PAPER

Mapping a Quantum Circuit to 2D Nearest Neighbor Architecture by Changing the Gate Order

Wakaki HATTORI[†a)], *Nonmember and* **Shigeru YAMASHITA**[††b)], *Senior Member*

SUMMARY This paper proposes a new approach to optimize the number of necessary SWAP gates when we perform a quantum circuit on a two-dimensional (2D) NNA. Our new idea is to change the order of quantum gates (if possible) so that each sub-circuit has only gates performing on adjacent qubits. For each sub-circuit, we utilize a SAT solver to find the best qubit placement such that the sub-circuit has only gates on adjacent qubits. Each sub-circuit may have a different qubit placement such that we do not need SWAP gates for the sub-circuit. Thus, we insert SWAP gates between two sub-circuits to change the qubit placement which is desirable for the following sub-circuit. To reduce the number of such SWAP gates between two sub-circuits, we utilize A* algorithm.

key words: *Nearest Neighbor Architecture (NNA), gate order*

1. Introduction

After the seminal papers by Shor [1] and Grover [2], there have been intensive researches for quantum computations. To realize general-purpose quantum computers, one of the major challenges is to find an efficient method to design *fault-tolerant* quantum circuits [3] in order to overcome the *decoherence* problem. When we perform an operation between distant two qubits, the error due to decoherence would occur frequently. Therefore, it has been considered to perform quantum circuits on an NNA (Nearest Neighbor Architecture) [4] where operations only on adjacent qubits are allowed.

To perform arbitrary quantum circuits on an NNA, we need to insert SWAP gates so that the two qubits related to each gate become adjacent. (Note that we assume quantum circuits consisting of only two-qubit gates like most of the previous works.) To reduce the number of inserted SWAP gates, there have been many optimization methods proposed; some methods consider the initial qubit placement, whereas other methods consider how SWAP gates are inserted.

Indeed there have been researches to develop design methods considering various kinds of NNAs, i.e., for one-dimensional (1D) [4]–[11], two-dimensional (2D) [12]–[15] and three-dimensional (3D) [16] architectures. As a most general model, some researches consider an arbitrary graph where each vertex corresponds to a qubit, and allow an operation only on the adjacent two vertices in the graph [17]–[19].

Recently, 2D architectures have been studied the most intensively because they have more adjacent qubits compared to 1D architectures, and should be much easier to be implemented than 3D ones. For 2D NNAs, PAQCS (Physical Design-Aware Fault-Tolerant Quantum Circuit Synthesis) [20] is a good heuristic methodology to reduce the inserted SWAP gates. To reduce the necessary inserted SWAP gates, PAQCS considers mainly two issues. First, it finds possibly good initial qubit placement based on a graph generated from a each given quantum circuit. Next it finds possibly a good way to "move" (the contents of) qubits in order to make the two qubits related to each gate adjacent.

In the above-mentioned process, PAQCS assumes the gate order is fixed from a given one; it does not consider what is a possibly good gate order to reduce the inserted SWAP gates. Note that almost all previous works for NNAs do not consider the gate order.

Considering the above situation, this paper seeks a new approach to optimize the number of necessary SWAP gates when we map a quantum circuit to a 2D NNA. Our new idea is to change the order of quantum gates (if possible) so that we can decrease the number of sub-circuits which has only gates performing on adjacent qubits. For each sub-circuit, we utilize a SAT solver to find the *best* qubit placement such that the sub-circuit has only gates on adjacent qubits in a 2D architecture. This contrasts with PAQCS which find a qubit placement heuristically.

Each sub-circuit may have a different qubit placement such that we do not need SWAP gates for the sub-circuit. Thus, we insert SWAP gates between two sub-circuits to change the qubit placement which is desirable for the following sub-circuit. To reduce the number of such SWAP gates between two sub-circuits, we utilize A* algorithm.

We confirmed that the above-mentioned new approach has a potential to reduce the number of necessary SWAP gates compared with the approach used in PAQCS. Note that we consider a regular 2D architecture in this paper, but our framework can be easily extended to any architecture.

This paper is organized as follows. We review previous design methods for 2D NNAs in Sect. 2. After that, in Sect. 3 we propose our design method, and explain how we can construct a sub-circuit for 2D NNAs, and how we can find a good sequence of inserting SWAP gates in our

Manuscript received December 24, 2018.
Manuscript revised June 13, 2019.
Manuscript publicized July 25, 2019.

[†]The author is with the Graduate School of Information Science and Engineering, Ritsumeikan University, Kusatsu-shi, 525–8577 Japan.
[††]The author is with the College of Information Science and Engineering, Ritsumeikan University, Kusatsu-shi, 525–8577 Japan.
a) E-mail: doyle@ngc.is.ritsumei.ac.jp
b) E-mail: ger@cs.ritsumei.ac.jp

DOI: 10.1587/transinf.2018EDP7439

q_0	q_1	q_2
q_3	q_4	q_5
q_6	q_7	q_8

Fig. 1 An example of qubit placement on a 2D grid architecture.

q_0	q_4	q_2
q_3	q_1	q_5
q_6	q_7	q_8

Fig. 2 qubit placement after an operation $S(q_1, q_4)$ is performed on qubit placement in Fig. 1.

method. We provide some preliminary experimental results in Sect. 4 to show the potential of our idea, i.e., to change the order of gates. Finally, Sect. 5 concludes the paper with our future works.

2. Nearest Neighbor Architectures

In a 2D grid architecture, qubits are placed on a 2D grid as shown in Fig. 1. A qubit has four neighboring qubits at most. For example, in Fig. 1, a qubit q_4 has four neighboring qubits which are q_1,q_3,q_5 and q_7.

When an operation is performed on distant qubits such as q_0 and q_4 in Fig. 1, the decoherence error is more likely to occur. On the other hand, it is expected to reduce the decoherence error by performing a quantum circuit on an NNA. Therefore, to perform a quantum circuit on an NNA, SWAP gates are inserted to swap quantum states, so that a control bit and a target bit are adjacent with each other when we perform an operation on distant qubits. In this paper, $S(q_i, q_j)$ means a SWAP gate between q_i and q_j. $C(q_i, q_j)$ means a CNOT gate between q_i and q_j. q_i and q_j of $C(q_i, q_j)$ mean a control bit and a target bit of $C(q_i, q_j)$ respectively.

When an operation $S(q_1, q_4)$ is performed on the qubit placement as shown in Fig. 1, the qubit placement is changed to one as shown in Fig. 2. Since q_0 and q_4 are adjacent on the qubit placement in Fig. 2, $C(q_0, q_4)$ is performed on adjacent qubits. Note that we do not change the qubit placement *physically* when we perform SWAP gates; only the quantum states of two qubits are swapped when we apply a SWAP gate.

When the initial qubit placement of a quantum circuit in Fig. 3 is one as shown in Fig. 1, for example, we can get circuits as shown in Fig. 4 and Fig. 5 after SWAP gates are inserted. The quantum states of qubits change by inserting SWAP gates, so the output of the quantum circuit will be different from the original one. Thus SWAP gates need to be inserted again to restore the output after all operations. The number of SWAP gates is 10 in Fig. 4, and the number of SWAP gates is 6 in Fig. 5. As these examples show,

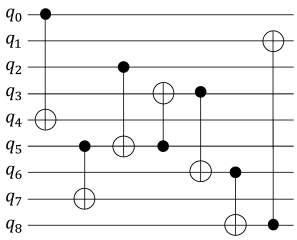

Fig. 3 An example of quantum circuit for explaining insertion of SWAP gates.

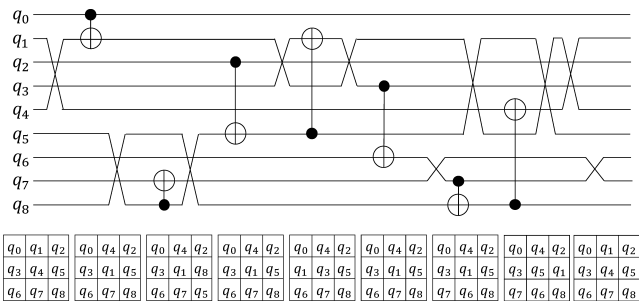

Fig. 4 An example of a quantum circuit on an NNA that 10 SWAP gates are inserted into a quantum circuit in Fig. 3.

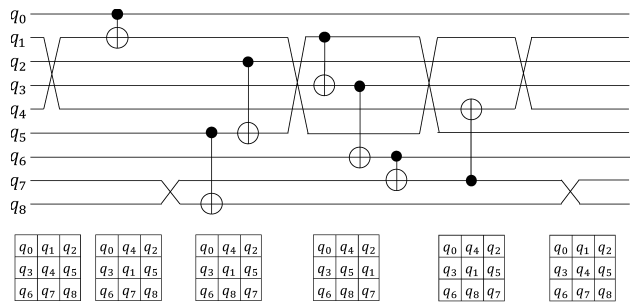

Fig. 5 An example of a quantum circuit on an NNA that 6 SWAP gates are inserted into a quantum circuit in Fig. 3.

the way of inserting SWAP gates affects the total number of necessary SWAP gates in order to map a quantum circuit to one on an NNA.

3. The Proposed Method

We divide a given quantum circuit into sub-circuits such that all operations in the sub-circuits can be performed without inserting SWAP gates in consideration of changing the gate order. In our proposed method, a SAT solver is used to determine if there exists such sub-circuits, and to construct sub-circuits. While constructing sub-circuits, the gate order is considered to construct sub-circuits so that they include more gates. After dividing a given quantum circuit into several sub-circuits, SWAP gates are inserted between two sub-circuit to change the qubit placement to the appropriate qubit placement for each sub-circuit. We employ A* algorithm to find how to insert SWAP gates to change the qubit placement. A* algorithm is a major searching method

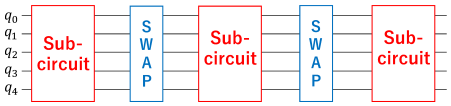

Fig. 6 An outline of the generated circuit by the proposed method.

and it is also used to map a quantum circuit to an NNA [17].

The overall flow of the proposed method is as shown in *Algorithm*1 and Fig. 6 illustrates an outline of the generated circuits by the proposed method. Details are explained in the following sections.

Algorithm 1 Algorithm to divide a given quantum circuit into sub-circuits and to insert SWAP gates between two sub-circuits

```
1: while there exists a quantum gate that is not added to a sub-circuit do
2:     Construct a gate dependency graph of quantum gates that are not added to sub-circuits
3:     Use a SAT solver for a sub-circuit that includes all quantum gates in the gate dependency graph
4:     if UNSAT then
5:         Fail ⇐ the number of quantum gates that are not added to sub-circuits
6:         Success ⇐ 0
7:         while Success − Fail > 1 do
8:             while there exists a sub-circuit that contains ⌈(Success + Fail)/2⌉ quantum gates which is not used for SAT solver do
9:                 Use a SAT solver for sub-circuits that contain ⌈(Success + Fail)/2⌉ quantum gates which is not used for SAT solver
10:                if SAT then
11:                    Success ⇐ ⌈(Success + Fail)/2⌉
12:                    break
13:                else
14:                    if we have already checked all the possible sub-circuits having ⌈(Success + Fail)/2⌉ quantum gates by a SAT solver then
15:                        Fail ⇐ ⌈(Success + Fail)/2⌉
16:                    end if
17:                end if
18:            end while
19:        end while
20:    end if
21: end while
22: Insert SWAP gates between two sub-circuit by using A* algorithm
```

3.1 Constructing Sub-Circuits in Consideration of Changing the Gate Order

A gate dependency graph is used to construct sub-circuits of a quantum circuit in consideration of changing the gate order. A gate dependency graph is a directed graph that shows the dependency of quantum gates in a quantum circuit. When quantum gates are not commutative, we define that there is dependency between those quantum gates.

A gate dependency graph of a quantum circuit in Fig. 7 is as shown in Fig. 8. In the quantum circuit in Fig. 7, a target bit of C_1 is the same as a control bit of C_2. Thus these quantum gates are not commutative, and C_2 must be performed after performing C_1. In the gate dependency graph

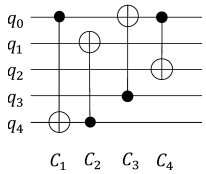

Fig. 7 A quantum circuit that has dependency between quantum gates.

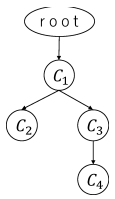

Fig. 8 A gate dependency graph of Fig. 7.

as shown in Fig. 8, there is a directed edge from node C_1 to node C_2. This means that C_1 and C_2 are not commutative, and C_2 must be performed after performing C_1. There is no path from C_2 to C_3 (or vice versa) in Fig. 8, and thus we can change the gate order of C_2 and C_3. While constructing sub-circuits, we consider the gate order by using a gate dependency graph to construct sub-circuits that includes more gates.

Let us show an example by using a circuit as shown in Fig. 9 and its gate dependency graph as shown in Fig. 10. In the following, a sub-circuit is denoted by S_i, and S_i is a set of quantum gates. We first consider $S_1 = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$ as a target sub-circuit of the quantum circuit in Fig. 9 that includes the largest number of quantum gates. When a SAT solver is used for S_1 to find a qubit placement such that all the operations in S_1 can be performed on an NNA architecture, the SAT solver returns that such a qubit placement does not exist. Accordingly, we try a new sub-circuit, S_2, which includes half number of quantum gates as S_1; S_2 includes 4 quantum gates.

When constructing S_2, it is necessary to select quantum gates from the root of the gate dependency graph in Fig. 10 in order to keep the dependency of quantum gates. Considering the above, we consider $S_2 = \{C_1, C_2, C_3, C_4\}$. When a SAT solver is used for S_2 to find a qubit placement that all operations can be performed on an NNA architecture, this time the SAT solver returns that there exists such a qubit placement. Because we want to find a sub-circuit that includes more quantum gates (if there is), we try another new sub-circuit, S_3, which contains $(|S_1| + |S_2|)/2 = 6$ quantum gates. This is because we already know that there is a desirable sub-circuit having $|S_2|$ gates, and also there is no such a sub-circuit having $|S_1|$ gates, and thus we try to find a circuit having the average number of $|S_1|$ and $|S_2|$ quantum gates; this is a standard binary-search technique.

Thus, by considering the dependency of quantum gates, we try $S_3 = \{C_1, C_2, C_3, C_4, C_5, C_6\}$ after S_2. When a SAT solver is used for S_3 to find a qubit placement that all operations can be performed on a NNA architecture, the

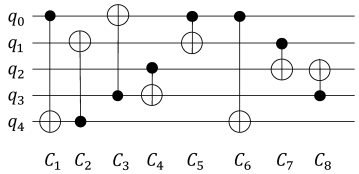

Fig. 9 An example of a quantum circuit to explain the constructing method of sub-circuits.

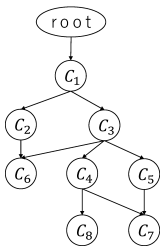

Fig. 10 A gate dependency graph of a quantum circuit in Fig. 9.

SAT solver returns that there does not exist such a qubit placement. Then, for the next trial, we use a SAT solver for $S_4 = \{C_1, C_2, C_3, C_4, C_6, C_8\}$ that also has 6 quantum gates, and then the SAT solver returns that there exists such a qubit placement.

Accordingly, in the same way as constructing S_3, we construct another new sub-circuit, S_5, which includes $(|S_1| + |S_4|)/2 = 7$ quantum gates. Thus, by considering the dependency of quantum gates, we consider $S_5 = \{C_1, C_2, C_3, C_4, C_5, C_6, C_8\}$ next. When a SAT solver is used for S_5, the SAT solver returns that there does not exist such a qubit placement. When a SAT solver is used for another sub-circuit that also has 7 quantum gates, the SAT solver returns that there does not exist such a qubit placement.

In conclusion, S_4 in Fig. 11 is a sub-circuit that has the largest number of quantum gates for our purpose. We can find a sub-circuit including the largest number of quantum gates by the above-mentioned binary search-based method.

Usually a sub-circuit containing the largest number of gates seems to be a good choice. However, sometimes such a sub-circuit needs too many SWAP gates to get the qubit placement for the sub-circuit. If we select such a sub-circuit, the final result may become worse. So, we try to select possibly the best sub-circuits as follows. We check how many SWAP gates are needed to get the qubit placement for all the possible sub-circuits to be performed on an NNA architecture. Then we select one sub-circuit such that the number of gates in the sub-circuit divided by that of inserted SWAP gates is the largest.

3.2 Qubit Placement with a SAT Solver

In the following, we consider qubits are placed on a 2D grid as shown in Fig. 13. If we choose the qubit placement as shown in Fig. 13, the control and the target bits are adjacent for all CNOT gates in the quantum circuit as shown in Fig. 12. Thus all operations in Fig. 12 can be performed

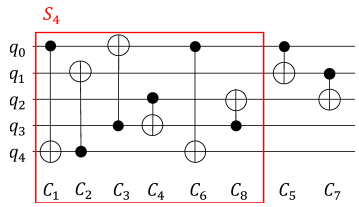

Fig. 11 sub-circuit $S_4 = \{C_1, C_2, C_3, C_4, C_6, C_8\}$.

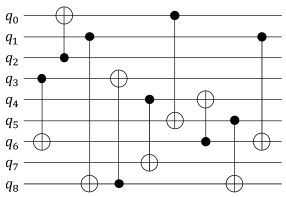

Fig. 12 A quantum circuit in which all operations can be performed on NNA without inserting SWAP gates.

q_5	q_0	q_2
q_8	q_3	q_7
q_1	q_6	q_4

Fig. 13 A qubit placement that allows the quantum circuit in Fig. 12 to be performed on an NNA without inserting any SWAP gate.

without inserting SWAP gates on this qubit placement.

In the following, we propose a method to find such a good 2D placement based on a Boolean satisfiability problem (SAT). Namely, we formulate a qubit placement problem as a Boolean function (i.e., a SAT problem instance) as follows: the derived Boolean function is *satisfiable* if and only if there exists a qubit placement for a given quantum circuit to be performed on an NNA without inserting SWAP gates. A SAT solver as explained below is used to figure out that such a qubit placement exists, and if it exists, the solver also finds how qubits are placed.

A SAT solver determines the satisfiability of a given Boolean function, and it can also provide a satisfying assignment when the problem is satisfiable. In our proposed method, one variable is used to express whether or not each qubit is placed on each cell on a 2D grid, and all the necessary conditions are expressed by Boolean formulas with such variables as we will explain in the following.

The following three conditions are needed to assign qubits on a 2D grid such that all operations in a sub-circuit can be performed without inserting SWAP gates.

Condition 1 A control bit and a target bit of all gates are adjacent.

Condition 2 Each qubit is assigned to only one cell on a 2D grid.

Condition 3 At most one qubit is assigned to each cell on a 2D grid.

As shown in Fig. 14, a cell of row i and column j on a 2D grid is expressed as (i, j). Logical variable $x_{i,j,k}$ expresses

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)

Fig. 14 An example of expressing cells on a 2D grid as coordinates.

whether qubit q_k is assigned to (i, j) or not. Namely, when qubit q_k is assigned to (i, j), $x_{i,j,k}$ becomes 1. Otherwise $x_{i,j,k}$ becomes 0. For example, if q_1 is assigned to $(2, 0)$, $x_{2,0,1}$ becomes 1. If q_1 is not assigned to $(2, 0)$, $x_{2,0,1}$ becomes 0.

First we consider the expression for Condition 1. For example, when there is a CNOT gate that has q_2 as a control bit and q_4 as a target bit, q_2 and q_4 have to be assigned adjacently. Thus, if q_2 is assigned to $(1, 1)$, q_4 has to be assigned to either one of $(0, 1)$, $(1, 0)$, $(1, 2)$ or $(2, 1)$. Accordingly, when $x_{1,1,2}$ is 1, either of $x_{0,1,4}$, $x_{1,0,4}$, $x_{1,2,4}$ or $x_{2,1,4}$ has to be 1. This condition can be expressed as Eq. (1).

$$(\neg x_{1,1,2}) \vee (x_{0,1,4} \vee x_{1,0,4} \vee x_{1,2,4} \vee x_{2,1,4}) \tag{1}$$

We consider such conditions of assigning q_2 and q_4 to adjacent qubits for each cell. Then, by ORing all the Boolean formulas for such conditions, we get a formula for the condition such that q_2 and q_4 should be placed adjacently. We consider such formulas for each pair of control and target bits for all gates, and we get the formula for Condition 1 by ANDing them.

Next we consider the expression for Condition 2. For example, q_0 has to be assigned to only one of cells on a 2D grid. This can be realized by considering the following two conditions: The first one is that q_0 is assigned to at least one cell, and the second one is that q_0 is assigned to at most one cell on a 2D grid.

The former condition can be expressed as at least one of $x_{i,j,0}$ has to be 1. Thus, as shown in Eq. (2), sum of $x_{i,j,0}$ needs to be 1. The condition also can be expressed as Eq. (3).

$$\sum_{i,j} x_{i,j,0} = 1 \tag{2}$$

$$x_{0,0,0} \vee x_{0,1,0} \vee x_{0,2,0} \vee \cdots\cdots \vee x_{i,j,0} \tag{3}$$

The latter condition can be expressed as follows: For example, if we do not want $x_{0,0,0}$ and $x_{0,1,0}$ to be 1 at the same time, we have Eq. (4) which means $x_{0,0,0}$ and $x_{0,1,0}$ cannot be 1 at the same time. That is, when Eq. (4) holds, q_0 cannot be assigned to both of $(0, 0)$ and $(0, 1)$ at the same time.

$$\neg x_{0,0,0} + \neg x_{0,1,0} = 1 \tag{4}$$

We consider similar formulas for all pairs of cells on a 2D grid as shown in Eq. (5). By ANDing these formulas, we have formulas for Condition 2 only for q_0.

$$\neg x_{i,j,0} + \neg x_{k,l,0} = 1 \quad ((i, j) \neq (k, l)) \tag{5}$$

If Eq. (3) and Eq. (5) hold, q_0 is assigned to at least one of cells on a 2D grid, and q_0 is assigned to at most one cell on a 2D grid. We can consider similar formulas for all qubits, and by ANDing them, we have a formulas for Condition 2.

To express Condition 3 as Boolean formulas, we use a similar method used to derive the formulas for Condition 2. At this time, no more than one qubit needs to be assigned to each cell. For example, Eq. (6) expresses a condition that prohibits assigning q_0 and q_1 to $(0, 0)$ at the same time.

$$\neg x_{0,0,0} + \neg x_{0,0,1} = 1 \tag{6}$$

We consider similar formulas for all pairs of q_i and q_j as shown in Eq. (7). By ANDing these formulas, we get a formulas of the condition that only one qubit is assigned to $(0, 0)$. We consider similar formulas for each cell, and we get the expression for Condtion 3 by ANDing them.

$$\neg x_{0,0,i} + \neg x_{0,0,j} = 1 \quad (i \neq j) \tag{7}$$

By ANDing the above expressions for Conditions 1, 2 and 3 all together, we finally obtain a SAT formula for our purpose. Thus, by using a SAT solver, we can determine if there exists a good qubit placement that allow us to perform a given circuit on an NNA without inserting SWAP gates. If such a qubit placement exists, a SAT solver find an satisfying variable assignment as well.

There may be a case when several qubit placements can satisfy the conditions. To get another satisfying variable assignment, we use a SAT solver repeatedly with adding the negated conditions obtained before. In such a case, we adopt a qubit placement with the smallest values of $h^*(n)$ which is a cost function to measure the quality of an intermediate solution in A* algorithm, which will be explained in detail in the next section.

3.3 Inserting SWAP Gates by A* Algorithm

After dividing a given quantum circuit into sub-circuits each of which can be performed without inserting SWAP gates, our remaining task is to insert SWAP gates between each pair of two sub-circuits to change the qubit placement so that each sub-circuit can be performed without inserting SWAP gates. We utilize A* algorithm to decide the way of inserting SWAP gates as we will explain in the following.

A* algorithm searches a graph to find a way from the start node S to the goal node G based on a cost function $f^*(n)$ which is the sum of $g^*(n)$ and $h^*(n)$ as shown in Eq. (8) where n means a node found during the search of a way in the graph. $g^*(n)$ is a cumulative cost from the start node to the current node, n, and $h^*(n)$ is a heuristic function that estimates the cost from the current node, n, to the goal node.

$$f^*(n) = g^*(n) + h^*(n) \tag{8}$$

Algorithm 3 shows our A* algorithm which inserts SWAP gates to change the qubit placement. We add a searched node to the *open list*. We sort the nodes in the open list based on $f^*(n)$. A node with the least $f^*(n)$ in the open

list is popped from the open list, and added to the closed list. This means that we select the node with the least $f^*(n)$ as promising to search first, and so we add all the nodes connected to it (i.e., the nodes we can reach by one move from the selected node) into the open list. These processes are repeated until we get to the goal, i.e., the objective qubit placement.

Each node in a graph used for our A* algorithm corresponds to a qubit placement. If two nodes are connected in the graph, a single SWAP gate can change the qubit placements between the two placements corresponding to the two nodes. In the following, S is the qubit placement of one subcircuit and G is the one of the following (next) sub-circuit; we find a way of inserting SWAP gates between two subcircuits by searching a shortest path from S to G in the graph for the A* algorithm.

$g^*(n)$ is the number of moves to reach n from S. That is, the number of necessary inserted SWAP gates to get to the qubit placement corresponding to n. $h^*(n)$ is the sum of manhattan distance between the locations of qubit q_i in the qubit placement corresponding to n and the objective qubit placement. When the qubit placement corresponding to n and G are as shown in Fig. 15 and Fig. 16, respectively, $h^*(n)$ is calculated as follows. q_0 is located on $(1, 1)$ in n. On the other hand, it is located on $(0, 0)$ in G. Thus, the manhattan distance of q_0 in these qubit placements is 2. The manhattan distance for other qubits is calculated in the same way, and the sum of the manhattan distance is as shown in Eq. (9).

Algorithm 3 A* algorithm inserting SWAP gates to change the qubit placement

```
Initialize the open list and the closed list
Add the starting node to the open list
while the open list is not empty do
    m ← openlist.pop()
    Add m to the closed list
    for each ḿ such that ḿ is a qubit placement obtained from m by inserting a single SWAP gate do
        if the qubit placement ḿ is equivalent to the one corresponding to G then
            break
        end if
        Calculate f*(ḿ) and add ḿ to the open list
    end for
    Sort nodes in the open list based on f*()
end while
```

$$h^*(n) = 2 + 1 + 2 + 1 = 6 \tag{9}$$

We show an example of inserting SWAP gates by A* algorithm as follows. In the example, we consider inserting SWAP gates to change the qubit placement from the one corresponding to S as shown in Fig. 15 to the one corresponding to G as shown in Fig. 16.

OL and CL stand for the open list and the closed list, respectively. At first, OL is $\{S\}$ and CL is $\{\}$ since S is the start node. Therefore, S is popped from OL and added to CL. As shown in Fig. 17, there are four ways to insert a SWAP gate to the qubit placement S and they are $S(q_2, q_3), S(q_1, q_3), S(q_0, q_1)$ and $S(q_0, q_2)$. These nodes are added to OL and then, OL is $\{A_1(5), A_3(5), A_4(5), A_2(9)\}$ after it is sorted based on $f^*(n)$ which are in the parenthesis.

Fig. 15 An example of qubit placement.

Fig. 16 The qubit placement corresponding to G after inserting SWAP gates.

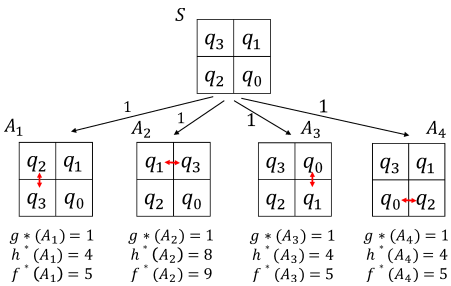

Fig. 17 An example of A* algorithm (Step1).

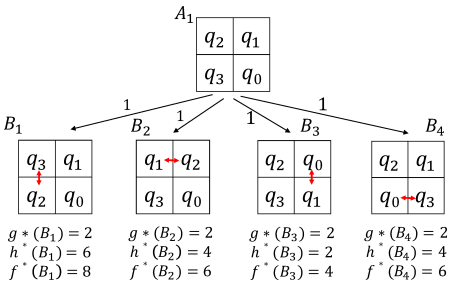

Fig. 18 An example of A* algorithm (Step2).

As shown in Fig. 18, A_1 is popped from OL and added to CL because A_1 is one of the nodes whose $f^*(n)$ is the smallest. Therefore, CL becomes as $\{S, A_1(5)\}$. There are four ways to insert a SWAP gate to the qubit placement A_1 and they are $S(q_2, q_3), S(q_1, q_2), S(q_0, q_1)$ and $S(q_0, q_3)$ as shown in Fig. 18. Since OL becomes as $\{B_3(4), A_3(5), A_4(5), B_2(6), B_4(6), B_1(8), A_2(9)\}$, B_3 is popped from OL and added to CL. Then, CL becomes as $\{S, A_1(5), B_3(4)\}$.

Similarly, there are four ways to insert a SWAP gate in the qubit placement B_3 as shown in Fig. 19 and they are $S(q_2, q_3), S(q_0, q_2), S(q_0, q_1)$ and $S(q_0, q_3)$. Now C_2 is the same qubit placement as the one corresponding to G, and so A* algorithm finishes.

The above example shows that it is possible to

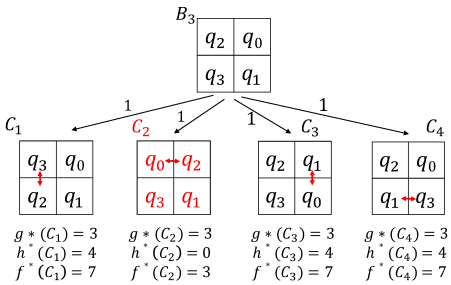

Fig. 19 An example of A* algorithm (Step3).

Table 1 The comparison between PAQCS and our proposed method.

Quantum circuits		SWAP gates		
Qubits	Two qubit gates	PAQCS	Proposed	Improvement(%)
16	50	110	49	55.45
16	100	221	91	58.82
16	200	445	187	57.98
25	50	162	76	53.09
25	100	331	139	58.01
25	200	665	287	56.84
36	50	208	103	50.48
36	100	443	194	56.21
36	200	895	428	52.18
49	50	272	138	49.26
49	100	562	264	53.02
49	200	1137	548	51.80

change the qubit placements corresponding to the change from S to G by inserting SWAP gates in the order of $S(q_2, q_3), S(q_0, q_1), S(q_0, q_2)$. Thus, by using the above A* algorithm, it is able to find a way of inserting SWAP gates to change the qubit placement for the following sub-circuit.

4. Experimental Results

We implemented the proposed method and PAQCS [20] in C++ to evaluate the performance of the proposed method. We generated 300 random benchmark quantum circuits consisting of only two-qubit gates whose control and target bits are chosen randomly. Then, we applied the proposed method and PAQCS to them in order to compare the average number of inserted SWAP gates. In the experiment, we utilized GlueMiniSat 2.2.8 for a SAT solver and a 4.20 GHz i7-7700K CPU with 16 GB RAM.

Each row of Table 1 reports the average number of inserted SWAP gates of 300 different random circuits by our method and PAQCS. Our proposed method can reduce the number of inserted SWAP gates by 54.43% on average compared to PAQCS. Even for larger quantum circuits, our method can find the solution within 10 minutes; On the other hand, PAQCS takes less than a minute. We confirmed that changing the order of quantum gates makes it possible to perform more gates on the same qubit placement (without inserting SWAP gates). We consider that this would be one reason why our method can reduce the inserted SWAP gates.

Our method inserts SWAP gates between each sub-circuit, and thus the number of sub-circuits affects the number of inserted SWAP gates in our method. In an extreme case, there is sometimes only one sub-circuit in our method when there are few quantum gates. In such a case, a SAT solver finds a qubit placement by which we can perform all the gates without inserting any SWAP gate. However, PAQCS may need to insert SWAP gates even in the same case because PAQCS determines the initial qubit placement heuristically unlike our method.

Note that both the SAT solver and the A* search used in our method need exponential time to the problem size. However, our experimental results show that our method can treat quantum circuits that are available currently like IBM-Q or in the near future. If much larger quantum circuits are available in the future, we may need to divide a large circuit into sub-circuits so that our method can treat each sub-circuit.

5. Conclusion

In this paper, we proposed a new idea to map a quantum circuit so that we can perform on an NNA; we proposed to change the order of quantum gates to decrease the number of inserted SWAP gates. By means of changing the order of quantum gates, we can indeed decrease the number of sub-circuits in which all the gates perform on adjacent qubits.

We utilize a SAT solver to find a good qubit placement such that the sub-circuit has only quantum gates performing on adjacent qubits in 2D architecture. Moreover, we utilize A* algorithm to insert SWAP gates for changing the qubit placement between two sub-circuits. As a result, we can reduce the number of inserted SWAP gates compared to the state-of-the-art heuristic, PAQCS.

In our proposed method, the performance of A* algorithm get worse when the target quantum circuit become larger. Thus, our future work is to find a way to insert SWAP gates to change the qubit placement more efficiently than our current A* algorithm. Also, as our future work, we should evaluate our framework by using benchmark circuits which are used in the research community of quantum circuit design.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 15H01677 and 18K19790, and by the Asahi Glass Foundation.

References

[1] P.W. Shor, "Polynomial-time algorithms for prime factorization and discrete log- arithms on a quantum computer," SIAM J. Comput., vol.26, no.5, pp.1484–1509, 1997.
[2] L.K. Grover, "A fast quantum mechanical algorithm for database search," Proc. Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp.212–219, 1996.
[3] H. Goudarzi, M.J. Dousti, A. Shafaei, and M. Pedram, "Design of a universal logic block for fault-tolerant realization of any logic operation in trapped-ion quantum circuits," Quantum Information Processing, vol.13, no.5, pp.1267–1299, May 2014.

[4] 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.
[5] R. Wille, M. Saeedi, and R. Drechsler, "Synthesis of Reversible Functions Beyond Gate Count and Quantum Cost," arXiv preprint arXiv:1004.4609, 2010.
[6] R. Wille, A. Lye, and R. Drechsler, "Optimal SWAP gate insertion for nearest neighbor quantum circuits," 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), pp.489–494, IEEE, 2014.
[7] M. Saeedi, R. Wille, and R. Drechsler, "Synthesis of quantum circuits for linear nearest neighbor architectures," Quantum Information Processing, vol.10, no.3, pp.355–377, June 2011.
[8] A. Shafaei, M. Saeedi, and M. Pedram, "Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures," Proc. 50th Annual Design Automation Conference, Article No. 41, ACM, 2013.
[9] M.M. Rahman and G.W. Dueck, "Synthesis of linear nearest neighbor quantum circuits," arXiv preprint arXiv:1508.05430, 2015.
[10] A. Chakrabarti, S. Sur-Kolay, and A. Chaudhury, "Linear nearest neighbor synthesis of reversible circuits by graph partitioning," arXiv preprint arXiv:1112.0564, 2011.
[11] A. Matsuo and S. Yamashita, "Changing the gate order for optimal LNN conversion," International Workshop on Reversible Computation, LNCS, vol.7165, pp.89–101, Springer, 2011.
[12] A. Shafaei, M. Saeedi, and M. Pedram, "Qubit placement to minimize communication overhead in 2D quantum architectures,"2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), pp.495–500, IEEE, 2014.
[13] D. Ruffinelli and B. Barán, "Linear nearest neighbor optimization in quantum circuits: a multiobjective perspective," Quantum Information Processing, vol.16, no.9, p.220, 2017.
[14] R. Wille, O. Keszocze, M. Walter, P. Rohrs, A. Chattopadhyay, and R. Drechsler, "Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits," 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), pp.292–297, IEEE, 2016.
[15] A. Farghadan and N. Mohammadzadeh, "Quantum circuit physical design flow for 2D nearest-neighbor architectures," Int. J. Circuit Theory and Applications, vol.45, no.7, pp.989–1000, July 2017.
[16] C.A. Pérez-Delgado, M. Mosca, P. Cappellaro, and D.G. Cory, "Single spin measurement using cellular automata techniques," Phys. Rev. Lett, vol.97, no.10, p.100501, 2006.
[17] A. Zulehner, A. Paler, and R. Wille, "An Efficient Mapping of Quantum Circuits to the IBM QX Architectures," arXiv preprint arXiv:1712.04722, 2017.
[18] A. Lye, R. Wille, and R. Drechsler, "Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits," 2015 20th Asia and South Pacific Design Automation Conference (ASP-DAC), pp.178–183, IEEE, 2015.
[19] D. Bhattacharjee and A. Chattopadhyay, "Depth-Optimal Quantum Circuit Placement for Arbitrary Topologies," arXiv preprint arXiv:1703.08540, 2017.
[20] C.C. Lin, S. Sur-Kolay, and N.K. Jha, "PAQCS: Physical Design-Aware Fault-Tolerant Quantum Circuit Synthesis," IEEE Trans. Very Large Scale Integration (VLSI) Systems, vol.23, no.7, pp.1221–1234, July 2015.

Wakaki Hattori received the B.E. degrees in Information Science and Engineering from Ritsumeikan University in 2018. He is currently a graduate student of Graduate School of Information Science and Engineering, Ritsumeikan University, Shiga, Japan. His research interests include quantum circuit design, quantum cost reduction and synthesis of Nearest Neighbor Architecture.

Shigeru Yamashita is a professor at the Department of Computer Science, College of Information Science and Engineering, Ritsumeikan University. He received his B.E., M.E. and Ph.D. degrees in Information Science from Kyoto University, Kyoto, Japan, in 1993, 1995 and 2001, respectively. His research interests include new types of computation and logic synthesis for them. He received the 2000 IEEE Circuits and Systems Society Transactions on Computer-Aided Design of Integrated Circuits and Systems Best Paper Award, SASIMI 2010 Best Paper Award, 2010 IPSJ Yamashita SIG Research Award, and 2010 Marubun Academic Achievement Award of the Marubun Research Promotion Foundation. He is a senior member of IEEE, and a member of IPSJ.