 1(a);
- Obstacle-2: the connectivity between two specific qubits is illegal, as shown in the blue line.

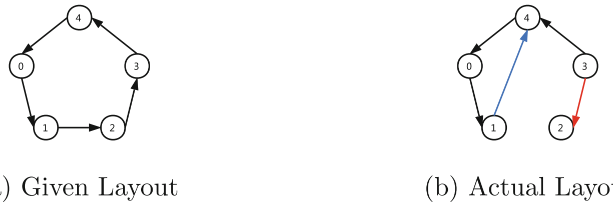


Fig. 1. An example of Obstacle-1 and Obstacle-2. (Color figure online)

For Obstacle-1, a common solution is to flip the direction by 4 additional H gates:

$$H_2 \times \text{CNOT}_{(q_1,\ q_2)} \times H_2 = \text{CNOT}_{(q_2,\ q_1)}. \qquad (1)$$

As for Obstacle-2, the basic idea is exchanging the states of qubits by SWAP gates. For example, although $cnot(q_1, q_4)$ is illegal in Fig. 1(a), we can use another way to accomplish the same task, such as the circuit shown in Fig. 2.

However, the additional overhead of this solution is costly, especially for sparse physical layouts. Specifically,

$$cost = 2m \times cost_{\mathrm{SWAP}}, \tag{2}$$

where m stands for the number of intermediate nodes on the shortest path between the control-qubit and the target-qubit, $cost_{\mathrm{SWAP}}$ stands for 3 CNOT gates and 4 H gates.

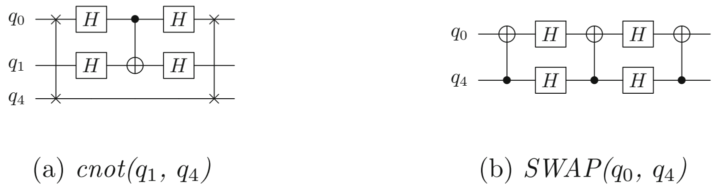


Fig. 2. Equivalent circuits of cnot(q_1, q_4) and $SWAP(q_0, q_4)$.

3 Our Optimizing Scheme

Our optimizing scheme is an efficient general solution. Specifically, we design the following three steps to adjust and optimize quantum programs.

3.1 The Global Adjustment of Qubits

This step can be described as Algorithm 1. In Algorithm 1, we extract all CNOT gates from the quantum program separately and traverse them from front to back. Once encountering an illegal CNOT gate, we try to find an available qubits' mapping to adjust the whole Open-QASM code without converting the traversed CNOT gates illegal. At each adjustment, we have $(d_{cq} \times d_{tq} - t)$ available mappings to choose, where t stands for the number of mappings which make some traversed CNOT gates illegal, d_{cq} and d_{tq} stand for the number of adjacent qubits of control-qubit and target-qubit in the given layout, respectively. The traversal terminates when there is no illegal CNOT gate or $(d_{cq} \times d_{tq} - t) = 0$.

Suppose that there are M possible mappings, where M is related to the given layout and the connectivity of quantum programs. At this point, we need to estimate the cost of solving Obstacle-2 in the program adjusted according to these $(M + 1)$ mappings (M mappings and one empty mapping) respectively. Then take the smallest one as the global adjustment mapping. The reason for estimation, rather than accurate calculation, and the estimation process are explained in the next part. Finally, we adjust the qubits of the original Open-QASM code according to the global mapping. As for the classical register, which stores the results of the measurement, does not need to be modified. For example, $cnot(q_1,\ q_4)$ is illegal in Fig. 1 and it can be adjusted by the global mapping $\{1 : 3,\ 3 : 1\}$, as shown in Fig. 3.

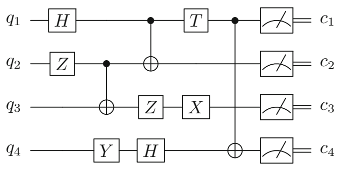
Algorithm 1. Global Adjustment

```
Input: The set of CNOT in QP, $C$; the set of legal CNOT, $A$; the record of all
       possible costs, $costs$; the record of all possible mappings, $maps$; the
       current mapping, $amap$;
Output: The mapping of qubits' ID, $map$
1  GlobalAdjust($costs, maps, amap$)
2      $costs \leftarrow [\,], maps \leftarrow [\,]$ and $amap \leftarrow [\,]$;
3      Adjust($C$, $A$, $amap$, $costs$, $maps$);
4      $i \leftarrow$getIndexofMinValue($costs$);
5      return $maps[i]$;
6
7  Adjust($C, A, amap, costs, maps$)
8      $alternativeMap \leftarrow [\,]$;
9      for CNOT $c$ in $C$ do
10         if $c$ not in $A$ then
11             $cq \leftarrow c[0]$ and $tq \leftarrow c[1]$;
12             $cqAdj \leftarrow$ getAdjacentQubit($cq$) and $tqAdj \leftarrow$ getAdjacentQubit($tq$);
13             $tMaps \leftarrow \{cq : tqAdj,\ tq : cqAdj\}$;
14             for map $m$ in $tMaps$ do
15                 $tempC \leftarrow C$;
16                 change qubits' ID in $tempC$ according to $m$;
17                 if no illegal CNOT in $tempC$ then
18                     add $m$ to $alternativeMap$;
19             break;
20     if $alternativeMap == [\,]$ then
21         $cost \leftarrow$ estimateCost();
22         add $cost$ to $costs$ and add $amap$ to $maps$;
23     for map $am$ in $alternativeMap$ do
24         $tempC \leftarrow C$ and add $am$ to $amap$;
25         change qubits' ID in $tempC$ according to $am$;
26         if no illegal CNOT in $tempC$ then
27             add $amap$ to $maps$ and add 0 to $costs$;
28         else
29             Adjust($C$, $A$, $amap$, $costs$, $maps$);
```

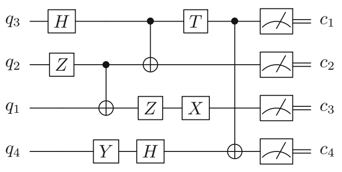
3.2 The Local Adjustment of Qubits

Compared with the basic solution described in Sect. 2, our scheme has the following differences:

- There is no need to use SWAP gates again to restore the state.
- The effect of exchanging control-qubit or target-qubit with intermediate qubits by SWAP gates is completely different for the subsequent code.



(a) Before adjusting



(b) After adjusting

Fig. 3. Adjust the circuit according to $\{1:3,\ 3:1\}$ and (b) can be executed on Fig. 1(a)

However, it is difficult to accurately calculate the costs of these two cases in the second difference. During the calculation, we will encounter several illegal CNOT gates, and for each illegal CNOT, we have two solutions. Actually, the solution space is a binary tree whose height is n and the number of leaf nodes is approximately $O(2^n)$, where n stands for the number of illegal CNOT gates. Obviously, we have to estimate the cost by greedy ideas. With the increase in the scale of programs, the manifestation of this greedy choice is more obvious, which can be seen in Sect. 4.

The cost of adjusting the Open-QASM code *qasm* is estimated by

$$cost_{qasm} = \sum_{i=1}^{n}[(\frac{n-i}{n})^2 \cdot m_i \cdot cost_{\text{SWAP}}], \tag{3}$$

where m_i stands for the number of intermediate qubits between the control-qubit and the target-qubit of the ith illegal CNOT. Among the various estimation formulas we tried, the result obtained by Eq. (3) is optimal. The reason for adding the correction factor $(\frac{n-i}{n})^2$ in Eq. (3) is that the later the CNOT gate is executed, the easier it is influenced by the previous adjustments. That is, estimation is not reliable for the later CNOT gates. Multiplying the factor, which will continue to decrease as the estimation progress, with the estimation results can have a certain correction effect.

For improving the accuracy of estimation, we accurately calculate the top 4 layers of the binary tree, and estimate the cost of the subsequent gates of the 2^4 cases respectively, where 4 is the optimal value determined after repeated trials. Then add the estimated result and the calculated result together and choose the smallest one among the 16 cases as our choice.

Specifically, we traverse the Open-QASM code. Whenever encountering an illegal CNOT, we call Algorithm 2 to adjust it and then update the subsequent code and the classical register until the traversal terminates. It can be seen from Algorithm 2 that the mapping generated by *Adjust* function only affects the subsequent code of *illC* and that is why we call this step *Local adjustment*.

At this point, there is no Obstacle-2 in quantum programs. Then we traverse the new Open-QASM code again to handle Obstacle-1 by Eq. (1).

Algorithm 2. Local Adjustment

```
Input: The Open-QASM code of the quantum program, $qasm$; the first illegal
       CNOT, $illC$; the rest CNOTs after $illC$ in $qasm$, $Cs$; the record of all
       possible costs, $costs$; the cost in the current case, $cost$; the record of all
       possible mappings, $maps$; the current mapping, $amap$; the depth of
       recursion, $d$
Output: The adjusted Open-QASM code, $qasm$
1  LocalAdjust($qasm$, $illC$, $Cs$)
2      $cost \leftarrow 0$, $costs \leftarrow$[ ], $amap \leftarrow$[ ], $d \leftarrow 1$ and $maps \leftarrow$[ ];
3      Adjust($illC$, $Cs$, $cost$, $costs$, $amap$, $maps$, $d$);
4      $i \leftarrow$getIndexofMinValue($costs$);
5      add SWAP gates to $qasm$ according to $maps[i]$;
6      change qubits'ID in $qasm$ according to $maps[i]$;
7      return $qasm$;
8
9  Adjust($illC$, $Cs$, $cost$, $costs$, $map$, $maps$, $d$)
10     $interQs \leftarrow$ getIntermediateNode($illC[0]$, $illC[1]$);
11     $cost \leftarrow cost + 34\times interQs$.length;
12     for qubit $q$ in $illC$ do
13         $tc \leftarrow cost$;
14         if $q$ is control-qubit then
15             $tc \leftarrow tc + 4$;
16         $tMap \leftarrow$ constructMapBetweenQ($interQs$,$q$);
17         change qubits' ID in $cnots$ according to $tMap$;
18         $nIllC \leftarrow$ getFirstIllegalCnot($cnots$);
19         $restC \leftarrow$ getAllCnotAfterNewIllC($cnots$);
20         if $map$ != [ ] then
21             $tMap \leftarrow map$;
22         if $nIllC$ == None then
23             add $tc$ to $costs$ and $tMap$ to $maps$;
24         else if $d == 4$ then
25             $tc \leftarrow tc +$ estimateCost();
26             add $tc$ to $costs$ and add $tMap$ to $maps$;
27         else
28             Adjust($nIllC$, $restC$, $tc$, $costs$, $tMap$, $maps$, $d + 1$);
```

3.3 The Mergence of Single-Qubit Gates

In this step, we will reduce the circuit depth by merging single-qubit gates. At first, we need to determine which kind of single-qubit gates can be merged.

The random quantum circuit shown in Fig. 4(a) contains three CNOT gates and these gates divide the execution processes of q_0, q_1, q_2 into three parts respectively. Obviously, single-qubit gates in these parts can be merged and we can reduce Fig. 4(a) to (b). Based on this example, we can draw a conclusion that for any qubit q, the n multi-qubit gates with q involved can divide the execution process of q into $n+1$ subintervals and the single-qubit gates in each subintervals can be merged into one gate.

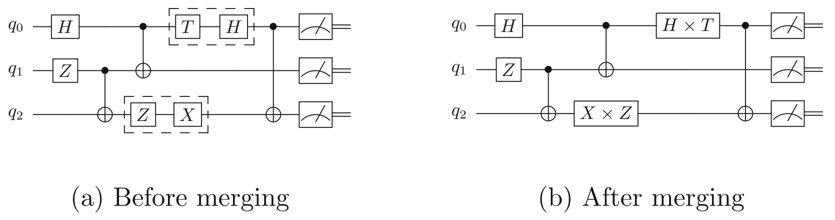


Fig. 4. The change of a quantum random circuit before and after merging single-qubit gates.

As mentioned before, all single-qubit gates in Open-QASM belong to $\{u1, u2, u3\}$. Therefore, merging single-qubit gates actually contains 9 different cases: $u1 \times u1$, $u1 \times u2$, $u1 \times u3$, $u2 \times u1$, $u3 \times u1$, $u2 \times u2$, $u3 \times u2$, $u2 \times u3$ and $u3 \times u3$. In order to handle these cases, we need to do **Z-Y decompositions** [9] for $u1$, $u2$ and $u3$. For the first five cases, we can directly merge them by $R_z(\lambda) \times R_z(\phi) = R_z(\lambda + \phi)$ [5]. As for the last four cases, we have:

$$\begin{aligned}
&R_z(\phi_1) \cdot R_y(\theta_1) \cdot R_z(\lambda_1) \cdot R_z(\phi_2) \cdot R_y(\theta_2) \cdot R_z(\lambda_2) \\
&= R_z(\phi_1) \cdot [R_y(\theta_1) \cdot R_z(\lambda_1 + \phi_2) \cdot R_y(\theta_2)] \cdot R_z(\lambda_2) \\
&= R_z(\phi_1) \cdot [R_z(\alpha) \cdot R_y(\beta) \cdot R_z(\gamma)] \cdot R_z(\lambda_2) \\
&= R_z(\phi_1 + \alpha) \cdot R_y(\beta) \cdot R_z(\gamma + \lambda_2) \\
&= u3(\beta, \phi_1 + \alpha, \gamma + \lambda_2).
\end{aligned} \tag{4}$$

The key of this kind of merging lies in how to transform the **Y-Z decomposition** of a quantum gate to the **Z-Y decomposition**. And we use QISKit's merge method proposed in [10] to solve this problem.

4 Numerical Results

In this section, we use the method proposed in the QISKit Developer Challenge to count the cost of gates:

$$cost = n_2 \times 10 + n_1 \times 1, \tag{5}$$

where n_2 and n_1 stand for the number of CNOT gates and single-qubit gates in optimized quantum circuit, respectively.

The experiments are designed as follow: for the 14 cases of qubits number from 3 to 16, we generate 10 different random quantum circuits respectively for 16 cases with circuit depth from 1 to 16 respectively. That means, in total,

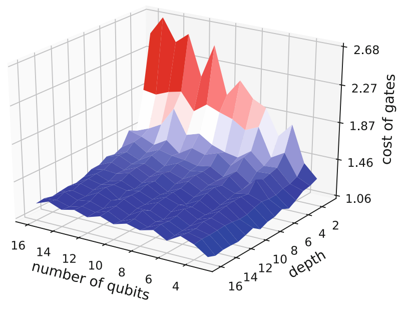
$14 \times 16 \times 10 = 2240$ circuits are generated. Then we chose four common connected graphs (linear, central, neighboring and circular) and use our optimizing scheme and QISKit's algorithm to adjust and optimize these 2240 random circuits according to these layouts, respectively. That is, each algorithm handles 8960 (2240×4) quantum circuits. Finally, the optimized quantum programs are executed by QASM-simulator. If the result of our scheme is consistent with QISKit's result, we count the cost and the execution time of each circuit.

Comparison with QISKit's Optimizing Method
Table 1 shows the quantum gates consumption of the 2240 original random quantum circuits, and the average cost of gates and compiler time required to adjust and optimize these 2240 circuits by our scheme and QISKit. Obviously, the quantum gates consumed by our scheme is 74.7% of QISKit, and the execution time is only 12.9%.

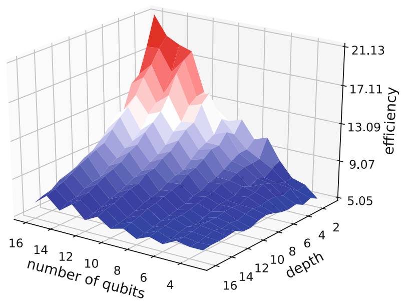
Table 1. The overall statistical

	Time (s)	Gate cost
Original circuit	0	3084391
Our scheme	16472.48	6703061
QISKit	127751.99	8974717

Specifically, the performance of our scheme varies for different scales of quantum circuits. Figure 5(a) and (b) illustrate the ratio of QISKit and our scheme about the cost of quantum gates and efficiency with various qubits q and circuit depths d, respectively. The two formulas are shown as follows:



(a) Gate Cost



(b) Efficiency

Fig. 5. Experimental results

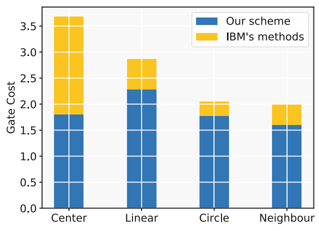
$$\text{cost}_{(n,d)} = \frac{qc_{(n,d)}}{c_{(n,d)}}, \quad \text{efficiency}_{(n,d)} = \frac{qt_{(n,d)}}{t_{(n,d)}}, \tag{6}$$

where qc and qt stand for the gate cost and execution time of QISKit's algorithm, and c and t indicate those of our method. Figure 6 shows that in all cases we executed, our algorithm can use fewer quantum gates to adjust and optimize the original circuits in less time. In the worst case (more qubits and more circuit depth), we can use 6% less gates and the efficiency is about 5 times; in optimal case (more qubits and less circuit depth), we can use 63% less gates and the efficiency is about 20 times.

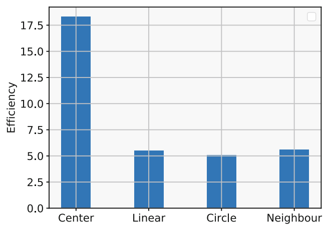
Performance in Different Physical Layouts
For the four layouts we have chosen, there are also significant differences in costs of quantum gates and execution time. In order to deal with different scales of circuits in a fair manner and avoid the statistical result being dominated by large-scale circuits, we no longer directly sum up the gate costs in different cases (as used in Table 1). Specifically, the statistical method is as follows:

$$\text{cost}_{l,c} = \frac{1}{2240}[\sum_{i=1}^{2240}(\frac{c_i}{o_i})], \quad \text{efficiency}_l = \frac{1}{2240}[\sum_{i=1}^{2240}(\frac{qt_i}{ot_i})]. \tag{7}$$

where $l \in \{Linear, Circle, Center, Neighbour\}$, $c \in \{oc, qc\}$, o_i, qc_i and oc_i stand for the gate cost of the ith original circuit, the ith circuit adjusted by QISKit and our scheme respectively, qt_i and ot_i stand for the time required to compile the ith circuit by QISKit and our scheme respectively.



(a) Costs of four layouts



(b) Efficiency of four layouts

Fig. 6. Experimental results of four different layouts

Figure 6(a) shows that for the *central* layout, our scheme requires 1.80 times the gate consumption of the original circuit, and the optimizing method of QISKit requires 3.68 times; for the *linear* layout, the gate cost of our scheme is 2.28 times as many as the original cost and the cost of QISKit is about 2.86 times; as for the *circle* and *neighbour* layouts, our scheme need to use 1.77 times and 1.60 times the gate cost respectively, while QISKit's method need 2.05 times and 2.01 times. Figure 6(b) illustrates that for the four different layouts, our scheme is at least 4 times faster than QISKit; especially for *central* layout, the efficiency is about 17.3 times as fast as QISKit's method.

5 Conclusions

Considering the cost of physical implement, layouts of most existing quantum chips are not fully connected, which sets additional barriers for implementing quantum algorithms and programming quantum programs. We propose a general optimizing scheme to accomplish the task by adding additional logic gates, exchanging qubits in the quantum register and merging single-qubit gates. Compared with QISKit's optimizing method, the quantum gates consumed by our scheme is 74.7% and the execution time is only 12.9% overall. For circuits with more qubits and less circuit depth, this advantage is more obvious. In addition, several common connected graphs (linear, central, neighboring and circular) are compared as well. In these four cases, our scheme has advantages. Especially for the central layout, we can use only 49% gates and 5.8% execution time of QISKit's optimizing algorithm to adjust and optimize the original quantum circuits.

Acknowledgments. The work is supported by National Key R&D Program of China (NO. 2017YFB0802000).

References

1. The backend information of IBM quantum cloud. https://github.com/QISKit/qiskit-backend-information/
2. QISKit developer challenge. https://qx-awards.mybluemix.net/
3. QISKit Python API. https://qiskit.org/
4. The url of alibaba's quantum cloud platform. http://quantumcomputer.ac.cn/index.html
5. Barenco, A., et al.: Elementary gates for quantum computation. Phys. Rev. A **52**(5), 3457 (1995)
6. Cheung, D., Maslov, D., Severini, S.: Translation techniques between quantum circuit architectures. AAPT (2007)
7. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 212–219. ACM (1996)
8. Linke, N.M., et al.: Experimental comparison of two quantum computing architectures. In: Proceedings of the National Academy of Sciences, p. 201618020 (2017)
9. Nielsen, M.A., Chuang, I.: Quantum Computation and Quantum Information (2002)
10. QISKit: The code of merging two u3 gates. https://github.com/QISKit/qiskit-sdk-py/blob/master/qiskit/mapper/_mapping.py
11. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev. **41**(2), 303–332 (1999)
12. Simon, D.R.: On the power of quantum computation. SIAM J. Comput. **26**(5), 1474–1483 (1997)
13. Xin, T., et al.: NMRCloudQ: a quantum cloud experience on a nuclear magnetic resonance quantum computer. Sci. Bull. **63**, 17–23 (2017)
14. Zhong, Y., et al.: Emulating anyonic fractional statistical behavior in a superconducting quantum circuit. Phys. Rev. Lett. **117**(11), 110501 (2016)