Logical-Depth-Oriented Reversible Logic Synthesis

Mona Arabzadeh, Morteza Saheb Zamani, Mehdi Sedighi and Mehdi Saeedi
Quantum Design Automation Lab, Computer Engineering Department
Amirkabir University of Technology
Tehran, Iran
{m.arabzadeh, szamani, msedighi, msaeedi}@aut.ac.ir

***Abstract*—Boolean reversible logic has applications in various research areas including quantum computing. In this paper, a synthesis method for reversible logic is proposed which concentrates on two objectives, namely logical depth and quantum cost. To this end, the input specification is represented in a cycle form, and a set of three distinct subsets of input cycles are distinguished for parallel execution where the first set is executed on the first n input qubits and the remaining two sets are applied on the set of $2n$ ancillae. The final results are constructed on the main qubits. For each set, the cycle-based synthesis method is applied equipped with negative-control gates and parallel gate execution. Experimental results on the attempted benchmarks demonstrate more than 50% improvement in the logical depth with less than 2% increase in the quantum cost on average.**

I. Introduction

In the current quantum computing technologies [1], qubits can hold their states only for a limited period of time due to decoherence. As such, implementing parallelized quantum circuits attracts research interest. Quantum circuit parallelism has been studied in various papers [2], [3] where a tradeoff between space complexity (i.e., ancillae) and time complexity (i.e., circuit depth) is discussed.

QNC is defined as the quantum analog of the efficient parallel class NC in classical computation [2]. It has been shown that various classes of circuits consisting of CNOT, Hadamard and controlled π-shifts can be parallelized to a logarithmic depth. It was also proved that any circuit consisting of CNOT gates can be parallelized to $O(\log n)$ depth with $O(n^2)$ ancillae. Hence, any family of such circuits is in QNC.

To parallelize quantum circuits, an automated technique was developed in [3] which translates a quantum circuit from the quantum circuit model of computation to the measurement-based model [4] in the presence of ancillae. The authors distinguished circuits with clifford gates of polynomial depth can be parallelized to a logarithmic depth by adding a polynomial number of ancillae.

In [5], a post-process algorithm was proposed to reduce the logical depth of a given quantum circuit. To do so, a set of circuit templates was introduced and applied to reduce quantum cost and logical depth. Additionally, a greedy algorithm was suggested to compact the number of circuit levels.

Various techniques were used by different researchers for particular quantum circuits too. Fast implementation of quantum modular exponentiation was discussed in [6]. It has been shown that the time complexity of modular exponentiation can be reduced from $O(n^3)$ to $O(\log^3 n)$ with the aim of $O(n^3)$ ancillae. In [7], an implementation of quantum carry-lookahead adder with $O(\log n)$ depth was introduced which uses $O(n)$ ancillae. Applying a post-process algorithm such as [5] can improve the result of [7] further.

While quantum cost is the main objective of reversible synthesis algorithms [8], [9], they do not consider logical depth explicitly. In other works [10], [11], a large number of ancillae was used which can restrict their usage in the current quantum technologies. The aim of this paper is to consider both logical depth and quantum cost in a synthesis algorithm for reversible logic, instead of applying a post-process method, without significant increase in quantum cost. To this end, the cycle representation of a given specification is considered and input cycles are partitioned into three subsets. Additionally, $2n$ ancillae are added to an n-input circuit. One subset is synthesized on input qubits and the other subsets are synthesized on the other sets of ancillae. The cycle-based synthesis algorithm [9] is applied with explicit consideration of circuit depth for each subset. Our experimental results show the logical depth of a given circuit can be improved by up to 67% using the proposed method.

The rest of the paper is organized as follows. Basic concepts are explained in Section II. Related work on reversible logic synthesis which is used in our method is described in Section III. The proposed cycle-based synthesis method is introduced in Section IV. Experimental results are reported in Section V and finally, Section VI concludes the paper.

II. Preliminaries

Basic concepts of reversible logic synthesis are described in the following subsections:

A. Permutation Function

Let B be any set and define $f : B \rightarrow B$ as a one-to-one and onto transition function. The function f is called a *permutation* function, as applying f to B leads to a set with the same elements of B and probably in a different order. If $B = \{1, 2, 3, ..., m\}$, there exist two elements b_i and b_j belonging to B such that $f(b_i) = b_j$. A k-cycle with *length* k is denoted as $(b_1, b_2, ..., b_k)$ which means that $f(b_1) = b_2, f(b_2) = b_3, ...,$ and $f(b_k) = b_1$. A given k-cycle $(b_1, b_2, ..., b_k)$ could be written in many different ways, such as $(b_2, b_3, ...b_k, b_1)$. Cycles c_1 and c_2 are called *disjoint* if they have no common members. Any permutation can be written

uniquely, except for the order, as a product of disjoint cycles. If two cycles c_1 and c_2 are disjoint, they can *commute*, i.e., $c_1c_2 = c_2c_1$. A cycle with length two is called *transposition*. A cycle or a permutation is called *even* (*odd*) if it can be written as an even (odd) number of transpositions. A k-cycle is even (odd) if k is odd (even).

B. Reversible Function

An n-input, n-output, fully specified Boolean function $f : B \rightarrow B$ over variables $X = \{a_1, ..., a_n\}$ is called *reversible* if it maps each input pattern to a unique output pattern. Each reversible function can be considered as a permutation function. In this paper, n is particularly used to refer to the number of inputs/outputs of a circuit (i.e., circuit size). Additional lines to a circuit is called *ancillae*.

C. Reversible Gate

An n-input, n-output gate is reversible if it realizes a reversible function. Previously, various reversible gates with different functionalities have been proposed. Among them, *multiple-control Toffoli* gate has been used by different synthesis methods [12], [13], [8], [14], [9], [15] and various optimization methods were introduced [16], [5], [17], [18]. A multiple-control Toffoli gate can be written as C^{m}NOT(C; t), where $C = \{x_{i_1}, \ldots, x_{i_m}\} \subset X$ is the set of *control* lines and $t = \{x_j\}$ with $C \cap t = \emptyset$ is the *target* line. The value of the target line is inverted if all of the control lines have the required zero or one values. A control line may be *positive* (*negative*) which means that if its value is one (zero), the target is inverted. For m=0 and m=1, the gates are called NOT and CNOT, respectively. For m=2, the gate is called C^{2}NOT or Toffoli.

D. Quantum Cost

In addition to the C^{m}NOT gate, several other gate types have been proposed in the literature [1]. Controlled-V (controlled-V†) changes the value on its target line using the transformation given by the matrix V (V†) if the control line has the value of 1.

$$V = \frac{1+i}{2}\begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix}, V^\dagger = \frac{1-i}{2}\begin{bmatrix} 1 & i \\ i & 1 \end{bmatrix}$$

The gates NOT, CNOT, controlled-V, and controlled-V$^+$ (with positive controls) have been efficiently simulated in some quantum computer technologies [19]. These gates are considered as *elementary gates* for reversible Boolean functions. The number of elementary gates required for simulating a given gate is called *quantum cost*. A *reversible circuit* includes a set of reversible gates and each reversible gate consists of several elementary gates. The simulation of multiple-control Toffoli gates are studied in different works [20], [5]. In our work, the results of [5] are used for calculating the number of elementary gates in C^{m}NOT gates with both positive and negative controls. Multiple-control Toffoli gates with both positive and negative controls are used in some synthesis and optimization methods of reversible circuits [21], [14], [17].

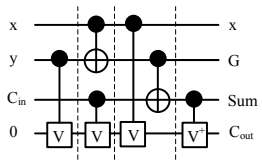

Fig. 1. A simplified and a level-compacted 3-input full adder [5].

E. Logical Depth

In a reversible circuit, elementary gates which can be applied in parallel are considered as one logic level [5]. The number of logical levels in a circuit is called *logical depth* or *depth* in short. The depth of a circuit is less than or equal to the number of elementary gates in that circuit. Decreasing the circuit depth reduces the execution time of a given circuit which mitigates the effect of decoherency. Figure 1 shows a simplified and a level-compacted circuit for a 3-input full adder [5]. The logical levels of this circuit are separated by dash lines. The depth of this circuit is 4 with 6 elementary gates (i.e., quantum cost = 6). The depth 4 is optimal here since four qubits are involved in the 4th qubit [5].

III. Previous work

Several synthesis methods have been proposed during the recent years most of which use heuristic methods with different kinds of input representations such as truth table [22], [14], binary decision diagrams [10], positive polarity Reed-Muller expansion [13], [8] and cycle forms [12], [9], [23]. Among them, the methods in [12], [9] and [23] use the cycle form of the input specification which is used in our work to decompose the input specification into subsets to be applied in parallel. In cycle representation, fixed rows of the truth table are removed and are not considered in the synthesis process. This helps to save space and also to eliminate the gates which may change and then fix the values of these rows. Additionally, cycle representation distinguishes disjoint subsets which is particularly important in our work for parallel synthesis.

In [12], an NCT-based synthesis method is proposed. In NCT-based synthesis, the library of gates consists of NOT, CNOT and Toffoli gates. In this method, first 0 and 2^i terms are positioned at their right locations. Then, the resulted k-cycle is decomposed into a set of transpositions. Each pair of transpositions $(a, b)(c, d)$ is converted into specific terms by a circuit called π circuit. The specific terms are $2^n - 4$, 2^n-3, 2^n-2 and 2^n-1. Then a pre-defined circuit, called κ_0, implements a 2-cycle over those specific terms i.e., ($2^n - 4$, 2^n-3) (2^n-2, 2^n-1). After that, the inverse π circuit, π^{-1}, is applied to convert the terms into their primary states a, b, c and d. Hence, the permutation $(a, b)(c, d)$ is implemented. For each pair of transpositions, the same $\pi\kappa_0\pi^{-1}$ circuit should be applied. The final circuit is a cascade of such sub-circuits.

In [9], a k-cycle-based synthesis algorithm was proposed. The authors proposed a set of cycles of lengths less than 6. The k-cycle method consists of two main parts:

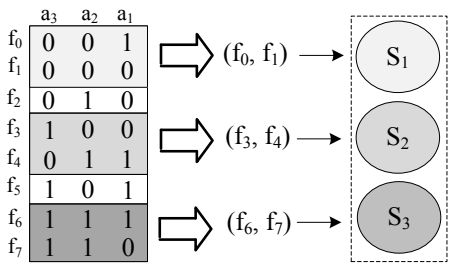

Fig. 2. An example of distinct subsets in a truth table with three variables.

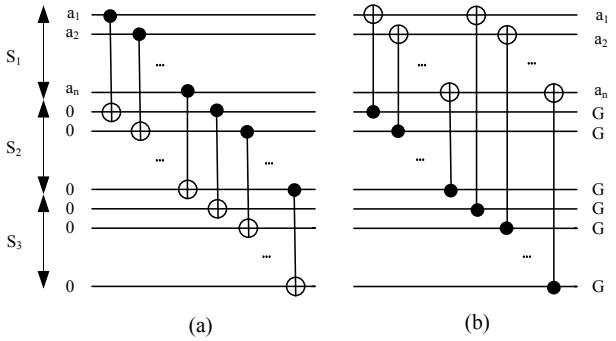

Fig. 3. The preparation circuit (a), and the terminal circuit (b) in the proposed parallel circuit structure.

- *Building Blocks Synthesis*: Direct synthesis of seven building blocks consisting of a pair of 2-cycles, a single 3-cycle, a pair of 3-cycles, a single 5-cycle, a pair of 4-cycles and a single 2-cycle (4-cycle) followed by a single 4-cycle (2-cycle) was proposed in this part. These building blocks were called *elementary cycles* in this work. For each building block, the structure $\pi\kappa_0\pi^{-1}$ was considered and a synthesis algorithm was proposed for each one.
- *Decomposition*: A given k-cycle should be decomposed into a set of elementary cycles. In this part the decomposition procedure was described and it was shown that an arbitrary permutation can be decomposed into a set of elementary cycles. The effect of decomposition on the result of cycle-based algorithm was considered in [23].

The k-cycle-based synthesis algorithm first fixes the positions of 0 and 2^i terms and then decomposes the resulted k-cycle into elementary cycles. After that, direct synthesis of each elementary cycle with $\pi\kappa_0\pi^{-1}$ structure are performed to synthesize the given permutation.

IV. Parallel Cycle-Based Synthesis method

In this section the main structure of our proposed synthesis method is introduced. The main goal of the method is to find distinct subsets in an input cycle-based specification and perform their constructions in a parallel manner. Cycle representation is used since disjoint cycles represent distinct subsets.

***Example** 1: Figure 2 shows the main idea of using parallelism in Boolean reversible circuit synthesis according to the input specification. The input specification of a reversible specification with three variables is given in truth table format. The cycle form of this permutation can be shown by (f_0, f_1), (f_3, f_4) and (f_6, f_7) transpositions. Each transposition can be synthesized independently in one subset in parallel with other transpositions.*

Each subset is synthesized using k-cycle-based method in [9] with three main differences:

- It uses a structure different from the $\pi\kappa_0\pi^{-1}$ structure (explained in subsection III).
- The input terms of κ_0 circuits are changed to improve the logical depth.
- Gates with negative control are also used in both κ_0 and π circuits for quantum cost reduction.

These three items are explained in subsection IV-A2 in more details.

The input cycles of a given specification are partitioned into three main subsets in the proposed method. The number of subsets can be extended up to the number of disjoint input cycles if sufficient qubits exist.

A. Parallel Structure

The synthesized circuit is constructed from three main parts. The first part or *preparation circuit* prepares the input data of the garbage lines. The middle circuit, which is the main part of the synthesis algorithm, is organized in three subsets and the input cycles of each subset are synthesized separately using cycle-based method. Finally, the terminal circuit takes the n outputs of the circuit from the $3n$ outputs of the structure and moves them to the first n qubits. In the rest of this section, each part of the parallel structure is discussed in more details.

1) Preparation Circuit: The preparation circuit is made by $2n$ CNOT gates with two logical levels. These gates copy the input qubits to the zero-initiated ancillae. Figure 3-a shows the preparation circuit for a circuit with n qubits and $2n$ added ancillae.

2) Middle Circuit: The middle circuit is organized in three subsets. The input cycles of each subset are synthesized using the k-cycle approach with the following remarks:

- *Sequential-cycle structure:* In the k-cycle synthesis approach proposed in [9], the $\pi\kappa_0\pi^{-1}$ structure is used. We propose a new structure in which $\pi\kappa_0$ circuits come first and then all π^{-1} circuits are placed in the reverse order after them. Note that the input terms of the second cycle (and as a result, all other cycles after that) should be computed from the applied gates in the π circuits up to that cycle. Therefore, the last Π^{-1} circuit, which is the chain of all π^{-1} circuits, may have redundant functions and may be simplified. Figure 4 shows the sequential-cycle structure in comparison to the previously used one.
- *Cycle operating interval:* In the proposed algorithm, the interval in which the cycle is implemented in κ_0 circuits

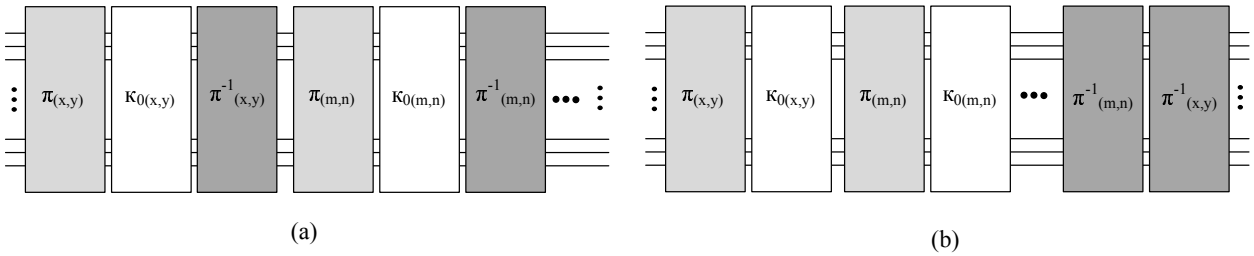

Fig. 4. The structure used in [12] and [9] (a), and sequential-cycle structure (b).

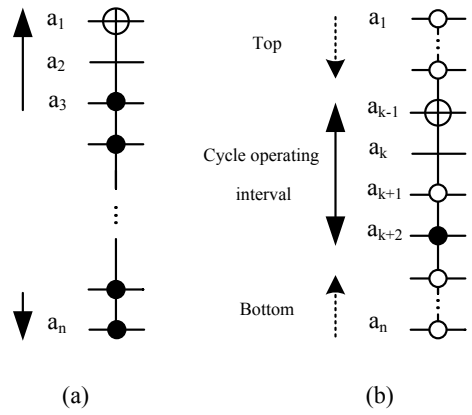

Fig. 5. The $\kappa_{0(2,2)}$ circuit in [9] (a), and the proposed $\kappa_{0(2,2)}$ circuit (b).

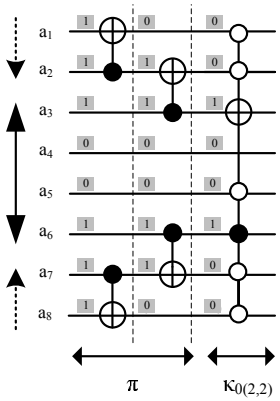

Fig. 6. An example of π circuit with depth considerations.

is limited and stands between two areas, so this chance is given to each input term to reach to the specific term in κ_0 circuit in a parallel manner from two sides. Figure 5 shows the $\kappa_{0(2,2)}$ circuit. The cycle interval and the top and bottom parts are illustrated in Fig. 5-b.

***Example** 2: For converting (11100111) to (00100100) where the second term is the specific term of κ_0 circuit, two logical levels are sufficient in the π circuit; each level with two CNOT gates as shown in Fig. 6.*

- *Negative controls:* In our elementary cycles, negative controls are used in κ_0 and also in π and π^{-1} circuit implementations to reduce the quantum cost. It has been studied in [5] that multiple-control Toffoli gates with negative controls can be simulated by elementary gates in the same manner as those with all positive controls. Therefore, negative controls are used as an abstract model and can be used in high-level synthesis. Since 2^i terms cannot be fixed before the synthesis of input cycles as in the k-cycle approach (according to the proposed parallelized structure), negative controls are needed for converting input terms to specific terms in κ_0 circuit (i.e., in some cases there is no 1 in the term to be used as positive control for CNOT or Toffoli gates in the synthesis algorithm).

3) Terminal Circuit: The two output sets of our three main subsets keep their input values and the other output set produces the original output of the circuit. The terminal circuit takes the outputs from the main three subsets using the circuit in Fig. 3-b and transfers the correct output to the first qubits. This circuit is constructed from $2n$ CNOT gates with two logical levels.

B. Elementary Cycles

In this subsection the elementary cycles which are used in our method are introduced. The differences of our internal structures with the k-cycle-based method in [9] are attributed to these elementary cycles. Input terms of κ_0 circuits for each cycle are listed below:

- Cycle$_{(2,2)}$:
  $(2^{k+2}, 2^{k+2}+2^{k-1})(2^{k+2}+2^k, 2^{k+2}+2^k+2^{k-1})$
- Cycle$_{(3)}$:
  $(2^{k-1}, 2^{k+1}+2^{k-1}, 2^{k+1}+2^k+2^{k-1})$
- Cycle$_{(3,3)}$:
  $(2^{k-1}, 2^{k+1}+2^{k-1}, 2^{k+1}+2^k+2^{k-1})$
  $(2^{k+2}+2^{k-1}, 2^{k+2}+2^{k+1}+2^{k-1}, 2^{k+2}+2^{k+1}+2^k+2^{k-1})$
- Cycle$_{(4,2)}$:
  $(2^{k+2}, 2^{k+2}+2^{k-1}, 2^{k+2}+2^k, 2^{k+2}+2^k+2^{k-1})$
  $(2^{k+2}+2^k+2^{k-1}+2^{k-2}, 2^{k+2}+2^{k-1}+2^{k-2})$

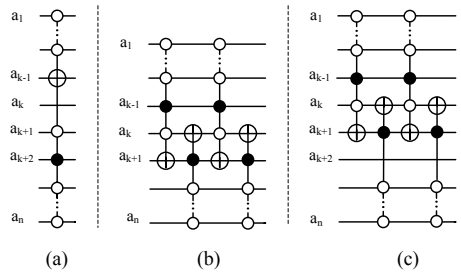

Fig. 7. The $\kappa_{0(2,2)}$, $\kappa_{0(3)}$ and $\kappa_{0(3,3)}$ circuits.

- Cycle$_{(4,4)}$:
  $(2^{k+2}, 2^{k+2}+2^{k-2}, 2^{k+2}+2^{k-1}, 2^{k+2}+2^{k-1}+2^{k-2})$
  $(2^{k+2}+2^{k}, 2^{k+2}+2^{k}+2^{k-2}, 2^{k+2}+2^{k}+2^{k-1}, 2^{k+2}+2^{k}+2^{k-1}+2^{k-2})$
- Cycle$_{(5)}$:
  $(2^{k+2}, 2^{k+2}+2^{k-1}+2^{k-2}, 2^{k+2}+2^{k-2}, 2^{k+2}+2^{k-1}, 2^{k+2}+2^{k}+2^{k-1}+2^{k-2})$
- Cycle$_{(5,5)}$:
  $(2^{k+2}, 2^{k+2}+2^{k-2}+2^{k-3}, 2^{k+2}+2^{k-3}, 2^{k+2}+2^{k-2}, 2^{k+2}+2^{k-1}+2^{k-2}+2^{k-3})(2^{k+2}+2^{k}, 2^{k+2}+2^{k}+2^{k-2}+2^{k-3}, 2^{k+2}+2^{k}+2^{k-3}, 2^{k+2}+2^{k}+2^{k-2}, 2^{k+2}+2^{k}+2^{k-1}+2^{k-2}+2^{k-3})$

Figures 7-9 show the κ_0 circuits for the elementary cycles of our synthesis algorithm. Table I summarizes the maximum quantum cost needed for implementing each elementary cycle. $Cost/length$ values are reported for the proposed method and the method in [9] in the table. It can be seen that by using negative controls in the synthesis of elementary cycles, the maximum cost of cycle implementation is reduced for each cycle.

Consider $(a_n...a_{k+2}a_{k+1}a_k a_{k-1}...a_1)$ binary representation model for each term where a_1 is the least significant variable in an n-variable function. For the input cycles $(a,b)(c,d)$, for the π circuit, at most n NOT gates are used to convert the first term a to (0...1000...0) and $3n$ CNOT gates (with at most three CNOT gates with negative control) are used to convert the changed input $\acute{b}$ to (0...1001...0), $\acute{c}$ to (0...1010...0) and $\acute{d}$ to (0...1111...0). At last, one Toffoli gate is used to convert the last term to (0...1011...0). Then, the κ_0 circuit of Cycle$_{(2,2)}$ with the quantum cost of $24n-88$ should be applied. The π^{-1} circuit, (i.e., the reverse of π circuit which is performed after κ_0 circuit), needs the same number of gates as π circuit. Totally, the maximum quantum cost which is needed for implementing $(a,b)(c,d)$ permutation is $32n-66$.

A similar approach is used for implementing all other elementary cycles. Note that the order of converting terms may be different in cycle implementation. Some gates are fixed in the procedure of term converting except for the terms that do not need any changes. For example, the last Toffoli gate in the pair of 2-cycles implementation is fixed. k is chosen to be $\lceil \frac{n}{2} \rceil$ in our implementation.

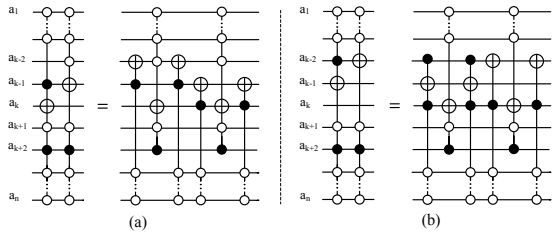

Fig. 8. The $\kappa_{0(4,2)}$ and $\kappa_{0(4,4)}$ circuits.

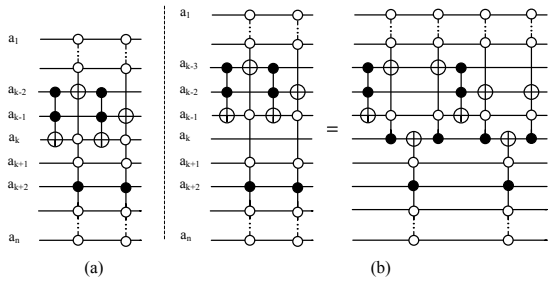

Fig. 9. The $\kappa_{0(5)}$ and $\kappa_{0(5,5)}$ circuits.

C. Decomposition Method

The goal of decomposition is to decompose an input permutation into the elementary cycles introduced in subsection IV-B. Our decomposition method follows the one which was described in [9]. The input permutation is decomposed into a set of 5, 4, 3 and 2 cycles in this method. Example 3 describes the decomposition method in more details.

***Example** 3: Consider the input permutation (5, 8, 4, 3, 15, 13, 0, 12, 10, 1, 14, 7) (2, 11) on 4 variables with two cycles of length 11 and 2 respectively. Decomposition procedure starts to cut off cycles of length 5. One element of each separated cycle should repeat in the remaining cycle to save the permutation connection. The procedure should be continued to obtain cycles of length less than 6. The resulted elementary cycles are (5, 8, 4, 3, 15) (13, 0, 12, 10, 1) (14, 7, 5, 13) (2, 11) for this input permutation.*

D. Distribution Method

Number of input cycles for our algorithm should be more than three. In other words, one cycle for each subset is necessary. More than three cycles are distributed in three subsets. For distribution, a greedy algorithm was developed which gives weight to each permutation according to $Cost/Length$ column in Table I after decomposing that permutation to its elementary cycles.

For even permutations, the algorithm considers that each subset should keep its permutation even. For odd permutations, it is enough for one subset to keep its permutation odd. Odd permutations need one more ancilla for implementation and fewer of them is preferred.

Example 4 describes the general idea of the proposed method and the resulted circuit of this example can be seen

TABLE I
MAXIMUM COST COMPARISON FOR THE PROPOSED ELEMENTARY CYCLES. THE COST/LENGTH VALUES ARE REPORTED FOR THE PROPOSED METHOD AND THE METHOD IN [9].

EC	Length	κ_0	π, π^{-1}	Total Cost	Cost/Length	Cost/Length [9]
(2,2)	4	24n-88	4n+11	32n-66	8n-16.5	8.5n-16
(3)	3	24n-88	3n+4	30n-80	10n-26.6	10.7n-27.3
(3,3)	6	24n-112	6n+26	36n-60	6n-10	6.3n-15.3
(4,2)	6	36n-180	6n+14	48n-152	8n-25.3	8.3n-20.3
(4,4)	8	36n-228	8n+46	52n-136	6.5n-17	7n-15.7
(5)	5	48n-166	5n+13	58n-140	11.6n-28	12n-26
(5,5)	10	36n-206	10n+57	56n-92	5.6n-9.2	6.4n-5.4

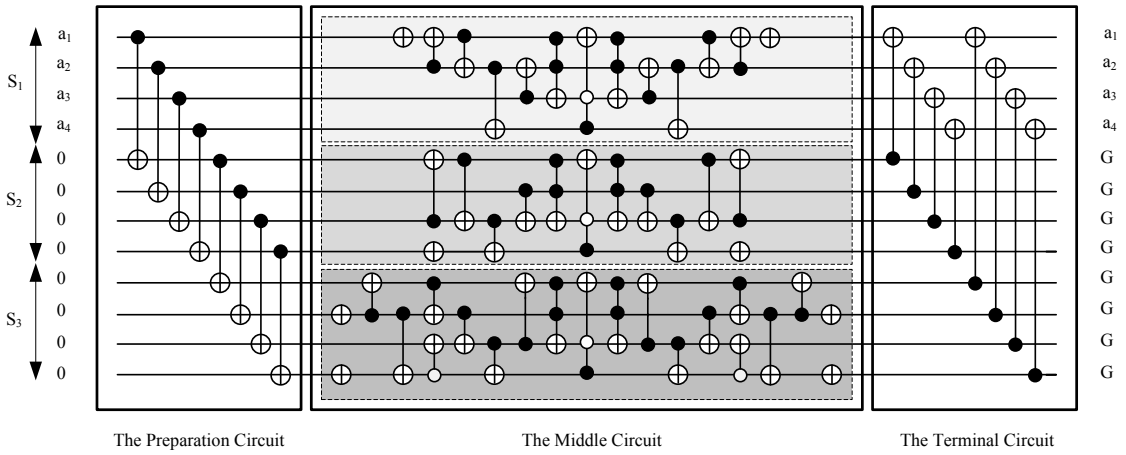

Fig. 10. An example of the proposed parallel cycle-based method for a 4-qubit function.

in Fig. 10.

Example *4: Assume that the input cycles (1,3) (7,10) (0,4) (6,15) (2,8) (5,13) are given for implementing a circuit with* 4 *qubits. Using the proposed method, each pair of 2-cycles is assigned to one subset using greedy distribution algorithm. Since the input cycles of each subset are elementary cycles, no decomposition procedure in needed in this case. Therefore, (1,3) (7,10) cycles are assigned to the first subset, (0,4) (6,15) cycles are assigned to the second subset and (2,8) (5,13) cycles are assigned to the third subset. The preparation circuit prepares the input data of the second and third subsets and each subset is synthesized independently with sequential-cycle structure for each subset. The terminal circuit is added at the end of the circuit for transferring the output of the circuit to the first* n *qubits. The depth of the middle circuit is equal to the maximum depth of the three subsets (i.e., 33). The depth of the whole circuit is equal to the depth of the middle circuit plus 4 (2 for the preparation circuit and 2 for the terminal circuit). The synthesized circuit is shown in Fig. 10.*

V. EXPERIMENTAL RESULTS

The proposed parallel cycle-based synthesis method was implemented in C++ and all of the experiments were done on an Intel Pentium IV 2.5GHz computer with 4GB memory. The quantum cost (labeled as QC) and logical depth (labeled as LD) results of the proposed cycle-based synthesis method are compared with the results of [9] in Table II. To evaluate the proposed synthesis method, reversible benchmark functions of [24] which were implemented using pure k-cycle approach [9] were attempted. The results of [5] are used for decomposing multiple-control Toffoli gates and quantum cost calculation for gates with negative control(s).

The results show that the maximum depth improvement of our method is 67% and its average is 50%. As it can be seen in the table, in some cases, the quantum cost of the proposed method is increased which is the effect of partitioning cycles into three groups as well as using 2^i terms in the synthesis process. Notice that using negative controls prevents the quantum cost to grow. For odd permutations, one more ancilla should be added in both methods as discussed in [12]. The average depth improvement of nth_prime benchmarks is less than hwb functions since the input cycles of those functions are unstructured with different cycle lengths which results in unbalanced subsets after distribution. Function nth_prime8_inc is eliminated from the list of benchmark functions since this function has one disjoint input cycle and the proposed synthesis method cannot be applied to it. It should be mentioned that no post-process quantum cost optimization algorithm is applied on the synthesized circuits.

VI. CONCLUSION

In this paper, the problem of Boolean reversible logic synthesis with logical-depth optimization was addressed. To this end, cycle representation was chosen and input cycles

TABLE II

QUANTUM COSTS (QC) AND LOGICAL DEPTHS (LD) OF THE PROPOSED PARALLEL CYCLE-BASED SYNTHESIS METHOD COMPARED WITH THOSE OF THE k-CYCLE-BASED APPROACH [9]. RUNTIME RESULTS ARE FROM A FEW SECONDS FOR $n \leq 9$ TO ≈ 1 HOUR FOR OTHER CASES.

Benchmark Function	n	Our Method			[9]		QC Impr. (%)	LD Impr. (%)
		Garbage	QC	LD	QC	LD		
hwb8	8	16	7400	2316	6940	6205	-6.6	62.6
hwb9	9	18	15376	4800	16173	14312	4.9	66.4
hwb10	10	20	38388	11787	35618	31908	-7.7	63.0
hwb11	11	22	89434	27079	90745	81440	1.4	66.7
hwb12	12	24	208992	64727	198928	184210	-5.0	64.8
hwb13	13	26	431054	131166	436305	397147	1.2	66.9
nth_prime7_inc	7	14	3108	1651	3172	2782	2.0	40.6
nth_prime9_inc	9	18	17744	15202	17975	15767	1.2	3.5
nth_prime10_inc	10	20	43026	14446	40301	35511	-6.7	59.3
nth_prime11_inc	11	22	93548	40316	95433	85093	1.9	52.6
nth_prime12_inc	12	24	217294	140507	208227	187006	-4.3	24.8
nth_prime13_inc	13	26	469422	281129	474660	431644	1.1	34.8
Average							-1.3	50.5

were partitioned into three subsets. The cycle-based synthesis method equipped with depth-consideration was used to synthesize the input cycles of each subset. The synthesized circuits in these subsets were applied in parallel. In addition, internal structures with depth-consideration were introduced in the synthesis method. Our experimental results show that the number of logical levels can be improved by up to 67% by adding $2n$ ancillae. A better distribution algorithm can improve the results in terms of quantum cost or depth.

REFERENCES

[1] M. Nielsen and I. Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2000.

[2] C. Moore and M. Nilsson. Parallel quantum computation and quantum codes. *SIAM Journal on Computing*, 31:799–815, 2001.

[3] A. Broadbent and E. Kashefi. Parallelizing quantum circuits. *Theoretical Computer Science*, 410(26), 2009.

[4] R. Jozsa. An introduction to measurement based quantum computation. *e-print, quant-ph/0508124*, 2005.

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

[6] R. V. Meter and K. M. Itoh. Fast quantum modular exponentiation. *Phys. Rev. A, Gen. Phys.*, 71(5), 2005.

[7] T. G. Draper, S. A. Kutin, E. M. Rains, and K. M. Svore. A logarithmic-depth quantum carry-lookahead adder. *e-print, quant-ph/0406142*, 2004.

[8] D. Maslov, G. W. Dueck, and D. M. Miller. Techniques for the synthesis of reversible Toffoli networks. *ACM Trans. Des. Autom. Electron. Syst.*, 12(4):42, 2007.

[9] M. Saeedi, M. Saheb Zamani, M. Sedighi, and Z. Sasanian. Reversible circuit synthesis using a cycle-based approach. *ACM Journal of Emerging Technologies in Computing Systems*, 6(4), December 2010.

[10] R. Wille and R. Drechsler. BDD-based synthesis of reversible logic for large functions. In *Design Autom. Conf.*, pages 270–275, San Francisco, CA, 2009.

[11] R. Wille, M. Soeken, and R. Drechsler. Reducing the number of lines in reversible circuits. In *Design Autom. Conf.*, pages 647–652, Anaheim, CA, 2010.

[12] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of reversible logic circuits. *IEEE Trans. on CAD*, 22(6):710–722, June 2003.

[13] P. Gupta, A. Agrawal, and N. K. Jha. An algorithm for synthesis of reversible logic circuits. *IEEE Trans. on CAD*, 25(11):2317–2330, 2006.

[14] M. Saeedi, M. Sedighi, and M. Saheb Zamani. A novel synthesis algorithm for reversible circuits. In *Design Autom. Conf.*, pages 65–68, San Jose, California, 2007.

[15] D. Große, R. Wille, G. W. Dueck, and R. Drechsler. Exact multiple control Toffoli network synthesis with SAT techniques. *IEEE Trans. on CAD*, 28(5):703–715, 2009.

[16] A. K. Prasad, V. V. Shende, K. N. Patel, I. L. Markov, and J. P. Hayes. Data structures and algorithms for simplifying reversible circuits. *J. Emerg. Technol. Comput. Syst.*, 2(4), October 2006.

[17] M. Arabzadeh, M. Saeedi, and M. Saheb Zamani. Rule-based optimization of reversible circuits. In *Asia and South Pacific Design Autom. Conf.*, pages 849–854, Taiwan, 2010.

[18] D. M. Miller, R. Wille, and R. Drechsler. Reducing reversible circuit cost by adding lines. pages 217–222, Barcelona, Spain, 2010.

[19] S. Lee, S. J. Lee, T. Kim, J. S. Lee, J. Biamonte, and M. Perkowski. The cost of quantum gate primitives. *Journal of Multiple-Valued Logic and Soft Computing*, 12(5-6), 2006.

[20] A. Barenco et al. Elementary gates for quantum computation. *APS Physical Review A*, 52:3457–3467, 1995.

[21] G. W. Dueck and D. Maslov. Reversible function synthesis with minimum garbage outputs. In *International Symposium on Representations and Methodology of Future Computing Technologies*, pages 154–161, Trier, Germany, 2003.

[22] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Design Autom. Conf.*, pages 318–323, Anaheim, CA, 2003.

[23] M. Saeedi, M. Sedighi, and M. Saheb Zamani. A library-based synthesis methodology for reversible logic. *Microelectron. J.*, 41(4):185–194, Apr 2010.

[24] D. Maslov. Reversible logic synthesis benchmarks page. *http://webhome.cs.uvic.ca/∼dmaslov*, 2011.