Oumarou, Oumarou; Paler, Alexandru; Basmadjian, Robert

Halving the width of Toffoli-based constant modular addition to n+3 qubits

Published in:
Physical Review A

DOI:
10.1103/PhysRevA.105.052436

Published: 01/05/2022

Document Version
Publisher's PDF, also known as Version of record

Please cite the original version:

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.
Halving the width of Toffoli-based constant modular addition to $n + 3$ qubits

Oumarou Oumarou,1 Alexandru Paler,2,3,4 and Robert Basmadjian1

1Institute of Informatics, Clausthal University of Technology, 38678 Clausthal-Zellerfeld, Germany
2School of Science, Aalto University, 02150 Espoo, Finland
3Electrical & Computer Engineering, University of Texas at Dallas, Richardson, Texas 75080, USA
4Computer Science, Transilvania University, 500036 Brașov, Romania

(Received 23 November 2021; accepted 10 May 2022; published 25 May 2022)

We present an arithmetic circuit performing constant modular addition having $O(n)$ depth of Toffoli gates and using a total of $n + 3$ qubits. This is an improvement by a factor of two compared to the width of the state-of-the-art Toffoli-based constant modular adder. The advantage of our adder, compared to the ones operating in the Fourier basis, is that it does not require small-angle rotations and their Clifford $+ T$ decomposition. Our circuit uses a recursive adder combined with the modular addition scheme proposed by Vedral et al. The circuit is implemented and verified exhaustively with QUANTIFY, an open-sourced framework. We also report on the Clifford $+ T$ cost of the circuit.

DOI: 10.1103/PhysRevA.105.052436

I. INTRODUCTION

Arithmetic quantum circuits, namely adders, modular adders, and multipliers, are an integral part of the implementation of practical quantum algorithms. In practice, adders, e.g., Refs. [1–4], are building blocks of more complex functions. In the Appendix we discuss quantum walks as applications of modular addition circuits.

Optimizing adders as a subcircuit can benefit the entire circuit due to the convoluted relation that exists between the arithmetic operations. A circuit for modular addition takes three integer inputs $a$, $b$, and the ring size $N$, and outputs $a + b$ or $a + b - N$ depending on whether or not $a + b < N$. In Ref. [5], the presented modular adder is composed of two comparators and an adder. The comparators in the modular adder circuit use approximately $2n$ qubits (i.e., width) but the adder only uses around $n$ qubits. Hence, the width of the entire circuit ($\sim 2n$) is dictated by the comparators.

In Ref. [6], no comparators are used, but still the width of the modular adder is even greater when compared to the aforedescribed circuit because the adder uses $3n + 1$ qubits.

Modular adders of low depth and using $O(n)$ qubits are known, but use the quantum Fourier transform (QFT) [2] approach. The QFT uses controlled rotation gates, and the angles are of the form $e^{2\pi i a/N}$, where the maximum value of $N$ is $2^n$ for addition on $n$ qubits. The rotation angles get smaller with an increasing number of qubits. When error-correcting such circuits, the controlled rotation gates have to be decomposed into Clifford $+ T$. The decomposition procedure introduces on the order of a hundred $T$ gates per rotation gate [7] (the exact number depends on the decomposition approximation precision), such that QFT modular addition is not necessarily resource efficient when error-corrected.

The main contribution of this paper is a method that performs constant modular addition using the adder from Ref. [5] while bypassing the need for comparators. Inspired by the modular addition method from Ref. [6], we combine it with the adder from Ref. [5] to yield a modular adder with a linear depth of $O(n)$ and a qubit width of $n + 3$. Consequently, the width of the constant modular addition is halved and reduced very close to its minimum of $n$ wires.

The rest of this paper is organized as follows: In Sec. II, we present the circuits used to construct our modular adder. In Sec. III, we present the design method and steps for our circuit. Finally, in Sec. IV we investigate different decompositions into Clifford $+ T$ scenarios, and analyze and compare them.

II. PRELIMINARIES

In the following, we review the construction of the recursive adder from Ref. [5], and the modular addition method from Ref. [6].

A. The incrementer and the carry gates

The incrementer (Fig. 1) is a circuit that adds one to the value of an integer stored in a quantum register. Incrementation can be achieved with the help of a quantum adder with $|a\rangle$ as the operand to be incremented, while the value of the second operand $|g\rangle$ is irrelevant (garbage register). The second register is used in order to perform a trick based on the two’s complement representation. We denote by $\bar{g}$ the bitwise negation of the number $g$. For numbers represented as two’s complement, $\bar{g} = -g - 1$, such that $g + \bar{g} = -1$. Thus, the value $a$ can be incremented by performing $a - g - \bar{g}$. In terms of a quantum circuit, the incrementation procedure is the following:

1. Subtract $g$ from $a$: $|a\rangle|g\rangle \rightarrow |a - g\rangle|g\rangle$. 

DOI: 10.1103/PhysRevA.105.052436
tion operation implemented using the inverse of an addition circuit. The input to the adder are a
bit of the sum ($a$), the incrementer of Sec. II A. The inputs to the adder are a
bit of the sum ($a$), and a garbage register (e.g., the middle two boxes). Those are on their
middle of the input integer $a$, a garbage qubit $|g\rangle$, and a classical constant $c$. It outputs the sum $|a + c\rangle |g\rangle$.

The recursive adder is an in-place adder: The construction uses the fact that the sum bit at position $m$ depends on the carry bit generated by the $m − 1$ bits before. For $m$ being the middle of the $|a\rangle$ bit string, $m = \frac{n}{2}$, and knowing the carry bit from the first half of the bits (use the carry gate), the second half of the bits can be treated as a separate number which is just incremented (use the incrementer gate). For simplicity of demonstration, we consider the garbage qubit to be initialized to $|0\rangle$. Later we present the general concept with an arbitrary value of $|g\rangle$. The addition procedure is performed as follows:

1. Split the register $|a\rangle = |a_{H}\rangle|a_{L}\rangle$ where $a_{H}$ and $a_{L}$ are respectively the higher and lower halves of the binary representation of $a$. Split also the constant $c$ in the same manner as $|a\rangle$.

2. Apply the carry gate to the $|a_{L}\rangle$ and $c_{L}$ using $|a_{H}\rangle$ as the garbage register and the garbage qubit $|g\rangle$ to store the carry bit of $a_{L} + c_{L}$.

3. Use the garbage qubit $|g\rangle$ to control whether the upper half $a_{H}$ should or should not be incremented. If $|g\rangle = |\bar{0}\rangle$, then we have a carry bit from $a_{L} + c_{L}$ and it should be added to $a_{H}$. Hence the incrementer should be applied. Otherwise, we do not apply it.

4. To reset the carry qubit to $|0\rangle$, reapply the carry gate.

5. Recursively apply the previous three steps to $a_{L}$ and $a_{H}$.

The upper part of Fig. 2 illustrates the recursion process for the case of 4-bit string $|a\rangle$. The leftmost part (e.g., the big box) represents the entire one 4-bit string which is divided into two 2-bit strings (e.g., the middle two boxes). Those are on their turn divided into four 1-bit strings (e.g., the right four small boxes). Note that the recursion stops when one $n$-bit string is subdivided into $n$ 1-bit strings.

The addition can work with arbitrary values stored in the garbage register by appending to the carry-incrementer-carry sequence in Fig. 2:

1. Left: an incrementer on $|a_{H}\rangle$ controlled by $|g\rangle$ a set of controlled-NOT (CNOT) gates also targeting $|a_{H}\rangle$ and controlled by $|g\rangle$ from the left.

2. Right: another set of CNOT gates.

This construction works because if the initial garbage is $|0\rangle$, the circuit is just as listed before. For $|g\rangle = |\bar{0}\rangle$, the first incrementer generates $|g + 1\rangle$, and the bitwise negation results in $|g\rangle = |g + 1\rangle = |\bar{g} + 1\rangle$. There are two options: (a) The first carry flips $|g\rangle$ such that the lowest bit is $|\bar{0}\rangle$, the incrementer and the second carry are not called, and the second negation returns the state to $|g + 1\rangle$; (b) the first carry does not flip $|g\rangle$, the controlled incrementer is called such that $|\bar{g} + 1\rangle$, the second carry is not called, and the final bit flips result in $|\bar{g} + 1\rangle = |g + 1\rangle = g$.

C. Modular addition

The intuitive way of constructing a modular adder is to use a comparator gate to test the sum of the two operands with the maximum value representative in the ring. Based on the comparison result, we either only add the two operands, or, in case there was an overflow, subtract $N$ from the sum. This
approach to modular addition, used by Ref. [5], requires $2n + O(1)$ wires because of the comparisons.

Compared to the intuitive approach from Ref. [5], the modular adder in Ref. [6] has a higher depth and a $4n + O(1)$ width. However, the modular addition approach is general and not tied to a particular adder design—the adder can be replaced. In Sec. III, we use in-place recursive adders for the additions from Fig. 3. Also, two of the inputs, accounting for 2n qubits, can be replaced by classical values in the case of constant addition. We successfully designed a modular adder that uses only $n + 3$ qubits which is an approximately 50% reduction compared to state-of-the-art modular adders such as in Ref. [5] while simultaneously maintaining the linearity in the depth.

III. METHODS

In Ref. [5], the comparator circuits are the culprits behind the $2n$ width of the modular addition, although the modular adder uses for the addition/subtraction only approximately half of the wires. At the same time, the modular adder from Ref. [6] does not use comparator circuits, and is agnostic of how the adders are implemented. We use a recursive adder of width $n + 2$ from Ref. [5] (in Fig. 2, $|a\rangle$ is of size $n + 1$ and there is an additional ancilla $g$ of size 1 qubit) to implement the modular addition. The advantage is that we halve the width because we eliminate the need for comparator circuits.

The comparator in Ref. [5] is implemented by applying their carry circuit, which uses $2n + O(1)$ qubits, out of which $n$ are garbage. When implementing recursive addition, the doubling of qubits is not an issue, because in the adder half of the $n$ qubits are garbage for the other half (cf. Fig. 2). However, for the comparator circuit this approach is not efficient, because the carry has to consider all $n$ qubits. Although one of the modular adder diagrams in Ref. [8] shows $n$ wires, internally the adder uses $n$ ancillas for a total width of $2n$.

It is possible to implement constant modular addition without a comparator circuit and the corresponding ancillas [6]. We implement the recursive adder using (a) the carry gate from Ref. [5], and (b) the incrementer is the linear-depth controlled adder (CtrlAdd) from Ref. [4]. The original CtrlAdd circuit has a width of $2n + 3$, and we can cut two of the ancillas because we made sure from the size of the input register $|a\rangle$ that the incrementation never overflows. The CtrlAdd circuit will be applied to only half of the bits from the recursive adder, and the other half are used as garbage [4] (see Fig. 2).

The original circuit from Ref. [6] (Fig. 3) has width $4n + O(1)$ where $3n + 1$ of which are used to store $a$, $b$, and $N$ (which were not hardwired). Implementing constant modular addition requires a single $n$-qubit quantum register, namely $|a\rangle$, while $c$ (we use $c$ instead of $b$ to highlight that it is a constant) and the size of the ring, $N$, are classical values. The modular addition is performed in the following steps (Fig. 4):

1. Add $a$ and $c$: $|a\rangle|g\rangle|0\rangle$ $\xrightarrow{\text{add}(a,c)} |a + c\rangle|g\rangle|0\rangle$.
2. Subtract $N$ from the previous sum by running the recursive adder in inverse with $a + c$ and $N$ as the integer and classical constant inputs, respectively.
3. If the flag bit of the result of the previous subtraction equals 1, then it is negative and we need to re-add $N$. Otherwise, if it is positive, we leave it as is.
4. Reset the flag qubit to its original state. Subtract $c$ from $a + c \mod N$. If it is 1 (positive), then the flag qubit should be flipped.
5. Add $c$ to the result to recover $a + c \mod N$.

After the second step the state is $|a + c - N\rangle|g\rangle|0\rangle$. The state of the most significant qubit (MSB) of $|a + c - N\rangle$ indicates whether it is positive or negative.

During the third step, if $a + c - N$ is indeed positive, then its MSB is $|0\rangle$ and $a + c - N = a + c \mod N$. On the other hand, if $a + c - N < 0$, the MSB is $|1\rangle$ and we should re-add the constant $N$. To implement both conditions in the circuit, we apply a CNOT gate between MSB and the flag qubit which is initialized to $|0\rangle$. As a result, the flag equals $|0\rangle$ if the $a + c \geq 0$ and equals $|1\rangle$ in the other case. During the third step, we hence apply the recursive adder controlled by the flag qubit adder with $a + c - N$ and $N$ as operands.

The fourth step resets the lowest qubit in Fig. 4, the flag qubit, to its original state $|0\rangle$. We subtract $c$ from $a + c \mod N$ by applying the inverse of the recursive adder. If the result is positive, the flag qubit will be flipped: Apply a CNOT between the most significant qubit of the result $a + c - c \mod N$.

Effectively, the constant modular adder using the method from Ref. [6] has the width of the adder used as a component. Using the recursive addition circuit, the total width of the resulting modular adder is $n + 3$. The input $|a\rangle$ is $n + 1$ qubits wide in order to store carry. There are also two ancillas: (1) an ancilla to control the incrementation procedures all along the recursive addition operation, and (2) the flag qubit from within the modular adder.

FIG. 3. Modular adder circuit design used in Ref. [6]. The values $a$, $b$ are the two integers and $N$ is the size of the ring. $|a\rangle$, $|b\rangle$ are quantum registers of size $n$ and $n + 1$, respectively. The addition and subtraction are performed using the adder in Ref. [6] requiring hence another quantum register of size $n$ for carry bits which is not depicted here.
IV. RESULTS AND RESOURCE ANALYSIS

The presented modular adder was implemented in QUANTIFY [9], which is open-sourced and available from Ref. [10]. We exhaustively tested the compiled Toffoli formulation of the adder using the Toffoli circuit simulator from Ref. [11].

Herein we focus on the Clifford + $T$ cost of the adder, because the Toffoli gate decomposition influences the resource efficiency of the compiled circuit. The resource analysis was implemented with QUANTIFY, too.

To determine the depth of our modular adder, we need first to determine the depth of the incrementer and the recursive adder. For the incrementer, let $D_{Tf}$ denote the depth of a decomposed Toffoli, and $A$ present the number of ancillas, then for $n > 1$ we have the following depth ($D$) and width ($W$):

$$D_{inc}(n) = (6n - 4)D_{Tf} + 2n - 4$$

$$W(n) = 2n + 1 + A.$$  

The recursive adder is built with two components, namely the incrementer and the carry gate. Unlike the incrementer, the depth of the carry gate, and consequently the recursive adder, depend on the value of the constant $c$. In the following, we will study the worst-case scenario that yields a maximum depth and which corresponds to $c$ being equal to $2^n - 1$. Using the same notation as from the previous equation, the depth of the carry gate for $n > 2$ is given by

$$D_{max}(n) = \begin{cases} 
(4n - 6)D_{Tf} + 4n - 2, & \text{if } n \geq 3, \\
4, & \text{if } n = 2, \\
1, & \text{if } n = 1.
\end{cases}$$

Hence, the depth and width of the recursive adder equal

$$D_{RA} = 2 \sum_{i=1}^{\log(n)} \left[ D_{inc}\left(\frac{n}{2^i}\right) + D_{max}\left(\frac{n}{2^i}\right) + 2 \right],$$

$$W = n + 2.$$  

Lastly, our modular adder being composed of four recursive adders, one controlled version (which has the same depth but two more Toffoli gates that replace two CNOT gates) and two CNOT gates, has the following depth,

$$D_{HBA} = 4 \times D_{RA} + D_{CRA} + 2.$$  

The adder is of the ripple-carry type, and all the Toffoli gates are sequential. When using a Toffoli decomposition (see the Appendix) which requires ancillas, only a constant number of ancillas would be needed since they can be reused for the rest of the Toffoli gates. The same fact applies for the carry gate.

V. CONCLUSION

We designed a constant modular adder that uses only $n + 3$ qubits which is a 50% reduction compared to the state-of-the-art constant modular adder from Ref. [5] while simultaneously maintaining the linearity in the depth. We conjecture that the addition method from Ref. [6] is a generalization of the incrementation trick (Sec. II A).

We implemented the constant modular adder in QUANTIFY [9], which is an open-sourced framework. Moreover, using this framework, we compiled the circuits using Toffoli gates and exhaustively verified the correctness. Future work is to use our modular addition for improving circuits from Ref. [12].

The proposed adder is useful for implementing, for example, quantum random walks or any other computation where state changes are a function of a constant. Our adder will also be useful for verifying very large quantum circuits that include constant modular addition. The size of the state vectors and the overall matrix representing the circuit is reduced to half and a quarter approximately when compared to Refs. [5,6], respectively. Such a reduction will have a quadratic and quartic speedup on the simulation of the quantum circuits.
The gate count of the recursive adder circuit is in \( O(n \log(n)) \). If gate parallelism is allowed, the depth can be reduced to \( O(n) \), with two options (cf. Fig. 2):

(i) With \( \frac{n}{2} \) ancilla: Introduce as many as necessary ancillas to use them as garbage qubits and hence parallelize the subcircuit blocks in each recursion. This would mean that \( \frac{n}{2} \) ancillas are added in total.

(ii) Without ancilla: Without loss of generality, we only execute the subcircuit blocks on the lower half \( |a_L\rangle \). We then use the qubits of the upper half \( |a_H\rangle \) as garbage qubits to parallelize the subcircuit blocks in each recursion round. Once the recursion is finished on the lower half \( |a_L\rangle \), we execute the subcircuit blocks on the upper half \( |a_H\rangle \) using \( |a_L\rangle \) as garbage qubits for the same purpose.

Without parallelism, the area (depth × width) of the adder is definitely worse than with parallelism. We have an \( O(n \log(n)) \) depth and still a linear width, even though no ancillas were introduced with 0AT3. As a result, the overall area is equivalent to \( O(n^2 \log(n)) \).

When parallelizing the adder and using 4AT1, the area scales asymptotically in \( O(n^2) \). However, with 4AT1 there is a depth ratio of \( \sim \frac{7}{9} \) compared to the third option with 0AT3 but with approximately twice as much width. As a result, when determining the area, the third decomposition scenario with the 0AT3 decomposition is better than the second alternative with the 4AT1 decomposition. Because the depth ratio \( \sim \frac{7}{9} \) is less than the width ratio \( \sim 2 \), the overall area of the adder decomposed with 4AT1 in the last scenario is then \( \sim \frac{14}{9} \times 2 = \frac{14}{9} \approx 1 \) bigger than that when 0AT3 is used.

---

1There are \( n/2 \) additions of single bits, \( n/4 \) additions of 2 bits, etc. Single-bit additions use CNOTs. Toffoli gates are used only starting with the 2-bit additions.