UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Corso di Laurea Triennale in Fisica

Tesi di Laurea

Tensor network simulation of Shor's algorithm for prime factorization

Simulazione in tensor network dell'algoritmo di Shor per la fattorizzazione in numeri primi

Relatore

Prof. Simone Montangero

Correlatore

Dr. Ilaria Siloi

Laureando

Alessandro Cavion

2016111

Anno Accademico 2023/2024

To my parents

Abstract

This thesis explores the implementation and simulation of a scalable version of Shor's algorithm for prime factorization. An acknowledged bottleneck in this algorithm lies in the modular exponentiation process. To address this challenge, we propose a design for the quantum circuit based on the model developed by Vedral, Barenco, and Ekert. The computational engine employed for simulations is a classical tensor network quantum emulator (Quantum Matcha Tea) which uses the matrix product states (MPS) ansatz for the the wavefunction. The main achievement of this study is the successful execution of Shor's quantum circuits with over 100 qubits, showcasing both the emulator's proficiency in handling substantial computational complexities and the correct crafting of the quantum circuit.

Sommario

Questa tesi esplora l'implementazione e la simulazione di una versione scalabile dell'algoritmo di Shor per la fattorizzazione in primi. Un noto punto critico di questo algoritmo risiede nel processo di esponenziazione modulare. Per superare questo problema, viene proposto un circuito quantistico basato sul modello sviluppato da Vedral, Barenco ed Ekert. Il motore computazionale impiegato per le simulazioni è un emulatore quantistico tensor network (Quantum Matcha Tea), che utilizza l'ansatz matrix product states (MPS) per la funzione d'onda. Il più importante risultato di questo studio è l'esecuzione riuscita di circuiti quantistici di Shor con oltre 100 qubits, dimostrando sia la capacità dell'emulatore nel gestire complessità computazionali significative, sia la corretta progettazione del circuito quantistico.

Index

Introduction

In the realm of computational theory, the advent of quantum computing has ignited a profound revolution, offering unprecedented opportunities to address certain types of problems, making use of computations physically unachievable for classical computers. In fact, quantum computers operate based on the principles of quantum mechanics, exploiting quantum bits or qubits to process information in ways fundamentally different from classical bits.
The sustained investment in quantum computation research of the last decades has yielded remarkable fruits in terms of quantum algorithms. By ingeniously taking advantage of the principles of quantum mechanics, these algorithms have the potential to exponentially surpass their classical counterparts in specific problem domains. Foremost among these quantum algorithms stands Shor's algorithm [1], invented by Peter Shor in 1994. This algorithm provides a quantum-based solution to a classical challenge, namely the integer factorization into prime numbers, which holds significant implications for real world applications like cryptography, particularly in systems like the RSA protocol. The RSA encryption method relies on the complexity of factoring large numbers into two prime factors. Therefore, even if someone has access to the encrypted information and the public key (number to factorize), it is very difficult for them to discover the private key (factors) needed to decode the message. However, Shor's algorithm's unparalleled efficiency in determining factors and, consequently, the private key, poses a significant threat to information security, prompting the urgent need for the development of new quantum-resistant cryptographic methods. In contrast, classical factoring algorithms like number field sieve (NFS) methods are relatively harmless to information security due to their inefficiency when compared to their quantum counterparts. In fact, these classical methods, at best, exhibit sub-exponential complexity [2].
This thesis develops a quantum circuit to compute Shor's algorithm for any given integer N, based on the modular exponentiation model proposed by Vedral, Barenco, and Ekert in 1996. While relying on a tensor network emulator, we scale the algorithm to include up to 100 qubits. Further, the use of intensive simulations allows the quantification of the required resources for the intermediate steps of the algorithm both in terms of gates and memory.
In the first chapter, we begin with an introductory overview of quantum computing and its basic elements, followed by a brief examination of quantum circuits and their capabilities, placing particular emphasis on the quantum Fourier transform. Subsequently, in the second chapter, we delve into the operational principles that drive Shor's algorithm, elucidating its theoretical framework and outlining the implementation strategy. Moreover, a comprehensive examination of the VBE (Vedral, Barenco, Ekert) modular exponentiation [3] method sheds light on the design of the quantum circuit proposed. In the third chapter, utilizing a tensor network emulator for quantum circuits, we conduct thorough analyses of our successful implementation, delving into both practical algorithm's aspects and the emulator's capabilities. We begin with a concise overview of program initialization and result interpretation, then proceed to explore

network complexity in terms of qubits and quantum logic gates. Finally, we evaluate memory requirements and perform correlation analysis.

Chapter 1

An introduction to quantum computing

1.1 Quantum bits

In classical computation and classical information theory, the bit serves as the fundamental unit, operating *exclusively* within a binary framework with states limited to either 0 or 1. Quantum computation and quantum information theory are founded on a similar principle, employing the quantum bit, or qubit [4]. Specifically, qubits are two-level physical systems that can be found in two possible states denoted as $|0\rangle$ and $|1\rangle$. However, the key distinction between classical bits and qubits lies in the properties of quantum mechanics, for which a qubit can exist in a state beyond the binary spectrum, for instance in a superposition of both $|0\rangle$ and $|1\rangle$. Consequently, a generic single qubit state can be expressed as:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{1.1}$$

where α and β are two complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$.
The special states $|0\rangle$ and $|1\rangle$ are usually known as computational basis states, which are conventionally expressed in matrix notation as:

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{1.2}$$

In reference to Eq. 1.1, it is natural to rewrite[1] $|\psi\rangle$ as:

$$|\psi\rangle = cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi} sin\frac{\theta}{2}\,|1\rangle\,, \tag{1.3}$$

associating the state of the qubit to a vector $|\psi\rangle$ on the Bloch sphere as shown in Fig. 1.1.

Note: Although the qubit is essentially a physical system, in the following we will only treat it as a mathematical object, allowing the construction of a general theory of quantum information regardless of the specific physical implementation.

[1] The physical properties of a state don't change when this is multiplied by a global phase. In Eq. 1.3 we have indeed factored out the relative phase contained in α and treated it as a global phase, which is not written.

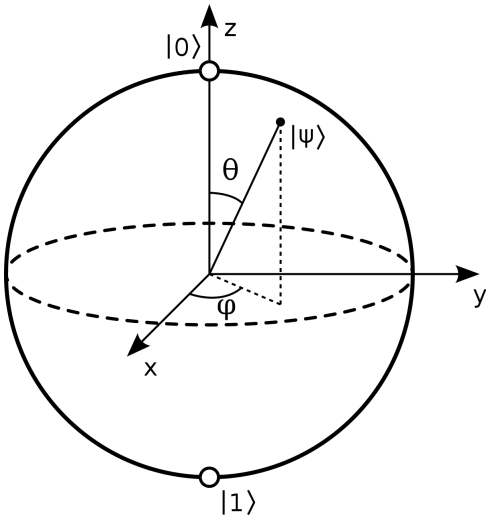


Figure 1.1: Graphical representation of the Bloch sphere.

Observing the structure of the Bloch sphere, the following question arises spontaneously: how much information is contained in a qubit? In fact, the sphere's surface is made of infinite points, can we use $\theta \in [0,\, \pi]$ and $\varphi \in [0,\, 2\pi[$ to encode an arbirtarily large amount of information? As one might suspect the answer in general is negative [4]. In reality, when we measure a qubit we will always get either $|0\rangle$ or $|1\rangle$ since, once the measurement happens, the superposition state collapses in the measured state. To obtain with arbitrarily precision the values of α and β, we should thus measure multiple copies of the same system and study the probability distribution of the outcomes. A process usually referred as *quantum tomography*.
At this point it is natural to ask ourselves what is the advantage of relying on a quantum computer in the first place. The fundamental concept to grasp is that quantum processes possess inherent properties beyond what can be directly observed through measurement, namely the exact values of α and β for a single qubit or the amplitudes for more general states. This intrinsic information influence the system's time evolution and consequently, when a measurement is **not** carried out, a qubit possesses more usable information compared to its classical counterpart [4].

1.2 Quantum circuits

Among the different models of quantum computation, the quantum circuit is the easiest one to generalize from the classical computation.
It mainly consists in three different stages:

- **Register initialization:** Initially, the qubits are prepared in specific quantum states, i.e. the starting conditions for our computation.
- **Processing:** A series of quantum gates operates on the state executing the computational tasks.
- **Measurement:** Finally, the quantum state of the qubits is measured, yielding classical outcomes that provide the result of the computation.

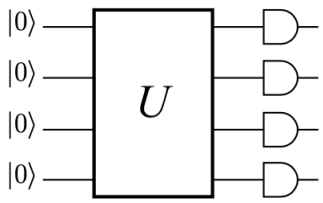


Figure 1.2: Scheme of a quantum circuit, time flow left to right. In this case the qubits are all initialized as $|0\rangle$ and arranged in different rows. U is a generic unitary operator acting on these qubits and consists in a collection of different quantum gates. On the right, attached to each qubit, there is the symbol of measurement applied.

Throughout the process, from initialization to measurement, the state of n qubits is consistently represented as a linear combination of tensor products of single qubit states $|\gamma_i\rangle = \{|0\rangle, |1\rangle\}$.

$$|\psi\rangle = \sum_{\vec{\gamma}} \Psi_{\gamma_1,\gamma_2,....,\gamma_N} \left(\bigotimes_{i=1}^{N} |\gamma_i\rangle \right) = \sum_{\vec{\gamma}} \Psi_{\gamma_1,\gamma_2,....,\gamma_N} |\gamma_1, \gamma_2, ..., \gamma_N\rangle , \tag{1.4}$$

where $\Psi_{\vec{\gamma}}$ are complex coefficients such that $\sum_{\vec{\gamma}} |\Psi_{\vec{\gamma}}|^2 = 1$ and $N = 2^n$.
For example, if we wanted to describe the initial state of Fig. 1.2, we would represent it as

$$|\psi_i\rangle = |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle \equiv |0000\rangle , \tag{1.5}$$

where the only non-null coefficient is $\Psi_{0,0,0,0} = 1$.
Or less trivially, we could describe an **entangled** state, such as the generalization to n qubits of the *Bell state* (GHZ state) defined as:

$$|\psi_{GHZ}\rangle = \frac{|00\cdots00\rangle + |11\cdots11\rangle}{\sqrt{2}}, \tag{1.6}$$

for which $\Psi_{00\cdots00} = \Psi_{11\cdots11} = 1/\sqrt{2}$.

1.3 Quantum logic gates

As mentioned before, a generic unitary quantum operator U can always be decomposed in a combination of specific quantum gates. We will now introduce the most important ones, and in particular those that have been implemented in the construction of the Shor's quantum circuit.

1.3.1 Single qubit gates

The single qubit gates are precisely those operations that acts only on a single qubit at a time and are represented by 2×2 unitary matrices, usually expressed in the computational basis $\{|0\rangle, |1\rangle\}$. The first important logic gate is the **Hadamard gate**, represented in matrix notation as:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \tag{1.7}$$

and schematically pictured as an H box on the qubit in which it is attached.

$$\text{—}\boxed{H}\text{—} \,. \tag{1.8}$$

If we simply apply the gate to our basis states through a matrix-vector product, we can easily obtain its truth table.

$$H\left|0\right\rangle = \frac{1}{\sqrt{2}}(\left|0\right\rangle + \left|1\right\rangle) \equiv \left|+\right\rangle \,, \qquad H\left|1\right\rangle = \frac{1}{\sqrt{2}}(\left|0\right\rangle - \left|1\right\rangle) \equiv \left|-\right\rangle \,. \tag{1.9}$$

Note that H is unitary at sight since $H^\dagger = H$ and H is clearly involutory[2], therefore by applying H to $\left|+\right\rangle$ or $\left|-\right\rangle$ we are able restore the corresponding initial states.
Another fundamental gate to consider is the **Phase-shift gate** which is defined as:

$$R_z(\varphi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{pmatrix} . \tag{1.10}$$

With the mathematical representation at hand, understanding its operation is quite straightforward. A phase factor $e^{i\varphi}$ is added only when the gate is applied to $\left|1\right\rangle$ states, modifying the relative phase in case of a superposition. Suppose we have a general state $\left|\psi\right\rangle = \alpha\left|0\right\rangle + \beta\left|1\right\rangle$ on which we want to apply the phase shift. Performing the necessary matrix-vector product calculations we obtain:

$$R_z(\varphi)\left|\psi\right\rangle = \alpha\left|0\right\rangle + e^{i\varphi}\beta\left|1\right\rangle . \tag{1.11}$$

The standard schematic representation of this gate is the following:

$$\text{—}\boxed{R_\varphi}\text{—} \,. \tag{1.12}$$

Ultimately, the last important gate to illustrate is the **NOT Gate**, which simply switch zeros with ones and viceversa, exacltly like its classical counterpart. It is represented by the σ_x Pauli matrix:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} , \tag{1.13}$$

and usually schematized in this way:

$$\text{—}\boxed{X}\text{—} \,. \tag{1.14}$$

In practice, we could assign a logic gate to each of the Pauli matrices (σ_i). However, as not all of them will be necessary for our subsequent discussions, they will not be presented here.
Instead, it's interesting to observe how the combination of Hadamard gates and phase shifts alone is sufficient to reconstruct the generic state of Eq. 1.3:

$$\left|\psi\right\rangle = R_z\left(\frac{\pi}{2} + \varphi\right) H R_z(\theta) H\left|0\right\rangle = cos\frac{\theta}{2}\left|0\right\rangle + e^{i\varphi} sin\frac{\theta}{2}\left|1\right\rangle . \tag{1.15}$$

[2]A matrix M is involutory if $M^2 = \mathbf{1}$, therefore M is a square root of $\mathbf{1}$.

1.3.2 Two qubits gates

From Eq. 1.4 we know that a generic state of a two qubits system can be written as follows:

$$|\psi\rangle = \gamma_1 |00\rangle + \gamma_2 |01\rangle + \gamma_3 |10\rangle + \gamma_4 |11\rangle . \tag{1.16}$$

The computational basis in this case is thus $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. It is common practice, when the dimension increases, to introduce a compact decimal notation that assigns to each vector in *ascending order* a number from 0 to $N-1$, with $N = 2^n$. In this way we are able to write $|\psi\rangle$ as:

$$|\psi\rangle = \sum_{i=0}^{N-1} \gamma_i |i\rangle . \tag{1.17}$$

The matrix expression of the basis states is simply an extension of the one qubit case:

$$|00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \quad |01\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \quad |10\rangle = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \quad |11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}. \tag{1.18}$$

The first two qubits logic gate to introduce is the ***CNOT* Gate** (Controlled Not). The $CNOT$ gate flips (NOT) the target qubit if and only if the controlled qubit is $|1\rangle$, and acts as an identity otherwise. The matrix representation is strightforward from the definition:

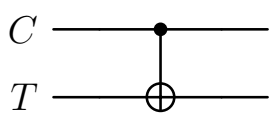
$$CNOT = \begin{pmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{pmatrix}. \tag{1.19}$$

To reinforce our understanding, we can build a simple truth table on the basis state that covers all the different cases:

C	T	In	Out
$\|0\rangle$	$\|0\rangle$	$\|0\rangle\|0\rangle$	$\|0\rangle\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle\|1\rangle$	$\|0\rangle\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle\|0\rangle$	$\|1\rangle\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle\|1\rangle$	$\|1\rangle\|0\rangle$

Table 1.1: We see that when the controlled (C) qubit is $|0\rangle$ the operator works as an identity, whereas when the controlled qubit is $|1\rangle$, the target (T) is flipped in the output.

The circuital representation of the $CNOT$ gate is:



, (1.20)

where the dot indicates the qubit in which the control is performed, and the cross is the target where the NOT operation is applied if $C = |1\rangle$.

An analogous gate is the ***C-PHASE*** (Controlled Phase), which works exactly like a $CNOT$, but with a phase shift applied to the target bit instead of a NOT operation.

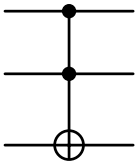
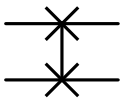
$$CPHASE = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\varphi} \end{pmatrix}. \tag{1.21}$$

From the matrix representation it is convenient to write again our truth table:

C	T	In	Out
$\|0\rangle$	$\|0\rangle$	$\|0\rangle\|0\rangle$	$\|0\rangle\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle\|1\rangle$	$\|0\rangle\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle\|0\rangle$	$(\mathbf{1}\otimes R_z(\varphi))\|1\rangle\|0\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle\|1\rangle$	$(\mathbf{1}\otimes R_z(\varphi))\|1\rangle\|1\rangle$

Table 1.2: The notation : $(\mathbf{1}\otimes R_z(\varphi))$ implies that the phase shift is only applied to the second (target) qubit, leaving the first one unchanged. Before we could have written the $CNOT$ operation in the same fashion as $(\mathbf{1}\otimes\sigma_x)$.

The list of quantum logic gates goes on, including more advanced options like ***CCX*** gates, which function similarly to a standard $CNOT$ gate but with two controlled qubits and one target; or ***SWAP*** gates, used for interchanging two qubits. For the sake of brevity, we won't dive further.

$CCX =$  , $SWAP =$  . (1.22)

1.4 Quantum algorithms

A quantum algorithm is a set of step-by-step instructions designed for execution on a quantum computer. Although, in general, quantum computers can always perform classical algorithms, the term quantum algorithm is usually used to describe those algorithms which are inherently quantum, that use features such as quantum superposition and entanglement.
Shor's algorithm [1], Deutsch-Josza [5], Grover [6], and the Quantum Fourier Transform are, among others, prime examples of powerful quantum algorithms that outperform their classical counterparts.
Before proceeding to show an example of such algorithms, to further concretize the potentiality of the inherently quantum processes, it's worthy to introduce a couple more concepts.

1.4.1 Qubits functions

Suppose to have a generic binary function $f : \{0,1\}^n \longrightarrow \{0,1\}$. To break it down into proper unitary gates, we need $f(x)$ to be *bijective*, which is not always the case, think about the classical AND. The solution for ensuring reversibility in the function is to store the input in a

dedicated register, effectively doubling the degrees of freedom n [4].
For instance, let's take a function $f(x) : \{0,1\} \longrightarrow \{0,1\}$, as said to guarantee reversibility we need to initialize two registers: the *data* register holding the input x, and the *target* register y, where the function output will be stored. By applying an appropriate sequence of logic gates, denoted as U_f, we can achieve the transformation $|x, y\rangle \xrightarrow{U_f} |x, y \oplus f(x)\rangle$, where $\oplus$ represents the XOR operation (or addition modulo 2). If $y = 0$, we can directly read the value of $f(x)$ from the target register [4].

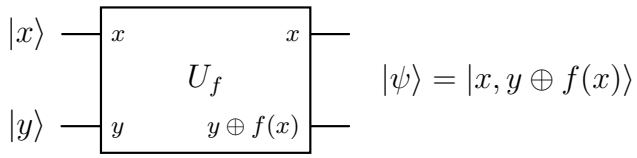


Figure 1.3: Scheme of a circuit designed to compute $f(x)$.

1.4.2 Quantum parallelism

If we run the circuit of Fig. 1.3 with $|x\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ (obtainable applying an Hadamard gate on a $|0\rangle$ state) and with $|y\rangle = |0\rangle$, we can witness a fascinating phenomenon called **quantum parallelism** [4]. In fact, if we write the resulting state after the U_f operator, we get:

$$|\psi\rangle = \frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}. \tag{1.23}$$

We can notice how the different terms of the superposition carry informations about both $f(0)$ and $f(1)$, the circuit thus evaluated $f(x)$ for two values of x *simultaneously*. This behaviour is completely different from the classical parallelism, here a **single** circuit can evaluate $f(x)$ for different values of x, taking *advantage* of the quantum superposition. This property finds direct application in many well-known algorithms, including the Deutsch-Jozsa [5] and Grover [6] algorithms.

1.4.3 The quantum Fourier transform

One of the most notable accomplishments in quantum computation and a fundamental component of this thesis is the quantum version of the discrete Fourier transform (DFT). To provide some context, the DFT takes in input a vector of complex numbers $\{x_0, ..., x_{N-1}\}$, and output another vector of complex numbers $\{y_0, ..., y_{N-1}\}$, defined by the mapping:

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i j k/N}. \tag{1.24}$$

As the discrete Fourier transform is a unitary operator, we can readily adapt it [4] to the case of an n-qubit register (where $N = 2^n$) and construct the quantum Fourier transform (QFT) as:

$$|j\rangle \xrightarrow{QFT} \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k/N} |k\rangle . \tag{1.25}$$

Equivalently, the action of the QFT on a generic state can be written,

$$\begin{aligned}\sum_{j=0}^{N-1} x_j \left|j\right\rangle &\xrightarrow{QFT} \sum_{j=0}^{N-1} x_j \underbrace{\left(\frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i jk/N} \left|k\right\rangle\right)}_{QFT\left|j\right\rangle} \\ &= \sum_{k=0}^{N-1} \underbrace{\left(\frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i jk/N}\right)}_{y_k} \left|k\right\rangle = \sum_{k=0}^{N-1} y_k \left|k\right\rangle .\end{aligned} \tag{1.26}$$

With a bit of basic algebra[3] (albeit dense in notation), we can write the effect of QFT as follows:

$$\left|j_1, ..., j_n\right\rangle \xrightarrow{QFT} \frac{\left(\left|0\right\rangle + e^{2\pi i 0.j_n} \left|1\right\rangle\right)\left(\left|0\right\rangle + e^{2\pi i 0.j_{n-1}j_n} \left|1\right\rangle\right) \cdots \left(\left|0\right\rangle + e^{2\pi i 0.j_1 j_2 \cdots j_n} \left|1\right\rangle\right)}{\sqrt{N}}, \tag{1.27}$$

where $j_1 j_2 ... j_n$ is the binary representation of j such that $j = j_1 2^{n-1} + j_2 2^{n-2} + \cdots + j_n 2^0$, and the notation $0.j_l j_{l+1} ... j_m$ is a way to represent the binary fraction $j_l/2 + j_{l+1}/4 + \cdots + j_m/2^{m-l+1}$ [4]. This construction provides us with all the instructions necessary to build the quantum circuit. If we call R_k the gate that denotes the unitary transformation (essentially a *phase-shift*)

$$R_k \equiv \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}, \tag{1.28}$$

the final circuit will be composed as:

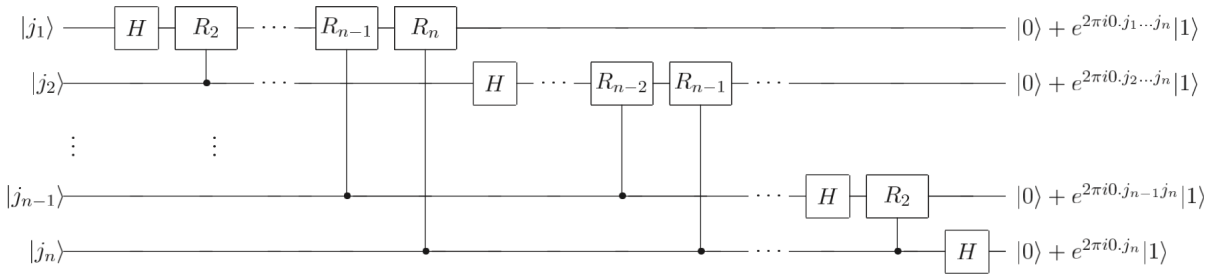


Figure 1.4: Circuit for the quantum Fourier transform [4], derived recreating the state of Eq. 1.27. The output qubits are swapped compared to the input ones, this can be corrected applying $n/2$ swap gates or renominating the qubits. We can see that R_k is used as a controlled phase shift.

From the circuit it is easy to count the total number of gate used (ignoring the eventual swaps). In fact, we have n gates applied to the first qubit, $n-1$ for the second etc., until the last one which has only one Hadamard gate applied. The QFT require thus $n(n+1)/2$ gates, and therefore this circuit provides a $O(n^2)$ algorithm to perform the quantum Fourier transform, even if we count the swaps.
On the contrary, the fastest classical algorithm known, the Fast Fourier Transform (FFT), can

[3]We won't cover the entire derivation here, but a very similar technique has been employed in the following chapter in Eq. 2.12 and can be utilized to derive Eq. 1.27.

compute the DFT using $O(n2^n)$ gates. This means that the QFT can achieve **exponential speedup** over its classical counterpart [4]. Unfortunately, the amplitudes in a quantum computer are not directly accessible through measurement, so a full "quantum replacement" for solving DFT-related classical problems would lose all the exponential speedup. Furthermore, just trying to generate arbitrary states such as the one in Eq. 1.26 is generally speaking an extremely difficult task. However, we can still take advantage of the exponential speedup using the QFT as a tool inside bigger quantum circuits, as we will see in the next chapter.

Chapter 2

From order finding to prime factorization

2.1 Fundamentals of modular arithmetic

To provide a complete theoretical description of the order finding problem, it is first necessary to introduce some basic concepts of modular arithmetic and number theory.
To begin, for any given integer number N, it is possible to define a set [7]:

$$\mathbb{Z}_N = \{1, ..., N-1\}, \tag{2.1}$$

in which operations, such as addition and multiplication, are taken *modulo* N
(e.g. $3 \cdot 2\,(mod\,5) = 1\,(mod\,5)$). This implies that sums or products between elements of $\mathbb{Z}_N$ are also elements of $\mathbb{Z}_N$.
In our context, we are not interested in the entirety of Z_N, but rather a subset of it, defined as follows:

$$\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N \mid gcd(a, N) = 1\}, \tag{2.2}$$

where $gcd(a, N)$ stands for the *greatest common divisor* of a and N.
The elements of $\mathbb{Z}_N^*$ are thus all the numbers $1 \leq a < N$ for which a and N are co-prime.
Of particular importance to us is the cardinality of $\mathbb{Z}_N^*$, which is commonly known as Euler's totient function $\phi(N)$ [8]. This function counts the positive integers less than or equal to N that are co-prime to N, finding remarkable applications, as illustrated in the following theorem:

Theorem 2.1.1 (Euler's theorem [9])**.** *If a and N are two positive integers and $a \in \mathbb{Z}_N^*$, then*

$$a^{\phi(N)} = 1\,(mod\,N), \tag{2.3}$$

with $\phi(N)$ being the Euler's totient function.

(e.g. for $N = 10$, $\mathbb{Z}_{10}^* = \{1, 3, 7, 9\}$ and therefore $\phi(10) = 4$. One can check and find out that all the elements of $\mathbb{Z}_{10}^*$ to the fourth power give $1\,modulo\,10$).
This theorem implies that the **order** of a, defined as the smallest integer r for which
$a^r = 1\,(mod\,N)$, exists and is finite for each $a \in \mathbb{Z}_N^*$, being by definition $r \leq \phi(N)$.
Furthermore, it can be shown that if a is an element of $\mathbb{Z}_N^*$, there exists a unique number x satisfying $a\,x = 1\,(mod\,N)$, called the *modular multiplicative inverse* of a [10].
This last concept will be crucial in the construction of the quantum circuit.

2.2 Theoretical analysis of the quantum circuit

Shor's factoring algorithm can be viewed as a special configuration of a *quantum phase estimation* algorithm. The concept underlying involves the existance of an algorithm that takes an unitary operator U_a and one of its eigenstates $|\psi_s\rangle$ such that:

$$U_a |\psi_s\rangle = e^{2\pi i \phi_s} |\psi_s\rangle , \tag{2.4}$$

and output an estimation of the angle[1] ϕ_s. A simplified scheme of the circuit is reported below:

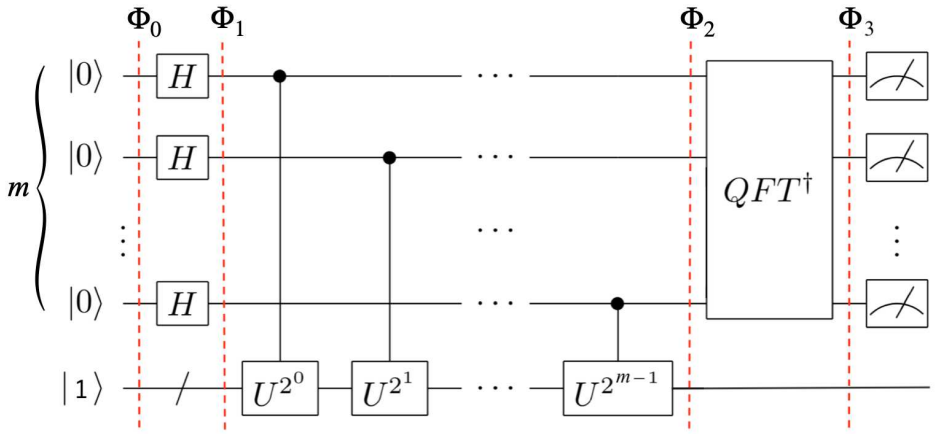


Figure 2.1: Simplified scheme of the circuit. The first register consists in m qubits initialized in $|0\rangle$, each undergoing a Hadamard gate operation. The second register encodes the binary number 1 for reasons that will be clear later. In correspondence of the red dashed lines, we will examine and record the states $|\Phi_i\rangle$ to track the time evolution of the system throughout the process.

Note that we're using the **qiskit convention** on qubits ordering, where the least significant qubit stays on the left side of the bit-string, thus on top of the register.
To understand how a phase estimation procedure can provide a solution to the order finding problem, we first have to construct the proper operator U_a and its eigenstates.
The former can be defined as [11]:

$$U_a |x\rangle = \begin{cases} |ax \ (mod\, N)\rangle & \text{if } 1 \leq x < N, \\ |x\rangle & \text{if } N \leq x < 2^n \end{cases}, \tag{2.5}$$

with $a \in \mathbb{Z}_N^*$ and $|x\rangle$ (encoded in n binary bits) the state that undergoes the application of U_a. We notice that the constraint in the choice of a being in $\mathbb{Z}_N^*$ guarantees that the operation is reversible. In fact, there is always a unique element $b \in \mathbb{Z}_N^*$ such that $a \cdot b = 1 \, (mod\, N)$, thus the relation below holds [7]:

$$U_{a^{-1}} U_a = U_{a^{-1}a} = U_1 = \mathbf{1}. \tag{2.6}$$

Furthermore, it can be shown that if a and N are co-prime, U_a is merely a permutation matrix, which is always unitary [11].

[1] From now on ϕ_s will be referred improperly as "phase", even if the actual phase is the entire complex exponential.

Proof. Suppose to have two different numbers x_1 and x_2: if $U_a |x_1\rangle = U_a |x_2\rangle$, then $ax_1 = ax_2 + kN$ for some integer $k \neq 0$. Therefore $x_1 - x_2 = kN/a$, i.e., kN/a is an integer. However, as N and a are co-prime, the least (positive) integer k which satisfies this is $k = a$, i.e., $x_1 - x_2 = N$. So it cannot be the case that $0 \leq x_1, x_2 < N$. This implies that for each integer x such that $0 \leq x < N$, $U_a |x\rangle$ gives a different integer in $0, ..., N-1$, therefore U_a is a permutation matrix. □

The case in which $N \leq x < 2^n$ is trivial since U_a acts as an identity.
We can now proceed to define our states $|\psi_s\rangle$ [4]:

$$|\psi_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i s k}{r}} \left| a^k \, mod \, N \right\rangle, \tag{2.7}$$

for $0 \leq s \leq r-1$. Turns out that these states are indeed eigenstates of U_a:

Proof.

$$\begin{aligned} U_a |\psi_s\rangle &= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i s k}{r}} U_a \left| a^k \, mod \, N \right\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i s k}{r}} \left| a^{k+1} \, mod \, N \right\rangle \\ &= \frac{1}{\sqrt{r}} \sum_{k=1}^{r} e^{-\frac{2\pi i s (k-1)}{r}} \left| a^k \, mod \, N \right\rangle = e^{\frac{2\pi i s}{r}} \frac{1}{\sqrt{r}} \sum_{k=1}^{r} e^{-\frac{2\pi i s k}{r}} \left| a^k \, mod \, N \right\rangle \\ &\overset{(a)}{=} e^{\frac{2\pi i s}{r}} \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i s k}{r}} \left| a^k \, mod \, N \right\rangle = e^{\frac{2\pi i s}{r}} |\psi_s\rangle, \end{aligned}$$

□

where in (a) we used the fact that results for $k = r$ and $k = 0$ are the same: $e^{-\frac{2\pi i s r}{r}} = 1 = e^{-\frac{2\pi i s \cdot 0}{r}}$, and $|a^r \, mod \, N\rangle = |1 \, mod \, N\rangle = |a^0 \, mod \, N\rangle$ by definition of r.

Here an issue arises, because in order to prepare these states we require the knowledge of r, which is precisely what we are attempting to determine.
Luckily, a simple solution allows us to get around the problem and proceed. In fact, we can show that an equal superposition of eigenstates $|\psi_s\rangle$ results in an easy-to-prepare state $|1\rangle$ [4]:

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\psi_s\rangle = |1\rangle. \tag{2.8}$$

Proof.

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \left(\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i s k}{r}} \left| a^k \, mod \, N \right\rangle \right) = \frac{1}{r} \left(r |1\rangle + \sum_{k=1}^{r-1} \underbrace{\left(\sum_{s=0}^{r-1} e^{-\frac{2\pi i s k}{r}} \right)}_{(*)} \left| a^k \, mod \, N \right\rangle \right)$$

If we can prove that $(*)$ is zero for $k > 0$, then we have our thesis.

Calling ω^s the phase $e^{-\frac{2\pi i s k}{r}}$, we define:

$\Sigma_s = \omega^0 + \omega^1 + \cdots + \omega^{r-1}, \qquad \omega \Sigma_s = \omega^1 + \omega^2 + \cdots + \omega^r$

$\Sigma_s - \omega \Sigma_s = \omega^0 - \omega^r \implies \Sigma_s = \dfrac{1 - \omega^r}{1 - \omega},$

but $\omega^r = e^{-\frac{2\pi i r k}{r}} = 1$ and $\omega \neq 1$, therefore $\Sigma_s = 0$.

□

Now that all the requirements have been taken care of, we can start analyzing the main steps of the circuit.
Initially we have a trivial configuration,

$$|\Phi_0\rangle = \underbrace{|0\rangle \cdot \cdots \cdot |0\rangle}_{m} |1\rangle \,, \tag{2.9}$$

with m qubits initialized in the state $|0\rangle$, and n qubits being used to display the state $|1\rangle$ in its binary form. Applying the Hadamard gates on the first register we have,

$$|\Phi_1\rangle = (H^{\otimes m} \otimes \mathbb{1}) |\Phi_0\rangle = \frac{1}{\sqrt{2^m}} \sum_{\gamma=0}^{2^m-1} |\gamma\rangle |1\rangle \,, \tag{2.10}$$

where γ is a decimal index as in Eq. 1.17.
Every qubit of the first register serves as a control bit for the controlled U_a operator as shown in Fig. 2.1, this acts on the superposition of the eigenstates $|\psi_s\rangle$ that is $|1\rangle$, resulting in:

$$\begin{aligned} |\Phi_2\rangle =& \frac{1}{\sqrt{2^m}} \left(|0\rangle + e^{\frac{2\pi i s}{r} 2^0} |1\rangle\right) \otimes \left(|0\rangle + e^{\frac{2\pi i s}{r} 2^1} |1\rangle\right) \\ & \otimes \cdots \otimes \left(|0\rangle + e^{\frac{2\pi i s}{r} 2^{m-1}} |1\rangle\right) |1\rangle \,. \end{aligned} \tag{2.11}$$

With a bit of manipulation we can obtain a much more compact form of the same state:

$$\begin{aligned} |\Phi_2\rangle &= \frac{1}{\sqrt{2^m}} \prod_{k=1}^{m} \left(\sum_{\gamma_k=0}^{1} e^{\frac{2\pi i s}{r} \gamma_k 2^{k-1}} |\gamma_k\rangle \right) |1\rangle \\ &= \frac{1}{\sqrt{2^m}} \sum_{\gamma_m=0}^{1} \cdots \cdot \sum_{\gamma_1=0}^{1} \left(\prod_{k=1}^{m} e^{\frac{2\pi i s}{r} \gamma_k 2^{k-1}} |\gamma_k\rangle \right) |1\rangle \\ &= \frac{1}{\sqrt{2^m}} \sum_{\gamma_m=0}^{1} \cdots \cdot \sum_{\gamma_1=0}^{1} e^{\frac{2\pi i s}{r} (\gamma_1 2^0 + \gamma_2 2^1 + \cdots + \gamma_m 2^{m-1})} |\gamma_1 \ldots . \gamma_m\rangle |1\rangle \\ &\overset{(a)}{=} \frac{1}{\sqrt{2^m}} \sum_{\gamma=0}^{2^m-1} e^{\frac{2\pi i s}{r} \gamma} |\gamma\rangle |1\rangle \,, \end{aligned} \tag{2.12}$$

where in (a) we shift to the decimal notation $\gamma = \gamma_1 2^0 + \gamma_2 2^1 + \cdots \gamma_m 2^{m-1}$. Before computing the $QFT^\dagger$, we rewrite our phase $\phi_s = s/r$ as $\alpha/2^m$, with $\alpha \in 0, \ldots . 2^m - 1$. The reasons for this will become clear later. Once this is done, we can proceed to execute the $QFT^\dagger$, that we remind has form:

$$QFT^\dagger = \frac{1}{\sqrt{2^m}} \sum_{\chi=0}^{2^m-1} e^{-\frac{2\pi i \chi \gamma}{2^m}} , \tag{2.13}$$

therefore,

$$\begin{aligned} |\Phi_3\rangle = QFT^\dagger |\Phi_2\rangle &= \frac{1}{2^m} \sum_{\chi=0}^{2^m-1} \sum_{\gamma=0}^{2^m-1} e^{\frac{2\pi i \alpha \gamma}{2^m}} e^{-\frac{2\pi i \chi \gamma}{2^m}} |\chi\rangle |1\rangle \\ &= \frac{1}{2^m} \sum_{\chi=0}^{2^m-1} \sum_{\gamma=0}^{2^m-1} e^{\frac{2\pi i \gamma}{2^m} (\alpha - \chi)} |\chi\rangle |1\rangle \,. \end{aligned} \tag{2.14}$$

Suppose now we make a measurement on the first register and obtain a m-bit binary number $\beta = \beta_1 2^0 + \beta_2 2^1 + \cdots \beta_m 2^{m-1}$. Computing the probability of getting β as a result, we have:

$$\begin{aligned}\mathcal{P}(\beta) = |\langle\beta|\Phi_3\rangle|^2 &= \left|\langle\beta| \frac{1}{2^m} \sum_{\chi=0}^{2^m-1} \sum_{\gamma=0}^{2^m-1} e^{\frac{2\pi i\gamma}{2^m}(\alpha-\chi)} |\chi\rangle\right|^2, \text{ with } \langle\beta|\chi\rangle = \delta_{\beta\chi} \\ &= \left|\frac{1}{2^m} \sum_{\gamma=0}^{2^m-1} e^{\frac{2\pi i\gamma}{2^m}(\alpha-\beta)}\right|^2 \implies \text{if } \beta = \alpha,\ \mathcal{P}(\beta) = 1.\end{aligned} \tag{2.15}$$

That means measuring β exhausts all the probability, and therefore we will measure exactly α. To be precise, we should take into account that the real phase may not be completely described by $\alpha/2^m$, as $\alpha/2^m$ is only an approximation accurate to m qubits [4].

Approximations accurate to m qubits

When reading a number α from a m-qubit register, where $\alpha \in \{0, ..., 2^{m-1}\}$, we are essentially interpreting it in its binary form: $\alpha = \alpha_0 2^0 + \alpha_1 2^1 + \cdots \alpha_{m-1} 2^{m-1}$. Dividing α by 2^m yelds a number $\tilde{\phi}$ in the interval $[0, 1)$.
In our context, where we aim to approximate ϕ_s, it's important to acknowledge the constraints imposed by only having m bits available. In fact, generally a δ factor is introduced:

$$\frac{\alpha}{2^m} = \frac{\alpha_0}{2^m} + \frac{\alpha_1}{2^{m-1}} + \cdots + \frac{\alpha_{m-1}}{2^1} + \underbrace{\cdots\cdots}_{\delta} = \phi_s. \tag{2.16}$$

Note that if $\alpha/2^m$ is the **best**[2] m bit representation of ϕ_s, then $0 \leq |\delta| \leq 1/2^{m+1}$ [12]. Furthermore, if we recall the geometric series $\sum_{n=1}^{\infty} 1/2^n = 1$, we should be easily convinced that with appropriate α_i coefficients and a sufficiently large m, we are able to write an arbitrarily accurate approximation of any real number in the interval $[0, 1)$.

[2] It is important to make clear that the best representation of a phase close to 1 might be 0, as we are working in the unitary circle.

Set the necessary foundations, we now know that a more accurate representation of the phase would be $\phi_s = (\alpha/2^m + \delta)$ [12]. Therefore, an important factor to explore further is: which is the required precision? That is, what is the minimum number of qubits that are sufficient to have a good estimation?
First of all, to ensure we don't mistake nearest possibilities, the closest number to s/r we have to assess is $s/(r+1)$ and the distance between these two numbers is:

$$\frac{s}{r} - \frac{s}{r+1} = \frac{s}{r(r+1)}. \tag{2.17}$$

Therefore, if we want our measurement to distinguish the two cases, we can write:

$$\left|\frac{\alpha}{2^m} - \frac{s}{r}\right| < \frac{s}{2r(r+1)}, \tag{2.18}$$

so that the error is less than a half of the distance between s/r and $s/(r+1)$, and $\alpha/2^m$ is closer to s/r than any other possibility [7]. However, we still don't know the value of r. We then have to consider the only information we hold, which is $r < N$, and make a greater restriction:

$$\left|\frac{\alpha}{2^m} - \frac{s}{r}\right| < \frac{s}{2N^2} < \frac{s}{2r^2}. \tag{2.19}$$

Taking $m = 2\lceil log_2(N)\rceil + 1 = 2n + 1$ guarantees that this inequality is satisfied by the estimation with an high chance [7]. In fact, since $\alpha/2^{2n+1}$ is an approximation of s/r accurate to $2n + 1$ qubits we have,

$$\left|\frac{\alpha}{2^{2n+1}} - \frac{s}{r}\right| \leq \frac{1}{2^{2n+1}} \leq \frac{1}{2N^2} \leq \frac{1}{2r^2}, \tag{2.20}$$

where we used the fact that $N < 2^n$. It's worth noting that choosing $m = 2n$ satisfies the inequality too, and actually, this is the specific value utilized in the construction of the real quantum circuit.
Now that we know how many qubits are needed for the first register in order to work properly, we have to understand how to find our values of s/r from the estimation of the phase ϕ_s. Luckily, due to the following theorem, we have a way to do it:

Theorem 2.2.1 (Continued fractions theorem [4])**.** *Suppose s/r is a rational number such that:*

$$\left|\frac{s}{r} - \phi\right| \leq \frac{1}{2r^2}, \tag{2.21}$$

then s/r is a convergent of the continued fractions for ϕ, and thus can be computed in $O(n^3)$ operations using the continued fractions algorithm.

In other words this theorem allows us to successfully apply the continued fractions algorithm that turns our approximation $\alpha/2^m$ into nearby fractions s'/r', including s/r.

2.2.1 Performance

Before proceeding with the determination of the factors of N, it is natural to ask ourselves: when does the order-finding algorithm fails? In fact, the chances of failure lay both on the phase estimation and on the continued fractions. We have seen before that the phase estimation outcomes for ϕ_s not directly writable as $\alpha/2^m$ won't be certain, however for such a ϕ_s, it turns out that applying the $QFT^\dagger$ produces the best m-bit approximation of ϕ_s with probability at least $\frac{4}{\pi^2} \approx 0.405...$ [12]. To see this we can repeat the calculation of Eq. 2.15 in the case in which $0 < |\delta| \leq 1/2^{m+1}$. For instance,

$$\mathcal{P}(\alpha) = \left|\frac{1}{2^m}\sum_{\gamma=0}^{2^m-1} e^{2\pi i\gamma(\phi_s - \alpha/2^m)}\right|^2. \tag{2.22}$$

Using the properties of the finite geometric series as in proof. 2.2, we can rewrite the probability as:

$$\mathcal{P}(\alpha) = \frac{1}{2^{2m}}\left|\frac{e^{2\pi i(2^m\phi_s - \alpha)} - 1}{e^{2\pi i(\phi_s - y/2^m)} - 1}\right|^2. \tag{2.23}$$

Now, taking into account the two inequalities [7] deriving from the relation between arcs and chords of the unit circle for $\delta \in [-\frac{1}{2}, \frac{1}{2}]$:

$$\left|e^{2\pi i\delta} - 1\right| \leq 2\pi|\delta|, \qquad 4|\delta| \leq \left|e^{2\pi i\delta} - 1\right|, \tag{2.24}$$

we can show that if $|\phi_s - \alpha/2^m| \leq 1/2^{m+1}$:

$$\mathcal{P}(\alpha) = \frac{1}{2^{2m}}\left|\frac{e^{2\pi i(2^m\phi_s - \alpha)-1}}{e^{2\pi i(\phi_s - y/2^m)-1}}\right|^2 \geq \frac{1}{2^{2m}}\left(\frac{4|2^m\phi_s - \alpha|}{2\pi|\phi_s - \alpha/2^m|}\right)^2 = \frac{1}{2^{2m}}\frac{16\cdot 2^{2m}}{4\pi^2} = \frac{4}{\pi^2}. \tag{2.25}$$

This means that the best outcome always occurs with a probability greater than 40%.
In general the causes of failure can be summarized in three main categories [4]. The first one is precisely the possibility of a bad estimation, but we exhaustively demonstrated how to manage and contain this type of issue. Another cause can be $s = 0$ or $s' = 0$ which tells us nothing about r. Finally, there is the possibility that s and r share common factors. In such cases, the continued fractions algorithm returns r', which will not be equal to r, but rather to a factor of it.

2.3 Post-quantum classical analysis

The last thing we need to cover is how solving the order-finding problem helps us with factoring. This part is purely classical and doesn't involve quantum computing. In particular, our aim is to break down the number N into its prime factors. Initially, we try to find two integers $b, c \geq 2$ such that their product equals N. Of course if N is a prime number, this operation will fail, but if it's not, we can proceed to split it. After splitting N, we repeat the process on the resulting factors until we obtain the prime factorization of N [7].
For brevity, let's assume that we've already selected a value of a from the set $\mathbb{Z}_N^*$, that the factors of N aren't trivial and N isn't a perfect power. With these conditions[3] in place, our approach is as follows [4]:

1. Let r be the order of $a^x \, mod \, N$, found with the QPE procedure.
2. If r is even and $a^{r/2} \neq -1 \, mod \, N$ ([4]):
   a) Compute $\gcd(a^{r/2} - 1, N)$.
   b) Compute $\gcd(a^{r/2} + 1, N)$.
3. See if one of these is a non-trivial factor, returning the factor if so.
4. If this point is reached, the algorithm failed to estimate the factors.

To see why this procedure leads to the solution, let's consider the numbers:

$$a^{r/2} - 1 \, (mod \, N), \qquad a^{r/2} + 1 \, (mod \, N), \tag{2.26}$$

as $X^2 - 1 = (X - 1)(X + 1)$, we can write

$$(a^{r/2} - 1)(a^{r/2} + 1) = a^r - 1. \tag{2.27}$$

But we know that $a^r \, (mod \, N) = 1$ by definition of order. So $a^r \, (mod \, N) - 1 = 0 \, (mod \, N)$. This implies that N is a divisor of $a^r \, (mod \, N) - 1 = (a^{r/2} - 1)(a^{r/2} + 1)$. Consequently, every prime factor of N must divide either $(a^{r/2} - 1), (a^{r/2} + 1)$ or both. Hence, we can factor N by computing the gcd, a task efficiently accomplished using Euclid's algorithm.

[3] If $a \notin \mathbb{Z}_N^*$ the $gcd(a, N)$ is a factor; if N is even the factors are $\{2, N/2\}$.

[4] This condition, relative to the numbers in Eq. 2.26, implies that none of the factors obtained is a multiple of N. This scenario causes the algorithm to fail. We can be tempted to write the condition $a^{r/2} \neq 1 \, mod \, N$, but it's unnecessary [11], since r is the smallest number that satisfy $a^x = 1 \, mod \, N$.

2.4 Implementation of the quantum circuit

Although the theoretical formulation of the problem is quite straightforward, implementing the controlled U_a operators can be challenging and resource-intensive, requiring a significant amount of qubits and gates.
The Qiskit circuit developed for this thesis is based on the modular exponentiation model proposed by Vedral, Barenco, and Ekert in 1996 [3]. This model involves an arrangement of custom gates designed for specific simple operations. In the following section, we will outline the fundamental behavior of each of these operators and explain how they are combined together to form the circuit.

2.4.1 Plain adder

The simplest gate to consider is the plain adder module, which takes two numbers and add them together:

$$|a, b\rangle \longrightarrow |a, a + b\rangle \,. \tag{2.28}$$

Note that the reversibility is guaranteed, as we are keeping track of the value of a in the first register.
If both a and b are encoded in n qubits, the second register must have $n + 1$ qubits to prevent overflow[5]. In addition, a third register (c) of $n - 1$ qubits initialized as $|0\rangle$ is needed to write provisionally the carries of the addition. A schematic illustrating the algorithmic construction of the plain adder is depicted in Fig. 2.2:

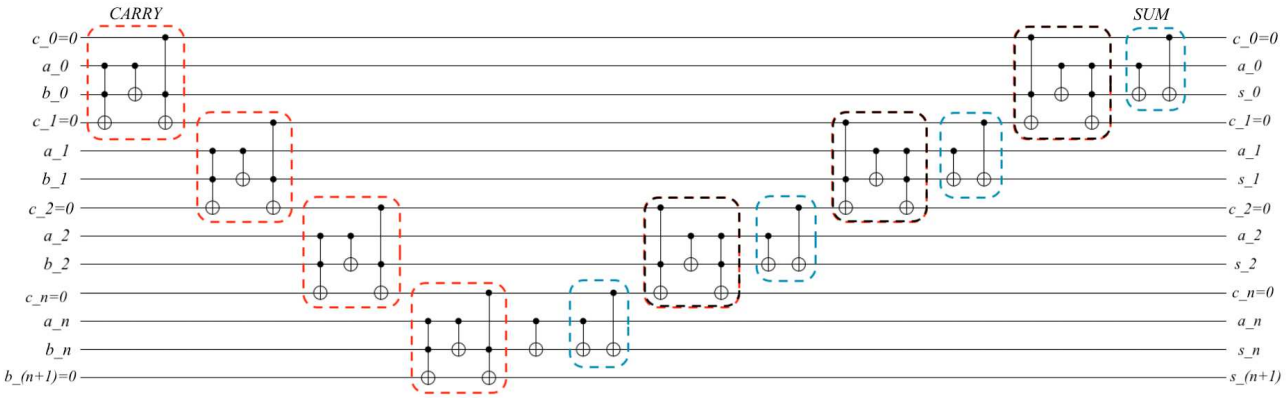


Figure 2.2: The plain adder network computes carries iteratively until the last carry determines the most significant digit of the result. Then all these operations apart from the last one are undone in reverse order, and the sum of the digits is performed correspondingly. This picture also show the fundamental operations used: *CARRY* (red), *INVERSE CARRY* (black), *SUM* (blue). This structure can be implemented for any n.

[5]For instance, consider the scenario where the a and b registers each consist of $n = 3$ qubits, and both are assigned the value $a = b = 6$. It becomes apparent that the b register lacks sufficient capacity to store the result of the addition. This situation is commonly referred to as overflow.

It's worth pointing out that the adder operation can be easily reversed by computing the circuit from right to left. This approach yields $(a,\, a-b)$ for $a \geq b$ and $(a, 2^{n+1} - (b-a))$ otherwise.

2.4.2 Modular adder

As the name might suggests, this gate takes in input two numbers and adds them together *modulo* N:

$$|a, b\rangle \longrightarrow |a, a + b \, mod \, N\rangle \,, \tag{2.29}$$

where $0 \leq a, b \leq N$. Note that this last condition also guarantees the reversibility, as an unconstrained choice of b could results in equal outcomes for different initial values. A schematic interpretation of the modular adder is illustrated in Fig. 2.3. The size and complexity of the circuit already prevents us to represent it explicitly.

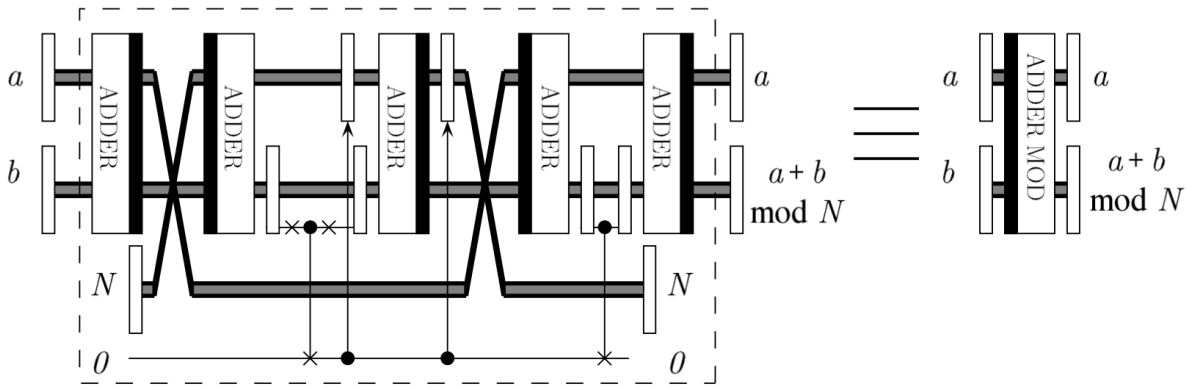


Figure 2.3: Modular adder network. The black bars indicate the orientation of the module attached: right-facing black bar signifies a standard adder application, while a left-facing one indicates the addition of an inverse adder (adder in reverse order).

After the first adder, the register encoding a is swapped with a third register containing N, giving $|N,\, a+b\rangle$, then an inverse adder is applied leading to the state $|N,\, a+b-N\rangle$. If an overflow happened in the subtraction $(a+b<N)$, this information is copied into a temporary register $|t\rangle$ initially prepared in state $|0\rangle$. Particularly, **in case of no overflow** $(t=1)$ the first register is reset[6] to zero. Consequently, the third adder has no effect, as we are adding zero to $(a+b-N) = (a+b \, mod \, N)$. Following that, N is conditionally restored, leaving the state $|N,\, a+b \mod N\rangle$, a subsequent swap reintroduce the original value of a in its register, resulting in the desired state $|a,\, a+b \mod N\rangle$. **In case of overflow** $(t=0)$, the third adder adds back N, resulting in the state $|N,\, a+b \mod N\rangle$, from this point the procedure follows the other case. The last two adders are used to bring back to zero the temporary qubit in a reversible way. In fact, if the first time there was no overflow $(a+b) > N$, then the second time with $|a, (a+b \, mod \, N) - a\rangle$ there will certainly[7] be. After the reset, an adder restore our desired state.

[6] It's possible to do so with control not gates since we know its original state encodes N.

[7] If $a+b-N>0$, then subtracting a gives $b-N<0$, as we are operating for values $a, b < N$.

2.4.3 Controlled modular multiplier

The following custom gate corresponds precisely to the previously introduced controlled unitary operator U_a. The function $f_{a,N}(x) = ax \, mod \, N$ can be efficiently implemented by adopting a binary perspective, exploiting the conditional modular additions: $ax = 2^0 ax_0 + 2^1 ax_1 + \cdots + 2^{n-1} ax_{n-1}$.

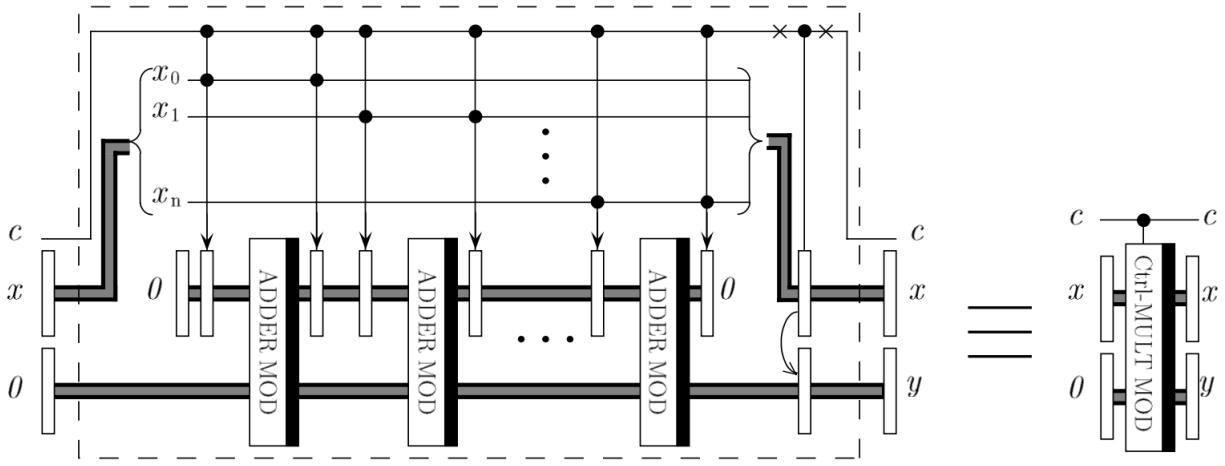


Figure 2.4: Controlled modular multiplier network. Each arrow indicates that an entire register is classically encoded if the two controls are satisfied. The bent arrow at the end of the circuit is a conditional "copy".

The functioning of the circuit is quite straightforward and can be summarized in the following equivalence:

$$|c; x, 0\rangle = \begin{cases} |c; x, ax \, mod \, N\rangle & \text{if } |c\rangle = |1\rangle \\ |c; x, x\rangle & \text{if } |c\rangle = |0\rangle \, . \end{cases} \tag{2.30}$$

Conditionally on the control qubit $|c\rangle$ and the x encoding qubits, through some CCX gate, before (and after) every modular adder, the value of $2^i a$ is classically implemented (or removed). If $|c\rangle = |0\rangle$ the register encoding x is copied in the lower register.

2.4.4 Modular exponentiation

The modular exponentiation is the effective calculation that happens applying the controlled U_a gates as in Fig. 2.1 to the state $|1\rangle$, since $f_{a,N}(x) = a^x \, mod \, N$ ([8]) can be rewritten as $a^x = a^{2^0 x_0} \cdot a^{2^1 x_1} \cdots a^{2^{m-1} x_{m-1}}$. These operations can be performed utilizing the previously constructed controlled multiplier, as illustrated in Fig 2.5.

[8] Note that there is an abuse of notation with the parameter x in the (2.4.3) and (2.4.4), as they are not the same object.

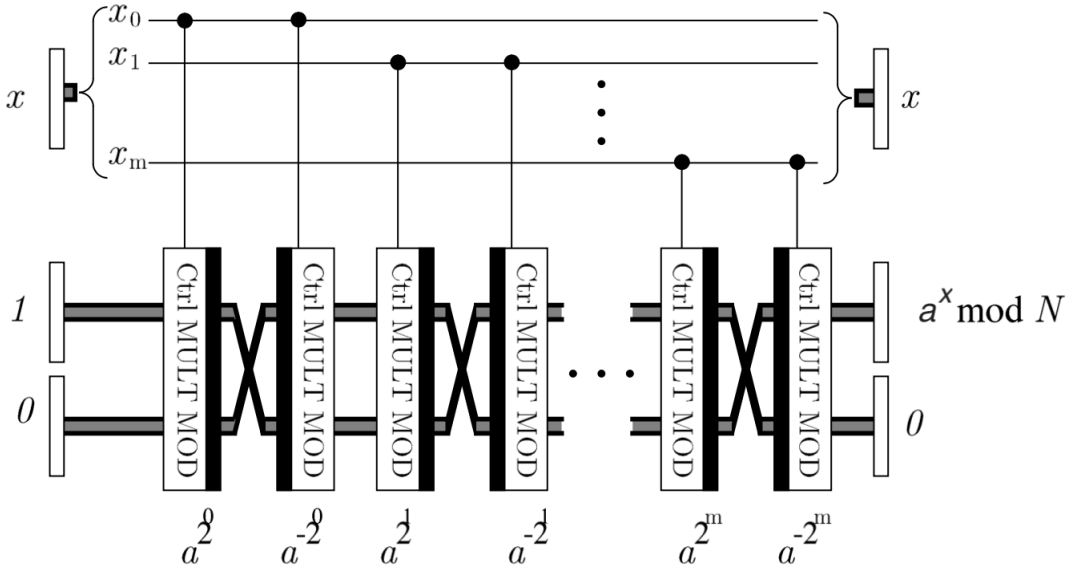


Figure 2.5: Modular exponentiation consists of successive modular multiplications by a^{2^i} . The even networks perform the reverse control modular multiplication by inverse of $a^{2^i}\, mod\, N$ thus resetting one of the registers to zero and freeing it for the next control modular multiplication.

In particular we have that, if $x_i = 1$, the operation performed is:

$$\left|a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}}, 0\right\rangle \longrightarrow \left|a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}}, a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}+2^ix_i}\right\rangle . \tag{2.31}$$

Otherwise, if $x_i = 0$ we obtain:

$$\left|a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}}, 0\right\rangle \longrightarrow \left|a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}}, a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}}\right\rangle . \tag{2.32}$$

The result can be written as $\left|a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}}, a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}+2^ix_i}\right\rangle$ in either cases.

To avoid storing partial data in the processes, an inverse control multiplier is attached after each control multiplier. However, it is crucial to emphasize that in this scenario, we must initialize the inverse controlled multiplier with the modular multiplicative inverse of a^{2^i}. The overall process looks like this,

$$\left|a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}}, 0\right\rangle \longrightarrow \tag{2.33}$$

$$\left|a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}}, a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}+2^ix_i}\right\rangle \quad \text{(Multiplication)} \tag{2.34}$$

$$\left|a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}+2^ix_i}, a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}}\right\rangle \quad \text{(swapping)} \tag{2.35}$$

$$\left|a^{2^0x_0+2^1x_1+\cdots+2^{i-1}x_{i-1}+2^ix_i}, 0\right\rangle . \quad \text{(resetting)} \tag{2.36}$$

Why is it necessary to compute the multiplicative inverse for the inverse modular multiplier, rather than solely relying on the inverse of the modular multiplier?
An intuitive explanation can be readily provided. Let's consider the action of a modular multiplier:

$$U_{a,N}\,|x, 0\rangle = |x, ax\, mod\, N\rangle . \tag{2.37}$$

The inverse operation should be able to take $|x, ax \, mod \, N\rangle$ and yield $|x, 0\rangle$. We can observe this by considering how the operator is constructed and operates at a lower level, as follows:

$$U_{a,N}^{-1} \left| x, 2^0 a x_0 + \cdots + 2^n a x_n \right\rangle = \left| x, 2^0 a x_0 + \cdots + 2^n a x_n - 2^0 a x_0 - \cdots - 2^n a x_n \right\rangle = |x, 0\rangle \, . \tag{2.38}$$

We can now transpose this reasoning on our initial problem. In particular, after swapping the registers, the state we are dealing with is $|ax \, mod \, N, x\rangle$. If we apply $U_{a,N}^{-1}$, we obtain:

$$\begin{aligned} U_{a,N}^{-1} \, |ax \, mod \, N, x\rangle &= U_{a,N}^{-1} \left| 2^0 a x_0 + \cdots + 2^n a x_n, x \right\rangle \\ &= \left| 2^0 a x_0 + \cdots + 2^n a x_n, x - 2^0 a x_0 a - \cdots - 2^n a x_n a \right\rangle . \end{aligned} \tag{2.39}$$

This procedure certainly does not result in the second register being zero. Let's try instead to apply $U_{a^{-1},N}^{-1}$:

$$\begin{aligned} U_{a^{-1},N}^{-1} \, |ax \, mod \, N, x\rangle &= \left| 2^0 a x_0 + \cdots + 2^n a x_n, x - 2^0 a x_0 a^{-1} - \cdots - 2^n a x_n a^{-1} \right\rangle \\ &= \left| ax \, mod \, N, x - \underbrace{(2^0 x_0 + \cdots + 2^n x_n)}_{x} \right\rangle = |ax \, mod \, N, 0\rangle \, . \end{aligned} \tag{2.40}$$

In this way we were able to successfully obtain the desired state of Eq. 2.36.

Chapter 3

Simulation and results

The algorithmic nature of the VBE (Vedral, Barenco, and Ekert) [3] modular exponentiation method enables the creation of specific quantum circuits for factorizing any integer N. By inputting the value of N, we can automatically generate a quantum circuit optimized for its factorization, offering a pathway to scalability.
To implement the quantum circuit we used the Qiskit framework, an open-source python-based SDK developed by IBM for creating and manipulating quantum programs. Whereas for the execution, we relied on the software Quantum Matcha Tea: a quantum computer emulator powered by matrix product states, in which it is possible to perform any measurement accessible on a quantum computer, such as projective measurements. Further, by accessing the entanglement entropy between different subsystems, the emulator allows the monitoring of the entanglement generated during the algorithm execution.
The tensor network ansatz used for the representation of the wave function is the matrix product state ansatz. This approach reduces the memory requirement from an exponential scaling with the system size to a linear scaling, though quadratic in the bond dimension, the critical parameter that controls the representation of quantum correlations.
In this framework, MPS simulations are *not constrained by the number of qubits but rather by the amount of entanglement* accounted for in the wave function, making them possibly well-suited for handling very large circuits. Particularly in our case, we were able to successfully simulate circuits with more than 100 qubits, showcasing both the emulator's proficiency in handling substantial computational complexities and the correct crafting of the quantum circuit.

3.1 The program for Shor's algorithm

In this section, we will provide a brief overview of the program's interface. Fig. 3.1 illustrates the steps to initialize it and obtain the desired results.

```
--------------
Version 2.0.0
--------------
Enter N, the modulus: 15
How to choose the value for a:
1. Enter desired value
2. Generate a random one
-> 2
Value of a extracted: 2 is coprime to N: 15
Proceding to the next step.....
Do you want to use only the modular exponentiation? [y/n] n
Enter max bond dimension value: 100
Wait.....
Where do you want to launch the simulation? [local/cluster] local
Running simulation.....
Total gate counts after pre-processing = 110211 nearest neighbor gates
Modular exponentiation executed
Execution time: 2.6922 minutes
```

Figure 3.1: Example of interface with $N = 15$.

To begin, the program prompts for the input of two values: N and a, where the value of a can be specified directly or chosen randomly. After confirming that a and N are co-prime, users are presented with the option to use the program solely as a modular exponentiation calculator ($a^x \, mod \, N$) or to execute Shor's algorithm in its entirety. The next parameter to enter is the maximum bond dimension value, which modulates the amount of entanglement that can be represented by the circuit ansatz. For instance, setting a low bond dimension risks compromising circuit functionality by an excessive truncation.
Finally, it is possble to choose the execution environment: either local or cluster. This last option works specifically on the INFN cluster of Padova.
Before presenting the results, the program provides information on both the number of gates used by the circuit and the execution time.
After the simulation is completed, the results are shown as depicted in the Fig. 3.2.

```
PHASE ESTIMATION 1:
Register Reading: 00000000
Result obtained 245/1024 times
Corresponding phase: 0.0
The value of r: 1 is uneven
Unsuccessfull result
---------------

PHASE ESTIMATION 2:
Register Reading: 01000000
Result obtained 253/1024 times
Corresponding phase: 0.25
The value of r: 4 is even
Successful factors finding!
The estimated factors are: 3 and 5
---------------

PHASE ESTIMATION 3:
Register Reading: 10000000
Result obtained 250/1024 times
Corresponding phase: 0.5
The value of r: 2 is even
Successful factors finding!
The estimated factors are: 3 and 5
---------------

PHASE ESTIMATION 4:
Register Reading: 11000000
Result obtained 276/1024 times
Corresponding phase: 0.75
The value of r: 4 is even
Successful factors finding!
The estimated factors are: 3 and 5
---------------
The correct estimation of the order is r: 4
Successfully obtained results with probability: 76.1 %
```

Figure 3.2: Results of the simulation for $N = 15$ and $a = 2$.

As common for quantum emulators, the simulation performs 1024 projective measurements on the final state, offering statistical insights other than a single bitstring. In the specific case illustrated in Fig. 3.2, we can see that the outcome presents four main readings, evenly distributed. Assuming the correctness of the final estimation with order $r = 4$, it is easy to see how each distinct phase estimation corresponds to a specific value of s.

- PHASE ESTIMATION 1: This corresponds to a value of $s = 0$. In this case the algorithm fails and the function responsible for the continued fractions returns $r = 1$ by default.
- PHASE ESTIMATION 2: This aligns with a value of $s = 1$, which produces a phase $\phi_s = s/r = 0.25$. Here the algorithm successfully finds the factors of N.
- PHASE ESTIMATION 3: This case ($s = 2$) is rather peculiar, given that s and r share common factors. In fact, as anticipated in Sec. 2.2.1, for such conditions the continued fraction algorithm yields r' that is a factor of r. Despite the algorithm's failure to identify the order, it remains worthwhile to proceed with classical processing, as it may still provide the correct factors.
- PHASE ESTIMATION 4: Finally, the case correspondent to a value $s = 3$, which yelds the correct results.

In general, the correct prime factors have been successfully identified in 76.1% of the cases. Furthermore, in the case of $N = 15$, we have estimated the order of the function $f(x) = 2^x \, mod \, 15$ to be $r = 4$. The correctness of this results is evident from the plot in Fig. 3.3:

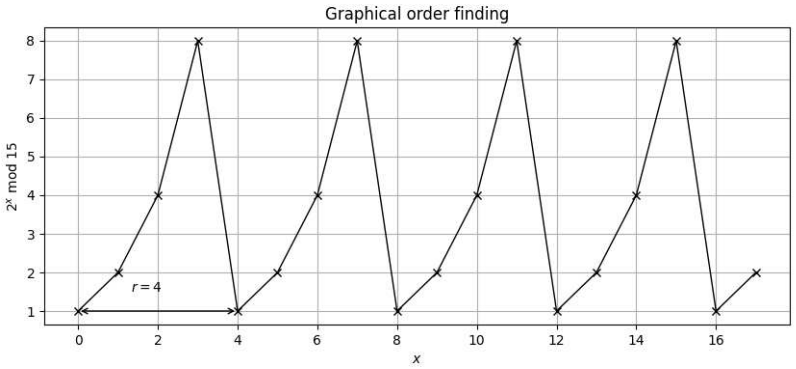


Figure 3.3: Graphical representation of the function $f(x) = 2^x \, mod \, 15$. The periodicity confirms the correct estimation of the order found by phase estimation.

The high accuracy of this simulation is attributable, in part, to the ability to consistently express the phase ϕ_s precisely as $\alpha/2^m$. This characteristic minimizes uncertainty, focusing it solely on the superposition of different s values.

3.2 Network complexity analysis

An essential topic we have to address is the complexity of our circuit: how the number of qubits and gates scales with the bit length n required to encode the integer number N.
Before delving into our analysis, it's important to note that in the VBE paper [3], indices range from zero to a certain number. For example, in Fig. 2.2, the value of a is actually encoded in $n + 1$ bits, as the indices range from a_0 to a_n. So in the following discussion, to avoid any misunderstandings, we'll consider the number of qubits only in relation to $n = \lceil \log_2(N) \rceil$.

3.2.1 Qubits resources requirements

Extracting a formula to express the total number of qubits q required to launch Shor's algorithm as a function of n is relatively easy, since its sufficient to count the registers dimensions in the modular exponentiation modules. For instance, the plain adder requires $3n + 1$ qubits, the modular adder introduces $n + 1$ more, the controlled modular multiplier adds $n + 1$ qubits[1], and ultimately the controlled qubits are $m = 2n$, as already mentioned in Sec. 2.2. Adding them together we obtain the following scaling for the required qubits, $q(n) = 7n + 3$. So we end up with a linear trend:

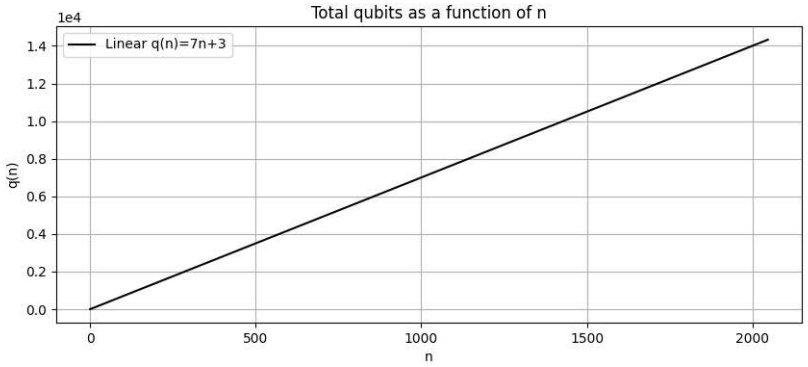


Figure 3.4: Relation between total qubits count and the bit length n.

The RSA encryption keys are typically encoded in 2048 bits. Consequently, from Fig. 3.4 we can see that a circuit designed to break such encryption requires approximately 14k logical qubits.

3.2.2 Gates resources requirements

Unlike for qubits, the exact number of gates cannot be directly expressed as a function of n, as it also depends on the specific values chosen for N and a. However, we can observe how the gate count scales with n in terms of order of magnitude. Following the approach used in the previous section, we can analyze the contribution of each component of the modular exponentiation [3]. The count of elementary gates in the plain adder, modular addition, and controlled-modular addition network increases linearly with n. Consequently, the controlled modular multiplication which involves n controlled modular additions, results in $O(n^2)$ elementary operations. Similarly, the exponentiation network that uses roughly n controlled modular multiplications, leads to a total count of elementary operations on the order of $O(n^3)$. Knowing from page 10 that the quantum Fourier transform scales as $O(n^2)$, we can conclude that the entire circuit uses $O(n^3)$ elementary operations.
The transpilation process involves rewriting a circuit in an equivalent manner, potentially using different types of basis gates or satisfying certain conditions. For instance, we might transpile the circuit to utilize only nearest neighbor gates or avoid three-qubit gates. Does the $O(n^3)$ complexity still hold in this context?

[1]The controlled qubit used in the controlled modular multiplier differs from the one employed in the modular exponentiation. While theoretically they serve the same purpose, in practice, a dedicated qubit acts as a bridge between the two and has to be taken into account.

To check this, we analyzed the number of gates in different transpilation conditions.
In Fig. 3.5 it becomes evident that the scaling remains unchanged despite the transpilation procedure.

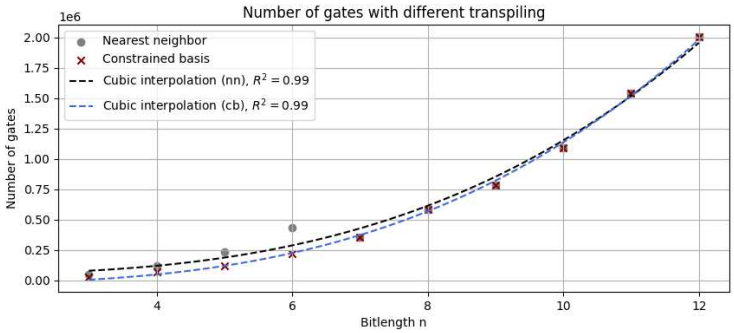


Figure 3.5: Scalings in two different transpiling conditions. The grey dots represent the number of gates for a nearest neighbor transpilation: meaning no ccx or long range cx, cp and swaps. The brown crosses represent the number of gates for a specific basis gate transpilation: in particular {cx, σ_i, cp, swaps}. Both cases fit best into a cubic interpolation with a value of $R^2 = 0.99$, where R^2 is the coefficient of determination that quantify the goodness of the model.

3.3 Memory costs

The computational demands of implementing Shor's algorithm program are considerable, particularly concerning RAM usage. For larger circuits ($n \geq 14$), it can easily exceed 200 GB at its peak.
During our simulations, we monitored memory usage continuously over time and observed a distinct pattern that can be divided into two main regions. The first region corresponds to the peak memory usage, typically occurring soon after the start of the simulation. This peak, crucial for selecting an appropriate computing environment (a machine with enough RAM available), is likely attributed to circuit preparation and transpilation processes. The second region follows a rapid drop from the peak and stabilizes at an approximately constant value. This stable memory usage, named *RAM cut* due to the sharp decrease, represents the memory required for the emulator to conduct computations. In Figure 3.6, we present data illustrating the RAM peak memory and RAM cut relative to n.

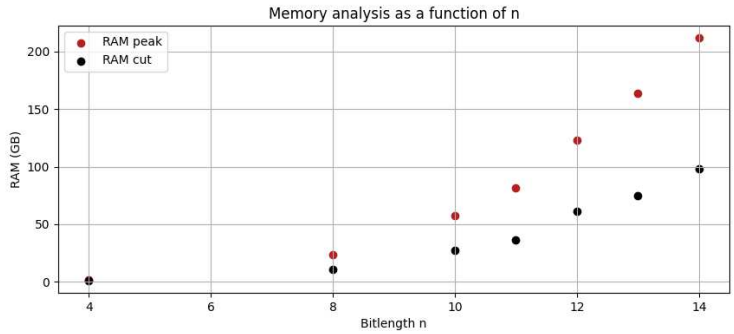


Figure 3.6: RAM peak and RAM cut over n. We can see the fast increase of the peak that surpass 200 GB for $n = 14$. The trend is polynomial in both cases.

It's important to emphasize that these values are not solely dependent on the parameter n, but are tightly linked to the maximum bond dimension set. Specifically, storing an MPS state requires $O(2q(n)\chi^2)$ numbers/memory, where χ denotes the bond dimension [13]. For this reason, having used different max bond dimensions for each simulation represented, we didn't associate any trendline.

3.4 Searching for optimal values

As mentioned earlier, MPS simulations are not restricted by the number of qubits but by the amount of entanglement generated in the quantum evolution. Keeping in mind that for any N a random value of $a \in \mathbb{Z}_N^*$ is chosen, we can analyze whether the chosen a minimizes entanglement usage, thereby optimizing the algorithm for the emulator.
To achieve this, we set a fixed value of $N = 119$ and conducted multiple simulations varying a in magnitude. Employing Quantum Matcha Tea functions, we extracted the maximum entanglement levels both after the modular exponentiation and after the QFT. If a significant correlation between a and the entanglement exists, it should be self-evident once collected enough data. In Fig. 3.7 the results are illustrated:

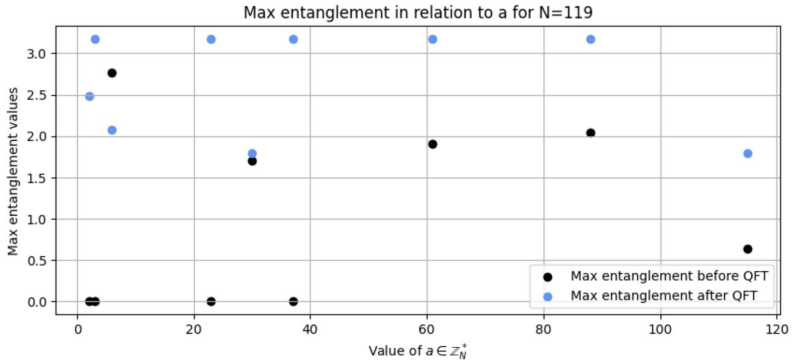


Figure 3.7: Max entanglement for different values of a at fixed $N = 119$. The black dots are the values before QFT, whether the blue ones corresponds to those after the QFT.

Based on the plot analysis, it appears that the data points are distributed without exhibiting any particular pattern.
Luckily, a more comprehensive answer emerged from new investigations of continuous monitoring of entanglement throughout the time evolution. In fact, we observed a significant correlation between entanglement levels and the order r. In particular, various configurations of a and N sharing the same order exhibited nearly identical entanglement patterns across all analyzed scenarios. Unfortunately, lacking prior knowledge of the order prevents us from capitalizing on this potentially advantageous property. Nevertheless, aiming for a deeper comprehension of this behavior certainly constitutes an intriguing objective for future research.

Conclusion and future works

For this thesis hundreds of simulations were performed and the results were always obtained, even if, according to the theory, fluctuations of the probabilities were encountered. The largest circuit we were able to execute was the one that factorize $N = 8453$, corresponding to a bit length $n = 14$, and therefore requiring a total number of qubits equals to 101. The simulation took around 30 hours running on the GPU of the INFN cluster of Padua, giving a success rate of 50.1% with a max bond dimension set at 10000, fairly small compared to the maximum bond dimension $2^{q(n)/2}$ for an MPS. Unfortunatly, it also consumed an amount of peak memory (RAM) of 212 GB, nearing the available limit of 256 GB (effective 245 GB). Consequently, simulations involving numbers with $n > 14$ were unachievable due to memory constraints. Nevertheless, it is clear that a computer with greater capacity could accommodate more expensive circuits in terms of qubits and gates.

Another noteworthy accomplishment of this thesis is undoubtedly the demonstration of the tensor network emulator Quantum Matcha Tea's capabilities. Through the utilization of Matrix Product States (MPS), we successfully simulated circuits of significant complexity. Far beyond the actual possibilities of a real quantum hardware, that wouldn't be able to compute the Shor's algorithm circuit even in the simplest cases.

Ultimately, we validated the network complexity across different transpilation configurations through experimental means and we investigated the relationship between the selection of a and entanglement, finding interesting results that instead seem to show a strong correlation between entanglement and *order*.

Looking ahead to future research, other than the already mentioned further investigation of this last concept, it would be newsworthy to explore the creation of a circuit for Shor's algorithm utilizing diverse models of modular exponentiation. While VBE is a possible approach, it is just one of several models available for executing this crucial step. For instance, there are QFT-based circuits that demonstrate efficacy with fewer qubits [14]. Given that these alternative methods may necessitate increased levels of entanglement, it would be interesting to conduct a comparative analysis of the emulator's efficiency with both approaches to discern any disparities. Specifically, considering that MPS are indifferent to the number of qubits but rather sensitive to the degree of entanglement, we could explore the trade-off between fewer qubits and possible greater entanglement. By evaluating success rates and resource requirements, we could thus determine the superior model of modular exponentiation to an MPS tensor network emulator.

Bibliography

[1] Peter W Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". In: *SIAM review* 41.2 (1999), pp. 303–332.

[2] Richard E Crandall and Carl Pomerance. *Prime numbers: a computational perspective*. Vol. 2. Springer, 2005.

[3] Vlatko Vedral, Adriano Barenco, and Artur Ekert. "Quantum networks for elementary arithmetic operations". In: *Physical Review A* 54.1 (1996).

[4] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[5] David Deutsch and Richard Jozsa. "Rapid solution of problems by quantum computation". In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439.1907 (1992), pp. 553–558.

[6] Lov K Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219.

[7] *Phase-estimation and factoring*. URL: `https://learning.quantum.ibm.com/course/fundamentals-of-quantum-algorithms/phase-estimation-and-factoring#introduction`.

[8] C.T. Long. *Elementary Introduction to Number Theory*. D. C. Heath, 1972. ISBN: 9780669627039. URL: `https://books.google.it/books?id=MthUAAAAYAAJ`.

[9] Leonhard Euler, Artur Diener, and Alexander Aycock. "Theoremata arithmetica nova methodo demonstrata". In: *arXiv preprint arXiv:1203.1993* (2012). Theorema 11.

[10] Victor Shoup. *A computational introduction to number theory and algebra*. Cambridge university press, 2009, p. 15.

[11] *Quantum Computing (CST Part II) - Lecture 10*. 2019. URL: `https://www.cl.cam.ac.uk/teaching/1920/QuantComp/Quantum_Computing_Lecture_10.pdf`.

[12] Richard Cleve et al. "Quantum algorithms revisited". In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454.1969 (1998), pp. 339–354.

[13] Marco Ballarin. *Introduzione all'emulatore di calcolatore quantistico HPC "Quantum Matcha TEA"*. Lesson 2. 2023. URL: `https://qcsc.dfa.unipd.it/introductory-lessons-to-quantum-computing/`.

[14] Stephane Beauregard. "Circuit for Shor's algorithm using 2n+ 3 qubits". In: *arXiv preprint quant-ph/0205095* (2002).

Acknowledgments

To begin with, I want to thank my supervisor Prof. Simone Montangero and my co-supervisor Dr. Ilaria Siloi, for leaving me substantial autonomy and flexibility to develop my work, allowing me to learn and grow significantly.
Secondly, a special thanks goes to my two tutors: Marco Ballarin, main developer of Quantum Matcha Tea, that introduced me to the quantum emulator and followed the progression of this thesis; and Lisa Zangrando, researcher at INFN responsible for the INFN cluster, that introduced me to the powerful virtual machines and gave me full technical assistance whenever I needed it.
Last but not least, I want to direct my gratitude to my parents and my sister, that have always supported me unconditionally. And to my grandmother, who generously provided me with free housing in Padua to facilitate my university experience.