A technical note for a Shor's algorithm by phase estimation

G. Fleury and P. Lacomme

Université Clermont-Auvergne, Clermont-Auvergne-INP,
LIMOS (UMR CNRS 6158),
1 rue de la Chebarde,
63178 Aubière Cedex, France

gerard.fleury@isima.fr, philippe.lacomme@isima.fr

ARTICLE INFO

Keywords:
Shor's algorithm
Phase

ABSTRACT

The objective of this paper concerns at first the motivation and the method of Shor's algorithm including an excursion into quantum mechanics and quantum computing introducing an algorithmic description of the method. The corner stone of the Shor's algorithm is the modular exponentiation that is the most computational component (in time and space). Second, a linear depth unit based on phase estimation is introduced and a description of a generic version of a modular multiplier based on phases is introduced to build block of a modular exponentiation circuit. Our proposal includes numerical experiments achieved on both the IBM simulator using the Qiskit library and on quantum physical optimizers provided by IBM.

1. Introduction

The most famous quantum algorithm is the Shor's algorithms that is dedicated to integer factorization and that received a considerable amount of attention due to its possible application in the cryptanalysis field. The algorithm proposes a super-polynomial execution speedup as regards the classical resolution on classical computers. The RSA encryption method lies on the hypothesis that it is exponentially harder to factor large numbers that achieving a multiplication ensuring secure send of information online.

Feynman has been the very first researcher considering that quantum mechanics could be more efficient computationally than a Turing machine considering that a computer based on quantum mechanics should avoid the expensive computation required for simulation on a classical computer. The question has been addressed first by (Deutsch and Jozsa, 1992) proving that some problems can be solved fast to optimality using quantum computers. In 1994 (Simon, 1994) introduced a circuit using an oracle to solve a problem (i.e. a black-box that can be checked) requiring a polynomial time since it requires an exponential time on a classical non quantum computer.

(Shor, 1994) introduced a quantum computer algorithm for factoring a integer number $N$ indirectly by computing the order of an element $a$ in the multiplicative group $(mod N)$ considering the lower integer $l / a^l \equiv 1 \ mod \ N$. He defines a new promising approach to factor a number into a product of primes and his proposal resulted in a significant improvement of the factoring algorithms efficiency. The corner stone of the Shor's algorithm is the modular exponentiation that is the most computational component (in time and space) of Shor's algorithm.

2. Proposition

Finding a factor of $N$ ($N$ is an odd number) requires a method to compute the order $l$ of $a$ where $a$ is a random integer number / $a \in [2; N-1]$:

\[
\begin{align*}
    a^l & \equiv 1 \ mod \ N \\
    a^l & \equiv 1 \ mod \ N
\end{align*}
\]

If $l$ is an even number we have:

\[
\left( a^{l/2} - 1 \right) \left( a^{l/2} + 1 \right) \equiv 0 \ mod \ N
\]

and the Euclidean algorithm of (Knuth, 1981) can be used to compute the $gcd(a^2 - 1; N)$ and $gcd(a^2 + 1; N)$ giving the two dividers of $N$. If $l$ is an odd number, it is possible to investigate both the even numbers $l - 1$ and $l + 1$.

Our work propose an architecture where the computation of the order of an element $a$ in the multiplicative group $(mod N)$ is replaced by the computation of the order of $a$ in the phase additive group $(mod 2\pi)$. The Quantum Circuit defines a probability phase distribution that aggregates probabilities on phases that indirectly model order of $a$. The measurement gates jointly gives a float value that defines the phase that will permit to find the order of $a$ with a high probability through further classical post-processing. The sampling of the probability distribution permits to create a heap $\chi$ of set of phases (step 5 to 8 in algorithm 1). The post-processing consists in iteratively considering the phases registered in $\chi$ which may gives on order of $a$: $l_i = \chi_i \times N$.

If $l_i \equiv 0 \ mod \ 2$ the post-processing consists in computing both $gcd(a^{l_i/2} - 1; N)$ and $gcd(a^{l_i/2} + 1; N)$ which define two dividers of $N$ and that are saved into the set of divider of $N$ referred to as $\varnothing$. If $l_i$ is an odd number, both the even number $l_i - 1$ and the even number $l_i + 1$ are investigated for giving factors of $N$. 

1
Note that the "necessary" conditions at steps 8, 9 and 24 have to be design over considerations of the objective function that could be, but not limited to: 1) computation of one divider only; 2) computation of a fixed number of dividers; 3) computation of all dividers.

### 2.1. Shor's Quantum Circuit based on phase

Given a, to find l such that $a^l \equiv 1 \mod N$, the circuit is based on the following considerations. First we find p the lowest power of 2 such that $2^p = q$ satisfy $2^p > N + 1$ and we compute n the lowest power of 2 such that $2^n > a$.

![Quantum Circuit](image)

The second step is easy since its entails putting each qubit of the first register into the superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and to compute $a^l \mod N$ in the second register leaving the circuit in the state:

$$|\psi\rangle = \frac{1}{\sqrt{q}} \sum_{j=0}^{q-1} |j\rangle \otimes |a\rangle$$

The most computational quantum part of the classical Shor's algorithm is the modular exponentiation $U_N^a$ that achieves the transformation to its quantum input:

$$U_N^a |l\rangle \otimes |y\rangle = |l\rangle \otimes |y \oplus (a^l \mod N)\rangle$$

with $0 \leq l < q$

$0 \leq y < 2^n$

The Hadamard gates on the first qubits creates a superposition of number l before application of the modular exponentiation.

$$|\psi_2\rangle = U_N^a \left( \frac{1}{\sqrt{q}} \sum_{j=0}^{q-1} |j\rangle \otimes |0\rangle^{\otimes n} \right)$$

$$|\psi_2\rangle = U_N^a \left( \frac{1}{\sqrt{q}} \sum_{j=0}^{q-1} |j\rangle \otimes |0 ... 0\rangle \right)$$

---

**Algorithm 1. Shor's algorithm based on phase**

**Input parameter**
- $N$: a positive integer number
- $n_s$: maximal number of measurement (sampling)

**Output parameter:**
- $\chi$: set of phases
- $\varphi$: set of dividers of N

**begin**
1. $\chi = \emptyset$ ; $\varphi = \emptyset$
2. repeat
3. $a := \text{random}(2; N-1)$
4. $|\varphi_1\rangle := \text{Shor\_quantum\_circuit}(a, N)$
5. repeat
6. phase $\varphi_i$ = Measurement of the circuit
7. $\chi = \chi \cup \varphi_i$
8. until (necessary) or (iterations exceed $n_s$)
9. while (1 in $\emptyset$ to card($\chi$)) and (necessary) do
10. $l_i = \chi_i \times N$
11. if $l_i \equiv 0 \mod 2$ then
12. Compute $\nu_1 = a^{\frac{l_i}{2}} + 1$ and $\nu_2 = a^{\frac{l_i}{2}} - 1$
13. $p_1 = \text{gcd}(\nu_1; N)$ ; $p_2 = \text{gcd}(\nu_2; N)$
14. $\varphi = \varphi \cup \{p_1, p_2\}$
15. end if
16. if $l_i \equiv 1 \mod 2$ then
17. Compute $\nu_1 = a^{\frac{l_i}{2}} + 1$ and $\nu_2 = a^{\frac{l_i}{2}} - 1$
18. Compute $\nu_3 = a^{\frac{l_i}{2}} + 1$ and $\nu_4 = a^{\frac{l_i}{2}} - 1$
19. $p_1 = \text{gcd}(\nu_1; N)$ ; $p_2 = \text{gcd}(\nu_2; N)$
20. $p_3 = \text{gcd}(\nu_3; N)$ ; $p_4 = \text{gcd}(\nu_4; N)$
21. $\varphi = \varphi \cup \{p_1, p_2, p_3, p_4\}$
22. end if
23 end while
24 until (necessary)
**end**
\[ |\psi_2\rangle = \frac{1}{\sqrt{q}} \sum_{j=0}^{q-1} |j\rangle \otimes |0 \oplus a \mod N\]  

Classical modular exponentiations are based on: 1) controlled multiplier/accumulator (Draper et al., 1998) (Bauregard et al., 2003) 2) quantum divider based on the algorithm introduced by (Grandlund and Montgomery, 1994).

We propose to replace the modular exponentiation \( U_N^{\varphi_a} \) by \( U_N^{\varphi_{a^l}} \) that achieves the transformation to its quantum input in:

\[ U_N^{\varphi_{a^l}} |l\rangle \otimes |y\rangle = |l\rangle \otimes |y \oplus \varphi_{a^l} \mod 2\pi\]  
with  
\[ 0 \leq l < q \]  
\[ 0 \leq y < 2^n \]  
The Hadamard gates at the first \( q \) qubits have created a superposition of number \( l \) before application of the modular exponentiation.

\[ |\psi_2\rangle = U_N^{\varphi_{a^l}} \left( \frac{1}{\sqrt{q}} \sum_{j=0}^{q-1} |j\rangle \otimes |0\rangle \otimes |\varphi_{a^l} \mod 2\pi\rangle \right) \]

\[ |\psi_2\rangle = U_N^{\varphi_{a^l}} \left( \frac{1}{\sqrt{q}} \sum_{j=0}^{q-1} |j\rangle \otimes |0 \cdots 0\rangle \right) \]

\[ |\psi_2\rangle = \frac{1}{\sqrt{q}} \sum_{j=0}^{q-1} |j\rangle \otimes |\overline{0} \oplus \varphi_{a^l} \mod 2\pi\rangle \]

The classical post-processing of the measurement in the computational basis after the Inverse Quantum Fourier Transform (QFT\(^{-1}\) block) is a sampling procedure that gives with high probability the period of the function \( f(l) = \varphi_{a^l} \mod 2\pi \).

During post-processing, all the phases \( \varphi_{a^l} \) are saved into a pool \( \chi \) of phases and all phases are iteratively investigated to check if the phase can permit to obtain one divider:

\[ \text{For all } l_i = \chi_i \times N / l_i \equiv 0 \mod 2, \text{ compute } v_1 = a^{l_i/2} - 1 \text{ and } v_2 = a^{l_i/2} + 1 \text{ and next } p_1 = gcd(v_1; N) \text{ and } p_2 = gcd(v_2; N). \text{ } p_1 \text{ and } p_2 \text{ are both dividers of } N. \]

\[ \text{For all } l_i = \chi_i \times N / l_i \equiv 1 \mod 2, \text{ compute } v_1 = a^{(l_i+1)/2} - 1, \text{ } v_2 = a^{(l_i+1)/2} + 1, \text{ } v_3 = a^{(l_i-1)/2} - 1, \text{ } v_4 = a^{(l_i-1)/2} + 1. \text{ Compute next } p_1 = gcd(v_1; N), \text{ } p_2 = gcd(v_2; N), \text{ } p_3 = gcd(v_3; N) \text{ and } p_4 = gcd(v_4; N) \text{ that are dividers of } N. \]

2.2. Quantum Fourier Transformation

\[ F_q(|k\rangle) = X|k\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} e^{2\pi i j k / 2^n} |l\rangle \]

\[ F_q(|k\rangle) = X|k\rangle = \frac{1}{\sqrt{q}} \sum_{j=0}^{q-1} e^{2\pi i j k / q} |l\rangle \]

\[ F_q(|k\rangle) = X|k\rangle = \frac{1}{\sqrt{q}} \sum_{j=0}^{q-1} \omega^{k j} |j\rangle \text{ avec } \omega^{j} = e^{2\pi i j / q} \]

By consequence

\[ F_q^\dagger(|k\rangle) = \frac{1}{\sqrt{q}} \sum_{j=0}^{q-1} \omega^{-k j} |j\rangle \]

2.3. Modular exponentiation of the phase : \( U_N^{\varphi_{a^l}} \)

The design uses \( q = 2^p \) qubits to represents \( l \), a controlled qubit for the Inverse Fourier Transformation, \( n \) qubits to represents \( a \) and a controlled qubit required by \( U_N^{\varphi_{a^l}} \) (figure 2).

The design uses \( q \) controlled-\( U \) (CU) block, and each block referred to as \( U_N^{\varphi_{a^l}} \) is a controlled modular \( 2\pi \) multiplier of its input (the lower quantum register) by constant \( \varphi_{a^l} \) by \( (2^{p-1} \mod N) \). Each block is controlled by the corresponding \( |l_i\rangle \) qubit of the upper quantum register.
Definition of $\varphi_{a^k}$

Let us note $\varphi_a = \frac{2\pi}{N}$. The phase of $a$ is:

$$\varphi_{a^j} = \left(2^{j-1} \mod N\right) \frac{2\pi}{N}.$$

The modular exponentiation $U_N^{\varphi_k}$ requires the iterative computation of $\varphi_{a^1}, \varphi_{a^2}, \varphi_{a^4}, \varphi_{a^8}, \ldots, \varphi_{a^k}, \ldots$

i.e. the computation of

$$\varphi_{a^0}, \varphi_{a^1}, \varphi_{a^2}, \varphi_{a^3}, \ldots, \varphi_{a^{p-1}}, \varphi_{a^p}, \ldots$$

We note

$$\varphi_{a^{2^p}} = \varphi_{a^{2^{p+1}}} + \sum_{k=1}^{2^{p-1}} \varphi_a = \varphi_{a^{2^{p+1}}} + 2^{p-1} \varphi_a$$

For example with $a = 3$, we have $\varphi_a = \frac{2\pi}{N} \cdot 3$, $\varphi_{a^2} = \frac{2\pi}{N} \cdot 3 + \frac{2\pi}{N} \cdot 3$ and

$$\varphi_{a^4} = \frac{2\pi}{N} \cdot 3 + \frac{2\pi}{N} \cdot 3 + \frac{2\pi}{N} \cdot 3 + \frac{2\pi}{N} \cdot 3$$

$$\varphi_{a^8} = \varphi_{a^2} \cdot 2^2, \varphi_a = \varphi_{a^2} + 4, \varphi_a = 8, \varphi_a$$

$$\varphi_{a^{16}} = \varphi_{a^8} \cdot 2^4, \varphi_a = \varphi_{a^8} + 8, \varphi_a = 16, \varphi_a$$

Definition of $U_N^{\varphi_k}$

A controlled $U_N^{\varphi_k}$ block is based on the application of $CP(a)$ conditional phase rotation gates that need to be repeated a logarithmic number of times regarding to $N$ and next on the application of the Inverse Fourier Transformation (Fig. 3). The diagonal symmetric controlled gate $CP(a)$ is used to induce a phase on the state of the target qubit number $n$ depending on the control qubit state.

Matrix representation of $CP(a)$ for two qubits

$$CP(a) = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes P = \begin{pmatrix} 1 & 1 \\ 1 & e^{ia} \end{pmatrix}$$

$n$ consecutive applications of $CP(a)$ gates are required for defining a $n$-qubit quantum number

$$|b\rangle = |b_{n-1} \rangle |b_{n-2} \rangle \cdots |b_1 \rangle |b_0 \rangle$$

that is transformed by a simple uncontrolled Inverse Quantum Fourier Transformation into a quantum state

$$|\varphi_{a^k}\rangle = |\varphi_{a^{k-1}}\rangle |\varphi_{a^{k-2}}\rangle |\varphi_{a^{k-3}}\rangle \cdots |\varphi_{a^k}\rangle$$

Fig. 3. Design of $U_N^{\varphi_k}$

We have:

$$F_n(b) = \frac{1}{\sqrt{2^N}} \sum_{j=0}^{2^n-1} e^{2\pi i \cdot \frac{b_j}{2^n} \cdot j}$$

The individual $j$th qubit $|\varphi_{a^j}\rangle$ of $|\varphi_{a^k}\rangle$:

$$|\varphi_{a^j}\rangle = \frac{1}{2} (|0\rangle + e^{2\pi i a \cdot b_j} |1\rangle)$$

3. Experimental validation

With the following examples, the goal is to illustrate the performances of this phase based Shor's algorithm on both simulator and physical quantum optimizer. Experiments are achieved considering 5 large numbers (introduced in table 1) with a number of dividers that varies from 4 to 24.

Table 1. Integer number used for experimentations

| $N$  | $\#$ of dividers | Dividers |
|------|-------------------|---------|
| 844  | 24                | 1; 3; 9; 37; 43; 59; 111; 129; 177; 333; 387; 531; 1591; 2163; 2537; 4773; 6549; 7611; 14119; 19647; 22933; 93869; 281607; 844821 |
| 1414 | 5                 | 1; 821; 1723; 1414383 |
| 1660 | 4                 | 1; 1299; 1471; 1660759 |
| 3173 | 4                 | 1; 1471; 2129; 3131759 |
| 5173 | 8                 | 1; 7; 13; 91; 56393; 394751; 733119; 5131763 |
| 5173 | 4                 | 1; 103; 49823; 5131769 |
| 7173 | 4                 | 1; 223; 31981; 7131763 |
| 7173 | 6                 | 1; 113; 63113; 7131769 |
| 1713 | 16                | 1; 3; 9; 1903529; 5710587; 17131761 |
| 3271 | 16                | 1; 11; 73; 263; 803; 1549; 2893; 17039; 19199; 131707; 211199; 407387; 1243847; 4461257; 29739251; 327131761 |
| 3271 | 8                 | 1; 1; 229; 233; 6131; 53357; 1403999; 1428523; 327131767 |
| 1593 | 8                 | 1; 3; 21211; 25147; 63363; 75441; 53129787; 1593389361 |
| 1593 | 4                 | 1; 31981; 49823; 1593389363 |
| 298500 | 156599 | 1; 139; 93049; 320951; 320951; 298500156599 |

The number of qubits required depends on both $N$ and $a$ : the experiments on the simulator are limited to 32 qubits and the experiments on quantum optimizer are limited to 124 qubits that is the largest IBM physical quantum optimizer available. A number with many dividers should be easier to factor in the sense that one circuit execution should permit to obtain several dividers in one sampling run. Note that for a theoretical point of view, the
value of \( a \) should be defined at random but from a practical point of view, an efficient implementation should investigate small values first to obtain smaller circuits.

Experiments have been achieved on the Brooklyn Quantum Physical Optimizers introduced in Table 2 and has been chosen taking into account the individual optimizers’ availabilities and the total pending jobs involved in experiments with the potential to lead the results in acceptable delays at the particular moment. Note that preliminary experiments achieved on Auckland, Montreal and Washington quantum optimizers push us into considering that very similar results should be obtained. Let us note that the Brooklyn physical quantum optimizer is not the best one in term of number of qubits and it is not the more efficient as regards the quantum volume.

| Physical quantum optimizer | # of qubits | QV |
|----------------------------|-------------|----|
| auckland                   | 27          | 64 |
| montreal                   | 27          | 128|
| brooklyn                   | 65          | 32 |
| washington                 | 127         | 64 |

3.1. Numerical experiments with 1591
For the small \( N = 1591 \) the probability distribution of \( l \) are plotted on figure 4. The values \( l = 37, l = 43 \) could occurred when factoring 1591 if \( a \) were chosen to be 2 for example. The graphic concerns small \( N \) and small \( a \) values to make the plot distinguishable but very similar structures of probabilities can be observed for every \( N \) and \( a \) values. Note that the probabilities plotted on figure 5 are related to an execution on the Brooklyn Physical Quantum Optimizer defining values very close to the values obtained with the simulator.

![Fig. 4. The probability \( P \) to observe values giving \( N = 1591 \) and \( a = 2 \) (execution on simulator)](image1)

![Fig. 5. The probability \( P \) to observe values giving \( N = 1591 \) and \( a = 2 \) (execution on Brooklyn Physical Quantum Optimizer)](image2)
Table 3. Dividers found when factoring 1591 with \( a = 2 \) (execution on the simulator)

| Phase | \( l \) | Divider \( D_1 \) | Divider \( D_2 \) | \( N_{D_1} \) | \( N_{D_2} \) |
|-------|-------|----------------|----------------|----------------|----------------|
| 0.03515625 | 56 | 43 | 1 | 37 | 1591 |
| 0.16801170509565028 | 267 | 1 | 43 | 1591 | 37 |
| 0.4531679550956503 | 721 | 37 | 1 | 43 | 1591 |
| 0.3786632949043497 | 603 | 1 | 43 | 1591 | 37 |
| 0.6249570449043498 | 994 | 1 | 43 | 1591 | 37 |
| 0.0703125 | 112 | 43 | 1 | 37 | 1591 |
| 0.5898007949043498 | 938 | 1 | 43 | 1591 | 37 |
| 0.93663670509565028 | 1485 | 43 | 1 | 37 | 1591 |
| 0.17582420509565028 | 287 | 1 | 43 | 1591 | 37 |
| 0.16792579490434972 | 287 | 1 | 43 | 1591 | 37 |
| 0.6601132949043498 | 721 | 1 | 43 | 1591 | 37 |
| 0.4296875 | 684 | 1 | 37 | 1591 | 43 |
| 0.4843320449043497 | 771 | 1 | 43 | 1591 | 37 |
| 0.02265625 | 320 | 1 | 43 | 1591 | 37 |
| 0.3789492050956503 | 603 | 1 | 43 | 1591 | 37 |

The phases in \( x \) define possible values of \( l \) for \( a^l \) such that \( l \) should be the order of \( a \). Table 3 gives the observed value of both the phase and of \( l \) obtained when factoring 1591 with \( a = 2 \): the post-processing enables us to obtain the two non-trivial dividers of 1541 namely 43 and 37.

Execution on the Brooklyn gives quite different probabilities but the observed phase value permits to obtain the two dividers 43 and 37.

3.2. Numerical experiments on simulator when factoring 3 131 759

Execution of the simulator for \( N = 3 \) 131 759 has been achieved for several values of \( a \) and permits to retrieve several dividers during a single run. With \( a = 2 \) the run is unsuccessful and the run 3 with \( a = 3 \) succeeds retrieving 1471 and 2129 (table 4).

Note that the number of measurements (150 here) should be increased to obtain more consistent results between runs.

Table 4. Shor's algorithm on IBM's Simulator for \( N = 3 \) 131 759

| \( N \) | Quantum optimizer | Sampling value | Run | \( a \) | Phases | First Divider \( D_1 \) | Second Divider \( D_2 \) | \( N_{D_1} \) | \( N_{D_2} \) |
|-------|----------------|----------------|-----|------|-------|----------------|----------------|----------------|----------------|
| 3 131 759 | Simulator | 150 | 3 | 3 | 0.0223457562799555 | 1471 | 1 | 2129 | 3131759 |
| | | | 2 | 4 | 0.9345292472200445 | 1 | 2129 | 3131759 | 1471 |
| | | | 3 | 4 | 0.007841506279955507 | 2129 | 1 | 1471 | 3131759 |
| | | | 1 | 5 | 0.3945292473095445 | 1 | 2129 | 3131759 | 1471 |
| | | | 2 | 5 | 0.7986281187304445 | 1 | 2129 | 3131759 | 1471 |
| | | | 1 | 6 | 0.521486381799555 | 1 | 1471 | 3131759 | 2129 |
| | | | 2 | 6 | 0.521486381799555 | 1 | 1471 | 3131759 | 2129 |
| | | | 3 | 6 | 0.0402968785 | 1 | 2129 | 3131759 | 1471 |

3.3. Numerical experiments for 3 numbers: analysis of the phases and dividers obtained for 3 small numbers

Table 2 gives the observed values of the phase and the dividers obtained when factoring 844 821 with \( a = 2 \) during a single run. The results are compliant with the results provided by the simulator and the algorithm succeeds in computing a large part of the dividers.

Experiments for 1 414 583 have been achieved with one run for value of \( a \) varying from 2 to 5 and the two dividers are identified with \( a = 5 \).

The last number 1660759 has two non-trivial dividers only (1 471 and 1 129) that are found for \( a = 3, 4, 5 \) by the algorithm in one single run as stressed in table 5.
### 3.4. Numerical experiments on IBM programmable noisy quantum optimizer for large number

An attempt for $3 \times 131 \times 759$ factorization is achieved with 5 run and $\alpha = 2$ and it succeeds leading to non-trivial dividers i.e. 1471 and 2129 (table 6). Note that 5 131 763 can be factorized on the Brooklyn Physical Quantum Optimizer and that a single run with $\alpha = 2$ gives several dividers including 1, 7, 13, 91, 5639, 733109, 394751 and 5 131 763. The algorithm succeeds in factoring 5 131 769 with $\alpha = 2$.

Computation of the dividers of both 7 131 763 and 7 131 769 succeed for small value of $\alpha$ as stressed in table 6. The algorithm succeeds in factoring the large number 327 131 761 and using several run (5 runs are used in the experiments) several dividers are computed including large dividers. Note that the sampling parameters could have to be tune considering a balance between runs and number of sampling efforts.

Half of the dividers of 327 131 767 are computing on a single run for $\alpha = 2$ including the non-trivial dividers 229, 233, 1 403 999 and 1 428 523.

For number greater that $1.10^9$, the algorithm succeeds as stressed on table 6 where the two dividers of 1 593 389 363 are obtained for $\alpha = 4$ with a single run.
### Table 6. Shor’s algorithm on IBM’s physical quantum optimizer: large numbers

| Number          | Physical quantum optimizer | Sampling value | Run | α          | First Divider D1 | Second Divider D2 | N D1 | N D2 |
|-----------------|-----------------------------|----------------|-----|------------|------------------|------------------|------|------|
| 3 131 759       | brooklyn                    | 150            | 1   | 2          | 0.44544659017277777 | 1471             | 1    | 2129 | 3131759 |
|                 |                              |                | 5   | 2          | 0.00391187311457643 | 1471             | 1    | 2129 | 3131759 |
| 5 131 763       | brooklyn                    | 150            | 1   | 2          | 3.174194980911911e-05 | 7                | 1    | 733109 | 5131763 |
|                 |                              |                | 2   | 2          | 2.923206440988988e-05 | 7                | 1    | 733109 | 5131763 |
|                 |                              |                | 2   | 2          | 0.0078125               | 7                | 13   | 733109 | 394751  |
|                 |                              |                | 2   | 2          | 0.500183582309082      | 91               | 1    | 56393 | 5131763 |
| 5 131 769       | brooklyn                    | 150            | 1   | 2          | 0.12597545733903687   | 103              | 1    | 49823 | 5131769 |
|                 |                              |                | 1   | 2          | 0.0100176334810352    | 103              | 1    | 49823 | 5131769 |
| 7 131 763       | brooklyn                    | 150            | 1   | 2          | 0.5019550323486328   | 223              | 1    | 31981 | 7131763 |
|                 |                              |                | 2   | 2          | 0.5044555644713965   | 223              | 1    | 31981 | 7131763 |
| 7 131 769       | brooklyn                    | 150            | 1   | 2          | 6.2831845453753573  | 1                | 113  | 7131769 | 63113 |
|                 |                              |                | 2   | 2          | 0.12499911898642438 | 113              | 1    | 63113 | 7131769 |
| 17 131 761      | brooklyn                    | 150            | 1   | 2          | 0.00012243709709325517 | 1                | 3    | 17131761 | 5710587 |
|                 |                              |                | 2   | 2          | 0.0001217053558474483 | 3                | 1    | 5710587 | 17131761 |
|                 |                              |                | 2   | 2          | 0.5000158937664992   | 9                | 1    | 1903529 | 17131761 |
|                 |                              |                | 2   | 2          | 0.5000151602634287   | 9                | 1    | 1903529 | 17131761 |
| 327 131 761     | brooklyn                    | 150            | 1   | 2          | 0.001953650584637534 | 11               | 1    | 2973925 | 32713161 |
|                 |                              |                | 1   | 2          | 0.001955051955527575 | 73               | 1    | 4481257 | 32713161 |
|                 |                              |                | 2   | 2          | 0.03128812776576149 | 803              | 1    | 407387  | 32713161 |
|                 |                              |                | 1   | 2          | 0.03137252794276344 | 11               | 327131761 | 29739251 |
|                 |                              |                | 2   | 2          | 0.12744569778442383 | 263              | 1    | 1243847 | 32713161 |
|                 |                              |                | 2   | 2          | 0.6250076120502817  | 1243847          | 1    | 263    | 32713161 |
|                 |                              |                | 4   | 2          | 0.7509803964041604  | 2893             | 1    | 113077 | 32713161 |
| 327 131 767     | brooklyn                    | 150            | 1   | 2          | 0.001975993978699339 | 233              | 1    | 1403999 | 32713167 |
|                 |                              |                | 2   | 2          | 0.03674316406265    | 229              | 1    | 1428523 | 32713167 |
|                 |                              |                | 2   | 2          | 0.6279297402345071  | 1                | 229  | 32713167 | 1428523 |
| 1 593 389 361   | brooklyn                    | 150            | 1   | 2          | 0.0078125             | 1                | 3    | 1593389361 | 531129787 |
|                 |                              |                | 2   | 2          | 0.12503055179393176 | 1                | 3    | 1593389361 | 531129787 |
| 1 593 389 363   | brooklyn                    | 150            | 1   | 4          | 0.531250071929831    | 1                | 31981| 1593389363 | 49823  |

3. Extra experiments with very large numbers

Experiments have been carried out with the following numbers 298500156599, 237504336099404000, 237504336099404013, and 237504336099404017 that require more than 60 qubits.

To conclude, a set of 3 experiments have been achieved on the Washington physical quantum optimizer using

\[
N = 15785823750433609940413 \\
N = 23751578582375043360994041323457 \\
N = 23751578582375043360994040132345711
\]

All the results are introduced in table 7 and confirm the approach efficiency.
4. Concluding remarks

Considering the factorization of a large constant is a challenging problem that can be addressed using a modular multiplier based on phases to build block of a modular exponentiation circuit. Such approach is theoretically accurate taking advantages of the well-known phase definition \( (\varphi_{a^j} = (2j \mod N), \frac{2\pi j}{N}) \) and particularly useful.

Because of the effectiveness of our models for modular computation that defines compact circuit in both number of qubits and number of gates, our model has permit successful experimentations on physical quantum optimizer for very large integer numbers requiring circuit with more than 100 qubits.
References
Simon D. R. "On the power of quantum computation". Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pp. 116-123, 1994.
Deutsch D. and R. Jozsa, "Rapid solutions of problems by quantum computation," in Proceedings of the Royal Society of London A, vol. 439, pp. 553, 1992.
Shor P. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, Nov. 20-22, 1994.
Draper T.G. Addition on a Quantum Computer, arxiv e-print quant-ph/0008033v1, Vol. 3. 1998.
Beauregard S., Circuit for Shor’s algorithm using 2n+3 qubits, Quantum Information and Computation, vol. 3, pp 175-185, 2003.
Granlund T. and P.L. Montgomery, Division by Invariant Integers using Multiplication, Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation p 61-72. 1994.
Knuth D. E. The Art of Computer Programming, Vol. 2: Semi-numerical Algorithms, Second Edition, Addison-Wesley. 1981.
Appendix. Examples of runs

Fig. A1. Run on Brooklyn physical quantum optimizer for 1 593 389 363

Fig. A2. Run on Brooklyn physical quantum optimizer for 237 504 336 099 404 000 (circuit of 63 qubits)

Fig. A2. Run on Brooklyn physical quantum optimizer for 237 504 336 099 404 017 (circuit of 63 qubits)