Grenoble INP – ENSIMAG & MSIAM

École Nationale Supérieur d'Informatique et de Mathématiques Appliquées de Grenoble
Master of Science in Industrial and Applied Mathematics

Master's thesis report

Introduction to Quantum computing

Suau Adrien

3rd year – ENSIMAG MMIS & MSIAM

August 2018

Centre Européen de Recherche et de
Formation Avancée en Calcul
Scientifique
31 057 Toulouse

Internship supervisor:
Gabriel Staffelbach
School tutor:
Christophe Picard

Abstract

Quantum computing is known to be able to solve efficiently problems that are hard for classical computers, such as prime factorisation. In this report, a study of different quantum algorithms is done to assess their applicability on two computational problems: solving sparse linear systems and detecting thermoacoustic instabilities in combustion chambers. Several limitations to the quantum algorithms studied have been identified and a method has been partially devised to apply quantum computing algorithms to the detection of thermoacoustic instabilities.

Résumé

Le calcul quantique est un domaine de recherche en pleine expension qui semble capable de résoudre efficacement certains problèmes compliqués pour un ordinateur classique. Ce document se concentre sur l'étude de potentielles applications du calcul quantique dans le domaine de la mécanique des fluides numérique. Dans ce document, une analyse de l'utilité de certains algorithmes quantiques est réalisée pour deux problèmes : la résolution de système linéaires creux et la détection d'instabilités thermo-accoustiques dans une chambre de combustion. L'analyse de ces deux problèmes a démontré un certain nombre de limitations inhérentes aux algorithmes étudiés et a donné lieu à l'élaboration partielle d'une méthode qui utilise un algorithme quantique pour cractériser les instabilités thermo-accoustiques.

Acknowledgements

I would like first to express my very great appreciation to all ENSIMAG and MSIAM teaching staff. I really enjoyed my first two years spent within ENSIMAG and my last year in the MSIAM master program. A special thank to Dr Christophe Picard, my ENSIMAG tutor for this internship, for his proofreading and advices.

I am particularly grateful for the assistance, advices and guidance given by Gabriel Staffelbach, my internship supervisor, all along the internship. He helped me a lot during these six months and made this internship particularly interesting and challenging.

I also wish to thank all the CSG team for the pleasant atmosphere in the office. Within the CSG team, I particularly want to thank Dr Isabelle d'Ast that helped me a lot with the IT environment and gave me precious advices during my research for a PhD.

I finally wish to acknowledge the help provided by Dr Jean-François Parmentier on the slides for the quantum computing meet-ups.

Table of Contents

List of Figures

List of Tables

Glossary

Quantum algorithm for linear systems of equations A quantum algorithm solving linear systems of equations, designed by Aram Harrow, Avinatan Hassidim, and Seth Lloyd. References: [1, 2]. 7, 17

Quantum Phase Estimation A quantum algorithm that estimates the eigenvalue associated to a known eigenvector of a unitary operator. References: [2, 3]. 14–16

Qubit 'Quantum bit': the quantum equivalent of the 'bit'. Used to store the data needed by the program being run. See A.1.1 for a detailled explanation. A.i

Variational Quantum Eigensolver A quantum iterative algorithm that computes the ground state eigenvalue of a unitary operator. References: [4]. 14–16

Acronyms

ALGO Algorithmique Parallèle – Parallel algorithmic. 2

API Application Programming Interface. 2, 7, 25

CERFACS Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique. 2, 11, 21, 27, 28

CES Consumer Electronics Show. 3

CFD Computational Fluid Dynamics. 2, 11

CNES Centre National d'Études Spatiales – National Centre for Space Studies. 2

COOP sCientific sOftware Operational Performances. 2

CSG Computer Support Group. 2

EDF Électricité De France – French electricity company. 2

FEM Finite Element Method. 18

GLOBC Modélisation du climat et de son changement global – Modeling of climate and its globalchange. 2
GUI Graphical User Interface. 2

HPC High Performance Computing. 2

ONERA Office National d'Etudes et de Recherches Aérospatiales – National Office for Aerospace Studies and Research. 2

PDE Partial Differential Equation. 18

QC Quantum Computing. 1, 2
QFT Quantum Fourier Transform. 23

SDK Software Development Kit. 7

Chapter 1

Introduction

1.1 General considerations

Quantum mechanics theory as we know it began in the mid-1920s with the work of Erwin Schrödinger [5], Werner Heisenberg [6], Max Born [7] and many other researchers. Using quantum mechanics, Yuri Manin and Paul Benioff [8] started to conceive a theory around QC (Quantum Computing), followed by Richard Feinman [9] and David Deutsch [10]. Until the second half of 1990, QC was considered only as a scientific curiosity: the existing quantum algorithms, although allowing an exponential speed-up in asymptotic complexity in some specific cases, could only solve artificial problems with no real interest apart from a theoretical point of view.

The vision of QC in the scientific community changed in the late 1990s, when Luv Grover and Peter W. Shor devised two of the most important algorithms of the field up to now: Grover's algorithm [11] and Shor's algorithm [12]. In its domain of application (function inversion) Grover's algorithm outperforms the best classical algorithm by providing a quadratic speed-up in the asymptotic complexity. On the other hand, Shor's algorithm is able to solve the problem of large integer factorisation with a polynomial asymptotic complexity (versus a sub-exponential one for the best known classical algorithm).

But at that time the hardware was not sufficiently developed to implement the algorithms, making them only of theoretical interest. Today, quantum hardware is still not developed enough for useful applications but the size (number of qubits) and the fiability (error-rates) of the quantum chips available started to increase very quickly in 2017, along with an increased disponibility to the public. The point when quantum computers will be able to solve some problems faster than classical ones (also called *quantum supremacy*) seems to be only a few years ahead.

This internship takes place at a time when huge actors like Google [13], Intel [14] or IBM [15] have decided to invest massively in quantum computers, mainly on the hardware side, and huge progress has been made: the maximum number of qubits on a single chip increased from 17 at the beginning of 2017 to 72 a year later. Alongside American companies, Europe is also investing in quantum technologies following a roadmap edited in 2016 [16] and summarised and updated at the end of 2017 [17]. Finally, quantum technologies have attracted European companies like Airbus [18] (searching for poten-

tial usages of quantum technologies) or Atos [19] (developing and selling a quantum simulator).

1.2 CERFACS and work team

CERFACS (Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique) is a private research laboratory created in 1987 and specialised in advanced methods for numerical simulations, especially simulations involving fluid mechanics. The company is managed by the representatives of its seven shareholders: Airbus; CNES (Centre National d'Études Spatiales – National Centre for Space Studies); EDF (Électricité De France – French electricity company); Météo-France, the French meteorological service; ONERA (Office National d'Etudes et de Recherches Aérospatiales – National Office for Aerospace Studies and Research); SAFRAN, a high-technology international group; and TOTAL, a multinational energy company.

The laboratory is divided into three research teams:

- ALGO (Algorithmique Parallèle – Parallel algorithmic)
- CFD (Computational Fluid Dynamics)
- GLOBC (Modélisation du climat et de son changement global – Modeling of climate and its globalchange)

A fourth team named CSG (Computer Support Group) is in charge of information technology and support. CSG also hosts the COOP (sCientific sOftware Operational Performances) team, in charge of code quality, API (Application Programming Interface) design, GUI (Graphical User Interface) design, HPC (High Performance Computing) and technology watch. The internship was supervised by Gabriel Staffelbach, a senior researcher of the COOP team.

1.3 Description of the work and outline of the document

The internship had several complementary goals: introducing the field of QC to CERFACS employees (PhD students, post-doctorates and researchers), determining if QC could be useful to CERFACS in their domain of activities (mostly CFD) and starting to implement some quantum algorithms.

The first step to complete these goals was to understand the concepts behind QC and how computations can be performed with quantum computers. The base concepts of QC are presented in Appendix A, along with a short introduction to quantum algorithms.

An important part of the internship consisted in a bibliographic work. This work can be divided into two parts:

- Building a list of the tools related to quantum computing and assessing their applicability and usefulness for the internship and for CERFACS.

- Understanding several quantum algorithms and determining if they could be applied to CERFACS field of expertise: CFD.

Chapter 2 summarises the first part (listing the technologies around quantum computing) by splitting it into three categories: the chips, the programming languages and the simulators.

Then, Chapter 3 recapitulates the research results obtained on the applicability of several quantum algorithms to computational problems in CFD.

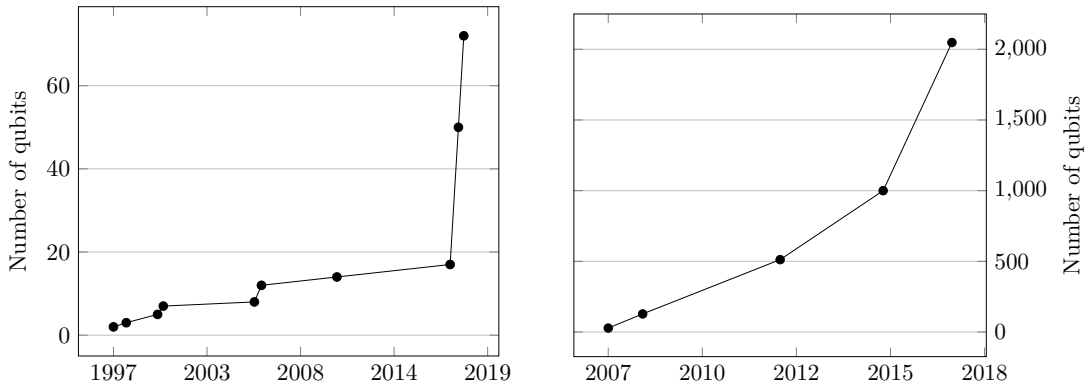
Finally, Chapter 4 outlines the different contributions made during the internship, both in terms of code and knowledge sharing.

Chapter 2

Existing quantum technologies

One of the early goals of the internship was to list as exhaustively as possible the technologies created and used in quantum computing. In this chapter, we use the word 'technology' for 'real quantum chip', 'quantum programming language' or 'quantum simulator'.

The first section of this chapter will list the existing physical implementations of quantum chips alongside with their characteristics. The second section will then introduce some quantum programming languages that are used to generate quantum machine instructions. The third and last section will finally discuss the purpose of quantum computer simulators and their development in a near future.



(a) Evolution in time of the number of qubits of *universal* quantum computers. Universal quantum computers are capable of running any quantum algorithm.

(b) Evolution in time of the number of qubits of *quantum annealing* computers. Quantum annealing computers are *not universal* and are specialised to a very restricted set of problems (quadratic unconstrained binary optimisation).

Figure 2.1 – Evolution in time of the number of qubits in quantum chips. Scales between universal qubits and quantum annealing qubits differ by 2 orders of magnitude.

2.1 Quantum chips

Before making a list of the existing quantum chips it is interesting to draw a time-line of the key dates in quantum computer physical implementation.

The first experimental demonstration of a working quantum computer took place in 1998. The quantum computer was composed of 2 qubits and ran Deutsch's algorithm [10]. In 2000, a research team at the Los Alamos National Laboratory constructed a 7-qubit quantum computer. The first quantum computer with more than 10 qubits was built in 2006 and had 12 qubits. One year later, DWave announced a 28-qubit quantum annealing processor. Since then the number of qubits in quantum computers has massively increased. Figure 2.1 summarises the evolution of the largest quantum chip (in term of number of qubits) along the time.

2.1.1 Existing quantum chips in July 2018

In July 2018, the major actors in quantum computer engineering are:

- **Intel** that announced during the 2018 CES (Consumer Electronics Show) a 49-qubit chip code-named Tangle Lake.
- **IBM** who built five quantum computers since 2017:
  - A 5-qubit computer made available on 24$^{\text{th}}$ January
  - Another 5-qubit computer made available on 25$^{\text{th}}$ September
  - A 16-qubit computer unveiled on 28$^{\text{th}}$ September
  - Two 20-qubit computers, made available to IBM partners at beginning of June
  - A 50-qubit computer that is still in test-phase and should be available to IBM partners in 2019

  Among these quantum chips, the two 5-qubit and the 16-qubit ones are publicly available via IBM's Q Experience website [15]. The 20-qubit chips are only available to IBM's commercial partners and the 50-qubit chip is not available yet.
- **Google** unveiled on 5$^{\text{th}}$ March 2018 its 72-qubit chip code-named Bristlecone.
- **DWave** which is specialised in the quantum annealing technology, has the D-Wave 2000Q chip which is composed of 2048 qubits.

Google announced in a blog post published on 18$^{\text{th}}$ July 2018 [20] that it plans to make its Bristlecone chip available in the cloud in the future. In July 2018, IBM's quantum computers are the only chips available to test the quantum codes produced during the internship as they are the only publicly accessible.

Table 2.1 summarises the biggest existing quantum chips in July 2018.

Company	Universal	Technology	Max. qubits	Publicly available
DWave	×	Superconducting	2048	on sale
Intel	✓	Superconducting	49	×
IBM	✓	Superconducting	50	in the future
Google	✓	Superconducting	72	×
Rigetti	✓	Superconducting	19	✓

Table 2.1 – Quantum chips owned by companies and known to public in July 2018. The column 'Technology' lists the physical qubits implementations used. This list is not exhaustive.

2.1.2 Physical implementation of quantum chips

Classical computer chips have not always been constructed with silicon transistors: in the early steps of computers, other technologies like electromechanical computers or vacuum tubes were used.

The same scheme holds for quantum computing. Currently, there are several implementations of qubits and quantum gates that are co-existing. The most widespread implementation, which is used in nearly all the 'industrial' quantum chips, is based on super-conducting qubits and on a quantum mechanical effect known as the Josephson effect. In addition to the Josephson junction, there exist plenty of different physical entities to represent a qubit, mostly used in research laboratories, such as photons, electrons or optical lattices.

Even if super-conducting qubits are currently the most used, there exist other alternatives that could be more interesting for a large-scale quantum computer such as:

1. Silicon-based qubits. Its advantage lies in the fact that silicon technology has already been used for decades in classical chips and companies like Intel or TSMC (Taiwan semiconductor founders) have gained an enormous amount of knowledge and expertise on this material.

2. Topological qubits. This implementation of a qubit uses a quasi-particle called anyon which is still highly theoretical. Topological qubits are considered as promising because they can theoretically achieve error-rates several order of magnitude better than the other technologies.

2.2 Programming languages

As for classical computers, quantum computers need to receive instructions from the developer in order to execute an algorithm. There exist two kinds of programming languages in quantum computing, like in classical computing:

- The **instruction sets** that are the quantum equivalent of classical assembly language like MIPS or x86-64.

- The **programming languages** that will be used to produce a code written in a given instruction set. These programming languages are the quantum equivalent of C++ or Python.

2.2.1 Quantum instruction sets

Quantum instruction sets are used to express a quantum algorithm in terms of machine instructions. Because of this, instructions sets are linked with the hardware and designed according to its key strengths and weaknesses.

In mid-2018, there are two main open quantum instruction sets: OpenQASM and Quil.

OpenQASM (Open Quantum ASsembly Language) [21] is the open-source instruction set designed by IBM for its quantum chips.

Quil [22] is the first quantum instruction set that introduced a model where quantum memory and classical memory are shared.

2.2.2 Quantum programming languages

There are plenty of quantum programming languages, Table 2.2 lists the main languages used to produce quantum machine instructions (i.e. a code in quantum assembly).

Language name	Free	Open-source	Inspired from	Supported by	Launched in
Q#	✓	×	C#	Microsoft	2017
QISKit	✓	✓	Python 3	IBM and Open-source	2017
Quipper	✓	✓	Haskell	Researchers	2013
Cirq	✓	✓	Python 3	Google	2018

Table 2.2 – Main quantum programming languages in July 2018. This list is not exhaustive.

In Table 2.2, each programming language uses a different paradigm to produce and run quantum code and has a different level of maturity.

- **Q#** [23], created by Microsoft. It can be integrated into C# or F# code. Q# code works like a black box: the user/developer does not have access to the low-level representation of its code (i.e. the translation of the code in quantum assembly). Everything is abstracted in order to make Q# programming a 'high-level' programming language.

  Q# comes with a quantum library developed by Microsoft. This library implements most of the basic quantum algorithms such as the Quantum Fourier Transform or Shor's algorithm.

- **QISKit** [24] is an open-source project launched by IBM. The QISKit project is composed of:
  - The **OpenQASM** language specification [21].
  - Several **SDKs** [25, 26, 27] allowing developers to generate OpenQASM code with their favourite programming language. The available languages in July 2018 are Python 3, Swift and JavaScript.
  - An **API** to submit quantum programs to IBM's quantum chips [28] that are in the cloud.

  The QISKit project provides a way to generate OpenQASM code from Python, Swift and JavaScript.
- **Quipper** [29] is designed to be a scalable programming language for quantum computing. Quipper is based on Haskell and uses the functional paradigm.
- **Cirq** [20] was released in July 2018. It is a Python 3 framework developed by Google that specifically targets Bristlecone, the quantum chip unveiled several months ago by Google.

2.3 Quantum simulators

Quantum simulators are the final brick in quantum computing. Just as their classical counterparts, quantum simulators are software designed to run on classical architectures that aim at simulating the behaviour of a quantum computer. The same principle exists in classical computing with the Intel Software Development Emulator that emulates the behaviour of new Intel chips without requiring access to one of those chips.

Simulators are crucial in quantum computing and address several major issues of the current quantum chips:

1. Real quantum hardware is not readily accessible to everyone.
2. Real quantum hardware is *noisy* (see 2.3.3), has resiliency issues.
3. Real quantum hardware is currently limited to a small number of qubits.

Each issue is explained in details in the sections 2.3.1, 2.3.2, 2.3.3.

2.3.1 Limited hardware accessibility

The first main issue with quantum chips is that not everyone is able to use them. In fact, most of the quantum chips are owned by private companies and only accessible to 'partners' that registered and paid to have access to those chips. The only quantum hardware currently available to everyone is IBM's chips, but strong conditions on intellectual property applies: to run the code on a IBM chip, you must give IBM all the rights on it (right to patent, copy, share, modify, etc.).

Quantum simulators settle this accessibility issue as they are just a classical software that is most of the time freely shared on specialised websites like GitLab or GitHub. Some simulators are not freely available and some are even linked with a specialised hardware that should be bought with the software. Nevertheless, there exist plenty of open-source quantum simulators and a list can be found in [30].

According to the documentation of the simulators, simulating at least 20 qubits should be accessible to everyone with a 'normal' computer. Most of the current quantum simulators are limited by the quantity of available RAM because they need to store a quantum state which is composed of 2^n complex numbers, n being the number of simulated qubits. Knowing that, we can easily compute the number of simulable qubits as a function of the available quantity of RAM. Function f computes for a given quantity of RAM M (in bytes) the number of simulable qubits.

$$f(M) = \log_2\left(\frac{M}{\text{sizeof(complex)}}\right) = \log_2\left(\frac{M}{8}\right)$$

For a computer equipped with 4GiB of RAM, $f(4 \times 2^{30}) = 29$ so 29 qubits are simulable if we neglect the quantity of RAM needed for the operating system and the other applications.

2.3.2 Hardware errors

Another big issue in quantum computing is the errors that appear during the computations. Errors during quantum computations can be caused by:

1. Imperfect implementation of quantum operations: each application of a quantum operation has a probability to fail and to introduce an error in the computation.

   On IBM's quantum chips, operations involving a unique qubit have an error rate in the order of 10^{-3}, operations on 2 qubits have an error rate of approximately $3.10^{-2}(= 3\%)$ and the error rate in measurement is between 2.10^{-2} and 11.10^{-2}, i.e. between 2% and 11%.

2. Interactions between the qubits and an external system that is not under control. The probability that no interaction occurs is given by an exponential law of parameter $\lambda = \frac{1}{T}$, where T is called the 'coherence time'. After a duration T, an interaction between the concerned qubit and an external system (i.e. an error) has occurred with a probability of $1 - e^{-1} \approx 0.63$. Coherence times of IBM's qubits are all below $100\mu s$ for the publicly available chips.

Because of these high error rates, actual quantum chips are only capable of running toy programs and are limited by coherence time and gate errors.

This problem is circumvented by using quantum simulators which are by definition 'perfect' in the sense that they do not suffer from the errors caused by physical phenomenon such as decoherence or imperfect operations.

If we take as example a quantum algorithm that would require $t = 1$ second to run on a real quantum computer composed of n qubits, taking $T = 100\mu s$ as the coherence

time of reference and neglecting gate errors, the probability for one qubit to executes without error is $e^{-\lambda t} = e^{-\frac{t}{T}} = e^{-10000} \approx 10^{-4343}$. The quantum program needing n qubits, the final probability that the whole program runs without error is $e^{-10000n}$. On the other hand, for the same quantum program, a quantum simulator would give an error-free result with probability 1 (neglecting errors that may happen on the classical hardware).

2.3.3 Limitation on the number of qubits

In this section, the distinction between *noisy* and *noise-less* qubits is crucial: a noisy qubit has imperfect operations (i.e. errors can occur when performing an operation on the qubit) and a finite coherence time whereas a noise-less qubit has perfect operations and an infinite coherence time.

Thanks to QECC (Quantum Error Correction Codes) it is possible to group *multiple* noisy qubits together to make them behave as a *unique* 'less-noisy' qubit (see [31]). When using QECC, it is convenient to introduce the notions of *physical qubits* and *logical qubits*:

- A *physical qubit* represents the 'raw' qubit, without QECC.
- A *logical qubit* is *something* that behave as a qubit. Examples of logical qubit are:
  - A physical qubit (it behaves like a qubit, so according to the definition it is a logical qubit).
  - A group a physical qubits aggregated with a QECC is a logical qubit. In this case, the logical qubit will have better caracteristics (longer coherence time, lower gate error-rate, etc.) than the physical qubits composing it.

With these definitions, we can clearly state the difference between a quantum simulator and a real quantum chip when speaking of their qubits:

- Quantum simulators simulate noise-less qubits, i.e. perfect qubits, without errors.
- Quantum computers implement noisy physical qubits, i.e. qubits with a non-negligible error-rate. A quantum computer can also use QECC to implement *noisy* logical qubits, with lower error-rates and greater coherence time than the noisy physical qubits. In practice, QECC can introduce huge factors in the number of needed qubits. See [32] for more details.

Speaking of the number of qubits, the existing hardware is still limited to a very small number of *noisy* qubits. The maximum number of noisy qubits on a quantum chip is detained by Google with the 72 qubits of its Bristlecone chips, but for a *publicly available quantum chip*, the maximum number of noisy qubits is only 16.

On the other hand freely accessible simulators are able to simulate at least 28 qubits.

Table 2.3 gives the characteristics of 3 quantum simulators.

Company	Free	Max. qubits free	Max. qubits non-free
Atos	×	×	40
Microsoft	✓	30	40
IBM	✓	30	×

Table 2.3 – List of quantum programming simulators in July 2018. This list is not exhaustive. The given numbers of qubits are the one advertised by the companies. The real number of qubits that are simulable in a reasonable time may be lower in practice.

Chapter 3

Quantum algorithms applied to scientific computing problems

Using the quantum programming languages presented in Section 2.2 and the formalism explained in Appendix A, quantum algorithms can be devised and implemented. Several quantum algorithms have already been published, and the main goal of the internship was to investigate the potential usefulness of these algorithms (and more generally quantum computing) in the field of scientific computing. A deeper study should be carried out on problems that are of particular interest for CERFACS or some of its shareholders.

During this internship, two practical problems that can be solved by classical computing were studied.

3.1 Detecting acoustic instabilities in combustion chambers

CERFACS is one of the leading research laboratory in the field of combustion. Jointly with 'IFP Energies nouvelles' (a public French research laboratory), CERFACS maintains and improves the AVBP solver, one of the most advanced solver in combustion.

One of the major issues that may arise in the design of combustion chambers is instabilities. These instabilities may appear in a combustion chamber can be classified in three categories: thermoacoustic combustion instabilities; static instabilities or flame blow-off and intrinsic flame instabilities. However thermoacoustic instabilities can destroy the engine in a few minutes and are complex to predict before the end of the design phase, that is why the study of instabilities in combustion chambers is often reduced to the study of thermoacoustic instabilities.

Thermoacoustic instabilities are characterised by pressure variations in the combustion chamber that have a well defined frequency, most of the time between 100 Hz and several thousands Hertz. These instabilities may have amplitudes which are high enough to damage the combustion system and in extreme cases provoke an explosion of the system. Such instabilities have already caused the explosion of several rocket engines during their test phase, such as for example the Saturne-F1 engine.

Because of the cost of these equipments and the difficulty to create experimental setups to find the instabilities, companies use CFD to compute numerically the potential

instabilities and their characteristics, in order to avoid them in the design of their rocket-engine.

Based on [33], section 3.1.1 will introduce succinctly the equations modeling the instability phenomenon and the discretisation used. Section 3.1.2 will then explain briefly the algorithms used to solve the differential equation governing the instability phenomenon. Finally, an analysis of the potentially interesting quantum algorithms for this problem is presented in section 3.1.3. The full development for sections 3.1.1 and 3.1.2 can be found in [33].

3.1.1 Numerical model to compute thermoacoustic instabilities

According to [33], the pressure in a combustion chamber is governed by the equation

$$\frac{\partial^2 p_1(\vec{x},t)}{\partial t^2} - \nabla . c_0^2(\vec{x})\,\vec{\nabla} p_1(\vec{x},t) = (\gamma-1)\frac{\partial}{\partial t} q_1(\vec{x},t), \tag{3.1}$$

with:

- $\rho_0(\vec{x})$ the density of the fluid in the combustion chamber,
- γ the compressibility coefficient of the fluid under consideration,
- $p(\vec{x},t) = p_0(\vec{x}) + p_1(\vec{x},t)$ where p_0 is the (constant) mean value of the pressure and p_1 the non-constant part of the pressure,
- $c_0^2(\vec{x}) = \gamma p_0/\rho_0(\vec{x})$,
- $q_1(\vec{x},t)$ the unsteady part of the heat emission rate,

We also assume that the solution in the frequential domain have harmonic oscillation: $p_1(\vec{x},t) = \hat{p}(\vec{x})\, e^{-i\omega t}$. In this specific case, thermoacoustic instabilities appear if ω_i, the imaginary part of ω, is non-negative (i.e. $\omega_i > 0$).

After some additional transformations, discretising the equation 3.1 by using the finite volume method gives the linear system

$$\left(A + \omega B + \omega^2 M\right)\vec{p} = M\vec{n}\vec{r}^{T}\vec{p},$$

which can be re-organised to

$$\left(M^{-1}A + \omega M^{-1}B - \vec{n}\vec{r}^{T}\right)\vec{p} = -\omega^2\vec{p} \tag{3.2}$$

with:

1. A a sparse symmetric matrix
2. B a diagonal matrix of imaginary numbers
3. M a diagonal matrix of real numbers

4. $\vec{n}\vec{r}^T$ a bloc sparse matrix

Finding the eigenpairs $(\omega, \vec{p})$ of equation 3.2 gives us all the information we need to compute the discrete repartition of the pressure field, i.e. to characterise the instabilities thanks to the relation in equation 3.3.

$$\begin{aligned} \vec{p_1} &= \Re\left(\vec{p}e^{-i\omega t}\right) \\ &= \Re\left(|\vec{p}|e^{i\vec{\phi}}e^{\omega_i t}e^{i\omega_r t}\right) \\ &= |\vec{p}|e^{\omega_i t}\cos(\vec{\phi} - \omega_r t) \end{aligned} \tag{3.3}$$

More precisely, the eigenvector $\vec{p} = |\vec{p}|e^{i\vec{\phi}}$ gives information on the discrete repartition of the pressure field and its amplitude. The imaginary part of the eigenvalue ω also gives information on the amplitudes of the pressure field and its evolution in time, whereas the real part represents the frequency of the pressure variations.

More precisely, we only need to find the eigenpairs that correspond to modes with a frequency between 100 Hz and several thousands of hertz as thermoacoustic instabilities frequencies are usually within this range.

3.1.2 Classical algorithms used to solve Helmholtz equation

Section 3.1.1 reformulated the problem of characterising the thermoacoustic instabilities in a combustion chamber as finding the eigenvalues and associated eigenvectors of a discretised operator. The required size for the discretised eigenvalue problem depends on the complexity of the combustion chamber topology and on the desired precision. For industrial combustion chambers, the complexity of the chamber topology and the desired accuracy impose a very fine discretisation step, which translates to a huge number of unknowns in the linear system 3.2 and a huge matrix size.

Because of computing capabilities limitations, we are currently limited to discretisations leading to a maximum of 20 million unknowns. In order to fully compute the instabilities, a discretisation using at least 100 million unknowns is needed but is computationally intractable at the moment.

Eigenpair problems with such a size cannot be solved with a direct numerical method, mainly because storing all the eigenvectors would require an enormous amount of memory (more than 2.8 Pio in single precision, nearly 5.7 Pio in double precision). Instead of a direct method, an iterative method needs to be considered.

Today, the most efficient iterative methods known for eigenpair problems are the Jacobi-Davidson method and the Arnoldi method. These methods are explained in details in [34, Chapter 6].

3.1.3 Quantum algorithms

Investigations on quantum algorithms that could replace the classical methods used nowadays have been carried out during the internship. There exist several quantum

algorithms related to eigenvalues and eigenvectors. The most cited and used one is the Quantum Phase Estimation (QPE) algorithm. Most of the other eigenvalue-related algorithms are based on QPE, except for the Variational Quantum Eigensolver (VQE). Both algorithms are presented in the following sections.

Quantum Phase Estimation

The QPE [35] algorithm is the first quantum algorithm to deal with eigenvalues and eigenvectors. It was introduced by Alexei Kitaev in 1995 as a generalisation of Shor's algorithm (that can be seen as a special case of the QPE algorithm).

Let U be a unitary matrix, i.e. $UU^\dagger = U^\dagger U = I$. Let $\vec{u_j}$ be an eigenvector of U. As U is unitary, $\vec{u_j}$ is of unit norm and the eigenvalue λ_j associated to the eigenvector $\vec{u_j}$ has necessarily a norm of 1.

As λ_j has a norm of 1, it can be expressed as $\lambda_j = e^{2\pi i\theta_j}$. In this particular case, the phase of the eigenvalue λ_j is enough to fully recover the eigenvalue. The QPE algorithm, as its name indicates, estimates the phase θ_j of the eigenvalue λ_j associated to a known eigenvector $\vec{u_j}$ of a given unitary matrix U.

In other words, given a unitary matrix U and one of its eigenvectors $\vec{u_j}$ (represented as the quantum state $|u_j\rangle$), the QPE algorithm estimates on n bits θ in the expression

$$U\,|u_j\rangle = e^{2\pi i\theta}\,|u_j\rangle\,.$$

The QPE being a probabilistic algorithm, it can return a bad result. If the value of θ can be exactly encoded on n bits, the QPE algorithm succeed with probability 1. Else, the algorithm succeed with a probability of *at least* $\frac{4}{\pi^2} \approx 0.41$ [3]. At the end of the algorithm, the state $|u_j\rangle$ is left unchanged and the best n-bit approximation of θ, $\tilde{\theta}$, is encoded on n quantum bits in the state $|\tilde{\theta}\rangle$.

The fact that the eigenvector $\vec{u_j}$ needs to be known in advance seems to be a major drawback of the algorithm but in fact the QPE can still estimates the eigenvalues of unknown eigenvectors. If instead of using an eigenvector $\vec{u_j}$ as input to the QPE algorithm we use a random vector $\vec{b} = \sum_i \beta_i \vec{u_i}$ (the decomposition is always possible as the eigenvectors form a basis), the QPE will act linearly on $\vec{b}$ and will construct the state $\sum_i \beta_i |\tilde{\theta_i}\rangle\,|u_i\rangle$ from the input state $|b\rangle = \sum_i \beta_i\,|u_j\rangle$.

Variational Quantum Eigensolver

The VQE [4] is a quantum algorithm that needs both a quantum computer *and* a classical computer to execute. The algorithm is particularly suited to computational chemistry because it aims at estimating the lowest eigenvalue (ground state) of a known Hermitian matrix along with its associated eigenvector.

The algorithm is based on two facts:

1. The Rayleigh-Ritz quotient

$$\frac{\langle\psi|\,\mathcal{H}\,|\psi\rangle}{\langle\psi|\psi\rangle}$$

reach its minimum value λ_{min} when $|\psi\rangle$ is $|u_{min}\rangle$, the eigenvector associated with the lowest eigenvalue λ_{min} of $\mathcal{H}$.

2. The quantity $\langle\psi|\,\mathcal{H}\,|\psi\rangle$ is efficiently computable on a quantum computer.

The idea behind the VQE algorithm is to apply a classical gradient-free optimisation algorithm (such as the basin-hopping algorithm) to minimise the quantity

$$\frac{\langle\psi_\Theta|\,\mathcal{H}\,|\psi_\Theta\rangle}{\langle\psi_\Theta|\psi_\Theta\rangle}$$

(which can be computed efficiently on a quantum computer) with respect to a set of parameters Θ that parametrise the state $|\psi_\Theta\rangle$. By doing so, the minimisation algorithm used will eventually compute a set of parameters Θ and the associated state $|\psi_\Theta\rangle$ that approximate the minimum of the Rayleigh-Ritz quotient, i.e. the ground state of $\mathcal{H}$.

3.1.4 Applicability of the quantum algorithms

The next step after understanding the two previous algorithms was to assess their usefulness in the computational problem of characterising the instabilities in combustion chambers (presented in 3.1).

Before analysing the applicability of the QPE and VQE algorithms, a quick summary of the problem might be useful.

We want to find the eigenpairs of a system $C\vec{p} = \lambda\vec{p}$ with C a non-hermitian matrix defined in 3.4 and $\lambda = -\omega^2$.

$$C = M^{-1}A + \omega M^{-1}B - \vec{n}\vec{r}^T \tag{3.4}$$

In the particular case summarised above, the VQE algorithm seems to be unsuited for at least two reasons. First, the matrix C is not hermitian, which is a necessary condition for the VQE algorithm. Secondly, the VQE algorithm only estimates the ground-state of the hermitian matrix given, but solving our problem requires to know all the eigenpairs satisfying a given criterion.

The first issue might be mitigated by adapting the initial problem to the algorithm requirements. For example, a non-hermitian matrix can be transformed into a hermitian matrix by using some mathematical tricks. One example of transformation would be to consider the matrix

$$C_h = \begin{pmatrix} 0 & C^\dagger \\ C & 0 \end{pmatrix}, \tag{3.5}$$

which is hermitian, instead of C. The transformation used needs to verify some properties in order to be interesting for our case:

- The transformation should not cost too much to apply.
- The transformed problem should be efficiently solvable by the VQE algorithm.

- The solution of the initial problem (the eigenpairs of C) should be computable by using the solution of the transformed problem (the eigenpairs of C_h). In other words, the transformation should be easy to invert.

Searching for a transformation satisfying the conditions above has not been done within the internship and is a possible working track for further research.

The QPE algorithm is more versatile than the VQE algorithm in the sense that it is not limited to the ground-state and can produce a quantum state involving all the eigenpairs of the considered matrix. But the matrix given to the QPE algorithm needs to be unitary, which is not the case in the eigenvalue problem 3.2.

As for the VQE algorithm, workarounds may exist. One possible workaround would be to transform our general eigenvalue problem into a hermitian eigenvalue problem (using a transformation that satisfies the properties discussed in the previous part) and then re-transform this hermitian eigenvalue problem into a unitary eigenvalue problem.

$$C_h \vec{p_h} = \lambda \vec{p_h}, \tag{3.6}$$

Let 3.6 be the hermitian eigenvalue problem, with C_h the hermitian matrix obtained by transforming C and p_h the transformed solution vector. Applying the transformation $A \mapsto e^{iA}$ to C_h gives us a matrix $C_u = e^{iC_h}$ with the following properties:

1. C_u is unitary.
2. If $\{(\lambda_j, u_j)\}_j$ are the eigenpairs of C_h then $\left\{\left(e^{i\lambda_j}, u_j\right)\right\}_j$ are the eigenpairs of C_u.

Supposing that the transformation from the initial problem to the hermitian one exists, we end up with an eigenvalue problem that is suitable for the QPE algorithm.

Independently of the quantum algorithm used, one more issue needs to be analysed: the eigenvectors computed by both algorithms are stored in the *coefficients* of a quantum state. This means that we cannot have access to the full eigenvector: we only have access to an expectation value of the form $\langle u_j | M | u_j \rangle$ where M is a linear operator. This data is likely to be insufficient to compute the pressure vector field by using equation 3.3.

One solution would be to devise a new algorithm that is able to give some information on the instabilities just by using a reduced number n of expectation values of the form $\langle u_j | M_i | u_j \rangle$ for $0 \leqslant i \leqslant n$. The number of expectation values needed n will be crucial for the complexity of this new algorithm, because as quantum states cannot be copied, n will represent the number of times the QPE algorithm needs to be repeated.

3.2 Solving linear systems of equations

In 2008, Harrow, Hassidim and Lloyd introduced the 'quantum algorithm for linear systems of equations', also known as 'HHL algorithm', in [1]. This algorithm is considered as a major breakthrough in the field of quantum computing for several reasons.

First, new quantum algorithms that have direct implications on real-world problems are rare: the only algorithms known before HHL were Grover's algorithm [11] and Shor's algorithm [12], both of them dating from the end of the 20^{th} century.

Secondly, solving linear systems of equations is a widespread problem which used in many fields of scientific computing. By solving this problem exponentially faster than classical methods, the HHL algorithm may help to improve the asymptotical complexity of other algorithms that need to solve linear systems of equations. Some examples of applications are mentioned in section 3.2.2.

3.2.1 The HHL algorithm

The HHL algorithm solves a linear system of equations

$$\mathcal{H}\vec{x} = \vec{b} \tag{3.7}$$

with $\mathcal{H}$ a s-sparse hermitian matrix of size $N \times N$ and $\vec{x}$ and $\vec{b}$ two vectors of N complex numbers[1]. The major advantage of the HHL algorithm over its classical counterparts is its asymptotic time complexity of

$$\tilde{\mathcal{O}}\left(\log(N)s^2\kappa^2/\epsilon\right) \tag{3.8}$$

whereas the best known classical method to solve linear systems (conjugate gradient method) has a time complexity of

$$\mathcal{O}\left(Ns\sqrt{\kappa}\log\left(\frac{1}{\epsilon}\right)\right). \tag{3.9}$$

In the equations above, κ is the condition number of $\mathcal{H}$, ϵ is the desired precision and $\tilde{\mathcal{O}}$ is the equivalent of the big-$\mathcal{O}$ notation but suppressing the more slowly-growing terms (see [1] for a complete description).

At the end of the algorithm, the vector $\frac{\vec{x}}{||\vec{x}||}$ is encoded in the coefficients of a quantum state $|x\rangle$. The fact that the solution is encoded in the coefficients is also a limitation of the HHL algorithm: once the HHL algorithm finished, the solution vector $\vec{x}$ cannot be completely recovered because of the property of quantum measurement that collapse the measured state. Even if the full vector is impossible to recover, expectation values like $\vec{x}^T M\vec{x}$ can be measured and can provide information on normalisation, moments, etc.

3.2.2 Possible applications of HHL algorithm

As explained in section 3.2.1, the HHL algorithm solves linear systems of equations exponentially faster (with respect to the problem size) than classical algorithms. Since its publication in 2008, the HHL algorithm has been the subject of many scientific papers and several applications in the field of scientific computing have been found. In each of the following sections, a quantum algorithm that uses the HHL algorithm as a subroutine and solve a scientific problem is presented.

[1] Other conditions apply, such as the necessity for the singular values of $\mathcal{H}$ to be between $1/\kappa$ and 1. See [1] for a summary of the pre- and post-conditions of the algorithm.

Finite Element Method

The FEM (Finite Element Method) is used to approximately solve a boundary value problem given as a PDE (Partial Differential Equation) with constraints on a defined boundary. Solving a PDE with the finite element method boils down to solve a sparse linear system of equations obtained after discretising the weak formulation of the original PDE over a given mesh.

The utilisation of the HHL algorithm instead of a classical solver in the FEM has been studied in [36] by Montanaro and Pallister. Table 3.1 summarise the theoretical asymptotical complexities they obtained depending on whether or not a pre-conditioner is used.

Algorithm	No preconditioning	Optimal preconditioning
Classical	$\tilde{\mathcal{O}}\left(\left(\lvert u\rvert_2/\epsilon\right)^{(d+1)/2}\right)$	$\tilde{\mathcal{O}}\left(\left(\lvert u\rvert_2/\epsilon\right)^{d/2}\right)$
Quantum	$\tilde{\mathcal{O}}\left(\lvert\lvert u\rvert\rvert\lvert u\rvert_2^2/\epsilon^3 + \lvert\lvert u\rvert\rvert_1\lvert u\rvert_2/\epsilon^2\right)$	$\tilde{\mathcal{O}}\left(\lvert\lvert u\rvert\rvert_1/\epsilon\right)$

Table 3.1 – FEM algorithms complexities according to [36]. u is the solution vector, $||.||$ is the classical Euclidean norm, $||.||_l$ is the Sobolev l-norm and $|.|_l$ is the Sobolev l-seminorm.

In the ideal case where the optimal pre-conditioner is available, the quantum algorithm achieve an exponential speed-up on the number of dimensions d. If the number of dimensions d is fixed, the quantum algorithm provides *at most* a polynomial speed-up depending on the value of d.

The asymptotical time complexity of the algorithms also depend on the 'smoothness' of the solution u. The classical algorithm time complexity depends on the amplitude of the second derivative of the solution, whereas the quantum algorithm time complexity is computed by using the amplitude of the first derivative of the solution. This means that if the solution is not 'smooth' enough, the quantum algorithm can provide a substantial speed-up over the classical method.

Quantum machine learning

Machine learning is a field of computer science that is currently exploding in terms of research paper published and obtained results. One of the principal problem that machine learning aims at solving is classification problem. Depending on the availability of labelled data, the classification problem has two different formulations:

- If labelled data is available, then we are performing 'supervised learning' and the problem is to be able to classify a given object within a given set of classes that are characterised by the labelled data.
- If the data we have is not labelled, then we are performing 'unsupervised learning'. In this case, the problem is to find the labels that best fit the provided data.

There exist plenty of classical algorithms in both supervised and unsupervised machine learning. The case of quantum algorithms applied to supervised or unsupervised machine learning has been investigated in [37]. In this papers, Lloyd, Mohseni and Rebentrost introduce and analyse a new quantum algorithm that mimics Lloyd's algorithm [38] (the author of this algorithm *is not* one of the authors of [37] even if the have the same name).

According to [37], the quantum version of Lloyd's algorithm for finding a local minima of the k-means problem has an asymptotic complexity of $\mathcal{O}\left(k \log\left(kMN\right)/\epsilon\right)$ with k the number of classes, M the number of points in the data set, N the dimension of the search space (i.e. the number of coordinates needed to represent one data point) and ϵ the desired precision. This complexity can be lowered down to $\mathcal{O}\left(\log\left(kMN\right)/\epsilon\right)$ if the k clusters representing the different classes are 'relatively well separated'. In comparison, Lloyd's algorithm on a classical computer has an asymptotic time complexity of $\mathcal{O}(kMNI)$ where I, the number of iterations needed, may scale as $2^{\Omega(\sqrt{n})}$ in the worst case.

Linear differential equations

Linear differential equations are omnipresent in the fields of science and engineering. First-order linear differential equations are even more important as any high-order linear differential equation can be reformulated as a first-order linear differential equation.

In its general form, a first order ordinary differential equation can be written as

$$\frac{\partial}{\partial t}\vec{x}(t) = A(t)\vec{x}(t) + \vec{b}(t). \tag{3.10}$$

If A and $\vec{b}$ are independent of time and A is a s-sparse hamiltonian matrix, a steady state can be found by solving $A\vec{x} = -\vec{b}$ with the HHL algorithm. But when A, $\vec{b}$ or both are time-dependent, the steady state is not trivially computable. In [39], Berry presents a quantum algorithm to solve the time-dependent first-order ordinary differential equation presented in 3.10 with the constraint that A is s-sparse.

The algorithm presented in [39] achieves an asymptotical time complexity[2] of

$$\tilde{\mathcal{O}}\left(\log(N_x) s^{9/2} \left(||A||\Delta t\right)^2 \epsilon^{-1}\right), \tag{3.11}$$

with N_x the number of spatial discretisation points, s the sparsity of the matrix A, $\Delta t = t - t_0$ the total time interval over which the differential equation is to be solved and ϵ the desired accuracy.

With a classical algorithm, the complexity of solving 3.10 should be at least $\mathcal{O}(N_x)$, i.e. the time needed to compute or read the matrix $A(t_0)$ or the vector $\vec{b}(t_0)$ for a given time t_0. So the quantum algorithm achieve an exponential speed-up with respect to the spatial discretisation size N_x.

[2]The expression of the asymptotical time complexity has been simplified to highlight the most important points and to avoid re-explaining the paper. See [39, p. 12] for the complete expression.

Chapter 4

Internship contributions

The internship was not limited to theoretical research but also included a more practical aspect: the implementation of quantum algorithms that are of interest for some of CERFACS shareholders. Another key purpose of the internship was to introduce quantum computing to CERFACS's researchers.

Sections 4.1 and 4.2 present the different implementations performed during the internship. More specifically, section 4.1 expose the different quantum algorithms successfully implemented and section 4.2 introduce two auxiliary tools dealing with quantum programs: `qasm2image` and `qasm2error`.

Section 4.3 is dedicated to the two quantum computing meet-ups that were organised at CERFACS. Each of these meet-ups consisted in a presentation related to quantum computing and was followed by an exchange on the presentation and more generally on quantum computing.

4.1 Implementation of quantum algorithms

Table 2.2 summarises the programming languages that were available during the internship (except for Cirq that has only been published at the end of the internship). Among these languages, QISKit was chosen for the implementations of quantum algorithms for multiple reasons:

1. The language is open-source and open to contributions.

2. The lead developers maintaining the QISKit library are very active and improve the library continually.

3. QISKit is highly oriented toward IBM's quantum chips, which were the targeted chips at the beginning of the internship.

4.1.1 Early developments

The internship started with a period of introduction to the field of quantum computing as I did not have any prior knowledge on this subject. During this period, my time was

split between reading introductory books ([40] and [41]) and experimenting with basic quantum programs.

The first experiments were performed on IBM's website with the online editor made available. A screen-shot of IBM's online editor is shown in Figure 4.1.

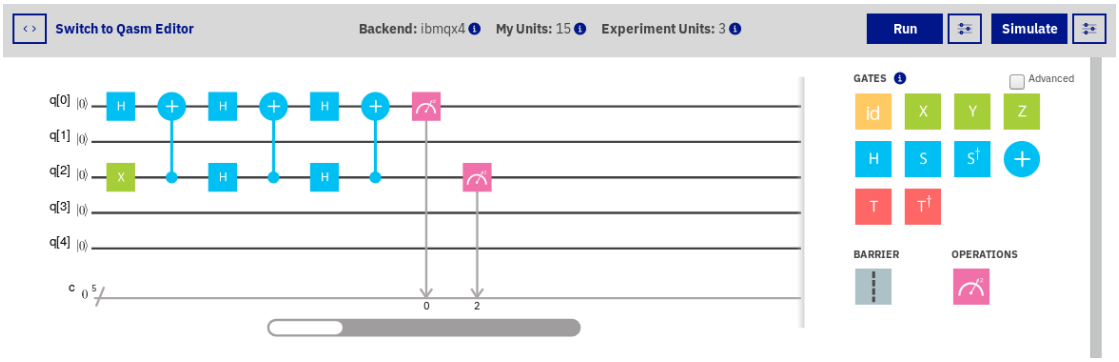


Figure 4.1 – Online editor on IBM's website [15]. The quantum circuit is represented on the left of the editors and quantum gates are distributed on the circuit by using drag-and-drop on the gates presented on the right of the editor.

The editor is convenient for testing small quantum programs and reasoning about the output, but becomes quickly impractical when dealing with medium-sized or large quantum programs. The QISKit library replaced the online editor very quickly for its convenience, even when dealing with large circuits. Moreover, as QISKit is a Python 3 library it did not require a long adaptation time.

After successfully implementing some 'toy' algorithms like Deutsch-Jozsa algorithm with the QISKit library, a more complex quantum algorithm was needed to assert the capabilities of QISKit. Shor's algorithm was chosen for several reasons. First, there are several research papers that investigate Shor's algorithm implementation and that explain in details the authors' implementation. Secondly, the algorithm is not trivial and was a good candidate to test the limits of the QISKit library. Finally, the introductory book [40] used at the beginning of the internship had a whole chapter dedicated to the practical implementation of subroutines used in Shor's algorithm.

4.1.2 Shor's algorithm

Shor's algorithm requires a quantum subroutine that performs a modular exponentiation. In other words, Shor's algorithm needs a quantum subroutine that performs the operation

$$|x\rangle \longrightarrow |a^x \mathrm{mod} N\rangle\,,$$

with a a given constant and $N = 2^n$ the number of superposed states considered. This operation is relatively easy to implement on a classical computer but becomes harder to implement efficiently on a quantum computer. In order to implement the modular exponentiation subroutine, base subroutines like addition or subtraction are needed.

There exist several quantum algorithms to perform an integer addition, each of them having advantages and drawbacks. Three different quantum addition algorithms have been implemented with QISKit:

1. The 'Conventional Quantum Plain adder' explained in [42].
2. The 'Ripple-Carry adder' introduced in [43].
3. The 'Quantum Fourier adder' presented in [44]

The last algorithm uses the QFT (Quantum Fourier Transform) as a subroutine to compute the sum of two integers. In order to complete its implementation, the QFT algorithm has also been implemented.

The next step to build the quantum modular exponentiator was to use the quantum adder to build a quantum modular adder. One such implementation is presented in [45]. The implementation of the quantum modular adder presented in [45] revealed a missing feature in QISKit: the library lacks the possibility to transform a given quantum gate U into its controlled-equivalent $c-U$. The implementation of the controlled gate could theoretically be done 'by hand', by re-writing the U gate and modifying its implementation to make it a controlled gate, but this method has several faults:

1. It leads to code duplication: the code that implements the U gate is duplicated in the $c-U$ gate.
2. Re-writing and adapting a code is time consuming and error-prone.
3. The approach is not extensible: if the gate we want is $cc-U$ (i.e. the U gate controlled by *two* qubits) then we will not be able to re-use the code for $c-U$.

Because the purpose of implementing Shor's algorithm (to gain experience with QISKit) was already reached, we decided to stop the implementation at that point and to wait for a possible improvement of the library. In August 2018, the QISKit library still does not implement the needed method.

4.1.3 HHL algorithm

It became evident quickly that the HHL algorithm was of central importance in quantum computing and especially within the context of this internship which is oriented towards scientific computing.

After a few weeks of bibliography on the algorithm it became clear that the implementation of HHL algorithm was not trivial at all and would not be possible to fulfil completly within the time limits of the internship. The main source of complexity in the implementation of the HHL algorithm is due to Hamiltonian simulation. [46] is the most comprehensive source that explains the problem of Hamiltonian simulation and lists algorithms to solve this problem in particular cases.

Because of the impossibility to implement the HHL algorithm within the time constraints another target had to be found. Two choices were studied:

1. Implementing only a part of the whole generic algorithm.

2. Making simplification hypothesis to implement the whole algorithm for a reduced set of inputs.

The second option has been chosen because it allows us to have a fully working 'toy' program on which we can perform various analysis (execution time, error-rates, results precision, ...).

The implementation was done following [47]. In this paper, the authors explain how they implemented the HHL algorithm for a fixed 4×4 matrix A:

$$A = \frac{1}{4} \begin{pmatrix} 15 & 9 & 5 & -3 \\ 9 & 15 & 3 & -5 \\ 5 & 3 & 15 & -9 \\ -3 & -5 & -9 & 15 \end{pmatrix} \tag{4.1}$$

During the implementation, multiple mistakes in [47] implementation have been found and corrected. In addition to the mistakes, the accuracy of parts of the algorithm has been improved by several orders of magnitude by running an optimisation procedure on the scalar coefficients given in the paper. This optimisation procedure found coefficients that lowered the error of the Hamiltonian simulation part from 5×10^{-3} to 2×10^{-8} which is 5 orders of magnitude better!

The implementation of the specialised HHL algorithm can be found on my GitHub repository [48], in the `HHL` directory.

4.1.4 Analysis of the HHL implementation

The implementation of the specialised HHL algorithm found in [47] with QISKit has been validated by testing different values of b in the system $Ax = b$, solving this 4×4 system with the HHL algorithm and with a classical algorithm and comparing the two results.

The *generated* assembly code is composed of 355 quantum gates. Once compiled for the `ibmqx5` quantum chip [49] the *compiled* assembly code contains 2242 quantum gate instructions. The huge increase in the number of gates between the *generated* and the *compiled* assembly codes is mainly due to:

1. The approximation of quantum gates with other quantum gates. The *generated* assembly code uses the 14 quantum gates made available by the IBM library but real quantum chip only implements 4 different quantum gates. Consequently, the *compiled* assembly code should only use the 4 quantum gates implemented by the hardware. All the gates in the *generated* assembly that are not supported by the hardware need to be translated in terms of hardware gates.

2. The qubit mapping. The qubits in the hardware are not fully connected and additional gates need to be used when the *generated* quantum gate does not respect the hardware connectivity.

Thanks to the `qasm2error` tool presented in 4.2.3 we are able to compute the time needed by each qubit to reach its final state. These times are reported in Table 4.1.

Qubit index	0	1	2	3	13	14	15
HHL execution (μs)	466	396	413	357	232	464	466
T_1 (μs)	36.6	41	45.7	41.7	50.2	28.9	47.5
T_2 (μs)	22.2	69.3	49.9	47	63.9	48.4	67

Table 4.1 – Time needed by each qubit to reach its final state (second line) compared to the coherence times of the `ibmqx5` qubits (third and fourth lines). The qubits with an index between 4 and 12 (inclusive) are not used by the HHL implementation and are omitted for clarity. The values for T_1 and T_2 listed in this table have been downloaded with QISKit API on 30$^{\text{th}}$ July, 2018. These values may change during the next device re-calibration. The exact signification of T_1 and T_2 can be found in appendix B

The times T_1 and T_2 are the parameters of the two laws representing the probabilities that an error occurred. More precisely, the decoherence errors of type 1 follow an exponential law of parameter $\lambda_1 = \frac{1}{T_1}$ and the decoherence errors of type 2 are governed by an exponential law of parameter $\lambda_2 = \frac{1}{2T_2}$. With these parameters we can easily compute the probability that a decoherence error occurred for one qubit.

Qubit index	0	1	2	3	13	14	15
Type 1 decoherence	3e−6	6e−5	1e−4	2e−4	1e−2	1e−7	6e−5
Type 2 decoherence	3e−5	6e−2	2e−2	2e−2	2e−1	8e−3	3e−2
Type 1 and 2 decoherence	8e−11	4e−6	2e−6	4e−6	2e−3	9e−10	2e−6

Table 4.2 – Probability that no decoherence error occurs during the execution of the specialized HHL algorithm. The first two lines give the probability for each type of decoherence error and the last line gives the probability that no decoherence error (independently of its type) occurred to the concerned qubit.

Table B.3 tells us that during the execution of the specialised HHL algorithm solving a 4×4 linear system, an error due to decoherence on qubit n°0 will happen with a probability of $p_{e_0} = 1 - 8 \times 10^{-11}$.

By combining the probability of success of each qubit, we can compute that the probability that the computation succeeds with no decoherence error is $p_{\text{success}} \approx 6\text{e}{-45}$. An important point to note is that this probability of success does *not* take into account the errors introduced by quantum gates imperfections. The real probability of success is consequently even lower than 6e−45.

This shows that current hardware is unusable at the moment for real purpose HHL. However, decoherence times have improved dramatically in the last few year, which is encouraging for the future.

4.2 Auxiliary tools

Auxiliary tools have also been developed alongside the implementation of quantum algorithms. These tools are not linked to quantum algorithms but are useful for the development of such algorithms.

4.2.1 Endianness management

The main issue that regularly came up during the internship was due to endianness. The endianness of a given classical architecture (a processor for example) reflects how the bits should be interpreted in a byte or in which order the bytes of a multi-byte value should be read. In this section, endianness refers to the order of the *bits* within a *byte* (in classical computing, endianness often refers to the other definition, i.e. how the *bytes* are organised within a *multi-byte* value).

The issue of endianness first came up when trying to verify the validity of the quantum adders implemented for Shor's algorithm. In order to address this problem, wrappers around the QISKit's `QuantumRegister` class have been made to convey the information about endianness. These wrappers were used and improved all along the internship but did not fully solve the original problem because the time initially spent on their design and implementation was not sufficient.

Future development will start by designing and implementing a robust solution to manage quantum register endianness. This solution should have the same characteristic as the one used in classical computing: the users should be able to use the solution transparently without worrying about endianness but should also have all the control if they need to.

4.2.2 `qasm2image`

The second issue encountered in this internship was the lack of development tools designed for quantum computing. At the beginning of the internship, quantum debuggers did not exist (and there was still no real quantum debugger at the end of July 2018). The only way to debug a quantum program was to look at the generated assembly and try to find the bug. This task can also be done visually, by representing the compiled quantum program as a quantum circuit and trying to find the problem visually. As most of the quantum algorithms are explained with the quantum circuit visualisation, it is easy to compare the two representations (the one given in the research paper presenting the algorithm and the one obtained from the quantum assembly created) and thus find the differences.

At the beginning of the internship, the QISKit library had a visualisation function to draw the graphic representation of a given quantum circuit. But this function had several limitations:

1. It was using a non-standard LaTeX package that needed to be installed as `root`.
2. It was limited to very small quantum circuits.

3. The produced image lacked quality.

For all these reasons, I decided to implement my own quantum circuit drawer. This drawer is now available on PyPi, the Python Package Index, and installable with the command `pip3 install qasm2image`. Source code can be found on my personnal GitHub page [50]. The `qasm2image` tool was also proposed as the official drawer of QISKit but the proposal was finally declined because of licensing incompatibilities and the parallel development of another tool with the correct licensing.

4.2.3 `qasm2error`

After successfully implementing a simplified version of the HHL algorithm (see Section 4.1.3), the question of its executability on a real quantum chip arose. Two points were particularly interesting to investigate:

1. Is the execution time T_e inferior to the coherence time of the qubits T_c? What is the probability that an error due to decoherence occurs?

2. Each quantum gate having a fixed-error rate, what is the probability that at least one quantum gate fails?

To sum up, it would be interesting to be able to estimate the errors that are caused by the imperfection of the hardware and that are not due to the probabilistic nature of the algorithm. This is exactly the purpose of the `qasm2error` tool: estimating the errors caused by hardware.

At the end of the internship, the `qasm2error` tool is able to estimate the occurrence probability of a decoherence error on a given IBM chip for a given quantum program. A first method to compute the errors due to quantum gate imprecisions has been implemented, but it does not take entanglement into account and so gives wrong results on nearly all the useful quantum circuits. A new method has been implemented but its validity could not be tested because of a problem of non-concordant error-rates between theory and real experiments on IBM's quantum chips.

4.3 Quantum computing meet-ups

Two meet-ups on quantum computing were organised in CERFACS during the internship. These meet-ups consisted in a presentation followed by a debate.

The first quantum computing meet-up was aimed at beginners in the field of quantum computing and presented:

1. The basic notions of quantum computing: superposition A.1.3, quantum measurement A.1.4 and entanglement A.1.5.

2. A quick summary of the quantum technologies presented in 2.

3. An introduction to the quantum circuit model A.2.5.

4. The idea behind Grover's algorithm [11].

The presentation was done in French and lasted approximately 3 hours in total (presentation and questions).

The goal of the second meet-up was to introduce to CERFACS researchers a quantum algorithm that could be used to solve problems they are confronted with. The chosen algorithm was the HHL algorithm [1] because of its potential implications in linear algebra, finite-element method or machine learning. This second presentation was done in English and lasted 1 hour in total.

Chapter 5

Conclusion and outlook

5.1 Results

The principal results of the internship are:

- The implementation of the `qasm2image` tool that can help for debuging quantum algorithms.
- A partial implementation of the `qasm2error` tool.
- The research done on the HHL algorithm and on the problem of caracterising the thermoacoustic instabilities of a combustion chamber.
- The implementation of a special case of the HHL algorithm.
- The construction of an extensive bibliographic archive related to quantum computing and more specifically to quantum algorithms that could be useful in scientific computing.
- The two quantum computing meet-ups organised at CERFACS.

5.2 Outlook

In the future, the two tools implemented within this internship may be improved:

1. With some minor graphical modifications, the `qasm2image` tool may be able to generate quantum circuit representations suitable for inclusion in scientific papers.
2. The `qasm2error` tool can be improved to support entanglement between qubits.

In addition to the tools improvements, the HHL algorithm will be studied more thoroughly during a one year contract that will follow the internship. More specifically, the problem of implementing a 'black-box hamiltonian simulator' (see [51]) will be studied.

Finally, Dr Jean-François Parmentier offered me to build a SPOC (Small Private Online Course) with him on quantum computing. This consists in building a small

introduction to quantum computing of approximately 2 hours with questions all along the presentation. This 2-hour presentation will then be accessible online.

Published articles

[1] Aram W. Harrow, Avinatan Hassidim and Seth Lloyd. 'Quantum algorithm for solving linear systems of equations'. In: (2008). DOI: 10.1103/PhysRevLett.103.150502. eprint: arXiv:0811.3171 (cit. on pp. 18, 19, 30).

[2] D. Dervovic et al. 'Quantum linear systems algorithms: a primer'. In: *ArXiv e-prints* (Feb. 2018). arXiv: 1802.08227 [quant-ph] (cit. on p.).

[3] R. Cleve et al. 'Quantum algorithms revisited'. In: *Proceedings of the Royal Society of London Series A* 454 (Jan. 1998), p. 339. DOI: 10.1098/rspa.1998.0164. eprint: quant-ph/9708016 (cit. on p. 16).

[4] A. Peruzzo et al. 'A variational eigenvalue solver on a photonic quantum processor'. In: *Nature Communications* 5, 4213 (July 2014), p. 4213. DOI: 10.1038/ncomms5213. arXiv: 1304.3061 [quant-ph] (cit. on p. 16).

[5] Erwin Schrödinger. 'Quantisierung als Eigenwertproblem'. In: *Annalen der Physik* 384.4 (1926), pp. 361–376. DOI: 10.1002/andp.19263840404. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/andp.19263840404 (cit. on p. 1).

[6] W. Heisenberg and W. Pauli. 'Zur Quantendynamik der Wellenfelder'. In: *Original Scientific Papers / Wissenschaftliche Originalarbeiten*. Ed. by Walter Blum, Hans-Peter Dürr and Helmut Rechenberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 8–68. ISBN: 978-3-642-70078-1. DOI: 10.1007/978-3-642-70078-1_1 (cit. on p. 1).

[7] M. Born and P. Jordan. 'Zur Quantenmechanik'. In: *Zeitschrift fur Physik* 34 (Dec. 1925), pp. 858–888. DOI: 10.1007/BF01328531 (cit. on p. 1).

[8] Paul Benioff. 'The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines'. In: *Journal of Statistical Physics* 22 (May 1980), pp. 563–591 (cit. on p. 1).

[9] Richard Feynman and Peter W. Shor. 'Simulating Physics with Computers'. In: *SIAM Journal on Computing* 26 (1982), pp. 1484–1509 (cit. on p. 1).

[10] David Deutsch. 'Quantum theory, the Church–Turing principle and the universal quantum computer'. In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 400.1818 (1985), pp. 97–117. ISSN: 0080-4630. DOI: 10.1098/rspa.1985.0070. eprint: http://rspa.royalsocietypublishing.org/content/400/1818/97.full.pdf (cit. on pp. 1, 6).

[11] Lov K. Grover. 'A Fast Quantum Mechanical Algorithm for Database Search'. In: *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Philadelphia, Pennsylvania, USA: ACM, 1996, pp. 212–219. ISBN: 0-89791-785-5. DOI: 10.1145/237814.237866 (cit. on pp. 1, 18, 30).

[12] Peter W. Shor. 'Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer'. In: *SIAM J. Comput.* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 0097-5397. DOI: 10.1137/S0097539795293172. eprint: quant-ph/9508027 (cit. on pp. 1, 18).

[17] A. Acín et al. 'The European Quantum Technologies Roadmap'. In: *ArXiv e-prints* (Dec. 2017). arXiv: 1712.03773 [quant-ph] (cit. on p. 1).

[22] R. S. Smith, M. J. Curtis and W. J. Zeng. 'A Practical Quantum Instruction Set Architecture'. In: *ArXiv e-prints* (Aug. 2016). arXiv: 1608.03355 [quant-ph] (cit. on p. 8).

[31] A. Paetznick. 'Resource optimization for fault-tolerant quantum computing'. In: *ArXiv e-prints* (Oct. 2014). arXiv: 1410.5124 [quant-ph] (cit. on p. 11).

[32] E. Knill. 'Quantum Computing with Very Noisy Devices'. In: (2004). DOI: 10.1038/nature03350. arXiv: quant-ph/0410199v2 (cit. on p. 11).

[33] C. Sensiau. 'Simulations numériques des instabilités thermoacoustiques dans les chambres de combustion aéronautiques - TH/CFD/08/127'. PhD thesis. Université de Montpellier II, - Institut de Mathématiques et de Modélisation de Montpellier, France, 2008 (cit. on p. 14).

[34] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Second. Society for Industrial and Applied Mathematics, 2003. DOI: 10.1137/1.9780898718003. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898718003 (cit. on p. 15).

[35] A. Y. Kitaev. 'Quantum measurements and the Abelian Stabilizer Problem'. In: *eprint arXiv:quant-ph/9511026* (Nov. 1995). eprint: quant-ph/9511026 (cit. on p. 16).

[36] A. Montanaro and S. Pallister. 'Quantum algorithms and the finite element method'. In: 93.3, 032324 (Mar. 2016), p. 032324. DOI: 10.1103/PhysRevA.93.032324. arXiv: 1512.05903 [quant-ph] (cit. on p. 20).

[37] S. Lloyd, M. Mohseni and P. Rebentrost. 'Quantum algorithms for supervised and unsupervised machine learning'. In: *ArXiv e-prints* (July 2013). arXiv: 1307.0411 [quant-ph] (cit. on p. 21).

[38] S. Lloyd. 'Least squares quantization in PCM'. In: *IEEE Transactions on Information Theory* 28.2 (Mar. 1982), pp. 129–137. ISSN: 0018-9448. DOI: 10.1109/TIT.1982.1056489 (cit. on p. 21).

[39] D. W. Berry. 'High-order quantum algorithm for solving linear differential equations'. In: *ArXiv e-prints* (Oct. 2010). arXiv: 1010.2745 [quant-ph] (cit. on p. 21).

[40] Eleanor Rieffel and Wolfgang Polak. *Quantum Computing: A Gentle Introduction*. 1st. The MIT Press, 2011. ISBN: 9780262015066 (cit. on p. 24).

[41] Hoi-Kwong Lo, Tim Spiller and Sandu Popescu. *Introduction to quantum computation and information*. World Scientific, 1998. ISBN: 9789810244101 (cit. on pp. 24, A.iv).

[42] K.-W. Cheng and C.-C. Tseng. 'Quantum Plain and Carry Look-Ahead Adders'. In: *eprint arXiv:quant-ph/0206028* (June 2002). eprint: quant-ph/0206028 (cit. on p. 25).

[43] Steven Cuccaro et al. 'A new quantum ripple-carry addition circuit'. In: (Nov. 2004). arXiv: 0410184 [quant-ph] (cit. on p. 25).

[44] Thomas Draper. 'Addition on a Quantum Computer'. In: (Sept. 2000) (cit. on pp. 25, A.x).

[45] S. Beauregard. 'Circuit for Shor's algorithm using 2n+3 qubits'. In: *eprint arXiv:quant-ph/0205095* (May 2002). eprint: quant-ph/0205095 (cit. on p. 25).

[46] Andrew Childs. 'Simulating Hamiltonian dynamics on a small quantum computer'. 2013 (cit. on p. 25).

[47] Y. Cao et al. 'Quantum circuit design for solving linear systems of equations'. In: *Molecular Physics* 110 (Aug. 2012), pp. 1675–1680. DOI: 10.1080/00268976.2012.668289. arXiv: 1110.2232 [quant-ph] (cit. on p. 26).

[51] D. W. Berry and A. M. Childs. 'Black-box Hamiltonian simulation and unitary implementation'. In: *ArXiv e-prints* (Oct. 2009). arXiv: 0910.4157 [quant-ph] (cit. on p. 31).

[52] C. M. Dawson and M. A. Nielsen. 'The Solovay-Kitaev algorithm'. In: *eprint arXiv:quant-ph/0505030* (May 2005). eprint: quant-ph/0505030 (cit. on p. A.iv).

[53] Robert Raussendorf, Daniel E. Browne and Hans J. Briegel. 'Measurement-based quantum computation with cluster states'. In: *International Journal of Quantum Information* 07.06 (2009), pp. 1053–1203. DOI: 10.1142/S0219749909005699. eprint: https://www.worldscientific.com/doi/pdf/10.1142/S0219749909005699 (cit. on p. A.viii).

[54] Dorit Aharonov et al. 'Adiabatic Quantum Computation Is Equivalent to Standard Quantum Computation'. In: *SIAM Review* 50.4 (2008), pp. 755–787. DOI: 10.1137/080734479. eprint: https://arxiv.org/abs/quant-ph/0405098 (cit. on p. A.viii).

[55] Chetan Nayak et al. 'Non-Abelian anyons and topological quantum computation'. In: *Rev. Mod. Phys.* 80 (3 Sept. 2008), pp. 1083–1159. DOI: 10.1103/RevModPhys.80.1083 (cit. on p. A.viii).

[56] Michael A. Nielsen. 'Cluster-state quantum computation'. In: *Reports on Mathematical Physics* 57.1 (2006), pp. 147–161. ISSN: 0034-4877. DOI: https://doi.org/10.1016/S0034-4877(06)80014-5 (cit. on p. B.ii).

Websites

[13] Julian Kelly. *A Preview of Bristlecone, Google's New Quantum Processor*. Mar. 2018. URL: https://research.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html (visited on 05/04/2018) (cit. on p. 1).

[14] *Intel: Quantum Computing*. Apr. 2018. URL: https://newsroom.intel.com/press-kits/quantum-computing/ (visited on 05/04/2018) (cit. on p. 1).

[15] *IBM Q Experience*. Apr. 2018. URL: https://quantumexperience.ng.bluemix.net/qx/experience (visited on 05/04/2018) (cit. on pp. 1, 6, 24).

[16] *QUROPE Quantum Information Processing and Communication in Europe*. URL: http://qurope.eu/h2020/qtflagship/roadmap2016 (visited on 05/07/2018) (cit. on p. 1).

[18] *Airbus's quantum computing brings Silicon Valley to the Welsh Valleys*. URL: https://www.telegraph.co.uk/finance/newsbysector/industry/12065245/Airbuss-quantum-computing-brings-Silicon-Valley-to-the-Welsh-Valleys.html (visited on 05/07/2018) (cit. on p. 1).

[19] *Atos Quantum*. URL: https://atos.net/en/insights-and-innovation/quantum-computing/atos-quantum (visited on 05/07/2018) (cit. on p. 2).

[20] Allan Ho. *Announcing Cirq: An Open Source Framework for NISQ Algorithms*. July 2018. URL: https://ai.googleblog.com/2018/07/announcing-cirq-open-source-framework.html (visited on 24/07/2018) (cit. on pp. 6, 9).

[21] *OpenQASM language specification*. Apr. 2018. URL: https://github.com/QISKit/openqasm (visited on 25/04/2018) (cit. on pp. 8, 9).

[23] *Microsoft's Q# main page*. Apr. 2018. URL: https://docs.microsoft.com/en-us/quantum/quantum-qr-intro?view=qsharp-preview (visited on 25/04/2018) (cit. on p. 8).

[24] *QISKit main page*. Apr. 2018. URL: https://github.com/QISKit/ (visited on 05/04/2018) (cit. on p. 9).

[25] *Software Development Kit – Python 3*. Apr. 2018. URL: https://github.com/QISKit/qiskit-sdk-py (visited on 05/04/2018) (cit. on p. 9).

[26] *Software Development Kit – Javascript*. Apr. 2018. URL: https://github.com/QISKit/qiskit-sdk-js (visited on 05/04/2018) (cit. on p. 9).

[27] *Software Development Kit – Swift*. Apr. 2018. URL: https://github.com/QISKit/qiskit-sdk-swift (visited on 05/04/2018) (cit. on p. 9).

[28] *IBM's publicly available quantum chips*. Apr. 2018. URL: https://quantumexperience.ng.bluemix.net/qx/devices (visited on 05/04/2018) (cit. on pp. 9, B.ii).

[29] *Quipper main page*. July 2016. URL: https://www.mathstat.dal.ca/~selinger/quipper/ (visited on 25/04/2018) (cit. on p. 9).

[30] *Quantiki: List of QC simulators*. URL: https://quantiki.org/wiki/list-qc-simulators (visited on 24/07/2018) (cit. on p. 10).

[48] *Quantum tools (all the code produced during the internship)*. URL: https://github.com/nelimee/quantum-tools (visited on 01/08/2018) (cit. on p. 26).

[49] *IBM Q 16 Rueschlikon specifications*. URL: https://github.com/Qiskit/qiskit-backend-information/tree/master/backends/rueschlikon/V1 (visited on 30/07/2018) (cit. on p. 26).

[50] *qasm2image*. URL: https://github.com/nelimee/qasm2image (visited on 01/08/2018) (cit. on p. 29).

Appendix A

Introduction to quantum computing

In this chapter, the bases of quantum computing are introduced. Section A.1 will first introduce the mathematical formalism used in quantum mechanics and explain the notions of 'quantum bit', 'quantum measurement', 'superposition' and 'entanglement'. Then, section A.2 will present the theoretical abstraction used in quantum computing and explain why a theoretical model of computation is crucial. To close this chapter, section A.3 will explain in more details the theoretical abstraction used in the majority of the scientific papers: the quantum circuit model.

A.1 Quantum computing basics

A.1.1 What is a quantum bit?

The most elementary block of classical computing is the bit, an element that can only takes two values: either 0 or 1. Physically, a bit can be implemented in various ways such as magnetic orientation (used in hard drive disks), flip-flop circuits (used in random access memory) or a voltage (used to transmit the information on a cable).

A quantum bit or 'qubit' can be seen as an *improved classical bit*: it has all the properties of a classical bit plus some additional properties that a classical bit does not have access to. The two core properties that differentiate a classical bit from a qubit are 'superposition' (see section A.1.3) and 'entanglement' (see section A.1.5). Before introducing these two notions, section A.1.2 will introduce the mathematical formalism used in the field of quantum computing.

A.1.2 Notation and mathematical formalism

One of the most common mathematical representation for a quantum state is 'a ray in a finite- or infinite-dimensional Hilbert space over the complex numbers equipped with the Euclidean norm'. In other words, a quantum state can be represented by a vector of complex numbers. Because of physical constraints, this vector is taken to be of unit-length (with respect to the Euclidean norm). A specific notation is used for the representation of quantum states in quantum mechanics (and by extension, also in quantum computing): the Bra-Ket notation.

Dirac (Bra-Ket) notation

Quantum mechanics and quantum computing use Dirac notation (also known as Bra-Ket notation) to represent quantum states. In Dirac notation, $|\,.\,\rangle$ is called a *ket* and is a unit-length vector representing a quantum state. The *bra* $\langle\,.\,|$ is the Hermitian conjugate of the ket: $\langle\,.\,| = |\,.\,\rangle^\dagger = \overline{|\,.\,\rangle}^T$.

Using bra-ket notation, we can define two of the most used states in quantum computing,

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix} \text{ and } |1\rangle = \begin{pmatrix}0\\1\end{pmatrix},$$

which represent the default basis in which all the other quantum states are decomposed.

The bra-ket notation is convenient for common mathematical operations on vectors such as scalar product or outer product. The scalar product between two vectors u and v is written $\langle u|v\rangle$. The outer product that construct a matrix from two vectors u and v representing quantum states is denoted by $|u\rangle\langle v|$. Finally, bra-ket notation is widely used in quantum mechanics because it is basis-independent: we do not have to specify a basis neither an order for this basis to represent a quantum state with bra-ket notation.

Tensor product

Dirac notation explained in section (A.1.2) is sufficient to describe fully a single-qubit state but is not enough to describe multi-qubit states (i.e. the state of a system composed of 2 or more qubits).

A good way to introduce multi-qubit systems is to first think about multi-bit systems. Let each of $\mathcal{S}_1^{(c)}$, $\mathcal{S}_2^{(c)}$ and $\mathcal{S}_3^{(c)}$ be a classical system that can be described with 2 classical bits.

$$\mathcal{S}_1^{(c)} \equiv (b_{11}, b_{12})^T \;,\; \mathcal{S}_2^{(c)} \equiv (b_{21}, b_{22})^T \text{ and } \mathcal{S}_3^{(c)} \equiv (b_{31}, b_{32})^T$$

Then, the system $\mathcal{S}_4^{(c)}$ composed of $\mathcal{S}_1^{(c)}$, $\mathcal{S}_2^{(c)}$ and $\mathcal{S}_3^{(c)}$ can be described with 6 values: the first 2 values describing $\mathcal{S}_1^{(c)}$, the next 2 values describing $\mathcal{S}_2^{(c)}$ and the last 2 values describing $\mathcal{S}_3^{(c)}$. Mathematically, combining n classical states is the same as applying a direct sum to their vector representation:

$$\mathcal{S}_4^{(c)} \equiv (b_{11}, b_{12})^T \oplus (b_{21}, b_{22})^T \oplus (b_{31}, b_{32})^T = (b_{11}, b_{12}, b_{21}, b_{22}, b_{31}, b_{32})^T$$

The dimension of $\mathcal{S}_4^{(c)}$ is the sum of the dimensions of the systems composing it (i.e. 6).

Multi-qubit systems follow the same scheme as multi-bit ones except that the operator to join 2 systems is no longer a direct sum but rather a tensor product. Transposing

the previous example in the quantum world, we would have:

$$\mathcal{S}_4^{(q)} = \begin{pmatrix} q_{11} \\ q_{12} \end{pmatrix} \otimes \begin{pmatrix} q_{21} \\ q_{22} \end{pmatrix} \otimes \begin{pmatrix} q_{31} \\ q_{32} \end{pmatrix} = \begin{pmatrix} q_{11} \otimes q_{21} \otimes q_{31} \\ q_{11} \otimes q_{21} \otimes q_{33} \\ q_{11} \otimes q_{22} \otimes q_{31} \\ \vdots \\ q_{12} \otimes q_{22} \otimes q_{32} \end{pmatrix}$$

With the tensor product, the dimension of the resulting state $\mathcal{S}_4$ is now the product of the dimensions of the state composing $\mathcal{S}_4$ (i.e. 8).

Using Dirac notation, the state of a *non-entangled*[1] 2-qubit system can be written $|q_1\rangle \otimes |q_2\rangle$. This notation is often shortened $|q_1 q_2\rangle$. An other abbreviation is used in the field of quantum computing: the multi-qubit state $|q_1 q_2 q_3 q_4\rangle$ may be noted $|n\rangle$ where n is the integer corresponding to the big-endian binary number $q_1 q_2 q_3 q_4$. Some examples are given in equation A.1.

$$\begin{aligned} |1\rangle \otimes |0\rangle \otimes |1\rangle &= |101\rangle &&= |5\rangle \\ |0\rangle \otimes |1\rangle \otimes |0\rangle \otimes |0\rangle &= |0100\rangle &&= |4\rangle \\ |1\rangle \otimes |1\rangle \otimes |1\rangle \otimes |0\rangle &= |1110\rangle &&= |12\rangle \end{aligned} \tag{A.1}$$

Common misuse of language of notation in quantum computing

Equivalence between quantum states and vectors A very common misuse of notation in quantum computing is to use the Dirac notation presented in A.1.2 to represent a vector of complex numbers. For example, it is frequent in quantum computing papers to read that $|u_j\rangle$ are the eigenvectors of a given unitary matrix.

In general, quantum states and vectors can be used interchangeably by using the equation

$$\vec{b} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{2^n-1} \end{pmatrix} \equiv \sum_{i=0}^{2^n-1} b_i |i\rangle = |b\rangle .$$

Equivalence between quantum operations and unitary matrices An other common misuse of language and notation is to use interchangeably the notions of 'quantum operation'[2] and 'unitary matrix'.

This misuse of language can be explained by the fact that ($\Rightarrow$) each quantum operation can be represented by a unitary matrix and ($\Leftarrow$) each unitary matrix can be approximated to an arbitrary precision by a sequence of quantum operations (see [52]).

[1] See A.1.5 for more precision. With entanglement, the general state of a 2 qubit system is given by $|q\rangle = \alpha_0 |0\rangle \otimes |0\rangle + \alpha_1 |0\rangle \otimes |1\rangle + \alpha_2 |1\rangle \otimes |0\rangle + \alpha_3 |1\rangle \otimes |1\rangle$.

[2] See A.3.1 for a precise definition of 'quantum operation'.

As a result, it is frequent to read about the eigenvectors of a quantum operation (i.e. the eigenvectors of the unitary matrix that represents the quantum operation) or the application of a unitary matrix on a quantum state (i.e. the application of the gate that is represented by the unitary matrix on the quantum state).

Now that the notation has been introduced, the next section will explain one of the two most important difference between a bit and a quantum bit: the fundamental principle of quantum superposition.

A.1.3 Superposition states

A *bit* is defined as a quantity that can have only one of two fixed values. A bit is always either in one value *or* in the other.

This property does not hold anymore for a qubit: a qubit can choose one value from an infinite set of possible states, called *superposition states*.

A good definition of *superposition* can be found in [41]:

> [...] a single quantum bit, or qubit, has the luxury of an infinite choice of so-called superposition states. Nature allows it [the qubit] to have a part corresponding to **0** *and* a part corresponding to **1** *at the same time*, analogous to the way a musical note contains various harmonic frequencies.

The difference between a classical bit and a quantum bit in terms of attainable states is clearly visible by using the Bloch sphere representation presented in Figure A.1.

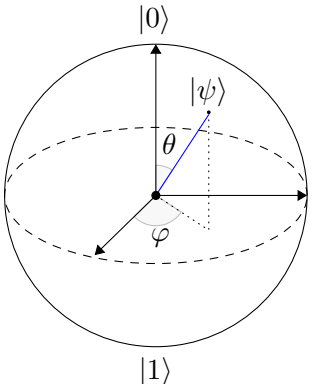


Figure A.1 – Drawing of the Bloch sphere representation. The Bloch sphere is the sphere of radius 1, centered at the origin. The possible states for a classical bit are the north and south poles of the sphere (only 0 and 1). The possible states for an isolated quantum bit are all the points located on the surface of the bloch sphere. Each point on the Bloch sphere represents a unique state.

Mathematically, a general 1-qubit state can be written

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{A.2}$$

with the following conditions on the coefficients α and β:

$$(\alpha, \beta) \in \mathbb{C}^2, \qquad |\alpha|^2 + |\beta|^2 = 1.$$

A qubit in the state A.2 is said to be in a superposition state if and only if $\alpha \neq 0$ and $\beta \neq 0$.

The coefficients α and β are tightly linked with the behaviour of quantum measurement. The next section is dedicated to this measurement operation and will explain how α and β play a role when measuring a quantum state.

A.1.4 Measurement of quantum states

In order to retrieve the result of an algorithm executed on a quantum computer, we need to be able to read the state of each qubit representing the solution at the end of the algorithm. The only way to read the value stored in qubits is to measure their state and to interpret the result of the measurement. But measurement in the quantum world does not behave as in the macroscopic world. The following sections will introduce the two main characteristics of the measurement operation, only on single-qubit states. Multi-qubits states are addressed in section A.1.5.

Measurement is probabilistic

Let first recall the mathematical representation of a single-qubit state in the basis $\{|u\rangle, |v\rangle\}$ (generalisation of equation A.2):

$$|\psi\rangle = \alpha |u\rangle + \beta |v\rangle, \text{ with } (\alpha, \beta) \in \mathbb{C}^2 \text{ and } |\alpha|^2 + |\beta|^2 = 1 \tag{A.3}$$

The outcome of measuring in the basis $\{|u\rangle, |v\rangle\}$ a qubit in the state $|\psi\rangle$ will be probabilistic and the probabilities associated to each outcome are known: $|\alpha|^2$ is the probability to measure the state $|u\rangle$ and $|\beta|^2$ is the probability to measure the state $|v\rangle$. This behaviour is an axiom of quantum mechanics and cannot be derived from physical principles but was verified in practice in various experiments.

Measurement destroys superposition

Taking back the state $|\psi\rangle$ in equation A.3, section A.1.4 told us that the outcome of measurement *can* be random (if $\alpha = 0$ or $\beta = 0$ then the measurement is deterministic, else it is a random process with pre-defined probabilities). An other important characteristic of quantum measurement is that it collapses the qubit state that is measured into the measured state.

For example imagine we have a qubit in the superposition state $|\psi\rangle$ described in equation A.3. We perform a measurement on this state and the state $|u\rangle$ is the outcome of the measurement. After the measurement the state $|\psi\rangle$ collapsed in $|u\rangle$ and is no longer in a superposition state (i.e. $|\psi\rangle = |u\rangle$). Measuring again the state $|\psi\rangle$ after the first measurement will then give a deterministic result ($|u\rangle$ for the previous example) as the qubit is no more in a superposition state.

A.1.5 Entanglement between qubits

Entanglement is the second fundamental difference between quantum bits and classical ones. In order to explain the notion of entangled states we need to define what is a separable state. A n-qubits quantum state is said to be *separable* if it can be written as the tensor product of the states of its individual components:

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$$

A n-qubits quantum state that is not separable is called an *entangled* state. Entangled states are quantum states that cannot be described only by looking individually at each of its components. In other words, there is a link between the qubits that cannot be explained just by looking at each qubit separately.

Entangled states are easier to understand with an example. A commonly used quantum state in the literature is the Bell state given by equation (A.4):

$$\left|\Phi^+\right\rangle = \frac{|0\rangle_A \otimes |0\rangle_B + |1\rangle_A \otimes |1\rangle_B}{\sqrt{2}} \tag{A.4}$$

If we consider only the state of qubit A in $|\Phi^+\rangle$, it comes out that the qubit has an equal probability to be in the state $|0\rangle$ or $|1\rangle$ after measurement. The same observation holds for the qubit B. But these individual descriptions are not sufficient to fully describe $|\Phi^+\rangle$. For example the state A.5 satisfies the 2 individual descriptions, but is not equal to $|\Phi^+\rangle$.

$$\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) \otimes \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) \tag{A.5}$$

In order to fully describe $|\Phi^+\rangle$ we should add another condition that will link the qubits A and B together. In the case of $|\Phi^+\rangle$ the condition 'when measured, the states A and B always output the same result' is sufficient to suppress the ambiguity: because of the destructive nature of measurement, measuring qubit A will *necessarily* collapse the state of qubit A to either $|0\rangle$ or $|1\rangle$. But as qubits A and B are entangled, the state of qubit B will also be impacted by the measurement and depending on the value measured for A, the final state after measuring qubit A is either $|0\rangle \otimes |0\rangle$ or $|1\rangle \otimes |1\rangle$.

To summarise, 2 or more qubits are entangled if their states are linked and cannot be fully described individually.

A.2 Theoretical background of quantum computing

Formalising a theoretical background that defines the concept of 'computation' in classical computing has been a major step in computer science. The most famous theoretical model in classical computing is probably the Turing machine that defines how information is stored (on an infinite tape) and how 'computations' are done (by moving a head over a cell, reading/writing the cell and moving left or right).

Such an abstraction is fundamental in classical computing as it allows to define the notions of *computability* and *complexity*. There exist many models of computation in

classical computing, such as the Turing machine, finite state machine, random access machines, etc.

Figure A.2 illustrates a generic model of computation.

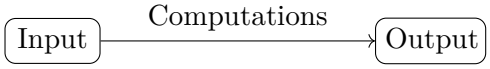


Figure A.2 – Illustration of an abstract model of computation. A real model of computation should define how data is represented (i.e. how the input and output are represented) and how computations are performed (i.e. how the input state is changed into the output state).

Following the lessons from classical computing, quantum computing researchers tried to define a 'quantum' model of computation. At the moment of writing (July 2018), several quantum models of computation are co-existing. Before briefly presenting the existing quantum models of computation we should define the notion of *universality*.

A.2.1 Model of computation and universality

A model of computation is said to be *universal* if it can map any input state to any output state.

Being universal for a classical computer means that it can map any input state $\{0,1\}^n$ to any output state $\{0,1\}^n$ just by using the logical gates allowed by the model of computation it implements.

Similarly, a quantum model of computation is said to be universal if it can map any n-qubit input state $|\psi_n\rangle$ to any n-qubit output state $|\phi_n\rangle$. But as quantum states are continuous, an other definition of universality is often used in quantum computing. Most of the time, a quantum model of computation is said to be universal when it can *approximate arbitrarily close* any output state $|\phi_n\rangle$ from any input state $|\psi_n\rangle$.

The next sections will present some of the quantum models of computation that have been defined and studied since the creation of quantum computing.

A.2.2 Measurement based quantum computer

The measurement based quantum computer, also called the one-way quantum computer, is a model exclusively based on measurement and its properties.

The qubits composing the quantum computer are initialised in a particular highly-entangled state called *cluster state* or *graph state*. Once the initial state has been prepared, computations are performed by measuring qubits in a specific order. Performing a measurement on one qubit of such an entangled state will, thanks to the properties of entanglement and measurement, change the state of the qubits that were entangled with the measured qubit.

An algorithm for such a quantum computer should output a sequence of qubit indexes, representing the qubits that should be measured. This sequence may depends on the results obtained on the previous measurements.

It has been proven that measurement based quantum computing is universal, i.e. it can approximate to an arbitrarily low precision any output state. See [53] for more information.

A.2.3 Adiabatic quantum computer

The adiabatic quantum computer model is based on Hamiltonian simulation and evolution. It consists in encoding the solution of a problem as the ground state of a given Hamiltonian and find this ground state.

To find this ground state, the model starts by constructing a simple Hamiltonian already in its ground state. Then, it evolves this simple Hamiltonian towards the one related to the solution we are searching for, by keeping the ground state of the evolved Hamiltonian thanks to the adiabatic theorem. A proof of the equivalence (in term of complexity) between the quantum circuit model and the adiabatic quantum model can be found in [54]. This makes the adiabatic quantum computer model a universal model.

A.2.4 Topological quantum computer

The topological quantum computer model is based on properties of a fundamental particle (anyons) that has not been experimentally observed at the moment. The model is also universal in the sense that it can simulate any quantum circuit with only a polynomial loss in complexity. More information can be found in [55].

Even if they are highly theoretical at the moment, topological qubits are an active subject of research because they would theoretically allow coherence times much larger than qubits based on super-conducting materials. An explanation of what is the coherence time of a qubit can be found in Appendix B.

Microsoft is currently investigating topological qubits.

A.2.5 Quantum circuit

The quantum circuit model is one of the most used model of computations in quantum computing today: most of the quantum algorithms are presented by using this model and nearly all the complexity analysis make use of this model to reason about the number of needed operations.

Definition of the model

The model of quantum circuit defines a computation as a sequence of quantum gates, which are reversible operations acting on n-qubits. In the quantum circuit model, all the qubits are initialised to the state $|0\rangle$.

The quantum circuit model is similar to the classical model of computation in two ways:

1. There are unitary blocks in both models: quantum gates for the quantum model and logical gates for the classical one.

2. The unitary blocks are chained in an ordered sequence.

But there is also a major difference between the quantum and the classical models: quantum operations need to be reversible whereas classical ones are not (`AND` logical gate for example). Quantum gates are presented more extensively in section A.3.1.

Quantum circuit representations

As said in section A.2.5, quantum circuit model is the most used one today in the field of quantum computing. Because there is no unique, standardised, *easily readable* and widely-used quantum language for the moment, quantum circuits and quantum algorithms are represented in a visual way.

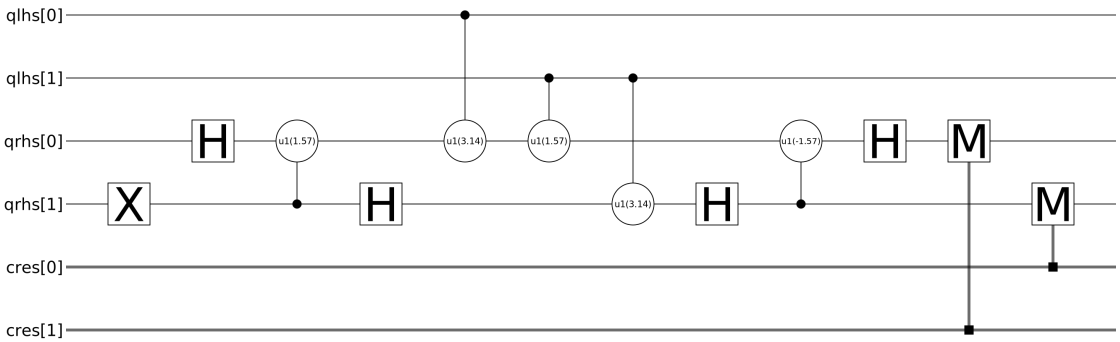


Figure A.3 – Visual representation of a quantum circuit. The quantum circuit is an in-place 2-qubit adder modulo 3 (i.e. no overflow checks are performed). The circuit is analysed in more details in section (A.2.5). Generated with the `qasm2image` tool (see 4.2.2).

Figure A.3 illustrates how a quantum circuit is represented. The graphical representation used follows the following rules:

- Time evolution goes from left to right.
- Each **qubit** is represented by an horizontal line. The qubit identifier can be written at the extremities of the line.
- Each **classical bit** is represented by a doubled horizontal line.
- Each **gate** has a unique representation. Single-qubit gates (like `X`) and measurement are represented by a square. Controlled single-qubit gates (`CX` for example) use circles.

Analysis of quantum circuit (A.3)

Registers names are written on the left of the figure: the circuit will use the quantum registers `qlhs` (for 'quantum left-hand side') and `qrhs` (for 'quantum right-hand side') and the classical register `cres` (for 'classical result'). Each of these registers contains 2 qubits identified by the numbers 0 and 1.

The quantum algorithm starts with the `X` gate on the left of the circuit. Section (A.3.1) explained the effect of the `X` gate on a quantum state: it acts like a classical `NOT` gate. Moreover, qubits are initialised to $|0\rangle$ so the `X` gate will leave the qubits in the state $|\texttt{qlhs}\rangle\,|\texttt{qrhs}\rangle = |00\rangle\,|10\rangle$.

Then, we use the algorithm presented in [44] to add the two integers represented by the registers `qlhs` and `qrhs`. Finally, measurements are performed to recover the result of the addition.

A.3 Quantum gates and algorithms

The previous section described quantum computing basics. This section will explain how qubits can be manipulated in practice to perform quantum computations:

1. Introduction to the mathematical formalism behind the operators acting on qubit states
2. Examples of some quantum operators

The whole section uses the model of *quantum circuit* and *quantum gates*. This model is similar to the one used in classical computing: a quantum circuit (resp. a *program* for classical computing) is composed of quantum gates (resp. logical gates) applied sequentially to a given number of qubits (resp. bits). Thanks to the similarity between the classical and the quantum models, many mathematical tools and formalisms from classical computing can be re-used in the quantum world.

In addition, this section will not deal with physical implementation of qubits and quantum gates. To do an analogy with classical computing, this section will explain how works the `AND` logical gate and how to use it in algorithms without showing the underlying electric circuit implementing it.

A.3.1 Qubit state transformation

Mathematical formalism

'Qubit state transformations' are operations applied on one or several qubit(s) that may change their state to an other *valid* state. From quantum mechanic properties, these quantum operations must be linear. This means that a quantum operation U can be represented as a matrix acting on the quantum state space.

$$U\left(a_1\,|\psi_1\rangle + \cdots + a_n\,|\psi_n\rangle\right) = a_1 U\,|\psi_1\rangle + \cdots + a_n U\,|\psi_n\rangle \tag{A.6}$$

The property of quantum operation to map quantum states to quantum states implies that unit length vectors must remain of unit length vectors, or equivalently that the operation is *unitary*.

$$U^\dagger U = UU^\dagger = I \tag{A.7}$$

Linearity of quantum operators is enough to prove a fundamental theorem of quantum programming: the no-cloning principle. This theorem say that a quantum operator U that clones a quantum state does not exists.

The quantum operator presented in this section is often called a *quantum gate*.

Quantum gates

Quantum gates are the quantum equivalent of logic gates in classical programming: they are used as base to construct more advanced quantum operations. Some of the most used quantum gates and their mathematical definition is given below.

Hadamard gate is one of the most used quantum gate in quantum computation. Hadamard gate acts on a single qubit and is used to create a superposed state from $|0\rangle$ or $|1\rangle$.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Applying the Hadamard gate on $|0\rangle$ gives a superposition state:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}.$$

Pauli-`X` gate is also called `NOT` gate because of its effects: it flips the state of the quantum bit.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Applying the `NOT` gate on the state $|0\rangle$ gives the state

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

`controlled NOT` gate is the most used 2-qubits gate. It takes as input a control qubit and a target qubit and applies a `NOT` gate to the target qubit if and only if the control qubit is in the state $|1\rangle$.

$$cX = CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This gate is commonly used to create entanglement between 2 qubits.

More generally, for any quantum gate U the quantum gate controlled-U or c-U is a gate that applies U to the target qubit if the control qubit is in the state $|1\rangle$.

Appendix B

Qubit coherence times

Quantum decoherence is, with quantum operation errors, one of the two sources of hardware errors on quantum chips. As one of the major source of errors in computation in today quantum chips, quantum decoherence is an active subject of research, both by physicists and exerimenters that tries to produce the most coherent qubits possible and by quantum computing researchers that are searching for ways to improve the coherence time of qubits with specific quantum procedures.

B.1 Quantum decoherence

B.1.1 Definition

The quantum decoherence phenomenon can be defined by 'the loss of coherence of the considered qubits'. Two qubits are said to be in a coherent state when there exists a definite phase relation between their states.

In other words, the theory of quantum decoherence tries to characterise how $\mathcal{S}_e^{(q)}$ (the external quantum system) affects $\mathcal{S}_i^{(q)}$ (the internal quantum system, composed of the qubits we control) by interacting with it.

Quantum decoherence is the phenomenon that explains why quantum chips need extremly low temperatures ($\approx 0.3K$) to work: low temperature avoids thermal excitation and reduce the interactions with the environment.

B.1.2 IBM's coherence times definition

Decoherence (i.e. interactions between particles) is a phenomenon that can be characterised by an exponential law of probability. IBM defined two different characteristic times related to decoherence: T_1 and T_2.

T_1 is the characteristic time for decoherence of type 1. $\lambda_1 = \frac{1}{T_1}$ is the parameter of the exponential law describing the probability that a qubit initially prepared in $|1\rangle$ interacts with its environment and ends up in its ground state $|0\rangle$. Reformulating, if we initialise a qubit with a characteristic time T_1 to the state $|1\rangle$ and measure it after a time t, the qubit will be in its ground state $|0\rangle$ with a probability of $1 - e^{-\frac{t}{T_1}}$.

T_2 is the characteristic time for decoherence of type 2. $\lambda_2 = \frac{1}{2T_2}$ is the parameter of the exponential law describing the probability that a qubit initially prepared in $|+\rangle = |0\rangle + |1\rangle$ evolves into the state $|0\rangle\langle 0| + |1\rangle\langle 1|$[1].

B.2 Coherence times for IBM's qubits

As explained in the previous section, IBM introduced its own definition of the characteristic times T_1 and T_2. The values advertised by IBM on their website are summarised in the tables below.

Qubit index	0	1	2	3	4
T_1 (μs)	62.40	55.10	48.40	49.00	53.30
T_2 (μs)	77.50	64.00	54.70	57.30	36.40

Table B.1 – Coherence characteristic times for the `ibmqx2` backend. The times were recovered on IBM's website [28] on 12$^{\text{th}}$ August.

Qubit index	0	1	2	3	4
T_1 (μs)	43.60	37.10	51.00	55.00	60.90
T_2 (μs)	48.60	23.00	51.30	24.20	11.60

Table B.2 – Coherence characteristic times for the `ibmqx4` backend. The times were recovered on IBM's website [28] on 12$^{\text{th}}$ August.

Qubit index	0	1	2	3	4	5	6	7
T_1 (μs)	38.00	40.40	44.50	47.00	54.10	52.00	51.80	44.90
T_2 (μs)	23.90	64.40	67.30	75.90	89.40	53.40	72.10	54.80
Qubit index	8	9	10	11	12	13	14	15
T_1 (μs)	48.50	45.60	57.80	46.40	44.40	52.20	36.80	36.30
T_2 (μs)	72.90	74.90	96.50	94.70	56.00	90.10	60.50	64.40

Table B.3 – Coherence characteristic times for the `ibmqx5` backend. The times were recovered on IBM's website [28] on 12$^{\text{th}}$ August.

[1] The notation used here represents a quantum state. It is called a mixed state. The specific state $|0\rangle\langle 0| + |1\rangle\langle 1|$ means 'the system has a probability of 0.5 to be in the state $|0\rangle$ and a probability of 0.5 to be in the state $|1\rangle$'. Warning: this state *is not* $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$. The difference between the two states is the following: $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ is a superposition of the states $|0\rangle$ and $|1\rangle$ whereas $|0\rangle\langle 0| + |1\rangle\langle 1|$ is either in the state $|0\rangle$ or in the state $|1\rangle$ but we do not know which one. See [56, Section 2.4] for a thorough explanation on mixed states formalism.