Estimation of Shor's Circuit for 2048-bit Integers based on Quantum Simulator

Junpei Yamaguchi[1], Masafumi Yamazaki[1], Akihiro Tabuchi[1], Takumi Honda[1], Tetsuya Izu[1], and Noboru Kunihiro[2]

[1] Fujitsu Limited, Japan
{j-yamaguchi,izu}@fujitsu.com
[2] University of Tsukuba, Japan

Abstract. Evaluating exact computational resources necessary for factoring large integers by Shor algorithm using an ideal quantum computer is difficult because simplified circuits were used in past experiments, in which qubits and gates were reduced as much as possible by using the features of the integers, though 15 and 21 were factored on quantum computers. In this paper, we implement Shor algorithm for general composite numbers, and factored 96 RSA-type composite numbers up to 9-bit using a quantum computer simulator. In the largest case, $N = 511$ was factored within 2 hours. Then, based on these experiments, we estimate the number of gates and the depth of Shor's quantum circuits for factoring 1024-bit and 2048-bit integers. In our estimation, Shor's quantum circuit for factoring 1024-bit integers requires 2.78×10^{11} gates, and with depth 2.24×10^{11}, while 2.23×10^{12} gates, and with depth 1.80×10^{12} for 2048-bit integers.

Keywords: Shor algorithm, integer factorization, quantum computer, quantum computer simulator

1 Introduction

The security of RSA, one of the standardized public key cryptosystems, is based on the difficulty of the integer factorization problem of large composite numbers. The current factoring record by a classical computer is the factorization of an 829-bit composite number [6], so that RSA with larger than 2048-bit integer is considered as secure for now. On the other hand, it is known that the integer factorization problem can be solved in polynomial time by Shor algorithm by using an ideal quantum computer [18]. Some factorization experiments on quantum computers have been reported for only $N = 15$ and $N = 21$ [14,13,17,15,16,1] because of the difficulty of realizing ideal quantum computers – free from the limitation of the number of quantum bits (qubits) and the noise on qubits. To make things worse, these experiments used the simplified Shor's circuits in which qubits and gates are reduced as much as possible by using the features of the integers and their factors. In spite of some efforts for estimating circuit resources for factoring 2048-bit integers with noisy qubits [8,9], it is too difficult to give exact estimations for factoring such large integers.

There are two major problems to break the situation. The first problem is the lack of computing resource, especially the number of qubits available on quantum computers. Though IBM developed a 433-qubit processor recently [10], because of the effects of the noise, still processing Shor algorithm is hard even for such computers. The second problem to solve is the lack of experimental results of Shor algorithm. In order to estimate the circuit resources for factoring 2048-bit integers, we need more experimental results on the same computational environments.

Contribution of This Paper

This paper has three contributions. Firstly, we implemented Shor algorithm applicable to general composite numbers based on known technique [12,4], which on input N, generates Shor's quantum circuit for factoring N. The dominant circuit is Mod-EXP which computes a modular exponentiation $f_{a,N}(x) = a^x \bmod N$. Mod-EXP can be constructed from some ADD circuits, and there are three ways for constructing ADD circuits: R-ADD, GT-ADD, and Q-ADD. We implemented these all ADD circuits for Shor algorithm based on known technique [12,4] with some bug-fixes and improvements, and compared required resources.

Secondly, we made a large scale of factorization experiments on a quantum computer simulator proceeded on a supercomputer. We used the quantum simulator mpiQulacs developed by Fujitsu [11], a State Vector (SV) type simulator which records all qubit status on memory with no noise and enables to simulate an ideal quantum computation [11]. In the experiments, we succeeded in factoring 96 RSA-type composite numbers up to 9-bit. In the largest case, $N = 511$ was factored within 2 hours.

Finally, we generated some quantum circuits for $n = 8, \ldots, 25$, and evaluated resources including the number of qubits, the number of elementary gates and the depth of the gate. Based on these data, we estimated the circuit resources required for factoring 1024-bit and 2048-bit integers. In our estimation, Shor's quantum gate for 1024-bit integers requires 5121 qubits, 2.78×10^{11} gates, and the depth with 2.24×10^{11} with R-ADD, while 2048-bit requires 10241 qubits, 2.22×10^{12} gates, and the depth with 1.79×10^{12} with R-ADD. Unfortunately, these circuits are too huge to proceed on a real quantum computer at the moment so that RSA with these integer is secure by the current technology.

The rest of the paper is organized as follows: Section 2 describes how to construct Shor's quantum circuit, especially the modular exponentiation circuit Mod-EXP using ADD circuits. Then, in section 3, concrete constructions of Mod-EXP from R-ADD, GT-ADD, and Q-ADD are explained. Section 4 summarizes factoring experiments by Shor's quantum circuit using the quantum computer simulator including the estimation for 1024/2048-bit integers.

2 Quantum Circuit of Shor Algorithm

This section describes quantum circuits of Shor algorithm for general composite numbers based on known technique [12,4].

In this paper, we consider the following quantum gates as the elementary gates for evaluating number of gates and the depth:

- 1 quantum gates including the Hadamard gate, NOT gate, the rotation gate, and the phase-shift gate,
- Controlled NOT (C-NOT) gate,
- Toffoli (C^2-NOT) gate.

2.1 Shor Algorithm and Factorization

Suppose we want to factor an n-bit composite integer N. For an integer a coprime to N, the order of a with regard to N is defined as the smallest positive integer such that $a^r \equiv 1 \pmod{N}$. In 1994, Shor proposed a quantum algorithm to compute the order r of a with regard to N in polynomial time [18]. The integer N can be factored by using Shor algorithm in the following way:

i) Choose an integer a from $\{2, \ldots, N-1\}$. If $\gcd(a, N) \neq 1$ then output $\gcd(a, N)$ and terminates (since a factor of N larger than 1 is found).
ii) Compute the order r from a, N by using Shor's quantum algorithm.
iii) If r is even, $a^{r/2}+1 \not\equiv 0 \pmod{N}$ and $\gcd(a^{r/2}\pm 1, N) \neq 1$, output $\gcd(a^{r/2}\pm 1, N)$ and terminates. Otherwise, return step i).

Note that step i) and iii) can be proceeded by classical computers. On the other hand, step ii) can be computed by Shor algorithm on a quantum computer in the following way:

1. Generate an initial state $|\phi_0\rangle = |\underbrace{0 \ldots 0}_{m}\rangle\, |\underbrace{0 \ldots 0}_{n-1} 1\rangle$, where the 1st qubit sequence has m qubits, while the 2nd qubit sequence has n qubits.
2. Apply the Hadamard operation H_m to the 1st sequence:
$$|\phi_1\rangle = H_m(|\phi_0\rangle) = \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} |x\rangle\, |\underbrace{0 \ldots 0}_{n-1} 1\rangle\,.$$
3. Apply the operation $U_{f_{a,N}}$ which corresponds to a modular exponentiation $f_{a,N}(x) = a^x \bmod N$, to the 2nd sequence:
$$|\phi_2\rangle = U_{f_{a,N}}(|\phi_1\rangle) = \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} |x\rangle\, |f_{a,N}(x)\rangle\,.$$
4. Apply the inverse quantum Fourier operation to the 1st sequence.
5. Observe the 1st sequence. Then the approximation of a multiple of $2^m/r$ is obtained.
6. Repeat step 1–5 until r can be estimated.

The parameter m is determined from the approximation precision in step 5. $m = 2n$ is used usually and in this paper.

2.2 Construction of Mod-EXP from ADD

Above steps except step 3. can be realized by elementary gates easily. On the other hand, step 3. requires complex circuits called Mod-EXP [12]. This subsection describes how to realize Mod-EXP from elementary gates. In fact, Mod-EXP can be constructed from ADD circuits, by transforming Mod-EXP to the following circuits step-by-step:

- Mod-EXP$(a) : |x\rangle |1\rangle \to |x\rangle |a^x \bmod N\rangle$
- Mod-MUL$(d) : |y\rangle \to |dy \bmod N\rangle$
- Mod-PS$(d) : |y\rangle |t\rangle \to |y\rangle |t + dy \bmod N\rangle$
- Mod-ADD$(d) : |y\rangle \to |y + d \bmod N\rangle$
- ADD$(d) : |y\rangle \to |y + d\rangle$

Construction of Mod-EXP from Mod-MUL For an exponent x represented in binary, namely, $x = \sum_{i=0}^{m-1} 2^i x_i$, a modular exponentiation Mod-EXP(a) is computed by a repetition of multiplying $d_i = a^{2^i} \bmod N$ and taking modulus by N, since $a^x \bmod N = a^{\sum_{i=0}^{m-1} 2^i x_i} \bmod N = \prod_{i=0}^{m-1} a^{2^i x_i} \bmod N$. In other words, Mod-EXP(a) can be computed by a repetition of the modular multiplication Mod-MUL(d_i) controlled by $|x_i\rangle$, so that Mod-EXP(a) requires m controlled-Mod-MULs, which is denoted by $C(x_i)$-Mod-MUL.

Construction of Mod-MUL from Mod-PS The modular multiplication Mod-MUL(d) for an integer $0 \le d < N$ and an n-bit integer y can be computed by using modular product-sums Mod-PSs in the following way:

$$\begin{aligned} |y\rangle |\underbrace{0 \dots 0}_{n}\rangle &\overset{\text{Mod-PS}(d)}{\to} |y\rangle |0 + dy \bmod N\rangle \overset{\text{SWAP}}{\to} |dy \bmod N\rangle |y\rangle \\ &\overset{\text{Mod-PS}(-d^{-1})}{\to} |dy \bmod N\rangle |y + (-d^{-1})(dy \bmod N) \bmod N\rangle \\ &= |dy \bmod N\rangle |0\rangle . \end{aligned}$$

Since $d_i = a^{2^i} \bmod N$ and $\gcd(a, N) = 1$, there always exists the inverse $d^{-1} \bmod N$. Thus, Mod-MUL can be computed by two Mod-PSs, one n-qubit SWAP with auxiliary qubits $|R_2\rangle = |\underbrace{0 \dots 0}_{n}\rangle$. Especially, $C(x_i)$-Mod-MUL requires two $C(x_i)$-Mod-PSs and one n-qubit C-SWAP. Moreover, an n-qubit C-SWAP can be realized by n 1-qubit C-SWAPs, and one 1-qubit C-SWAP can be realized by two C-NOTs and one Toffoli gate.

Construction of Mod-PS from Mod-ADD When the 2nd sequence is represented as $|y\rangle = |y_{n-1} \dots y_0\rangle$, for an integer $0 \le d < N$, we have $dy = \sum_{j=0}^{n-1} d2^j y_j$. Thus, a modular product-sum Mod-PS(d) on a bit sequence $|R_2\rangle$ can be computed by a repetition $R_2 \leftarrow R_2 + d2^j \bmod N$ if $y_j = 1$ for $j = 0, 1, \dots, n-1$ which is equivalent to $C(y_j)$-Mod-ADD$(d2^j \bmod N)$. That is, Mod-PS can be realized by n 1-controlled Mod-ADDs, and $C(x_i)$-Mod-PS can be realized by n 2-controlled Mod-ADD, namely, $C(x_i, y_j)$-Mod-ADDs.

Construction of Mod-ADD from ADD There are two constructions, Type 1 and Type 2 for realizing $C(x_i, y_j)$-Mod-ADD [12]. From the efficiency point of view, Type 2 is suitable for R-ADD and Q-ADD, while Type 1 for GT-ADD. Because of the space limitation, we omit describing the detail.

2.3 Construction of ADD

This subsection describe how to construct ADD circuits from elementary gates in three ways: R-ADD, GT-ADD, and Q-ADD. Here, for an n-qubit register $|R_2\rangle = |R_{2,n-1} \dots R_{2,0}\rangle$, we consider the circuit to add an n-bit integer $p = p_{n-1} \dots p_0$. Considering the carry-over, the result is represented by $|R_1 R_2\rangle$ with 1-qubit register $|R_1\rangle$. All ADD circuits use another 1-qubit register $|R_3\rangle$, and R-ADD uses further $(n-1)$-qubit sequence $|c\rangle$. In total, GT-ADD and Q-ADD require $m+n+1+n+1 = m+2n+2 = 4n+2$ qubits, while R-ADD requires $m+2n+2+(n-1) = m+3n+1 = 5n+1$ qubits. On the other hand, the number of elementary gates is estimated by $270n^3$ for R-ADD, $16/3n^5$ for GT-ADD, and $97n^4$ for Q-ADD [12].

Construction of R-ADD R-ADD is a ripple carry adder [5,21], which computes $R_2 + p$ by using the following addition table (1):

$$\begin{array}{rccccccc} & c_{n-1} & c_{n-2} & \dots & c_1 & & \\ & R_{2,n-1} & R_{2,n-2} & \dots & R_{2,1} & R_{2,0} & \\ +) & p_{n-1} & p_{n-2} & \dots & p_1 & p_0 & \\ \hline R_1 & R_{2,n-1} & R_{2,n-2} & \dots & R_{2,1} & R_{2,0} & \end{array} \tag{1}$$

Here, $c = c_{n-1} \dots c_1$ is an auxiliary $(n-1)$-bit registers with initial value 0, which is used for storing carry-overs. R-ADD consists of three circuits, CARRY (for computing carry bits), SUM (for computing additions), and CARRY^{-1} (for resetting carry bits). As in Figure 2 of Vedral, Barenco and Ekert's paper [21], R-ADD firstly computes all carry-overs for $k = 0, 1, \dots, n-1$ by using CARRY circuit described in Figure 1 (set $c_n = R_1$ when $k = n-1$). When $p_{n-1} = 1$, apply the NOT gate to $R_{2,n-1}$. Finally, for $k = n-1, \dots, 0$, update $R_{2,k}$ by using SUM circuit described in Figure 2 and reset c_k by using CARRY^{-1} circuit, which is CARRY circuit. When $k = 0$, CARRY^{-1} is omitted. Thus, R-ADD can be constructed from Toffoli gates, C-NOT gates, and NOT gates.

Type 2 Mod-ADD requires 1-controlled R-ADD and 2-controlled R-ADD, which require not only C-NOT gate and Toffoli gate, but 3-controlled NOT and 4-controlled NOT gates. Barenco et al. showed two conversions from a C^k-NOT gate to Toffoli gates [3]. The first conversion converts a C^k-NOT gate to $2k-3$ Toffoli gates by using $k-2$ clean auxiliary qubits (qubits with their state known to be $|0\rangle$). The second converts a C^k-NOT gate to $4k-8$ Toffoli gates by using $k-2$ dirty (unclean) auxiliary qubits. Both auxiliary qubits return the initial state after the usage. According to Kunihiro's paper [12], the first conversion can convert all C^k-NOT gates.

(a) $p_k = 0$ **(b)** $p_k = 1$

Fig. 1: CARRY Circuit

(a) $p_k = 0$ **(b)** $p_k = 1$

Fig. 2: SUM Circuit

Construction of GT-ADD For $k = 0, 1, \ldots, n-1$, GT-ADD adds p by repeatedly computing $R_2 \leftarrow R_2 + 2^k$ when $p_k = 1$. An addition by 2^k can be realized by C^ℓ-NOT gates ($1 \leq \ell \leq n-k$) and NOT gates as in Figure 3. Type 1 Mod-ADD requires, in addition to GT-ADD, 1/2/3-controlled GT-ADD, that is NOT gates, C-NOT gates, Toffoli gates, and C$^\ell$-NOT gates ($3 \leq \ell \leq n+3$). Both conversions described in 2.3 can be used in RT-ADD, however, since it is difficult to use clean qubits, Kunihiro used the second conversion [12].

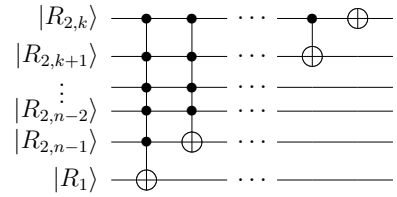

Fig. 3: Adder 2^k to $|R_2\rangle$

Construction of Q-ADD Q-ADD is an adder by using the quantum Fourier Transform (QFT) [4,7]. For simplicity, we set $|R_{2,n}\rangle := |R_1\rangle$ and assume that $|R_2\rangle$ has $n+1$ qubits in this subsection. Also set $p_n = 0$. Different from R-ADD or GT-ADD, Q-ADD computes $|R_2\rangle \leftarrow |R_2 + p \bmod 2^{n+1}\rangle$. Denote the state after applying QFT to the register $|R_2\rangle$ (Figure 9 in [4]) as $\phi(|R_2\rangle)$. Then, Q-ADD computes, for $j = n, n-1, \ldots, 0$, in the following way: for $k = 1, 2, \ldots, j+1$, apply the Z-rotation gate

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}$$

to $\phi(|R_{2,j}\rangle)$ when $p_{j-k+1} = 1$. Inverse Quantum Fourier Transform (QFT^{-1}) is required to obtain the result of the addition. Thus, Q-ADD can be realized by rotation gates only.

Type 2 Mod-Add requires 1/2-controlled Q-ADDs, that is, 1/2-controlled R_k gates are required. Here, 1-controlled R_k gate can be realized by 2 C-NOTs and 4 1-qubit gates, and 2-controlled R_k gate can be realized by 6 C-NOTs and 8 1-qubit gates [3]. The estimated number of gates of Mod-EXP is about $97n^4$ on average.

Construction of 1/2-controlled R_k is as follows. Arbitrary unitary matrix W can be represented by

$$W = \Phi(\delta)Rz(\alpha)Ry(\theta)Rz(\beta) \tag{2}$$

for parameters $\alpha, \beta, \theta, \delta \in [0, 2\pi]$, where

$$\Phi(x) = \begin{pmatrix} e^{ix} & 0 \\ 0 & e^{ix} \end{pmatrix},\ Ry(x) = \begin{pmatrix} \cos x/2 & \sin x/2 \\ -\sin x/2 & \cos x/2 \end{pmatrix},\ Rz(x) = \begin{pmatrix} e^{ix/2} & 0 \\ 0 & e^{-ix/2} \end{pmatrix}.$$

Then 1-controlled W gate can be represented as in Figure 4, where

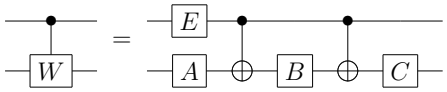

Fig. 4: Conversion of 1-controlled R_k gate

$$A = Rz(\alpha)Ry(\theta/2), B = Ry(-\theta/2)Rz(-(\alpha+\beta/2)),$$
$$C = Rz((\beta-\alpha)/2), E = Rz(-\delta)\Phi(\delta/2).$$

Thus, for $W = R_k$, 1-controlled R_k can be realized by 2 C-NOTs and 4 1-qubit gates as in Figure 4 by determining parameters $\alpha, \beta, \theta, \delta$. Similarly, by applying Lemma 6.1 in [3], 2-controlled R_k gate can be realized by 6 C-NOTs and 8 1-qubit gates.

2.4 Required Resources

Let us summarize the required resources required in Shor's circuit to factor an n-bit integer.

Shor's circuit has three main circuits, Hadamard, Mod-EXP, and QFT^{-1}. Required number of gates for each of Hadamard and QFT^{-1} is $O(n)$, while Mod-EXP requires $G_{\mathrm{ModEXP}}(\text{R-ADD}) = 270n^3$ with R-ADD, $G_{\mathrm{ModEXP}}(\text{GT-ADD}) = 16/3n^5$ with GT-ADD, and $G_{\mathrm{ModEXP}}(\text{Q-ADD}) = 97n^4$ with Q-ADD. Therefore, required number of gates for Shor's circuit can be identified by these numbers: $G_{\mathrm{Shor}}(\text{R-ADD}) = 270n^3$, $G_{\mathrm{Shor}}(\text{GT-ADD}) = 16/3n^5$, and $G_{\mathrm{Shor}}(\text{Q-ADD}) = 97n^4$. Unfortunately, no estimation for the depth is known,

Required numbers of qubts are $Q_{\mathrm{Shor}}(\text{R-ADD}) = 5n + 1$ with R-ADD, and $Q_{\mathrm{Shor}}(\text{GT-ADD} = Q_{\mathrm{Shor}}(\text{Q-ADD}) = 4n+2$ with GT-ADD and Q-ADD. Though Takahashi and Kunihiro proposed a reduced Shor circuit with $2n+2$ qubits [20], since extra device is required in the circut and the number of gates remains in the same order, so we do not use this construction in this paper.

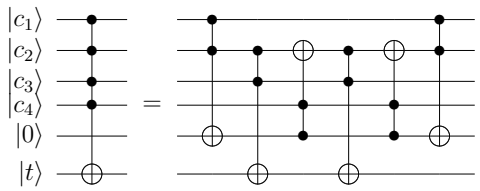

Fig. 5: Conversion from a C^4-NOT to C^2-NOTs with 1 clean qubit

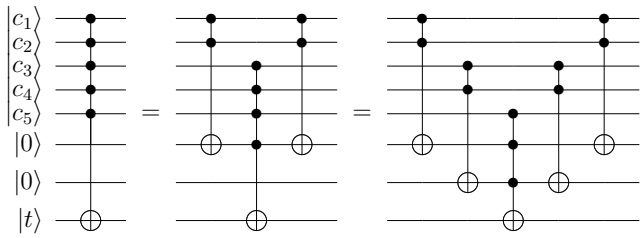

Fig. 6: Conversion from a C^5-NOT to C^2-NOTs with 2 clean qubits

3 Implementation of Shor's Quantum Circuit

This section describes how to implement Mod-EXP with each of R-ADD, GT-ADD, and Q-ADD, respectively, based on Kunihiro's paper [12]. We also show bug-fixes and improvements from them.

3.1 Mod-EXP with R-ADD

For implementing Mod-EXP with R-ADD, we use Type 2 Mod-ADD in order to minimize the number of gates. We also apply the following bug-fixes and improvements.

Bug-fix on Converting C^3-NOT, C^4-NOT to Toffoli Gate For all C^k-NOT ($k = 3, 4$) gates used in 1/2-controlled R-ADD, the first conversion described in section 2.3 converts them to $2k - 3$ Toffoli gates with $k - 2$ clean auxiliary qubits. In practice, $k - 2$ clean qubits are not available in some cases. In such cases we propose to take the following procedures. When $k = 3$ and no clean qubit is available, then use the second conversion described in section 2.3. When $k = 4$, use the second conversion described in section 2.3 if no qubit is available, and use the conversion described in Figure 5 if 1-qubit is available. Compared to the first conversion, 1 Toffoli gate is increased when $k = 3$, and 3/1 Toffoli gates are increased when $k = 4$ with 0/1 clean qubit. Though this increases the number of Mod-EXP gate, since it is at most $O(n)$ (explained later), so that there is no effect on the total number and its coefficient.

Effect by the Bug-fix In step 2 of Shor algorithm, we apply the Hadamard gate to the m-qubit sequence. By changing this operation to applying the Hadamard gate to x_i just before each $C(x_i)$-Mod-MUL, $x_{i+1}, \ldots, x_{m-1}$ can be used as clean qubits in $C(x_i)$-Mod-MUL. Thus, for $i = 0, \ldots, m-3$, x_{i+1}, x_{i+2} can be used as clean qubits and there is no increase on the number of gates.

On the other hand, when $i = m-2, m-1$, available clean qubits are less than 2 and additional circuits are required. When $i = m-1$, CARRY and CARRY^{-1} for $c_1, \ldots, c_{n-3}$ and SUM for $R_{2,0}, \ldots, R_{2,n-2}$ can use c_{n-2} and c_{n-1} as clean qubits so that no additional circuits are required. Clean qubits may not be available in at most 6 circuits: CARRY and CARRY^{-1} for c_{n-2}, c_{n-1}, CARRY for R_1, and SUM for $R_{2,n-1}$. Thus, the increased number of gates in C^k-NOT is constant. and that in controlled R-ADD is also constant. When $i = m-2$, the increased number of gates is constant similarly. Since Mod-ADD is applied $2n$ times in Mod-EXP, the total increased number of gates is $O(2cn) = O(n)$, which is negligible.

Clean Qubits Management It is difficult to figure out which qubit is clean or not manually when C^k-NOT conversion is required. So we implemented the management function to automatically list the status of auxiliary qubits.

- When a quantum gate is added to the circuit, set the status of the target qubit of the gate as 'dirty' (not clean). If the gate makes the status clean (such as CARRY^{-1}), set 'clean'.
- Use 'clean' qubits in C^k-NOT conversion.

This management makes the number of gates of Mod-EXP minimum.

3.2 Mod-EXP with GT-ADD

For implementing Mod-EXP with GT-ADD, Type 1 Mod-ADD is used to minimize the number of gates. Kunihiro used the second coversion described in section 2.3 for converting C^k-NOT gates (for $3 \leq k \leq n+3$) to Toffoli gates. This paper proposes to use clean qubits as much as possible by a greedy approach to decrease the number ofgates.

Greedy Method Suppose $1 \leq c \leq k-3$ clean qubits and sufficient dirty qubits are available in the conversion of a C^k-NOT gate to Toffoli gates. Our greedy method proceeds as follows:

1. Generate a null circuit `circ`.
2. Set X be a set of indexes of k control qubits.
3. Select two indexes from X, and delete these indexes from X.
4. Select one clean bit with changing its status as 'dirty' in clean qubit management, and add its index to X.
5. Generate a 2-controlled Toffoli gate, controlled by selected indexed-qubits and targeted to the selected clean qubit.

6. Add the generated Toffoli gate to `circ`.
7. Repeat 2. to 6. c times.
8. Generate a C^{k-c}-NOT gate controlled by $(k-c)$ indexes in X, and targeted to the same qubit as the original C^k-NOT gate, and convert to $4(k-c)-8$ Toffoli gates by using the 2nd conversion, and add to `circ`.
9. Add all Toffoli gates generated in step 2. to 7. in the reversed order to`circ`.
10. Output `circ`.

The number of Toffoli gates generated by the above greedy method is $c+4(k-c)-8+c = 4k-8-2c$. Figure 5 shows an example with $k=4, c=1$, while Figure 6 with $k=5, c=1,2$.

Greedy Method in Mod-EXP For all conversions from C^k-NOT gates ($3 \leq k \leq n+3$) to Toffoli gates appeared in Mod-EXP with GT-ADD, we use the 1st conversion described in subsection 2.3 when more than $k-2$ clean qubits are available, the greedy method when 1 to $k-3$ clean qubits are available, and the 2nd conversion when no clean qubit is available. Since GT-ADD uses $4n+2$ qubits in total and $k+1$ qubits are used in the greedy method, the rest $4n+2-(k+1)$ qubits are available as 'dirty' qubits. We also use the clean qubit management described in subsection 3.1 in the greedy method,

3.3 Mod-EXP with Q-ADD

Bug-fix in Q-ADD Since Q-ADD requires to apply QFT to the registers $|R_1 R_2\rangle$, QFT just before $C(x_0)$-Mod-MUL in Q-ADD (Figure 2 in [4]), and QFT^{-1} just before C-SWAP and QFT just before C-SWAP in $C(x_i)$-Mod-MUL should be added. Thus the number of gates are increased to $4n+2$ QFT for Mod-EXP from the original [12]. Furthermore, the original number of gates did not consider C-SWAP, so that mn Toffoli and $2mn$ C-NOT should be added. However, since these increase is at most $O(n^3)$, it does not effect on the total number of Mod-EXP at all.

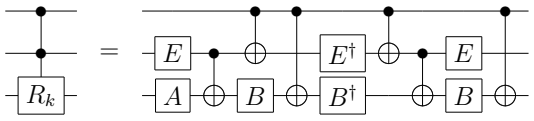

Fig. 7: Conversion of 2-controlled Rotation Gate

Change of Mod-ADD When Type 2 Mod-ADD is used for Q-ADD, 4 QFTs and 4 QFT^{-1}s are required, and the number of gates of Mod-EXP will be increased (the order is same, but the coefficient becomes larger). So, we propose to use Beauregard's Mod-ADD which requires 2 QFTs and 2 QFT^{-1}s [4].

			R-ADD				GT-ADD				Q-ADD			
N	n	a	Q	G	D	T	Q	G	D	T	Q	G	D	T
15	4	2	21	12937	10507	2.4	18	12595	9838	0.91	18	38967	20208	3.5
21	5	2	26	26155	20779	89.9	22	25325	18824	5.2	22	78334	40409	18.0
33	6	5	31	46935	36870	–	26	44461	31436	92	26	145620	76578	404
35	6	2	31	47662	37775	–	26	55387	38869	115	26	155329	79693	426
39	6	2	31	47843	38214	–	26	61941	43483	129	26	160315	81152	441
51	6	2	31	46991	37413	–	26	55755	39348	116	26	152468	78285	421
55	6	2	31	47845	38513	–	26	61899	43507	129	26	160613	80877	441
57	6	5	31	47555	38028	–	26	51360	36346	107	26	154085	78686	431
65	7	3	36	76341	59902	–	30	82676	56199	2430	30	251424	132329	10545
69	7	2	36	78035	61939	–	30	98774	66690	2866	30	271832	138888	11329
77	7	2	36	77066	61391	–	30	104285	70616	3033	30	267042	135177	11125
85	7	2	36	75704	60041	–	30	99407	67570	2906	30	256625	132179	10719
87	7	2	36	78196	62751	–	30	120027	80999	3485	30	284083	142164	11792
91	7	2	36	77819	62369	–	30	116234	78729	3369	30	279204	141000	11594
93	7	2	36	77659	62319	–	30	108070	73227	3150	30	276912	140313	11516
95	7	2	36	78550	63480	–	30	125960	85061	3664	30	289797	144364	12098
111	7	2	36	78692	63633	–	30	124959	84533	3646	30	289793	144261	12020
115	7	2	36	78591	63151	–	30	109922	74503	3188	30	282238	141557	11809
119	7	2	36	78563	63477	–	30	122960	83264	3577	30	287020	142555	11978
123	7	2	36	78691	63672	–	30	118337	80519	3452	30	286730	143475	11899

Table 1: Factorization of N up to 7-bit (with 1-node)

Gate Reduction of Controlled Rotation Gate Conversion When 1/2-controlled R_k gates are converted to elementary gates, one 1-qubit quantum gate can be reduced by setting parameters properly. In fact, in $W = R_k$ in (2), set $\alpha = \beta = -\pi/2^k,\ \theta = 0,\ \delta = \pi/2^k$. Then, C becomes an identity matirix and can be omitted. Similarly, in 2-controlled R_k gates, setting $\alpha = \beta = -\pi/2^{k-1},\ \theta = 0,\ \delta = \pi/2^{k-1}$ reduces one 1-qubit quantum gate as in Figure 7, where † denotes an inversion.

4 Experimental Results

This section reports our factorization results based on our implementation described in section 3 by using the quantum computer simulator mpiQulacs [11], a distributed version of the quantum simulator Qulacs [19]. We used an A64FX-based cluster system similar to *Todoroki* [11] with 512 nodes, which enables 39-qubit operations. A64FX is an ARM-based CPU that is also equipped in the world's top Fugaku supercomputer.

The experiments were conducted by the following steps:

1. For an n-bit RSA-type composite number (a product of two different odd primes) N, choose the integer a which induces the factorization (for efficiency reason).

2. Generate the quantum circuit for factoring N by Shor algorithm. Here we have three choices for ADD circuit.
3. Input the quantum circuit to the simulator.
4. Observe the 1st bit sequence 10,000 times to estimate the order r.
5. Output $\gcd(a^{r/2} \pm 1, N)$.

Note that, since the observation in step 4. does not destroy the quantum state, it is enough to proceed each quantum circuit once in the experiments.

4.1 Naive Circuit

Firstly, we factored small RSA-type composite numbers, a product of two different odd primes, up to 7-bit with 1-node by using Shor's quantum circuits generated by our implementation. Table 1 shows the required resources and timings for factorization, where Q, G, D, T denote the number of required qubits, the number of elementary gates, the depth of Shor's circuit, and the timing data in seconds. Since we used 1-node only, 30 qubits are available for factorization. Thus, circuits with R-ADD for 6-bit and 7-bit integers cannot be proceeded (denoted by '–' in the table).

GT-ADD				No Greedy			Greedy			Ratio		
N	n	a	Q	G_0	D_0	T_0	G_1	D_1	T_1	G_1/G_0	D_1/D_0	T_1/T_0
15	4	2	18	17881	17763	1.5	12595	9838	0.91	0.71	0.56	0.61
21	5	2	22	37044	36867	10.1	25325	18824	5.2	0.69	0.52	0.52
33	6	5	26	66679	66433	227	44461	31436	92	0.67	0.48	0.41
35	6	2	26	83216	82966	282	55387	38869	115	0.67	0.47	0.41
39	6	2	26	93136	92886	315	61941	43483	129	0.67	0.47	0.41
51	6	2	26	83790	83541	285	55755	39348	116	0.67	0.48	0.41
55	6	2	26	93156	92906	315	61899	43507	129	0.67	0.47	0.41
57	6	5	26	77400	77151	262	51360	36346	107	0.67	0.48	0.41
65	7	3	30	126462	126133	6814	82676	56199	2430	0.66	0.45	0.36
69	7	2	30	151490	151157	8121	98774	66690	2866	0.66	0.45	0.36
77	7	2	30	159842	159509	8546	104285	70616	3033	0.66	0.45	0.36
85	7	2	30	152208	151875	8165	99407	67570	2906	0.66	0.45	0.36
87	7	2	30	183909	183575	9864	120027	80999	3485	0.66	0.45	0.36
91	7	2	30	178045	177711	9537	116234	78729	3369	0.66	0.45	0.36
93	7	2	30	165750	165417	8857	108070	73227	3150	0.66	0.45	0.36
95	7	2	30	193219	192885	10358	125960	85061	3664	0.66	0.45	0.36
111	7	2	30	191313	190979	10257	124959	84533	3646	0.66	0.45	0.36
115	7	2	30	168479	168145	9048	109922	74503	3188	0.66	0.45	0.36
119	7	2	30	188369	188035	10112	122960	83264	3577	0.66	0.45	0.36
123	7	2	30	181029	180695	9692	118337	80519	3452	0.66	0.45	0.36

Table 2: Factorization of N with GT-ADD without and with the greedy method (with 1-node)

As in the table, required resources are depend on the parameters N and n, but on n mainly. The ratio G/D seems to be a constant depending on the features of R-ADD, GT-ADD, and Q-ADD. Since Q-ADD has many 1-qubit operations and is easy to parallelize, so that the ratio G/D is smaller (0.50-0.53) compared to other ADDs (0.79 to 0.81 for R-ADD, 0.68-0.79 for GT-ADD). Though G and D are in expected order, $O(n^3)$ for R-ADD, $O(n^5)$ for GT-ADD, and $O(n^4)$ for Q-ADD, coefficients differ from expected ones. The reason is that parameters are so small that other terms rather than the dominant term affect. The difference may be smaller for larger parameters.

4.2 Greedy Circuit

Next, in order to show the superiority of the greedy method, we factored (same) small RSA-type composite numbers up to 7-bit, without and with the greedy method described in subsection 3.2 for GT-ADD. Results are summarized in Table 2, where results in the 'Greedy' column coincide the results shown at the center column in Table 1.

As in the table, the greedy method reduces the number of gates to 0.66 to 0.71, and the depth to 0.45 to 0.56. Since generated Toffoli gates by the greedy method can be parallelized easily, the effect on the depth is much larger than that on the number of gates. In addition, the greedy method works well when k becomes larger. Analyzing the effect of greedy method mathematically is the future work.

4.3 Optimized Circuit

Then, we factor larger integers, 8-bit and 9-bit integers with 512-nodes. GT-ADD is used for the experiment because it requires less number of qubits and gates compared to other ADDs. In order to decrease the number of gates and the depth as much as possible, we used `optimize_light` option of *Qulacs* which unifies successive 1-qubit gates to one gate. However, the effect was very limited: it reduce the number of gates by only 1 percent.

Since factorization of 9-bit integers require 38-qubits, and 256-nodes are sufficient for the computation, other 256-nodes can be used for the speed-up. To do so, we used the `fused_swap_option` option of *mpiQulacs* which enables to distribute tasks to identified nodes for efficient computation.

Table 3 summarizes the factorization results. As in the table, we have succeeded factoring all RSA-type integers up to 9-bit. The largest integer we factored here was $N = 511$, which requires 8226 seconds (2.3 hour). On the other hand, `optimize_light` option works very well for Q-ADD, since Q-ADD uses a lot of successive 1-qubit gates. In fact, the optimized quantum circuit for factoring $N = 511$ with Q-ADD requires 225523 gates and 187618 depth, and it factors $N = 512$ in 7050 seconds (1.96 hours) in the experiment.

4.4 Circuit Estimation

Finally we estimated the quantum circuit resources for factoring 1024-bit and 2048-bit integers. For each $8 \leq n \leq 25$, we generated 10 composite numbers N randomly (180 composite numbers in total). Then, we generated the quantum circuit for each N with `optimize_light` option, and evaluate the number of elementary gates and the depth. Here, we used R-ADD since resources become smaller than others for larger N's. Next, we compute the average of resources for each n. Average values, lowest values, and highest values for each n are summarized in Table 4.

From averaged values for $8 \leq n \leq 25$, we obtain the approximation polynomials

$$G(n) = 258.7738275n^3 - 274.4792312n^2 - 672.6639462n + 3829.183247,$$
$$D(n) = 208.4410876n^3 - 202.3885578n^2 - 1143.4193n + 6530.413897.$$

with assuming that $G(n) = O(n^3)$ and $D(n) = O(n^3)$. Then, by substituting $n = 1024$ and $n = 2048$ to these polynomials, we obtain approximations as in Table 5. Compared to the estimation by Kunihiro, our estimation decreases to about 4% for the number of gates. We do not discuss the feasibility of such a huge quantum computer, however, if the quantum circuit for factoring 2048-bit integers are proceeded by an ideal quantum computer which can proceed the operation in the same speed as Google's Sycamore [2], that took 200 seconds to sample 10^6 times with a circuit with depth 40, factoring requires about 104.17 days, which seems infeasible by the current quantum technology.

5 Concluding Remarks

In this paper, we implemented Shor algorithm for factoring general composite numbers using 3 different ADD circuits (R-ADD, GT-ADD, and Q-ADD) and succeeded in factoring all RSA-type composite numbers up to 9-bit composite numbers in addition to the maximum 9-bit composite number 511, using the quantum computer simulator developed by Fujitsu. The time required for factorization of $N = 511$ is within 2 hours. We also estimated the number of gates and depth required of Shor's quantum circuit for larger composite numbers by actually generating quantum circuits, and gave the estimation for 1024 and 2048-bit integers. The future work will be to complete the experiments in factorization and estimation, provide a method to minimize the circuit depth, and conduct integer factorization experiments of larger composite numbers.

Acknowledgments

The third author conducted the research supported by JST CREST JPMJCR2113 and JPSP KAKENHI Grant Number JP21H03440.

References

1. Amico, M., Saleem, Z.H., Kumph, M.: Experimental study of Shor's factoring algorithm using the IBM Q Experience. Physical Review A **100**(1), 012305 (2019). https://doi.org/10.1103/PhysRevA.100.012305
2. Arute, F., Arya, K., Babbus, R., et al.: Quantum supremacy using a programmable superconducting processor. Nature **574**, 505–510 (2019). https://doi.org/10.1038/s41586-019-1666-5
3. Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H.: Elementary gates for quantum computation. Physical review A **52**(5), 3457 (1995). https://doi.org/10.1103/PhysRevA.52.3457
4. Beauregard, S.: Circuit for shor's algorithm using 2n+3 qubits. arXiv preprint quant-ph/0205095 (2002). https://doi.org/10.48550/arXiv.quant-ph/0205095
5. Beckman, D., Chari, A.N., Devabhaktuni, S., Preskill, J.: Efficient networks for quantum factoring. Physical Review A **54**(2), 1034 (1996). https://doi.org/10.1103/PhysRevA.54.1034
6. Boudot, F., Gaudry, P., Guillevic, A., Heninger, N., Thomé, E., Zimmermann, P.: Factorization of RSA-250. https://web.archive.org/web/20200228234716/https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2020-February/001166.html (2020)
7. Draper, T.G.: Addition on a quantum computer. arXiv preprint quant-ph/0008033 (2000). https://doi.org/10.48550/arXiv.quant-ph/0008033
8. Gidney, C., Ekerå, M.: How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. Quantum **5**, 433 (2021). https://doi.org/10.22331/q-2021-04-15-433
9. Gouzien, E., Sangouard, N.: Factoring 2048-bit RSA integers in 177 days with 13 436 qubits and a multimode memory. Physical Review Letters **127**(14), 140503 (2021). https://doi.org/10.1103/PhysRevLett.127.140503
10. IBM: 433–qubits quantum processor, Osprey. https://research.ibm.com/blog/next-wave-quantum-centric-supercomputing
11. Imamura, S., Yamazaki, M., Honda, T., Kasagi, A., Tabuchi, A., Nakao, H., Fukumoto, N., Nakashima, K.: mpiqulacs: A distributed quantum computer simulator for A64FX-based cluster systems. arXiv preprint arXiv:2203.16044 (2022). https://doi.org/10.48550/arXiv.2203.16044
12. Kunihiro, N.: Exact analyses of computational time for factoring in quantum computers. IEICE transactions on fundamentals of electronics, communications and computer sciences **88**(1), 105–111 (2005). https://doi.org/10.1093/ietfec/e88-a.1.105
13. Lanyon, B.P., Weinhold, T.J., Langford, N.K., Barbieri, M., James, D.F., Gilchrist, A., White, A.G.: Experimental demonstration of a compiled version of Shor's algorithm with quantum entanglement. Physical Review Letters **99**(25), 250505 (2007). https://doi.org/10.1103/PhysRevLett.99.250505
14. Lu, C.Y., Browne, D.E., Yang, T., Pan, J.W.: Demonstration of a compiled version of Shor's quantum factoring algorithm using photonic qubits. Physical Review Letters **99**(25), 250504 (2007). https://doi.org/10.1103/PhysRevLett.99.250504
15. Lucero, E., Barends, R., Chen, Y., Kelly, J., Mariantoni, M., Megrant, A., O ' Malley, P., Sank, D., Vainsencher, A., Wenner, J., et al.: Computing prime factors with a Josephson phase qubit quantum processor. Nature Physics **8**(10), 719–723 (2012). https://doi.org/10.1038/nphys2385

16. Martin-Lopez, E., Laing, A., Lawson, T., Alvarez, R., Zhou, X.Q., O'brien, J.L.: Experimental realization of Shor's quantum factoring algorithm using qubit recycling. Nature photonics **6**(11), 773–776 (2012). https://doi.org/10.1038/nphoton.2012.259
17. Politi, A., Matthews, J.C., O'brien, J.L.: Shor's quantum factoring algorithm on a photonic chip. Science **325**(5945), 1221–1221 (2009). https://doi.org/10.1126/science.1173731
18. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th annual symposium on foundations of computer science. pp. 124–134. Ieee (1994). https://doi.org/10.1109/SFCS.1994.365700
19. Suzuki, Y., Kawase, Y., Masumura, Y., Hiraga, Y., Nakadai, M., Chen, J., Nakanishi, K.M., Mitarai, K., Imai, R., Tamiya, S., et al.: Qulacs: a fast and versatile quantum circuit simulator for research purpose. Quantum **5**, 559 (2021). https://doi.org/10.22331/q-2021-10-06-559
20. Takahashi, Y., Kunihiro, N.: A quantum circuit for shor's factoring algorithm using $2n+2$ qubits. Quantum Information and Computation **6**(2), 184–192 (2006). https://doi.org/10.26421/QIC6.2-4
21. Vedral, V., Barenco, A., Ekert, A.: Quantum networks for elementary arithmetic operations. Physical Review A **54**(1), 147 (1996). https://doi.org/10.1103/PhysRevA.54.147

N	n	a	Q	G	D	T	N	n	a	Q	G	D	T
129	8	7	34	152780	100141	256	321	9	7	38	260877	166496	5899
133	8	2	34	169108	111205	247	323	9	2	38	304490	193554	5956
141	8	2	34	183453	120170	287	327	9	2	38	322336	204745	6115
143	8	2	34	207514	135907	311	329	9	3	38	285506	182113	6099
145	8	6	34	158918	105271	262	335	9	2	38	349246	222013	8104
155	8	2	34	198473	130150	311	339	9	2	38	317273	201779	7109
159	8	2	34	217743	142924	335	341	9	2	38	291468	186213	6363
161	8	3	34	155238	103030	238	355	9	2	38	310783	197410	7491
177	8	5	34	168876	111997	259	365	9	2	38	322926	206125	6346
183	8	2	34	207468	136410	297	371	9	2	38	324641	206674	6287
185	8	3	34	180752	119593	282	377	9	3	38	316691	202612	6676
187	8	2	34	208281	137192	328	381	9	2	38	321134	204686	5860
201	8	7	34	170050	112064	244	391	9	2	38	326281	207709	6697
203	8	2	34	193163	126762	285	393	9	5	38	281956	179878	6014
205	8	3	34	178117	117326	276	395	9	2	38	319088	203494	7307
209	8	3	34	165014	109327	243	403	9	2	38	307506	195485	6271
213	8	2	34	184210	121450	272	407	9	2	38	338095	214301	7907
215	8	2	34	204621	134697	327	411	9	2	38	335319	214006	7404
217	8	5	34	178741	118044	255	413	9	2	38	327370	208569	6648
219	8	2	34	204160	134522	299	415	9	2	38	359587	228199	7723
221	8	2	34	200121	131790	283	417	9	5	38	267426	171328	5940
235	8	2	34	198443	130597	285	427	9	2	38	324243	207582	6862
237	8	2	34	193348	127347	286	437	9	2	38	314856	200771	5925
247	8	2	34	208086	136900	289	445	9	2	38	339458	216426	6572
249	8	11	34	186487	123502	292	447	9	2	38	373035	237421	7448
253	8	2	34	202159	133987	306	451	9	2	38	306484	195876	5999
259	9	2	38	288684	183065	6143	453	9	2	38	286538	183164	6146
265	9	6	38	272685	173346	5620	469	9	2	38	303229	193946	6246
267	9	2	38	309270	196137	6572	471	9	2	38	343707	219148	7473
287	9	2	38	359003	228259	7511	473	9	3	38	303975	194528	6933
291	9	2	38	308155	195603	6542	481	9	3	38	281077	180267	6815
295	9	2	38	334848	212590	6370	485	9	2	38	305606	195586	6502
299	9	2	38	321523	204402	7094	489	9	7	38	302012	193333	7218
301	9	2	38	317493	202575	6461	493	9	2	38	329162	210756	6188
303	9	2	38	353151	224856	7559	497	9	3	38	296472	189877	5750
305	9	3	38	285798	182560	6350	501	9	2	38	322414	207063	6335
309	9	2	38	309354	196737	6358	505	9	6	38	313370	200596	6811
319	9	2	38	367923	233944	7419	511	9	3	38	395310	252188	8226

Table 3: Factorization of N up to 9-bit with GT-ADD (with 512-nodes)

	Average		Lowest		Highest	
n	G	D	G	D	G	D
8	113227.4	90682.7	111867	89084	114916	91792
9	164064.8	131693.9	162562	130386	165761	133579
10	228620.5	183741.9	227406	181906	229925	184448
11	307189.8	246775.1	302661	242850	309775	248542
12	404000.3	324664.2	397526	318331	406308	327116
13	517945.8	416032.7	513422	410828	522196	419008
14	650122.7	522313.3	640454	514596	655329	527115
15	805998.7	647481.2	797863	641646	810234	649475
16	982330.4	790253.8	977792	787101	988121	795903
17	1184249.2	951764.9	1179409	947741	1191981	956522
18	1411723.6	1134871.5	1406205	1129190	1420032	1140315
19	1666814.8	1342127.5	1661379	1338674	1671055	1345156
20	1948712.5	1568464.2	1943035	1563174	1956088	1571912
21	2265657.1	1824869.0	2261356	1821422	2274226	1831784
22	2613616.4	2104499.0	2607625	2098512	2619419	2108826
23	2991653.5	2409196.1	2986364	2405673	3007653	2415521
24	3407453.3	2744574.6	3395725	2733019	3424474	2754929
25	3857799.5	3107256.0	3843059	3100858	3867954	3116754

Table 4: Resources of optimized Shor's circuit with R-ADD

	$n = 1024$			$n = 2048$		
	Q	G	D	Q	G	D
Ours	5121	2.78×10^{11}	2.24×10^{11}	10241	2.23×10^{12}	1.80×10^{12}
Kunihiro [12]	3074	2.90×10^{11}	–	6146	2.32×10^{12}	–

Table 5: Circuit estimation for factoring 1024/2048-bit integers