Changing the Gate Order for Optimal LNN Conversion

Atsushi Matsuo and Shigeru Yamashita

Graduate School of Science and Engineering Ritsumeikan University
1-1-1 Noji Higashi, Kusatsu, Shiga 525-8577, Japan
comp@ngc.is.ritsumei.ac.jp, ger@cs.ritsumei.ac.jp

Abstract. While several physical realization schemes have been proposed for future quantum information processing, most known facts suggest that quantum information processing should have intrinsic limitations; physically realizable operations would be only interaction between neighbor qubits. To use only such physically realizable operations, we need to convert a general quantum circuit into one for an so-called Linear Nearest Neighbor (LNN) architecture where any gates should be operated between only adjacent qubits. Thus, there has been much attention to develop efficient methods to design quantum circuits for an LNN architecture. Most of the existing researches do not consider changing the gate order of the original circuit, and thus the result may not be optimal. In this paper, we propose a method to convert a quantum circuit into one for an LNN architecture with the smallest number of SWAP gates. Our method improves the previous result for Approximate Quantum Fourier Transform (AQFT) by the state-of-the-art design method.

Keywords: Quantum Circuit, Linear Nearest Neighbor, Adjacent Transposition Graph.

1 Introduction

Since the invention of the integrated circuit in 1958, the number of transistors on an integrated circuit has doubled approximately every two years. Moreover, the size of transistors has been decreased by the advance of the semiconductor technology. However the size of transistors cannot be smaller than the atomic scale: we are approaching to the fundamental limits of the advance of the semiconductor technology.

Consequently, much attention has been paid to another computing paradigm such as quantum computing [1]. A quantum computer is a device to make computations by exploiting quantum mechanical phenomena, which enables to solve some problems more efficiently than classical computers such as factoring of numbers [2].

Several impressive researches have been studied for physically implementing quantum computers. With the advance of the quantum computing technology, it is getting clearer that there should be some intrinsic limitations on implementing quantum computers [3]. One of such limitations is that we cannot interact apart qubits by one basic operation [4]. By this intrinsic limitation, it is considered very difficult to make an interaction between two far apart qubits for most quantum technologies. For this reason, quantum circuits may be realized on an so-called Linear Nearest Neighbor (LNN) architecture which permits interactions only between adjacent (nearest neighbor) qubits.

A. De Vos and R. Wille (Eds.): RC 2011, LNCS 7165, pp. 89–101, 2012.

Therefore, for the coming "quantum computing era," it should be very important to establish a design technology for quantum circuits on an LNN architecture. Indeed, there have been many researches for this issue. Some researches designed specific quantum circuits on an LNN architecture manually, e. g., circuits for approximate quantum Fourier transform [5], Shor's factorization algorithm [4,6], quantum addition [7], and quantum error correction [8]. Others have developed methods to design general quantum circuits on an LNN architecture. For example, Hirata et al. proposed a heuristic to convert any quantum circuits to one for an LNN architecture [9]. Their method inserts SWAP gates in an initial circuit so that all gates are performed on adjacent qubits. Recently, Saeedi et al. have developed a very efficient design framework that utilizes their ideas of template matching and reordering strategies. The LNN AQFT circuit designed manually [5] has been improved by [9], and then successively improved further by [10]. Most of the above-mentioned techniques adopt the insertion of SWAP gates and the reorder of the initial qubit lines. In their methods, it has not been considered to change the gate order of the given initial circuit.

Our Contribution. In this paper, we explicitly consider a possibility to change the initial gate order in a case where the gate reordering is possible. More precisely, we consider a problem to find the smallest number of added SWAP gates for the LNN conversion by changing the order of gates if they can commute. Our main contribution is to formulate such a problem as a search problem on the *adjacent transposition graph*. Accordingly we can find the best solution with respect to the number of SWAP gates to be added. Our method can find an AQFT circuit with the fewest SWAP gates, which improved the result by [10].

The remainder of this paper is organized as follows. In Section 2, quantum circuits and LNN architectures are explained. Section 3 describes the adjacent transposition graph and our proposed method. We then provide experimental results in Section 4. Finally, we conclude this paper with a summary and future works in Section 5.

2 Preliminaries

In this section, we provide some basics necessary for our paper.

2.1 Quantum Circuit

A quantum circuit is a model of quantum computing. A quantum circuit indicates the order of basic unitary operators (called quantum gates) corresponding to a given quantum algorithm. Quantum bits and quantum gates are drawn on quantum circuits. An example of a quantum circuit is shown in Fig. 1, where each horizontal line indicates a quantum bit (denoted by "Qubit 1" to "Qubit 5"), and each dashed circle indicate a specific unitary operation called a quantum gate. On quantum circuits, time flows from left to right which means that quantum gates are applied from the leftmost gate in sequence. We explain quantum bits and quantum gates in detail below.

Quantum Bit. While in classical computers, a bit has to be either 1 or 0, in quantum computer, quantum bits (qubits in short) can be 1, 0 or the superposition state as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ which is the any linear combination of 1 and 0.

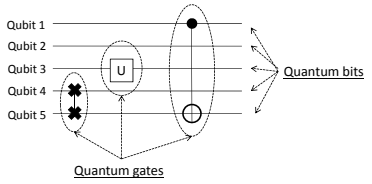

Fig. 1. A quantum circuit

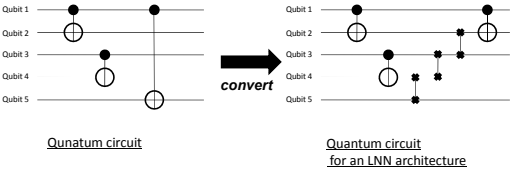

Fig. 2. Conversion of a quantum circuit to an LNN circuit

Quantum Gate. A quantum circuit consists of a cascade of quantum gates. A quantum gate indicates what unitary operator is applied to which qubit. For example, in Fig. 1, the leftmost gate indicates that a SWAP gate is applied to Qubits 4 and 5. The following three gates are mainly used in this paper.

- A SWAP gate is the leftmost gate in Fig. 1. A SWAP gate has two target bits x_{t1} and x_{t2}, and interchanges the values of the target bits.
- A one-qubit unitary gate is the middle gate in Fig. 1. A one-qubit unitary gate applies any unitary operations to the target bit.
- A CNOT gate is the rightmost gate in Fig. 1. A CNOT gate has a control bit and a target bit. In Fig. 1, a control bit of the CNOT gate is Qubit 1 and a target bit of the CNOT gate is Qubit 5. The following matrices: $\sigma_x = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ is applied to its target bit iff the state of its control bit is $|1\rangle$.

CNOT gates and one-qubit gates are universal so any quantum gates can be decomposed into a combination of CNOT gates and one-qubit gates.[1]

2.2 LNN Architecture

With the advance of the quantum computing technology, it is getting clearer that there should be some intrinsic limitations on implementing quantum computers. One of such limitations is that we cannot interact apart qubits by one basic operation. By this intrinsic limitation, it is considered very difficult to make an interaction between two far apart qubits for most quantum technologies. For this reason, quantum circuits may be realized on an so-called Linear Nearest Neighbor (LNN) architecture which permits interactions only between adjacent (nearest neighbor) qubits.

If a gate interacts two far apart qubits, we can make the gate interact two adjacent qubits by inserting (possibly many) SWAP gates before the gate. Thus by inserting SWAP gates, it is possible to convert a circuit to one with the same functionality for an LNN architecture. Fig. 2 shows an example of converting a quantum circuit to one for an LNN architecture. On the left circuit of Fig. 2, two target bits of the rightmost gate are apart. By inserting SWAP gates, the quantum circuit is converted to one for an LNN architecture. The right circuit in Fig. 2 has the same functionality, and it uses interactions between only adjacent qubits. Hereafter, let **an LNN circuit** denotes a circuit for an LNN architecture, and **a non-LNN circuit** denotes a circuit for a non-LNN architecture.

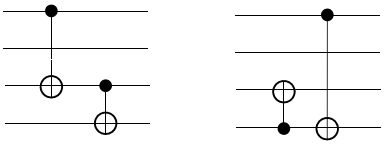

Fig. 3. Quantum gates that are not commutative by Condition 1

3 Changing the Gate Order for Optimal LNN Conversion

3.1 Problem Definition

The problem to convert a circuit into an LNN circuit is formulated as follow.

LNNizing problem. Given a quantum circuit that consists of one-qubit gates and two-qubit gates and the order of initial qubits, the problem is to convert the given quantum circuit to an LNN circuit with the smallest number of SWAP gates. The order of qubits after conversion should be the same as the initial qubits since the circuit may be used as a sub-circuit for a large quantum circuit. Formally, the input and the output of the problem are as follows.

Input : a given quantum circuit, and initial qubit order.
Output : an LNN circuit with the same qubit order as the initial one.

3.2 Dependence between Quantum Gates

In a quantum circuit, there is usually dependence between a pair of two gates such that one gate should be applied before the other gate, i. e., some gates cannot commute. When we convert a quantum circuit to an LNN circuit, the dependence between quantum gates has to be kept, otherwise the logical functionality of the quantum circuit is changed after converting it to an LNN circuit.

Quantum gates that satisfy one of the following conditions are not commutative. In the following two conditions, let two gates be A and B. Let also the control bits and the target bits of A be A_{cb} and A_{tb}, respectively. Also B_{cb} and B_{tb} have the same meaning for the gate B.

Condition 1: The first condition is that $A_{cb} = B_{tb}$. This means that the control bit of one gate is the same as the target bit of the other gate. Fig. 3 shows an example of two quantum gates that are not commutative by Condition 1.

Condition 2: The second condition is $A_{tb} = B_{tb}$ and two unitary matrices that are applied to A_{tb} and B_{tb} are not commutative. The target bits of two quantum gates are the same and the unitary matrices that are applied to the target bits are not commutative, then the two gates are not commutative. Fig. 4 shows an example of two quantum gates that are not commutative by Condition 2. The gate denoted by H (in Fig. 4) is called *Hadmard* gate that apply the following matrix: $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ to its target bit. The matrices σ_x and H are not commutative so that interchanging these two gates will change the logical functionality of the quantum circuit.

Fig. 4. Quantum gates that are not commutative by Condition 2

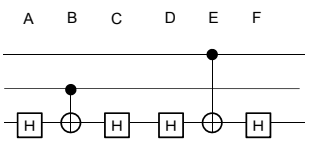

Fig. 5. Groups of quantum gates that are commutative

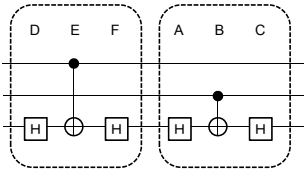

Fig. 6. The interchanged groups of quantum gates

A pair of quantum gates that satisfies one of the above two conditions is not commutative. If a quantum circuit has such a pair, we should not change the order of the pair of gates when we convert the circuit to an LNN circuit.

Two quantum gates cannot be interchanged (with each other) if one of the above two conditions do hold. Even in such a case, we may swap two groups of quantum gates. We do not consider such a special case in this paper, but only mention such an example below. For example, a quantum gate E and a quantum gate D or C in Fig. 5 are not commutative by Condition 2. Two quantum gates C and B in Fig. 5 are not commutative by Condition 2. Accordingly, we conclude that the two gates, E and B, cannot be swapped, and thus when we convert the circuit of Fig. 5 to an LNN circuit, we only consider the case where the gate B should be applied before gate E. However, we can swap two groups of gates in this example. Namely, the circuit in Fig. 5 can be transformed to one in Fig. 6, and thus we can also consider the case where the gate E is applied first when we convert the circuit to an LNN circuit.

In Fig. 6, despite the fact that quantum gates A, B, C and D, E, F are not commutative by Condition 2, the groups of quantum gates are commutative because interchanging two groups of the quantum gates will not change the logical functionality of the quantum circuit.

The above example tells us that we may change the order of application of two quantum gates even if either one of the two conditions holds. Considering such a possibility may reduce the cost of conversion. However it is difficult and almost impractical to consider a possibility of swapping two groups of gates; we consider only the possibility of swapping individual gates in this paper.

3.3 Gate Dependence Graph

From a non-LNN circuit, we can construct a *gate dependence graph* by considering Condition 1 and 2. A gate dependence graph is a directed graph that shows the dependence of quantum gates in a given quantum circuit. Each node, n_i, in a gate dependence graph corresponds to one specific gate, g_i, in the given circuit. An edge between two nodes n_i and n_j means that gate g_i should be applied before gate g_j, i. e., we cannot

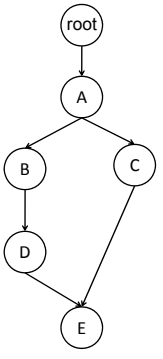

Fig. 7. A gate dependence graph

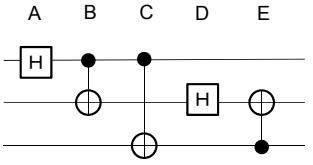

Fig. 8. A quantum circuit before LNN conversion

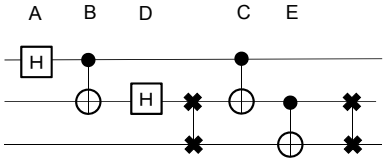

Fig. 9. A quantum circuit after LNN conversion

change the order of application of gates g_i and g_j. This can be obviously determined by Condition 1 and 2. By technical convenience, we have one special root node in a gate dependence graph. Each node connected to the root node indicates that the corresponding gate does not depend on any other gates, i. e., the gate can be applied first in the circuit. Fig. 7 shows the gate dependence graph for a quantum circuit in Fig. 8. Fig. 9 shows a quantum circuit after converting the circuit in Fig. 8 to an LNN circuit. By this conversion, the gate order is changed but the dependences of quantum gates remain unchanged.

In the following, we will use the following notation: for a set of gates Γ, and a gate dependence graph G (where we assume each gate g_i in Γ has the corresponding node n_i in G), $\kappa_{\Gamma, G}$ denotes a set of gates g having the following two properties: (1) g is included in Γ, and (2) there is no g' in Γ such that the corresponding node of g' is a predecessor of the corresponding node of g in G.

3.4 Adjacent Transposition Graph

In this section, adjacent transposition graphs are explained. In an adjacent transposition graph, a node corresponds to a permutation, and an edge corresponds to a SWAP gate. The numbers in each node indicates the order of quantum bits. Hereafter, let n denote the number of qubits of a given quantum circuit. Each node has $(n-1)$ edges. The total numbers of the nodes and the edges in an adjacent transposition graph for an n-qubit circuit are $n!$ and $\frac{(n-1)n!}{2}$, respectively. Fig. 10 is an example of an adjacent transposition graph with $n = 4$.

By finding the shortest path (in an adjacent transposition graph) corresponding to a sequence of qubit orders that realizes all quantum gates on an LNN architecture, optimal conversion that considers the order of the quantum gates is possible. In the following example, a quantum circuit in Fig. 11 is converted to an LNN circuit by utilizing an adjacent transposition graph. Fig. 12 is a gate dependence graph of the quantum circuit in Fig. 11, and Fig. 13 shows how to convert a quantum circuit to an LNN circuit by

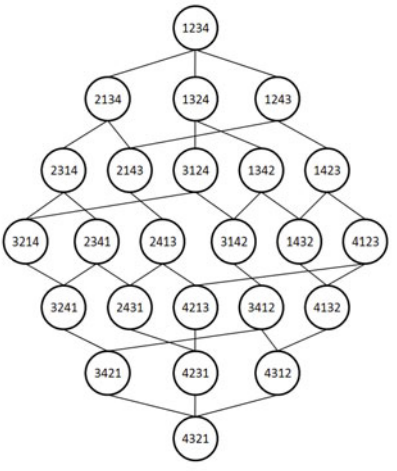

Fig. 10. A adjacent transposition graph with $n = 4$

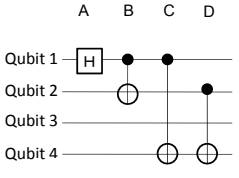

Fig. 11. A non-LNN circuit

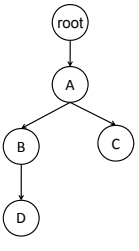

Fig. 12. The gate dependence graph for the quantum circuit in Fig. 11

utilizing an adjacent transposition graph. Numbers on the quantum circuit of Fig. 13 indicates the order of the qubits at the indicated point of time.

We will use the following notations:

- A gate is said to be **LNN-realizable on the qubit order *N*** if it can be applied by interacting only adjacent qubits when the qubit order is N. In the following, let λ_N be a set of LNN-realizable gates on the qubit order N.
- A gate is said to be **applicable with respect to a set of gates Γ, and gate dependence graph *G*** (where we assume each gate g_i in Γ has the corresponding node n_i in G), if it is included in $\kappa_{\Gamma, G}$ defined in Sec. 3. 3.
- A gate is said to be **realizable at a qubit order *N* with a remaining gate set Γ and a gate dependence graph *G***, if it is included in $\lambda_N \cap \kappa_{\Gamma,G}$. In other words, if the current situation permits to apply some of the gates in Γ, such gates are called "realizable". We will use this terminology in the rest of this paper.

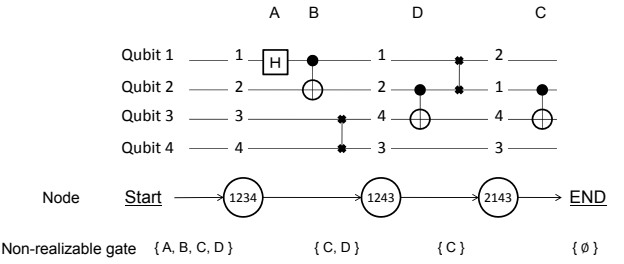

Fig. 13. Conversion of a quantum circuit to an LNN circuit by utilizing the adjacent transposition graph

3.5 Overview and an Example of the Proposed Method

Our main idea to convert a non-LNN circuit into an LNN-circuit is to formulate the problem as finding an optimal path (with respect to some desired property explained below) on an adjacent transposition graph. The overview is as follows.

We first let Γ be the set of all the gates in the given non-LNN circuit. Then, from Γ, we remove a gate one by one if the gate is realizable at the current situation. Realizable gates can be applied in the converted LNN circuit, so we remove all such gates from Γ, and place them at the end of the converted LNN circuit. In this way, the converted LNN circuit will grow.

If there is no realizable gates are remained in Γ, we insert (possibly many) SWAP gates at the end of the converted LNN circuit so that the changed qubit order make some gates in Γ realizable. We continue this until Γ becomes empty.

Note that putting a SWAP gate at the end of the LNN circuit corresponds to a move from one node to adjacent node in its adjacent transposition graph. Thus, our problem is essentially to find "the best" path in the adjacent transposition graph.

Now let us provide with an example to help understanding the above. The quantum circuit in Fig. 11 is composed of four gates A, B, C and D. At first, we set $\Gamma = \{A, B, C, D\}$.

The initial qubit order is $N = 1234$, and $\lambda_{1234} = \{A, B\}$. In Fig. 12, $\kappa_{\Gamma, G} = \{A\}$ at first. Then $\lambda_{1234} \cap \kappa_{\Gamma, G} = \{A\}$, and thus A can be applied in an LNN circuit. After placing A at the LNN circuit, we remove A from Γ. This can be formally written as $\Gamma \leftarrow \Gamma \setminus \lambda_{1234} \cap \kappa_{\Gamma, G}$. By removing A from Γ, Γ will be changed and so does $\kappa_{\Gamma, G}$. The new $\kappa_{\Gamma, G}$ will be $\kappa_{\Gamma, G} = \{B, C\}$. Therefore new realizable quantum gates will be $\lambda_{1234} \cap \kappa_{\Gamma, G} = \{B\}$, and B can be applied in the LNN circuit. In Fig. 13, when $N = 1234$, quantum gates, A and B, are realized on an LNN architecture.

After realizing quantum gates, the other quantum gates sometimes become realizable like quantum gates, A and B. For this reason, the operation $\Gamma \leftarrow \Gamma \setminus \lambda_N \cap \kappa_{\Gamma, G}$ is repeated until $\lambda_N \cap \kappa_{\Gamma, G}$ becomes ϕ.

By inserting a SWAP gate, $N = 1234$ can be changed to $N = 1243$. Then $\lambda_{1243} = \{D\}$ and $\kappa_{\Gamma, G} = \{C, D\}$. Therefore $\lambda_{1243} \cap \kappa_{\Gamma, G} = \{D\}$ so we remove D from Γ. Again, this can be written as $\Gamma \leftarrow \Gamma \setminus \lambda_{1243} \cap \kappa_{\Gamma, G}$. In Fig. 13, when $N = 1243$, the quantum gate D is realized on an LNN architecture. Then the new $\kappa_{\Gamma, G}$ is $\kappa_{\Gamma, G} = \{C\}$ and $\lambda_{1243} \cap \kappa_{\Gamma, G}$ is ϕ, and so we go to the next order of the qubits.

Algorithm 1. BFS by using ATG (Γ, N, G).

```
1: Γ ← Γ \ λ_N ∩ κ_{Γ, G}
2: if Γ is φ then
3:     terminate searching
4: end if
5: M ← φ
6: M[N] ← {Γ}
7: Q.push ( (N, Γ) )
8: while Q is not empty do
9:     (N, Γ) ← Q.pop()
10:    for all N' ∈ the adjacent nodes of N do
11:        while λ_{N'} ∩ κ_{Γ, G} is not φ do
12:            Γ ← Γ \ λ_{N'} ∩ κ_{Γ, G}
13:        end while
14:        if Γ is φ then
15:            terminate searching
16:        else
17:            if M[N'] has not been registered then
18:                M[N'] ← {Γ}
19:                Q.push ( (N', Γ) )
20:            else
21:                if ∃x ∈ M[N'] such that x ⊆ Γ then
22:                    continue
23:                else
24:                    M[N'] ← M[N'] ∪ {Γ}
25:                    Q.push ( (N', Γ) )
26:                end if
27:            end if
28:        end if
29:    end for
30: end while
```

By utilizing the adjacent transposition graph in Fig. 10, we can essentially do the above procedure to convert the quantum circuit in Fig. 11 to the LNN circuit in Fig. 13. In the next section, we will describe a formal algorithm to do so as the breadth first search on an adjacent transposition graph.

3.6 The Breadth First Search by Utilizing the Adjacent Transposition Graph

Now we are ready to show our breadth first search algorithm formally as *Algorithm 1*. In the algorithm, M is a map from a qubit order to a set of sets of quantum gates that have not been placed on a converted LNN circuit yet. Q is a queue of pairs of the order of qubits, and quantum gates that have not been made realizable yet. For a qubit order (which is a permutation) N, there is the corresponding node in the adjacent transposition graph, and they are conceptually the same. Therefore, for an easy writing, we will use the notation N to mean a qubit order, or a node in the adjacent transposition graph, interchangeably depending on the context.

In the breadth first search, we may get to the same permutation node many times during the breadth first search. If the search get to the same (permutation) node again and there is no essential improvement in the set of gates that have not been made realizable, it is useless to continue the further search from the node. The reason is that we can always find the better solution from the same (permutation) node that has been visited before.

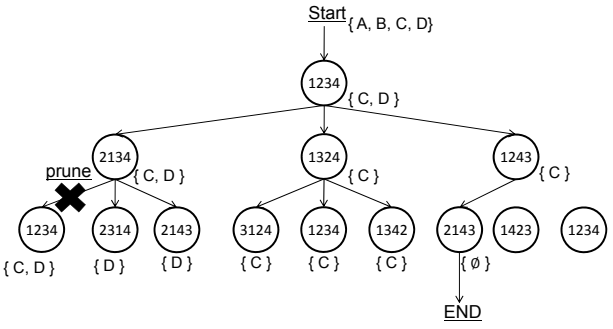

Fig. 14. The breadth first search by utilizing the adjacent transposition graph

We now explain the detail of the *Algorithm 1*. From lines 1 to 7, variables are initialized. If a given quantum circuit can be converted to an LNN circuit with no SWAP gates, we terminate the search at line 2. The breadth first search starts from line 8. At line 9, we pop a pair of a permutation N and gate set Γ from the queue, and we keep searching from the permutation node N. To do so, from line 10 to line 13, we move to the adjacent node, N', of N one by one, and we remove some gates that become realizable at the move (if any) from Γ. This operation can be formally written at lines 11 to 13.

If Γ becomes empty, we can conclude that we have found the best solution, and thus we stop the further search at line 14.

The lines after line 17 are for pruning the redundant search (from the same permutation node with no improvement from the previous visit) as mentioned above. In the algorithm, M is used as a cache to store the previous results not to perform the redundant search.

The detail is explained in the following. From lines 17 to 19, the case when $M[N']$ has not been registered indicates that the node N' in the adjacent transposition graph has never bee visited while searching. In such a case, Γ is registered into $M[N']$, and the pair of N and Γ is pushed into the queue.

The lines after line 21 deal with the case when $M[N']$ has already been registered. This indicates that the node N' in the adjacent transposition graph has already been visited while searching. In this case, we need to check whether further search from this node is useful or not. One searching path is not useful when (previous) another search path has already visited the same node N' in the adjacent transposition graph, and the set of the quantum gates that have not been made realizable yet at that time is the subset of Γ. This condition is checked at line 21.

In the example as shown in Fig 14, the left most node satisfies the above condition, and thus the further search from this node is pruned. When $N = 1234$ is visited at the first time (the starting node of the graph), the set of the quantum gates that have not been made realizable yet is $\{C, D\}$. When $N = 1234$ is visited again at the left most node, the quantum gates that have not been made realizable yet is $\{C, D\}$. Accordingly, the further search is pruned because $\{C, D\} \subseteq \{C, D\}$. This means that there is no possibility to find the better answer even if the search is continued.

Table 1. Experimental results

			naive method		proposed method	
Circuit	n	gc	# of SWAP gates	Time (sec)	# of SWAP gates	Time (sec)
3_17_15	3	10	4	0.01	4	0.01
decod24-v0_40	4	9	6	0.01	6	0.01
decod24-v1_42	4	9	6	0.01	6	0.01
decod24-v2_44	4	9	6	0.01	6	0.01
decod24-v3_46	4	9	6	0.01	6	0.01
fredkin_5	3	7	2	0.01	2	0.01
miller_12	3	8	6	0.01	2	0.01
toffoli_double_3	4	7	8	0.01	4	0.01
SteaneEncoding	7	14	26	1.43	18	1.85
SteaneErrorDetection	10	12	38	1.42	34	4572.59
add8_173	25	48	46	0.33	–	–

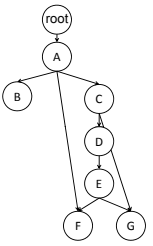

Fig. 15. The gate dependence graph of *fredkin_5*

4 Experimental Results

In this section, we show some experimental results. The proposed method was implemented in C++, and the experiments were done on an Intel Core i7-929 2.66GHz with 24GB memory.

Before showing our results, we would like to note that our method can always find the best result with respect to the number of SWAP gates. Therefore, we do not need to show the comparison with respect to the quality of the converted results. The reason is that our method essentially performs an exhaustive search. Thus, we show (1) how large problems our method can treat, and (2) how close the results of a naive method are to the best results (by our method). Also, we would like to show that (3) our method could indeed find better results compared to the state-of-the-art existing method.

To show the above (1) and (2), we compared the proposed method to the naive approach, in which the gate order is not considered, i. e., the gate order is not changed from the original one. The results are shown in Table 1. The first column gives the names of the circuits in RevLib [11], except for SteaneEncoding and SteaneErrorDetection. The second and the third columns denote the number of qubits (n), and the gate count (gc) of the circuits, respectively. The following columns show the number of SWAP gates and run-time for the conversion by the naive method and the proposed method.

As can be observed from Table 1, in some cases, a naive method cannot find the best solution. Also, as expected, our method could not deal with a large problem; the quantum circuit, add8_173, in Table 1 could not be converted to an LNN circuit by the proposed method because of the memory explosion. Let (the number of qubits-1) be M, and the depth of the breadth first search to find the best solution be d. Then, the space complexity of the proposed method is obviously $O(M^d)$. Therefore, when the depth of the breadth first search to find the best solution increases, its space complexity increases exponentially. By considering this and the experimental results, we can observe that our method may be able to find the best solution when we need to insert SWAP gates up to around 35 or so. For this reason, if the converted LNN circuit need many SWAP gates, it is not practical to use our proposed method; we may need a heuristic search method.

For the quantum circuits, 3_17_15, decode24, fredkin_5 in Table 1, the numbers of SWAP gates of the naive method and the proposed method are the same. The reason can be seen in their gate dependence graphs. For example, see the gate dependence graph

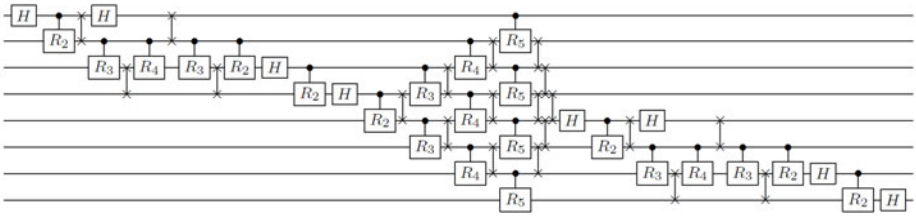

Fig. 16. The LNN AQFT circuit by the method by Saeedi et al. [10]

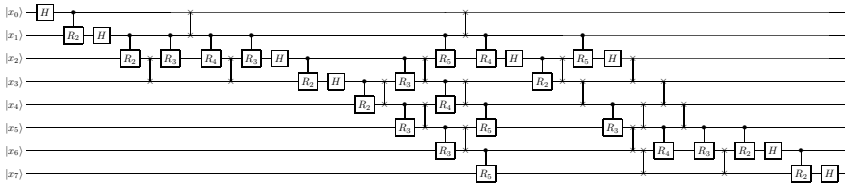

Fig. 17. The LNN AQFT circuit by the proposed method

for *fredkin_5* in Fig. 15, it is observed that we can only change the order of B and C, and/or F and G. Thus, there is no much difference between the naive method and the proposed method.

To show the above (3), we compared the results of converting the AQFT circuit with the state-of-the-art existing method by Saeedi et al. [10]. As described in Sec. 1, the original LNN circuit for AQFT [5] has been improved by the method [9], and then by Saeedi et al. [10]. Fig. 16 shows the LNN circuit for AQFT by Saeedi et al. [10] which has been considered to be the best. Fig 17 shows an LNN circuit for AQFT by our proposed method. As can be seen, the gate order is a bit changed, and the SWAP gates are reduced; The numbers of SWAP gates used are 20 in the method by Saeedi et al. and 18 in our proposed method, respectively.

5 Conclusions

In this paper, we have proposed how to convert a non-LNN quantum circuit into an LNN circuit with the smallest number of SWAP gates by considering the gate order. When the number of SWAP gates is small, the proposed method is able to find the optimum conversion easily with considering the gate order. The experiments show that our method can reduce the number of SWAP gates by two compared to the LNN AQFT circuit converted by the method by Saeedi et al. .

However, if the number of SWAP gates is large, the space complexity increases exponentially. Thus, to convert a large quantum circuit to an LNN circuit in reasonable time, we first need to divide the circuit into small partial circuits, and then apply the proposed method to each partial circuit one by one. Another way to convert a large non-LNN circuit is a heuristic conversion method; it would be an interesting to develop a heuristic.

Because our problem formulation can be seen as a search problem on a graph, we may be able to utilize an idea of the existing traversal algorithms for our purpose.

Acknowledgement. This research was partially supported by Kayamori Foundation of Informational Science Advancement.

References

1. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press (2000)
2. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26, 1484–1509 (1997)
3. Ross, M., Oskin, M.: Quantum computing. Commun. ACM 51(7), 12–13 (2008)
4. Fowler, A.G., Devitt, S.J., Hollenberg, L.C.: Implementation of shor's algorithm on a linear nearest neighbour qubit array. Quantum Information and Computation 4(4), 4:237–4:251 (2004)
5. Takahashi, Y., Kunihiro, N., Ohta, K.: The quantum fourier transform on a linear nearest neighbor architecture. Quantum Information and Computation 7(4), 383–391 (2007)
6. Kutin, S.A.: Shor's algorithm on a nearest-neighbor machine. Technical report, Asian Conference on Quantum Information Science (2007)
7. Choi, B.S., Meter, R.V.: Effects of interaction distance on quantum addition circuits. ArXiv e-prints (September 2008)
8. Fowler, A.G., Hill, C.D., Hollenberg, L.C.L.: Quantum-error correction on linear-nearest-neighbor qubit arrays. Phys. Rev. A 69(4), 042314 (2004)
9. Hirata, Y., Nakanishi, M., Yamashita, S., Nakashima, Y.: An efficient coversion o quantum circuits to a linear nearest neighbor architecture. Qnantum Information and Computation 11(1), 142–166 (2011)
10. Saeedi, M., Wille, R., Drechsler, R.: Synthesis of quantum circuits for linear nearest neighbor architectures. Quantum Information Processing, 1–23 (2010), 10.1007/s11128-010-0201-2
11. Wille, R., Große, D., Teuber, L., Dueck, G.W., Drechsler, R.: RevLib: An online resource for reversible functions and reversible circuits. In: International Symposium on Multiple Valued Logic, pp. 220–225 (May 2008)