UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Corso di Laurea in Fisica

Tesi di Laurea

Quantum computing programming languages

Relatore

Prof. Simone Montangero

Laureanda

Alice Pagano

Anno Accademico 2018/2019

Abstract

In this Thesis we analyze the quantum software developed by IBM and Google, respectively Qiskit and Cirq. It is reviewed how to program a simple quantum algorithm on both software and they are compared. In order to test their performance, we implement a circuit to verify particular N-qubit entangled states, called GHZ states, in both software. In fact, one of the main goal of quantum computation, and of quantum science in general, is the creation of a highly entangled state of many particles, because entangled states are the cornerstone of quantum speedups. We quantify the goodness of the state created through fidelity measurements. These provide a fundamental criterion for the comparison of two quantum states. We test the quantum circuit on the cloud service made available by IBM. In Cirq, no cloud service is yet available, therefore we test that circuit adding quantum noise channels, in order to reproduce and study noise effects in a model of real hardware.

Contents

Introduction

What is quantum computing? All computing systems rely on the fundamental ability to store and manipulate information. Current classical computers manipulate individual bits, which store information as binary 0 and 1 states. On the contrary, quantum computers leverage quantum mechanical phenomena, such as superposition and entanglement, to manipulate information. To do this, they rely on quantum bits, or qubits. Therefore, a quantum software allows to manipulate such qubits, translating a few lines of code in real quantum physical phenomena on the quantum hardware.

Research in quantum science is currently exploring a new exciting challenge: the *entanglement frontier* [4]. Quantum entanglement is the essential feature that makes information processing by quantum systems very different from the processing done by an ordinary digital computer. Currently, we are acquiring and improving the tools to build and precisely control highly entangled quantum states of many particles, that can not be simulated even with the best digital computers available or characterized using existing theoretical tools.

Quantum computers could spur new breakthroughs in science: untangling the complexity of molecular and chemical interactions leading to the discovery of new medicines and materials, finding the best solutions for ultra-efficient logistics and global supply chains, making facets of artificial intelligence such as machine learning much more powerful when data sets are very large and much more.

Today quantum computers are small, noisy, and not nearly as powerful as current classical computers, but Noisy Intermediate-Scale Quantum (NISQ) computers, with a number of qubits ranging from 50 to few hundreds, will be available soon. New tools for exploring the physics of many entangled particles will be provided making captivating applications a solid reality.

While on the one hand hardware development is a critical aspect of quantum computation, on the other hand part of nowadays researches focus on quantum software development that will allow anyone to use quantum computers. Multinational technology companies, as IBM and Google, are investing a great amount of resources in the development of a more user-friendly quantum software, making quantum computation widely approachable. Thanks to its versatility, Python programming language is often used, indeed both IBM and Google's quantum software framework, respectively Qiskit and Cirq, are based upon it.

In this Thesis we approach Qiskit and Cirq software. We illustrate their main features, in order to explain how a simple quantum circuit can be implemented and executed on both software. Then, we implement in each software a circuit for creating Multipartite entanglement in Greenberger-Horne-Zeilinger (GHZ) states. They are particular entangled states and are important because the ability to generate entangled states is indicative of high fidelity gate operations. In Qiskit, the circuit is tested in the real hardware provided by IBM. Google does not provide any real device, therefore, in Cirq, we test the circuit adding quantum noise channels in order to study noise effects in a model of real hardware.

In the first chapter, we provide a brief introduction to the fundamental concepts of quantum computation and information. First, we define the qubit and we illustrate its representation in state vectors formalism. Then, we introduce the density matrix representation of a state. We define what entanglement is and how it could be quantified. Afterwards, we illustrate the structure of a generic quantum circuit and we list the main operations that can act on the qubit. Finally, we introduce the quantum noise channels that are useful tools to reproduce the noise acting on an open system.

In the second chapter, firstly we briefly explain how a quantum hardware is built. Then we introduce the Qiskit and Cirq library. Both software are analyzed and their main features are described; it is explained how a simple quantum algorithm could be programmed using them.

In the third chapter, we implement a circuit for creating a N-qubit GHZ state, both in Qiskit and Cirq software. We follow the protocol described in [3] for calculating the fidelity of these states. In fact, we tried to verify the GHZ states in different devices and various conditions. In Qiskit, the circuit is executed both in simulators and in real devices as the IBM Q Yorktown and the IBM Q Melbourne. In Cirq, we execute the circuit only in simulators, because the Google hardware are not publicly available. We add quantum noise channels to the circuit to study how the state changes as a function of the parameters of the channels. Afterwards, we measure the density matrix of a 5-qubit GHZ state. Finally, we consider a Bell state, an entangled state of 2 qubits, and we quantify the entanglement through concurrence measurements.

Chapter 1

Fundamental concepts of quantum computation and information

This chapter provides a brief introduction to quantum computation and quantum information [1]. Firstly, quantum bits, or qubits, are introduced and their properties are illustrated in the state vector formalism. We also define the measurement operation, through which we infer information from a quantum state but destroy its coherence. Then, we introduce the density matrix representation of the states. It is a useful tool for describing the interaction of an isolated system with an external environment. We illustrate the concept of entanglement, a genuine quantum phenomenon, and the evolution of a quantum system through the density matrix formalism. Then, we introduce the quantum circuits and their building blocks: the quantum gates. Finally, we introduce quantum noise and show how it can be represented through quantum channels.

1.1 Qubit

1.1.1 Single qubit

A qubit is a two-level quantum system that can be represented by a unit vector in a two-dimensional Hilbert space. A base of this space can be defined by the sets of orthonormal states $\{|0\rangle, |1\rangle\}$, namely:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{1.1}$$

Generally, a qubit can be written as a linear combination, or *superpositions*, of these states:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \qquad \text{with} \quad |\alpha|^2 + |\beta|^2 = 1 \quad \alpha, \beta \in \mathbb{C}. \tag{1.2}$$

It is useful to introduce a geometric representation to describe qubit states because many operations on single qubits are neatly described within this picture. Since any global phase is undetectable, i.e. $|\psi\rangle \equiv e^{i\gamma}\,|\psi\rangle$ $(\forall\gamma \in \mathbb{R})$, it is possible to rewrite the generic state as:

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle \qquad 0 \le \theta \le \pi \quad 0 \le \phi < 2\pi \tag{1.3}$$

The angles θ and ϕ define a point on the three-dimensional unit sphere, called the *Bloch sphere*, shown in Figure 1.1. The states $|\psi\rangle$ defined in Eq. (1.2) are in a one-to-one correspondence with the points of this surface in $\mathbb{R}^3$. For instance, the $\hat{z}$ unit vector represents the $|0\rangle$ state and $-\hat{z}$ the $|1\rangle$ state.

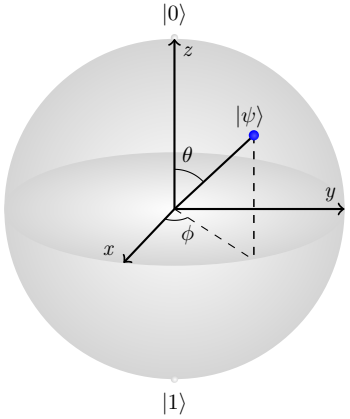


Figure 1.1: Bloch sphere representation of a qubit. It is a unitary sphere, where the numbers θ and ϕ identify a point. The $\hat{z}$ unit vector represents the $|0\rangle$ state and $-\hat{z}$ the $|1\rangle$ state.

1.1.2 Many qubit systems representation

Let us consider now a system of n distinguishable qubits. This system lives in a Hilbert space given by the tensor product of the Hilbert spaces of the single qubits $\mathcal{H}_i$:

$$\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \cdots \otimes \mathcal{H}_n \tag{1.4}$$

Computational basis states of such a system are built by tensor products of each qubit basis vectors, namely $\{|j\rangle_i\}_{i=1,\dots,n}$. A generic state $|\psi\rangle \in \mathcal{H}$ can be written as (1.5).

$$|\psi\rangle = \sum_{\vec{j}} c_{\vec{j}} \, |j\rangle_1 \, |j\rangle_2 \dots |j\rangle_n \qquad \text{with} \quad \sum_{\vec{j}} |c_{\vec{j}}|^2 = 1 \tag{1.5}$$

Therefore, a generic basis state of a many qubit system takes the form $|x_1 x_2 \dots x_n\rangle$ where $x_i \in \{0, 1\}$. For n qubits, the Hilbert space has dimensions $d = 2^n$. Let's consider an example of a two qubits system: a computational basis is thus in $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$, where the states are represented by the vectors in 1.6.

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \qquad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}. \tag{1.6}$$

A generic state $|\psi\rangle$ of this system is given by a superposition of these four states:

$$|\psi\rangle = \alpha_{00} \, |00\rangle + \alpha_{01} \, |01\rangle + \alpha_{10} \, |10\rangle + \alpha_{11} \, |11\rangle \qquad \text{with} \quad \sum_{x \in \{0,1\}^2} |\alpha_x|^2 = 1 \tag{1.7}$$

where the label x encodes the basis vector for the two qubits.

1.1.3 Measurements

In quantum mechanics, an observable $\hat{A}$ it is represented by an Hermitian operator and its average value for a generic state $|\psi\rangle$ is obtained by $\langle \hat{A} \rangle_\psi = \frac{\langle \psi | \hat{A} | \psi \rangle}{\|\psi\|^2}$. In practice, the value of expectation is achieved by repeating the measurement process several times over copies of the same initial state. The process of measurement consists on an interaction with an external physical system which plays the role of the measurement apparatus and the observable in this case is $\hat{\sigma_z}$, whose eigenstate are $|0\rangle$ and $|1\rangle$. Supposing a generic state of a single qubit $|\psi\rangle = \alpha \, |0\rangle + \beta \, |1\rangle$ with $\alpha, \beta \in \mathbb{C}$, a projective measurement will make the system collapses in the states $|0\rangle$ or $|1\rangle$ with probability $|\alpha|^2$ or $|\beta|^2$, respectively. The measurement either takes the value 0 if the qubit is measured in the state $|0\rangle$, and value 1 if the qubit is measured in the state $|1\rangle$. So, it is important to keep in mind that measurement is an irreversible operation, destroying quantum information and converting it into classical information.

A projective measurement is denoted in a quantum circuit using a 'meter' symbol, shown in Figure 1.2. This symbol will be clearer in the next sections.

Figure 1.2: Meter symbol in a quantum circuit.

1.2 Density matrix

In this section, we introduce an alternative formulation for describing quantum states: the *density matrix* representation. Firstly, we focus on single-qubit isolated system, whose state is completely known. The states relative to isolated systems are said to be *pure*; their density matrix carries all the information about the state. If the system interacts with an external environment, the state of the system is not pure anymore and it is said to be *mixed*. We characterize the density matrix for a mixed state, then we focus on density matrix definition and properties; we define the *reduced density matrix*, an indispensable tool in the analysis of composite quantum systems. Then we introduce the notion of *entanglement*. Finally, we define the evolution of quantum states.

1.2.1 From pure states to mixed states

A generic state $|\psi\rangle$ of a single qubit can be described in the density matrix representation:

$$\rho(\theta,\phi) = |\psi\rangle\langle\psi| = \begin{pmatrix} \cos^2\frac{\theta}{2} & \cos\frac{\theta}{2}\sin\frac{\theta}{2}e^{-i\phi} \\ \cos\frac{\theta}{2}\sin\frac{\theta}{2}e^{-i\phi} & \sin^2\frac{\theta}{2} \end{pmatrix}. \tag{1.8}$$

This state is completely characterized, so it is called *pure*. In this case, this alternative formulation is mathematically equivalent to the state vector approach used before.

It is possible to rewrite this matrix in the basis formed by identity matrix and the Pauli matrices using the spherical coordinates[1]:

$$\rho = \frac{1}{2}\begin{pmatrix} 1+z & x-iy \\ x+iy & 1-z \end{pmatrix} = \frac{1}{2}(\mathbb{1} + x\hat{\sigma_x} + y\hat{\sigma_y} + z\hat{\sigma_z}) \tag{1.9}$$

Now, let us suppose that a system initially in a pure state interacts with an external environment, where the environment is meant to be something outside the system[2]. After the interaction, part of the information relative to the state is lost; we can only infer the probabilities relative to some measurements. The new state is said to be *mixed*.

1.2.2 Density matrix formalism and properties

Mixed states are described by the density matrix formalism. Consider a quantum system among a number of states $|\psi_i\rangle$, with respective probabilities p_i. The *density matrix* of the system is defined by the equation:

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i| \tag{1.10}$$

The properties of the density matrix ρ are:

1. ρ is a hermitian operator. In matrix elements representation it means $\rho_{ij} = \rho^*_{ji}$.
2. $\mathrm{Tr}\,\rho = 1$.
3. ρ is a positive operator, i.e. $\forall\,|\psi\rangle$ is $\langle\psi|\,\rho\,|\psi\rangle \geq 0$.

[1] The spherical coordinates are: $\{x = \sin\theta\cos\phi, y = \sin\theta\sin\phi, z = \cos\theta\}$

[2] We will do a deeper analysis of system-environment interaction in the section 1.4, where we define the quantum channels.

If it exists $|\psi\rangle$ such that it is possible to write $\rho = |\psi\rangle\langle\psi|$, a quantum system is in a *pure state*, otherwise ρ describes a *mixed state*; it is said to be a mixture of the different pure states in the ensamble for ρ. Another way for distinguishing between the two types of states is analyzing the $\mathrm{Tr}(\rho^2)$ [3] , also called *purity* of the state. If $\mathrm{Tr}(\rho^2) = 1$ the system is in a *pure state*, if $\mathrm{Tr}(\rho^2) < 1$ it is in a *mixed state*.

Consider a single qubit system, an arbitrary density matrix for a mixed state may be written in the Bloch sphere representation [1] as:

$$\rho = \frac{\mathbb{1} + \vec{r} \cdot \vec{\sigma}}{2} \tag{1.11}$$

where $\vec{r}$ is a real three-dimensional vector such that $\|\vec{r}\| \leq 1$, called the *Bloch vector* for the state ρ. The state ρ is *pure* if and only if $\|\vec{r}\| = 1$. Therefore, the possible density matrices for a single qubit system, that are generic vectors on the Bloch sphere, correspond to pure states ($\|\vec{r}\| = 1$), while the vectors inside the sphere correspond to mixed states.

Consider the matrix ρ, the diagonal terms are called *terms of population*, while the terms outside the diagonal are the *terms of coherence*. The last one describes correlations between the single states.

1.2.3 Reduced density matrix

Let us consider two physical systems A and B, whose state is described by the density matrix ρ^{AB}. Let us suppose the reduced density matrix for system A is defined by:

$$\rho^A = \underset{B}{\mathrm{Tr}}(\rho^{AB}) \tag{1.12}$$

where $\underset{B}{\mathrm{Tr}}$ is called *partial trace* over the system B.

If $\{|a_k\rangle\}_{i=1,\dots,m}$ and $\{|b_l\rangle\}_{j=1,\dots,p}$ are orthonormal bases for the systems A and B respectively, the *partial trace* of A over the system B is obtained by summing only over the basis vector of B, namely:

$$(\rho^A)_{ij} = (\underset{B}{\mathrm{Tr}}\rho^{AB})_{ij} = \sum_{l}^{p} \langle b_l| \rho^{AB} |b_l\rangle = \sum_{l}^{p} \rho_{ib_l}^{jb_l} |i\rangle\langle j| \tag{1.13}$$

The reduced density matrix operation do not preserve the *purity* of the state. It means that if ρ^{AB} represents a pure state, then ρ^A or ρ^B could be mixed.

1.2.4 Entanglement

The entanglement is a genuine quantum phenomenon that entails an intrinsic correlation between the constituents of a quantum system. This concept could be understood considering as example a 'quantum book' [4], a book whose pages are not to be read individually, because the information in that quantum book is not imprinted on the individual pages; it is encoded almost entirely in how the pages are correlated with one another .

An entangled state of a quantum system is characterized by the fact that its state cannot be factorized as a tensor product of states of its local constituents. An entangled state can be written as:

$$(\rho^{AB}) = \sum_i p_i(\rho_i^A \otimes \rho_i^B). \tag{1.14}$$

Now, let's consider few examples:

- for a two-qubit system a type of entangled states are the *Bell states*, represented as

$$\left|\phi^+\right\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \qquad \left|\phi^-\right\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}} \qquad \left|\psi^+\right\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}} \qquad \left|\psi^-\right\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \tag{1.15}$$

[3] The trace operation is defined as $\mathrm{Tr}(B) = \sum_{j=1}^{\dim \mathcal{H}} \langle\chi_j| B |\chi_j\rangle$, where $\{|\chi_k\rangle\}_{k=1,\dots,\dim \mathcal{H}}$ is an ortonormal basis of the Hilbert space.

- for a system of $N > 2$ qubits, an important entangled state is the Greenberger–Horne–Zeilinger state (*GHZ state*), that has the form

$$|\mathrm{GHZ}\rangle = \frac{|0\rangle^{\otimes N} + |1\rangle^{\otimes N}}{\sqrt{2}}. \tag{1.16}$$

  The importance of these states is that the GHZ states are maximally entangled quantum states. They can be used for estimating the fidelity of the gate operations as we will see in Chapter 3.

How to quantify entanglement

In order to quantify the entanglement, we introduce the Von Neumann *entropy* associated to a density matrix ρ:

$$S(\rho) \equiv -\operatorname{Tr}(\rho \log_2 \rho) \tag{1.17}$$

If ρ_{AB} is a pure state which describes the state of the systems A and B, a good measurement of entanglement is the entropy of Von Neumann of the reduced density matrix of one of the two subsystems, for instance:

$$E(\rho) = S(\rho_A). \tag{1.18}$$

If instead the state ρ is mixed we have to modify the previous definition: let us consider the mixed state $\rho_{AB} = \sum_{j=1}^{k} p_j |\psi_j\rangle \langle\psi_j|$ where $\{|\psi_j\rangle\}$ are basis of $\mathcal{H}_{\mathrm{AB}}$. A measurement of entanglement is

$$E(\rho) \equiv \min_{|\psi_j\rangle\langle\psi_j|} \sum_{i=1}^{k} p_i S(\rho_A^i) \tag{1.19}$$

where the count is done for all families $\{|\psi_j\rangle \langle\psi_j|\}$ of states of ρ_{AB}.

If we consider a two-qubit system, an entanglement measurement can by done using the *concurrence* method [12]. The **concurrence** is defined for a state ρ of two qubits as:

$$\mathcal{C}(\rho) \equiv \max(0, \lambda_1 - \lambda_2 - \lambda_3 - \lambda_4) \tag{1.20}$$

in which $\lambda_1, ..., \lambda_4$ are the eigenvalues, in decreasing order, of the Hermitian matrix

$$R = \rho\tilde{\rho} \tag{1.21}$$

with

$$\tilde{\rho} = (\sigma_y \otimes \sigma_y)\rho(\sigma_y \otimes \sigma_y). \tag{1.22}$$

1.2.5 Density matrix evolution

The evolution of the density matrix ρ is described by the *Liouville-von Neumann* equation:

$$i\hbar \frac{\mathrm{d}\rho(t)}{\mathrm{d}t} = [H, \rho(t)]. \tag{1.23}$$

The evolution of a quantum system, that can be either closed or open, can be described also by the quantum operation formalism. Suppose that the system is in the state ρ, the quantum state transforms as

$$\rho' = \mathcal{E}(\rho) \tag{1.24}$$

where the map $\mathcal{E}$ is known as *quantum operation* [1].

Closed quantum systems

Suppose we have a closed quantum system whose evolution is described by the unitary operator $\hat{U}$. The evolution of the density operator is described by the equation:

$$\rho(t) = \mathcal{E}(\rho) \equiv \hat{U}\rho\hat{U}^\dagger \tag{1.25}$$

The unitary evolution preserves the *purity* of the state. It means that a pure state can unitary evolve only to pure states and at the same way a mixed state evolves only to mixed states (only if the system remains closed). The unitary evolution can be represented as a circuit model, as shown in Figure 1.3.

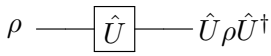


Figure 1.3: Circuit model of closed quantum system.

Open quantum systems

Consider a system, called the *principal system*, interacting with an *environment*. The system is said to be open due to its interaction with the environment. The dynamics of such a quantum system $\rho(t=0) = \rho_S \otimes \rho_E$ can be described as:

$$\rho(t) = \mathcal{E}(\rho_S) \equiv \operatorname*{Tr}_E[\hat{U}(\rho_S \otimes \rho_E)\hat{U}^\dagger] \tag{1.26}$$

Now, we describe the evolution of an open system. Consider a system defined by the state $\rho = \rho_S \otimes \rho_E$, that is the tensor product of the *principal system* state and the *environment* state. The dynamics of such a quantum system could be described as arising from an interaction between these two systems, which together form the closed quantum system ρ. In other words, suppose to have a system ρ_S, which is sent into a box which is coupled to the environment ρ_E. This coupling is described by the action of the $\hat{U}$ operation. After the gate application, the system no longer interacts with the environment and we can perform the partial trace over the environment to obtain the reduced state of ρ_S, namely:
The dynamics of the open quantum system is modeled in Figure 1.4.

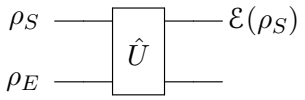


Figure 1.4: Circuit model of open quantum systems. An open quantum system consists of two parts: ρ_S, the principal system, and ρ_E the environment system.

Moreover, we can analyze the open quantum systems evolution representing quantum operations in the *Kraus representation* [2]. Let $\{|e_k\rangle\}$ be an orthonormal basis for the state space of the environment and $|e_0\rangle$ its initial state; it is possible to write:

$$\mathcal{E}(\rho_S) = \sum_k E_k \rho_S E_k^\dagger \quad \text{with} \quad \sum_k E_k^\dagger E_k = 1 \tag{1.27}$$

where $E_k \equiv \langle e_k | U | e_0 \rangle$ is an operator on the state space of the principal system. The operators $\{E_k\}$ are known as *Kraus operators* and they are not unique.

1.3 Quantum computation

The information carried by quantum states can be manipulated by using quantum gates. They are the building blocks of quantum circuits, as well as classical logic gates for conventional digital circuits. A quantum circuit is constituted by a discrete sets of components which describe a computational procedure. We suppose to consider a system of n qubits, let's describe the main steps to realize a quantum circuit:

1. First, each qubit is initialized in a quantum state. Generally, the initial state of a qubit is chosen as $|0\rangle$, the ground state.
2. Lines are drawn in the circuit to represent *wires* in the quantum circuit and they represent also the flux of time from left to right.
3. Quantum gates $\hat{U}$ can be applied to one or more qubits.
4. Finally, the quantum states could be measured by measurement operation.

A general example of a n qubits system is shown in Figure 1.5; it is important to underline that a circuit has to be read from left-to-right.

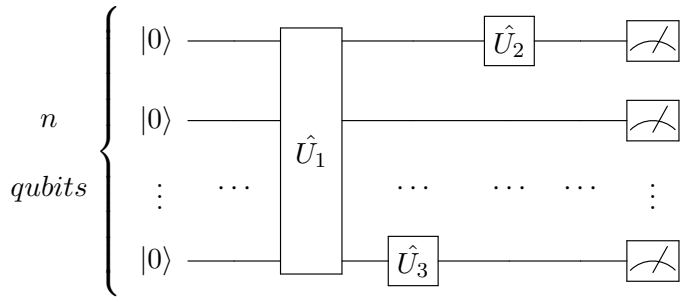


Figure 1.5: An example of a generic quantum circuit of n qubits. The qubits are initialized in the state $|0\rangle$. A n qubits gate is applied on all qubits and other gates are applied randomly. Finally all the qubits are measured.

Quantum gates are represented by unitary matrices generically indicated by $\hat{U}$. As we will see later, this assumption is valid when the system is isolated. Let us stress that unitarity property makes a quantum gate reversible: input and output for a given number of qubits gate must be equal; a gate which acts on n qubits is represented by a $2^n \times 2^n$ unitary matrix, namely the dimension of the Hilbert space.

The action of the quantum gate is given by the product of the matrix representing the gate with the vector which represents the quantum state:

$$\left|\psi'\right\rangle = \hat{U}\left|\psi\right\rangle . \tag{1.28}$$

We illustrate now some quantum gates: single and two-qubits gates. Gates involving more qubits can be decomposed in terms of these ones. We show in particular the gates u_1, u_2, u_3, which are the native gates implemented on the IBM devices, as we will discuss later in Chapter 2. For each gate we show the symbol used for representing it in the circuits.

1.3.1 Single qubit gates

A gate which acts on a single qubit is represented by a 2×2 unitary matrix $\hat{U}$. The most general form of a single qubit unitary matrix [11] is:

$$\hat{U}(\theta, \phi, \lambda) = \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i\lambda+i\phi}\cos(\theta/2) \end{pmatrix}. \tag{1.29}$$

where $0 \leq \theta \leq \pi$, $0 \leq \phi < 2\pi$ and $0 \leq \lambda < 2\pi$.
Now, let's illustrate the most common gates.

Identity gate

The Identity gate is a single qubit operation that leaves a generic state $|\psi\rangle$ unchanged:

$$\mathbb{I} = \hat{U}(0,0,0) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \text{—}\boxed{\mathbb{I}}\text{—} \tag{1.30}$$

Pauli gates

- **X: Bit-flip gate**
  The Pauli-X gate is a single-qubit π rotation around the x-axis:

$$X = \hat{U}(\pi, 0, \pi) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \text{—}\boxed{X}\text{—} \tag{1.31}$$

  It maps $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$. It is possible to visualize how the X-gate acts on an input state $|\psi\rangle = |0\rangle$, in the Bloch sphere representation in Figure 1.6.

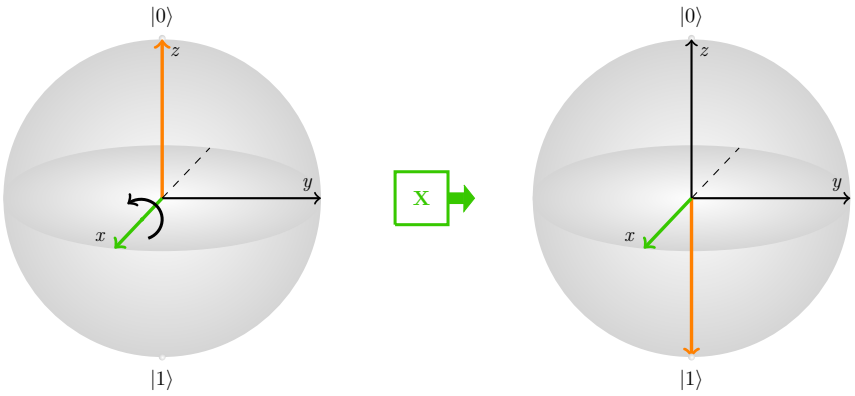


Figure 1.6: Visualization of the bit-flip gate on the Bloch sphere, acting on the input state $|0\rangle$. The gate maps the input state $|0\rangle$ to the $|1\rangle$ state and viceversa.

- **Y: Bit-and phase-flip gate**
  The Pauli-Y gate is a single-qubit π rotation around the y-axis:

$$Y = \hat{U}(\pi, \pi/2, \pi/2) = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \text{—}\boxed{Y}\text{—} \tag{1.32}$$

  It maps $|0\rangle$ to $i\,|1\rangle$ and $|1\rangle$ to $-i\,|0\rangle$.
- **Z: Phase-flip gate**
  The Pauli-Z gate is a single-qubit π rotation around the z-axis:

$$Z = \hat{U}(0, 0, \pi) = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad \text{—}\boxed{Z}\text{—} \tag{1.33}$$

  It leaves the basis state $|0\rangle$ unchanged and maps $|1\rangle$ to $-\,|1\rangle$.

Hadamard gate

The Hadamard gate is a single-qubit operation that maps the basis state $|0\rangle$ to $\frac{(|0\rangle+|1\rangle)}{\sqrt{2}}$ and $|1\rangle$ to $\frac{(|0\rangle-|1\rangle)}{\sqrt{2}}$, thus creating an equal superposition of the two:

$$H = \hat{U}(\pi/2, 0, \pi) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad \text{—}\boxed{H}\text{—} \tag{1.34}$$

As the gates X,Y,Z, also the Hadamard one implements a rotation on the Bloch sphere. The difference is that it rotates around an axis located halfway between x and z in the plane $\perp$ to y, as we show in Figure 1.7. Here, the input state $|0\rangle$ is rotated by π angle around the $\frac{\hat{x}+\hat{z}}{2}$ axis into the $\frac{(|0\rangle+|1\rangle)}{\sqrt{2}}$ state.

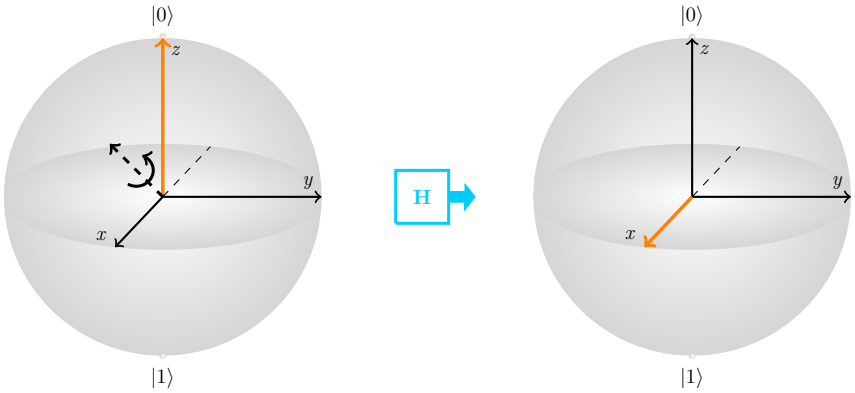


Figure 1.7: Visualization of the Hadamard gate on the Bloch sphere, starting from the input state $|0\rangle$. The state $|0\rangle$ is rotated around the $\frac{\hat{x}+\hat{z}}{2}$ axis into the $\frac{(|0\rangle+|1\rangle)}{\sqrt{2}}$ state.

Phase-shift gate

The Phase-shift is a single-qubit operation that leaves the basis state $|0\rangle$ unchanged and adds a relative phase on the basis state $|1\rangle$ mapping it to $e^{i\phi}|1\rangle$:

$$R_z(\delta) = \hat{U}(0,0,\delta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\delta} \end{pmatrix} \qquad \text{—}\boxed{R_z(\delta)}\text{—} \tag{1.35}$$

Native gates implemented in the IBM devices

In the following chapters, we will work on the IBM quantum computers whose software implements gates in a certain form. Therefore, we illustrate the native single-qubit gates implemented in the IBM device:

$$u_3(\theta,\phi,\lambda) \equiv \hat{U}(\theta,\phi,\lambda) = \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i\lambda+i\phi}\cos(\theta/2) \end{pmatrix} \tag{1.36}$$

$$u2(\phi,\lambda) \equiv \hat{U}(\pi/2,\phi,\lambda) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\phi+\lambda)} \end{pmatrix} \tag{1.37}$$

$$u_1(\delta) \equiv \hat{U}(0,0,\delta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\delta} \end{pmatrix}. \tag{1.38}$$

1.3.2 Two-qubit gates

In this section we introduce some fundamental gates which cannot be decomposed into single-qubit gates: SWAP and controlled.

SWAP gate

The SWAP gate literally swaps the state of two qubits. For example, it transforms the basis vectors as $|00\rangle \to |00\rangle$, $|01\rangle \to |10\rangle$, $|10\rangle \to |01\rangle$, $|11\rangle \to |11\rangle$. The SWAP gate matrix representation and the relative circuit symbol are, respectively:

$$\text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \text{—}\times\text{—}\\ \text{—}|\text{—}\\ \text{—}\times\text{—} \tag{1.39}$$

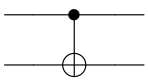
Controlled gates

A common multi-qubit gate involves the application of a gate to one qubit, conditioned on the state of another qubit. In the controlled two-qubit gate $C_{\hat{U}}$, a unitary transformation $\hat{U}$ is applied to one of the two qubits, known as *target* qubit, if the other, the *control* qubit, is for example in the state $|1\rangle$. The most general form [11] is the following [4]:

$$C_{\hat{U}}(\theta,\phi,\lambda) \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{-i(\phi+\lambda)/2}\cos(\theta/2) & -e^{-i(\phi-\lambda)/2}\sin(\theta/2) \\ 0 & 0 & e^{i(\phi-\lambda)/2}\sin(\theta/2) & e^{i(\phi+\lambda)/2}\cos(\theta/2) \end{pmatrix}. \tag{1.40}$$

For instance, we consider the **Controlled-X** gate that flips the target qubit when the control qubit is in the state $|1\rangle$. For example, let us suppose that the first qubit is the control one and the second is the target, therefore the CX gate maps $|00\rangle$ to $|00\rangle$ and $|10\rangle$ to $|11\rangle$. The action of the gate can be summarized as $|A, B\rangle \to |A, B \oplus A\rangle$, where $\oplus$ is addition modulo two. The circuit representation for the CX is the following:

$$C_X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad \tag{1.41}$$



where the fill dot represents the control qubit (upper line), while the $\oplus$ represents the transformation applied to the target qubit (bottom line).

1.3.3 Example of applications of quantum circuits

What is a quantum computer good for? Practically speaking, many interesting problems are impossible to solve on a classical computer. This is not because they are in principle insoluble, but because of the huge resources required. The spectacular promise of quantum computers is to enable new algorithms to solve problems which need resources unavailable on a classical computer. There are two main classes of quantum algorithms. The first class is based on the so called *quantum Fourier trasnsform* [1]: it includes remarkable algorithms for solving the phase estimation and factory problems, providing a striking *exponential* speedup over the best known classical algorithms. The second class of algorithms is based upon Grover's algorithm for performing *quantum searching* [1]. They provide a *quadratic* speedup over the best possible classical algorithms.

1.4 Quantum noise: quantum channels

Quantum noise is a consequence of non-unitary effects involved in the dynamics of a system that interacts with an external environment during its dynamics. The *Kraus representation* previously introduced must be used to model the effects of noise on quantum systems. In fact, quantum effects can be modeled by introducing a set of quantum channels that act as the evolution $\rho \to \rho' = \sum_k E_k \rho E_k^\dagger$. Therefore each quantum channel is characterized by a collection of *Kraus operators*.

Single qubit quantum channels can be illustrated with a geometric representation, namely an affine transformation of the coordinates of the ρ state [1]. Let us consider a mixed state ρ and its representation on the Bloch sphere, an arbitrary trace-preserving quantum operation is equivalent to a map of the form:

$$\vec{r} \xrightarrow{\mathcal{E}} \vec{r'} = M\vec{r'} + \vec{c}. \tag{1.42}$$

This affine transformation transforms the Bloch sphere in a translated ellipsoid with a minor or equal volume.

The related *Kraus operator* has the form:

[4] We consider the basis vectors of the two-qubit system ordered as $|00\rangle$,$|01\rangle$,$|10\rangle$,$|11\rangle$.

$$E_i = \alpha_i \mathbb{1} + \sum_{k=1}^{3} a_{ik}\sigma_k \tag{1.43}$$

where the parameters are

$$M_{jk} = \sum_{l=1}^{3}\Big[a_{lj}a_{lk}^* + a_{lj}^* a_{lk} + (|\alpha_l|^2 - \sum_{p=1}^{3} a_{lp}a_{lp}^*)\delta_{jk} + i\sum_{p=1}^{3}\varepsilon_{jkp}(\alpha_l a_{lp}^* - \alpha_l^* a_{lp})\Big] \tag{1.44}$$

$$c_k = 2i\sum_l \sum_{jp} \varepsilon_{jpk} a_{lj} a_{lp}^*. \tag{1.45}$$

In the following subsection we illustrate the main quantum channels, showing for each the Kraus operators.

1.4.1 Bit-flip and Phase-flip channels

The action of Bit-flip, Phase-flip or, Bit-phase flip on a quantum state ρ is defined as the process $\rho \to \rho' = E_0\rho E_0^\dagger + E_1\rho E_1^\dagger$.

- The **Bit-flip** channel flips the state of a qubit from $|0\rangle$ to $|1\rangle$ (and vice versa) with probability $1-p$. It has Kraus operators:

$$E_0 = \sqrt{p}\,\mathbb{1} = \sqrt{p}\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} \qquad E_1 = \sqrt{1-p}\,X = \sqrt{1-p}\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix} \tag{1.46}$$

  The effect of the Bit-flip channel is illustrated in Figure 1.8a on the Bloch Sphere. The state on the $\hat{x}$ axis is left alone, while the $\hat{y}-\hat{z}$ plane is uniformly contracted by a factor of $1-2p$.
- The **Phase-flip** has Kraus operators:

$$E_0 = \sqrt{p}\,\mathbb{1} = \sqrt{p}\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} \qquad E_1 = \sqrt{1-p}\,Z = \sqrt{1-p}\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix} \tag{1.47}$$

  The effect of the Phase-flip channel is illustrated in Figure 1.8b on the Bloch Sphere. The state on the $\hat{z}$ axis is left alone, while the $\hat{x}-\hat{y}$ plane is uniformly contracted by a factor of $1-2p$.
- The **Bit-phase flip** has Kraus operators:

$$E_0 = \sqrt{p}\,\mathbb{1} = \sqrt{p}\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} \qquad E_1 = \sqrt{1-p}\,Y = \sqrt{1-p}\begin{pmatrix}0 & -i\\ i & 0\end{pmatrix} \tag{1.48}$$

  The effect of the Bit-phase flip channel is illustrated in Figure 1.8c on the Bloch Sphere. The state on the $\hat{y}$ axis is left alone, while the $\hat{x}-\hat{z}$ plane is uniformly contracted by a factor of $1-2p$.

1.4.2 Depolarizing channel

The **Depolarizing** channel *depolarizes* a single qubit with probability p, namely it is replaced by the completely mixed state, $\mathbb{1}/2$. With probability $1-p$ the qubit is left untouched. The Kraus operators are:

$$E_k = \left\{\sqrt{1-p}\,\mathbb{1}, \sqrt{\frac{p}{3}}\,\sigma_i\right\} \tag{1.49}$$

Thus the transformation is $\rho \to \rho' = \frac{1}{3}p(\sigma_x\rho\sigma_x^\dagger + \sigma_y\rho\sigma_y^\dagger + \sigma_z\rho\sigma_z^\dagger) + (1-p)\rho$. The effect of the Depolarizing channel, illustrated in Figure 1.8d, is to contract uniformly as a function of ρ the entire sphere. Thus, the affine transformation is:

$$\vec{r} \to \Big(1 - \frac{4}{3}p\Big)\vec{r}. \tag{1.50}$$

It is clear that applicating this channel many times leads to $\vec{r} = 0$ that corresponds to the state $\rho = \mathbb{1}/2$.

1.4.3 Amplitude Damping

The **Amplitude Damping** channel is important for the description of *energy dissipation*. Suppose we have a qubit and $|0\rangle$ is the ground state. Given an initial superposition of $|0\rangle$ and $|1\rangle$, due to the interaction with an external environment the system might disperse energy in the environment. The state would therefore change, increasing the population of the state $|0\rangle$ and reducing that of $|1\rangle$. Supposing ρ is the density matrix of the system, this process is described by the transition $\rho \to \rho' = E_0 \rho E_0^\dagger + E_1 \rho E_1^\dagger$, where the Kraus operators are defined as:

$$E_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-p} \end{pmatrix} \qquad E_1 = \begin{pmatrix} 0 & \sqrt{p} \\ 0 & 0 \end{pmatrix} \tag{1.51}$$

It acts on a generic density matrix ρ as in the following:

$$\rho = \begin{pmatrix} \rho_{00} & \rho_{01} \\ \rho_{10} & \rho_{11} \end{pmatrix} \quad \Rightarrow \quad \rho' = \begin{pmatrix} \rho_{00} + p\rho_{11} & \sqrt{1-p}\rho_{01} \\ \sqrt{1-p}\rho_{10} & (1-p)\rho_{11} \end{pmatrix}. \tag{1.52}$$

For example $\rho = |1\rangle\langle 1|$ is mapped into $\rho' = p\,|0\rangle\langle 0| + (1-p)\,|1\rangle\langle 1|$. Note that a pure state has become a mixed one.

Now, suppose to apply this channel n times consecutively, the element ρ_{11} of the density matrix ρ decades exponentially:

$$\rho_{11}^{(n)} = (1-p)^n \rho_{11}^{(0)} = e^{n\log(1-p)} \rho_{11}^{(0)} \underset{n\to\infty}{\longrightarrow} 0 \tag{1.53}$$

It means that the state after many applications of the channel is $\rho^{(\infty)} = |0\rangle\langle 0|$. Now, we replace p with a time-varying function like $1 - e^{-t/T_1}$, where T_1 is called 'relaxation time'. The effect of Amplitude Damping for a single qubit system can be visualized on the Bloch sphere representation as a flow in which every point in the unit ball moves towards a fixed point at the north pole, corresponding to $|0\rangle$, as represented in Figure 1.8e.

1.4.4 Phase Damping

The **Phase Damping** channel describes the loss of quantum information without loss of energy. Suppose we have a generic initial state ρ:

$$\rho = \begin{pmatrix} p_0 & \alpha \\ \alpha^* & 1-p_0 \end{pmatrix}. \tag{1.54}$$

In the Phase Damping channel the interaction of the qubit with the environment is obtained by random rotation $R_z(\theta)$:

$$R_z(\theta) = \begin{pmatrix} \exp\left(-\frac{i\theta}{2}\right) & 0 \\ 0 & \exp\left(\frac{i\theta}{2}\right) \end{pmatrix} \tag{1.55}$$

called *phase kick*, that modifies the coherence terms α.

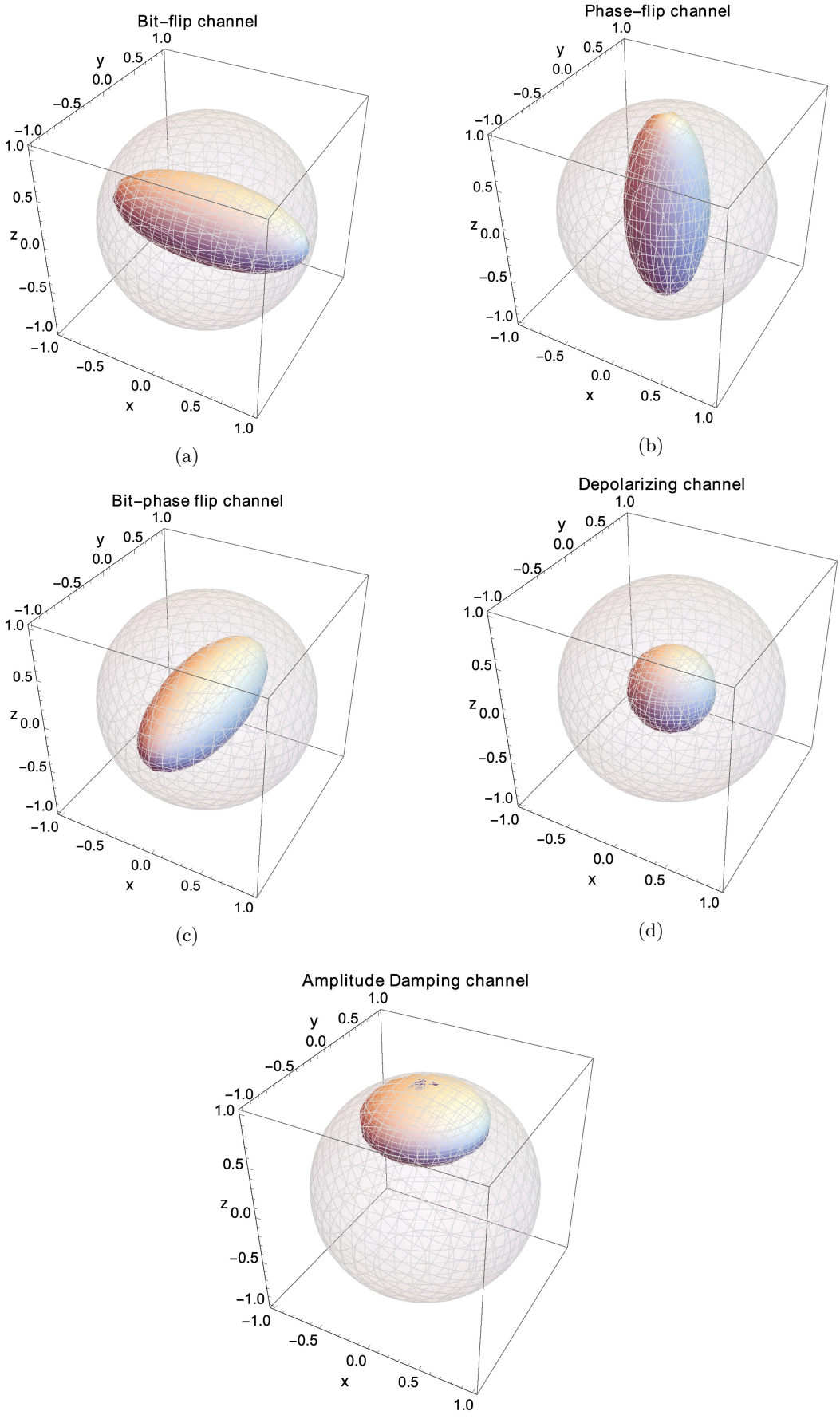
Suppose that θ is a random variable picked from a gaussian distribution with standard deviation λ. Therefore, the final state is obtained integrating over all the possible transformations, weighted by their probability of being realized, by:

$$\rho \to \rho' = \int_{-\infty}^{+\infty} \mathrm{d}\theta p(\theta) R_z(\theta) \rho R_z^\dagger(\theta). \tag{1.56}$$

The result of this integral returns

$$\rho' = \begin{pmatrix} p & \alpha\, e^{-\lambda} \\ \alpha^*\, e^{-\lambda} & 1-p \end{pmatrix}. \tag{1.57}$$

Phase damping is the same as the phase-flip channel [1], visualized in Figure 1.8b. It is often associated to a relaxation time T_2 , where $e^{-t/2T_2} = \sqrt{1-\lambda}$.



(a) (b) (c) (d) (e)

Figure 1.8: The effects of the noise channels on the Bloch sphere for $p = 0.7$. For each channel, the sphere in trasparency represents the set of all pure states and the deformed sphere represents the states after going through the channel.

Chapter 2

Quantum computing: from hardware to software

In this chapter, we briefly discuss how a quantum computer is realized. Today these computers are small, noisy, and not nearly as powerful as current classical computers, but Noisy Intermediate-Scale Quantum (NISQ) computers will be available in the next few years. Here "intermediate scale" refers to the size of quantum computers with a number of qubits ranging from 50 to a few hundred; that's beyond what can be simulated by brute force using the most powerful existing classical supercomputers [4].

In particular, we focus on the hardware and software of the quantum computers developed by the companies IBM and Google. We illustrate the 'superconducting qubits' technology, on which both of the IBM and Google hardware are based upon. It is important to underline that this chapter is only a brief introduction to the quantum hardware and that a deep analysis is beyond the aim of this Thesis. Then, we discuss about the software, needed as an interface between the quantum hardware apparatus and final users. In particular, the quantum software frameworks provided by IBM and Google are Qiskit and Cirq, respectively. This section gives tools to create a quantum circuit on each software, run them on a simulator or a real quantum device, when provided, and view the data results. Both software use Python programming language.

2.1 Quantum hardware: superconducting qubits

The implementation of a real quantum hardware must satisfy five conditions, called DiVincenzo's criteria [13]:

1. **Scalability**: a scalable physical system with well characterized qubits.
2. **Reset**: the ability to initialize the state of the qubits to a simple fiducial state, such as $|000\ldots0\rangle$.
3. **Long decoherence times**: long relevant decoherence times, much longer than the gate operation time.
4. **'Universal' gate**: a 'universal' set of quantum gates.
5. **Efficiently read-out**: a qubit-specific measurement capability.

One of the main challenges of implementing an efficient quantum hardware is to keep the system decoupled from external influences, except during the write, control and measurement phases [5] . One of the main approaches for implementing a quantum computer is based upon 'quantum integrated circuits', where qubits are constructed from collective electrodynamic modes of macroscopic electrical elements. The advantage is that qubits may be coupled together via non-dissipative linear electrical elements as capacitors, inductors and transmission lines. The difficult part is isolating these qubits from the external noise. The first requirement for an integrated circuit, in order to behave quantum mechanically, is the absence of dissipation for all metallic parts that constitute the circuit at the qubit operating temperature. Therefore, the low temperature superconductors are the best choice for this task and such a quantum integrated circuit implementation is called '*superconducting qubit*'. The temperature of the integrated circuit has to be also less than the energy associated to the transition between the ground state, $|0\rangle$, and the excited state $|1\rangle$. The temperature of the wires of the control and read-out gates connected to the chip has to be low too. Quantum signal processing is performed in

integrated circuit by non-linear dissipative elements that consist of superconducting tunnel junctions, also called Josephson junctions.

2.1.1 Josephson junctions

The Josephson junctions are used in quantum integrated circuits because they are the only electronic elements that are both non-dissipative (in a good approximation) and non-linear at arbitrarily low temperature. They are constituted by two layers of superconducting films separated by a thin insulating layer, as illustrated in Figure 2.1. The Josephson junctions are based on the tunnel effect: the insulating layer is thin enough to allow tunneling of discrete charges through the barrier [6]. In fact, at low temperatures, the superconducting films produce *Cooper's pairs* that can pass trough the potential wall, originating a charge flow.

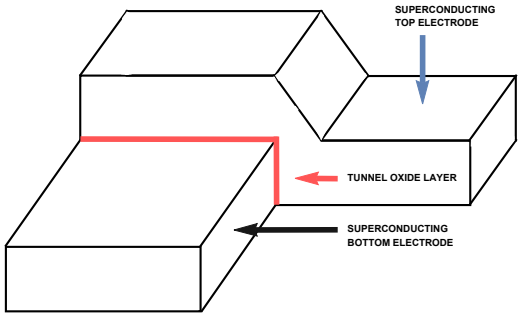


Figure 2.1: A Josephson junction realized with two superconducting thin films separated by an insulating layer that allow tunneling of discrete charges through the barrier.

The most simple single qubit implementation using Josephson junctions is the charge qubit. A 'charge qubit' is obtained by biasing the Josephson junction, that has an intrinsic capacity C_J, with a voltage source U_g in series with a capacitor C_G, as shown in Figure 2.2. The Cooper's pairs could not pass trough the capacitor shells without spending energy, therefore the superconducting layer exposed to the capacitor is called *island*. The junction can be described by two parameters: N_g, the number of Cooper's pairs in the island, and θ that is called 'superconductive phase'. The Hamiltonian of such a system is:

$$H = E_C(N - N_g)^2 - E_J \cos\theta \qquad [\theta, N] = i \tag{2.1}$$

where $E_C = \frac{(2e)^2}{(2(C_J+C_g))}$ is the charging energy of the island of the box (e represents the electron charge) and E_J is called *Josephson's energy*.

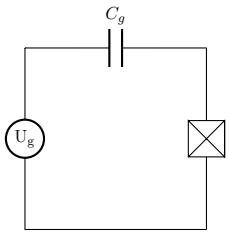


Figure 2.2: Charge qubit circuit scheme. The U_g represents a voltage source in series with a capacitor C_G, while the square box represents a Josephson junction.

The Hamiltonian could be wrote in the basis of the Cooper's pairs $|N\rangle$ as:

$$H = E_C \sum_N (N - N_g)^2 |N\rangle \langle N| - \frac{1}{2} E_J \sum_N |N+1\rangle \langle N| + |N\rangle \langle N+1| \tag{2.2}$$

where the second term is a coupling term, that relates the state $|N\rangle$ with the state $|N+1\rangle$.

2.2 Quantum software: Qiskit and Cirq

Today, a wide range of quantum computing software is available [7]. Most of the quantum computing companies, as IBM, grant the access to a real quantum computer thanks to the cloud platform and the interaction between users and quantum computers is mediated by a software platform with API (Application Programming Interface) access [8]. Other companies, such as Google, have no current capability to connect to a quantum computer and provide just the simulator as yet. Many practical considerations emerge when using a software interface to communicate with a real or simulated quantum computer. For example, we can ask how easy is to create, work with and manipulate quantum circuits or how easy is to parametrize algorithms for near-term quantum computing. Another good question can be asking quantum computer simulator can be used to test algorithms, how many qubits can be simulated and if the simulator is noiseless or noisy. If the platform gives access to real quantum devices, we can ask what are their main features, as the number of qubits or gate operations natively built-in. It's also important to understand how data results can be visualized. In this section we try to answer to these questions for the quantum software frameworks of IBM and Google, Qiskit and Cirq respectively. The first one provides the access to real quantum devices, the second one only makes simulators available.

2.2.1 Qiskit software

Qiskit is an open-source framework for working with the OpenQASM quantum language and quantum processors in the IBM Q experience [10]. It is organized in four distinct components:

- **Terra**: Qiskit Terra provides tools for composing quantum circuits at the level of quantum machine code and manages the execution of that code on quantum hardware. It also provides tools to allow quantum circuits to be optimized for a particular device.
- **Aer**: Qiskit Aer provides simulators for studying quantum computing algorithms and applications in devices affected by noise; it permits to build noise models for noisy simulations.
- **Ignis**: Qiskit Ignis includes tools for characterizing errors through method as tomography and for doing quantum error-correction.
- **Aqua**: Qiskit Aqua provides quantum algorithms that can be directly used to build applications for quantum computing.

Manipulating a quantum circuit

Generally, a quantum circuit on Qiskit terra can be built in three main steps:

1. Suppose n and m fixed integer. In general, a quantum circuit is composed by two registers: a quantum register of n qubits and a classical register of m bits.
2. Each qubit of the quantum register has an initial state of $|0\rangle$. Quantum gates can be applied on the n qubits available for changing the quantum state.
3. The quantum circuit created can be visualized choosing the desired output.

All the possible unitary gates can be implemented by defining a sequence of simpler gates implemented in the hardware. Then, there are options for doing measurement or reset of the state to $|0\rangle$, in the middle of the computation.

It is important to underline that the qubits of a multi-qubit system in Qiskit are ordered with the first qubit on the right-most side and the last qubit on the left-most side. Therefore, the gate matrix representation is different to the gate matrix form illustrated in the Chapter 1, but the difference is slightly so we don't illustrate the correct matrix form implemented in Qiskit.

An example of how to build and manipulate a quantum circuit is shown in Figure 2.3. Here, we do the quantum Fourier transform of the state $|000\rangle$, implementing the quantum Fourier transform (QFT) protocol for $n = 3$ qubits [1].

```python
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from math import pi
#select the number of qubit
n = 3
#define the registers and create the quantum circuit
q = QuantumRegister(n,'q')
c = ClassicalRegister(n,'c')
circ = QuantumCircuit(q, c)
#add gate operation
circ.h(q[0])
circ.cu1(pi/2,q[1], q[0])
circ.cu1(pi/4,q[2], q[0])
circ.h(q[1])
circ.cu1(pi/2,q[2], q[1])
circ.h(q[2])
circ.swap(q[0],q[2])
for i in range(n):
    circ.measure(q[i], c[i])
#draw the circuit
circ.draw(output='latex')
```

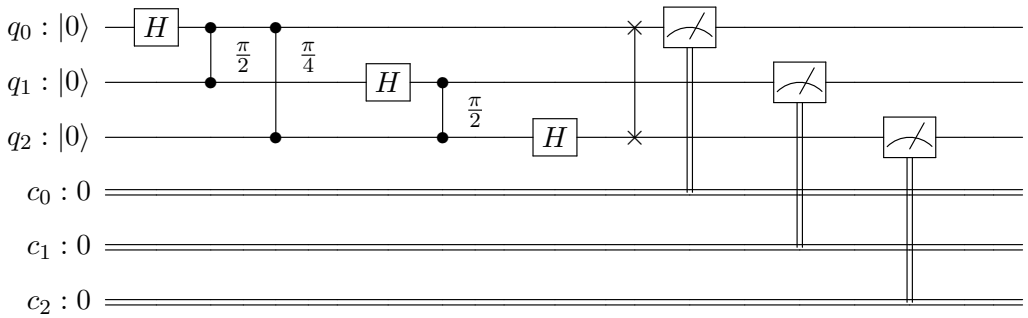


Figure 2.3: Top: the source python code in Qiskit for building the quantum circuit of QFT algorithm for $n = 3$ qubit is displayed. The QFT of the state $|000\rangle$ is done. Bottom: that quantum circuit is visualized in 'latex' output. The first three lines represent the quantum register, while the other three represent the classical register.

Parameterized circuits

Parameterization is a common feature of many quantum algorithms. Parameters/angles of an algorithm can be iteratively changed to minimize an energy or cost function. A 'Parameter' class is available and it can be used to specify a placeholder wherever a numeric parameter can be used. For instance, we illustrate a single-qubit parameterized circuit in Figure 2.4.

```python
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from math import pi
from qiskit.circuit import Parameter
#define parameter k
k = Parameter('k')
#create the circuit
n = 1
circ = QuantumCircuit(1, 1)
circ.rz(k, range(1))
```

Figure 2.4: Example of a parameterized circuit for $n = 1$ qubit in Qiskit software.

Running a quantum circuit

First of all, let us focus on what 'running a quantum circuit' actually means. In a given run of a quantum circuit with n measurements, the result will be one of the 2^n possible n-bit binary strings. If the experiment is run for a second time, even if the measurement is perfect and has no error, the outcome may be different, due to the fundamental randomness of quantum physics. The results of a quantum circuit executed many different times can be represented as a distribution over the full 2^n possible outcomes. It is not scalable to represent all possible outcomes; therefore, we keep only those outcomes that happen in a given experiment, and represent them as a histogram. In the histogram/bar graph representation the height of the bar represents the fraction of instances the outcome occurs during the experiment. Only those outcomes that occurred at least once are included. If all the bars are too small for visualization, they are collected into single bar called "other values".

Qiskit provides a useful interface for running quantum circuits. It is composed by three main components: the providers, which access backends and provide backend object, the backends, that run the quantum circuit, and the jobs, that keep track of the submitted jobs on a real hardware. There are two main providers available. The first one is the 'Qiskit Aer Provider' that includes backends to simulate the run of a quantum circuit on a quantum device. That simulation is done in a classical computer so exponential computational resources are used. The second provider is the 'IBM Q Provider' that support the access to IBM Quantum experience backends. Therefore, backends represent either a simulator or a real quantum computer and are responsible for running quantum circuits and returning results.

Simulators

The simulators available to simulate a run of a quantum circuit on a quantum device are contained mainly in the Qiskit Aer provider. It includes three simulator backends:

- `qasm_simulator`: is designed to mimic an actual device; allows ideal and noisy multi-shot executions of qiskit circuits and returns a count dictionary containing the final values of any classical register in the circuit. The maximum number of qubits supported by this simulator is 28 qubits.
- `statevector_simulator`: allows ideal single-shot executions of qiskit circuits and returns the final statevector of the simulation. If a circuit contains measure or reset gate operations the final statevector will be a conditional statevector after simulating wave-function collapse to the outcome of a measure or reset. Also for this simulator the maximum number supported is of 28 qubits.
- `unitary_simulator`: allows ideal single-shot executions of qiskit circuits and returns the final unitary matrix of the circuit itself. Note that the circuit cannot contain measure or reset operations for this backend and the maximum number of qubits supported is 14 qubits.

The IBM Q Provider supports the access to IBM Quantum experience backends that contains a remote optimized simulator backend called '`ibmq_qasm_simulator`'. This remote simulator is accessible by a cloud service and is capable of simulating up to 32 qubits.

If none is specified the simulators are noiseless, otherwise it is possible to implement a noise model for doing noisy simulations. There are three key classes in Qiskit Aer for building a noise model: the 'NoiseModel' class which stores a noise model, the 'QuantumError' class which describes gate errors and the 'ReadoutError' which describes classical readout errors.

An example of how Qasm Simulator acts is shown in Figure 2.7. In that example, the quantum circuit previously built as an instance, illustrated in Figure 2.3, is simulated on that backend. The results, that are plotted in the histogram on the left, are consistent with the theory [1], as expected in a noiseless simulation.

Real quantum devices

The IBM Quantum experience backends supported by IBM Q Provider are constituted by the remote simulator previously presented and by three real public quantum devices. They are made available publicly by IBM through a cloud service.

Let us illustrate the real quantum device provided by IBM:

- `ibmq_16_melbourne`: the IBM Q Melbourne is a $n = 14$ qubit real quantum device.
- `ibmqx2`: the IBM Q Yorktown ('ibmqx2') is a $n = 5$ qubits real quantum device.
- `ibmqx4`: the IBM Q Tenerife ('ibmqx4') is a $n = 5$ qubits real quantum device.

For each device, the native gate implemented are u_1, u_2, u_3, CX, $\mathbb{I}$, whose matrix forms are illustrated in Chapter 1. For the IBM Q Melbourne, the device layout and its connectivity are shown in Figure 2.5, while the Table 2.1 shows experimental parameters for each qubit of this device. For the IBM Q Yorktown and IBM Q Tenerife, look at the Figure 2.6 and the Table 2.2.

For instance, in Figure 2.7, we execute the QFT circuit, previously illustrated in Figure 2.3, in the IBM Q Melbourne device. The histogram on the right shows the results obtained. They are very different from the one obtained by the ideal simulation, due to noise that affects the real hardware.

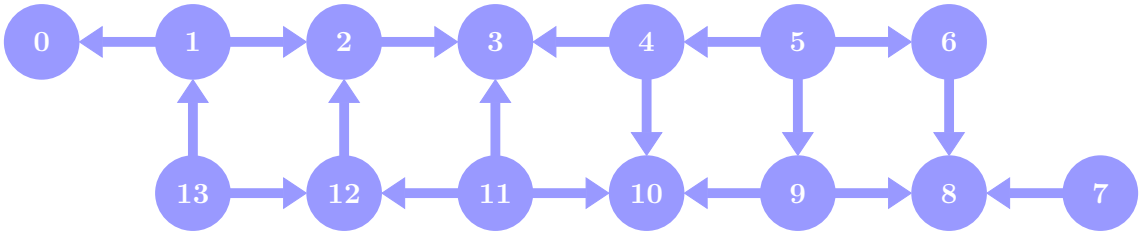


Figure 2.5: Layout and connectivity on 'ibmq melbourne'. The arrows represent the coupling map of this device.

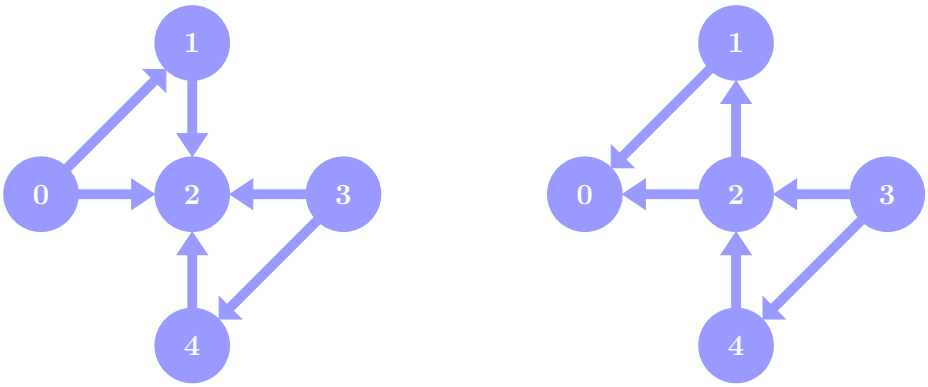


Figure 2.6: Layout and connectivity on left of 'ibmqx2' and on right of 'ibmqx4'. The arrows represent the coupling map of these devices.

Qubit	Frequency (GHz)	$T_1(\mu s)$	$T_2(\mu s)$	Readout error
Q_0	5.10	66.96	21.76	0.061
Q_1	5.24	64.06	101.46	0.069
Q_2	5.03	35.55	59.15	0.085
Q_3	4.89	21.92	28.49	0.128
Q_4	5.03	42.14	18.81	0.062
Q_5	5.07	26.42	48.51	0.073
Q_6	4.92	90.90	86.00	0.067
Q_7	4.97	51.10	86.05	0.091
Q_8	4.74	59.44	90.43	0.050
Q_9	4.96	50.33	86.21	0.045
Q_{10}	4.95	48.88	57.08	0.044
Q_{11}	5.00	54.33	91.86	0.036
Q_{12}	4.76	58.29	89.07	0.076
Q_{13}	4.97	21.71	35.82	0.056

Table 2.1: Typical qubit parameters for the 'ibmq melbourne' device, where T_1 and T_2 are the relaxation times of the qubit.

Qubit	Frequency (GHz)	$T_1(\mu s)$	$T_2(\mu s)$	Readout error
Q_0	5.29	56.01	60.43	0.028
Q_1	5.24	65.03	48.00	0.024
Q_2	5.03	71.12	60.17	0.014
Q_3	5.30	66.55	34.31	0.041
Q_4	5.08	56.09	28.91	0.027

Qubit	Frequency (GHz)	$T_1(\mu s)$	$T_2(\mu s)$	Readout error
Q_0	5.25	38.64	14.22	0.059
Q_1	5.30	52.83	9.97	0.079
Q_2	5.34	47.49	29.52	0.182
Q_3	5.43	48.66	27.64	0.356
Q_4	5.17	46.85	5.30	0.250

Table 2.2: Top: typical qubit parameters for the 'ibmqx2' device. Bottom: typical qubit parameters for the 'ibmqx4' device. T_1 and T_2 are the relaxation times of the qubit.

```
from qiskit import Aer, execute
from qiskit.tools.visualization import plot_histogram
# Select the QasmSimulator from the Aer provider
simulator = Aer.get_backend('qasm_simulator')
# Execute and get counts
result = execute(circ, simulator).result()
counts = result.get_counts(circ)
plot_histogram(counts, title='QFT algorithm counts')

from qiskit import *
IBMQ.load_accounts(hub=None)
from qiskit.tools.monitor import job_monitor, backend_monitor, backend_overview
# Select the IBM Q Melbourne backend
backend=IBMQ.get_backend('ibmq_16_melbourne')
# Execute and get counts
job = execute(circ,backend)
job_monitor(job)
result = job.result()
counts = result.get_counts(circ)
plot_histogram(counts, title='QFT algorithm counts')
```

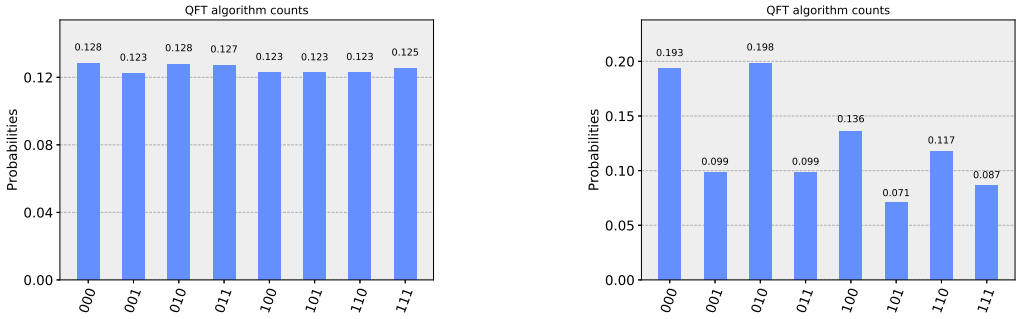


Figure 2.7: The QFT circuit described in Figure 2.3 is executed both on Qasm Simulator, in ideal condition, and on IBM Q Melbourne. At the top the python code is reported and at the bottom the histograms of count results are displayed. Left histogram: Qasm Simulator counts. The results are very closed to the one expected by the theory. Right histogram: IBM Q Melbourne counts. The results are different from the one obtained in the ideal case, due to the noise that affects the real hardware.

2.2.2 Cirq software

Cirq is an open-source framework for editing Noisy Intermediate Scale Quantum (NISQ) circuits. It provides a python library for working with quantum circuits and a high-performance local quantum circuit simulator. Cirq is still under development [9].

Manipulate a quantum circuit

In Cirq, a quantum circuit can be built using the `Circuit` class or the `Schedule` class. The `Circuit` object is related to the abstract quantum circuit model, as the `QuantumCircuit` object in Qiskit Terra, while the `Schedule` object is like the abstract quantum circuit model but includes also detailed information about the timing and duration of the gates. It is available a custom scheduler that converts the first object into the second one, therefore we just focus on `Circuit` class. A `Circuit` is a collection of `Moments`; each moment contains a set of `Operations` that all act during the same abstract time slice on a specific subset of qubits, as illustrated in Figure 2.8. The qubits can have two main geometrical structures depending on the natural structure of the device we want to use: they could be organized in a grid, `GridQubits`, or in a line, `LineQubits`. The `Gate` represents an operation that occurs on qubits. In Cirq, elementary gates as the Pauli gates, Hadamard gate, Controlled NOT gate or SWAP, are implemented. However, any unitary gate can be easily implemented by giving its matrix representation and defining 'Magic methods' in order to support functionality beyond basic tasks. To my experience, this functionality makes the gate implementation in Cirq simpler than in Qiskit, where a general gate must be written by the product of elementary gates. Cirq platform allows to perform arithmetic with circuits: for example the sum of two circuits is a circuit consisting of all the moments from the first circuit followed by all the moments from the second circuit. Also, circuit multiplication consists in the repeated addition of the same circuit. These features allow manipulation of quantum circuits in Cirq in an easy way (also Qiskit provides these functions).

In Figure 2.9, we illustrate an example of how a quantum circuit, with $n = 3$ qubits, is built in Cirq software. The circuit is defined into a Python function called 'circuit'. It transforms the state $|000\rangle$ into a GHZ state [1].

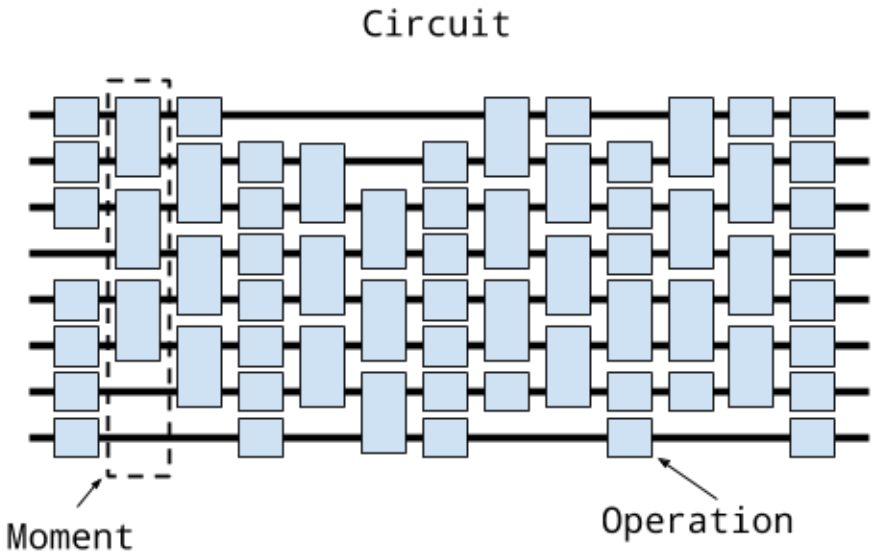


Figure 2.8: In Cirq a quantum circuit is built thanks to the `Circuit` class. A `Circuit` object is a collection of moments and each moment is constituted by operation, as gate operations, which act on a subset of qubits. Image from: https://cirq.readthedocs.io/en/stable/circuits.html.

[1] The GHZ state for $n = 3$ qubits is only an example for understanding how to write a simple code in Cirq, however in the Chapter 3 we will focus more deeply in the creation of GHZ states.

```
import cirq
import numpy as np
from math import pi
import matplotlib.pyplot as plt
#select the number of qubit and define LineQubits structure
n = 3
qubits = cirq.LineQubit.range(n)
#create the circuit
def circuit():
    circuit = cirq.Circuit()
    circuit.append(cirq.H(qubits[0]))
    for i in range(n-1):
        circuit.append(cirq.CNOT(qubits[i],qubits[i+1]))
    #measurement
    circuit.append(cirq.measure(*qubits, key='x'))
    print(circuit)
    return circuit
#select the simulator  and execute the circuit
def simulation(circuit):
    simulator = cirq.Simulator()
    results = simulator.run(circuit, repetitions=100)
    counts = cirq.plot_state_histogram(results)
#main function
def main ():
    simulation(circuit())
if __name__ == '__main__':
    main()
```

Figure 2.9: Python code for building the quantum circuit for creating a GHZ state of $n = 3$ qubits. In the Python function 'circuit' the quantum circuit is built, while in the function 'simulation' the circuit is executed in the desired simulator. The results of the execution are shown in Figure 2.11 on the left.

Parameterized circuits

The Cirq platform provides useful features for working with parameterized circuits. Cirq allows gates with parameters to have `Symbols`, which can be resolved by a `ParamResolver` with a given set of values, rather than having to create a new quantum circuit for every new set of variables, in fact we only have to change the value of the 'ParamResolver' to change the circuit. An example of a parameterized circuit is shown in Figure 2.10.

```
import cirq
import numpy as np
from math import pi
import sympy
#define LineQubits structure
qubits = cirq.LineQubit.range(1)
#create the circuit
circuit = cirq.Circuit()
# add a gate with a symbol which can take any value
gate = cirq.X**sympy.Symbol('s')
circuit.append(gate(qubits[0]))
circuit.append(cirq.measure(qubits[0], key="z"))
# get a param resolver
resolver = cirq.ParamResolver({'s': np.pi / 4.0})
# run the resolved circuit using the param resolver
simulator = cirq.Simulator()
results = simulator.run(circuit, resolver, repetitions=100)
counts = cirq.plot_state_histogram(results)
```

Figure 2.10: Python code for building a parameterized circuit for $n = 1$ qubit in Cirq software. The results of the execution are shown in Figure 2.11 on the right.

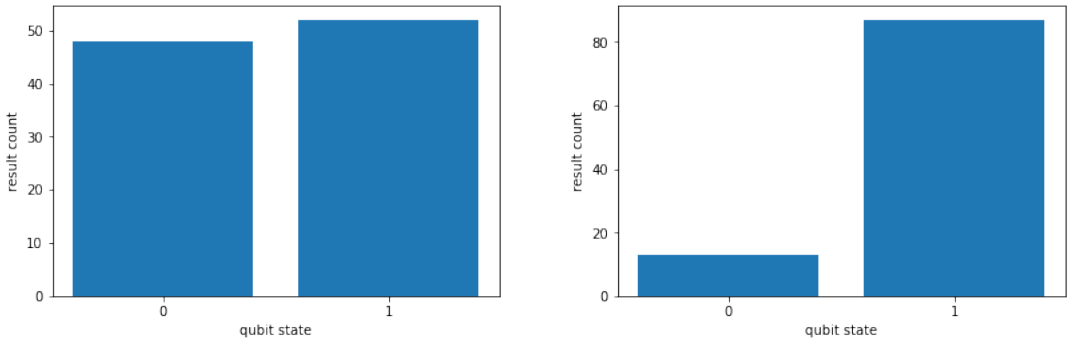


Figure 2.11: Left: histogram plot of data results of the GHZ quantum circuit of Figure 2.9. Right: histogram plot of data results of the parameterized circuit of Figure 2.10.

Running quantum circuits and simulators

In Cirq software 'running' a quantum circuit has the same meaning described yet in the section of Qiskit software, so we assume that this concept is already clear. So far, Cirq is not connected to any real device, but according to statements from Google, quantum computer will be available over the cloud in the near future, using Cirq as an interface. Indeed, Cirq already provides details on these devices as for example the architecture of the 22-qubit FoxTail computer or the 72-qubit Bristlecone computer. Althought the cloud service is still unavailable to general users, Cirq provides built in Python simulators for testing small circuits. The two main types of simulations that Cirq supports are pure state and mixed state. The simulators available are:

- `Simulator`: works for generic gates implementing their unitary matrix. There are two types of methods that simulator supports, the 'run methods' and the 'simulate methods'. The 'run methods' ('run and 'run_sweep') emulate a run on a quantum computer hardware and only return measurement results without giving access to the wave function. The output is returned as a 'Counter' object (Python built-in class) that displays key-value pairs corresponding to the output and number of times that output was recorded. The 'simulate methods' ('simulate', 'simulate_sweep', and 'simulate_moment_steps') can be used for full access to the wave function at the end of the simulation of the circuit. Once the circuit has been simulated, the result is stored and this output supports several useful features as Dirac notation of the state, access to the wave function or the computation of the (reduced) density matrix of the system.
- `google.XmonSimulator`: is specialized for the native gate set of Google's Xmon hardware. This simulator supports the same method as `Simulator` for doing different types of simulations. The only difference is that each gate used, or the map of the circuit, must be adapted for Xmon devices.
- `DensityMatrixSimulator`: supports involving mixed states simulation. In fact, it is a simulator for density matrices and noisy quantum circuits. This simulator supports the same methods of `Simulator` for different types of simulation.

For simulation of a simple quantum algorithm, any quantum computer simulator is essentially equivalent. However, the simulator chosen can become significant when larger numbers of qubits and gates are used in a quantum circuit. The maximum number of qubits that can be simulated depends on memory of the user's computer, in fact a large RAM implies larger circuits. The simulators of Cirq are good utilities but Qiskit ones have a high-performance [7] [8].

Currently, Cirq supports modeling noise via *operator sum representations* of noise. It consists on the possibility to add in the circuit common channels as Depolarizing channel, Bit flip channel, Phase flip channel or Amplitude Damping channel. These channels are added in the circuit in the same way as they were gate operations. Unfortunately, at the moment this is the only possibility for modeling noise in Cirq. In fact it is not possible to build a noise model for a given device as in Qiskit software.

In Figure 2.9 or in 2.10, the `Simulator` class is used for doing noiseless simulations. For each circuit, the histogram plots of the data results are shown in 2.11.

Chapter 3

Verifying N-qubit GHZ States

In this chapter, we focus on the creation through a quantum circuit of a particular entangled state, the GHZ state, in order to test the IBM and Google quantum computers. Quantum computers are beset with a certain amount of noise, so we need to quantify the goodness of a prepared state respect to the state we want to create. This can be done through fidelity measurements, that measure the probability that the prepared state coincides with the state we want.

First of all, we illustrate an experimental method to verify the generation of the GHZ state. It consists on measuring the multiple quantum coherences (MQC) [3] to infer the fidelity of the state we prepared. In fact, from MQC measurement it is possible to bound the state fidelity. The circuit implemented in this method is called MQC circuit.

Then such a MQC circuit of N-qubit is implemented in Qiskit software to be run on the IBM quantum computer; it is simulated on Qasm Simulator in ideal and noisy conditions. MQC measurements are done in both conditions. Afterwards the N-qubit circuit is executed on IBM Q Yorktown and IBM Q Melbourne, and we characterize the maximum number of N-qubit GHZ state that these devices support by MQC estimations. After that, we focus on Cirq software. The MQC circuit is implemented for $N = 5$ qubits and we add noise channels, as amplitude damping or depolarizing channels, to the circuit to study the fidelity bounds of the GHZ state in noisy conditions. Then, we build a circuit for creating a GHZ state of $N = 5$ qubits and we add noise channels as previously. We measure the elements of the density matrix of the system. Finally, a circuit designed to create a *Bell state*, is implemented, noise channels are added and we quantify the entanglement through concurrence measurements.

3.1 Experimental method to measure MQC

The *ideal* GHZ state is illustrated in Eq. (1.16). It has amplified sensitivity to phase rotations of each of the individual qubits in the entangled state. If each qubit has a phase rotation of ϕ, then the N-qubit rotates collectively by $N\phi$. In presence of noise, the GHZ will be partially mixed and therefore a different phase may be observed; by observing how sensitive a *non-ideal* GHZ state responds to phase rotations, it is possible to deduce how much entangled the state is. To this purpose, we need to measure multiple quantum coherence (MQC). The quantum circuit we use to prepare the GHZ state and measure MQC is shown in Figure 3.1. The collective rotation applied on each qubit is given by the unitary operator $U_\phi = e^{-i\frac{\phi}{2}\sum_j \sigma_z^j}$.

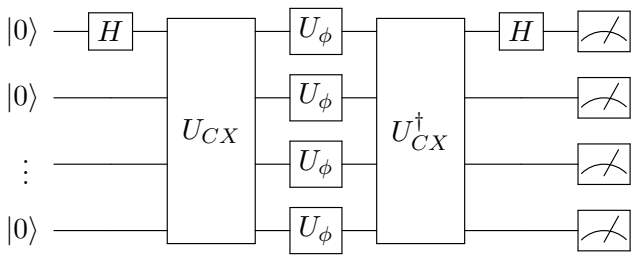


Figure 3.1: MQC quantum circuit for N qubits.

The quantum circuit for measuring MQC can be described in four steps [3]:

1. Starting from the N-qubit ground state: $|\text{GS}\rangle = |000\ldots0\rangle$, apply a Hadamard gate on qubit 0 followed by a sequence of CX gates. Ideally this brings the system into the GHZ state: $|\text{GHZ}\rangle = \frac{1}{\sqrt{2}}(|000\ldots0\rangle + |111\ldots1\rangle)$.
2. Apply a collective rotation given by the unitary U_ϕ on all qubits. This amounts to adding a phase $N\phi$ to the GHZ state: $|\text{GHZ}\rangle = \frac{1}{\sqrt{2}}(|000\ldots0\rangle + e^{-iN\phi}|111\ldots1\rangle)$.
3. Disentangle the GHZ state by performing the CX gate sequence in reverse order. The amplified phase is mapped onto qubit 0: $\frac{1}{\sqrt{2}}(|0\rangle + e^{-iN\phi}|1\rangle) \otimes |00\ldots0\rangle$. Finally, apply a Hadamard gate on qubit 0: $(\cos\frac{N\phi}{2}|0\rangle + i\sin\frac{N\phi}{2}|1\rangle) \otimes |00\ldots0\rangle$.
4. Read out the amplified phase by measuring the probability of the system returning to its initial state: $|\text{GS}\rangle$. Therefore, a measurement gate on each qubit it is applied.

Now, define $U_{GHZ} = U_{CX}H_0$, where H_0 refers to Hadamard gate applied on qubit 0. The measured signal is given by:

$$S_\phi = \left|\langle 000\ldots0| U_{GHZ}^\dagger U_\phi U_{GHZ} |000\ldots0\rangle\right|^2 = \text{Tr}(\rho_\phi \rho) \tag{3.1}$$

where $\rho = U_{GHZ}|000\ldots0\rangle\langle 000\ldots0| U_{GHZ}^\dagger$ and $\rho_\phi = U_\phi \rho U_\phi^\dagger$. In the ideal case, if there is no decoherence Eq. (3.1) reduces to

$$S_\phi^{\text{ideal}} = \frac{1}{2}(1 + \cos N\phi) \tag{3.2}$$

Any difference between the measured S_ϕ and S_ϕ^{ideal} is an indication that the GHZ state is imperfect. The parameter is $\phi = \frac{\pi j}{N+1}$, where $j = 0, 1, 2, \ldots, 2N+1$.

The **MQC amplitudes** are defined as the discrete Fourier transform of S_ϕ:

$$I_q = \mathcal{N}^{-1}|\sum_\phi e^{iq\phi} S_\phi| \tag{3.3}$$

where $\mathcal{N} = 2(N+1)$ is a normalization factor. Let us note that they are symmetric:

$$I_q = I_{-q}. \tag{3.4}$$

For an ideal N-qubit GHZ state , the nonzero elements in the density matrix resides only in the four corners. Therefore only three components arise in the expansion: $\rho^{GHZ} = \rho_0^{GHZ} + \rho_N^{GHZ} + \rho_{-N}^{GHZ}$. Explicitly they are given by

$$\begin{aligned} \rho_0^{GHZ} &= \frac{1}{2}(|000\ldots0\rangle\langle 000\ldots0| + |111\ldots1\rangle\langle 111\ldots1|) \\ \rho_N^{GHZ} &= \frac{1}{2}|000\ldots0\rangle\langle 111\ldots1| \\ \rho_{-N}^{GHZ} &= \rho_N^{GHZ\dagger} \end{aligned} \tag{3.5}$$

The corresponding multiple quantum amplitudes are

$$I_0^{GHZ} = \frac{1}{2}, \quad I_N^{GHZ} = \frac{1}{4}, \quad I_{-N}^{GHZ} = I_N^{GHZ} \tag{3.6}$$

The I_q are used to quantify the state fidelity. The N-qubit GHZ state fidelity defined as $F = \langle GHZ|\rho|GHZ\rangle$ can be bounded by [1]:

$$2\sqrt{I_N} \leq F \leq \sqrt{I_0/2} + \sqrt{I_N}. \tag{3.7}$$

For a perfect GHZ state F=1: $I_0 = 2I_N = 1/2$ and all other I_q being zero. For a N-qubit state to have multipartite GHZ entanglement, it needs to have a minimal fidelity of 0.5 [14].

[1] The proof can be found in the supplementary material section of [3].

3.2 MQC Circuit on Qiskit software

Following the protocol given in the previous section, we realize the experimental quantum circuit for measuring MQC, called MQC circuit, for different values of N qubits and we execute it on different backends provided by IBM. The experimental matrix U_ϕ implemented in the circuit has the following form:

$$U_\phi = \begin{pmatrix} e^{-i\frac{\phi}{2}} & 0 \\ 0 & e^{i\frac{\phi}{2}} \end{pmatrix} \tag{3.8}$$

3.2.1 Measurements and analysis

For each value of N and regardless of the backend used, it is possible to describe generally how the results are obtained and analyzed:

1. The value of ϕ is fixed; the backend executes the quantum circuits k times and returns a count dictionary containing the final values of any classical registers in the circuit.
2. The count of the state $|000\dots0\rangle$ is taken because it correspond with the measurement S_ϕ in Eq. (3.1). Then that count is divided by k to obtain it in the range from 0 to 1.
3. The first two steps are executed again for each value of $\phi = \frac{\pi j}{N+1}$, where $j = 0, 1, 2, \dots, 2N+1$. In that way, a list of counts at different values of ϕ is obtained.
4. A plot of the function S_ϕ^{ideal} in Eq. (3.2) is done; it represents the theoretical results the experiment should give. Then the experimental list of data is plotted in function of ϕ and it is compared with such a theoretical plot.
5. The MQC amplitudes are calculated by the discrete Fourier transform I_q, in Eq. (3.3), of that list and plotted in the range $[-N, N]$. They are compared with the MQC amplitudes obtained by the Fourier transform of S_ϕ^{ideal}. Then, the fidelity can be bounded as in Eq. (3.7).

3.2.2 Simulation

The N-qubit MQC circuit is executed on Qasm Simulator of IBM. Firstly, we suppose an execution of the circuit in an ideal condition, then we consider a noise model based upon the real quantum device IBM Q Melbourne.

Ideal model

Consider an ideal execution, where the quantum circuit is supposed to be run on an ideal quantum device. The backend Qasm Simulator executes the circuit $k = 8192$ times and returns a list of counts of the state $|000\dots0\rangle$ for different ϕ, as described in Section 3.2.1. The circuit is implemented and ideally simulated from $N = 2, \dots, 24$ qubits, until computational resources are enough to execute the circuit [2]. The results obtained are very close to the theoretical expectation up to a statistical fluctuations given by the probabilistic simulation. For each N the fidelity bounds, defined by Eq. (3.7), are ~ 1.

For instance, we suppose $N = 6$ qubits. The corresponding MQC circuit implemented in Qiskit is shown in Figure 3.2 and the ideal execution results are plotted in Figure 3.4. In the plot on the left we compare the theoretical results S_ϕ(line in red) to the experimental counts of the simulation (points in blue). The right plot shows the Fourier transform of S_ϕ^{ideal} (line in red) and the MQC amplitudes (3.3) in such an ideal case for $N = 6$ qubit (points in blue). The plots for all the other values of N we have simulated are similar and the fidelity is still good: for example, the results for $N = 22$ qubit are shown in Figure 3.4.

[2] The computational resources refers to a classical computer of 8 Gbyte of RAM.

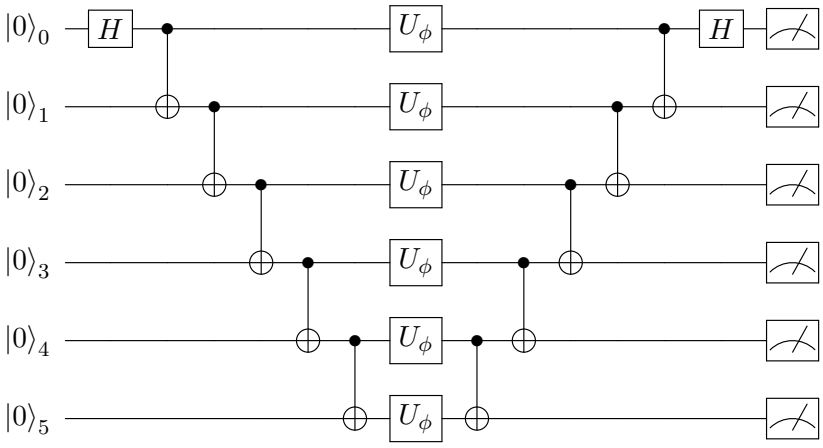


Figure 3.2: MQC quantum circuit for $N = 6$ qubits.

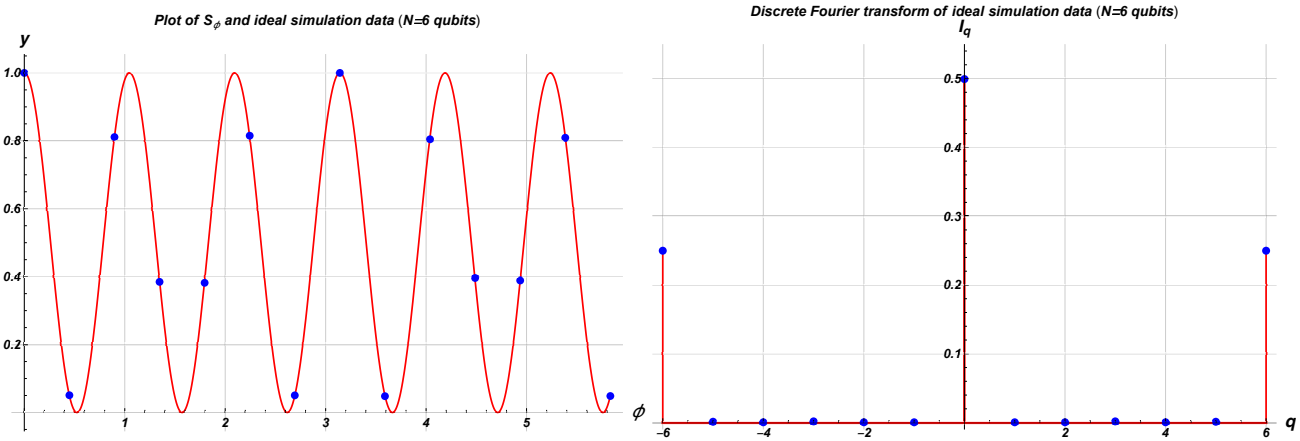


Figure 3.3: **N=6** qubits. Left: plot of S_ϕ^{ideal} is represented in red, experimental measurements S_ϕ obtained by the simulation in the **Qasm Simulator** backend in the ideal case are drawn in blue. Right: corresponding MQC amplitudes I_q of S_ϕ^{ideal} and S_ϕ, respectively in red and blue.

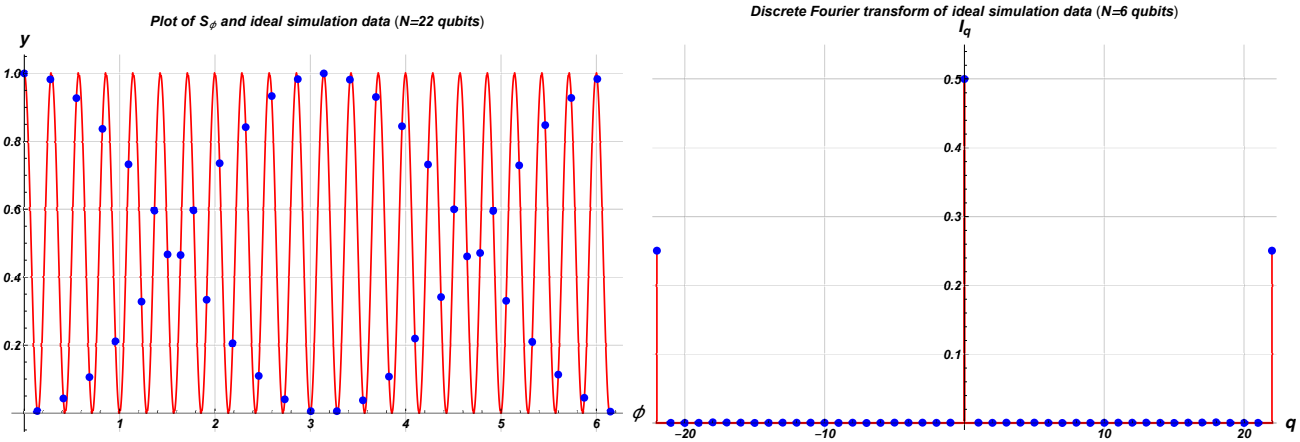


Figure 3.4: **N=22** qubits. Left: plot of S_ϕ^{ideal} is represented in red, experimental measurements S_ϕ obtained by the simulation in the **Qasm Simulator** backend in the ideal case are drawn in blue. Right: corresponding MQC amplitudes I_q of S_ϕ^{ideal} and S_ϕ, respectively in red and blue.

Noise model

We apply the same protocol to a circuit implemented with a noise model associated to the IBM Q Melbourne device. Noise is added on all qubits and on all basis gates of this device. The read-out error is also implemented. These error values are available online [11]. In addiction, the typical gate time are provided. Since the IBM documentation does not currently provide the gate times for gates of this device we manually insert them, basing on values found online. The code used for doing the model is shown in Figure 3.5.

The circuit is simulated for $N = 2, \ldots, 9$ qubits. For each value of N the fidelity bounds are estimated as previously by measuring MQC obtained by discrete Fourier transform of S_ϕ. We plot the fidelity lower and upper bounds in function of the number N of qubits in Figure 3.7. They are respectively the green and the orange points. If the fidelity bounds are in the range $[0.5, 1]$, we consider the GHZ state verified, as we mentioned in Section 3.1. Otherwise, the fidelity level is not acceptable and the device is not able to create the GHZ state with a good confidence. In the case we are considering, the device is able to create a good GHZ state until $N = 7$ qubit. In Figure 3.6, we show the plots of S_ϕ and of I_q for $N = 7$ qubits .

```
device = IBMQ.get_backend('ibmq_16_melbourne')
properties = device.properties()
gate_times = [
    ('u1', None, 0), ('u2', None, 100), ('u3', None, 200),
    ('cx', [1, 0], 678), ('cx', [1, 2], 547), ('cx', [2, 3], 721),
    ('cx', [4, 3], 733), ('cx', [4, 10], 721), ('cx', [5, 4], 800),
    ('cx', [5, 6], 800), ('cx', [5, 9], 895), ('cx', [6, 8], 895),
    ('cx', [7, 8], 640), ('cx', [9, 8], 895), ('cx', [9, 10], 800),
    ('cx', [11, 10], 721), ('cx', [11, 3], 634), ('cx', [12, 2], 773),
    ('cx', [13, 1], 2286), ('cx', [13, 12], 1504), ('cx', [], 800)]

# Construct the noise model from backend properties and custom gate times
noise_model = noise.device.basic_device_noise_model(properties, gate_times=gate_times
    )
print(noise_model)
```

Figure 3.5: Python code on Qiskit for building the device noise model based upon 'ibmq melbourne' real quantum device. The gate times are taken from [11].

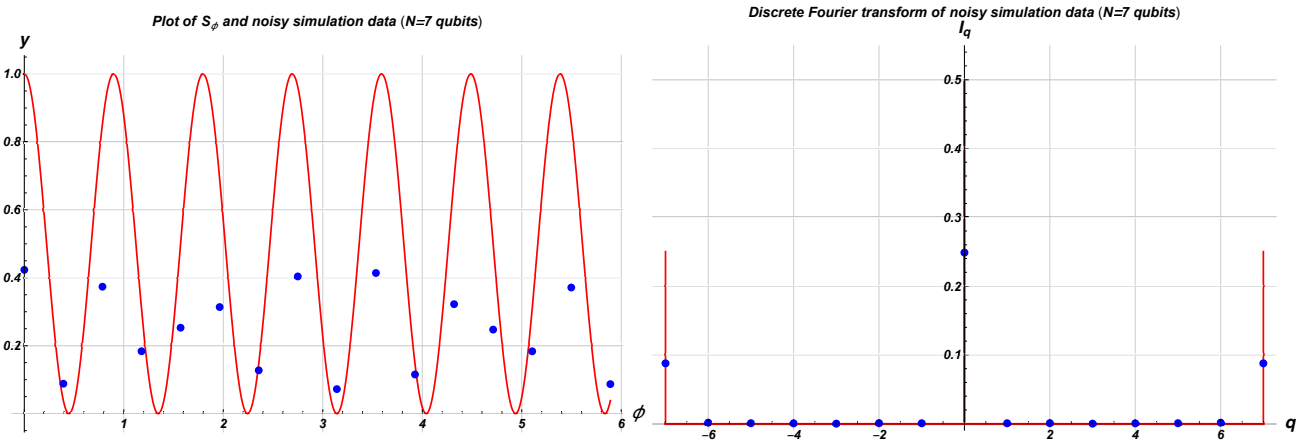


Figure 3.6: **N=7** qubits. Left: plot of S_ϕ^{ideal} is represented in red, experimental measurements S_ϕ obtained by the simulation in the **Qasm Simulator** backend in noisy conditions are drawn in blue. Right: corresponding MQC amplitudes I_q of S_ϕ^{ideal} and S_ϕ, respectively in red and blue.

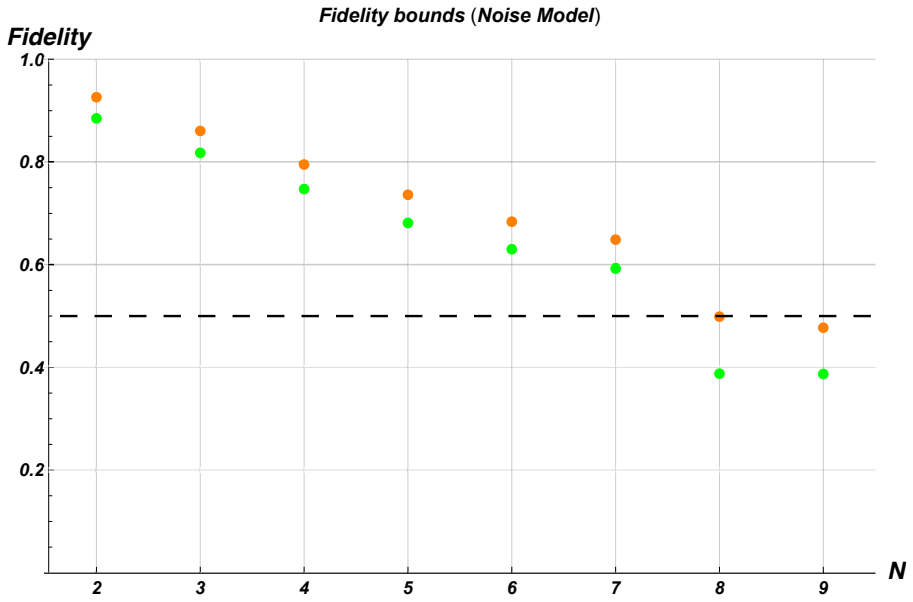


Figure 3.7: Fidelity lower bounds and upper bounds, respectively in green and orange, for the N-qubit MQC circuit, executed in a user-built **noisy device**, are plotted in function of the number of qubits N. The line dashed in black represents the limit of 0.5 and under that we consider the fidelity of the state not acceptable.

3.2.3 Run on real devices

The MQC circuit is executed in two backends: IBM Q Yorktown and IBM Q Melbourne devices. For each device, we implement the circuit in two different ways:

1. In the first case, we add the gates in the MQC circuit taking no care on how the real hardware is built. It means that we always use the physical qubits from 0 to N, in order, of each device and that a sequence of CX gate is added between one qubit and its subsequent.
2. In the second case, the circuit is slightly modified taking into account the coupling map of the considered device and qubit's gate errors. In particular, the coupling maps shown in Figure 2.6 and 2.5 are analyzed and depending on qubit connectivity and gate error, we pick up the most advantageous physical qubits for our aim. The physical qubits used for the IBM Q Yorktown and IBM Q Melbourne backends, are shown in Table 3.1.

Backend: IBM Q Yorktown

We consider the execution of MQC circuit in the IBM Q Yorktown device. This backend supports 5 qubits as illustrated in Chapter 2. The qubit's properties are listed in Table 2.2 and the device's coupling map is shown in Figure 2.6. The MQC circuit is built in both of the methods shown before and it is executed for $N = 2, \ldots, 5$ qubits, which is the maximum number available.

In the first case, the physical qubits used are in order $[0, 1, 2, 3, 4]$. In the second case, after the analysis of the coupling map, almost the same sequence of qubits has been chosen. In fact, the former pattern showed up as the better solution by chance. Therefore, the two implementation methods are approximately equivalent. Nevertheless, we illustrate the physical qubits used in the second method for the IBM Q Yorktown device in Table 3.1 on the left.

The fidelity bounds obtained by the two methods are plotted as a function of the N qubits in Figure 3.9. It is possible to notice that, as already mentioned, the two plots are quite similar within statistical errors. Anyway, the fidelity bounds for $N = 2, 3, 4$ qubits are acceptable and the corresponding N-GHZ states are verified. For $N = 5$ the lower bound is not acceptable, but the upper is it. Nevertheless, we consider that state not verified. In Figure 3.8, we illustrate the S_ϕ and the I_q plots for $N = 4$ qubits.

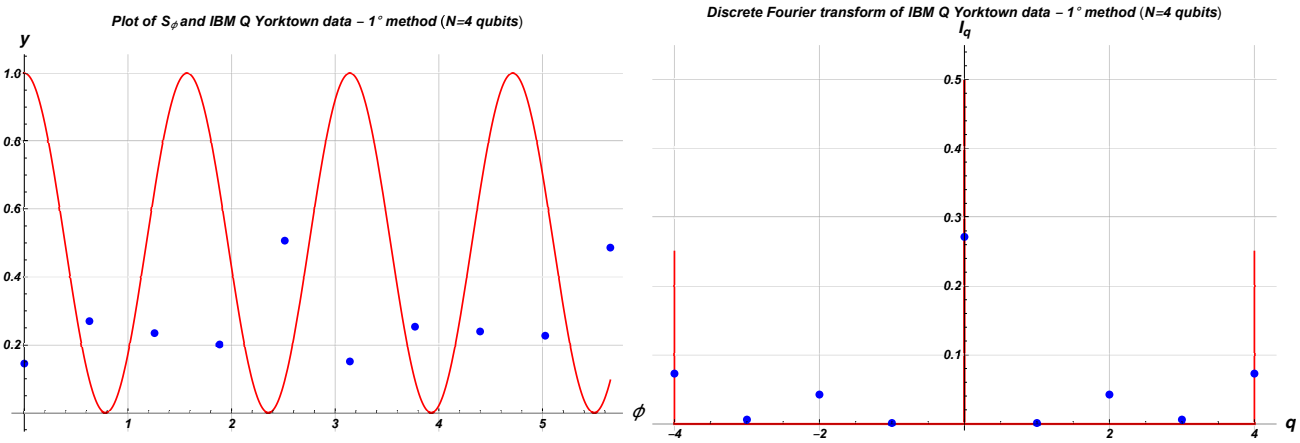


Figure 3.8: **N=4** qubits. Left: plot of S_ϕ^{ideal} is represented in red, experimental measurements S_ϕ obtained by the simulation in the **IBM Q Yorktown** backend are drawn in blue. Right: corresponding MQC amplitudes I_q of S_ϕ^{ideal} and S_ϕ, respectively in red and blue.

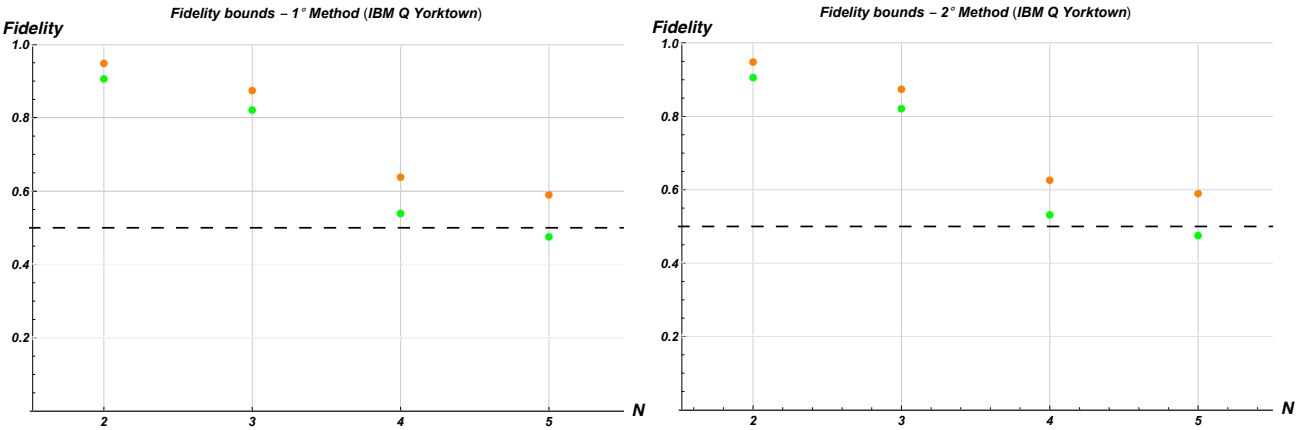


Figure 3.9: Fidelity lower bounds and upper bounds, respectively in green and orange, for the N-qubit MQC circuit, executed in the **IBM Q Yorktown** backend, are plotted in function of the number of qubits N. The line dashed in black represents the limit of 0.5 and under that we consider the fidelity of the state not acceptable. Left: first method plot. Right: second method plot.

Backend: IBM Q Melbourne

Now, we consider the execution of the MQC circuit in the IBM Q Melbourne device, which supports 14 qubits. Its properties and coupling map are illustrated in Table 2.1 and in Figure 2.5. We implement the circuit in both of the two ordering schemes, described above, for $N = 1, \ldots, 6$ qubits. In the first method, the physical qubits used are $[0, 1, \ldots, 5]$ in order. In the second method, we distinguish two cases: for $N = 2, 3, 4$, we do a better choice of the physical qubits used, while for $N = 5, 6$ we also slightly modify the circuit. In Figure 3.10, we illustrate the new circuit implemented for $N = 5$ qubits. The difference with the one in Figure 3.2 is that the CX gate is no more added between one qubit and its subsequent.

The results obtained by the two methods are quite different, unlike the IBM Q Yorktown device. The fidelity bounds in both cases are plotted in Figure 3.12. In the first case only the GHZ state for $N = 2$ qubits is verified, in fact all the other fidelity bounds are not acceptable, because they are lower than 0.5. In the second case, the GHZ states for $N = 2, 3, 4$ qubits are verified and the results for $N = 5, 6$ qubits are not acceptable, but they are however better than the first method. These surprisingly results show how the IBM Q Melbourne quantum device is highly sensitive to a correct qubits choice. It is also possible to notice that IBM Q Melbourne device is more unstable than the IBM Q Yorktown, due to the more number of qubits it has to manage. To be thorough, in Figure 3.11 we show the S_ϕ and I_q plots for $N = 4$ qubits, obtained by the second method.

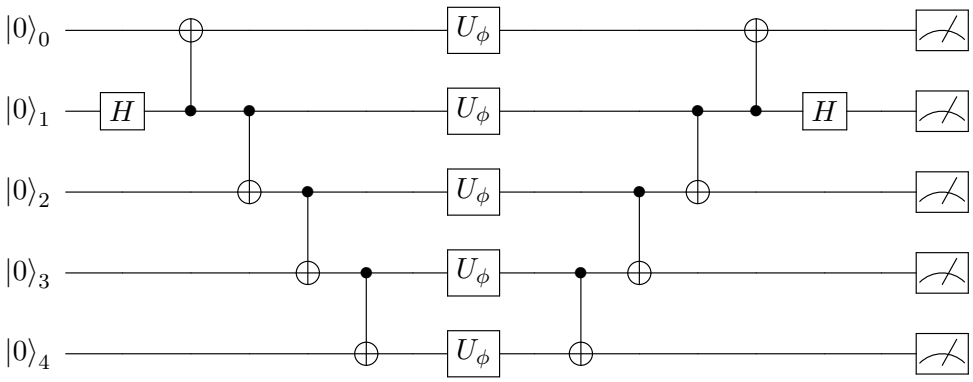


Figure 3.10: 5-qubit MQC circuit implemented in the IBM Q Melbourne device in the second method.

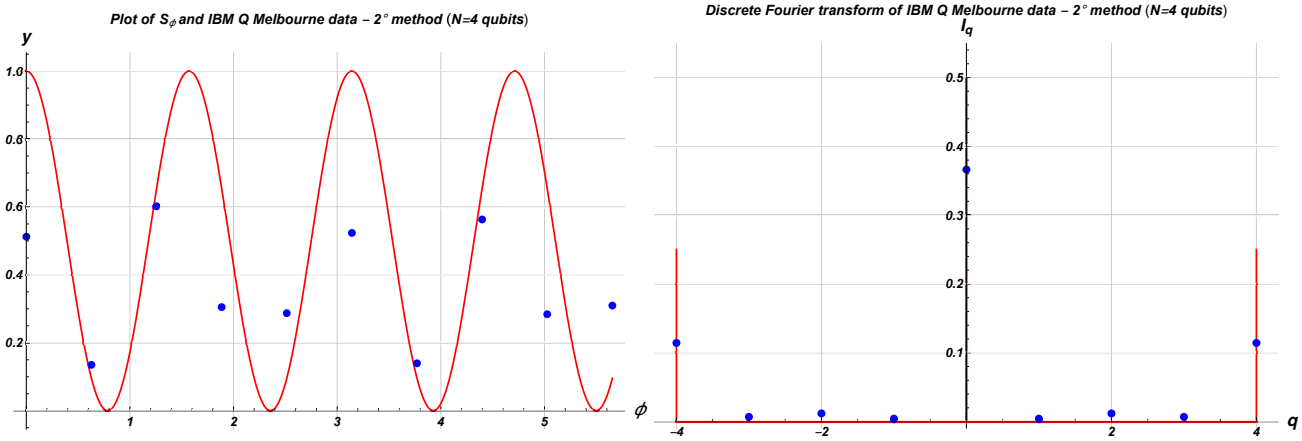


Figure 3.11: **N=4** qubits. Left: plot of S_ϕ^{ideal} is represented in red, experimental measurements S_ϕ obtained by the simulation in the **IBM Q Melbourne** backend are drawn in blue. Right: corresponding MQC amplitudes I_q of S_ϕ^{ideal} and S_ϕ, respectively in red and blue.

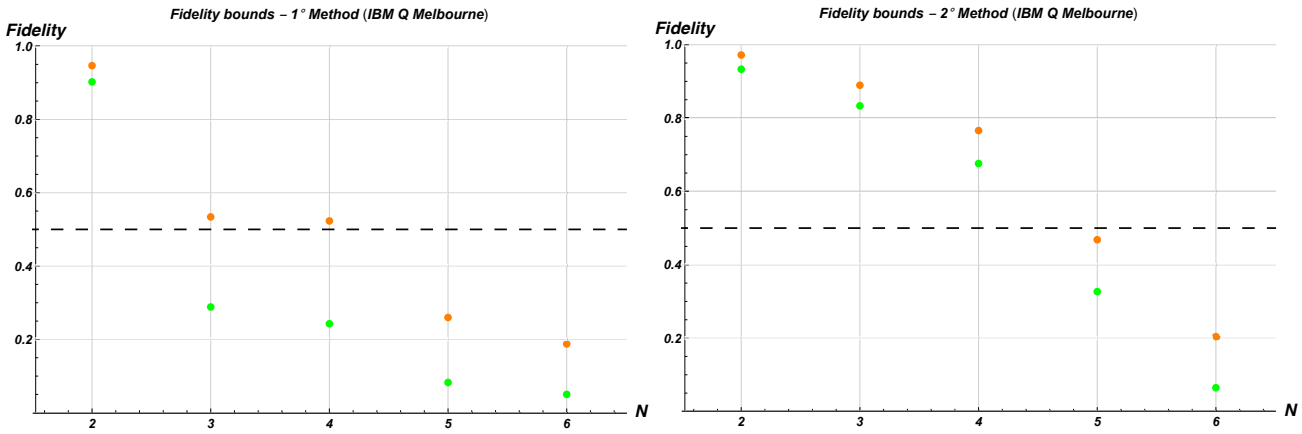


Figure 3.12: Fidelity lower bounds and upper bounds, respectively in green and orange, for the N-qubit MQC circuit, executed in the **IBM Q Melbourne** backend, are plotted in function of the number of qubits N. The line dashed in black represents the limit of 0.5 and under that we consider the fidelity of the state not acceptable. Left: first method plot. Right: second method plot.

N	Physical qubits used
2	[0, 1]
3	[0, 1, 2]
4	[0, 1, 2, 4]
5	[0, 1, 2, 3, 4]

N	Physical qubits used
2	[11, 10]
3	[11, 12, 2]
4	[13, 12, 2, 3]
5	[1, 13, 12, 2, 3]
6	[0, 1, 13, 12, 2, 3]

Table 3.1: Table of physical qubits used in the MQC circuit implemented in the real devices, in the second method. Left: IBM Q Yorktown device. Right: IBM Q Melbourne device.

3.3 MQC Circuit on Cirq software

Now, let us test the Cirq software, the Google quantum computer interface. The analysis is carried out in two phases. First, we implement the MQC circuit for different values of N, in noiseless condition. The matrix U_ϕ has the form of Eq. (3.8) and the results are obtained and analyzed as illustrated in the Section 3.2. In particular, we execute the circuit for $N = 2, \ldots, 22$ qubits on the Density Matrix Simulator with 6000 shots. The results S_ϕ and I_q are the same of the ones obtained by the simulation in the Qasm Simulator of IBM in the ideal case, so they are not illustrated again. Then, we consider for instance the MQC circuit of $N = 5$ qubits [3] and we add different types of quantum channels on it; in fact, it is interesting to focus on a system with few qubits and answer to the question: how does the state change as a function of the parameters of the channels? We execute the circuit in the Density Matrix Simulator with 6000 shots and, for each type of channel, we plot the fidelity bounds as a function of the parameter of the channels.

3.3.1 Adding quantum channels

We add quantum noise channels to the MQC circuit illustrated in Section 3.1. The main noise channels have been previously illustrated in Section 1.4. They are added to the circuit in the following way:

1. First, we choose the type of noise channel to add in the circuit. For instance, we consider the Amplitude damping channel or the Depolarizing channel.
2. The parameters of all the channels we will insert are fixed at the same value γ.
3. Noise channels, all of the same type, are inserted to the circuit between any two contiguous unitary gates. In particular, we add the channel following a definite procedure: the number of channels inserted between one gate and its subsequent is proportional to the time interval between them. For instance, we divide in time slices the MQC circuit without any noise channels. Each slice represents a Moment of the circuit (see Section 2.2.2). We suppose that the fixed time between two contiguous time slices is $\Delta t = 1\,\mu s$. Therefore, if for example two contiguous gate are four time slices distant, we insert four channels and each of them has the parameter fixed at the value γ. If they are one time slice distant, we insert only one channel.

The new circuit is shown in Figure 3.13, where the noise channels are represented as single-qubit gates with a number n. The number value, of each gate symbol, represents the number of times the noise channel is inserted.

We execute the circuit for different values of the parameter of the channels in the Density Matrix Simulator with 6000 shots. In particular, the results for a fixed value are obtained and analyzed, as illustrated in Section 3.2.1. Then, we plot the fidelity bounds as a function of the fixed parameter of the channels. We take the larger value of the parameter acceptable, for which the fidelity bounds are upper than 0.5 and the 5-qubit GHZ state is verified. Afterwards, we calculate the maximum relaxation time associated to the channel. For example, suppose that the largest value of the parameter is p. The relaxation time T is calculated from the relation:

[3] We note that with a computer of 8 Gb of RAM we could not simulate over than $N = 12$ qubits in noisy condition, because of the high computational costs.

$$p = 1 - e^{-\Delta t/T} \quad \Longrightarrow \quad T = -\frac{\Delta t}{\log(1-p)}. \tag{3.9}$$

We underline that the values obtained are not reliable. In fact, more sophisticated experiments should be done for calculating the relaxation times of a system [11]. Therefore, they represent just an estimate and do not reflect the actual value.

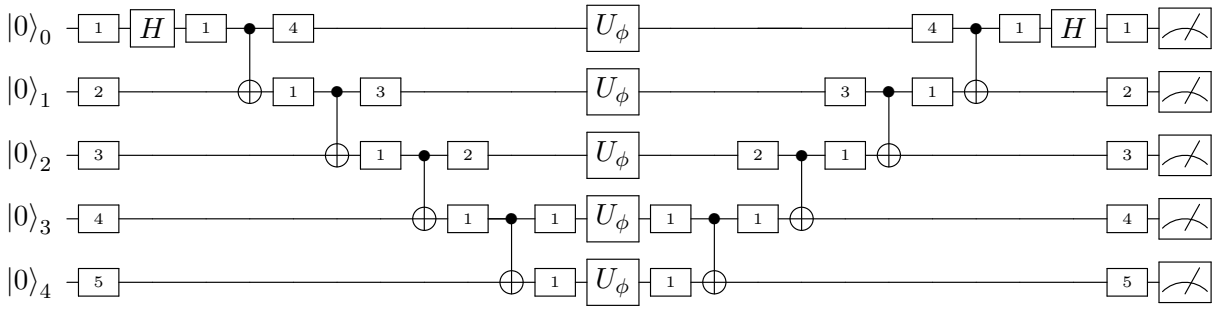


Figure 3.13: 5-qubit MQC circuit with quantum noise channels. The number value, of each gate symbol, represents the number of times the noise channel is inserted.

Now, we apply the procedure described in the case we add respectively only Amplitude Damping or Depolarizing channels.

Amplitude damping channels

In Figure 3.14, we illustrate the fidelity bounds of the 5-qubit GHZ state as a function of the parameter γ of the Amplitude Damping channel. The state is verified until the parameter reaches the value of $p = 0.08$. The relaxation time is calculated as in Eq. (3.9) and it results $T_{AD} = 11.99\,\mu s$. We remind that, for the Amplitude Damping channel, the relaxation time T_{AD} is the time in which the state of the system decays into the ground state (see Section 1.4.3). Therefore, if we suppose that $\Delta t = 1\,\mu s$ is the time between the application of two subsequent gates, after ~ 12 gate applications, the state decays into the ground state.

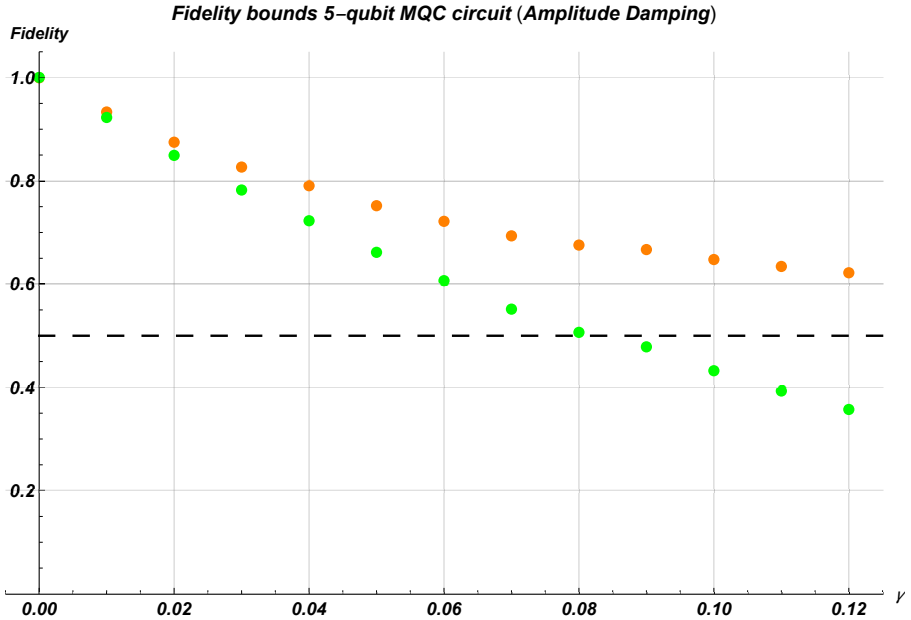


Figure 3.14: Fidelity lower bounds and upper bounds, respectively in green and orange, for the 5-qubit MQC circuit, executed in the Density Matrix Simulator, are plotted in function of the values of the **Amplitude Damping** channel parameter. The line dashed in black represents the limit of 0.5 and under that we consider the fidelity of the state not acceptable.

Depolarizing channels

In Figure 3.15, we illustrate the fidelity bounds of the 5-qubit GHZ state as a function of the parameter γ of the Depolarizing channel. The state is verified until the parameter reaches the value of $p = 0.022$. The relaxation time is calculated as in Eq. (3.9) and it results $T_D = 44.95\,\mu s$. For the Depolarizing channel, the relaxation time T_D is the time in which the state of the system decays into a completely mixed state (see Section 1.4.2). Therefore, it decays after ~ 45 gate operations.

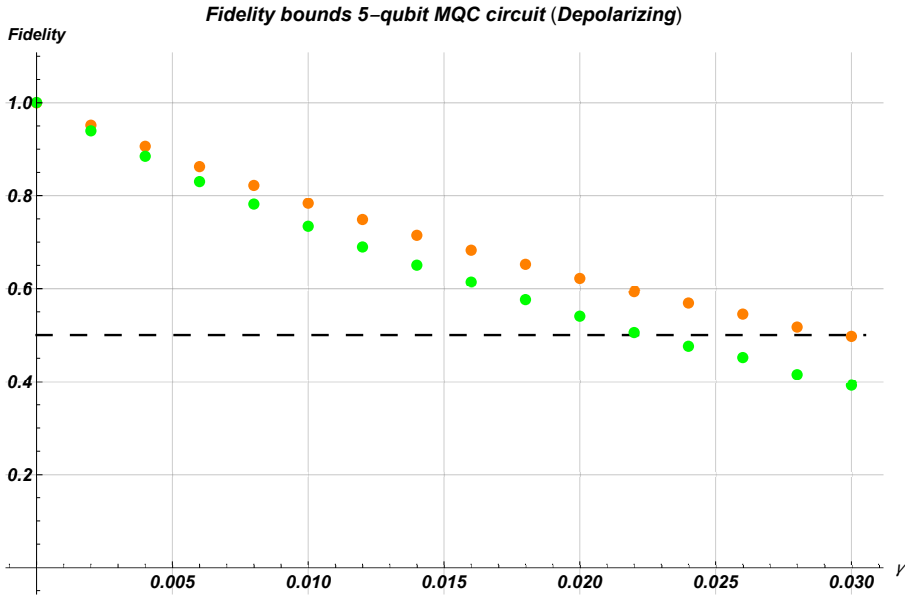


Figure 3.15: Fidelity lower bounds and upper bounds, respectively in green and orange, for the 5-qubit MQC circuit, executed in the Density Matrix Simulator, are plotted in function of the values of the **Depolarizing** channel parameter. The line dashed in black represents the limit of 0.5 and under that we consider the fidelity of the state not acceptable.

In conclusion, we note that $T_{AD} < T_D$, therefore the largest time for which the state does not decay is T_{AD}. In reality, the effects act together, so probably even less time would be obtained.

3.4 GHZ state: density matrix elements

Another way to recognize a GHZ state is to study some peculiar matrix elements, i.e. the four corners of the density matrix of the system. In fact, for an ideal GHZ state, the nonzero elements in the density matrix reside only in the four corners and their value is of 0.5. Therefore, we consider a 5-qubit GHZ state prepared following the protocol illustrated in Section 3.3.1 and the circuit implementation is shown in Figure 3.16. As done previously, we distinguish two cases with noise due only to Amplitude Damping channels or only to Depolarizing channels. We fix the parameter γ of the channels and we simulate the circuit in the Density Matrix Simulator. Then, we analyze the goodness of the state obtained by measuring the four corners of the density matrix of the system.

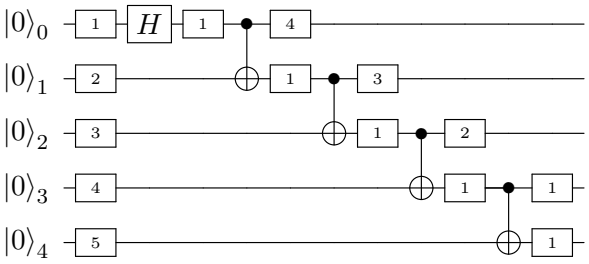


Figure 3.16: 5-qubit circuit with quantum noise channels. The number value, of each gate symbol, represents the number of times the noise channel is inserted.

Amplitude damping channels

In Figure 3.17, we show the value of the four corners elements of the density matrix of the system as a function of the parameter γ of the Amplitude Damping channel. We note that by increasing the parameter value, the system state decays into the ground state: the matrix element ρ_{00} tends asymptotically to 1, while the other tend to 0.

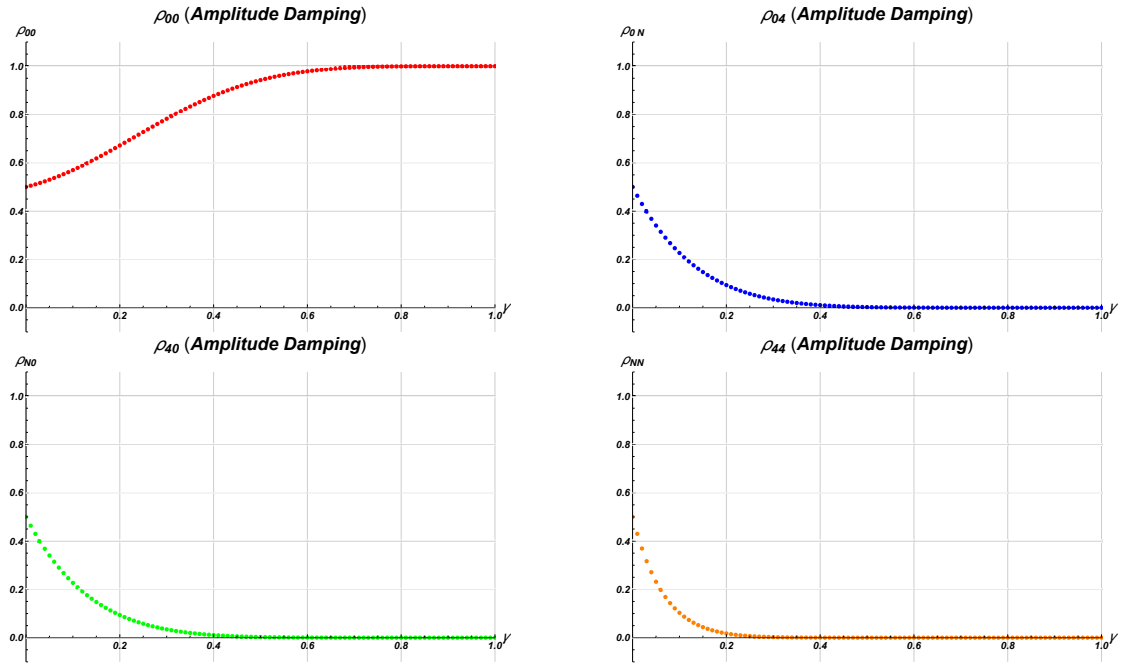


Figure 3.17: Corner elements of the density matrix of the 5-qubit system in function of the parameter γ of the **Amplitude Damping** channel. The system state decays into the ground state.

Depolarizing channels

In Figure 3.18, we show the value of the four corners elements of the density matrix of the system as a function of the parameter γ of the Depolarizing channel.

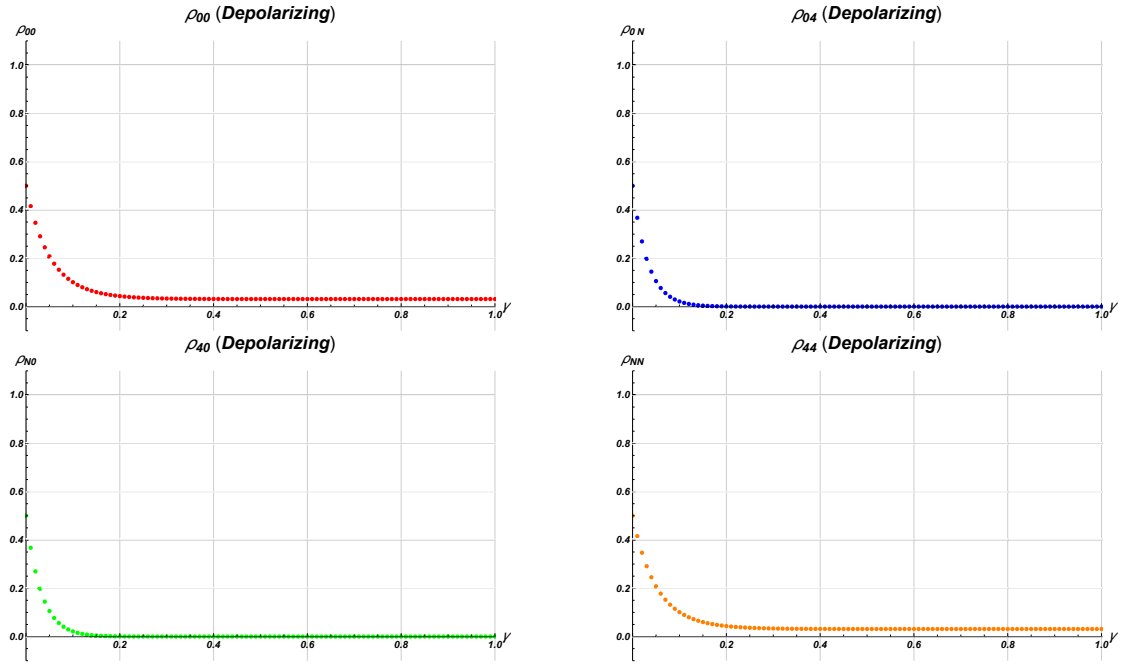


Figure 3.18: Corner elements of the density matrix of the 5-qubit system in function of the parameter γ of the **Depolarizing** channel. The system state decays into a completely mixed state.

For the Depolarizing channel, we note that by increasing the parameter value, the system state decays into a completely mixed state, for which the nonzero density matrix elements have the same value and reside only in the diagonal. In particular, the density matrix of the 5-qubit system we are considering, has the form 32×32, therefore the elements ρ_{00} and ρ_{44} tend to $\frac{1}{32}$, as we can see in Figure 3.18. Also the other elements in the diagonal tend to this value, while all the elements outside tend to 0.

3.5 Bell state: concurrence measurements

In the Cirq software, we implement a simple circuit of $N = 2$ qubits that makes a *Bell state*. Quantum channels are added following the protocol illustrated in Section 3.3.1. Firstly, we insert only Amplitude Damping channels in the circuit, then only Depolarizing channels. The circuit for both configuration is illustrated in Figure 3.19.

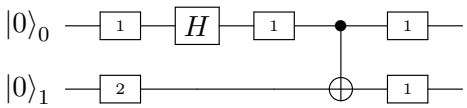


Figure 3.19: 2-qubit circuit with quantum noise channels. The number value, of each gate symbol, represents the number of times the noise channel is inserted.

For both configuration, we fix the parameter γ of the channels and we simulate the circuit in the Density Matrix Simulator, that gives us the access to the density matrix of the state. We quantify the order of entanglement of the state by doing **concurrence** measurements, defined in Eq. (1.20). For each case, the concurrence values are plotted as a function of the parameter of the channels in Figure 3.20. As we expected, the concurrence value decreases when the parameter value increases.

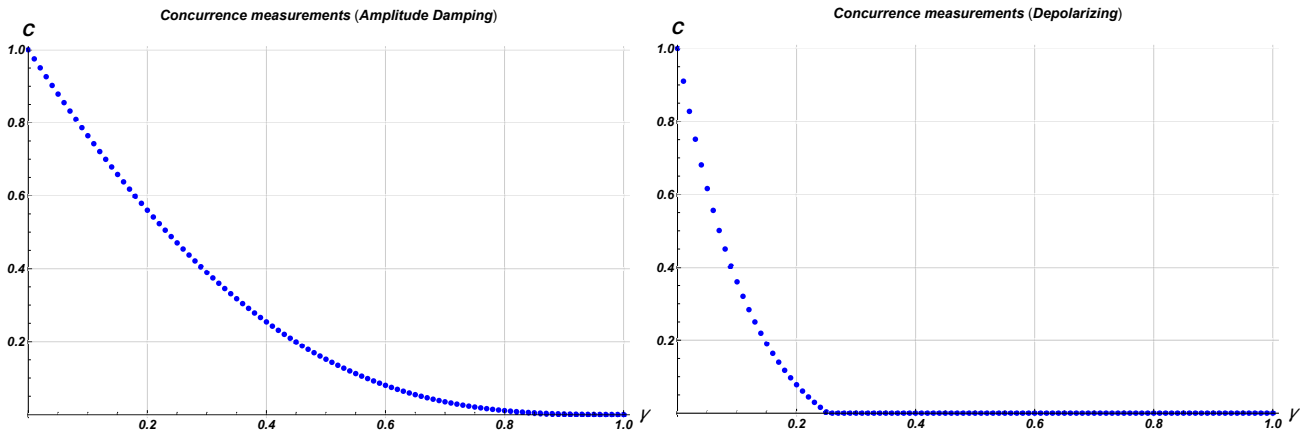


Figure 3.20: Concurrence values for the 2-qubit system as a function of the parameter γ of the channels. Top: only Amplitude Damping channels are inserted into the circuit. Bottom: only Depolarizing channels are inserted into the circuit.

Conclusions

To date, available quantum computers are small, noisy, and not nearly as powerful as current classical computers. While on the one hand hardware development is a critical aspect of quantum computation and computers with more qubits are being built, on the other hand part of nowadays researches focus on quantum software development.

In this Thesis, after we have introduced the fundamental concepts of quantum computation and information, we have analyzed the two software developed by the IBM and Google companies, Qiskit and Cirq. In general, we find that currently the Qiskit platform is more developed than the Cirq one. A comprehensive tutorial and a wide community for Qiskit exist, which make this software easly approachable for a beginner, but more importantly a cloud service allows a general user to interact with a real hardware. The Cirq does not provide the same utilities, in particular only a library documentation is available and no cloud service is accessible by the public. It is expected to change in the future, indeed Google has developed its own quantum computers that are expected to be made available over the cloud soon. Apart from this, both Qiskit and Cirq are good tools for writing and manipulating quantum circuits. They are both based on the Python programming language which allows a clean syntax of the code. Qiskit is organized in four distinct components which contains tools for doing different studies, as manipulating circuit, simulating it or building a noise model and doing error correction. A simple circuit is created by providing a quantum and a classical register, applying quantum gates on qubits and doing measurements. Then, the circuit is executed on a backend, which is accessed trough a provider. In Cirq, a circuit is a collection of moments, which consist on operations applied on qubits at the same time slice. It could be executed in different types of simulators. Probably, the biggest missing feature of Cirq is the ability to simulate noisy quantum circuits, which Qiskit instead does very well. In fact, in Cirq the modeling of noise is possible only through quantum channels added in the circuit. Instead, in Qiskit it is possible to build a noise model that consists on various types of error on each gate or classical readout errors.

In order to test the performance of the two software, we have implemented a circuit of N qubits, called MQC circuit, for studying the creation of a GHZ state, in both software. The fidelity bounds of such a state were calculated through multiple quantum coherences measurements [3]. In Qiskit, we executed the circuit in different backends for different values of N. Firstly, it was executed in ideal conditions in the Qasm Simulator, giving the expected result of fidelity equal to one. Then, it was executed in noisy conditions in the same simulator, where the noise model was based upon the parameters of the IBM Q Melbourne real device, and the results showed a decrement of the fidelity values as expected. After that, the circuit was executed in two different real devices. We find that, in IBM Q Yorktown, the maximum GHZ state acceptable [14] corresponds to $N = 4$ qubits. It is not a bad result if we think that this device supports only 5 qubits. In IBM Q Melbourne, that is a 14 qubits device, the limit is $N = 4$ qubits too. It means that only 4 qubits of the 14 available could be used to create a good entangled state. These results show how the public device available are beset with a certain amount of noise, that does not provide high fidelity gate operations. Therefore, at the moment they are unusable for complex tasks. In Cirq, we have created the 5-qubit MQC circuit and we have executed it in the Density Matrix Simulator in noiseless conditions. After we have checked that the results give a fidelity equal to one, we have added quantum channels for studying how they affect the goodness of the 5-qubit GHZ state. Firstly, Amplitude Damping channels and Depolarizing channels, with a fixed parameter γ, are added in two distinct circuits. In both cases, we have applied a different number of channels in function of the time slices, in order to reproduce noise that increases in time as in a real hardware. We have calculated the relaxation time T associated to the quantum noise

channels. The values obtained are not reliable, but they represent just an estimate. For the Amplitude Damping case, the maximum 5-qubit GHZ state acceptable is for $\gamma = 0.08$, which corresponds to $T_{AD} = 11.99\,\mu s$. For the Depolarizing, we have obtained $\gamma = 0.022$ and $T_D = 44.95\,\mu s$. Then, we have created a 5-qubit MQC circuit and even in this circuit we have added noise channels in function of the time slices. We have analyzed the state obtained by measuring the four corners of the density matrix of the state. As expected, for the Amplitude Damping channel the system state decays into the ground state, while for the Depolarizing channel it decays into a completely mixed state. Finally, we have implemented a circuit for creating a *Bell state*. The goodness of the state was analyzed through concurrence measurements and the results are consistent with the theory.

In conclusion, we have compared Qiskit and Cirq software and we have tested them for creating a N-qubit GHZ state. Cirq provides good tools for working with circuits for near term quantum computers, but Qiskit provides more tools and at the moment they are more developed too. Taking into account the current grade of development of the Cirq platform, it is necessary a growth of the Cirq community in order to make Cirq library more approachable. Maybe, it will happen when Google cloud service will be available for the public. In order to resolve more complex tasks, it is necessary that in the near future quantum computers provided by companies as IBM and Google will make available more qubits and more stable gate operations. Nowadays, a great amount of energy is spent by researchers and private companies to ensure that quantum computers will become a solid reality.

Bibliography

[1] Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information.* United Kingdom, Cambridge University Press, 2016.

[2] Benenti, G., Casati, G., and Strini, G. *Principles of quantum computation and information: Volume II: Basic Tools and Special Topics.* World Scientific Publishing Company, 2007.

[3] Ken X. Wei, Isaac Lauer, Srikanth Srinivasan, Neereja Sundaresan, Douglas T. McClure, David Toyli, David C. McKay, Jay M. Gambetta, and Sarah Sheldon. *Verifying Multipartite Entangled GHZ States via Multiple Quantum Coherences.* IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA, 15 May 2019. arXiv:1905.05720

[4] John Preskill. *Quantum Computing in the NISQ era and beyond.* Institute for Quantum Information and Matter and Walter Burke Institute for Theoretical Physics, California Institute of Technology, Pasadena CA 91125, USA, 30 July 2018.

[5] M. H. Devoret, A. Wallraff, and J. M. Martinis. *Superconducting Qubits: A Short Review.* Department of Applied Physics, Yale University, New Haven, CT 06520 and Department of Physics, University of California, Santa Barbara, CA 93106. 2 February 2008.

[6] John M. Martinis, Kevin Osborne. *Superconducting qubits and the physics of the josephson junctions.* 1National Institute of Standards and Technology, 325 Broadway, Boulder, CO 80305-3328, USA. 1999.

[7] Ryan LaRose. *Overview and Comparison of Gate Level Quantum Software Platforms.* Department of Computational Mathematics, Science, and Engineering, Michigan State University. 22 June 2018.

[8] Ryan LaRose. *Practical Quantum Computing with Cirq.* Department of Computational Mathematics, Science, and Engineering, Michigan State University. `https://quantumcomputingreport.com/scorecards/review-of-the-cirq-quantum-software-framework/`

[9] The Cirq Developers. *Cirq Documentation.* Release 0.5.0, 24 April 2019. `https://cirq.readthedocs.io/en/stable/index.html`

[10] The Qiskit Developers. *Qiskit API documentation.* Release 0.8.0, 9 March 2019. `https://qiskit.org/documentation/index.html`

[11] The Qiskit Developers. *Qiskit tutorials.* GitHub repository. 2019. `https://github.com/Qiskit/qiskit-tutorials`

[12] Scott Hill and William K. Wootters *Entanglement of a Pair of Quantum Bits.* Department of Physics, Williams College, Williamstown MA 01267 24 March 1997. arXiv:quant-ph/9703041

[13] David P. DiVincenzo *The Physical Implementation of Quantum Computation.* IBM T.J. Watson Research Center, Yorktown Heights, NY 10598 USA February 1, 2008. arXiv:quant-ph/0002077

[14] Otfried Guhne and Geza Toth *Entanglement detection.* 27 Feb 2009. arXiv:0811.2803