THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale EDI2S

Unité de recherche LIRMM

Implémentation, analyse et améliorations bas-niveau d'algorithmes quantiques pour le calcul scientifique.

Présentée par Adrien Suau
le 27 Octobre 2022

Sous la direction de Aida Todri-Sanial, Gabriel Staffelbach et Eric Bourreau

Devant le jury composé de

Aida Todri-Sanial	Directeur de recherche	CNRS, LIRMM, Université de Montpellier	(Maître de thèse)
Gabriel Staffelbach	Chercheur	CERFACS	(Co-encadrant)
Éric Bourreau	Professeur associé	LIRMM, Université de Montpellier	(Co-encadrant)
Vedran Dunjko	Professeur associé	LIACS, Université de Leiden	(Rapporteur)
Omar Fawzi	Directeur de recherche	École Normale Supérieure de Lyon	(Rapporteur)
Simon Perdrix	Directeur de recherche	INRIA, LORIA	(Examinateur)
Stephan De Bièvre	Professeur	Université de Lille	(Examinateur)
Marko Rančić	Chef de projet	TotalEnergies	(Invité)

UNIVERSITÉ DE MONTPELLIER

Remerciements

Cette thèse résulte d'une collaboration entre TotalEnergies, le Laboratoire d'Informatique, Robotique et Microelectronique de Montpellier (LIRMM) et le Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique (CERFACS). Elle a été financée par TotalEnergies.

Je tiens tout d'abord à remercier les membres du jury pour avoir accepté d'évaluer mes travaux de thèse. Un grand merci à Vedran Dunjko et Omar Fawzi pour l'intérêt qu'ils ont porté à mes travaux en tant que rapporteurs, ainsi qu'à Simon Perdrix et Stephan De Bièvre pour leurs remarques en tant qu'examinateurs. Je veux aussi remercier Marko Rančić pour ses encouragements.

Un énorme merci à mes encadrants de thèse, Aida Todri-Sanial, Gabriel Staffelbach et Éric Bourreau qui m'ont épaulé durant ces 3 années et ont su me guider jusqu'à la fin de la rédaction du manuscrit. Ils ont su m'offrir un environnement propice au développement de ma recherche scientifique et rendre l'expérience de ces 3 années de thèse agréable.

Je veux réserver dans ces remerciements une place spéciale à Henri Calandra, sans qui cette thèse n'aurait jamais vu le jour. Il m'a permis de commencer cette thèse dans un environnement de recherche passionnant au coeur d'un groupe soudé. Je suis aussi très reconnaissant de toutes les discussions que nous avons pu avoir ensemble, que ce soit sur le calcul quantique ou d'autres sujets, autour d'un "café virtuel". Je tiens aussi à remercier les membres de ce fameux groupe, Elvira Shishenina, Charles Moussa et Jean-Baptiste Latre, pour toutes les discussions que nous avons eu ensemble et les aventures que l'on a pu partager.

Les premiers mois de ma thèse ont été effectués au LIRMM, où j'ai pu rencontrer au fil des années mes collègues Montpellierains que je remercie pour l'ambiance dans le couloir et la bonne humeur générale. Parmis eux, j'aimerai tout particulièrement remercier Siyuan Niu, qui a réalisé sa thèse en calcul quantique en même temps que moi, pour toutes les collaborations que nous avons pu réaliser ensemble, telles que l'animation d'un hackathon à Montpellier, une collaboration de recherche, la participation à un autre hackathon, ... Finalement, je veux remercier Jean-Michel Tores pour son indéfectible bonne humeur et son support tout au long de ces 3 années.

La majorité de ma thèse s'est déroulée au CERFACS, à Toulouse, où j'ai rencontré énormément de collègues qui sont au fil du temps devenus des amis. Merci à Thomas L., Félix, Théo, Antoine, Clovis, Jonathan, Guillaume, Lionel, Nicolas B., Thomas M., Etienne, Minh, Nicolas U., Aurélien et toutes les personnes avec qui j'ai pu discuter au CERFACS! Un grand merci aussi à l'équipe administrative du CERFACS, Brigitte, Lydia, Michèle ainsi qu'à l'équipe CSG, Fred, Isabelle, Nicolas, Patrick, Fabrice et Gérard, qui m'ont accueilli dans leurs bureaux pendant quelques mois et ont toujours été là en cas de soucis informatique.

Parmis ces amis du CERFACS déjà cités, je veux tout particulièrement remercier les "Canal riders", qui m'ont permis de passer la dernière semaine de rédaction, la plus compliquée, dans le meilleur environnement possible!

Je veux aussi remercier mes amis de longue date qui m'ont supportés pendant cette période de ma vie, Paul, Emeline, Jean-Côme, Eddy, Patrick et Mathilde. La fin de ma thèse aurait été bien plus compliquée sans le support de ma copine, Laura, que je tiens à remercier du fond du coeur pour son soutien.

Finalement, rien de ce qui est écrit dans les pages qui suivent n'aurait été possible sans le soutient continu de ma famille. Un énorme merci à ma mère et à mon père, qui m'ont permis de suivre la voie que je voulais tout au long de mes études et qui m'ont toujours soutenu, ainsi qu'à mes frères Paul et Gabriel et à toute ma famille!

Abstract

Quantum computing is a new paradigm that may be able to solve some very specific and interesting problems faster than classical computing. But to reach the regime where quantum computers can outperform their classical counterparts, several crucial milestones must be attained. Hardware needs to improve its error rates and qubit number, algorithms have to evolve in order to be able to run correctly on noisy hardware, implementations and compiler should be tailored to the targeted hardware, analysis of quantum programs and quantum states will likely need improvements.

In this thesis, we study several of these milestones and extend over the state of the art, trying to get closer to the quantum supremacy regime. We first implemented a non-trivial quantum algorithm, analysed its behaviour and performed advanced resources estimation. From this implementation and analysis, we confirmed that current quantum hardware could not run such an implementation, even after taking into account hardware intricacies and an extensive and comprehensive optimisation efforts. These optimisation efforts revealed a lack of tools to synthetically visualise the quantum circuit that could guide optimisation similar to classical computing ones. This omission led to the development of qprof, a quantum program analysis tool able to efficiently provide a human-readable structure and cost report from the analysis of a given quantum circuit.

Next, we present a classical algorithm to solve the qubit routing problem, one of the most costly steps in quantum compilers, by taking into account hardware calibrations to tailor the final solution to the targeted hardware. We describe how the algorithm works and find that it can improve the fidelity of the quantum computations when it is used. We also implement a variational algorithm to solve linear systems of equations and analyse its requirements and behaviour when executed on real quantum hardware. Finally, we performed an extensive study on single-qubit quantum noise and introduced a new visualisation of quantum states. Using this new representation, we isolated errors in qubits prepared in a known quantum state that are currently not corrected by the automatic calibration of quantum chips.

Résumé

Le calcul quantique est nouveau paradigme qui serai en capacité de résoudre certains problèmes spécifiques plus vite qu'un ordinateur classique. Mais pour atteindre le régime dans lequel les ordinateurs quantiques seront plus efficaces que leurs équivalents classiques, plusieurs étapes cruciales doivent être atteintes. Les taux d'erreur des puces quantiques actuelles doivent être drastiquement améliorés, tout comme leur nombre de qubits, les algorithmes doivent évoluer pour prendre en compte le taux d'erreur élevé des puces, les compilateurs et implémentations doivent être adaptés à la puce quantique qui sera utilisée pour l'exécution et les façons d'analyser des états ou programmes quantiques nécessitent des améliorations.

Dans cette thèse, nous nous concentrons sur certaines de ces étapes et améliorons l'état de l'art en essayant de se rapprocher le plus possible du régime où les ordinateurs quantiques auront un avantage sur les ordinateurs classiques. Nous commençons par montrer comment nous avons implémenté un algorithme quantique non trivial et analysé son comportement ainsi que les ressources nécessaires à son exécution. Cette analyse a confirmé que les puces quantiques actuelles étaient incapables d'exécuter ce genre d'implémentation, même après avoir pris en compte les subtilités dues au matériel et optimisé grandement le programme. Cet effort d'optimisation a donné lieu à l'implémentation de l'outil qprof, un profileur de programme quantique capable d'analyser des implémentations et de générer un rapport textuel et lisible résumant la hiérarchie des appels et leur coût approximatif.

Nous présentons aussi un algorithme classique pour résoudre le problème du routage de qubit, une des étapes les plus coûteuse en terme de portes quantiques additionnelles. Nous avons aussi implémenté un algorithme variationnel permettant de résoudre des systèmes linéaires et analysé son comportement sur les puces quantiques actuelles ainsi que les besoins en ressources afin d'exécuter cet algorithme. Finalement, nous avons effectué une étude sur les erreurs présentes sur des qubits isolés et introduit une nouvelle visualisation permettant de mettre en avant des erreurs systématiques et non mitigées par les calibrations automatiques réalisées à intervalles réguliers sur les puces.

Contents

Contributions

The work described in this manuscript has been has been performed within a collaboration between CERFACS (Centre Européen de Recherche et Formation Avancée en Calcul Scientifique), LIRMM (Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier) and TotalEnergies, that funded this PhD. The thesis started at a cornerstone moment of quantum computing, when several world-wide companies introduced their own framework to help developing quantum algorithms. Some companies such as IBM even made publicly available some of their quantum chip to let researchers perform experiments on real quantum hardware.

These incredible advances and the fact that experiments "hands-on" with a quantum computer became possible helped a lot to change the public vision on quantum computing from a purely theoretical field to a more practical research subject that might help in solving real-world problems. But with this realisation came the question of the timescales involved and the engineering and research efforts required to solve such problems using quantum computing.

This thesis aims at providing an answer to this question with the current state of the art in quantum computing. More specifically, we study in this manuscript several questions related to the application of quantum computing to solve scientific computing problems. The following paragraph briefly introduce the different chapters of this thesis along with the work presented in these chapters.

Chapter 1 (Introduction to Quantum Computing) We introduce the mathematical tools and notations commonly used to define quantum computing and re-used in every section of this manuscript. We define the different models of quantum computation and delve into the definition of the gate-based model this work is based on. We end the chapter by introducing a few visualisations that are re-used in Chapters 3 to 7.

Chapter 2 (Scientific computing and quantum computing) We review the problems included in the "scientific computing" field and perform a more in-depth explanation of some of the most representative and impactful applications that were made possible by the rise of scientific computing. We then study how these problems are formulated using a mathematical formalism, and list some of the classical algorithms and methods used to solve these mathematically reformulated problems. The second part of the chapter is interested in the quantum algorithms and methods that have been introduced in the last few years or decades to solve these mathematical problems. The chapter ends by introducing the current state-of-the-art on software and hardware used in quantum computing.

Chapter 3 (PDE solver) We implement a non-trivial partial differential equation solver able to solve the 1-dimensional wave equation with Dirichlet boundary conditions. A complete resources requirement analysis has been performed on the implementation and show that the implementation is effectively able to solve the wave equation by testing it on a quantum computer simulator and comparing it to the result obtained with classical methods. We finish the study by reporting on our findings on the optimisation of the implementation and we present a real-case study on the usage of profilers for quantum programs.

Chapter 4 (qprof) We introduce a new software able to analyse any quantum circuit implementation and to output a profiling report. The tool introduced, called qprof (which stands for **q**uantum **prof**iler), is able to understand quantum circuit from a variety of different quantum computing frameworks thanks to a framework-agnostic representation of the quantum circuit model implemented specifically for qprof. Its performances are studied on several large quantum circuits implementing Shor's algorithm, Grover's algorithm or the wave equation solver presented in Chapter 3. The reports produced by qprof are standard and can be post-processed with well-established tools to produce call-graphs, a very concise visual representation of the hierarchy between each of the quantum routines used in the circuit. qprof has been the most helpful tool when trying to optimise the quantum circuit implementation of Chapter 3.

Chapter 5 (Hardware aware compiler) We improve a crucial part of the quantum circuit compiling stack by introducing a qubit-mapping algorithm able to dynamically change its results according to the hardware calibrations data provided. In addition to the calibration-aware method, we introduce an innovative way of choosing the kind of quantum gate that will be inserted at each step of the algorithm. By introducing a new way of choosing, at each step, the best quantum gate to insert in order to make the compiler quantum circuit compliant with the hardware topology, the new method introduced outperforms its competitor both in terms of circuit fidelity and additional gates while retaining a good asymptotic complexity and being efficient in practice.

Chapter 6 (Variational quantum linear solver) We study the practical implementation of a variational quantum algorithm able to solve systems of linear equations. We start by performing a review of the currently available quantum hardware, often denoted as Noisy Intermediate-Scale Quantum (NISQ) hardware. With their low (often < 100) number of qubits and their relatively high (typically $\approx 1\%$ for 2-qubit quantum gates) error-rates, today's quantum hardware cannot realistically execute quantum algorithms such as Shor's or Grover's algorithm. Variational quantum algorithms try to address this issue by reformulating the problem as the minimisation of a cost function, which helps in reducing the size of the quantum circuits executed. The runtime and convergence of the Variational Quantum Linear Solver (VQLS) algorithm is then studied on several systems of linear equation of interest. The chapter ends by comparing the results obtained on real quantum hardware with the different results obtained on simulators.

Chapter 7 (Single qubit tomography visualisation) We end the manuscript by an exploratory study of single-qubit quantum state tomography as a way to improve our understanding of hardware noise through measurements. In this chapter we introduce a new way of visualising single-qubit tomography data. We perform topography experiments on several quantum chips from IBM and show that our new visualisation is able to highlight unexpected noise features. We envision that this visualisation can be exploited to improve our understanding of single-qubit noise as well has the impact of error-mitigation techniques on said noise. We also note inconsistencies across the different reconstruction methods employed in the tomography study and that are very likely due to systematic biases in the output of the quantum chips.

Part I

Foreword

Chapter

1 Introduction to Quantum Computing

We start this manuscript by a short introduction to quantum computing. In Section 1.1 we first dive into the historical steps and research milestones that ended up in the creation of the field. Then, the mathematical framework used to describe quantum computing is introduced in Section 1.3 and Section 1.2. Finally, a few visualisations that will be used across this manuscript are introduced in Section 1.4.

Contents

1.1 History of quantum computing

The story of quantum computing begins in the end of the 19$^{\text{th}}$ century with a few singular behaviours observed by physicists that the theory of *classical physics* was unable to explain. It was for example the case of what is now known as black-body radiation, for which classical physics was failing to correctly describe all the features observed in practice.

A few decades of research forward, these "holes" in the theory of classical physics were gradually patched by introducing several hypothesis that would explain on a case-by-case basis the conflicting observations: energy is distributed as discrete *quantas* (or "energy elements" from Planck's initial formulation [1]), particles can show wave characteristics, and waves can exhibit particle characteristics. Even though these assumptions were successfully explaining most of the controversial behaviours observed physically, they were only a few patches introduced in order to "fill the gap" in the current theory and a complete theory that would explain rigorously these behaviours from the ground up was still missing.

A real formalisation of the theory, now called *modern quantum mechanics*, only started in 1927 with Heisenberg formulating an early version of his famous uncertainty principle [2]. The basis of the theory of modern quantum mechanics was then introduced in 1930 by Paul Dirac in his famous textbook [3]. Two years later, John Von Neumann formulated a rigorous basis for quantum mechanics using linear operators on Hilbert spaces [4], still widely used nowadays.

In parallel to the advances in our understanding of quantum physics, a new field that will soon be called "computer science" is introduced in 1936 by Turing [5]. In his work, Turing defines the theoretical framework that will ultimately lead to "computers", machines able to perform computations way faster than humans.

The reunion of these two fields, quantum mechanics and computer science, only happens several decades later with the introduction and formalisation of "quantum computing", a field of researched concerned about performing computations by using the laws of quantum mechanics. The field of quantum computing has been initially formalised by Benioff [6, 7]. In his works, Benioff defines (in 1980) and refines (in 1982) the idea of a quantum Turing machine, the analogue of the classical Turing machine, the theoretical foundation of all modern classical computing introduced by Turing.

The definition of a quantum Turing machine given by Benioff in [6, 7] is based on what will be known later as *Hamiltonian simulation*, one of the motivational problem for the emergence of research in the field of quantum computing and the subject of a foundational paper by Feynman [8]. But the computational model defining "quantum computations" was not complete up until the work of Deutsch introducing in 1985 the *universal quantum Turing machine* in [9].

Since the complete mathematical formalisation of what is a "quantum computation", a lot of research have been performed to find quantum algorithms, i.e., algorithms able to use the additional properties offered by a quantum computer over its classical counterparts and that may make it more efficient at solving some tasks. One of the first quantum algorithm showing in practice that quantum computers are able to solve some problems faster than their classical counterpart has been described by Deutsch and Jozsa in 1992. In their work [10], Deutsch and Jozsa introduce a quantum algorithm able to solve in constant time a problem that requires a classical computer to perform an exponential number of operations.

The algorithm introduced by Deutsch and Jozsa is the first of a long list of quantum algorithms. Among the most important milestones in this list, one can cite the algorithms of Grover [11], that showed a quadratic speed-up of quantum computing over classical computing on a very generic problem, and Shor [12] that introduced a quantum algorithm able to find the prime factors of a given integer exponentially faster than classical computers, leading to the proof that most of the asymmetrical cryptography algorithms used in the world were not "quantum-proof", i.e., are easy to break with a quantum computer.

A particularly interesting algorithm for the content of this manuscript has been introduced in 2008 by Harrow, Hassidim, and Lloyd in [13] and is able to solve sparse systems of linear equations exponentially faster than what a classical computer might be able to[1]. This quantum algorithm will be the entry point for new researches to apply quantum computing to the field of scientific computing, which is the subject of this manuscript.

1.2 Models of quantum computation

Before diving into the mathematical framework formalising quantum computing and that we will be using through this manuscript, we present in this section a few of the models used to describe quantum computation. These models of computation helps in defining without ambiguity what is a "quantum computation" and how it can performed. Knowing about the different models of computation is important as quantum algorithms might be defined using any one of these

[1] All the assumptions underlying this claim are detailed later in this manuscript.

models. In the framework of this manuscript, we will be mostly using the gate-based model of computation presented in Section 1.2.3, with only a few cited algorithms using quantum annealing, a relaxed version of the model presented in Section 1.2.1.

The first complete model of computation that defined what a "quantum computation" might be is the universal quantum Turing machine, introduced by Deutsch in [9]. Since then, different models of computation have been devised through the years. These models of computations are more "practical" than the universal quantum Turing machine model in the sense that they are easier to use and reason about for a computer scientist.

All the models presented in the next sections have been shown to be equivalent to the universal Turing machine model. Loosely speaking, this means that they are all able to represent any quantum computation. Moreover, any quantum computation described within one of these model can be translated to any other model with only a small (polynomial) impact on its asymptotic complexity.

1.2.1 Adiabatic quantum computing

The adiabatic model of computation have been first introduced by Farhi et al. in [14] and is universal as shown in [15, 16].

The model of computation always follow the same 2-step pattern. First, prepare the qubits the computation will happen on in a quantum state $|\phi_0\rangle$ that is known to be the ground state of the Hamiltonian H_0 representing the interaction currently applied on the qubits. Secondly, change *slowly* the interaction Hamiltonian to H_1, whose ground state encodes the solution to our problem. If the evolution happens slowly enough, the state in which the qubits are left in at the end of the evolution (i.e., when H_1 is the interaction Hamiltonian) is the ground state of H_1. This is guaranteed by the *adiabatic theorem* that has been proved in [17].

1.2.2 Measurement-based quantum computing

Measurement-based quantum computing, also known as "One-way quantum computing", is a model of computation in which operation are only performed by single-qubit measurements on a known quantum state. The model of measurement-based quantum computing can also be decomposed into 2 main steps that are state-preparation and measurement: all the computations start by preparing a known, highly entangled, quantum state and the actual computing logic is implemented by performing successive single-qubit measurements on this quantum state. This model as been first introduced by Raussendorf and Briegel in [18], and its universality when using cluster states [19], has been proved in [20].

1.2.3 Gate-based quantum computing

The models of computation presented in Sections 1.2.1 and 1.2.2 are quite different from the classical model of computation programmers and theoreticians are used to. Closer to well known models of classical computation, the gate-based model of quantum computation is used extensively in the quantum computing community. This model of quantum computation can be described by a *sequential* process consisting of *three* steps.

First, *all* the qubits that will be used during the quantum computation should be initialised to a known quantum state. It is common to initialise individually each of the qubits to the $|0\rangle$ quantum state. This choice is motivated by the fact that the quantum state $|0\rangle$ is often represented physically by the ground state of the qubit, which is "easy" to prepare. Then, the actual computation is performed. Quantum gates are applied in a specified order to evolve the qubits toward the desired final quantum state. Finally, one or more qubits are measured in order to extract the result of the quantum computation.

There exist a few variations to the gate-based model of quantum computation described here, but these variations do not change the "computational power" of the model. To name a few, the initial state might be changed to any other quantum state, mid-circuit measurements or classical-feedback (i.e., using the result of a mid-circuit measurement to change the gates that will be executed next) might also be allowed.

The description of the full quantum computation, containing the three steps of qubit initialisation, quantum gate application and final measurements listed above, is often called a *quantum circuit*. Quantum circuits are the most widely used way to represent a quantum algorithm in the literature as of today. As noted in the beginning of this section, the gate-based is also very close to the usual model of classical computing where instructions are executed sequentially on a processing unit.

This manuscript will exclusively use the gate-based model of quantum computation and quantum circuits to describe algorithms. In order to efficiently and unambiguously communicate a quantum computation, we will use the standard representation of quantum circuits that is described in Section 1.4.2.

1.3 Quantum computing theoretical framework

Before diving into the relationship between quantum computing and scientific computing, we introduce the theoretical foundations that have been mostly introduced in [4] and are used to describe quantum computations. Having a well established and robust framework that describes the different components defining a "computation" is crucial. It allows to build algorithms according to the rules defined, reason about their computational complexity, and restrict the different models of computation to match with what is physically possible.

The theoretical framework formalising the field of quantum computing was introduced in 1932 by John Von Neumann in [4] and is still widely used nowadays. It makes an extensive use of linear algebra to define each of the components used to formally explain what is a "quantum computation". This section is devoted to introduce these components. The definitions introduced in the following paragraphs will be re-used throughout the document.

1.3.1 Closed quantum system

A quantum system can be any physical object, for example a few atoms, photons or electrons, that can be measured to extract some physical property of the system such as momentum, position, charge, etc. For a measured quantity that has N possible classical outcomes denoted ψ_0, ψ_1, ..., ψ_{N-1}, the quantum measurement outcomes are often written using the Dirac (or bra-ket) notation: $|\psi_0\rangle$, ..., $|\psi_{N-1}\rangle$ (see Definition 2).

A closed quantum system is a quantum system that is supposed to be perfectly isolated from any exterior interaction. Closed quantum systems are particularly useful as they are considered to be systems on which we have full control, i.e., that does not experience any errors during the computation.

Definition 1 (Hilbert space)**.** A Hilbert space H is a complex-valued vector space equipped with a scalar product $\langle \cdot, \cdot \rangle$ that is:

- Conjugate symmetric:

$$\forall (x, y) \in H^2,\ \langle x, y \rangle = \overline{\langle y, x \rangle} \tag{1.1}$$

- Linear in its first argument:

$$\forall (x_1, x_2, y) \in H^3,\ \forall (a, b) \in \mathbb{C}^2,\ \langle a x_1 + b x_2, y \rangle = a \langle x_1, y \rangle + b \langle x_2, y \rangle \tag{1.2}$$

- Positive definite:

$$\forall x \in H, \begin{cases} \langle x, x \rangle > 0 & \text{if } x \neq 0 \\ \langle x, x \rangle = 0 & \text{if } x = 0 \end{cases} \tag{1.3}$$

Postulate 1 (State space and state vector)**.** *Any closed quantum system can be associated with a corresponding Hilbert space H known as the* state space*. The system state is completely (but not necessarily uniquely) described by its* state vector*, a unit-norm (according to the 2-norm) vector in H.*

Quantum states are often visually represented by using the standard Dirac (or bra-ket) notation used in quantum mechanics and presented in Definition 2.

Definition 2 (Braket notation)**.** The ket $|\cdot\rangle$ is used to represent a vector (quantum state) from the Hilbert space H considered when studying a particular quantum system. The bra $\langle\cdot|$ represents the complex-conjugate of the corresponding ket, i.e.,

$$\forall x \in H, \langle x| = |x\rangle^\dagger = \overline{|x\rangle^T}. \tag{1.4}$$

The scalar product is often written using a condensed form of the braket notation:

$$\forall (x, y) \in H^2, \langle x, y \rangle = \langle x| \, |y\rangle = \langle x|y\rangle \, . \tag{1.5}$$

The quantum state representing the state of a given closed system is denoted as a *pure* quantum state. Following Postulate 1, pure quantum states are represented as unit-norm vectors in the N-dimensional Hilbert space defined by the orthonormal basis $(|\psi_i\rangle)_{0 \leqslant i \leqslant N-1}$. As such, any *pure* quantum state can be written as

$$\sum_{i=0}^{N-1} \alpha_i \, |\psi_i\rangle \tag{1.6}$$

with $\alpha_i \in \mathbb{C}$, $\forall 0 \leqslant i \leqslant N-1$ and verifying the unit-norm condition

$$\sum_{i=0}^{N-1} |\alpha_i|^2 = 1 \tag{1.7}$$

of Postulate 1.

The simplest quantum mechanical system, and the system we will be using the most in this manuscript, is called the *qubit*. It is represented as a unit-norm vector from 2-dimensional state space spanned by the orthonormal basis $\{|0\rangle \, , |1\rangle\}$, also called the *computational basis*.

Definition 3 (Quantum superposition)**.** One of the fundamental properties of a quantum system that differentiate it from a classical system is superposition. Mathematically, a qubit is in a "superposition" if its state can be written as

$$|\psi\rangle = \alpha \, |0\rangle + \beta \, |1\rangle \tag{1.8}$$

with $\alpha \neq 0$ and $\beta \neq 0$.

1.3.2 Quantum operations

In order to be able to perform computations on a quantum system we should be able to manipulate its quantum state through the application of *quantum operations*. Quantum operations are defined by Postulate 2 as unitary evolutions.

Postulate 2 (Evolution of a closed quantum system)**.** *A closed quantum system evolution between any initial time t_1 and final time t_2 is described by a* unitary *transformation. This means that for any times t_1 and t_2, there is a unitary matrix U such that*

$$|\psi_{t_2}\rangle = U\,|\psi_{t_1}\rangle\,. \tag{1.9}$$

Equivalently, the state of a closed quantum system is governed by the Schrödinger equation

$$i\frac{d}{dt}\,|\psi\rangle = H\,|\psi\rangle \tag{1.10}$$

where H is the Hamiltonian *of the quantum system and the reduced Planck constant $\hbar$ has been set to* 1 *(or equivalently absorbed in the Hamiltonian H).*

Quantum operations (or evolutions) are often denoted as quantum gates, using an analogy with logical gates from classical computing. In the particular case of a single isolated qubit, several gates are of particular interest. Specifically, Pauli matrices X, Y and Z defined as

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{1.11}$$

widely used, along with the Hadamard gate denoted H and that acts as

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{1.12}$$

1.3.3 Composite quantum systems

Two closed quantum systems Q_0 and Q_1 can be represented as one larger closed quantum system Q, called a *composite* quantum system. In such a case, the state spaces and state vectors from each of the independent and closed quantum systems Q_0 and Q_1 are composed according to Postulate 3.

Postulate 3 (Composite closed quantum systems)**.** *The state space H of the composite closed quantum system Q composed of two closed quantum systems Q_1 and Q_2 with state space H_1 and H_2 can be described with the tensor product of H_1 and H_2:*

$$H = H_1 \otimes H_2. \tag{1.13}$$

The quantum states compose in the same way:

$$\forall\ (|\psi_1\rangle\,, |\psi_2\rangle) \in (H_1, H_2),\ |\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \in H. \tag{1.14}$$

The properties of the tensor product operation $\otimes$ allow to deduce that the dimension of a composite quantum system is given by the product of the dimensions of its constituents. Specifically, let Q_0, Q_1 and Q_2 be three qubits (i.e., 2-dimensional state spaces), then the composite system $Q = Q_0 \otimes Q_1 \otimes Q_2$ is a 8-dimensional state space. In general, a composite system composed of n qubits is a 2^n-dimensional state space.

Composite quantum systems consisting of n qubits are very frequently used. When using the standard computational basis, it is common to use the simplified notation $|0\rangle \otimes |1\rangle \otimes |0\rangle = |010\rangle$ for these composite systems. Moreover, if the context is sufficiently clear, the notation can be simplified even further by replacing the binary digits by their representation in base 10: $|010\rangle = |2\rangle$.

Note that this notation needs context at least to avoid any confusion on the endianness convention used (either big-endian $|10110\rangle = |24\rangle$ or little-endian $|10110\rangle = |13\rangle$) and the actual number of qubits used (the state $|2\rangle$ might be interpreted, using the big-endian convention, as $|10\rangle$, $|010\rangle$, $|0010\rangle$, ...).

Definition 4 (Quantum entanglement)**.** The second fundamental property of quantum physics that complements the quantum model of computation is entanglement. Mathematically, two quantum systems are entangled when it is not possible to write their quantum state as a tensor product of the individual state of each system. In other words, it is not possible to describe fully one of the entangled quantum system without considering the other.

For example, the Bell state

$$\left|\Psi^+\right\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}} \tag{1.15}$$

is entangled because it impossible to express it as a tensor product of two single-qubit states:

$$(\alpha_1 |0\rangle + \beta_1 |1\rangle) \otimes (\alpha_2 |0\rangle + \beta_2 |1\rangle) \tag{1.16}$$

with $|\alpha_1|^2 + |\beta_1|^2 = 1$ and $|\alpha_2|^2 + |\beta_2|^2 = 1$. On the other side, the state

$$\frac{|00\rangle + |01\rangle}{\sqrt{2}} = |0\rangle \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{1.17}$$

is not entangled as it can be decomposed as the tensor product of two 1-qubit states.

The introduction of entanglement raise the question of how it is created between two quantum systems. As for any manipulation involving quantum states, entanglement between two quantum systems Q_0 and Q_1 is created via the application of a specific quantum operation to the composite quantum system $Q = Q_1 \otimes Q_2$.

A particularly interesting group of quantum gates when it comes to entanglement are *controlled* gates that are able to apply a quantum operation to a quantum system (called *target*) conditionally to the state of another quantum system (called *control*). One of the most used controlled quantum gate is probably the controlled-NOT (also called controlled-X or CX). It applies the X quantum gate (see Equation (1.11)) to a target qubit only when the state of the control qubit is $|1\rangle$. The matrix representation of the CX gate when qubit are read using the big-endian ordering is

$$\mathrm{CX} = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{1.18}$$

In general, controlled quantum gates are not limited to one *control* qubit and can have an arbitrary number of controls. In this case, the underlying quantum operation is applied only if all the control qubits are in the state $|1\rangle$. The n-controlled-X gate that is the analogue of the CX gate with n control qubits can be written as

$$\mathrm{C}^n\mathrm{X} = \sum_{i=0}^{2^n-2} |i\rangle\langle i| \otimes I + |2^n - 1\rangle\langle 2^n - 1| \otimes X. \tag{1.19}$$

The 2-controlled-X gate is commonly named the Toffoli gate.

1.3.4 Quantum measurement

Considering a quantum system to be perfectly and completely isolated from the outside world and evolving unitarily does raise a major issue: how does an implementer is supposed to read the result of the quantum computation if the system the computation was performed on is not reachable nor observable?

In practice, in order to measure the result of any quantum computation, the quantum system used to perform the computation will need to be measured (or observed) by an external physical

system, a measurement device, that will make the result of the measurement available to the experimenter. This means that, when measuring a quantum state, the quantum system is no longer isolated and the quantum state might experience non-unitary transformations.

Postulate 4 (Quantum measurement or Born's rule)**.** *A given quantum measurement is represented by a set of* measurement operators $\{M_m\}$ *with* m *representing the index of the measurement outcome associated with the measurement operator* M_m*. For a given quantum state* $|\psi\rangle$*, the probability of measuring the outcome* m *is*

$$\langle\psi| M_m^\dagger M_m |\psi\rangle \tag{1.20}$$

and the quantum state after *the measurement is*

$$\frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m^\dagger M_m |\psi\rangle}}. \tag{1.21}$$

In order for the measurement operators to be physically meaningful, they should check

$$\sum_m M_m^\dagger M_m = I \tag{1.22}$$

where I *is the identity matrix. The above condition ensures that the measurement probabilities sum to* 1 *when considering all the possible outcomes* m*.*

For closed quantum systems, a special case of Postulate 4 is used to model the measurement operation: Projection-Valued Measurement (PVM) or projective measurement. A PVM is fully described by a hermitian operator M with spectral decomposition

$$M = \sum_m m P_m \tag{1.23}$$

where m is an eigenvalue of M and P_m a projector onto the eigenspace of M associated with m. The possible measurement outcomes for the measurement encoded by M are its eigenvalues m, and the probability of measuring the outcome m on the state $|\psi\rangle$ is

$$\langle\psi| P_m |\psi\rangle . \tag{1.24}$$

After the measurement, if the measured quantity was m, the state $|\psi\rangle$ becomes

$$\frac{P_m |\psi\rangle}{\sqrt{\langle\psi| P_m |\psi\rangle}}. \tag{1.25}$$

The most simple example of a PVM is the Pauli Z matrix, often denoted as σ_z. When explicitly writing down the spectral decomposition of the σ_z matrix

$$Z = \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = 1 \times \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + (-1) \times \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = |0\rangle\langle 0| - |1\rangle\langle 1| \tag{1.26}$$

we end up having a PVM with the measurement outcomes 1 and -1, projecting onto the sub-spaces created by the eigenvectors $|0\rangle$ and $|1\rangle$, i.e., the computational basis. This is the reason why the standard measurement in quantum computing hardware is often denoted as a Z measurement: the Z Pauli matrix is a PVM that models a measurement in the computational basis.

1.3.5 Non-closed quantum systems

In Section 1.3.4 we had to introduce briefly the notion of non-closed quantum system in order to model quantum measurement. But the assumption that a given quantum system can be perfectly isolated does not hold in practice due to the immense engineering effort it would require, if at all possible.

Non-closed quantum systems can be modelled by quantum states known as *mixed states*. A mixed state is essentially a probability distribution of pure quantum states. Mathematically, mixed quantum states are represented using the *density operator* formalism. Density operators are represented using density matrices, conventionally denoted by the Greek letter ρ, that are positive-definite, hermitian and of unit trace.

The density matrix representing a quantum state that is in the state $|\psi_i\rangle$ with a *classical* probability p_i is

$$\rho = \sum_i p_i \, |\psi_i\rangle\langle\psi_i| \, . \tag{1.27}$$

The density matrix representing the pure quantum state $|\psi\rangle$ is simply $\rho = |\psi\rangle\langle\psi|$.

The operations described in Sections 1.3.2 and 1.3.4 have to be re-defined for the density matrix formalism. Quantum gates are still represented as unitary matrices but *applying* a unitary transformation U to the quantum state represented by the density matrix ρ gives the density matrix

$$\rho' = U\rho U^\dagger. \tag{1.28}$$

Equivalently, quantum measurements can be re-framed into the density matrix formalism. Let M_m be measurement operators and $\mathrm{Tr}\,[\cdot]$ be the trace operation, the probability of obtaining the result m by measuring the quantum state ρ is

$$p(m) = \mathrm{Tr}\left[{M_m}^\dagger M_m \rho\right]. \tag{1.29}$$

After measuring m, the quantum state ρ becomes

$$\rho_m = \frac{M_m \rho {M_m}^\dagger}{\mathrm{Tr}\left[{M_m}^\dagger M_m \rho\right]}. \tag{1.30}$$

This definition completes the theoretical framework used in quantum computing. But before diving into the relationship between quantum computing and scientific computing in Chapter 2 we should mention the different models of computations that have been devised and used across the years and formalise a few visualisations that will be used across the manuscript.

1.4 Visual representations in quantum computing

Visual representations of computations are of prime importance in quantum computing as they are a very efficient way to represent high-level algorithms in a standard and well-known format that is very quick to parse visually. Even though writing down the algorithm used to generate the quantum computation is the most precise way of describing the said computation, the visual representation allows a high-level overview of the main steps of the algorithm implementation and as such is very much complementary to the written-down version.

In general, visualising a generic quantum state is a hard task. But being able to visualise quickly small (i.e., 1-qubit) quantum states may help in understanding how they evolve when quantum gates are applied on them, which proved to be a good tool to visualise dynamical-decoupling sequences and single-qubit decoherence.

In this manuscript, we use extensively the quantum circuit visualisation presented in Section 1.4.2. Moreover, Chapter 7 extensively use the 1-qubit state visualisation introduced in Section 1.4.1.

1.4.1 Visualisation of quantum states

As defined in Section 1.3.1, pure quantum states can be represented by a state vector. A n-qubit pure quantum state is represented by a vector of 2^n complex numbers. The total number of degrees of freedom of a n-qubit pure quantum state is

$$\mathrm{df}_{\mathrm{pure}}(n) = 2^{n+1} - 2 \tag{1.31}$$

as each of the 2^n complex number has 2 degrees of freedom, the unit-norm condition removes one degree of freedom and the fact that quantum states only differing by a global phase are equivalent also fixing one degree of freedom.

When considering mixed states the density matrix formalism is used, as defined in Section 1.3.5. A n-qubit mixed state is represented by a positive semi-definite, hermitian matrix of trace 1. Counting the degrees of freedom of such a matrix leads to

$$\mathrm{df}_{\mathrm{mixed}}(n) = 2^{2n} - 1 = 4^n - 1 \tag{1.32}$$

degrees of freedom.

Any visualisation attempting to represent faithfully a n-qubit pure (resp. mixed) quantum state should be able to represent its $2^{n+1} - 2$ (resp. $4^n - 1$) degrees of freedom in a readable way. This task quickly becomes unmanageable due to the exponential scaling of the number of degrees of freedom for both pure and mixed quantum state and the fact that visualising high-dimensional data is a hard problem as it requires to project the data to a 2- or 3-dimensional space and making sure that the projected data is easily interpretable. But even though representing large quantum states graphically seems to be an unbearable task, there exist ways to represent few-qubit quantum states.

Note 1. For $n = 1$, $\mathrm{df}_{\mathrm{pure}}(1) = 2$ and $\mathrm{df}_{\mathrm{mixed}}(1) = 3$. This means that any 1-qubit pure quantum state can be described by 2 independent parameters, and 3 independent parameters are needed for a generic 1-qubit mixed quantum state.

The Bloch sphere is probably one of the most well-known visualisation for $n = 1$ qubits. In order to define correctly the visualisation, let consider a generic 1-qubit *mixed* state represented by its density matrix ρ.

Theorem 1 (Pauli decomposition)**.** Any 2×2 density matrix ρ can be decomposed as

$$\rho = \frac{1}{2}\left(I + \vec{a} \cdot \vec{\sigma}\right) = \frac{1}{2}\left(I + a_x\sigma_x + a_y\sigma_y + a_z\sigma_z\right) \tag{1.33}$$

where $\vec{a} \in [-1, 1]^3$ is called the *Bloch vector* and should check $|\vec{a}| \leqslant 1$.

Possible values for the different coordinates of the Bloch vector are exactly defining the unit sphere in 3 dimensions, called the *Bloch sphere* in this context. Any mixed quantum state can be represented as a point, defined by its Bloch vector $\vec{a}$, within the Bloch sphere. Pure quantum states are represented by a Bloch vector $\vec{a}$ such that $|\vec{a}| = 1$, i.e., points on the surface of the Bloch sphere. An example of the Bloch sphere can be seen in Figure 1.1.

1.4.2 Representation of quantum computation

Quantum circuits, the element representing a computation in the gate-based model, is a convenient, easy-to-understand and standard representation of an actual quantum computation. It allows one to uniquely and unambiguously represent any quantum computation using the gate-based model.

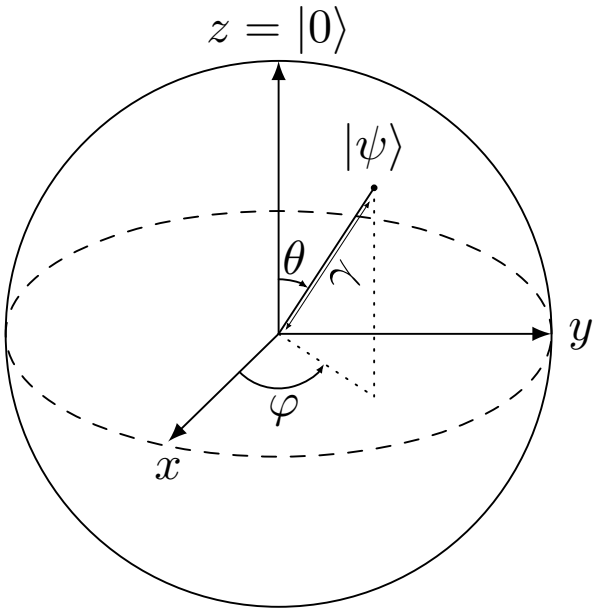


Figure 1.1: *Representation of a* 1*-qubit quantum state* $|\psi\rangle$ *on the Bloch sphere.* $\gamma = |\vec{a}| = \mathrm{Tr}\left[\rho^2\right]$ *is the purity of* $|\psi\rangle$*. For* $\gamma = 1$*, the quantum state* $|\psi\rangle$ *is pure.*

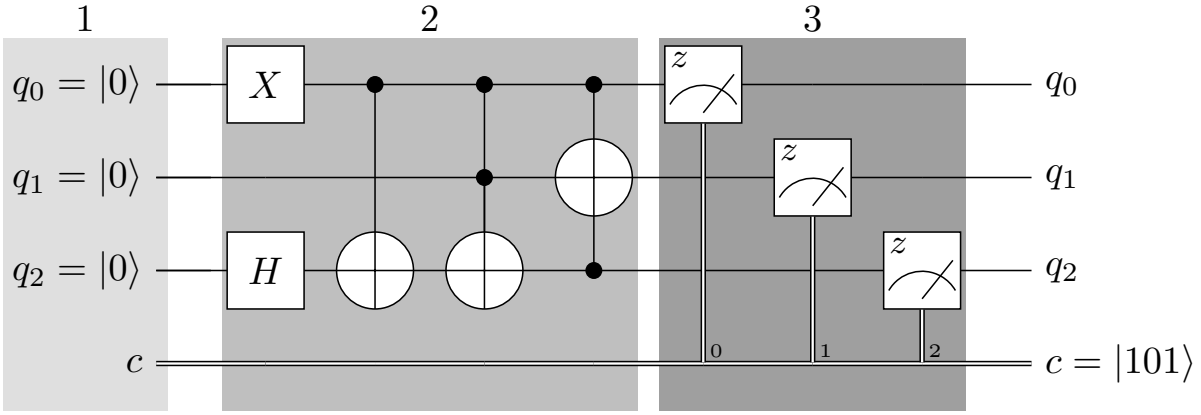


Figure 1.2: *Standard quantum circuit visualisation. The content of this image is described in Section 1.4.2.*

In this manuscript, we will mostly focus on the gate-based model of quantum computations presented in Section 1.2.3. Consequently, quantum computations will be represented using the quantum circuit formalism.

The standard visualisation used to represent a quantum circuit can be seen in Figure 1.2. In the standard representation of a quantum circuit, each classical and quantum bit is represented by an horizontal line. Classical (resp. quantum) bits are often denoted with the letter c (resp. q). The classical (resp. quantum) bits might be indexed, in which case the index is added near the qubit label, for example c_3 (resp. q_3). Classical or quantum bits that serve the same *purpose*, e.g., several bits that are part from the same register, might be grouped together as a unique horizontal line when there is no ambiguity.

Time evolves from left to right so qubits start in their initial state at the left-end of their corresponding line and operations applied on a specific line (i.e., qubit) have to be executed following their order from left to right. The operations ordering *across* qubits is not defined, meaning that an operation O_1 on a given qubit being "before" (i.e., to the left of) another operation O_2 on *a different* qubit does not imply that O_1 is executed before O_2.

In the first block of Figure 1.2, labelled 1, quantum register are all initialised in the state $|0\rangle$. Qubits might be named after their purpose, for example q_{in} for an input qubit or q_{carry} for a qubit that will store the carry of an half-adder. Explicitly writing the $|0\rangle$ state is not required as, by default, the qubits are supposed to be initialised in the $|0\rangle$ state. The initial value of classical bits (or registers) is also often striped out from the visualisation as classical register will most of the times be initialised when performing a quantum measurement.

The second block in Figure 1.2 contains quantum operations, also known as quantum gates. Quantum gates are represented by rectangles and are overlapping with the horizontal lines representing the qubits they act on. For example, the first quantum gate applied on qubit q_0 is the Pauli X gate. Similarly, the first gate applied to q_2 is a Hadamard gate H.

The third gate applied in the quantum circuit represents a controlled-X operation: a small filled black dot appears on the *control* qubit q_0 and the gate (here an alternative representation of the X gate) is applied on q_2. Multiply-controlled quantum gates are shown just after the controlled-X gate with two Toffoli (or doubly-controlled X) gates. The first Toffoli is controlled by q_0 and q_1 whereas the second Toffoli gate is controlled by q_0 and q_2.

Finally, the third and last block represents the measurement operations applied at the end of the quantum computation. A quantum measurement "gate" is always applied on a quantum and a classical bit (or register). Without any precision, measurements are performed in the computational basis (or Z basis). In order to avoid any ambiguity about the target classical bit of each quantum measurement, a small index may be added near the classical register horizontal line. In Figure 1.2, the result of the quantum measurement performed on q_0 is stored in the classical bit 0.

The final states of the classical or quantum bits might also be explicitly represented as in the quantum circuit visualisation example in Figure 1.2.

Chapter

2 Scientific computing and quantum computing

In this PhD we are interested by the potential applications of quantum computing to the scientific computing field. This chapter is dedicated to introduce a few representative problems that are studied within the scientific computing field. Most of the problems encountered in this field can be re-phrased using one out of three generic problems that are linear systems, partial differential equations and optimisation problems. These mathematical formulations are formalised and several classical algorithms used to solve them are presented in this chapter. Finally, we introduce some of the quantum algorithms that have been devised to solve these mathematical problems, starting by the special case of Hamiltonian simulation and following with quantum algorithms that can be applied to solve systems of linear equations, partial differential equations or optimisation problems.

Contents

2.1 Scientific computing

Scientific computing encompasses a variety of different research fields that all have in common their usage of computers to improve the understanding of complex problems and how to solve them. Since the inception of the field, a large number of problems have benefited from the power offered by computers. Thanks to its wide range of application and to the importance of the problems it is able to solve, scientific computing is nowadays a very attractive field of research that allows researchers and companies to push further the boundaries of what is possible. We present in Section 2.1.1 some applications that were made possible thanks to the advances of scientific computing. It turns out that most of these applications are instances of only a few mathematical problems that are presented in Section 2.1.2. Finally a summary of the state of the art of the hardware and tools used in the field is performed in Section 2.1.3.

2.1.1 Examples of applications scientific computing

The number of research problems that take advantage of the drastic increase of computing power we have seen in the last few decades or that are using this computing power today is substantial. The domains that have been heavily impacted range from weather forecast to financial optimisation, with applications in engineering, social sciences or astrophysics. Some of these applications are explained and developed in the following paragraphs.

Weather forecast

One of the most iconic examples of field that has been revolutionised by computers is probably weather forecast. The goal of weather forecast is to predict accurately the weather conditions (temperature, humidity, wind, rain, etc.) at a given physical location at a given time, having access to anterior weather data. The accurate prediction of such quantities involves solving complex physical process taking place in the atmosphere of the Earth and requires the collection and aggregation of large amounts of data on past weather conditions.

The complexity of the equations involved along with the accuracy requirements and the large amount of weather data that should be processed in order to predict the weather naturally led to a numerical approach of the problem. One of the first successful attempt at predicting the weather was performed on the ENIAC computer after at least one unsuccessful attempt at solving the equations by hand-calculation [21].

Since then, the accuracy of weather forecast has been tremendously improved thanks to the continuous increase of computing power available, the greatly enhanced quality of weather data collected by meteorological stations and new numerical methods introduced.

Numerical simulation in the aeronautic industry

Building fast, secure and efficient planes is huge challenge that involves solving multiple problems from radically different fields.

Due to the relatively high speed at which planes operate, they are particularly subject to friction forces that ultimately reduce their efficiency through a loss of energy. In order to minimise the effect of friction over the plane, its aerodynamics should be studied and improved. The cost of designing, building and testing each and every envisioned variations of a plane in order to determine which one result in the best aerodynamics (i.e., the less impact of friction forces on the plane) would be tremendous and it turns out that simulating the aerodynamics of the plane numerically is a way more affordable way.

But the study of aerodynamics is far from being the only problem encountered in aeronautic industries. Another source of improvement that have been extensively studied (and still is) concerns the engines efficiency. The more efficient an engine is, the less fuel it will need to consume in order to complete a flight, which directly translates into reduced costs, pollution and engineering to design a plane with a large enough tank.

This leads to yet another problem solved via numerical simulations: plane design and structure. The problem is to assert if a given plane design is able to withstand the physical constraints imposed on the plane at take-off, landing and during the flight.

Finally, scientific computing and the increased computing capabilities are also used to optimise the aircraft take-off, known as the “aircraft climb optimisation problem”, in which all the decisions from the take-off runway to the cruise phase are optimised to try to reduce to the minimum the overall consumption of fuel.

Structure optimisation and simulation

Aerodynamics and structural integrity studies are also central problems in other industries such as the automotive industry or in civil engineering.

Despite having a way lower operational speed than planes, cars are also subject to non-negligible aerodynamic friction forces which are detrimental to their efficiency. In addition to the study of their aerodynamic behaviour at different speeds, numerical simulations also help in planning the structural integrity the car and how its interior will be deformed in the event of an accident.

Computers are also used to perform structural integrity simulations of buildings or bridges in diverse situations such as earthquakes or strong wind. These simulations are able to check numerically if a given building design will be able to withstand earthquakes of a given intensity and how their structure will react to such an event, the final goal being to ensure that the building will not collapse if an earthquake happens. Numerical simulations are also used to modelise the effects of different wind profiles and intensity on the building by predicting the situations where the it will be able to dampen resonance behaviours correctly.

Process optimisation

Scientific computing can also be used to solve problems based purely on the optimisation of a quantity (i.e., that do not require the simulation of a physic system). A well known optimisation problem that has several applications nowadays is the recommendation problem that consists in providing suggestion for new content or items to a specific user, knowing its preference for other contents or items and in general the preferences of multiple users. This problem is encountered in many online streaming services when trying to optimise the time spent using the service and on e-commerce websites to increase the volume of items sold.

A lot of "public use" infrastructures also require to solve complex optimisation problems. For example, the problem of train scheduling can be reformulated as a large optimisation problem that consists in maximising the "value" of the train offer to users (i.e., having trains as frequently as possible while still being profitable and avoiding delays in case of problems) while obeying to constraints such as the minimum required safety distance between each trains, the maximum number of trains that can be stopped in a station at a given time, etc.

2.1.2 Mathematical problems encountered in scientific computing

Each of the problems or applications listed in Section 2.1.1 can be modelled as an instance of an abstract mathematical problem. This section introduces three of the most used mathematical problems in the field of scientific computing in general: systems of linear equation (Section 2.1.2), partial differential equations (Section 2.1.2) and general optimisation problems (Section 2.1.2).

Systems of linear equations

Solving a system of linear equations is a problem that arises frequently in scientific computing applications and that has been studied for decades. A system of linear equation is a set of one or more linear equations involving the same variables. A simple example is given in Equation (2.1).

$$\begin{cases} 3x + y = 0 \\ y + 3 = 0 \end{cases} \tag{2.1}$$

Generically, a system of linear equations with m linear equations and n variables can be written down as

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + b_1 = 0 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + b_2 = 0 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n + b_m = 0 \end{cases} \tag{2.2}$$

where the a_{ij} and b_i can be real, complex, or even more generic elements provided that the operations of addition and multiplication are well defined operations on these elements.

Systems of linear equations are often re-phrased using the linear algebra formalism of vectors and matrices. Using this formalism, the generic linear system in Equation (2.1) becomes

$$Ax + b = 0 \tag{2.3}$$

with

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \tag{2.4}$$

and

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}. \tag{2.5}$$

Partial differential equations

Partial different equations (PDEs) are probably one of the most used mathematical formalism to describe the evolution of any physical system independently of its size or complexity. In the formalism of PDEs, the state of the system studied is described by a function (often denoted by f, u or v) that verifies a set of equations involving its partial derivatives. The function values are unknown for most of its inputs, and the problems described by this PDEs requires to find the values of the function for some set of inputs.

One of the most simple example of partial differential equation describes how heat spreads in a conductive material with thermal diffusivity α. For a 3-dimensional material, the function $u(x, y, z, t)$ that gives the temperature at the point (x, y, z) at time t follows the following equation

$$\frac{\partial u}{\partial t} = \alpha \Delta u = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) \tag{2.6}$$

with Δ the differential operator often called "Laplacian".

In order for the partial differential equation to be well-defined, the values of the solution should be restricted for some sets of inputs, often called *initial conditions* or *boundary conditions* depending on the input variables used.

For example, the temperature of an empty room filled with a uniform gas (e.g., air) at an initial temperature $t_0 = 10\text{K}$, that has a thermal diffusivity of α, and with a discrete source of heat located at the point $(0, 0, 0)$ and initially at the temperature $T_0 = 1000\text{K}$ is given by the solution of Equation (2.6) equipped with the initial conditions

$$\begin{cases} u\left(0, 0, 0, 0\right) = T_0 \\ u\left(x, y, z, 0\right) = t_0, \ \forall \left(x, y, z\right) \end{cases}. \tag{2.7}$$

Another good example is the temperature of piece of another material M (e.g., metal) at an initial temperature $t_0 = 300\text{K}$ (i.e., 26.85°C which is approximately room temperature) and with a thermal diffusivity of α that is immersed into boiling water that always stay at temperature $T = 100°\text{C} = 373.15\text{K}$. The temperature evolution of the material M is given by the solution of Equation (2.6) equipped with the initial and boundary conditions

$$\begin{cases} u\left(x, y, z, 0\right) = t_0, \ \forall \left(x, y, z\right) \in M \\ u\left(x, y, z, t\right) = T, \ \forall t \geqslant 0, \ \forall \left(x, y, z\right) \notin M \end{cases}. \tag{2.8}$$

The heat equation shown in Equation (2.6) is one simple example of a PDE but there exist plenty of other, often more complex, PDE that describe the evolution of physical systems. Examples include PDEs describing fluid mechanics (Navier-Stokes equations), electric and magnetic fields (Maxwell's equations), the price evolution of a European call or put in the Black-Scholes financial model (Black-Scholes equation) or the evolution of systems abiding to the rules of quantum mechanics (Schrödinger equation).

Optimisation problems

Optimisation problems can be found everywhere, from finding the best path to take in order to go to a specific location to trying to optimise an industrial process. The most generic mathematical formulation for an optimisation problem is

$$x^* = \arg\min_{x \in X} f(x) \tag{2.9}$$

with X the space of possible solutions, f a function that returns a real value (often called "cost function") and $x^* \in X$ the optimum value that minimises the value of $f(x)$.

Optimisation problems are often split into two categories depending on whether the set of possible solutions X is discrete or continuous. If X is discrete then the optimisation problem is called a *combinatorial optimisation problem* or *discrete optimisation problem*. Else, if X is continuous (e.g., $[0, 1]$, $\mathbb{R}$ or $\mathbb{R}^8$), the optimisation problem is a *continuous optimisation problem*.

2.1.3 How are these problems solved on classical computers?

Most of the problems encountered in practical applications requires to solve an instance of one of the mathematical problems listed in Section 2.1.2. In practice, the problem instances are too large to be solved by manual computations and computers need to be used.

This is where scientific computing come into play and combine the compute power of classical computers, that are able to perform an incredible number of operations per seconds, with decades of research on algorithms and tools to implement, debug and benchmark these algorithms.

Classical computing algorithms

The first important piece that is needed to solve the problems described in Section 2.1.2 and consequently help solving applications listed in Section 2.1.1 are algorithms. Each of the problems described in Section 2.1.2 has been extensively studied, which led to a number of algorithms being devised to solve them. The following paragraphs introduce some of these algorithms.

Partial differential equations, as shown in Section 2.1.2, are often solved by approximating them as systems of linear equations. There are a variety of approximation methods, each with advantages and drawbacks that should be taken into account in order to ensure that the approximation performed is valid for the problem studied. Known approximations include finite differences, finite elements, finite volumes, multi-grid or spectral methods [22, Chapter 2]. Some algorithms have also been devised for specific fields such as computational fluid dynamics (CFD) that uses lattice Boltzmann methods or specific approximations for turbulent flows such as the Reynolds-Averaged Navier-Stokes (RANS) or Large Eddy Simulation (LES) methods [23].

As one of the central piece of scientific computing, systems of linear equations have been thoroughly studied and a plethora of algorithms exist to solve instances of this problem. The standard numerical method to solve a generic square system of linear equations is a version of Gaussian elimination that has been improved in order to avoid numerically unstable operations that can occur during the algorithm. If the same system of linear equations should be solved for several right-hand sides, it is more efficient to first perform the LU decomposition of the system's matrix, which costs as much as the Gaussian elimination algorithm but allows to solve

any subsequent systems with the same matrix much faster. Even though the method based on LU decomposition is able to solve any square system of linear equations, more efficient methods have been devised for special classes of systems. If the system's matrix is symmetric and positive definite then using a method based on Cholesky decomposition is twice as fast as using the LU decomposition. Also, using Levinson recursion is a more efficient algorithm to solve systems of linear equations represented by a Toeplitz matrix. Finally, for very large or sparse systems of linear equations, iterative methods are often used to approximate to a sufficient degree of precision the solution [22].

When considering optimisation problems, the distinction between *discrete* and *continuous* problems is fundamental. In fact the algorithms used to solve a discrete optimisation problem are radically different from the algorithms able to solve a continuous problem.

Continuous optimisation problems are characterised by a continuous set of possible solutions X that is most of the time considered to be a subset of $\mathbb{R}^n$. In this case, the cost function is defined as $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ and can be studied using well established results from calculus. In particular, the type of algorithm that may be used to solve the minimisation problem will depend on the ability to compute numerically the partial derivatives of f. If none of the partial derivatives of f can be numerically computed, the available optimisations algorithm will rely on heuristics methods. If the gradient of f can be obtained numerically, gradient descent algorithms or variation thereof can be used. Finally, if the Hessian (i.e., the second-order partial derivatives) of f can be numerically computed at an acceptable computational cost, the Newton algorithm may be used to minimise the value of f.

Discrete optimisation problems are more diverse than their continuous counterparts and do not fit a unique model, which lead to very diverse algorithms to solve them and even often problem-specific algorithms. This is for example largely the case on discrete problems defined on graphs, with for example the Dijkstra or A^* algorithms that are specifically built to solve the shortest path problem and cannot solve any other problem. The simplex algorithm is another famous algorithm, that has been specifically devised to solve linear programming problems.

Classical hardware, programming languages, libraries and compilers

The algorithms described in the previous section are theoretical constructions, very much like recipes that explain a succession of steps that have to be executed in the correct order to solve a specific problem.

The increasing size of numerical problems directly translates in an increase of the number of operations that are performed by these algorithms, up to numbers of operations that are no more realistically computable by a human. This led to the introduction of computers, machines that are able to perform operations at an incredible speed and with a very low probability of errors.

The ever-increasing need to solve bigger instances of the problems introduced in Section 2.1.2 pushed the computing power of computers to an incomparable level. Figure 2.1 shows the evolution of the computing power (in FLoating-point OPerations per Seconds, often abbreviated FLOPS or FLOP/s) of the 500 most powerful computers with publicly available benchmarks since 1993.

The increase of computing power witnessed in the last decades came with an increase of the computers physical and logical complexity, with the necessity to distribute computations over several thousands processing units that might have different characteristics. Due to the hardware complexity, a large number of tools have been devised to help performing computations efficiently on this kind of hardware.

One of the first abstraction and tool introduced to lower down the complexity of writing code for a specific hardware are compilers and programming languages. The introduction of these tools allowed each programmer to implement algorithms by using a "programming lan-

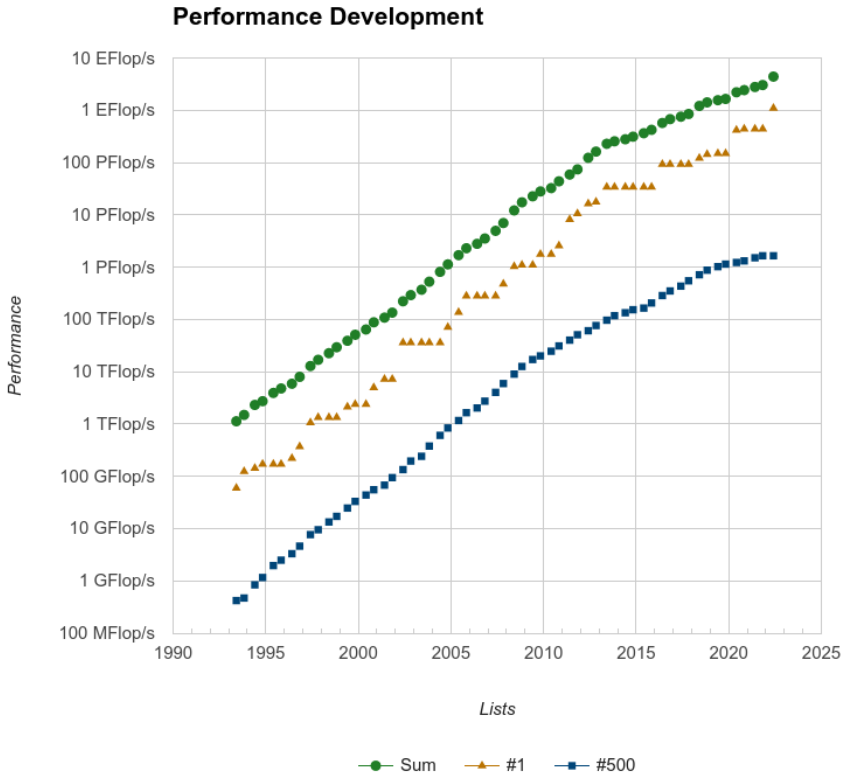


Figure 2.1: *Evolution of the compute power of computers in the TOP500 ranking from 1993 to 2022. Image obtained from* https://www.top500.org/statistics/perfdevel/ *the 13th of August 2022.*

guage", a more human-friendly way of describing computations than the machine language (or assembly language), and to feed the code written in this programming language to a "compiler", that will translate the human-friendly code into machine instructions. The separation between programming language and machine language has been a huge advance as it allows to write computations in a machine-agnostic language and leave nearly all the machine-specific considerations (translation to machine instructions, optimisation for the target hardware, ...) to an automated software, the compiler.

Another crucial abstraction that has been successfully used since several decades in classical computing is called "libraries". Libraries can be seen as a list of functions that can be reused by several different programs. Probably one of the most influential example is the Basic Linear Algebra Subroutines library, often called BLAS. The BLAS library stems from an effort made between 1970 and 1988 to implement routines performing linear algebra computations as efficiently as possible on the existing hardware. Thanks to its organisation as a library, any programmer was able to benefit from the optimised routines within BLAS.

Finally, a variety of tools to help developers have been gradually introduced. Examples include debuggers, tools used to understand the behaviour of a piece of code and follow its execution step by step to find potential bugs, profilers that are used to gather statistics on the classical program execution and report these statistics in a readable way, and even static code analysers that are designed to read the code in real-time and warn about any potential mistake, even before the compilation step.

2.2 Usage of quantum computing

The fact that scientific computing can have a significant impact on the efficiency of industrial process, might be able to save costs by optimising the allocation of some resources or even can

check the validity of a design before even starting producing it explains why it has attracted so much money and research from companies and countries. Computing power is in fact considered as a very valuable resource by many. Examples of significant amounts of money spent to build supercomputers can be found in abundance, for example with the United States Department of Energy that announced in 2018 that it will provide three exascale computers (i.e., capable of performing more than 10^{18} FLOPS) to three national laboratories for a budget of $400 to $600 millions per computer.

But it is becoming increasingly costly and difficult to scale these super-computers to higher compute power for several practical reasons that include the need for large amounts of energy (nearly 30MW for the Fugaku supercomputer [24]), the complexity and power of the cooling systems required to cool these supercomputers, the increasing communication costs between CPUs, etc. Moreover, most of the classical algorithms described in Section 2.1.3 requires resources that scale at best polynomially with the size of the inputs (e.g., the size of the linear system, the required precision for the discretisation of the PDE or the number of unknowns of the optimisation problem). This means that scaling to larger problem sizes will eventually become prohibitive in terms of cost and building even more powerful supercomputers will not be profitable anymore.

The field of quantum computing comes as a *potential* way of matching the demand for an increasing computing power and continue to scale up the size of the problems we would like to numerically solve. The following sections present some of the quantum algorithms that might be used to solve the mathematical problems discussed in Section 2.1.2.

2.2.1 Hamiltonian simulation

From all the partial differential equations that exist, the Schrödinger equation (presented in Equation (1.10)) holds a very special place in the field of quantum computing. As noted in Postulate 2, the Schrödinger equation governs how an isolated quantum system evolves over time. Due to the fact that quantum computers are considered to be a closed quantum system, they can be seen as machines that are only able to perform one specific task: solve the Schrödinger equation for a given Hamiltonian H and time t.

The special place held by this equation in quantum computing led to a large variety of algorithms aiming at solving the "Hamiltonian simulation" problem that is formalised in Problem 1 and that roughly consists in finding a way to evolve the quantum state represented by qubits according to a given Hamiltonian H for a time t.

Problem 1 (Hamiltonian simulation)**.** From a given Hamiltonian H, precision ϵ, evolution time t and gate set S, construct a quantum circuit C only containing gates from S and that implement a unitary U such that

$$\left|\left|U - e^{-iHt}\right|\right| \leqslant \epsilon \tag{2.10}$$

where $||\cdot||$ is the spectral norm.

Most of the algorithms devised to solve the Hamiltonian simulation problem only consider a restricted version of Problem 1. A typical restriction imposed in a majority of algorithms is to consider the Hamiltonian matrix H to be s-sparse (see Definition 5).

Definition 5 (s-sparse matrix)**.** A s-sparse matrix with $s \in \mathbb{N}^*$ is a matrix that has at most s non-zero entries per row and per column

Definition 6 (sparse matrix)**.** A sparse matrix is a s-sparse matrix with $s \in \mathcal{O}(\log(N))$, N being the size of the matrix.

One of the first algorithm devised to solve the s-sparse Hamiltonian simulation problem has been introduced in [25] and use a method known as Trotterisation that consists in splitting the

Hamiltonian H into a sum of simpler Hamiltonians and building the target evolution e^{-iHt} from the evolutions of the simpler Hamiltonians constructed $\left\{e^{-iH_j t'}\right\}_{1\leqslant j\leqslant m}$. Algorithms based on Trotterisation are explained in more details later in this manuscript, in Section 3.2.2.

The idea of using Trotterisation to solve the s-sparse Hamiltonian simulation problem has inspired several other algorithms. In [26], Childs and Kothari improve the automatic decomposition procedure devised in [25] in order to lower down the asymptotic complexity of the overall algorithm when a suitable decomposition of H is not known in advance. The order in which the Hamiltonian evolutions of each H_j are recomposed is also studied extensively in [27] where the authors use a divide and conquer approach to build groups of H_j matrices with similar norms and optimise the simulation within each group. Childs, Ostrander, and Su studied the possibility to randomise the order in which the evolutions are recomposed in [28] and shows that introducing randomness improves the expected asymptotic cost (or equivalently improve the expected precision obtained for a given cost) at the cost of having a probabilistic approach.

A central limitation on the asymptotic complexity of generic s-sparse Hamiltonian simulation algorithm states that no such algorithm can have a sub-linear asymptotic complexity in time. This result is obtained in [25, Theorem 3] and shows that any generic s-sparse Hamiltonian simulation algorithm valid for any time t can at best scale linearly in t (i.e., $\mathcal{O}(t)$). Algorithms based on Trotterisation are able to achieve a scaling in $\mathcal{O}\left(t^{1+\frac{1}{2k}}\right)$ for any $k \in \mathbb{N}^*$ at the cost of a multiplicative constant scaling exponentially with k in front of the asymptotic complexity. Additionally to the super-linear scaling with respect to t, algorithms based on Trotterisation all suffer from the theoretical complexity to devise tight generic bounds to ensure a given precision ϵ. The fact that the known bounds are loose has been shown in [29] where the authors compare the costs obtained with the analytical bounds available and the optimal cost found empirically.

The first algorithm to break the super-linear time scaling barrier is presented in [30]. It succeeded in lowering down the superlinear scaling in $\mathcal{O}\left(t^{1+\frac{1}{2k}}\right)$ of the methods based on Trotterisation to a linear scaling $\mathcal{O}(t)$ by using quantum walks.

A few years later, Berry et al. introduce in [31] a new algorithm that improves the asymptotic complexity of solving the Hamiltonian simulation problem with respect to the desired precision ϵ, changing the previously linear scaling in $\mathcal{O}\left(\frac{1}{\epsilon}\right)$ into a sub-logarithmic scaling $\mathcal{O}\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$. The exact same set of authors re-iterated by devising another algorithm, presented in [32], that achieve the same scaling with respect to $\frac{1}{\epsilon}$ but this time with a method based on truncated Taylor series. Another method introduced by Berry, Childs, and Kothari in [33] obtain again the same asymptotic complexity, this time by using an algorithm based on linear combinations of quantum walks. Along with the method, the authors also prove that the optimal asymptotic complexity of simulating a generic s-sparse Hamiltonian for a time t and with a precision ϵ cannot be greater than

$$\mathcal{O}\left(s||H||_{\max}t + \frac{\log\left(\frac{1}{\epsilon}\right)}{\log\log\left(\frac{1}{\epsilon}\right)}\right). \tag{2.11}$$

This optimal lower bound is reached by an algorithm presented in [34] and based on a new technique called "quantum signal processing". This optimal asymptotic complexity bounds was also obtained by another quantum algorithm based on Qubitization and introduced by the same authors in [35].

2.2.2 Systems of linear equations

As discussed in Section 2.1.2, the problem of solving a given system of linear equations is a central piece of scientific computing and as such it attracted a lot of interest from quantum algorithms researchers.

The first quantum algorithm to show an improvement over classical algorithms is the now famously known HHL (from the name of its authors Harrow, Hassidim, and Lloyd) algorithm [13]. By using Hamiltonian simulation (see Section 2.2.1), quantum phase estimation and amplitude amplification, the HHL algorithm is able to solve a sparse square system of N linear equations with N unknowns in a number of operations that scales as $\mathcal{O}\left(s^2\kappa^2 \log\left(N\right)/\epsilon\right)$ where κ is the condition number of the linear system matrix, s is the maximum number of non-zero elements on each column and ϵ is the desired precision.

The asymptotic complexity of the HHL algorithm brought hope that quantum computing might be a good candidate to continue scaling the problem sizes up as the number of resources needed to solve a given linear system with the HHL algorithm grows *logarithmically* with its size. This means that, in order to solve a system of linear equations with twice as many equations and unknowns, only a *constant* number of additional quantum resources have to be available.

Even-though the HHL algorithm showed that it was theoretically possible to solve sparse square systems of linear equations in a number of operations that scales logarithmically with the number of equations, the quadratic dependence on the condition number κ is less than ideal knowing that classical algorithms based on the conjugate gradient method scale linearly with respect to κ. Ambainis improved the asymptotic scaling with respect to the condition number κ at the cost of an increased dependence on the desired precision ϵ with an algorithm using variable-time amplitude amplification that has an asymptotic complexity of $\mathcal{O}\left(\frac{\kappa \log^3(\kappa/\epsilon)}{\epsilon^3} \log^2\left(\frac{1}{\epsilon^2}\right)\right)$ [36].

An improvement to the original HHL algorithm has then been introduced by [37] in which the matrix representing the system of linear equations is preconditioned to lower down its condition number. The quantum algorithm introduced is able to find a good preconditioning matrix M and solve the preconditioned system of linear equations $(MA)\,x = Mb$ with an asymptotic complexity of

$$\mathcal{O}\left(d^7 \kappa \log\left(N\right)/\epsilon^2\right) \tag{2.12}$$

where d is the number of nonzero elements per row of a specific least-square problem introduced in [37, Eq. (11)]. Alongside the dependence on the condition number of the matrix κ, the authors Clader, Jacobs, and Sprouse also raise another few potential problems of the HHL algorithm that might reduce its usability on some linear systems, such as the need to be able to prepare the right-hand side $|b\rangle$ into a quantum register and the limitations imposed by the fact that the solution is stored in the amplitudes of a quantum state, which limits the way the solution can be read.

These two limitations, that are fundamental to quantum computing, along with the limitation imposed by the linear scaling of the HHL algorithm with respect to κ and the limited set of matrices that can be efficiently used within the framework of the HHL algorithm are succinctly and nicely summarised by Aaronson in [38].

Subsequent works successfully improved the runtime of the HHL algorithm, first by improving the asymptotic scaling with respect to the desired precision ϵ from $\mathcal{O}\left(\epsilon^{-1}\right)$ to $\mathcal{O}\left(\text{poly}\log\left(\epsilon^{-1}\right)\right)$ using an approach known as Linear Combination of Unitary originally introduced in Hamiltonian simulation algorithms [39]. A new algorithm suitable for dense systems of linear equations has then been devised in [40] by using a quantum singular value estimation subroutine introduced by Kerenidis and Prakash in [41]. A few other quantum algorithms to solve systems of linear equations have been introduced in the recent years [42–45] without significant advances in terms of asymptotic complexity.

Finally, several variational quantum algorithms have been devised to solve systems of linear equations [46–48]. These algorithms rely on classical optimisers to minimise a cost function whose global minimum represent the solution to the system of linear equations encoded. Their theoretical scaling has not been found yet as it is highly dependent on the cost function properties and the optimisation algorithm used.

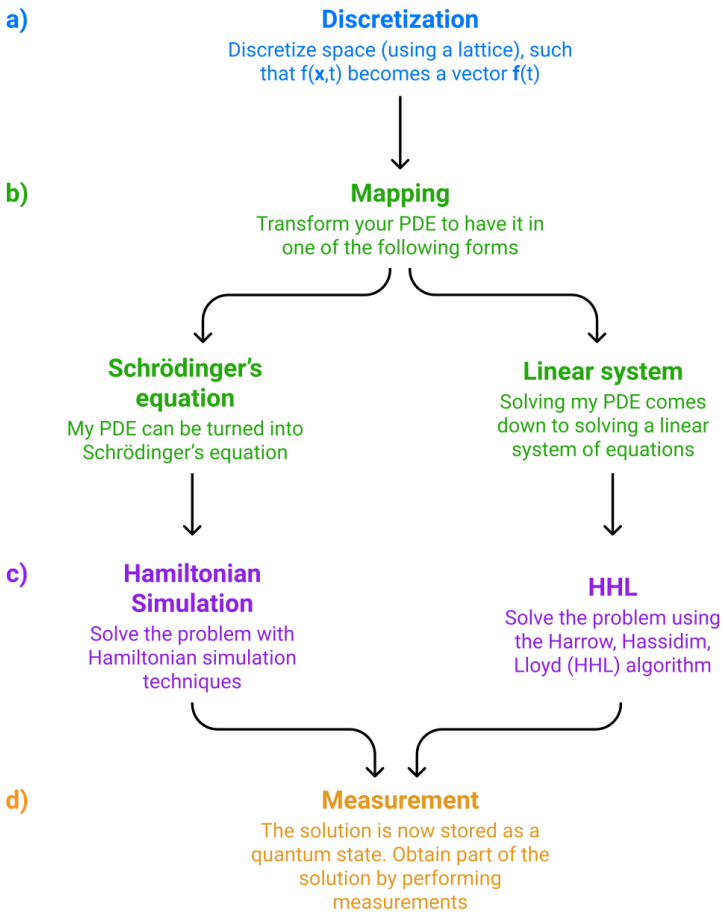


Figure 2.2: *Most common steps followed to solve a given PDE by using a quantum algorithm. Image obtained from [49].*

2.2.3 Partial differential equation solvers

Most quantum algorithms devised to solve partial differential equations start by discretising the solution space and then follow one of two paths: either they find a way to reformulate the PDE into a Schrödinger equation and then use a Hamiltonian simulation algorithm to solve the resulting rephrased problem or they solve the system of linear equations obtained after discretising the PDE by using the HHL algorithm or an improvement thereof. These two different paths are well illustrated in Figure 2.2.

One of the first quantum algorithm devised to solve differential equations has been introduced by Leyton and Osborne in [50] and uses the first method that consists in rephrasing the problem as an instance of the Schrödinger equation. It is able to solve systems of n non-linear ordinary differential equations (ODE), a specialisation of PDE where the unknown function has only 1 input variable, using a number of operations that scale polylogarithmically with the number of equations $\mathcal{O}(\text{poly}\log n)$ at the cost of an exponential dependence on the input variable of the n unknown functions.

The exponential scaling with the input variable of [50] is not acceptable for practical applications and [51] improves this to a quadratic scaling for first-order linear differential equations of the form

$$\frac{d}{dt}x(t) = A(t)x(t) + b(t) \tag{2.13}$$

where x and b are functions returning a N-component vectors and $A(t)$ is a sparse $N \times N$ matrix. The algorithm works by discretising the first-order linear differential equation by using

a multi-step approach in order to reformulate the problem as a system of linear equations, solved by the HHL algorithm.

Another quantum algorithm able to solve generic partial differential equations is due to Berry et al. in [52]. The algorithm is able to reach a polylogarithmic scaling with respect to $\frac{1}{\epsilon}$ by applying the linear combination of unitaries (LCU) algorithm to solve a linear system constructed from a truncated Taylor serie obtained by writing down the analytic solution to Equation (2.13) when A and b are time independent.

Additionally, several quantum algorithms have been devised to solve particular types of partial differential equations. These include the Poisson equation [53–55], the wave equation [56] (studied in details in Chapter 3), equations from plasma physics [57, 58], the Navier-Stokes equation [59, 60] or even the Black-Scholes equation [61].

Quantum algorithms based on the finite element [62] or spectral [63] methods have also been devised. They that use well known classical schemes to reformulate the problem as a system of linear equations that can be then solved using an efficient version of the HHL algorithm.

Finally, as for systems of linear equations, variational algorithms to solve partial differential equations have been introduced in [64, 65].

2.2.4 Quantum algorithms for optimisation

As noted in Section 2.1.2, optimisation problem can be split into two categories depending on whether they are discrete or continuous. It turns out that most of the literature about quantum algorithms to solve optimisation problem is interested in discrete (or combinatorial) optimisation problems.

A first notable class of problem that has been studied and for which quantum algorithms exist is semi-definite programming (SDP). Semi-definite programming optimisation problems can be written as

$$\begin{aligned} &\min_{x_1,\dots,x_n\in\mathbb{R}^n} \sum_{(i,j)\in[\![1,n]\!]} c_{i,j}\left\langle x^i, x^j\right\rangle \\ &\text{subject to } \sum_{(i,j)\in[\![1,n]\!]} a_{i,j,k}\left\langle x^i, x^j\right\rangle \leqslant b_k, \text{ for all } 1\leqslant k\leqslant m \end{aligned} \tag{2.14}$$

where $c_{i,j}$, $a_{i,j,k}$ and b_k are all real numbers and $\langle\cdot,\cdot\rangle$ is the scalar product.

The first quantum algorithm solving semi-definite programming problems and showing a polynomial speed-up over its classical counterparts has been presented in [66]. In this paper, Brandao and Svore used ideas from a classical algorithm along with the amplitude amplification quantum algorithm to obtain a quadratic asymptotic improvement in the preparation of Gibbs states on a quantum computer, which led to a quantum method quadratically faster than the classical version. New quantum algorithms, also inspired by classical optimisation algorithms, have been introduced in the following years, improving the asymptotic complexity [67–70]. New algorithms based on interior point methods have also been introduced in [71, 72], achieving the best asymptotic complexity to date.

Quadratic unconstrained binary optimisation (QUBO) problems are natively solved by quantum annealers, quantum computers following a relaxed version of the adiabatic quantum computing model of computation presented in Section 1.2.1. A QUBO problem consists in solving

$$x^* = \arg\min_{x\in\{0,1\}^n} \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}x_i x_j \tag{2.15}$$

with $q_{ij}\in\mathbb{R}$ for $1\leqslant i\leqslant j\leqslant n$. Many problems from combinatorial optimisation have been reformulated in the QUBO formalism as shown in [73] for several NP-complete problems and in [74] for industrial problems.

Part II

Algorithm implementation

Chapter

3 PDE solver

This chapter focuses on an implementation of a particular "monolithic" quantum algorithm that solves a partial differential equation. This chapter, that originates from [75], explains in great details the implementation from scratch of a partial differential equation solver on a quantum computer. This implementation has been the first step in defining what would be the requirements and the challenges to define and implement a *QBLAS* library.

Contents

3.1 Problem considered

3.1.1 Type of problems

In order to be able to understand the needs for re-usable routine definitions and highly optimised implementations of such routines, we decided to implement a non-trivial quantum algorithm from scratch, with the frameworks and tools provided. Several problems of interest attracted our attention during the initial study of the existing algorithms.

First we considered the ubiquitous problem of solving (sparse) linear systems. This is a particularly appealing problem for our goal as it is omnipresent in the field of scientific computing and might require linear algebra subroutines that are especially interesting. Finally, the HHL algorithm, named after its inventors Harrow, Hassidim, and Lloyd and introduced in [13], claims

to solve efficiently this problem on a quantum computer and enters in the category of "complex" algorithms, i.e., algorithms that are estimated to be non-trivial to implement in practice.

Another highly appealing class of problem that is encountered massively in a large number of different fields is partial differential equations. We ended up studying this class of problem rather than the quantum linear system solver for several reasons. First, an initial literature review showed that the already existing algorithms to implement a quantum solver for partial differential equations was more diversified, with several published and peer-reviewed algorithms [25, 50, 51, 53, 76]. Secondly, one of our auxiliary goals was to explore partial different equation solvers for quantum computers, which could have been done with a linear system solver by using discretisation schemes but was less straightforward than directly implementing a partial differential equation solver. Finally, a very detailed study of a linear system solver has already been done previously [77], which was not the case for a partial differential equation solver, Hamiltonian simulation excluded [78].

3.1.2 Choice of the PDE

Now that the problem of interest has been established, we still have to decide which partial differential equation we will solve. One of the main criteria to chose a partial differential equation over another in this context is its simplicity: the simpler the partial differential equation is, the more straightforward it will be to implement a solver. This work being the first implementation from scratch of a partial differential equation solver using quantum technologies, we wanted to start as simple as possible.

The need to have a "simple" partial differential equation already rules out a few very interesting candidates such as the Navier-Stokes or Black-Scholes equations. The "go-to" partial differential equation when dealing with quantum computers is the Schödinger equation. As explained in Section 1.3.2, any closed quantum system is supposed to evolve according to Equation (1.10). As such, even though the Schödinger equation is considered as complex in classical computing, it is one of the most suited for a quantum computer. The problem of solving the Schrödinger equation is called Hamiltonian simulation and has been presented in Section 2.2.1 and Problem 1.

There exists a plethora of quantum algorithms to solve the Hamiltonian simulation problem [25, 30, 32, 34, 79–88] as discussed in Section 2.2.1. Each algorithm has its advantages and drawbacks, some of them being theoretically non-optimal but easier to implement in practice while others achieve an optimal asymptotic complexity but require a major amount of work to implement. A complete study on different implementations of Hamiltonian simulation algorithms was already published in [78] and the boundary and initial conditions that can be meaningfully used with the Schrödinger equation are quite different from those found in other partial differential equations.

For this reason, we considered a more "standard" partial differential equation: a simplified version of the wave equation on the 1-dimensional line $[0, 1]$ where the propagation speed c is constant and equal to 1. This equation can be written as

$$\frac{\partial^2}{\partial t^2}\phi(x,t) = \frac{\partial^2}{\partial x^2}\phi(x,t). \tag{3.1}$$

Moreover, we only considered solving Equation (3.1) with the Dirichlet boundary conditions

$$\frac{\partial}{\partial x}\phi(0,t) = \frac{\partial}{\partial x}\phi(1,t) = 0. \tag{3.2}$$

No assumption is made on initial speed $\phi(x,0)$ and initial velocity $\frac{\partial\phi}{\partial t}(x,0)$, even though in practice they will need to be "efficiently preparable" in a quantum state (see Definition 7) in order to hope having any kind of quantum speed-up.

Definition 7 (Efficiently preparable quantum state)**.** A n-qubit quantum state $|\psi\rangle$ is said to be "efficiently preparable" is there exist a quantum circuit C implementing a unitary U_C such that

1. $U_C |0\rangle = |\psi\rangle$, i.e., the quantum circuit C prepares $|\psi\rangle$.
2. The number of quantum gates needed to implement C is in $\mathcal{O}(\text{poly}(n))$.
3. The number of ancilla qubits needed to implement C is in $\mathcal{O}(\text{poly}(n))$.

The resolution of this simplified wave equation on a quantum computer is an appealing problem for the first implementation of a PDE solver for several reasons. First, the wave equation is a well-known and intensively studied problem for which a lot of theoretical results have been verified. Secondly, the difficulty in solving the wave equation seems well balanced and checks our requirements of being simple while not being trivially solvable. Finally, the theoretical implementation of a quantum wave equation solver has already been studied in [56].

In this chapter, we present the complete implementation of a 1-dimensional wave equation solver using quantum technologies based on `qat` library. To the best of our knowledge, this work was the first to consider the implementation of an entire PDE solver that can run on a quantum computer. Specifically, we explain all the implementation details of the solver from the mathematical theory to the actual quantum circuit used. The characteristics of the solver are then discussed and analysed, such as the estimated gate count and estimated execution time on real quantum hardware. We show that the implementation follows the theoretical asymptotic behaviours devised in [56]. Moreover, the wave equation solver algorithm relies critically on an efficient implementation of a Hamiltonian simulation algorithm, which we have also implemented and analysed thoroughly.

3.2 Implementation

The algorithm used to solve the wave equation is explained in [56] and uses a Hamiltonian simulation procedure. Costa, Jordan, and Ostrander chose the Hamiltonian simulation algorithm described in [33] for its nearly optimal theoretical asymptotic behaviour. But even-though nearly optimal in theory, this algorithm is very complex to implement in practice, which might translate to high constants hidden in the big-O notation. For this reason, we privileged instead the Hamiltonian simulation procedure explained in [25, 79] for its very good experimental results based on [78] and its simpler implementation (detailed in Section 3.2.2).

The code has been written using `qat`, a Python library shipped with the Quantum Learning Machine (QLM), a package developed and maintained by Atos. It has not been extensively optimised yet, which means that there is still a large room for possible improvements.

All the circuits used in this paper have been generated with a subset of `qat`'s gate set:

$$\{H, X, R_y(\theta), P_h(\theta), CP_h(\theta), CNOT, CCNOT\} \tag{3.3}$$

with H and X as defined in Equations (1.11) and (1.12), $R_y(\theta) = e^{-i\theta Y}$, $P_h(\theta) = e^{i\theta} e^{-iZ\theta}$, and gates with the prefix C being controlled versions of the gate after the prefix. The quantum circuits have then been translated to a realistic hardware gate set to study their scaling in the most realistic setting possible by using the `qat` library, to the gate set

$$\{U_1(\lambda), U_2(\lambda, \phi), U_3(\lambda, \phi, \theta), CNOT\} \tag{3.4}$$

for U_1, U_2 and U_3 defined in Equation (7) of [89] as follow:

$$U(\lambda, \phi, \theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\theta}{2}\right) & e^{i(\lambda+\phi)}\cos\left(\frac{\theta}{2}\right), \end{pmatrix} \tag{3.5}$$

$$U_3(\lambda, \phi, \theta) = U(\lambda, \phi, \theta), \tag{3.6}$$

$$U_2(\lambda, \phi) = U\left(\frac{\pi}{2}, \lambda, \phi\right), \tag{3.7}$$

$$U_1(\lambda) = U(0, 0, \lambda). \tag{3.8}$$

The gate set presented in Equation (3.4) has been used extensively by IBM to represent the native gate set of their quantum chips. It does not correspond to the native gate set anymore. The native gate set implemented by IBM hardware has been explained in [89]. The choice of using the gate set from Equation (3.4) is justified by the fact that, when the research was performed, IBM only provided hardware characteristics such as gate times for the gate set of Equation (3.4) and not for the real hardware gate set.

This implementation aims at validating in practice the theoretical asymptotic complexities of Hamiltonian simulation algorithms and providing a proof-of-concept showing that it is possible to solve a partial differential equation on a quantum computer. We also study extensively the resource requirements of the wave equation solver implementation in a setup that is as close as possible to current quantum hardware.

We start by presenting the Hamiltonian simulation algorithm used in the implementation in Sections 3.2.1 and 3.2.2. We then explain how Equation (3.1) is solved in Section 3.2.3 and detail the simulated Hamiltonian along with the implementation of the required oracles in Sections 3.2.4 and 3.2.5.

3.2.1 Sparse Hamiltonian simulation algorithm

In the past years, a lot of algorithms have been devised to simulate the effect of a Hamiltonian on a quantum state [25, 30–34, 39, 80, 81, 85–87]. Among all these algorithms, only few have already been implemented for specific cases [90, 91] but to the best of our knowledge no implementation is currently capable of simulating a generic sparse Hamiltonian.

The domain of application of the already existing methods being too narrow, we decided to implement our own generic sparse Hamiltonian simulation procedure. We based our work on the product-formula approach described in [25, 79]. One advantage of this approach is that product-formula based algorithms have already been thoroughly analysed both theoretically [25, 79] and practically [77, 78], and several implementations are publicly available, though restricted to Hamiltonians that can be decomposed as a sum of tensor products of Pauli matrices. Moreover, [79] provides a lot of implementation details that allowed us to go straight to the development step.

Our implementation is capable of simulating an arbitrary sparse Hamiltonian provided that it has already been decomposed into a sum of 1-sparse hermitian matrices with either only real or only complex entries, each described by an oracle. The implementation has been validated with several automated tests and a more complex case involving the simulation of a 2-sparse Hamiltonian and described in Section 3.2.3. Furthermore, it agrees perfectly with the theoretical complexities devised in [25, 79] as studied and verified in Section 3.3.

3.2.2 Product-formula implementation details

Hamiltonian simulation

Hamiltonian simulation is the problem of constructing a quantum circuit that will evolve a quantum state according to a Hamiltonian matrix, following the Schrödinger equation, as shown in Problem 1. The s-sparse Hamiltonian simulation problem is a specialisation of Problem 1 when the Hamiltonian H is s-sparse as defined in Definition 5.

Several quantum algorithms have been developed in the last few years to solve the problem of s-sparse Hamiltonian simulation [25, 30–34, 39, 80, 81, 85–87]. Among these algorithms we decided to implement the product-formula approach [25, 79], for the reasons presented in Section 3.2.1.

The product formula algorithm has three main steps: decompose, simulate, recompose. It works by first decomposing the s-sparse Hamiltonian matrix H that should be simulated as a sum of hermitian matrices H_j that are considered easy to simulate

$$H = \sum_{j=0}^{m-1} H_j. \tag{3.9}$$

The second step is then to simulate each H_j separately, i.e. to create quantum circuits implementing $e^{-iH_j t}$ for all the H_j in the decomposition in Equation (3.9). The last step uses the simulations computed in step two to approximate e^{-iHt}.

The very first questions that should be answered before starting any implementation of the product-formula algorithm are "What is an easy to simulate matrix?" and "What kind of hermitian matrices are easy to simulate?".

Easy to simulate matrices

One of the most desirable properties for an "easy to simulate" matrix is the possibility to simulate it exactly, i.e. to construct a quantum circuit that will perfectly implement e^{-iHt}. This property becomes a requirement when one wants rigorous bounds on the error of the final simulation. Another enviable property of these matrices is that they can be simulated with a low gate number and only a few calls to the matrix oracle.

Definition 8 (Easy to simulate matrix)**.** A hermitian matrix H can be qualified as "easy to simulate" if there exist an algorithm that takes as input a time t and the matrix H and outputs a quantum circuit $C(H)_t$ such that

1. The quantum circuit $C(H)_t$ implements exactly the unitary transformation e^{-iHt}, i.e.
$$\left|\left|e^{-iHt} - C(H)_t\right|\right| = 0.$$
2. The algorithm only needs $\mathcal{O}(1)$ calls to the oracle of H and $\mathcal{O}(\log N)$ additional gates, N being the dimension of the matrix H.

With this definition of an "easy to simulate" matrix, we can now search for matrices or group of matrices that satisfy this definition.

Multiples of the identity The first and easiest matrices that fulfil the easy to simulate matrix requirements are the multiples of the identity matrix $\{\alpha I, \alpha \in \mathbb{R}\}$ with I the identity matrix. The quantum circuit to simulate this class of matrices can be found in [92].

Integer-weighted, 1-sparse, hermitian matrices A larger class of matrices that can be efficiently and exactly simulated are the 1-sparse, integer weighted, hermitian matrices. Quantum circuits simulating exactly 1-sparse matrices with integer weights can be found in [79].

Note 2. Procedures simulating 1-sparse matrices with real (non-integers) weights are also described in the paper, but these matrices do not fall in the "easy to simulate" category because the procedures explained are exact only if all the matrix weights can be represented exactly with a fixed-point representation, which is not always verified.

Note 3. Multiples of identity matrices presented in Section 3.2.2 are a special case of 1-sparse matrices. The two classes have been separated because more efficient quantum circuits exists for αI matrices and these algorithms are exact for any coefficient $\alpha \in \mathbb{R}$.

Decomposition of H

Once the set of "easy to simulate" matrices has been established, the next step of the algorithm is to decompose the s-sparse matrix H as a sum of matrices in this set.

There are two possible ways of performing this decomposition, each one with its advantages and drawbacks: applying a procedure computing the decomposition automatically, or decompose the matrix H beforehand and provide the decomposition to the algorithm.

The first solution, which is to automatically construct the oracles of the H_j matrices from the oracle of the H matrix has been studied in [79] and [81]. Thanks to this automatic decomposition procedure, we only need to implement one oracle. This simplicity comes at the cost of a higher gate count: each call to the automatically constructed oracles of the matrices H_j will require several calls to the oracle of H along with additional gates.

On the other hand, the second solution offers more control at the cost of less abstraction and more work. The decomposition of H is not automatically computed and should be performed beforehand. Once the matrix H has been decomposed as in Equation (3.9), the oracles for the matrices H_j should be implemented. This means that we should now implement m oracles instead of only 1 for the first solution. The main advantage of this method over the one using automatic-decomposition is that it gives us more control, a control that can be used to optimise even more the decomposition of Equation (3.9) (less H_j in the decomposition, H_j matrices that can be simulated more efficiently, ...).

All the advantages and drawbacks weighted, we chose to implement the second option for several reasons. First, the implementation of the automatic decomposition procedure adds a non-negligible implementation complexity to the whole Hamiltonian simulation procedure. Moreover, the automatic decomposition procedure can be implemented afterwards and plugged effortlessly to the non-automatic implementation. Finally, our use-case only required to simulate a 2-sparse Hamiltonian that can be decomposed as the sum of two 1-sparse, easy to simulate, hermitian matrices, which makes the manual decomposition step manageable.

Simulation of the H_j

Once the matrix H has been decomposed following Equation (3.9) with each H_j being an "easy to simulate" matrix, the simulation of H_j becomes a straightforward application of the procedures described in Section 3.2.2.

After this step, we have access to quantum circuits implementing e^{-iH_jt} for $j \in [0, m-1]$ and $t \in \mathbb{R}$.

Re-composition of the e^{-iH_jt}

The ultimate step of the algorithm is to approximate the desired evolution e^{-iHt} with the evolutions e^{-iH_jt}. In the special case of mutually commuting H_j, this step is trivial as it boils down to use the properties of the exponential function on matrices and write $e^{iHt} = e^{i\sum_j H_jt} = \prod_j e^{iH_jt}$. But in the more realistic case where the matrices H_j do not commute, a more sophisticated method should be used to approximate the evolution e^{-iHt}. To this end, we used the first-order Lie-Trotter-Suzuki product formula defined in Definition 9.

Definition 9 (Lie-Trotter-Suzuki product formula [78, 93, 94])**.** The Lie-Trotter-Suzuki product formula approximates

$$\exp\left(\lambda \sum_{j=0}^{m-1} \alpha_j H_j\right) \tag{3.10}$$

with

$$S_2(\lambda) = \prod_{j=0}^{m-1} e^{\alpha_j H_j \lambda/2} \prod_{j=m-1}^{0} e^{\alpha_j H_j \lambda/2} \tag{3.11}$$

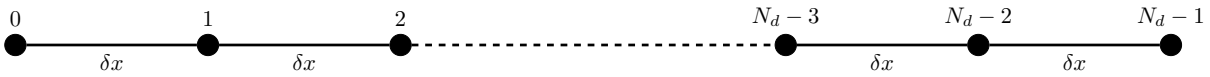


Figure 3.1: *Graph $G_{\delta x}$ built from the discretisation of the 1-dimensional line $[0, 1]$ with N_d discretisation points (i.e. $\delta x = \frac{1}{N_d - 1}$).*

and can be generalised recursively to higher-orders

$$S_{2k}(\lambda) = \left[S_{2k-2}(p_k\lambda)\right]^2 \times S_{2k-1}((1 - 4p_k)\lambda) \times \left[S_{2k-2}(p_k\lambda)\right]^2 \tag{3.12}$$

with $p_k = \left(4 - 4^{1/(2k-1)}\right)^{-1}$ for $k > 1$. Using this formula, we have the approximation

$$e^{\lambda H} = \left[S_{2k}\left(\frac{\lambda}{n}\right)\right]^n + \mathcal{O}\left(\frac{|\lambda|^{2k+1}}{n^{2k}}\right). \tag{3.13}$$

We used the Lie-Trotter-Suzuki product formula with $\lambda = -it$ to approximate the operator e^{-iHt} up to an error of $\epsilon \in \mathcal{O}\left(\frac{t^{2k+1}}{n^{2k}}\right)$.

3.2.3 Quantum wave equation solver

Using the Hamiltonian simulation algorithm implementation, we successfully implemented a 1-dimensional wave equation solver using the algorithm described in [56] and explained in Sections 3.2.4 and 3.2.5.

For the specific case considered (Equations (3.1) and (3.2)), solving the wave equation for a time T on a quantum computer boils down to simulating a 2-sparse Hamiltonian for a time $f(T)$, the function f being thoroughly described in [56] and Equation (3.55). The constructed quantum circuit can then be applied to a quantum state representing the initial position $\psi(x, 0)$ and velocity $\frac{\partial\phi}{\partial t}(x, 0)$, and will evolve this state towards a quantum state representing the final position $\phi(x, T)$ and velocity $\frac{\partial\phi}{\partial t}(x, T)$.

As for the Hamiltonian simulation procedure, the practical results we obtain from the implementation of the quantum wave equation solver seems to match the theoretical asymptotic complexities. See Section 3.3 for an analysis of the theoretical asymptotic complexities.

3.2.4 Hermitian matrix construction and decomposition

One of the main challenge in implementing a quantum wave equation solver lies in the construction and implementation of the needed oracles. This section describes the first step of the implementation process: the construction and decomposition of the Hamiltonian matrix that will be simulated using the Hamiltonian simulation procedure introduced in Section 3.2.2.

This section follows the analysis performed in [56] and adds details and observations that will be refereed to in Section 3.2.5 when dealing with the actual oracle implementation.

Hamiltonian matrix description

In order to devise the Hamiltonian matrix that should be simulated to solve the wave equation, the first step is to discretise Equation (3.1) with respect to space. Such a discretisation can be seen as a graph $G_{\delta x}$ whose vertices are the discretisation points and with edges between nearest neighbour vertices. The graph $G_{\delta x}$ is depicted in Figure 3.1.

The graph Laplacian of $G_{\delta x}$, defined as

$$L(G_{\delta x})_{i,j} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } (i \neq j) \wedge (v_i \text{ adjacent to } v_j) \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

can then be used to approximate the differential operator $\frac{\partial^2}{\partial x^2}$. By using the discretisation approximation

$$\frac{\partial^2 \phi}{\partial x^2}(i\delta x, t) \approx \frac{\phi_{i-1,t} - 2\phi_{i,t} + \phi_{i+1,t}}{\delta x^2} \tag{3.15}$$

with $\phi_{i,t} = \phi(i\delta x, t)$, and approximating $\phi(x,t)$ with a vector $\phi = [\phi_{i,t}]_{0 \leqslant i < N_d}$, the matrix

$$A = -\frac{1}{\delta x^2} L(G_{\delta x}) \tag{3.16}$$

approximates the second derivative of ϕ when $\delta x \to 0$ as

$$[A\phi]_i = \frac{\phi_{i-1,t} - 2\phi_{i,t} + \phi_{i+1,t}}{\delta x^2} \approx \frac{\partial^2 \phi}{\partial x^2}(i\delta x, t). \tag{3.17}$$

The approximation in Equation (3.16) is then used in Equation (3.1) to approximate the spatial derivative operator

$$\frac{\partial^2}{\partial t^2}\phi = -\frac{1}{\delta x^2} L(G_{\delta x})\phi. \tag{3.18}$$

Based on this formula, [56] shows that simulating

$$H = \frac{1}{\delta x} \begin{pmatrix} 0 & B \\ B^\dagger & 0 \end{pmatrix} \tag{3.19}$$

with

$$BB^\dagger = L(G_{\delta x}) \tag{3.20}$$

constructs a quantum circuit that will evolve a part of the quantum state it is applied on according to the discretised wave equation in Equation (3.18).

A matrix B satisfying Equation (3.20) can be obtained directly from the graph $G_{\delta x}$ representing the discretisation. The algorithm to construct the matrix B can be decomposed in three steps. First, the vertices (discretisation points) should be arbitrarily ordered by assigning them a unique index in $[0, N_d - 1]$. Then, each edge of the graph is arbitrarily oriented and indexed with indices in $[0, N_d - 2]$. Finally, B is computed with the following definition

$$B_{ij} = \begin{cases} 1 & \text{if edge } j \text{ is a self-loop of vertex } i, \\ 1 & \text{if edge } j \text{ has vertex } i \text{ as source}, \\ -1 & \text{if edge } j \text{ has vertex } i \text{ as sink}, \\ 0 & \text{otherwise} \end{cases} . \tag{3.21}$$

Note that edges' orientation and vertices/edges ordering is completely arbitrary. Changing either the edges orientation on one of the orderings will change the matrix B but will not affect $BB^\dagger$ which should be equal to $L(G_{\delta x})$. This freedom in the ordering and orientation choices takes a crucial importance in the oracle implementation as it allows us to pick the ordering/orientation that will produce an easy-to-implement matrix B.

Dirichlet boundary conditions

Fixing boundary conditions is a requirement for most of the partial differential equations to admit a unique well-defined solution. There exist several boundary conditions such as Neumann, Dirichlet, Robin or Cauchy ones. For simplicity, we restricted ourselves to the study of Equation (3.1) with Dirichlet boundary condition of Equation (3.2).

In the case of Dirichlet boundary conditions on the 1-dimensional line $[0, 1]$, the two boundary nodes at $x = 0$ and $x = 1$ can be ignored as their value is always equal to 0. Moreover, [56] shows that the graph $G^D_{\delta x}$ representing the discretisation with Dirichlet boundary conditions of Equation (3.2) is simply $G_{\delta x}$ with self-loops on the two outer nodes (i.e. the ones indexed 1 and $N_d - 2$ as 0 and $N_d - 1$ are ignored). $G^D_{\delta x}$ is depicted in Figure 3.2. The algorithm to construct the matrix B remain the same as explained in Section 3.2.4.

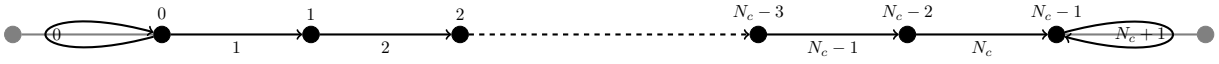


Figure 3.2: *Graph $G^D_{\delta x}$ representing the discretisation of the 1-dimensional line $[0,1]$ with Dirichlet boundary conditions. The points and edges in grey are only drawn for illustration purpose and are ignored in the analysis because the boundary condition impose a value of 0 on these vertices. Loops are added to $G_{\delta x}$ to encode the fact that this graph represents Dirichlet boundary conditions. Vertices (resp. edges) are ordered with indices within $[0, N_c-1]$ (resp. $[0, N_c+1]$). N_c is the number of considered points and is equal to N_d-2 (the two extreme points are ignored).*

Matrices construction

All the pieces are now in place to start building the matrix $B_\mathrm{d} \in \{-1,0,1\}^{(N_c-1)\times N_c}$. Using the definition of the matrix B written in Equation (3.21) and the graph $G^D_{\delta x}$ depicted in Figure 3.2 we end up with

$$B_\mathrm{d} = \begin{pmatrix} 1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -1 & 1 \end{pmatrix}. \tag{3.22}$$

We can easily check that $\frac{1}{\delta x^2} B_\mathrm{d} B_\mathrm{d}^\dagger$ is equal to the well-known discretisation matrix

$$\frac{1}{\delta x^2} B_\mathrm{d} B_\mathrm{d}^\dagger = \frac{1}{\delta x^2} \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}, \tag{3.23}$$

which validate the method of construction of B_d.

Computing $\widetilde{H_\mathrm{d}}$, the Hamiltonian matrix that should be simulated to evolve the quantum state according to the wave equation in Equation (3.1) with Dirichlet boundary conditions, is now straightforward. Using Equation (3.19), we directly obtain

$$\widetilde{H_d} = \frac{1}{\delta x} \begin{pmatrix} 0 & \cdots & \cdots & 0 & 1 & 1 & 0 & \cdots & 0 \\ \vdots & & & \vdots & 0 & -1 & 1 & \ddots & \vdots \\ \vdots & & & \vdots & \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 0 & \cdots & 0 & -1 & 1 \\ 1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 1 & -1 & \ddots & \vdots & \vdots & & & & \vdots \\ 0 & 1 & \ddots & 0 & \vdots & & & & \vdots \\ \vdots & \ddots & \ddots & -1 & \vdots & & & & \vdots \\ 0 & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix} \tag{3.24}$$

As explained in Section 3.2.2, the Hamiltonian simulation algorithm implemented requires that the Hamiltonian to simulate is split as a sum of 1-sparse hermitian matrices. There are a lot of valid decompositions for the matrix $\widetilde{H_d}$ and we are free to choose the decomposition that will simplify the most the oracle implementation or reduce the gate complexity.

We made the choice to decompose B_d as two 1-sparse matrices and then reflect this decomposition on $\widetilde{H_d}$. Let B_1 and B_{-1} defined as

$$B_1 = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & 1 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix} \tag{3.25}$$

$$B_{-1} = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & -1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 0 \end{pmatrix} \tag{3.26}$$

we have $B_\mathrm{d} = B_1 + B_{-1}$. Let also

$$\widetilde{H_1} = \frac{1}{\delta x}\begin{pmatrix} 0 & B_1 \\ {B_1}^\dagger & 0 \end{pmatrix}, \quad \widetilde{H_{-1}} = \frac{1}{\delta x}\begin{pmatrix} 0 & B_{-1} \\ {B_{-1}}^\dagger & 0 \end{pmatrix}, \tag{3.27}$$

it is easy to see that $\widetilde{H_d} = \widetilde{H_1} + \widetilde{H_{-1}}$ and that both $\widetilde{H_1}$ and $\widetilde{H_{-1}}$ are 1-sparse hermitian matrices.

For convenience, we also define

$$H_1 = \begin{pmatrix} 0 & B_1 \\ {B_1}^\dagger & 0 \end{pmatrix}, \quad H_{-1} = \begin{pmatrix} 0 & B_{-1} \\ {B_{-1}}^\dagger & 0 \end{pmatrix}, \tag{3.28}$$

and $H_d = H_1 + H_{-1}$, the $\widetilde{H_1}$, $\widetilde{H_{-1}}$ and $\widetilde{H_d}$ matrices re-scaled to contain only integer weights. These matrices have the interesting property that simulating $\widetilde{H_d}$ (resp. $\widetilde{H_1}$, $\widetilde{H_{-1}}$) for a time t is equivalent to simulating H_d (resp. H_1, H_{-1}) for a time $\frac{t}{\delta x}$. This property will be used in the following sections as it offers us the opportunity to simulate the "easy-to-simulate" (see Definition 8), integer-weighted matrices H_d, H_1 and H_{-1} instead of the real-weighted ones $\widetilde{H_d}$, $\widetilde{H_1}$ and $\widetilde{H_{-1}}$ that are not within the "easy-to-simulate" category as defined in Definition 8.

Note also that a lower bound of the number of qubits needed to solve the wave equation for N_d discretisation points can be computed from the dimensions of H_d. As the non-empty upper-left block of matrix H_d is of dimension $(2N_d - 1) \times (2N_d - 1)$, we need at least

$$\lceil \log_2(2N_d - 1) \rceil \tag{3.29}$$

qubits to simulate it. This estimation does not take into account ancilla qubits that may be needed to implement the oracles.

3.2.5 Oracle construction

Oracles can be seen as the interface between a quantum procedure and real-world data. Their purpose is to encode classical data such that a quantum algorithm can process it efficiently.

Oracle interface

In order to work as a bridge between the classical and the quantum worlds and to be used by the quantum algorithm, a clear interface for the oracle should be established.

We chose to use the interface described in [79, Eq. 4.4] with slight modifications improving the arity of the oracle for our specific case of 1-sparse matrices.

More precisely, our oracles O implement the following interface

$$O\left|x_0\right\rangle_x \left|0\right\rangle_m \left|0\right\rangle_v \left|0\right\rangle_s = \left|x_0\right\rangle_x \left|m(x_0)\right\rangle_m \left|v(x_0)\right\rangle_v \left|s(x_0)\right\rangle_s \tag{3.30}$$

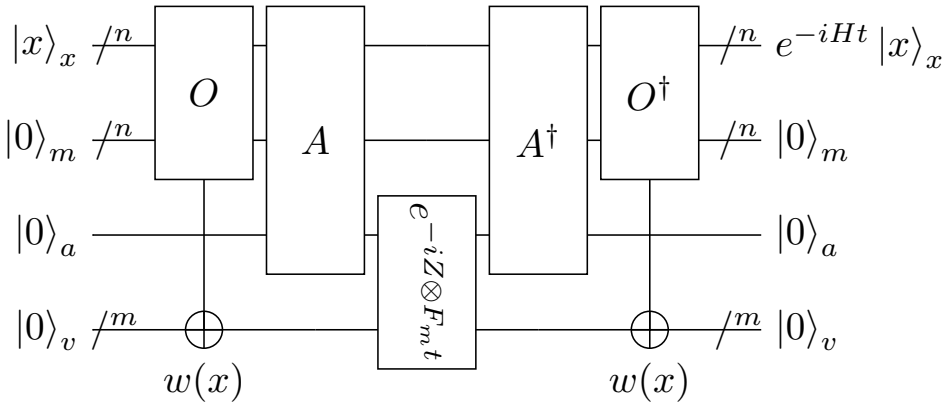


Figure 3.3: *Quantum circuit re-created from [79, p. 71] that simulates a 1-sparse integer-weighted Hamiltonian for a given time t. O is the implementation of the oracle, A is a quantum circuit defined in [79, p. 70]. F_m is defined as the diagonal matrix with diagonal entries increasing from 0 to $2^m - 1$ (see Equation (3.36)).*

with $|x_0\rangle_x$ encoding a row index as a unsigned integer, $m(x)$ the function that returns the column index of the only non-zero element in row x, $v(x) = |w(x)|$ the absolute value of the weight $w(x)$ of the first (and only) non-zero element in row x and

$$s(x) = \begin{cases} 0 \text{ if } w(x) > 0 \\ 1 \text{ if } w(x) < 0 \end{cases} \tag{3.31}$$

the sign of the first non-zero entry in row x. The sign $s(x)$ is purposely not defined for rows x that do not have any non-zero entry (i.e., $w(x) = 0$). The specific case of empty rows is discussed in Claim 1.

Note 4. The quantum register are labelled with their respective usage: x for the index of of the row considered, m for the index of the column considered, v for the value of the element at (row, index) and s for the sign of the element at (row, index). A fifth label "a" is used along the paper to label a register used as an ancilla.

The interface of the oracle O can also be obtained with 3 separate oracles that will each take care of computing one output:

$$M\,|x_0\rangle_x\,|0\rangle_m = |x_0\rangle_x\,|m(x)\rangle_m \tag{3.32}$$

$$V\,|x_0\rangle_x\,|0\rangle_v = |x_0\rangle_x\,|v(x)\rangle_v \tag{3.33}$$

$$S\,|x_0\rangle_x\,|0\rangle_s = |x_0\rangle_x\,|s(x)\rangle_s \tag{3.34}$$

Optimisation of M and S

Claim 1. The simulation algorithms provided by [79] have the interesting property that if the oracle V encodes a weight of zero for some inputs (i.e. $v(x) = 0$ for some x) then the outputs of oracles M and S are ignored for these inputs.

Proof. The circuit simulating a 1-sparse m-bit-integer weighted Hamiltonian H depicted in Figure 3.3 is taken from [79]. In our special case of 1-bit weights (i.e. $m = 1$), the third quantum

gate $e^{-iZ\otimes F_m t}$ can be written as

$$\begin{aligned} e^{-iZ\otimes F_m t} &= e^{-iZ\otimes F_1 t} \\ &= \exp\left[-i\begin{pmatrix} F_1 & 0 \\ 0 & -F_1 \end{pmatrix} t\right] \\ &= \begin{pmatrix} e^{-iF_1 t} & 0 \\ 0 & e^{iF_1 t} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-it} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{-it} \end{pmatrix}. \end{aligned} \tag{3.35}$$

where

$$F_m = \begin{pmatrix} 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & \ddots & & & & \vdots \\ \vdots & \ddots & 2 & \ddots & & & \vdots \\ \vdots & & \ddots & 3 & \ddots & & \vdots \\ \vdots & & & \ddots & 4 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 2^m - 1 \end{pmatrix}. \tag{3.36}$$

It follows from the matrix notation that if the second qubit $e^{-iZ\otimes F_1 t}$ is applied on is in the state $|0\rangle$, the gate $e^{-iZ\otimes F_1 t}$ is the identity transformation, i.e. the unitary operation $e^{-iZ\otimes F_1 t}$ sends $|00\rangle$ (resp. $|10\rangle$) to $|00\rangle$ (resp. $|10\rangle$). This means that if the oracle O does not set the last qubit to $|1\rangle$ (i.e. the oracle encodes a weight of 0 for the x^{th} row of H), the quantum circuit depicted in Figure 3.3 can be simplified up to an identity transformation as the effects of O (resp. A) are reverted by $O^\dagger$ (resp. $A^\dagger$).

Rephrasing, if the x^{th} row of matrix H has no non-zero entries, the effects of the oracle O is ignored, which implies that the effects of the oracles M and S that compose O are also ignored. □

Using the result of Claim 1, we are free to implement any transformation that best suits us for the set of inputs $|x\rangle$ such that the x^{th} row of the considered hermitian matrix (H_1 or H_{-1}) has no non-zero elements as long as the oracle V implements the right transformation.

To illustrate clearly the *implemented* transformations we chose to encode with M and S, the next sections will re-write the matrices H_1 and H_{-1} according to Equation (3.28) but with one $\mathbf{0}$ or $-\mathbf{0}$ in each empty row. A $\mathbf{0}$ entry at position (i, j) in the matrix means that the row i was empty, the oracle M will map $|i\rangle_x$ to $|j\rangle_m$ and the oracle S will encode a positive sign, i.e. $|0\rangle_s$. The same reasoning applies for $-\mathbf{0}$ entries, except that the encoded sign is now negative, i.e. $|1\rangle_s$.

The following sections will explain step by step the construction of each of the three oracles M, V and S, both for the matrix H_1 (M_1, V_1 and S_1) and the matrix H_{-1} (M_{-1}, V_{-1} and S_{-1}).

About arithmetic and logic quantum gates

Implementing the oracles M, V and S for the matrices H_1 and H_{-1} requires several arithmetic and logic quantum gates such as `or`, `add` or `compare`. All these gates have been implemented prior to the oracle implementation and the implementation steps are detailed in this section.

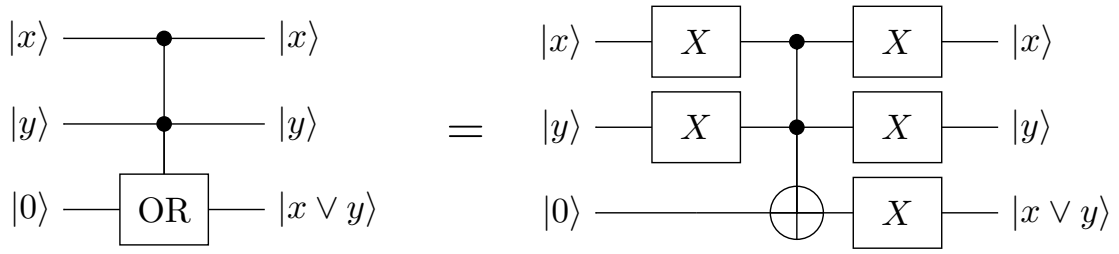


Figure 3.4: *Implementation of the* `or` *gate.*

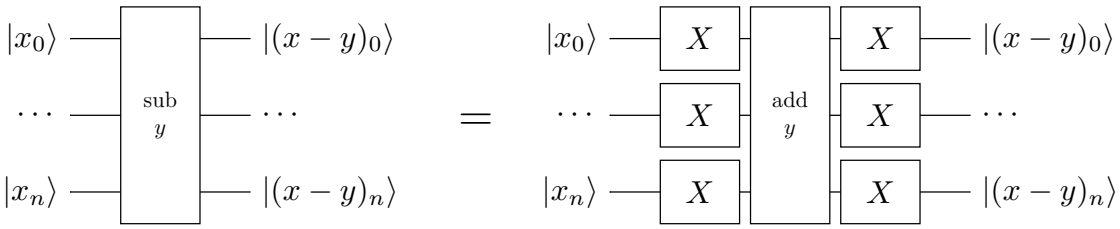


Figure 3.5: *Implementation of the* `sub` *gate from an* `add` *gate. The y value encoding is intentionally omitted. The substractor will use the same encoding as the adder (i.e. either the y value is encoded on a quantum register or it is encoded directly in the quantum circuit implementing the adder). Note that the y value is not negated.*

The `or` gate The `or` gate is easily implemented using only `X` and `CCX` (or Toffoli) gates. The implementation used is depicted in Figure 3.4 and uses the famous Boole algebra formula linking `not`, `or` and `and`: $x \vee y = \neg(\neg x \wedge \neg y)$.

The `add` and `sub` gates Most of the research papers presenting an implementation of the `add` or `sub` gates only consider the case where the two numbers to add or subtract are stored in quantum registers.

In our case, the oracles implementation requires an adder and substractor that can add or subtract to a quantum register a quantity known when the quantum circuit is generated, i.e. not necessarily encoded on a quantum state.

Claim 2. Implementing a substractor is trivial once an adder procedure is available.

Proof. A substractor can be implemented from a generic adder by using the identity

$$a - b = (a' + b)' \tag{3.37}$$

where $'$ denotes the bit-wise complementation.

The circuit resulting of the application of this identity is depicted in Figure 3.5 and only requires one call to the adder and $2n$ additional gates, n being the number of qubits used to represent one of the operands.

□

Note 5. Following Claim 2 we will restrict the study to implementing an adder. Implementing a substractor is trivial and cheap in term of additional quantum gates used once an adder is available.

Definition 10. *Generation-time value* A generation-time value is a value that is known by the programmer when generating the quantum circuit. Knowing a value at generation-time may allow to optimise even further the generated quantum circuit. The closest analogue in classical programming would be C-like macros or recent C++ constexpr expressions.

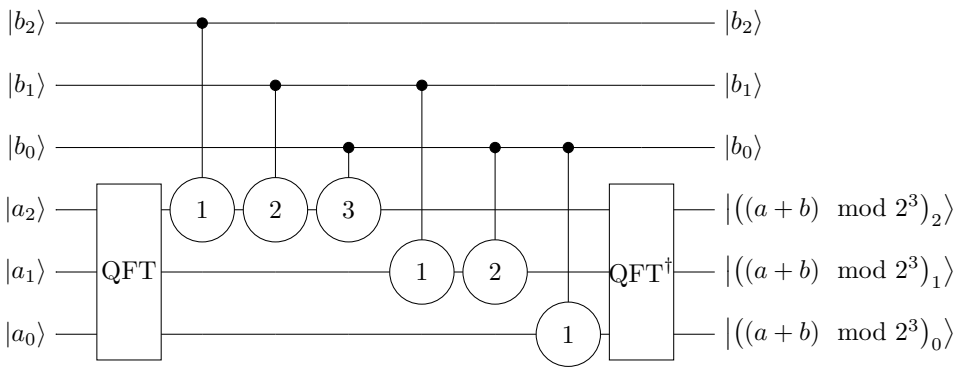


Figure 3.6: *Original Draper's adder example for* 3*-qubit registers* $|a\rangle$ *and* $|b\rangle$*. The round gates between the two applications of the Quantum Fourier Transform (`QFT` gates) are controlled phase gates and are defined in [97]. Note that the adder wraps on overflow, meaning that if an overflow happens, the result will be* $(a+b) \mod 2^3$*.*

The easiest solution to overcome the problem caused by the non-compatible input formats between our problem (with a generation-time value) and the existing adders (with two values encoded on quantum registers) is to encode the quantity known at generation-time into ancillary qubits and then use the regular adder algorithms to add to a quantum register the value encoded in a second quantum register. Even if this solution is trivial to implement, it has the huge downside of requiring $\mathcal{O}(\log_2 b)$ additional ancillary qubits to temporarily store the generation-time value b.

Another answer to the problem would be to adapt a quantum adder originally devised to add two quantum registers to a quantum adder capable of adding a constant value to a quantum register. Several adders [95–97] have been studied to check if they can be modified to allow a generation-time input, i.e. if it possible to remove completely the quantum register storing the right-hand-side (or left-hand-side) of the addition.

The task of removing the quantum register storing one of the operands appears to be challenging for adders based on classical arithmetic like [95, 96] but trivial for Draper's quantum adder introduced in [97].

Claim 3. Draper's quantum adder can be adapted into an efficient adder that takes as right-hand side input a unsigned "generation-time" integer value and add this value to a sufficiently large quantum register encoding another unsigned integer.

Proof. The original Drapper's adder as introduced in [97] is illustrated in Figure 3.6. The only quantum gates using the quantum register $|b\rangle$ are the controlled-phase gates. Moreover, they only use the qubits of the right-hand-side register $|b\rangle$ as controls. In the case of a constant value of b known at generation time, we can replace each controlled-phase gate by either a phase gate if the corresponding bit of b is 1 or by an identity gate (or a "no-op" gate) if the bit of b is 0. Once this transformation has been performed, the quantum register $|b\rangle$ is no longer used and can be safely removed from the circuit. □

The final quantum `add` gate implementation is depicted in Figure 3.7, requires $\mathcal{O}(n^2)$ gates and has a depth of $\mathcal{O}(n)$. Following [97–99], the asymptotic gate count can be improved to $\mathcal{O}(n \log(n))$ by removing the rotation with an angle below a given threshold that depend on hardware noise.

The `cmp` gate For the same reasons exposed in the adder implementation in Section 3.2.5, the `cmp` gate cannot be implemented using the arithmetic comparator presented in [96] because removing the right-hand side qubits seems to be a challenging task.

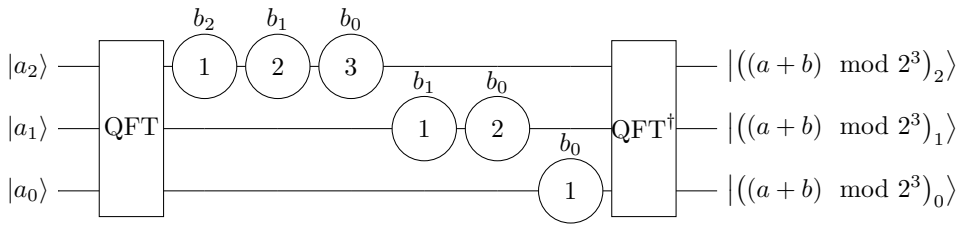


Figure 3.7: *Modified Draper's adder example for 3-qubit register $|a\rangle$ and 3-bit classical constant b. The round gates between the two applications of the Quantum Fourier Transform (`QFT` gates) are phase gates and are defined in [97]. A label b_i above a phase gate means that the phase gate should only be applied when the i^{th} bit of b is set to 1. Note that the adder wraps on overflow, meaning that if an overflow happens, the result will be $(a+b) \mod 2^3$.*

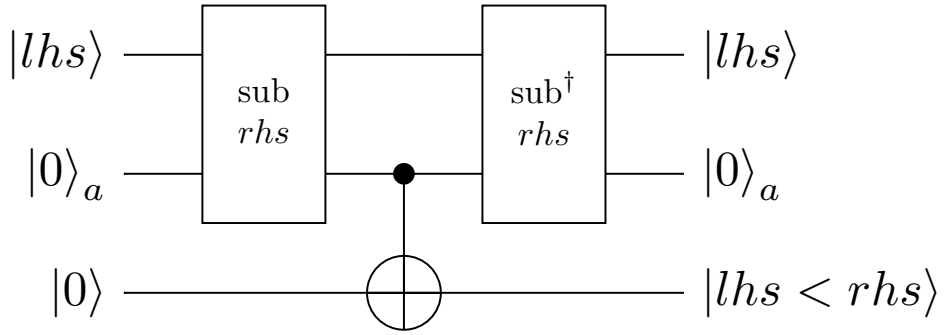


Figure 3.8: *Computation of the high-bit of $lhs - rhs$ with a $(n+1)$-qubit substractor. The second quantum register is an ancilla qubit that is appended to the quantum register storing $|\text{lhs}\rangle$ in order to form a $(n+1)$-qubit register. The result is stored in a third quantum register as $|1\rangle$ if $lhs < rhs$, else $|0\rangle$.*

Instead, we use the idea from [96, Section 4.3] that explain how to implement a comparator only by using a quantum adder. The comparison algorithm works by computing the high-bit of the expression $a - b$. If this high-bit is in the state $|1\rangle$ then $a < b$.

In order to compute the high-bit of $a - b$, several options are open. The two most promising options are described in the following paragraphs.

The first option is to use a substractor acting on $n+1$ qubits and behaving nicely on underflow (i.e. underflow result in cycling to the highest-value), as illustrated in Figure 3.8. This approach requires 2 calls to the substractor and 1 additional 2-qubit quantum gate.

Another solution would be to use Equation (3.37) to change the subtraction into an addition and then use a specialised procedure to compute the high-bit of the addition of two numbers a and b (a being encoded on a quantum register and b a constant). Computing the high-bit of an addition between a quantum register and a constant can be performed with the `CARRY` gate introduced in [100]. This approach requires $\mathcal{O}(n)$ Toffoli, `CNOT` and `X` gates.

Each of the described methods has its advantages and drawbacks.

For example, the first method crucially relies on a quantum substractor, and will have the same properties as the substractor used. In our specific case, we use the substractor implemented with Drapper's adder [97] as explained in Section 3.2.5, which in turn uses the quantum Fourier transform. The main disadvantage of using the QFT when looking at practical implementation on quantum hardware is that the QFT involves phase gates with exponentially small angles. These gates may be implemented correctly up to a given threshold, but very small rotation angles will inevitably not be as precise as *normal* rotation angles due to the hardware limitations in precision. This problem can be circumvented by using an approximate QFT algorithm [98, 99] that will cut all the rotation gates that have a rotation angle smaller than a given threshold from the generated circuit but the algorithm will not be exact anymore (small probability of incorrect result).

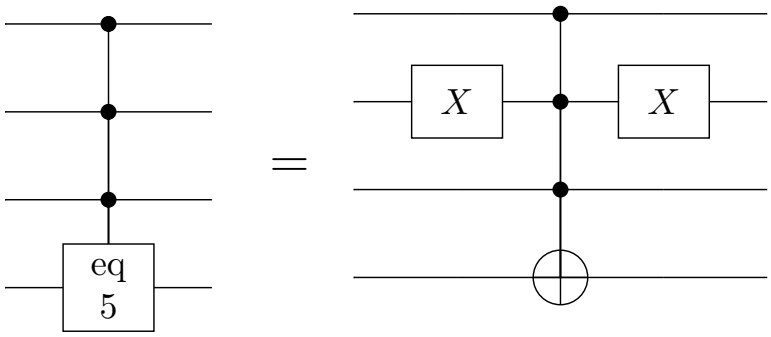


Figure 3.9: *Example of* `eq` *gate implementation for the compile-time value* 5. `X` *gates are applied to the second control qubit because the only bit set to* 0 *in the big-endian binary representation of* $5 = 101_2$ *is at the second (middle) position.*

On the other hand, the `CARRY` gate involves only `X`, controlled-`X` and Toffoli gates. This restriction makes this implementation more robust than the first one to hardware approximations. Another difference is the connectivity needed by the approaches: the first method relies on a adder implemented with the quantum Fourier transform, which use an all-to-all connectivity whereas the `CARRY` gate, once the qubits correctly ordered, only contains gates on adjacent qubits. As a side note, the exclusive use of logical gates `X`, controlled-`X` and Toffoli may allow us to simulate efficiently the `CARRY` gate on classical hardware as it only involves classical arithmetic.

As a last word, in the future, the QFT may be implemented directly into the hardware chips to make it more efficient because it is one of the most used quantum procedure (and so one of the best candidate for optimisation). Taking this possibility into account seems a little premature right now but may have a high impact on the efficiency and precision of the first solution presented.

After summarising all the drawbacks and advantages, we decided to use the arithmetic comparator for its linear number of gates, because it is based on arithmetic which does not involve exponentially small rotation angles and because the need to have $n-1$ dirty qubits to lend to the procedure is not an issue in our implementation.

The `eq` gate The last gate the oracle implementation will need is an `eq` gate, testing the equality between an integer stored in a quantum register and a generation-time constant integer.

This gate has been implemented with a multi-controlled Toffoli gate and a few `X` gates before and after the control qubits of the Toffoli gates that should be equal to $|0\rangle$. The `X` gates are necessary because a raw Toffoli gate set its target qubit only when all its control are in the state $|1\rangle$, but we want each control qubit to be equal to a specific bit of the generation-time constant integer, which can be either $|0\rangle$ or $|1\rangle$.

An implementation example is available in Figure 3.9.

Implementing a `NOT` gate controlled by n qubits can be done with only one ancilla qubit or $n-2$ garbage qubits and requires $\mathcal{O}(n)$ `X`, controlled-`X` or Toffoli gates [101].

Oracles for H_1

As noted in Section 3.2.5, the oracles M_1 and S_1 can be optimised by using the fact that they can encode anything for $|x\rangle_x$ when the x^{th} row of H_1 is empty.

We decided to use this optimisation opportunity to add regularity to the description of the H_1 matrix. The *implemented* matrix H_1, denoted as H_1^{impl}, is described in Equation (3.38).

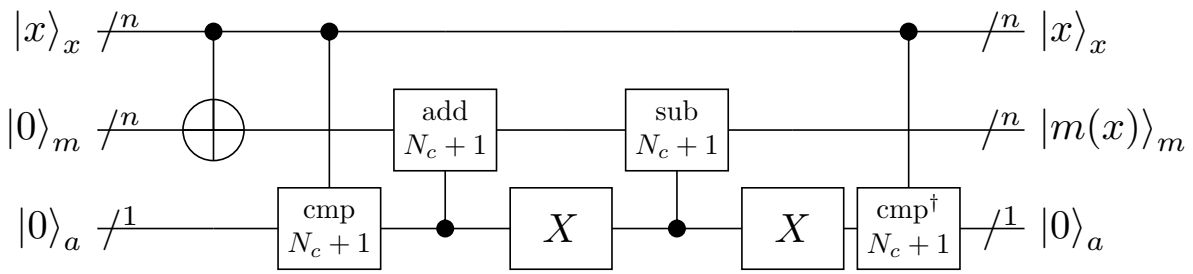


Figure 3.10: *Implementation of the oracle M_1. The* `cmp` *gate compare the value of the control quantum register (interpreted as a unsigned integer) with the parameter given (written below the* `cmp`*). If the control register is strictly lower than the parameter, the gate set the qubit it is applied on to $|1\rangle$. The* `add` *(resp.* `sub`*) gate used in this quantum circuit add (resp. subtract) the value of its parameter to (resp. from) the quantum register it is applied on only if the control qubit is in the state $|1\rangle$.*

Note 6. All indices start at 0. The first row of a matrix has the index 0, the second row the index 1 and so on. This convention is used to match Python's indexing that starts at 0.

$$H_1^{\text{impl}} = \begin{array}{c} \\ N_c \\ N_c+1 \\ 2^q-(2N_c+1) \end{array} \overbrace{}^{N_c} \overbrace{}^{N_c+1} \overbrace{}^{2^q-(2N_c+1)} \begin{pmatrix} 0 & \cdots & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & & & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & & & & & \vdots \\ \vdots & & & \vdots & \vdots & & \ddots & \ddots & 0 & \vdots & & & & & \vdots \\ 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & 0 & 1 & 0 & & & & & \vdots \\ 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & \cdots & 0 & \mathbf{0} & & & & & \vdots \\ 1 & \ddots & & \vdots & \vdots & & & & \vdots & 0 & & & & & \vdots \\ 0 & \ddots & \ddots & \vdots & \vdots & & & & \vdots & \vdots & & & & & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots & & & & \vdots & \vdots & & & & & \vdots \\ 0 & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & 0 & \mathbf{0} & 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & & & & \ddots & \mathbf{0} & \ddots & & \vdots & \vdots & & & & & \vdots \\ \vdots & & & & & \ddots & \ddots & \ddots & \vdots & \vdots & & & & & \vdots \\ \vdots & & & & & & \ddots & \mathbf{0} & 0 & 0 & & & & & \vdots \\ \vdots & & & & & & & \ddots & \mathbf{0} & 0 & \ddots & & & & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & \mathbf{0} & 0 & 0 & \cdots & \cdots & 0 \end{pmatrix} \tag{3.38}$$

According to the shape of the matrix in Equation (3.38), the oracle M_1 should implement the transformation

$$M_1|x\rangle_x|0\rangle_m \mapsto \begin{cases} |x\rangle_x \otimes |x + (N_c+1)\rangle_m & \text{if } x < (N_c+1) \\ |x\rangle_x \otimes |x - (N_c+1)\rangle_m & \text{else} \end{cases}. \tag{3.39}$$

M_1 can be easily implemented with the quantum circuit depicted in Figure 3.10.

The oracle V cannot be simplified using the results from Claim 1. It should implement the transformation written in Equation (3.40).

$$V_1|x\rangle_x|0\rangle_v \mapsto \begin{cases} |x\rangle_x|1\rangle_v & \text{if } (x < 2N_c+1) \wedge (x \neq N_c) \\ |x\rangle_x|0\rangle_v & \text{else} \end{cases}. \tag{3.40}$$

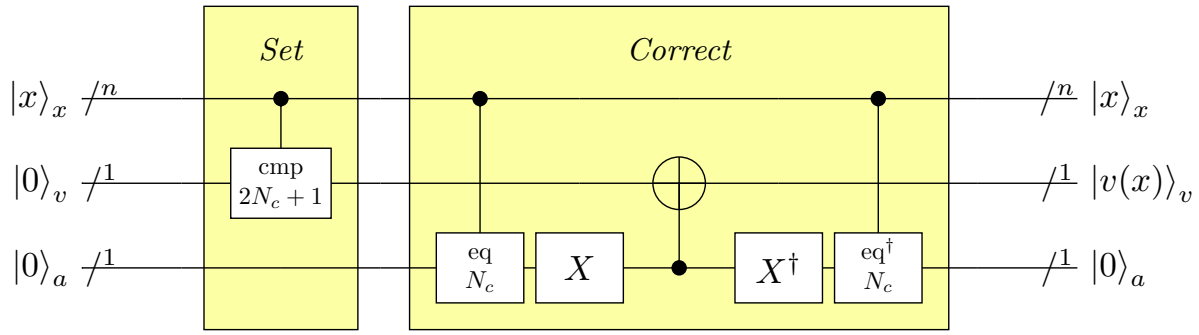


Figure 3.11: *Implementation of the oracle V_1. The* `cmp` *gate compare the value of the control quantum register (interpreted as a unsigned integer) with the parameter given (written below the* `cmp`*). If the control register is strictly lower than the parameter, the gate set the qubit it is applied on to $|1\rangle$. The* `eq` *gate used in this quantum circuit sets its target qubit to $|1\rangle$ if the value of its parameter is equal to the value encoded on the quantum register controlling the gate.*

The implementation of the oracle V_1 is depicted in Figure 3.11. The first part, *Set*, sets the weight qubit to 1 for all $|x\rangle_x$ such that $x < 2N_c + 1$. As this does not correspond to the correct expression of V, the second part *Correct* is here to set the weight register back to $|0\rangle_v$ when $x == N_c$.

The last oracle left to implement in order to be able to simulate H_1 is S_1, the oracle encoding the signs of the non-zero entries of H_1. The convention used to encode the sign of an entry has been taken from [79] and is: a positive sign is encoded as $|0\rangle_s$, a negative sign is encoded as $|1\rangle_s$. As shown in Equation (3.38), H_1 only contains positive non-zero entries so the sign oracle S_1 should implement the simple transformation of Equation (3.41): the identity.

$$S_1|x\rangle_x|0\rangle_s \mapsto |x\rangle_x|0\rangle_s \tag{3.41}$$

Oracles for H_{-1}

The matrix H_{-1} has less regularity than H_1, which will lead to a more complex implementation. The *implemented* matrix H_{-1}, denoted as H_{-1}^{impl}, is described in Equation (3.42).

$$H_{-1}^{\text{impl}} = \begin{array}{c} \\ N_c \\ N_c+1 \\ 2^q-(2N_c+1) \end{array} \overbrace{\phantom{\begin{pmatrix}0&\cdots&\cdots&0&1&0&\cdots&\cdots&0&0&\cdots&\cdots&\cdots&\cdots&0\end{pmatrix}}}^{N_c} \overbrace{\phantom{\begin{pmatrix}0&\cdots&\cdots&0&1&0&\cdots&\cdots&0&0&\cdots&\cdots&\cdots&\cdots&0\end{pmatrix}}}^{N_c+1} \overbrace{\phantom{\begin{pmatrix}0&\cdots&\cdots&0&1&0&\cdots&\cdots&0&0&\cdots&\cdots&\cdots&\cdots&0\end{pmatrix}}}^{2^q-(2N_c+1)}$$

$$\begin{pmatrix}
0 & \cdots & \cdots & 0 & 1 & 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
\vdots & & & \vdots & 0 & -1 & \ddots & & \vdots & \vdots & & & & & \vdots \\
\vdots & & & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & & & & & \vdots \\
0 & \cdots & \cdots & 0 & 0 & \cdots & 0 & -1 & 0 & \vdots & & & & & \vdots \\
1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & \cdots & 0 & \vdots & & & & & \vdots \\
0 & -1 & \ddots & \vdots & \vdots & & & & \vdots & \vdots & & & & & \vdots \\
\vdots & \ddots & \ddots & 0 & \vdots & & & & \vdots & \vdots & & & & & \vdots \\
\vdots & & \ddots & -1 & 0 & & & & \vdots & \vdots & & & & & \vdots \\
0 & \cdots & \cdots & 0 & \mathbf{-0} & \ddots & & & \vdots & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & 0 & \mathbf{-0} & \ddots & & \vdots & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
\vdots & & & & & \ddots & \ddots & \ddots & \vdots & \vdots & & & & & \vdots \\
\vdots & & & & & & \ddots & \ddots & 0 & \vdots & & & & & \vdots \\
\vdots & & & & & & & \ddots & \mathbf{-0} & 0 & & & & & \vdots \\
\vdots & & & & & & & & \ddots & \mathbf{-0} & \ddots & & & & \vdots \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & \mathbf{-0} & 0 & \cdots & \cdots & 0
\end{pmatrix} \tag{3.42}$$

Following the placement of the non-zero and the $\mathbf{0}$ or $-\mathbf{0}$ entries in the matrix H_{-1}^{impl} of Equation (3.42), the oracle M_{-1} should implement the transformation

$$M_{-1}|x\rangle_x|0\rangle_m \mapsto \begin{cases} |x\rangle_x|x+N_c\rangle_m & \text{if } x < N_c \\ |x\rangle_x|x-N_c\rangle_m & \text{else} \end{cases}. \tag{3.43}$$

This transformation is quite similar to the one implemented by the oracle M_1 in Equation (3.39): N_c+1 from the transformation M_1 has been replaced by N_c in the transformation M_{-1}. Thanks to this similarity, the implementation of M_{-1} will be a nearly-exact copy of the implementation of M_1. The full implementation of the M_{-1} oracle is depicted in Figure 3.12.

The weight oracle V_{-1} is the simplest to implement for the matrix H_{-1}, even if it cannot take advantage of the optimisation discussed in Claim 1. The transformation that should be implemented by the oracle V_{-1} is shown in Equation (3.44).

$$V_{-1}|x\rangle_x|0\rangle_v \mapsto \begin{cases} |x\rangle_x|1\rangle_v & \text{if } x < 2N_c \\ |x\rangle_x|0\rangle_v & \text{else} \end{cases}. \tag{3.44}$$

The implementation of the weight oracle V_{-1} is illustrated in Figure 3.13.

The last oracle left to implement is S_{-1}, the sign oracle. Due to the sign irregularity in the matrix H_{-1}^{impl}, the implementation of S_{-1} is more involved and requires several ancillary qubits. According to the shape of the matrix H_{-1}^{impl}, the sign oracle S_{-1} should implement the

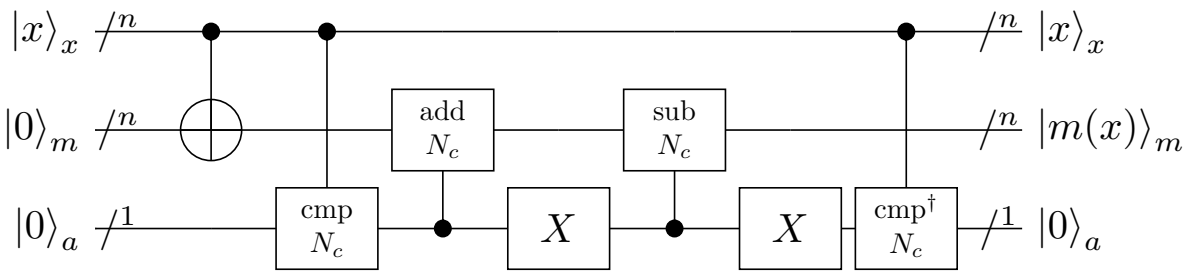


Figure 3.12: *Implementation of the oracle M_{-1}. The* `cmp` *gate compare the value of the control quantum register (interpreted as a unsigned integer) with the parameter given (written below the* `cmp`*). If the control register is strictly lower than the parameter, the gate set the qubit it is applied on to $|1\rangle$. The* `add` *(resp.* `sub`*) gate used in this quantum circuit add (resp. subtract) the value of its parameter to (resp. from) the quantum register it is applied on only if the control qubit is in the state $|1\rangle$.*

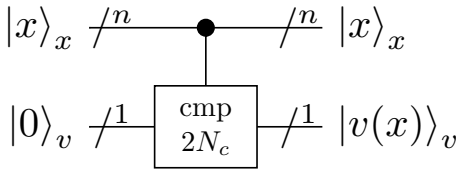


Figure 3.13: *Implementation of the oracle V_{-1}. The* `cmp` *gate compare the value of the control quantum register (interpreted as a unsigned integer) with the parameter given (written below the* `cmp`*). If the control register is strictly lower than the parameter, the gate set the qubit it is applied on to $|1\rangle$.*

transformation defined in Equation (3.45).

$$S_{-1}|x\rangle_x|0\rangle_s \mapsto \begin{cases} |x\rangle_x|0\rangle_s & \text{if } (x=0) \vee (x=N_c) \\ |x\rangle_x|1\rangle_s & \text{else} \end{cases}. \tag{3.45}$$

An implementation of the oracle S_{-1} is illustrated in Figure 3.14.

3.3 Results

Due to the size of the quantum circuits studied, this study will only use quantum simulators, i.e., classical software simulating the behaviour of a quantum computer, instead of existing quantum hardware. Using a simulator instead of a real quantum computer has several advantages. In terms of development process, a simulator allows the developer to perform several actions that

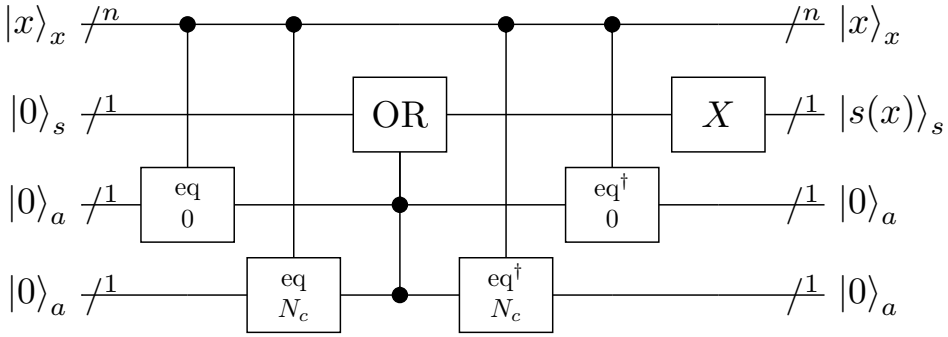


Figure 3.14: *Implementation of the oracle S_{-1}. The* `eq` *gate used in this quantum circuit is presented in Figure 3.9 and test if the value encoded in its control qubits is equal to the compile-time value given. The* `OR` *gate flips the target qubits if and only if at least one of the two control qubits is in the state $|1\rangle$.*

are not possible as-is on a quantum processor such as describing a quantum gate with a unitary matrix instead of a sequence of hardware operations. Another useful operation that is possible on a quantum simulator and not currently achievable on a quantum processor is efficient generic state preparation.

Our implementation uses only standard quantum gates and does not leverage any of the simulator-only features such as quantum gates implemented from a unitary matrix. In other words, both the Hamiltonian simulation procedure and the quantum wave equation solver are "fully quantum" and are readily executable on a quantum processor, provided that it has enough qubits. As a proof, and in order to benchmark our implementation, we translated the generated quantum circuits to IBM Q Melbourne gate-set (see Equation (3.4)). IBM Q Melbourne [102] is a quantum chip with 14 usable qubits made available by IBM on the 23$^{\text{th}}$ of September, 2018.

Note 7. We chose IBM Q Melbourne mainly because, at the time of writing, it was the publicly accessible quantum chip with the largest number of qubits. It is important to note that even if IBM Q Melbourne has 14 qubits, the quantum circuits constructed in this paper can not be executed "as is" because they require more qubits. Consequently, because of this hardware limitation, hardware topology has also been left apart of the study.

This translation to IBM Q Melbourne gate set allowed us to have an estimation of the number of hardware gates needed to either solve the wave equation or simulate a specific Hamiltonian on this specific hardware. Combining these numbers and the hardware gate execution time published in [103], we were able to compute a rough approximation of the time needed to solve the considered problem presented in Equations (3.1) and (3.2) on this specific hardware.

3.3.1 Hamiltonian simulation

As explained in Section 3.2.1, the Hamiltonian simulation algorithm implemented has been first devised in [25, 79]. A quick review of the algorithm along with implementation details can be found in Section 3.2.2. This Hamiltonian simulation procedure requires that the Hamiltonian matrix H to simulate can be decomposed as

$$H = \sum_{j=1}^{m} H_j \tag{3.46}$$

where each H_j is an efficiently simulable hermitian matrix.

In our benchmark, we simulated the Hamiltonian described in Equation (3.24). According to [79], real 1-sparse hermitian matrices with only 1 or 0 entries can be simulated with $\mathcal{O}(n)$ gates and 2 calls to the oracle, n being the number of qubits the Hamiltonian H acts on. The exact gate count can be found in Table 3.1 in the row `1-sparse HS`.

Let O_i be the gate complexity of the oracle implementing the i^{th} hermitian matrix H_i of the decomposition in Equation (3.46), we end up with an asymptotic complexity of $\mathcal{O}(n + O_i)$ to simulate H_i. Once again, the exact gate count is decomposed in Table 3.1.

Applying the Trotter-Suzuki product-formula of order k (see Definition 9 in Section 3.2.2 for the definition of the Trotter-Suzuki product-formula) on the quantum circuit simulating the hermitian matrices produces a circuit of size

$$\mathcal{O}\left(5^k \sum_{i=1}^{m} (n + O_i)\right). \tag{3.47}$$

This circuit should finally be repeated r times in order to achieve an error of at most ϵ, with

$$r \in \mathcal{O}\left(5^k m\tau \left(\frac{m\tau}{\epsilon}\right)^{\frac{1}{2k}}\right), \tag{3.48}$$

and $\tau = t \max_i ||H_i||$, t being the time for which we want to simulate the given Hamiltonian and $||\cdot||$ being the spectral norm [25].

Merging Equations (3.47) and (3.48) gives us the complexity

$$\mathcal{O}\left(5^{2k} m\tau \left(\frac{m\tau}{\epsilon}\right)^{\frac{1}{2k}} \sum_{i=1}^{m} (n + O_i)\right). \tag{3.49}$$

This generic expression of the asymptotic complexity can be specialised to our benchmark case. The number of gates needed to implement the oracles is $\mathcal{O}(n^2)$ and the chosen decomposition contains $m = 2$ hermitian matrices, each with a spectral norm of 1. Replacing the symbols in Equations (3.47) and (3.48) results in the asymptotic gate complexity of

$$\mathcal{O}\left(5^k n^2\right) \tag{3.50}$$

for the circuit simulating $e^{-iHt/r}$ and a number

$$r \in \mathcal{O}\left(5^k t \left(\frac{t}{\epsilon}\right)^{\frac{1}{2k}}\right) \tag{3.51}$$

of repetitions, which lead to a total gate complexity of

$$\mathcal{O}\left(5^{2k} n^2 t \left(\frac{t}{\epsilon}\right)^{\frac{1}{2k}}\right). \tag{3.52}$$

In order to check that our implementation follows this theoretical asymptotic behaviour, we chose to let $k = 1$ and plotted the number of gates generated versus the three parameters that have an impact on the number of gates: the number of discretisation points N_d (Figure 3.15a), the time of simulation t (Figure 3.15b) and the precision ϵ (Figure 3.15c). The corresponding asymptotic complexity should be

$$\mathcal{O}\left(n^2 \frac{t^{3/2}}{\sqrt{\epsilon}}\right) = \mathcal{O}\left(\log_2(N_d)^2 \frac{t^{3/2}}{\sqrt{\epsilon}}\right). \tag{3.53}$$

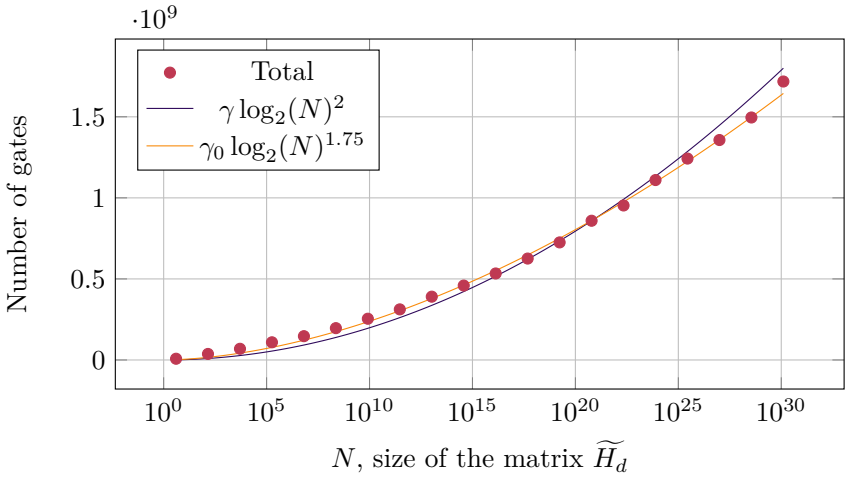
A small discrepancy can be observed in Figure 3.15a: the theoretical asymptotic number of gates is $\mathcal{O}\left(\log_2(N)^2\right)$ but the experimental values seems better fitted with an asymptotic behaviour of $\mathcal{O}\left(\log_2(N)^{7/4}\right)$. This may be caused by the asymptotic regime not being reached yet.

3.3.2 Wave equation solver

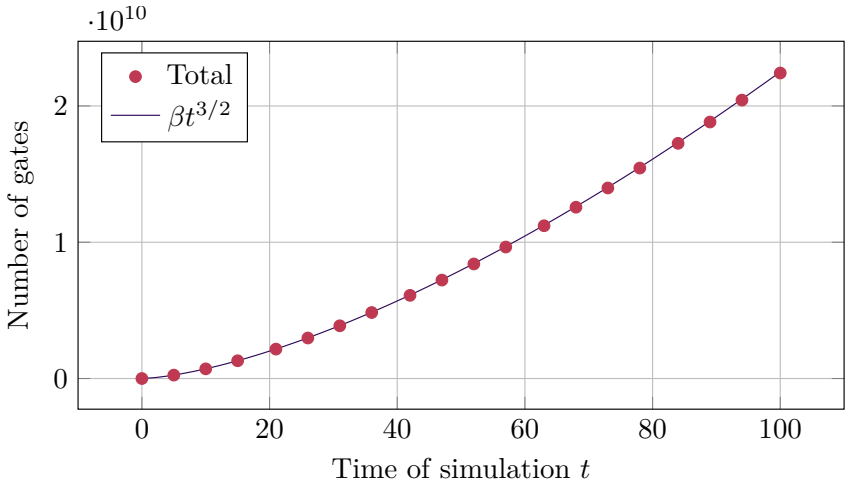
The first characteristic of the wave equation solver that needs to be checked is its validity: is the quantum wave equation solver capable of solving accurately the wave equation as described in Equations (3.1) and (3.2)?

To check the validity of the solver, we used `qat` simulators and Atos QLM to simulate the quantum program generated to solve the wave equation with different values for the number of discretisation points N_d, for the physical time t and for the precision ϵ. Figure 3.17 shows the classical solution versus the quantum solution and the absolute error between the two solutions for $N_d = 32$, $t = 0.4$ and $\epsilon = 10^{-3}$. The solution obtained by the quantum solver is nearly exactly the same as the classical solution obtained with finite differences. The error between the two solutions is of the order of 10^{-7}, which is 4 orders of magnitudes smaller than the error we asked for.

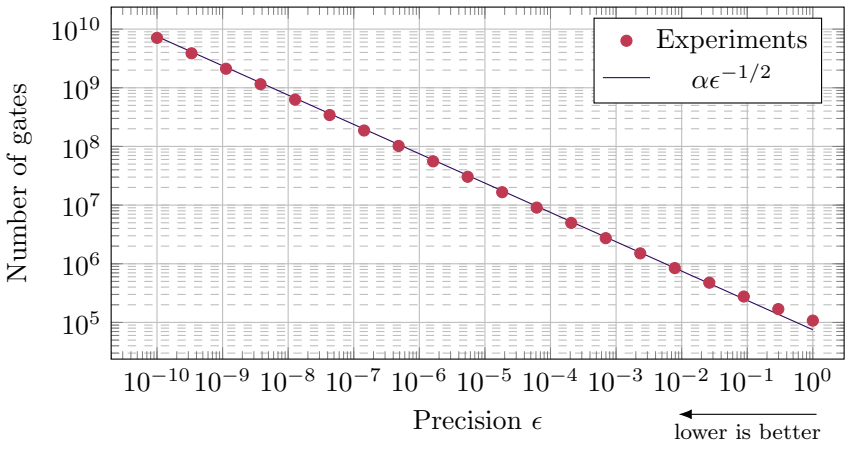
Once the validity of our solver has been checked on multiple test cases, the next interesting property we would like to verify is the asymptotic cost: does the implemented simulator seems to agree with the theoretical asymptotic complexities derived from [56] and [25]?



(a) *Number of quantum gates versus simulated matrix size. The values of the constants* $\gamma = 250\,000$ *and* $\gamma_0 = 2\,000\,000$ *have been chosen arbitrarily to fit the experimental data.*



(b) *Number of quantum gates versus physical time. The value of* $\beta = 39\,000\,000$ *has been chosen arbitrarily to fit the experimental data.*



(c) *Number of quantum gates versus targeted solution precision. The value of* $\alpha = 130\,000$ *has been chosen arbitrarily to fit the experimental data.*

Figure 3.15: *Number of quantum gates needed to simulate the Hamiltonian described in Section 3.2.4 using the oracles implemented following Section 3.2.5. Graphs generated with a Trotter-Suzuki product-formula order* $k = 1$*, 32 discretisation points (i.e.* $n = 6$ *qubits) for Figures 3.15b and 3.15c, a physical time* $t = 1$ *for Figures 3.15a and 3.15c and a precision* $\epsilon = 10^{-5}$ *for Figures 3.15a and 3.15b.*

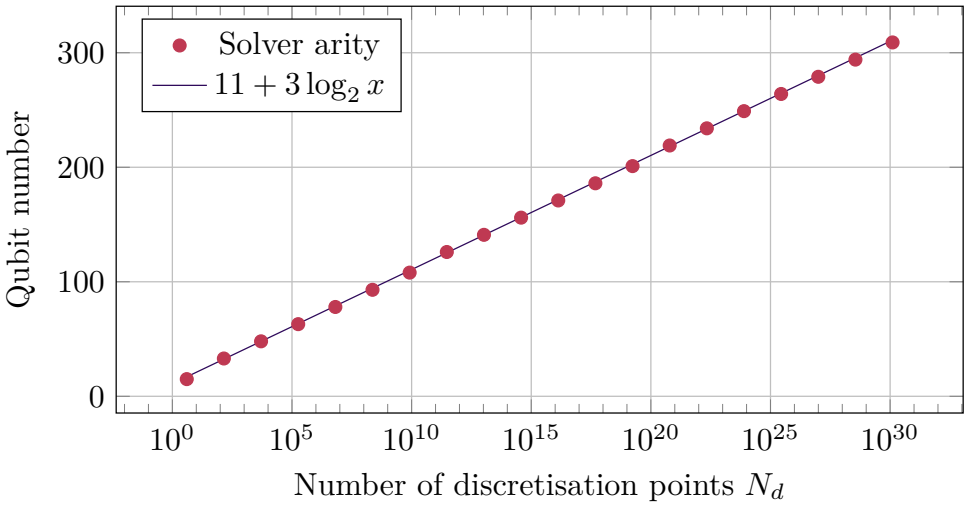


Figure 3.16: *Plot of the number of logical qubits needed to run the wave equation solver for a time $t = 1$, a precision $\epsilon = 10^{-5}$ and a Trotter-Suzuki product-formula of order $k = 1$. The constants offset 11 and factor 3 have been chosen arbitrarily to fit the experimental data. The number of physical qubits needed will depend on their error rate as noted in [104]. Multiplying the number of logical qubits by 3 to 4 orders of magnitude might be a good estimate of the actual number of physical qubits required.*

In our specific case, the Hamiltonian H to simulate can be decomposed in two 1-sparse hermitian matrices, both of them having a spectral norm of 1. The exact decomposition can be found in Section 3.2.4. We chose to let the product-formula order be equal to $k = 1$ and reuse the asymptotic complexity found in Equation (3.52) by changing the time of simulation t by the time $f(t)$:

$$\mathcal{O}\left(5^{2k} n^2 f(t) \left(\frac{f(t)}{\epsilon}\right)^{\frac{1}{2k}}\right). \tag{3.54}$$

Following the study performed in [56],

$$f(t) = \frac{t}{\delta x} = t\,(N_d - 1) \tag{3.55}$$

where δx is the distance between two discretisation points. Moreover, it is possible to prove (see Section 3.2.4) that

$$n = \lceil \log_2(2N_d - 1) \rceil \tag{3.56}$$

Replacing $f(t)$ and n in Equation (3.47) and Equation (3.48) gives us a gate complexity of

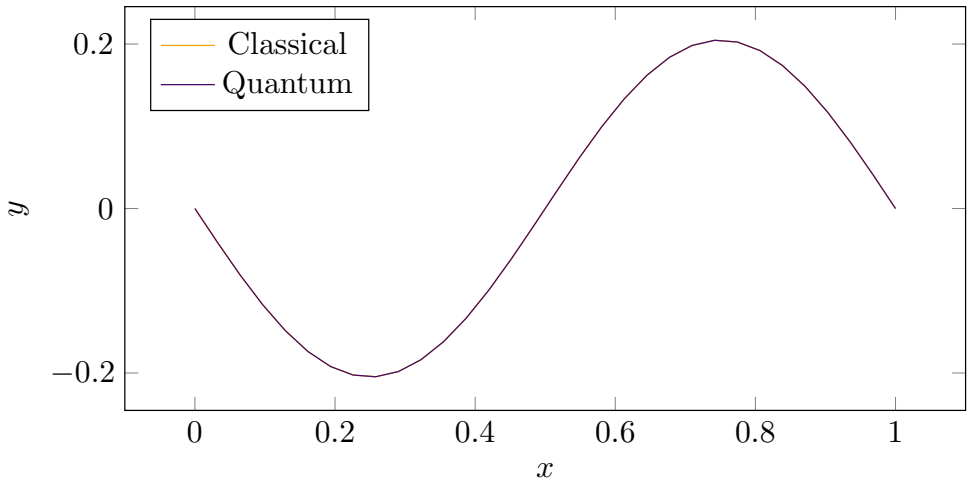
$$\mathcal{O}\left(5^k \log_2 (N_d)^2\right) \tag{3.57}$$

to construct a circuit simulating $e^{-iHt/r}$ and a number of repetitions

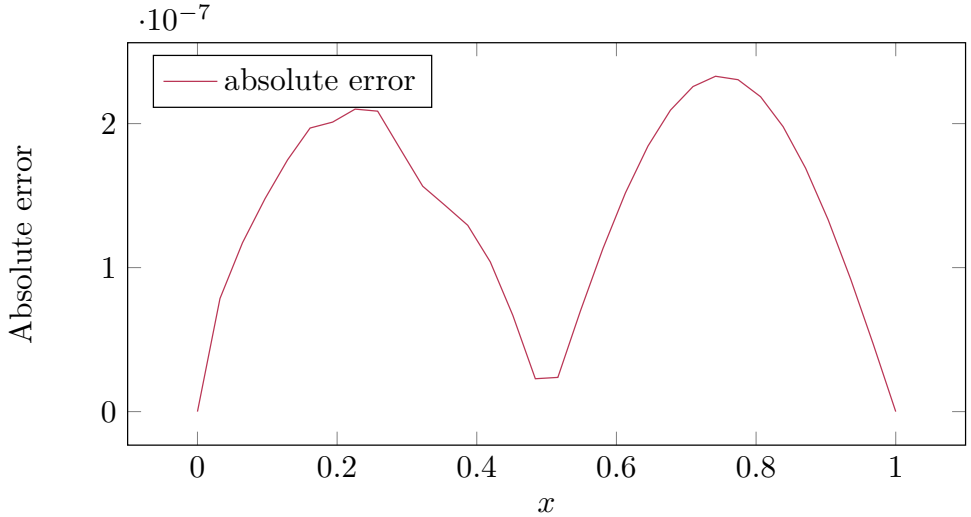
$$r \in \mathcal{O}\left(5^k t N_d \left(\frac{tN_d}{\epsilon}\right)^{\frac{1}{2k}}\right). \tag{3.58}$$

Merging the two expression results in a gate complexity of

$$\mathcal{O}\left(5^{2k} t N_d \log_2(N_d)^2 \left(\frac{tN_d}{\epsilon}\right)^{\frac{1}{2k}}\right). \tag{3.59}$$



(a) *Quantum versus classical solution. Solutions are not visually distinguishable on the graph, see the associated absolute error.*



(b) *Absolute error between the solution obtained by a classical finite-difference solver and the solution computed with the quantum solver.*

Figure 3.17: *Comparison of the classical solver and the quantum solver. Both solvers solved the* 1*-D wave equation with* $N_d = 32$ *discretisation points and a physical time of* $t = 0.4$*. The classical solver uses finite-differences with a very small time-step in order to avoid as much as possible errors due to time-discretisation. The quantum solver was instructed to solve the wave equation with a precision of at least* $\epsilon = 10^{-3}$*, used a Trotter-Suzuki order of* $k = 1$*. The solutions of the two solvers are too close to be able to notice a difference (they overlap on the graph), that is why a second graph plotting the absolute error between the two solvers is included.*

Choosing the Totter-Suzuki formula order $k = 1$ gives us a final complexity of

$$\mathcal{O}\left(N_d^{3/2}\log_2(N_d)^2\frac{t^{3/2}}{\sqrt{\epsilon}}\right) \tag{3.60}$$

to solve the wave equation presented in Equation (3.1). This theoretical result is verified experimentally in Figure 3.18a.

3.3.3 Gate count analysis

Precise subroutines gate counts

Table 3.1 summarises the gate count and ancilla qubit requirements for all the major subroutines used in the wave equation solver implementation. Using the entries of this table, it is possible to compute an estimation of the number of gates required to solve the wave equation.

As explained in Section 3.2.2, we need to simulate each of the 1-sparse Hamiltonians in the decomposition. Aggregating the estimates in Table 3.1 we obtain the costs in Table 3.2 for the Hamiltonian simulation part. Note that the cost of the adder has been voluntarily omitted from the computations in order to be able to compare the cost with different adder implementations. Let a be the gate cost of the adder implementation chosen, the cost of simulating the Hamiltonian needed to solve the 1-dimensional wave equation is: $82n - 35 + 12a_{\texttt{Toffoli}}$ Toffoli gates, $84n + 21 + 9\,[0, n-1] + 12a_{\texttt{CNOT}}$ `CNOT` gates and $\mathcal{O}(n) + 12a_{1-\texttt{qubit}}$ 1-qubit gates.

Choosing an adder implementation and simplifying the gate counts by omitting negligible terms we obtain the gate counts summarised in Table 3.3. It is interesting to note that even if the arithmetic-based adder adds huge constants in the gate count, it does not change the asymptotic complexity whereas Draper's adder changes the number of `CNOT` gates required from $\mathcal{O}(n)$ to $\mathcal{O}(n^2)$.

Impact of the precision requirements

The gate counts presented in Tables 3.1 to 3.3 are only valid when the precision of the solver is not accounted for. When the solver precision matters, an additional step that consists is splitting the Hamiltonian Simulation into r steps needs to be performed as noted in [78, arXiv: Appendix F].

Several bounds exist to determine a $r \in \mathbb{N}^*$ that will analytically ensure that the maximum allowable error ϵ is not exceeded. The definition of such bounds can be found in [78, arXiv: Appendix F] and [108].

The first bound has been devised by analytically bounding the error of simulation due to the Trotter-Suzuki formula approximation by ϵ_0

$$\left\| \exp\left(-it\sum_{j=0}^{m-1} H_j\right) - \left[S_{2k}\left(-\frac{it}{r}\right)\right]^r \right\| \leqslant \epsilon_0 \tag{3.61}$$

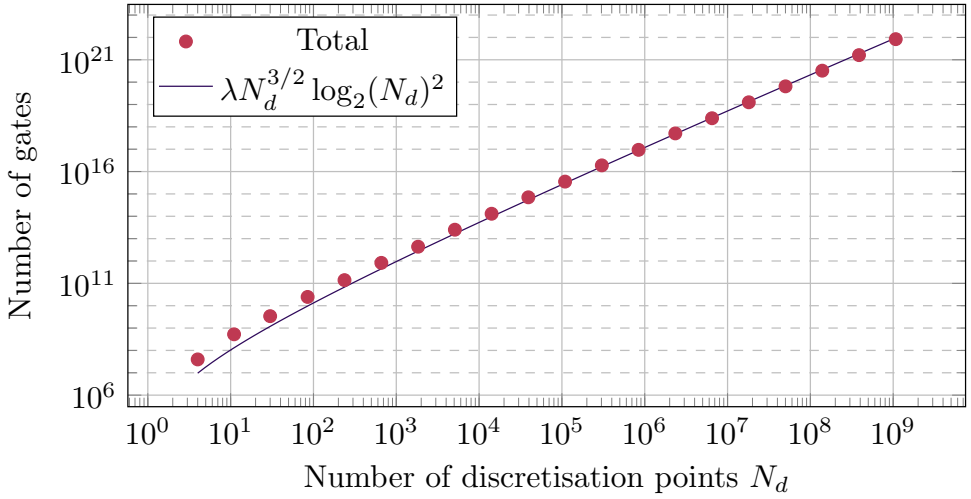
and then let $\epsilon_0 \leqslant \epsilon$ for a given desired precision ϵ. If we let $\Lambda = \max_j ||H_j||$ and

$$\tau = 2m5^{k-1}\Lambda|t| \tag{3.62}$$

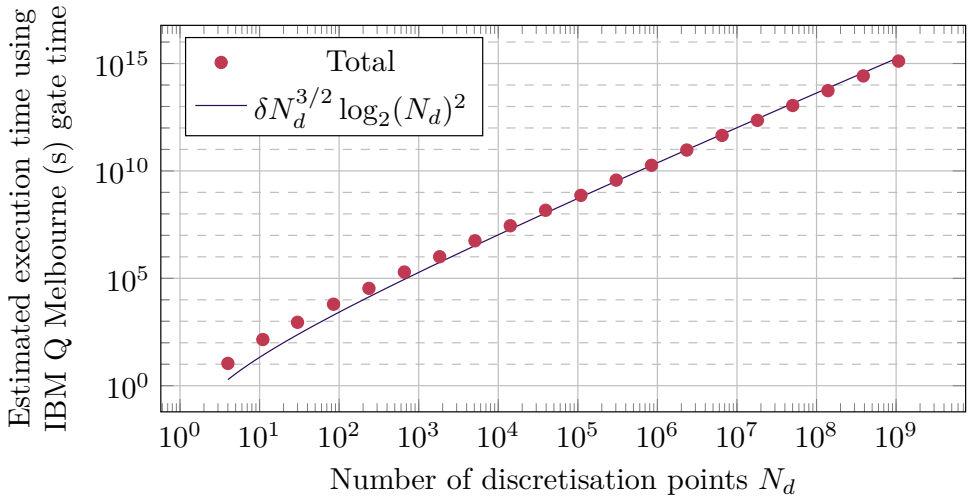
then

$$r_{2k}^{\text{ana}} = \left\lceil \max\left\{\tau, \sqrt[2k]{\frac{e\tau^{2k+1}}{3\epsilon}}\right\}\right\rceil. \tag{3.63}$$

This bound is called the *analytic bound*.



(a) *Number of quantum gates needed to solve the wave equation described in Equation (3.1) versus discretisation size. The value of* $\lambda = 300\,000$ *has been chosen arbitrarily to fit the experimental data.*



(b) *Estimated execution time of the wave equation solver on IBM Q Melbourne hardware. Individual gate times have been extracted from [105] and [103]. GF pulse time has been approximated via arithmetic mean to* 347*ns, GD pulse time is* 100*ns and buffer time is* 20*ns. The value of* $\delta = 0.06$ *has been chosen arbitrarily to fit the experimental data.*

Figure 3.18: *Graphs generated with a Trotter-Suzuki product-formula order* $k = 1$*, a physical time* $t = 1$ *and a precision* $\epsilon = 10^{-5}$*.*

A better bound called the *minimised bound* can be devised by searching for the smallest possible r that satisfies the conditions detailed in [78, Propositions F.3 and F.4]. This bound is rewritten in Equation (3.64).

$$r_{2k}^{\min} = \min \left\{ r \in \mathbb{N}^* : \frac{\tau^{2k+1}}{3r^{2k}} \exp\left(\frac{\tau}{r}\right) < \epsilon \right\} \tag{3.64}$$

Another bound involving nested commutators of the H_i is described in [108] and gives

$$r_{2k}^{\text{comm}} \in \mathcal{O}\left(\frac{\alpha_{comm}^{\frac{1}{2k}} t^{1+\frac{1}{2k}}}{\epsilon^{\frac{1}{2k}}} \right) \tag{3.65}$$

where k is the order of the product-formula used, t the time of simulation, ϵ the error and

$$\alpha_{comm} = \sum_{i_0, i_1, \ldots, i_p = 0}^{m-1} \left|\left| [H_{i_p}, \ldots [H_{i_1}, H_{i_0}]] \right|\right| . \tag{3.66}$$

Once the value of r has been computed, the quantum circuit simulating the matrix H for a time $\frac{t}{r}$ should be repeated r times. This adds a factor of r in front of all the gate counts computed in Tables 3.1 to 3.3.

Impact of error-correction

The final implementation of the solver requires a large number of quantum gates as can be seen in Figure 3.18a. But quantum chips are currently experiencing high levels of noise, most of the time above 10^{-4} for single-qubit gates and even higher for measurement and 2-qubit gates, which rules out the possibility to run such a large quantum circuit on a bare-bone NISQ chip (see Definition 11). One particularly interesting improvement to quantum chips is error-correction that is able to "build" error-free qubits by using several noisy ones.

Definition 11 (Noisy Intermediate-Scale Quantum (NISQ) chip)**.** The NISQ acronym has been introduced in [109] and refers to quantum chips with 50 to a few hundreds imperfect qubits that experience noise.

When error-correction is studied, two gates are particularly important: T and Toffoli gates. The T gate has a prohibitive cost when compared to the Clifford quantum gates and implementing a Toffoli gate requires 7 of such T gates as noted in [104] and [110, Fig. 1].

Table 3.4 summarise the cost of the non Clifford quantum gates used in the implementation of the 1-dimensional wave equation solver. The rotation gates need to be approximated. One solution to approximate the R_n and P_h gates is given in [106]. In order to obtain practical results as opposed to theoretical ones, we chose to use the number computed in [111, Table 1].

The final T-count is summarised in Figure 3.19. From Figure 3.19b it is clear that the `add_arith` implementation is more efficient than the `add_qft` one.

3.4 Additional work

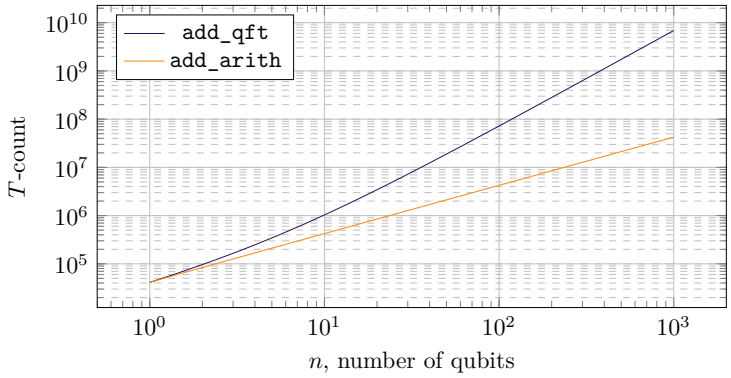
3.4.1 Implementation of higher-order Laplacians

The results shown in Section 3.3 have all been generated using the second-order discretisation formula shown in Equation (3.15). Higher-order formulas are studied in [56, VI, VII and C].

Note 8. As shown in [56, VII.B], higher-order discretisations with Neumann boundary conditions are not implementable using the algorithm described in Section 3.2.4.

Adder used	T-count
`add_qft`	$6870n^2 + 34660n - 245$
`add_arith`	$42510n - 1225$

(a) *Number of T-gates needed to simulate the Hamiltonian used to solve the 1-dimensional wave equation depending on the adder implementation used. Based on Table 3.3 and Table 3.4.*



(b) *Plot of the T-count devised in Figure 3.19a for the two different adder implementations.*

Figure 3.19: *Analysis of the T-count of the 1-dimensional wave equation solver quantum implementation with respect to the adder implementation used.*

In this appendix we replace the second-order formula given in Equation (3.15) and used in this manuscript by the fourth-order formula given in [56, Eq. (46)] and re-written below

$$\frac{\partial^2\phi}{\partial x^2}(i\delta x, t) \approx \frac{1}{\delta x^2}\left(\frac{5}{2}\phi_{i,t} - \frac{4}{3}\left(\phi_{i-1,t} + \phi_{i+1,t}\right) + \frac{1}{12}\left(\phi_{i-2,t} + \phi_{i+2,t}\right)\right). \tag{3.67}$$

We are left to devise the matrix B_d^4 that satisfy $B_d^4 B_d^{4\dagger} = \Delta_4$ where Δ_4 is the discretisation matrix arising from the fourth-order finite-differences approximation in Equation (3.67).

[56, Eq. (47) and VII.C] devised an analytic formula for $\hat{B}_d^4$, the B_d^4 matrix with periodic boundary conditions, using the matrix $\hat{S}$ representing the cyclic permutation $\{1, 2, \ldots, N\}$ with entries $\hat{S}_{i,j} = \delta_{i,(j+1 \mod N)}$ as shown in Equation (3.68).

$$\hat{S} = \begin{pmatrix} 0 & 0 & \cdots & \cdots & \cdots & 0 & 1 \\ 1 & 0 & & & & & 0 \\ 0 & \ddots & \ddots & & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & 0 \end{pmatrix} \tag{3.68}$$

With this definition of $\hat{S}$, the analytic formula for $\hat{B}_d^4$ is given in Equation (3.71), with b and c being solution of [56, Phys. Rev. A, Eqs. (53,54,55)]. The exact values for b and c are:

$$b = \pm\frac{1}{2\sqrt{3}\sqrt{7 \pm 4\sqrt{3}}} \tag{3.69}$$

$$c = \pm\sqrt{\frac{7 \pm 4\sqrt{3}}{12}} \tag{3.70}$$

with the $\pm$ signs that can be chosen freely. Note that, $b = \pm\frac{1}{2\sqrt{3}\sqrt{7\pm4\sqrt{3}}}$ and $c = \frac{1}{12b}$ are irrational because $\sqrt{3}\sqrt{7 \pm 4\sqrt{3}} = \sqrt{3}\sqrt{2 + \sqrt{3}}^2 = \sqrt{3}\left(2 + \sqrt{3}\right) = 2\sqrt{3} + 3$ is irrational.

Equation (3.72) shows the matrix shape with its entries.

$$\hat{B}_d^4 = c\hat{S} - (b+c) * \mathbb{I} + b\hat{S}^\dagger \tag{3.71}$$

$$\approx \begin{pmatrix} b+c & b & 0 & \cdots & \cdots & \cdots & 0 & c \\ c & b+c & b & \ddots & & & & 0 \\ 0 & c & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & b & 0 \\ 0 & & & & \ddots & c & b+c & b \\ b & 0 & \cdots & \cdots & \cdots & 0 & c & b+c \end{pmatrix} \tag{3.72}$$

Because periodicity has not been studied in the main use-case of this manuscript, we would like to also remove the need of periodic boundary conditions in this higher-order Laplacian discretisation. This can be achieved by removing the upper-right entry of $\hat{S}$ by changing it from 1 to 0. The resulting matrix S is shown in Equation (3.73).

$$S = \begin{pmatrix} 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 1 & 0 & & & & & \vdots \\ 0 & \ddots & \ddots & & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & 0 \end{pmatrix} \tag{3.73}$$

Using the exact same formula we can devise B_d^4:

$$B_d^4 = cS - (b+c) * \mathbb{I} + bS^\dagger \tag{3.74}$$

$$\approx \begin{pmatrix} b+c & b & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ c & b+c & b & \ddots & & & & \vdots \\ 0 & c & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & b & 0 \\ \vdots & & & & \ddots & c & b+c & b \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & c & b+c \end{pmatrix} \tag{3.75}$$

Replacing B_d^4 in Equation (3.19) we obtain

$$\widetilde{H_d^4} = \frac{1}{\delta x}\begin{pmatrix} 0 & \cdots & \cdots & \cdots & 0 & b+c & b & 0 & \cdots & 0 \\ \vdots & & & & \vdots & c & b+c & b & \ddots & \vdots \\ \vdots & & & & \vdots & 0 & c & \ddots & \ddots & 0 \\ \vdots & & & & \vdots & \vdots & \ddots & \ddots & b+c & b \\ 0 & \cdots & \cdots & \cdots & 0 & 0 & \cdots & 0 & c & b+c \\ b+c & c & 0 & \cdots & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ b & b+c & \ddots & \ddots & \vdots & \vdots & & & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 & \vdots & & & & \vdots \\ \vdots & \ddots & \ddots & b+c & c & \vdots & & & & \vdots \\ 0 & \cdots & 0 & b & b+c & 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}. \tag{3.76}$$

Claim 4. One of the main difference with the second-order approximation used all along this paper is that, with the fourth-order approximation, the entries of the matrix $\widetilde{H_d^4}$ are no longer multiples of a common number $\alpha \in \mathbb{R}$.

Proof. It is convenient to first remark that

$$c = \left(7 \pm 4\sqrt{3}\right) b. \tag{3.77}$$

Now let assume that $\exists \alpha \in \mathbb{R}$ such that $\exists (k_1, k_2, k_3) \in \mathbb{Z}$ verifying

$$\begin{cases} b = \alpha k_1 \\ c = \alpha k_2 \\ b + c = \alpha k_3 \end{cases}. \tag{3.78}$$

The 3rd equality is trivially a reformulation of the 2 others as if $\exists\, (k_1, k_2) \in \mathbb{Z}^2$ satisfying the first 2 conditions, then the 3rd condition is also verified with $k_3 = k_1 + k_2$. Moreover, as $b \neq 0$ and $c \neq 0$, $\alpha \neq 0$ so we can divide both sides of both of the remaining equations by α to obtain the equivalent system of equations:

$$\begin{cases} k_1 = \dfrac{b}{\alpha} \\ k_2 = \dfrac{c}{\alpha} \end{cases}. \tag{3.79}$$

This system can be reformulated using Equation (3.77) as

$$\begin{cases} k_1 = \dfrac{b}{\alpha} \\ k_2 = \left(7 \pm 4\sqrt{3}\right) k_1 \end{cases} \tag{3.80}$$

This is a contradiction as $7 \pm 4\sqrt{3}$ is irrational, so by definition of an irrational number $k_2 = \left(7 \pm 4\sqrt{3}\right) k_1$ cannot be an integer. □

This means that we cannot write down $\widetilde{H_d^4}$ as an integer weighted matrix multiplied by a real number, and so the trick used in Section 3.2.4 to simulate the integer weighted matrix H_d for a time αt is no longer applicable.

Consequently, and independently of the decomposition we use for $\hat{H}_d^4$, at least one of the matrices in the decomposition of $\hat{H}_d^4$ will not be "easy to simulate" as defined in Definition 8.

Ultimately, the main consequence of this observation is that we will have to use a real-weighted Hamiltonian simulation procedure. Such a procedure can be found in [79] but requires to approximate the real-weighted entries with a fixed-point representation that has at least 2 evident caveats:

1. It is impossible to encode both b and c exactly with a fixed-point representation as shown in Section 3.4.1. This means that we add another layer of approximation, even before the approximation caused by the use of a product-formula.

2. The Hamiltonian simulation procedures used for real numbers requires more qubits. More precisely, the number of additional qubits required depends on the desired precision ϵ and grows as $-\log_2(\epsilon)$.

Note 9. Even if the H_d^4 matrix seems quite hard to simulate, it is still a 3-sparse matrix. This means that it is still manageable to hand-write the oracles. Moreover, having a small number of matrices in the decomposition helps in reducing the error introduced by product-formulas.

3.4.2 Optimisation of the implementation

Once the correctness of the implementation validated, one of the most important remaining work is to try to optimise the implementation. The optimisation of a software is often performed as an iterative task.

The first step is to define a figure of merit, a quantity we want to minimise during the optimisation process. Among the most obvious figures of merit are the total number of gates, the number of CNOT gates or the total execution time of the quantum program. More complex quantities can also be considered, such as the execution time using error correction codes or the final state fidelity. In this chapter we decided to take into account an estimation of the total execution time of the quantum program on an imaginary device that shares today's chips characteristics.

The second step of the optimisation process consists in isolating the subroutines that contribute the most to the figure of merit. As an example, if the quantum program spend 90% of the total execution time in one subroutine, this subroutine should be the first place to look for optimisations.

After the isolation of one or two subroutines, the actual optimisation can take place. The goal of this third step is to decrease the impact of the subroutines considered on the overall figure of merit without changing the final result of the implementation.

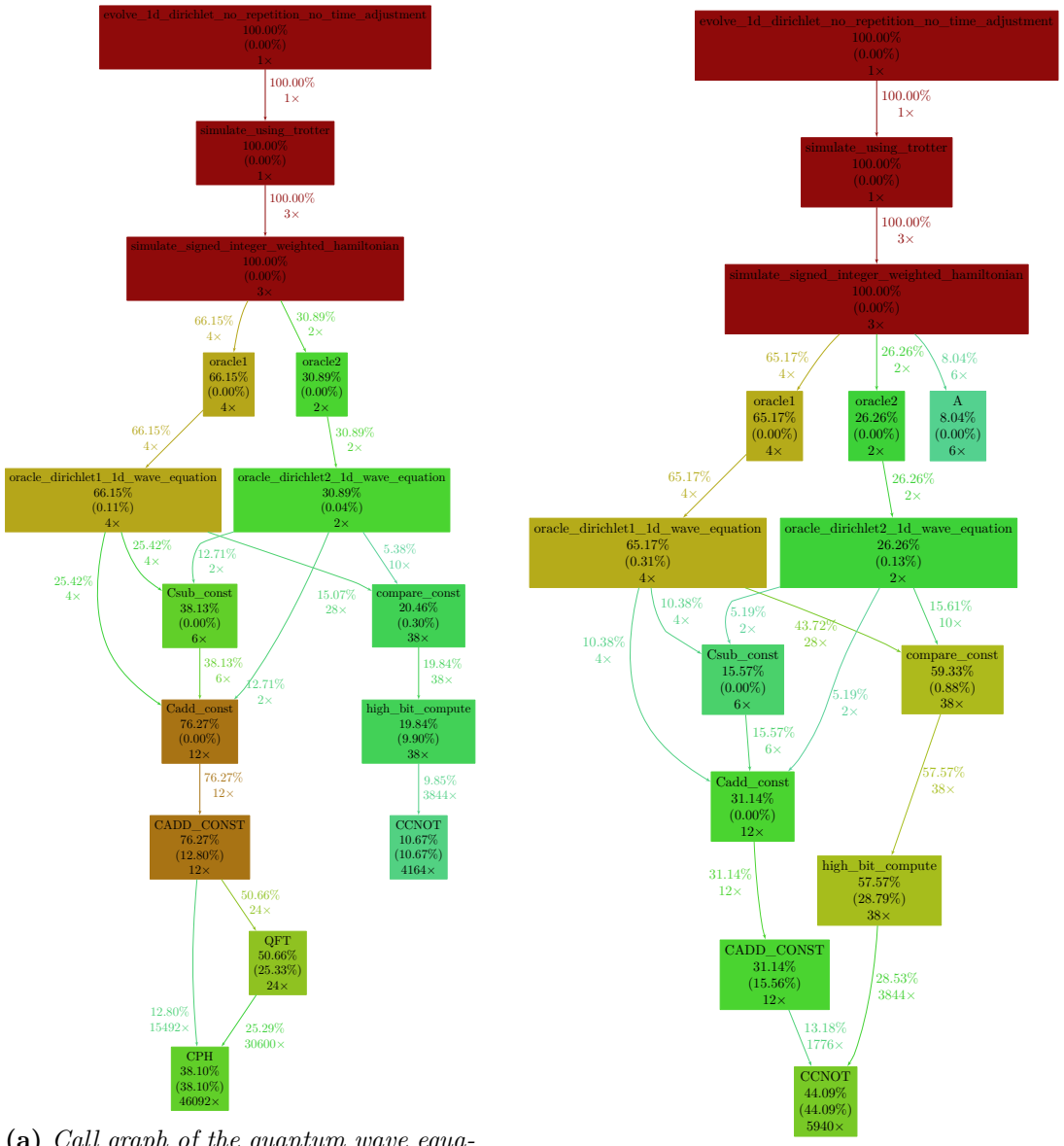
Finally, once the optimisation is performed, the optimisation process can be repeated by re-starting at the second step, until the program is considered sufficiently optimised.

One of the main difficulty we encountered when applying this optimisation process was to correctly isolate the most time-consuming subroutines. In classical computing, this step is usually performed with specialised tools such as `gprof` or a more advanced profiler, but no such tool exist for quantum programs. In order to fill this gap we developed `qprof`, a tool that analyses a quantum program and generates a report similar to the one generated by `gprof`. Using `qprof` and some of the various tools compatible with `gprof`, we plotted the call-graph shown in Figure 3.20a. The qprof tool is presented in greater detail in Chapter 4.

From this call-graph, it is clear that the adder is *the* most costly subroutine and that it should be optimised. The adder internally uses the Quantum Fourier Transform (QFT), which takes more than 50% of the total execution time. The issue is that the QFT implementation is already very concise and we do not expect to be able to optimise it enough to cut significantly its overall cost. This leads us to the conclusion that a new algorithm that do not require the QFT should be used to implement an adder. Such an algorithm can be found in [107].

Changing the implementation of the adder from Draper's adder to the arithmetic-based adder from [107] improves drastically the total execution time of the quantum program and produce

the call-graph in Figure 3.20b, which shows that, everything else unchanged, the relative cost of the adder over the total cost went from 76.24% when using Draper's adder to 31.14% with the arithmetic-based adder.



(a) *Call graph of the quantum wave equation solver using Draper's adder (QFT-based).*

(b) *Call graph of the quantum wave equation solver using an arithmetic-based adder.*

Figure 3.20: *All the gates or subroutines that account for less than 5% of the total execution time are not displayed. Execution times for u1, u2, u3 and cx gates have been averaged over all the data available for the quantum chip IBMQ Melbourne. Using the arithmetic-based adder, the overall execution time improved by a factor of* 31.

3.5 Discussion

In this chapter, we focused on the implementation of a PDE solver with a focus on the practical cost of implementing a 1-dimensional quantum wave equation solver on a quantum computer. We showed that a quantum computer is able to solve partial differential equations by constructing and simulating the quantum circuits described. We also studied the scaling of the solver with

respect to several parameters of interest and show that the theoretical asymptotic bounds are mostly verified.

Several challenges have been encountered during the development of the research presented in this chapter. One of the most time-consuming part in terms of development was the implementation of the different oracles, which also turned out to be very error-prone and hard to debug. The implementation of oracles will likely still be a necessary step for all the practical implementations of partial differential equation solvers and would, in our opinion, require more automatised tools to assist the work.

Recent developments introduced the possibility to describe the quantum oracle using a domain-specific language and generate the corresponding code automatically, simplifying the development process by allowing the developer to focus on the high-level description. This is for example exactly the purpose of the recently introduced HODL library [112] or of the commercial software developed by Classiq [113].

3.6 Supplementary material

The implementation of the quantum wave equation solver is available at `https://gitlab.com/cerfacs/qaths`. The `qprof` tool is available at `https://gitlab.com/qcomputing/qprof/qprof`.

Gate	Toffoli count	CNOT count	1-qubit gate count	# ancillas	notes
or	1	0	5	0	
QFT	0	$3\left(2n^2-2n+\lfloor\frac{n}{2}\rfloor\right)$	$2\left(n^2+n\right)$ H $4\left(n^2-n\right)$ T $\frac{n^2-n}{2}$ R_n	1 $\|0\rangle$-init	R_n gates might need to be decomposed [106].
add_arith	$20n-10$	$22n$	0	$n-1$ $\|0\rangle$-init	See [107].
add_qft	0	$6\left(2n^2-2n+\lfloor\frac{n}{2}\rfloor\right)$	$2\left(n^2+n\right)$ H $4\left(n^2-n\right)$ T $3\frac{n^2-n}{2}$ R_n	1 $\|0\rangle$-init	See QFT note on R_n. Figure 3.7.
sub_qft	0	$6\left(2n^2-2n+\lfloor\frac{n}{2}\rfloor\right)$	$2\left(n^2+n\right)$ H $4\left(n^2-n\right)$ T $3\frac{n^2-n}{2}$ R_n $2n$ X	1 $\|0\rangle$-init	See QFT note on R_n. Figure 3.5.
CARRY	$2(n-1)$	$2+[0,n-1]$	$2n+[0,n-1]$ X	$n-1$ borrowed	See [100].
n-contr. CNOT	$4n$	0	0	n borrowed	See [101].
eq	$4n$	0	$2\,[0,n]$ X	n borrowed	Figure 3.9.
cmp	$2\,(n-1)$	$2+[0,n-1]$	$4n+[0,n-1]$ X	$n-1$ borrowed	See CARRY and Section 3.2.5.
A	$2n$	$4n$	$3n$ H $3n$ S $2n$ T $2n$ X	0	See [79, Fig. 4.3.].
$e^{-iZ\otimes Z\otimes Ft}$	$8n$	$24n$	$36n$ P_h 8 X	0	Adapted from [79, Fig. 4.6]
1-sparse HS	$10n$	$28n$	$3n$ H $3n$ S $2n$ T $2n+8$ X $36n$ P_h	0	Oracle implementation cost not included. 2 calls to the oracle are required. Figure 3.3.
M_1	$4\,(n-1)$	$5+2\,[0,n-1]$	$10n+2+[0,n-1]$ X	1 $\|0\rangle$-init $n-1$ borrowed	add implementation cost not included. 2 calls to add are required. Figure 3.10.
V_1	$2\,(n-1)$	$2+[0,n-1]$	$4n+[0,n-1]$ X	$n-1$ borrowed	Figure 3.11.
S_1	0	0	0	0	See Equation (3.41).
M_{-1}	$4\,(n-1)$	$5+2\,[0,n-1]$	$10n+2+[0,n-1]$ X	1 $\|0\rangle$-init $n-1$ borrowed	add implementation cost not included. 2 calls to add are required. Figure 3.12.
V_{-1}	$2\,(n-1)$	$2+[0,n-1]$	$4n+[0,n-1]$ X	$n-1$ borrowed	Figure 3.13.
S_{-1}	$16n+1$	0	$5+8\,[0,n]$ X	n borrowed	Figure 3.14.

Table 3.1: *Precise gate count for the most important subroutines used in the quantum implementation of the wave equation solver. n always represent the size of the input(s), except for the n-controlled CNOT where n is the number of controls. When the number of gates depends on a generation-time value, the range of all the integer values possible is shown with square brackets. For example, $[0, n-1]$ means that, depending on the generation-time value provided, the number of gates will be an integer between 0 and $n-1$ included. $|0\rangle$-init ancillas represent the standard ancilla-type: qubits that are given in the state $|0\rangle$ and should be returned in that exact same state. On the other side, borrowed ancillas can be given in any state and should be returned in the exact same state they were borrowed in.*

Unitary	Toffoli count	CNOT count	1-qubit gate count	# ancillas	notes
$e^{-iH_1 t}$	$22n - 12$	$28n + 7 + 3\,[0, n-1]$	$3n\ H \quad 3n\ S$ $2n\ T \quad 36n\ P_h$ $30n + 10 + 2\,[0, n-1]\ X$	1 $\|0\rangle$-init $n-1$ borrowed	`add` implementation cost not included. 4 calls to `add` are required.
$e^{-iH_{-1} t}$	$38n - 11$	$28n + 7 + 3\,[0, n-1]$	$3n\ H \quad 3n\ S$ $2n\ T \quad 36n\ P_h$ $30n + 15 + 10\,[0, n]\ X$	1 $\|0\rangle$-init $n-1$ borrowed	`add` implementation cost not included. 4 calls to `add` are required.
e^{-iHt}	$82n - 35$	$84n + 21 + 9\,[0, n-1]$	$9n\ H \quad 9n\ S$ $6n\ T \quad 108n\ P_h$ $90n + 35 + 14\,[0, n]\ X$	1 $\|0\rangle$-init $n-1$ borrowed	`add` implementation cost not included. 12 calls to `add` are required.

Table 3.2: *Number of gates and ancillas needed to simulate the easy-to-simulate Hamiltonians H_1 and H_{-1} that are part of the decomposition of H as well as e^{-iHt}. It is important to realise that the gate counts for e^{-iHt} are only valid up to a given t or ϵ (once one is fixed, the value of the other can be computed). In order to make the gate count generic for any t and ϵ, the number of repetitions should be computed (see n in Equation (3.13)). Note that some of the $[0, n-1]$ ranges have been simplified to $[0, n]$ for conciseness.*

Adder used	Toffoli count	CNOT count	1-qubit gate count	# ancillas
`add_qft`	$82n - 35$	$144n^2 - 60n$	$24n^2 + 25n\ H \quad 9n\ S$ $48n^2 - 42n\ T \quad 108n\ P_h$ $18n^2 - 18n\ R_n$ $114n + 35 + 14\,[0, n]\ X$	2 $\|0\rangle$-init $n-1$ borrowed
`add_arith`	$222n - 175$	$348n + 21 + 9\,[0, n-1]$	$9n\ H \quad 9n\ S$ $6n\ T \quad 108n\ P_h$ $90n + 35 + 14\,[0, n]\ X$	n $\|0\rangle$-init $n-1$ borrowed

Table 3.3: *Number of gates and ancillas needed to simulate the Hamiltonian used to solve the 1-dimensional wave equation depending on the adder implementation used. It is important to realise that the gate counts for e^{-iHt} reported in this table are only valid up to a given t or ϵ (once one is fixed, the value of the other can be computed). In order to make the gate count generic for any t and ϵ, a number of repetitions r should be computed (named n in Equation (3.13) and studied in [78, arXiv: Appendix F] and [108]). Note that the gate counts have been simplified by removing negligible terms when possible.*

Gate	T count	Notes
T	1	
S	2	
`CCNOT`	7	See [104].
P_h	379	$\epsilon = 10^{-15}$, approximated from [111].
R_n	379	$\epsilon = 10^{-15}$, approximated from [111].

Table 3.4: *T-gate cost of the non Clifford quantum gates used in the wave equation solver implementation.*

Part III

Algorithm analysis

Chapter

4 qprof

This chapter deals with a crucial part of software development: debugging and optimising programs in order to ensure their correctness and efficiency. These tasks are of critical importance in any program implementation and require a deep and comprehensive knowledge of the analysed implementation along with efficient ways to isolate bugs or bottlenecks. The work presented in this chapter provides a new and human-readable way to visualise highly complex quantum programs. qprof offers unique insights on the quantum circuits studied and greatly improve the manual optimisation process and visualisation of a given implementation. The tool has been presented in [114].

Contents

4.1 Introduction

The quantum computing field has been evolving at an increasing rate in the past few years and is currently gaining more traction. Several quantum chips, the underlying hardware that enable researchers and companies to run quantum algorithms, have been announced by different research teams. The error rates and number of qubits provided by these chips have greatly improved in the last few years, with quantum hardware reaching up to 127 qubits in the end of 2021 [115].

Software has also seen a tremendous rise with the emergence of several quantum computing frameworks and languages such as Qiskit [116], Q# [117], PyQuil [118], Cirq [119] or myQLM [120] to name a few. These frameworks help in speeding-up the process of implementing a quantum algorithm by providing their own "standard library". Most of them also include specialised libraries whose purpose is to facilitate the development and testing of new quantum algorithms. For example, all the quantum computing frameworks cited previously include a library to simulate quantum circuits, some even implement several simulation algorithms such as a full state-vector simulator, a simulator for stabiliser circuits [121, 122] or a simulator using matrix-product states [123, 124]. Most of the frameworks that target real quantum chips also include libraries to characterise a given quantum hardware, using for example randomised benchmarking [125–129] methods, or hardware noise mitigation [130, 131].

Finally, a large majority of the quantum computing frameworks provide a way to automatically optimise a quantum circuit. This optimisation is often performed during compilation, when the abstract quantum circuit representation is translated to be compliant with the targeted hardware. Automatic optimisation of quantum circuits is a broad area of research with algorithms based on pattern-matching [132–134], gate optimisation algorithms [135, 136] or even pulse-level optimisation [137–139].

But even though automatic optimisation has already been shown to be successful in optimising complex quantum circuits [29], most algorithms only perform local optimisations, most of the time on a flattened quantum circuit, without prior knowledge of the algorithms used to construct the circuit.

Identifying the usage of a non-optimal algorithm in the implementation and replacing it with a more efficient one is, for example, an optimisation that cannot be performed in general by automatic optimisers. This improvement should rather be spotted and optimised by the developer.

Currently, the only way one has to optimise a given quantum implementation beyond what is provided by automatic methods is "trial and error". First, try to locate a "hot spot" (i.e. a subroutine that takes a considerable amount of resources) in the implementation, either by a tedious theoretical analysis or a manual counting of the routine calls. Then, optimise the hot spot found, either by improving the implementation or using a better algorithm. Finally check if the optimisation performed improved the overall performance of the implementation. This process has a severe drawback that makes it impractical on real-world implementations: the first step that consists in finding the hot spots is either imprecise or potentially very long, tedious and error-prone on large implementations.

qprof aims at replacing this manual, tedious and error-prone step by automatically generating a report with all the useful information needed to find the hot spots of the given quantum program implementation. The qprof tool has been strongly inspired by classical profilers such as `gprof` [140, 141] which try to solve the exact same issue but in classical (non-quantum) programming.

The chapter is organised as follows. In Section 4.2, we review the related work around classical profilers and quantum resource estimation. Section 4.3 explains the internals of qprof and details its architecture, the design choices made, and their impact on the tool efficiency, extensibility and usability. We then include in Section 4.4 a theoretical and practical analysis of

the tool runtime. Code snippets and practical examples are provided in Section 4.5 to illustrate the tool usage. Finally, we discuss some of the limitations and potential improvements of qprof in Section 4.6.

4.2 Related work

4.2.1 Classical profilers

Classical profilers are tools that have been used since the beginning of programming languages back in the 1970 decade. One of the first profiler was `prof`, included in the Linux kernel in 1972 [142]. `gprof` [140] came out in 1982, extending `prof` by performing a complete call-graph analysis. Since then, a lot of different profilers using different methods to profile programs were introduced, each of the profiling methods having its strengths, weaknesses and compromises.

For example statistical profilers, that sample the program call-stack at regular intervals, are less precise due to their finite sampling rate but have a very low overhead (for example, between 1 and 3% as reported by the maintainers of OProfile [143], a statistical profiler, on the tool's FAQ). On the other side of the spectrum, instead of executing the profiled program directly on the target hardware, "Instruction Set Simulators" can be used to run the program in an isolated and entirely controlled environment. Profilers using this technique have the advantage of being very accurate and allow the collection of a large variety of indicators, with the drawback of a considerable runtime overhead. Another technique used by some profilers such as `gprof` [140] is to instrument the code. The information that can be gathered by this kind of profilers is less exhaustive than the instruction set simulator method, but the overhead they add to the program runtime execution is in general relatively low. Finally, some profilers use static analysis in order to gather data without even executing the program. For classical computers, these profilers are limited to information such as the instruction count and variations thereof due to the highly complex way current classical processors execute instructions.

Independently of the method used by the profiler, its goal is to gather data about the profiled program execution in order to give a synthetic and readable report to the user. This report will most of the time be used to find one or several "hot spots", which are portions of code or functions that take a considerable amount of an important resource, frequently the total execution time. Finding hot spots is a necessary step to optimise the implementation of the profiled program as it allows to isolate small portions of code that should be improved in order to lower down the amount of resources needed by the program.

A profiling report obtained thanks to the `gprof` profiler has been included in Figure 4.1 with a simple C code in Figure 4.1a and the resulting profiling report in Figure 4.1b.

4.2.2 Quantum profilers

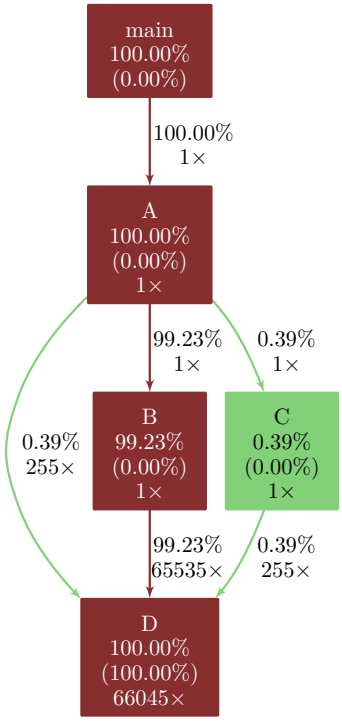
Most of the quantum computing frameworks available today only provide basic resource estimate capabilities that range from a shallow analysis of the quantum circuit to a report containing the gate counts for some fixed set of gates. Moreover, all the framework analysed are only able to analyse quantum circuits written in their ecosystem, without any cross-framework capabilities.

This is for example the case of Qiskit that performs a shallow analysis of its `QuantumCircuit` instances by using the `count_ops` method, returning a dictionary containing the number of times each subroutine is called. Note that this method is limited as it does not recurse into the subroutines called by the main routine. The myQLM framework provides the same features with its `Circuit.statistics` method.

The ScaffCC compiler [144] provides a little bit more information than Qiskit and myQLM by computing the gate count (for the gates $\{X, Z, H, T, T^\dagger, S, S^\dagger, CX\}$) for each routine encountered in the compiled quantum program. This report is useful to perform cost estimation, but the list

```
void D(void) {
    for(unsigned count = 0; count < 0xFFFF; count++);
}

void C(void) {
    for(unsigned count = 0; count < 0xFF; count++)
        D();
}

void B(void) {
    for(unsigned count = 0; count < 0xFFFF; count++)
        D();
}

void A(void) {
    B();
    C();
    for(unsigned count = 0; count < 0xFF; count++)
        D();
}

int main(void) {
    A();
    return 0;
}
```

(a) *C code to be profiled by* `gprof` *and compiled with* `gcc -pg -o profile-exec profile.c`.



(b) *Profiling report obtained with* `gprof` *and post-processed with the* `gprof2dot` *tool.*

Figure 4.1: *Example of call graph that can be generated with* `gprof` *on a trivial classical program written in C and compiled with* `gcc`*. Even though one could count manually the number of operations performed by each functions on such a trivial program, using* `gprof` *is less error-prone and outputs a clear view of the program hot spot: the function* `D`*. Someone wanting to optimise the execution time of this simple program can directly see on* `gprof` *report that there are only* 3 *potential approaches to reduce the program runtime: optimise* `D` *directly, reduce the number of times* `B` *is calling* `D` *or remove the unique call to* `B` *from* `A` *and replace it with something more efficient.*

of basis gates used does not seem to be modifiable and the information about which routine is calling which subroutine is lost.

Quipper [145], a quantum computing framework written in Haskell, has been created specifically to perform resource estimations on huge quantum circuits in an efficient manner. However, even though very efficient, the Quipper framework seems limited to compute simple features such as the total number of gates, total number of qubits or total number of ancillary qubits.

Finally, Q# has some interesting proofs of concepts on one specific implementation of Shor's algorithm. Using Q# Trace Simulator, a Flame graph [146] exporter has been built. This exporter is only able to count one type of gate from a fixed set.

Each of the four examples provided in this section are limited to one specific quantum computing framework and cannot be easily re-used to analyse quantum circuits built with other frameworks. Moreover, half of the frameworks are only performing a shallow exploration, stopping at the first level (i.e. stopping at the subroutines called directly by the profiled routine and not recursing into deeper subroutines). Finally, none of the profiling features provided by the four frameworks presented above have a direct way to deal with gates of variable execution time that can be found in real hardware.

Note 10. Q# profiles quantum programs by "executing" them on a fake quantum processor that will track and record data on the execution of the quantum program. Consequently, Q# profiler should, in theory, be capable of handling dynamic quantum circuits (quantum circuits that contains quantum measurements and that adapt the gates executed according to the measurement result). Due to its static approach, and as discussed in Section 4.6.6, qprof is not able to analyse dynamic quantum circuits yet.

4.3 How does qprof works?

4.3.1 General structure

The general structure of qprof is composed of 3 main parts that interact with each other: a framework-agnostic quantum circuit representation, core data structures and logic, and several exporters. The framework-agnostic representation of a quantum circuit has been packaged in another Python package named qcw.

The overall workflow of qprof is schematically explained in Figure 4.2. In this workflow, qprof can be seen as a black-box that takes a "quantum circuit" as input and returns a "profiler report". This black-box view should be enough for users that only want to use the qprof tool, but experienced users or plugins developers might need more details on the internals of qprof in order to understand how it works.

The following sections will introduce in details the three different parts that compose qprof. Section 4.3.2 describes qcw and the framework-agnostic quantum circuit representation it provides and that is used by qprof. A description of the core data structures and core logic is then provided in Section 4.3.3. Finally an explanation of the different exporters natively provided by qprof is given in Section 4.3.4.

4.3.2 The qcw package

The qprof tool aims at being the standard for profiling quantum circuits, independently of the framework they are written with. In order to be versatile and support as many current and future quantum computing frameworks as possible, qprof has been split into two packages: `qprof` that implements all the profiling logic and exporters and `qcw` that is responsible of adapting the different framework-specific quantum circuit formats into a standardised representation. This section presents the qcw package.

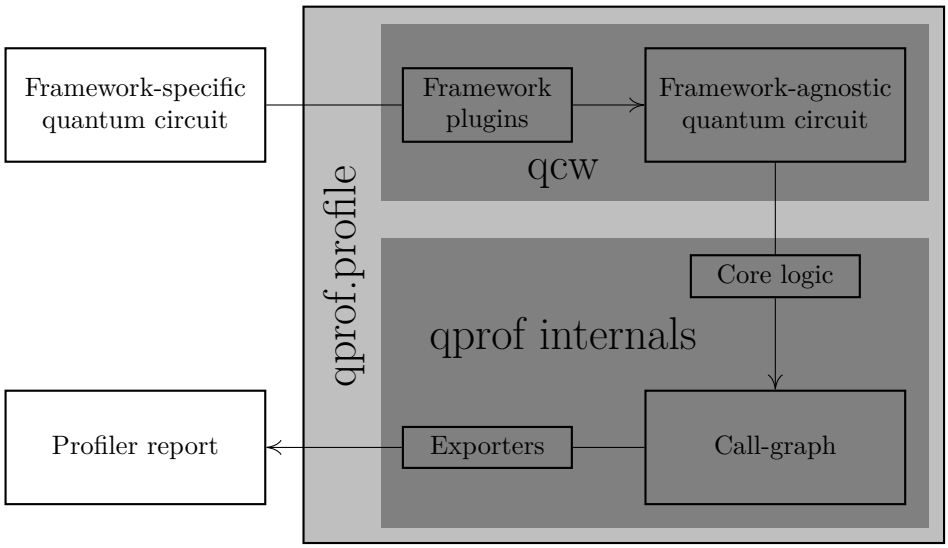


Figure 4.2: *Schematic representation of qprof workflow. Internally, qprof uses qcw to recover a framework-agnostic representation of the quantum circuit. Then, this "universal" representation is used to profile the given quantum circuit. Finally the profiling results are exported using one of qprof exporters and returned to the user.*

The qcw package

qprof extensibility is achieved via a companion Python package called qcw and whose purpose is to abstract away all the specificities of the framework used to represent the quantum circuit and provide a unified interface for all the implemented frameworks. To fulfil its goals of being framework-agnostic and easily extensible, qcw provides a plugin mechanism that allows anyone to implement a wrapper for a specific framework and make it available through the qcw package. This high-extensibility is obtained thanks to the fact that plugins do not have to be part of the main qcw package to be recognised by qcw: they can be developed, used and published by anyone. This allows several situations that may help improving qcw (and consequently qprof) compatibility with quantum computing framework and extensibility. For example, users might decide to roll-out their own plugin to support a new framework they are using internally. Another important situation that is made possible by qcw and its architecture is that framework vendors have the opportunity to provide a qcw plugin along with their framework and to maintain it as an official plugin, effectively making qprof compatible with their framework without having to support a code base outside of their framework.

Finally, such an architecture based on an external package that accept plugins allows the user to only install the plugins and frameworks needed instead of installing all of them along with qprof. This simple side improvement greatly reduces the installation time, installation size and plugin discovery time as it avoids installing and loading unused quantum computing frameworks.

Framework support

The goal of qcw is to provide a unique interface to access information about quantum programs that can be written using a variety of different frameworks. Taking into account that several of the most successful quantum computing frameworks such as Qiskit, Cirq, PyQuil or myQLM are Python libraries, and in order to ease its integration with these already existing frameworks, qcw has naturally been designed as a Python library too. It is important to note that this does not impede the capacity of qcw to support non-Python frameworks such as XACC, QCOR, Q# or Quipper.

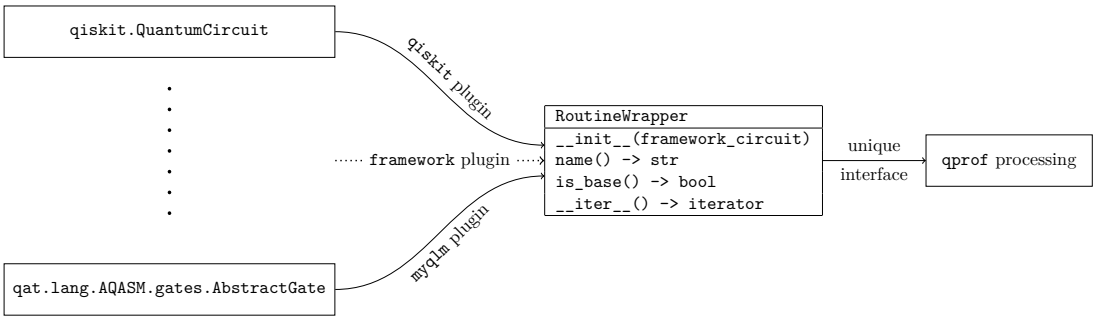


Figure 4.3: *Schematic overview of the framework architecture used in qcw. Each framework-specific representation is wrapped by a* `RoutineWrapper`*. Each supported framework should have a corresponding qcw plugin that implements the* `RoutineWrapper` *interface. The* `__init__` *method initialises a* `RoutineWrapper` *instance with an instance of the framework-specific quantum circuit representation. The* `name` *method returns the name of the currently wrapped routine.* `is_base` *returns* `True` *if the routine is a native routine as defined in Definition 14, else it returns* `False`*. Finally, the* `__iter__` *method returns an object that can be iterated on and whose iterates are the different subroutines called by the current routine.*

In order to be as generic as possible, qcw uses an abstract common interface to represent the concept of "quantum (sub)routine". This concept is formally defined in Definitions 12 to 14.

Definition 12. Quantum routine: a possibly parameterised, named, sequence of quantum subroutines.

Definition 13. Quantum subroutine: a quantum routine that is part of a higher-level quantum routine (i.e. that is called by another quantum routine).

Definition 14. Native quantum subroutine: a quantum routine that represents a native hardware operation and that does not call any quantum subroutine.

Using Definitions 12 to 14, a common interface for the concept of "quantum routine" emerges. First, a quantum routine should have a name that can be retrieved. Secondly, we should be able to distinguish between native quantum routines and non-native ones. Finally, for each non-native quantum routines, we need a way to iterate over all the subroutines composing it.

This interface, schematised in Figure 4.3, is the core abstraction layer of qcw that allows it to be as independent as possible from the underlying quantum computing framework used to represent the profiled quantum circuit and to provide a unified interface across a wide range of different frameworks.

Currently, qcw has been used to successfully access quantum circuits built with the Qiskit and myQLM frameworks. OpenQASM 2.0 support is also implemented using Qiskit translation capabilities by building a `qiskit.QuantumCircuit` instance from the given OpenQASM 2.0 code and using the Qiskit wrapper of qcw. Using the same idea, an experimental XACC wrapper has been implemented by exporting XACC code to OpenQASM 2.0. Finally, Q# and Quipper support is currently being envisioned and should be implementable as both framework implement either Python bindings or a method to export to OpenQASM 2.0 code.

4.3.3 Core data structures and logic

Now that the issue of adapting qprof to the various quantum computing frameworks has been solved, we can start considering the main problem of profiling a quantum circuit.

Section 4.3.3 introduces the different quantities that might be interesting to include in a quantum program profiling report, comparing with classical computing quantities when appropriate. Then, Section 4.3.3 explains the main graphical representation used through this

chapter and in qprof: the call-graph representation. Finally, Section 4.3.3 introduce respectively the data-structures and the algorithms used internally by qprof to profile a quantum circuit implementation.

Interesting data to profile

Profiling a program is the action of gathering data on its execution. For classical programs and profilers, the list of data that can be gathered is quite extensive ranging from high-level quantities such as the time spent in a given function or the memory used during the program execution to low-level information recovered via hardware counters such as cache misses or branch-prediction-misses.

But for quantum computing, the quantities of interest need to be adapted as several classical data such as cache-miss or branch-prediction-miss do not have any meaning anymore. Nevertheless some classical quantities have a quantum analogue that may be useful for optimisation purposes.

This is the case for the classical "instruction number" quantity, that translates trivially to its quantum counterpart "native gate number" (or "hardware gate number"). The number of native gates executed by a quantum routine is a useful information for several reasons: it is simple, the routine worst-case execution time can be computed from it and a lower-bound of the routine error rate can also be devised using this information.

Another classical quantity that can be translated to quantum computing is the "time spent in routine". This quantity can be subdivided in two more specific figures: the "time spent exclusively in routine" (sometimes called "self time") and the "time spent in subroutines called by the routine". This separation is often done in classical profiling programs as having these two execution times gives very useful information about the profiled routine that cannot be obtained from the "time spent in routine" only.

The last classical quantity with a meaningful quantum counterpart is the "memory usage", which may be translated as "number of qubits needed" when using quantum computers.

About quantities without a clear classical parallel but potentially useful, one can cite the "routine depth" as an approximation of the total execution time of the routine, the "T-count" for error-correction estimates, the "idle time" to estimate the potential effects of qubit decoherence on the routine, the needed "chip topology" in order to execute the routine, the "quantum gate parallelism" the implementation is able to reach, etc.

Graph representation (call-graph)

Following Definitions 12 to 14 and the `RoutineWrapper` interface we defined in Section 4.3.2, a graph-like representation of a quantum program seems to be particularly well suited. In this representation, nodes are quantum routines and an oriented edge from node `A` to node `B` means that the quantum routine represented by `A` calls the quantum subroutine represented by `B`. This representation of a program is called a call-graph in classical computing.

Figure 4.4 shows a call-graph representation of one possible implementation of Grover's algorithm. Even though this representation is valid according to the general definition of a call-graph, it contains a lot of redundant information that scrambles the useful data in visual noise. Because of this, most of the call graph representations avoid the duplication of nodes, i.e. create one node for a specific routine and re-use this node whenever the routine is called.

Figure 4.5 shows another possible call-graph representation of the same implementation of Grover's algorithm. Here, a graph node represents a unique routine and is re-used whenever this routine is called.

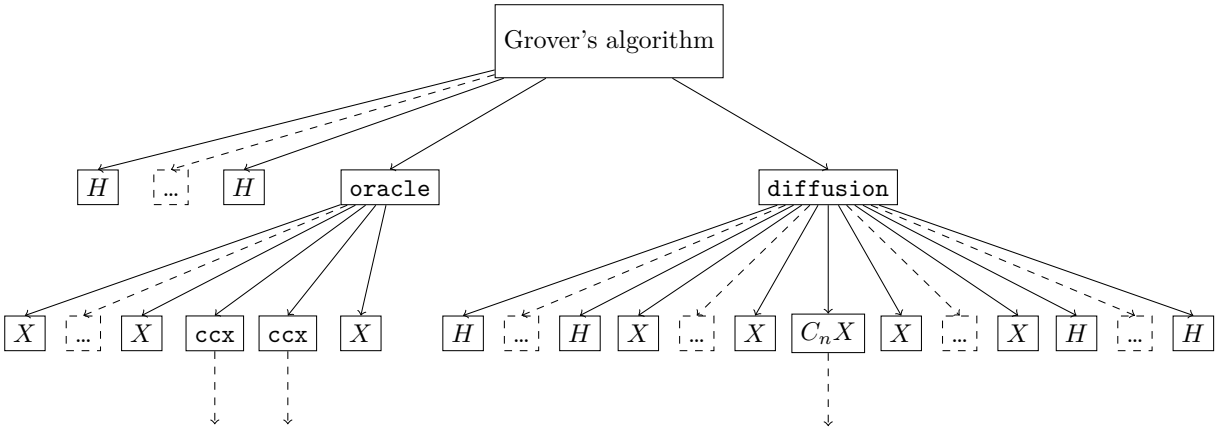


Figure 4.4: *Call-graph representation of one possible implementation of Grover's algorithm. Dashed squares with dots within them mean a repetition of the two gates around the dashed node. Dashed arrows starting from the two* `ccx` *nodes and the* C_nX *node represent a sub-graph that has not been included here for readability reasons.*

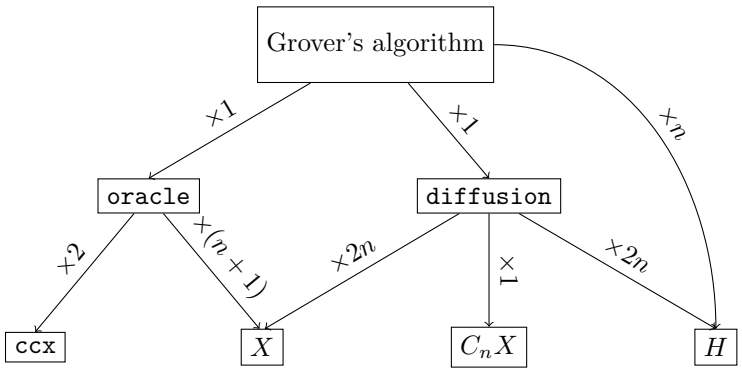


Figure 4.5: *Call-graph representation of one possible implementation of Grover's algorithm. Each node represents one routine rather than one call to the routine. Edges have been regrouped and labelled with the number of calls for readability purposes. In the internal representation used by qprof, edges are not regrouped and are ordered to account for the original quantum program subroutine call order.*

RoutineNode	
subroutines:	List[RoutineNode]
cost:	double
T**-count**:	integer
topology:	Topology

Figure 4.6: *Example of a possible* `RoutineNode` *containing information on the cost, the T-count and the required topology of the routine it represents. Text on the left of each line represents the name of the stored information and is followed by the type that stores this information.* `RoutineNode` *instances always store a (possibly empty) list of subroutines that are called by the represented routine. This list encodes the edges of the call-graph, i.e. if the* `RoutineNode` *A has B in its* `subroutines`*, an edge from A to B will be present in the call-graph.*

```
RoutineWrapper
__init__(framework_circuit)
name() -> str
is_base() -> bool
__iter__() -> iterator
__eq__(other_routine) -> bool
__hash__() -> int
```

Figure 4.7: *Final* `RoutineWrapper` *interface.* `__init__`*,* `name`*,* `is_base` *and* `__iter__` *methods are described in Figure 4.3. The* `__eq__` *method tests if* `other_routine` *is equal to the current instance.* `__hash__` *computes an integer hash of the currently wrapped routine.*

Data structures

To profile a given quantum circuit (or equivalently a given call-graph), qprof will naturally have to explore it and gather data through the exploration. The exploration is performed using a data structure inspired from graph exploration: `RoutineNode`.

A `RoutineNode` represents one node of the call graph (i.e. one routine of the quantum circuit) and stores information about the represented node. An example of a possible `RoutineNode` is given in Figure 4.6.

In order to be as efficient as possible on a wide class of quantum circuits, qprof does its best to reduce the number of call-graph nodes it has to explore. To do so, qprof caches instances of `RoutineNode`: the first time a routine is seen, its corresponding `RoutineNode` will be created and saved in order to be re-used without having to re-create the `RoutineNode` instance each time the routine is encountered.

This cache mechanism is implemented using a factory pattern: a `RoutineNode` should only be created indirectly through a dedicated `RoutineNodeFactory` instance. The `RoutineNodeFactory` instance keeps track of all the `RoutineNode` it has already created and implements the cache using Python `dict` data structure, internally implemented as a hash table. The cache implemented by `RoutineNodeFactory` has no maximum size, meaning that it will keep each `RoutineNode` instance created. This absence of cache invalidation is not an issue as every cached routine is already present in the profiled quantum circuit, meaning that qprof memory usage is at worst equivalent to the profiled quantum circuit memory usage.

Due to the requirements of the hash table data structure, `RoutineNode` instances should be hashable and comparable with other `RoutineNode` instances. These requirements are offloaded by qprof to the qcw `RoutineWrapper` data structure to leave the possibility to use hash and equality operators provided by the wrapped framework. The final interface of the `RoutineWrapper` data structure is shown in Figure 4.7.

Note 11. The implementation of the hash and equality operators should be performed with care as their characteristics are crucial for qprof runtime and accuracy. The main requirements are imposed by the hash table data structure used by qprof: hash and equality operators should

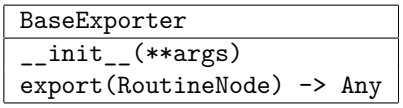


Figure 4.8: *Exporter interface. Any plugin that implements an exporter should be a derived from the* `BaseExporter` *class and implement this interface. The return type of the* `export` *method is not specified and can be anything. The main* `profile` *function will return the output of the exporter.*

have a complexity in $\mathcal{O}(1)$, and the hash operator should have the best *quality* possible (i.e. the lowest collision rate possible).

Implementing correct hash and equality operations with a complexity of $\mathcal{O}(1)$ may be non-trivial, as the constant complexity requirements prevents the operators from exploring each of the gates contained in the tested routine. qcw implements the hash and equality operators using the name and the parameters of the routine at hand, with the assumption that two routines with the same name and the same parameters will contain exactly the same gates (and consequently, are equal). This assumption might be invalidated in the case of randomised routines.

qprof algorithms

The main procedure and only function accessible from qprof interface, **`qprof.profile`**, is described in Algorithm 1.

Algorithm 1: **`qprof.profile`**, the main qprof function

Input: **main_routine** a quantum circuit, **gate_costs** a dictionary-like data-structure storing the cost for each native quantum gate, **exporter** the qprof exporter to use, **framework_arguments** arguments forwarded to the quantum computing framework used to represent **main_routine**

Output: **exporter_report** the report returned by the given exporter

```
1 factory ← new RoutineNodeFactory() ;
2 qcw_routine ← qcw.Routine(main_routine, framework_arguments) ;
3 tree_root ← factory.get(qcw_routine, gate_costs) ;
4 return exporter.export(tree_root) ;
```

This procedure calls the method `RoutineNodeFactory.get` that is detailed in Algorithm 2. A study of the runtime complexity of Algorithm 2 is provided in Section 4.4.1.

Algorithms used by the different exporters to summarise the call-graph built with `RoutineNode` instances may use internal data structures and other algorithms in order to generate a report. These are specific to the exporter and Section 4.3.4 gives an example with some details and a description of the data structure used by the `gprof`-compatible exporter along with the limitation it imposes to the quantum circuits that can be handled by the exporter.

4.3.4 Exporters

qprof also implement several exporters that will transform the abstract quantum program representation described in Section 4.3.3 to a more usable format.

Exporters should implement a specific interface schematised in Figure 4.8. qprof natively implements two textual exporters: one that outputs a `gprof`-compatible format and another that returns a JSON-formatted string that directly represents a flat call-tree structure used internally by the `qprof` exporter.

Algorithm 2: `factory.get`, qprof processing applied for each node of the call-graph. Gate cost is the only information computed by qprof for the moment. Making qprof compatible with other information such as topology or program parallelism will require to update this algorithm.

Input: **routine** a quantum circuit wrapped by qcw, **factory** a qcw routine factory, **gate_costs** a dictionary-like data-structure storing the cost for each native quantum gate

Output: **routine_node** an instance of `RoutineNode`, the internal qprof data structure to represent one node of the call-graph

```
   /* 1.  Try to find the routine in the cache and return it if found     */
 1 if routine in factory.cache then
 2 |   return factory.cache[routine] ;
 3 end
   /* 2.  The node has never been encountered yet, build it and add it to
      the cache                                                           */
 4 routine_node ← RoutineNode() ;
 5 routine_node.self_cost ← 0 ;
 6 routine_node.subroutine_costs ← 0 ;
 7 routine_node.subroutines ← list () ;
 8 routine_node.routine_name ← routine.name() ;
   /* 3.  Test if the explored node is a leaf (native gate)               */
 9 if routine_node.routine_name in gate_costs or routine.is_base() then
10 |   routine_node.self_cost ← gate_costs[routine_node.routine_name] ;
11 |   return routine_node ;
12 end
   /* 4.  Else, if the explored node is not a leaf, recurse into its
      children                                                            */
13 foreach subroutine in routine.__iter__() do
14 |   child_node ← factory.get(subroutine, gate_costs) ;
15 |   routine_node.subroutines.append(child_node) ;
   |   /* Important note:  the following line assumes that the "cost" is an
   |      additive quantity.  It will have to be updated for non-additive
   |      quantities such as error rates or topology.                    */
16 |   routine_node.subroutine_costs ← routine_node.subroutine_costs +
   |    child_node.self_cost + child_node.subroutine_costs ;
17 end
   /* 5.  Update the cache with the computed routine before returning     */
18 factory.cache[routine] ← routine_node ;
19 return routine_node ;
```

Flat call-tree representation

Before the profiler report generation, it is convenient to summarise the information contained in the generic call-graph structure presented in Section 4.3.3. To do so, the `gprof` and JSON exporters both rely on a flat structure that represents a directed call-tree (i.e. a directed call-graph without loops).

This structure puts an additional restriction to the quantum programs that can be profiled using these exporters: the interdiction to have recursive subroutines (a subroutine that ends up calling itself). It is important to realise that this restriction does not have a huge impact on the area of application of qprof because, as of today, recursive subroutine calls do not seem to be widespread in quantum computing programs and the restriction only applies to the `gprof` and JSON exporters, the core logic of qprof being capable of handling recursive subroutine calls without any issue.

The flat call-tree structure stores, for each subroutine `A` encountered in the call-graph exploration, a list of all the subroutines `B` called by `A`. Along with each called subroutine `B`, the structure stores the number of times `B` has been called by `A` and the cost associated with these calls. Finally, in order to simplify the report generation, each called routine `B` will also store a list of the routines `A` it has been called by. Within this list is also stored the number of calls to `B` that have been performed from each `A` and the cost associated with these calls.

`gprof` output

The `gprof` exporter aims at generating a profiler report that is compatible with the profiler report returned by `gprof`, a well-known classical profiler. Being compatible with a tool that has been around for decades and is still actively used has several advantages.

First and foremost, the fact that a tool that has been stable for decades and is still actively used shows that it provides satisfaction to its users, meaning that the output format includes enough information and is sufficiently easy to read and use in practice.

Secondly, a decades-old, largely used, output format is likely to have a lot of official or user-contributed tools to help analysing and representing it in the best way possible. This is the case for the `gprof` format that can be translated to a call-graph using the `gprof2dot` tool and the `dot` executable from Graphviz library.

Finally, the `gprof` output is simple to generate: it is a textual file with a simple and regular format.

Reading a `gprof`-based call-graph

As explained in Section 4.3.4, `gprof` output (and qprof output when used with the `gprof`-compatible exporter) can be visualised as a call-graph using the `gprof2dot` tool and the `dot` executable from the Graphviz library. Such a call-graph is depicted in Figure 4.9.

Each node of the graph represents a unique quantum routine. The name of this quantum routine can be read on the first line of text inside the node. The second line of text in the node is the percentage of the total cost associated with the routine represented by the node, including the cost of subroutines called by the routine (also called `total_time` when the considered cost is the execution time). The third line represents the total cost of the routine represented by the node, but excluding subroutines (also called `self_time` when the considered cost is the execution time). The fourth line represents the number of times the routine is called in the program. Finally, each node is coloured according to the total time spent in the routine it represents from dark-red for high-cost routines to light-green for low-cost routines.

Each directed edge of the graph represents a subroutine call: if a directed edge that goes from node `parent` to node `child` is present, it means that the routine represented by node `parent` is calling the subroutine represented by node `child` at least once. Each edge is annotated with the

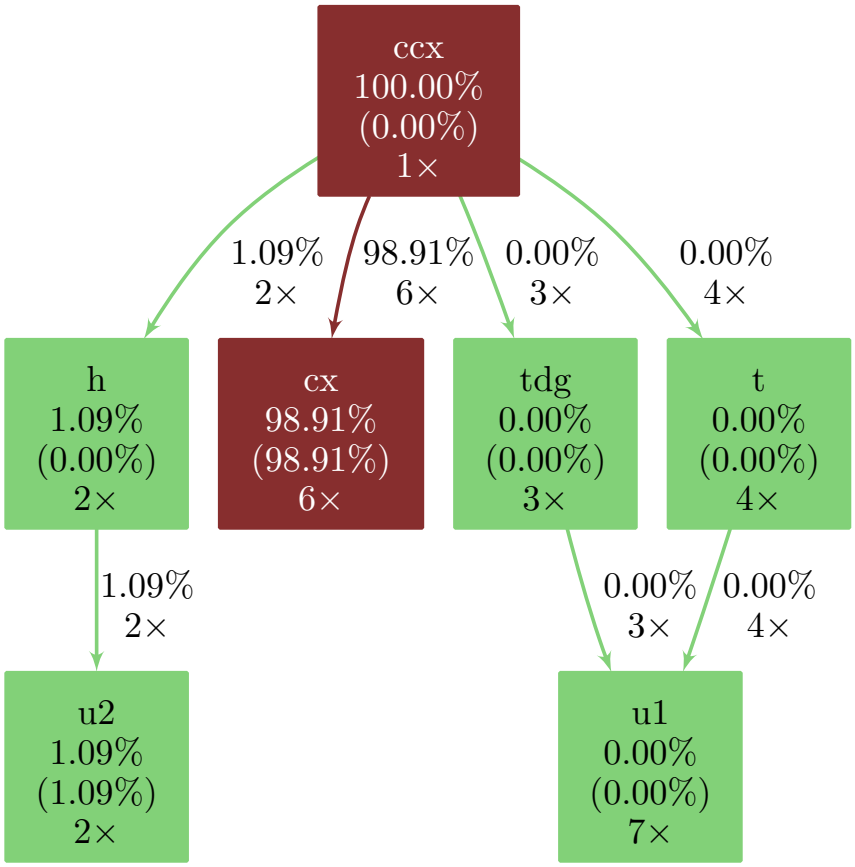


Figure 4.9: *Example of a simple call-graph generated with the help of* `gprof2dot` *and the* `dot` *tool from the Graphviz library.*

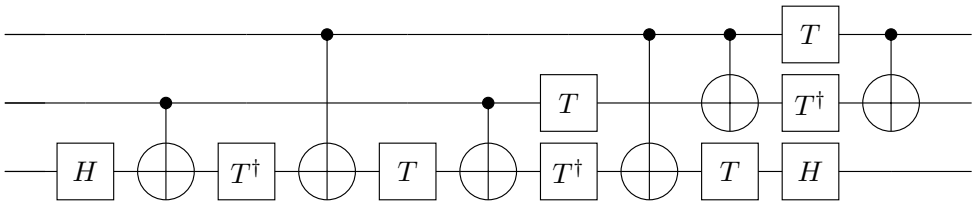


Figure 4.10: *Standard representation of a quantum circuit. Time goes from left to right, giving the order in which quantum gates are applied. Horizontal lines represent qubits. Gates are represented by rectangles, with the notable exception of the* `CNOT` *gates that has a control qubit (small black dot) and a target qubit (circle with a cross in the inside) linked together by a straight line.*

percentage of the total cost transferred from `parent` to `child` (i.e. the cost that was consumed calling `child` from within `parent`) and the number of times `parent` is calling `child`.

With these definitions, and provided that the cost used is an additive quantity, the main routine will always have an execution time of 100% and the sum of the percentages of each outgoing edge of a given node should be equal to the `total_time` of this node.

Advantages of the call-graph visualisation

There are several advantages to the call-graph representation used in Figure 4.9 when compared to the other possible representations of a quantum circuit.

One of the most widespread way of representing a quantum circuit in the quantum computing community is depicted in Figure 4.10. This representation has the advantage of being simple to understand and precise with respect to which quantum operation should be applied and when. One of the disadvantages of this representation is that it becomes quickly unreadable for quantum circuits containing a lot of quantum gates. It is also a shallow representation: the only way of representing a main quantum circuit C that calls the subroutine R without inlining the call to R in C is by representing C and R separately. This becomes quickly unmanageable for complex quantum circuits that may call tens of nested subroutines.

The call-graph representation has the advantage of complementing the standard quantum circuit representation of Figure 4.10: its main strength is its ability to represent very large and deeply nested quantum circuits in one synthetic and concise graph, providing a readable and global representation of the whole quantum circuit. In the call-graph representation, all the routines of the profiled quantum circuit are represented and the relationship between each routine (which one calls and which one is called) is explicit.

4.4 Complexity and runtime analysis

4.4.1 Asymptotic complexity of qprof

The runtime efficiency of qprof is one of its strength: it will be very efficient on most of real-world quantum circuit implementation.

Let first recall that qprof only access a quantum circuit through the interface provided by qcw and summarised in Figure 4.7. This means that computing the asymptotic complexity of profiling a given quantum circuit depends on the complexity of the qcw methods and on the number of call to such methods qprof needs to perform.

Algorithm 2 details the algorithm used by qprof to initialise its internal data structures. This algorithm is only applied once, on the routine to profile (the call-graph root, i.e. the only node that does not have any incoming edge), and then recurses into the call graph to explore all the nodes needed.

qcw interface is implicitly or explicitly called on six lines of Algorithm 2. First on lines 1 and 2, the hash and equality operators are called in order to perform hash table operations. Then, on line 8, the name of the currently explored routine is retrieved once. A test to check if the routine is considered as "native" is performed with a call to the `is_base` method at line 9. The for-loop on line 13 is also calling the `__iter__` method once. Finally, line 18 is calling the hash and equality operators again to add an entry in the cache implemented as a hash table.

Note 12. The `__iter__` method is only called once but will iterate over all the subroutines of the current routine even those that have already been seen and cached by qprof. The already cached subroutines will simply end the recursion for this branch of the call-graph in the call to `factory.get` without exploring their subroutines.

A summary of the number of calls to the different methods provided by qcw interface is provided in Table 4.1.

Table 4.1: *Number of calls of qprof implementation to the qcw interface for each explored node of the call-graph. Note that a few optimisations that do not appear on Algorithm 2 for readability purpose have been performed in the implementation. This table provides the counts of the optimised implementation. c is a number that depends on the implementation of the hash table and the quality of the hash function used and that represents the expected average number of equality tests that should be performed at each access to the hash table. "Nodes" in the last column encompass both "Leafs" and "Non-leafs".*

`RoutineWrapper` method	Leafs, non-cached	Non-leafs, non-cached	Nodes, cached
`name () -> str`	1	1	0
`is_base () -> bool`	1	1	0
`__iter__() -> iterator`	0	1 (see Note 12)	0
`__eq__(other) -> bool`	c	2c	c
`__hash__() -> int`	1	2	1

Even though c in Table 4.1, the average number of calls to `__eq__`, seems hard to bound in general, Python documentation provides guarantees on the asymptotic complexity of the operations on a `dict` instance: access and modification of the data structure, which are the 2 operations performed by qprof are $\mathcal{O}(1)$ on average and $\mathcal{O}(n)$ on amortised worst case. This mean that for each explored nodes of the call-graph, qprof will only have to perform $\mathcal{O}(1)$ operations on average.

In the end, qprof asymptotic complexity depends entirely on the number of nodes of the call-graph it needs to explore. This number depends on the profiled circuit and no general formula that include the number of gates in the profiled quantum circuit can be devised.

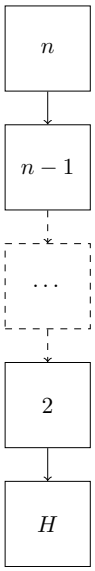
To illustrate this claim, two example quantum circuits are provided. Figure 4.11 provides an example of a quantum circuit that contains only 1 quantum gate but that will require qprof to visit an arbitrarily large number N of nodes in the call-graph. On the other side, Figure 4.12 shows a quantum circuit that contains $N = 2^n$ quantum gates but that will only require qprof to explore $\mathcal{O}(\log_2 N)$ nodes of the call-graph.

We can still have an upper-bound of the number of operations qprof will have to perform on a given quantum circuit by restricting each routine to call at most $N_{\text{subroutine}}$ subroutines and by using the number of unique quantum gates N_u used in the circuit. For example, the quantum circuit depicted in Figure 4.12b has $N_u = 4$ because it contains 4 unique gates: $\{H, 2, 3, 4\}$ and the quantum circuit depicted in Figure 4.11b has $N_u = n$ unique quantum gates: $\{H, 2, \ldots, n-1, n\}$. For a quantum circuit in which routines are restricted to call at most $N_{\text{subroutine}}$ subroutines, qprof will explore at most $(N_{\text{subroutine}} \times N_u)$ nodes of the call-graph.

Algorithm 1: `get_linear_circuit`

Input: **n** an integer
Output: **circuit** a quantum circuit
if $n == 1$ **then**
  return H gate ;
end
circuit $\leftarrow$ `empty_circuit(name` $= n$`)` ;
circuit.`append(get_linear_circuit(`$n-1$`))` ;
return circuit ;

(a) *Pseudo-code of the algorithm to generate the* ***linear*** *quantum circuit.*



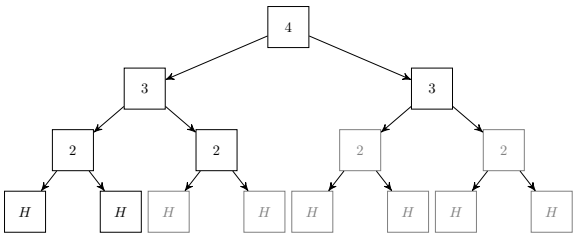
(b) *Call-graph representation of the* ***linear*** *quantum circuit.*

Figure 4.11: *Example of quantum circuit that contains a constant number of quantum gates (here only 1) but that will require qprof* $\mathcal{O}(n)$ *operations to analyse and output a profile report.*

Algorithm 1: `get_binary_tree_circuit`

Input: **n** an integer
Output: **circuit** a quantum circuit
if $n == 1$ **then**
  return H gate ;
end
circuit $\leftarrow$ `empty_circuit(name` $= n$`)` ;
circuit.`append(get_binary_tree_circuit(`$n-1$`))` ;
circuit.`append(get_binary_tree_circuit(`$n-1$`))` ;
return circuit ;

(a) *Pseudo-code of the algorithm to generate the* ***binary_tree*** *quantum circuit.*



(b) *Call-graph representation of the* ***binary_tree*** *quantum circuit for* $n = 4$*. Nodes drawn in grey are not processed by qprof thanks to the caching mechanism described in Section 4.3.3.*

Figure 4.12: *Example of quantum circuit that contains an exponential number of quantum gates (here* 2^n*) but that will require qprof only* $\mathcal{O}(n)$ *operations to analyse and output a profile report.*

4.4.2 Real-world execution time

We benchmarked the execution time of qprof on several well-known use-cases. These benchmarks were performed on one core of a Intel Xeon Platinum 8260M cadenced at 2.40GHz. Table 4.2, Table 4.3 and Table 4.4 give the average and standard deviation of the profiling time for quantum circuits implementing three different use cases. The "Profiling time" and "Saved time" measurements have been performed 100 times and each table contains the average time and the standard deviation observed over the 100 executions.

"Saved time" is an estimation of the execution time saved thanks to the caching mechanism implemented. It is computed by saving the time needed to profile a routine when it is first encountered and then incrementing a counter by this exact same time each time the routine is seen again and the cache is used. This methodology tends to produce noisy results because an imprecision in the first measurement will lead to an accumulation of errors, but the computed standard deviations are always relatively low compared to the average which is a good indicator that the obtained "Saved time" is close to the real saved time.

Table 4.2: *qprof observed runtime on quantum circuits generated using the quantum circuit described in [75] and also used in Listing 4.3. The* `evolve_1d_dirichlet` *function was used with an evolution time of* 0.1*, a desired precision* $\epsilon = 10^{-3}$*, a trotter order of* 1 *and a varying number of discretisation points given in the N column. The nearly instantaneous generation times have to do with how the myQLM framework is working: the circuit is generated lazily when needed. Consequently, the Profiling time and Saved time column also include the time needed to construct the quantum circuits. Profiling time and Saved time columns provide* `average` ± `standard_deviation` *numbers obtained by profiling* 100 *times the generated circuit.*

N	# Qubit	Gate number	Generation (s)	Profiling time (s)	Saved time (s)
2^3	4	126846	0.000	0.01 ± 0.00	0.82 ± 0.01
2^4	5	528768	0.000	0.02 ± 0.00	2.99 ± 0.03
2^5	6	1953720	0.000	0.04 ± 0.01	10.17 ± 0.14
2^6	7	6773868	0.000	0.09 ± 0.02	33.26 ± 0.31
2^7	8	22575672	0.000	0.24 ± 0.03	106.92 ± 1.52
2^8	9	73323792	0.000	0.66 ± 0.03	333.43 ± 4.26
2^9	10	233816544	0.000	1.90 ± 0.04	1043.10 ± 14.02
2^{10}	11	735473520	0.000	5.48 ± 0.07	3215.83 ± 44.62
2^{11}	12	2289028896	0.000	15.73 ± 0.15	9914.92 ± 132.58
2^{12}	13	7063525944	0.000	45.77 ± 0.42	30473 ± 275
2^{13}	14	21643231428	0.000	132.21 ± 1.15	92083 ± 1133
2^{14}	15	65922050880	0.000	383.65 ± 6.79	270824 ± 5494

4.5 Code examples and practical applications

This section includes several examples of qprof usage on various quantum circuits ranging from a simple Toffoli gate decomposition in Section 4.5.1 to more complex algorithm implementations such as Grover's algorithm in Section 4.5.2. All these benchmarks are performed on circuits generated using the `qiskit` framework. An example of benchmarking a quantum implementation of a 1-dimensional wave equation solver written using the myQLM framework is finally provided in Section 4.5.3.

Table 4.3: *qprof observed runtime on quantum circuits generated using the function* `qiskit.algorithms.HHL`*. The linear system matrices were constructed with the function* `qiskit.algorithms.linear_solvers.matrices.TridiagonalToeplitz`*(N, 1, 0.5) and the right-hand side b has been picked randomly. Profiling time and Saved time columns provide* `average` ± `standard_deviation` *numbers obtained by profiling* 100 *times the generated circuit.*

N	# Qubit	Gate number	Generation (s)	Profiling time (s)	Saved time (s)
2^1	5	1049	0.087	0.02 ± 0.01	0.06 ± 0.05
2^2	9	8759	0.465	0.03 ± 0.00	0.19 ± 0.00
2^3	13	34866	1.523	0.09 ± 0.02	0.45 ± 0.03
2^4	16	192104	7.572	0.18 ± 0.00	1.56 ± 0.01
2^5	20	581170	21.881	0.63 ± 0.09	4.14 ± 0.33
2^6	24	1744225	63.612	2.09 ± 0.25	11.63 ± 0.79
2^7	28	4937772	175.546	7.23 ± 0.89	31.41 ± 1.09
2^8	32	12310383	441.949	25.67 ± 0.73	91.72 ± 3.58
2^9	36	33471747	1234.263	98.59 ± 0.12	289.60 ± 3.76

Table 4.4: *qprof observed runtime on quantum circuits generated using the function* `qiskit.algorithms.Shor` *trying to factor the number N. Profiling time and Saved time columns provide* `average` ± `standard_deviation` *numbers obtained by profiling* 100 *times the generated circuit.*

N	# Qubit	Gate number	Generation (s)	Profiling time (s)	Saved time (s)
15	18	35049	2.644	0.07 ± 0.00	0.44 ± 0.00
77	30	216651	11.840	0.21 ± 0.03	2.15 ± 0.45
221	34	340817	17.460	0.26 ± 0.00	2.96 ± 0.02
437	38	511039	24.161	0.30 ± 0.00	3.84 ± 0.04
899	42	737301	34.197	0.36 ± 0.00	4.89 ± 0.06
2021	46	1030547	42.204	0.44 ± 0.00	6.69 ± 0.07
4087	50	1402681	58.869	0.51 ± 0.01	8.50 ± 0.10
6557	54	1866567	76.888	0.59 ± 0.16	10.03 ± 1.31
14351	58	2436029	98.285	0.62 ± 0.01	11.29 ± 0.14
30967	62	3125851	109.153	0.73 ± 0.01	14.47 ± 0.12
38021	66	3951777	142.007	0.81 ± 0.01	16.85 ± 0.20

4.5.1 Benchmarking a simple program

One of the most simple quantum program that can be benchmarked is the implementation of a Toffoli gate. Such a benchmark has the benefit of being simple enough to be studied by hand which means that we will be able to verify qprof results by hand-computing them.

The decomposition of a Toffoli gate as implemented in the `qiskit` framework is depicted in Figure 4.13. A complete example using qprof to profile the default Toffoli gate decomposition in `qiskit` is shown in Listing 4.1.

The output of qprof, which is here in a `gprof`-compatible format, can then be analysed. For the sake of readability and brevity, the full `gprof`-compatible profiler report will not be included verbatim in this chapter and will rather be visualised using the `gprof2dot` tool that allows representing `gprof` reports as call-graphs. The call-graph obtained from the report generated in Listing 4.1 is depicted in Figure 4.14.

From the call-graph depicted in Figure 4.14, it is clear that the cost of a Toffoli gate comes from its 6 controlled-X gates, that account for more than 98% of the total execution time. It is also interesting to note that the T gate, known to be very costly when error-correction is needed, is "free" on IBM Quantum chips when error-correction is not needed as it is equivalent

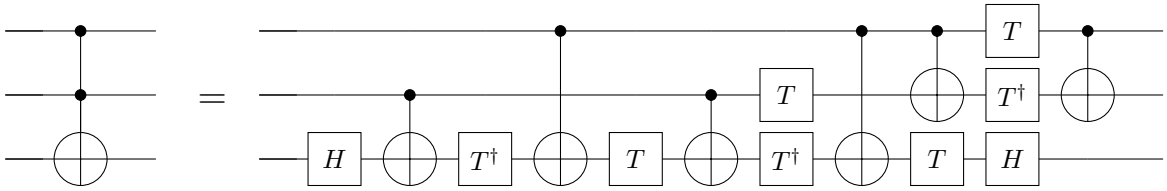


Figure 4.13: *Standard decomposition of a Toffoli gate into* 1− *and* 2− *qubit gates.*

Code Listing 4.1: *Python code needed to use qprof on the Toffoli gate implementation and save the profiler report in a* `gprof`*-compatible format in a file named* `toffoli.qprof`*.*

```
from qiskit import QuantumCircuit
from qprof import profile

# Circuit construction
circuit = QuantumCircuit(3, name="one_ccx_circuit")
circuit.ccx(0, 1, 2)
# Profiling
gate_costs = {"u1": 0, "u2": 10, "u3": 30, "u": 30, "cx": 300}
gprof_output = profile(
    circuit, gate_costs, "gprof", include_native_gates=True
)
with open("toffoli.qprof", "w") as f:
    f.write(gprof_output)
```

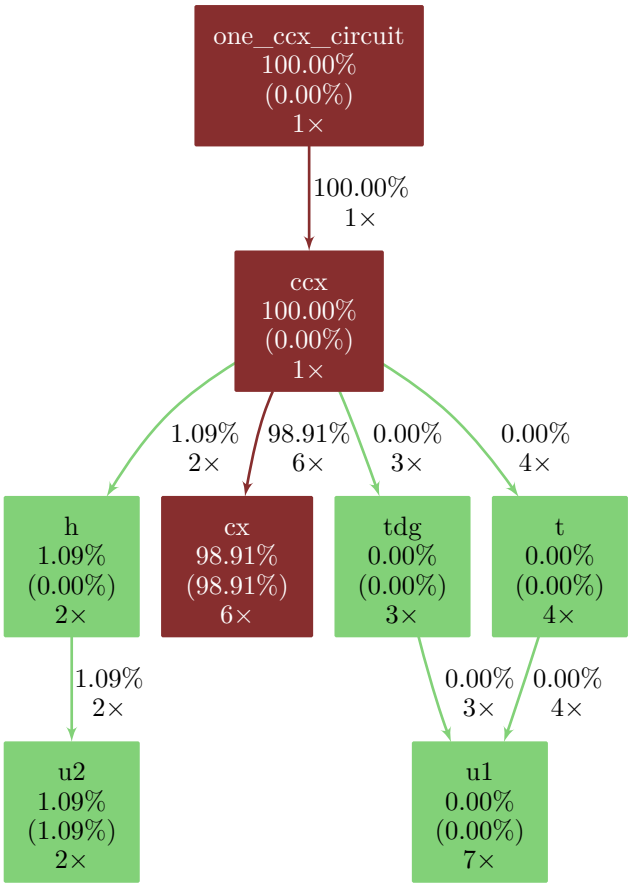


Figure 4.14: *Call-graph for the Toffoli gate implementation. Quantum gates included in the* `gate_costs` *variable (here* `u`*,* `u1`*,* `u2`*,* `u3` *and* `cx`*) are considered as native gates. Only native gates have a non-zero self-time as they are the only gates that are really executed on the hardware.*

Code Listing 4.2: *Python code needed to use qprof on the Grover implementation and save the profiler report in a `gprof`-compatible format in a file named `grover.qprof`.*

```
from qiskit.algorithms import AmplificationProblem, Grover
from qiskit.circuit.library import PhaseOracle
from qprof import profile
# Circuit construction
oracle = PhaseOracle("(v0 | ~v1) & (~v2 & v3)")
problem = AmplificationProblem(oracle, is_good_state=oracle.evaluate_bitstring)
grover = Grover(iterations=1)
circuit = grover.construct_circuit(problem)
# Profile
gate_costs = {"u1": 0, "u2": 10, "u3": 30, "u": 30, "cx": 300}
gprof_output = profile(circuit, gate_costs, "gprof", include_native_gates=True)
with open("grover.qprof", "w") as f:
    f.write(gprof_output)
```

to a phase change.

4.5.2 Grover's algorithm

The Toffoli gate is a good example to start and understand the meaning of qprof's output but the end goal of qprof is to be able to profile large and complex quantum circuits. A good first candidate to show how qprof performs on a more complex circuit is Grover's algorithm.

In this example we use Grover's algorithm on four qubits to find the three quantum states that verify the following formula:

$$(q_0 \vee \neg q_1) \wedge (\neg q_2 \wedge q_3). \tag{4.1}$$

The only three 4-qubit quantum states verifying Equation (4.1) are $|0001\rangle$, $|1001\rangle$ and $|1101\rangle$, q_0 being the left-most qubit in the bra-ket notation.

The code needed to generate the `gprof`-compatible output for Grover's algorithm with the oracle presented in Equation (4.1) is given in Listing 4.2. The resulting call-graph, included in Figure 4.15, clearly shows that the controlled-X gate is still the major contributor to the total cost. But this time, contrarily to the Toffoli example shown in Section 4.5.1, the controlled-X gate is called by three different subroutines that all contribute significantly to the overall cost: `c3z`, `ccz` and `mcx`.

Thanks to qprof it is now easy to understand the subroutines that contribute the most to the total cost. More importantly, the `gprof`-compatible report and the call-graph representation give very insightful information about subroutines calls that are crucial for circuit optimisation. Such information can be used to weight the impact of a given optimisation and then decide whether or not it is worth applying it.

For example, knowing that the `ccz` subroutine takes 18.61% of the total time, it is easy to deduce that a 20% improvement in the implementation of `ccz` will translate into a tiny $\frac{18.61\%}{5} = 3.72\%$ improvement to the overall cost, which might not be worth the effort. On the other hand, optimising the `c3z` subroutine to reduce its cost by 20% improves the overall cost by 9.22%, which is nearly 10% and might be an interesting optimisation target. Finally, the call-graph visualisation conveys clearly the information that the `cx` gate is the most costly subroutine of the Grover's circuit, meaning that even a slight optimisation of the `cx` cost will have a high impact on the overall implementation cost.

4.5.3 Quantum wave equation solver

Finally, we include in this chapter a more complex example that has been implemented in a previous work [75] with `myQLM`, a quantum computing framework maintained by Atos. The

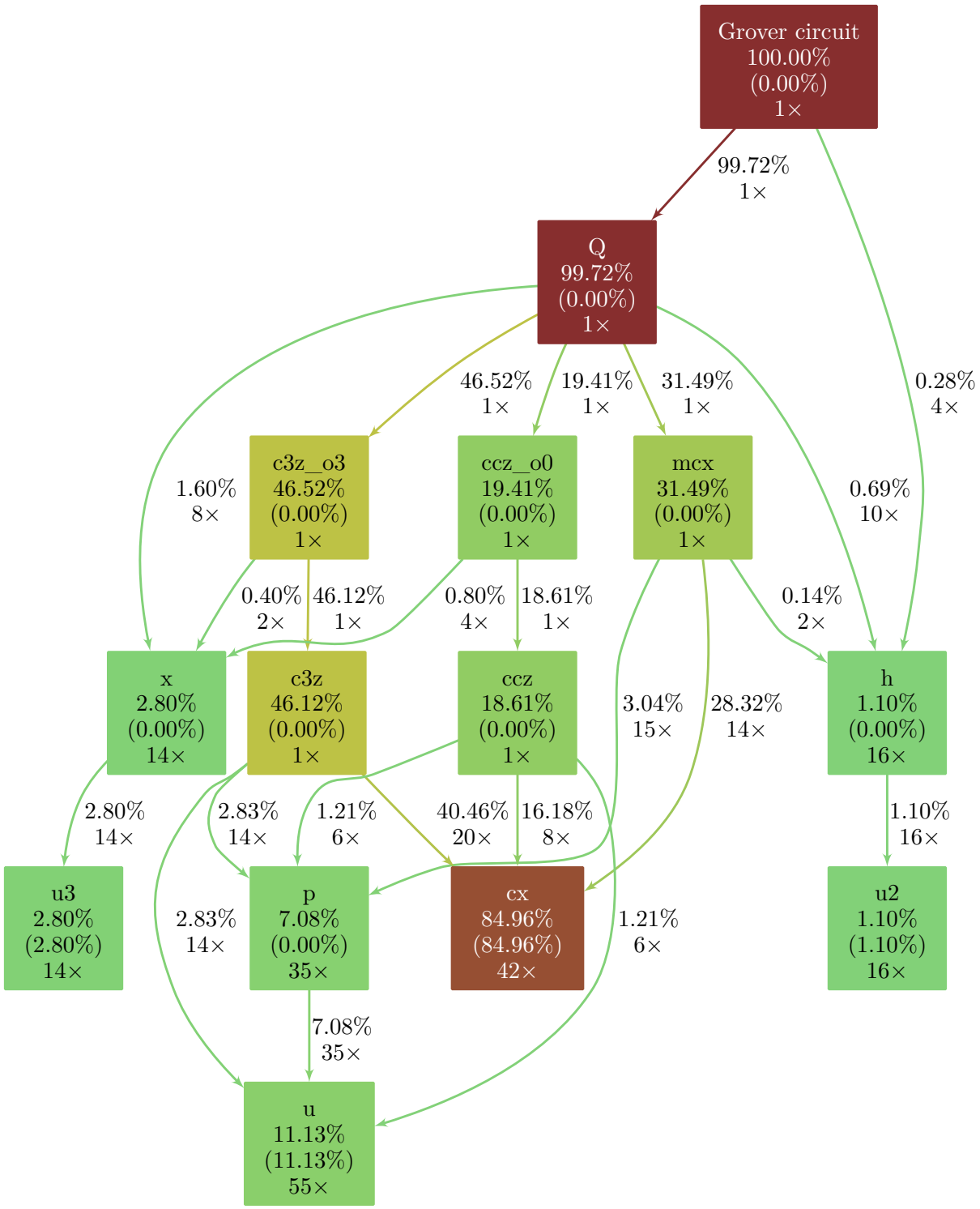


Figure 4.15: *Call-graph for the Grover's algorithm implementation. Quantum gates included in the* `gate_costs` *variable (here* `u`*,* `u1`*,* `u2`*,* `u3` *and* `cx`*) are considered as native gates. Only native gates have a non-zero self-time as they are the only gates that are really executed on the hardware. Some percentages might not add up to exactly* 100% *due to rounding errors.*

Code Listing 4.3: *Python code needed to use qprof with the QatHS library, on top of myQLM, and save the profiler report in a `gprof`-compatible format in a file named `qaths.qprof`.*

```
from qaths.applications.wave_equation.evolve_1D_dirichlet import
                                        evolve_1d_dirichlet
from qaths.applications.wave_equation.linking_sets.arithmetic_adder import
                                        get_linking_set as arith_linking_set
from qprof import profile
# Circuit generation
time = 0.1
discretisation_size = 2 ** 10
epsilon = 1e-3
trotter_order = 1
routine = evolve_1d_dirichlet(time, discretisation_size, epsilon, trotter_order)
# Gate execution time definition
G = {"u1": 0, "u2": 89, "u3": 178, "cx": 930}
gate_costs = {
    "cu1": 2 * G["cx"] + 2 * G["u1"],
    "cu2": 2 * G["cx"] + 2 * G["u3"],
    "X": G["u3"],
    "H": G["u2"],
    "CNOT": G["cx"],
    "CCNOT": 6 * G["cx"] + 2 * G["u2"] + 7 * G["u1"],
    "CH": 2 * G["cx"] + 2 * G["u3"],
    "PH": 3 * G["u1"] + 2 * G["cx"],
    "CPH": 3 * G["u1"] + 2 * G["cx"],
    "CCPH": None,
}
gate_costs["CCPH"] = 3 * gate_costs["CPH"] + 2 * gate_costs["CCNOT"]
gate_costs["CCCNOT"] = 6 * gate_costs["CCNOT"] + 2 * gate_costs["cu2"] + 7 *
                                        gate_costs["cu1"]
# Profiling
result = profile(
    routine,
    gate_costs,
    linking_set=arith_linking_set(discretisation_size),
    exporter="gprof",
)
with open("qaths.qprof", "w") as f:
    f.write(result)
```

code used to generate the benchmarked quantum program is available at https://gitlab.com/cerfacs/qaths/.

This example demonstrates that, as can be seen in Listing 4.3, qprof interface stays nearly the same even though the framework used is now completely different. The only exceptions are some additional parameters (such as `linking_set` in Listing 4.3) that are directly forwarded to the framework plugin used and additional gate definitions in the `gate_costs` data structure because of the way gate decomposition is handled in `myQLM`.

The call graph obtained by running Listing 4.3 is reproduced in Figure 4.16. In order for the call-graph to be readable on a paper format, negligible subroutines and calls (i.e. nodes and edges respectively) have been discarded from the graphical representation. The call-graph clearly shows that most of the execution time is spent in the oracle implementation. Moreover, multi-controlled-X gates are the major contributors to the total execution time.

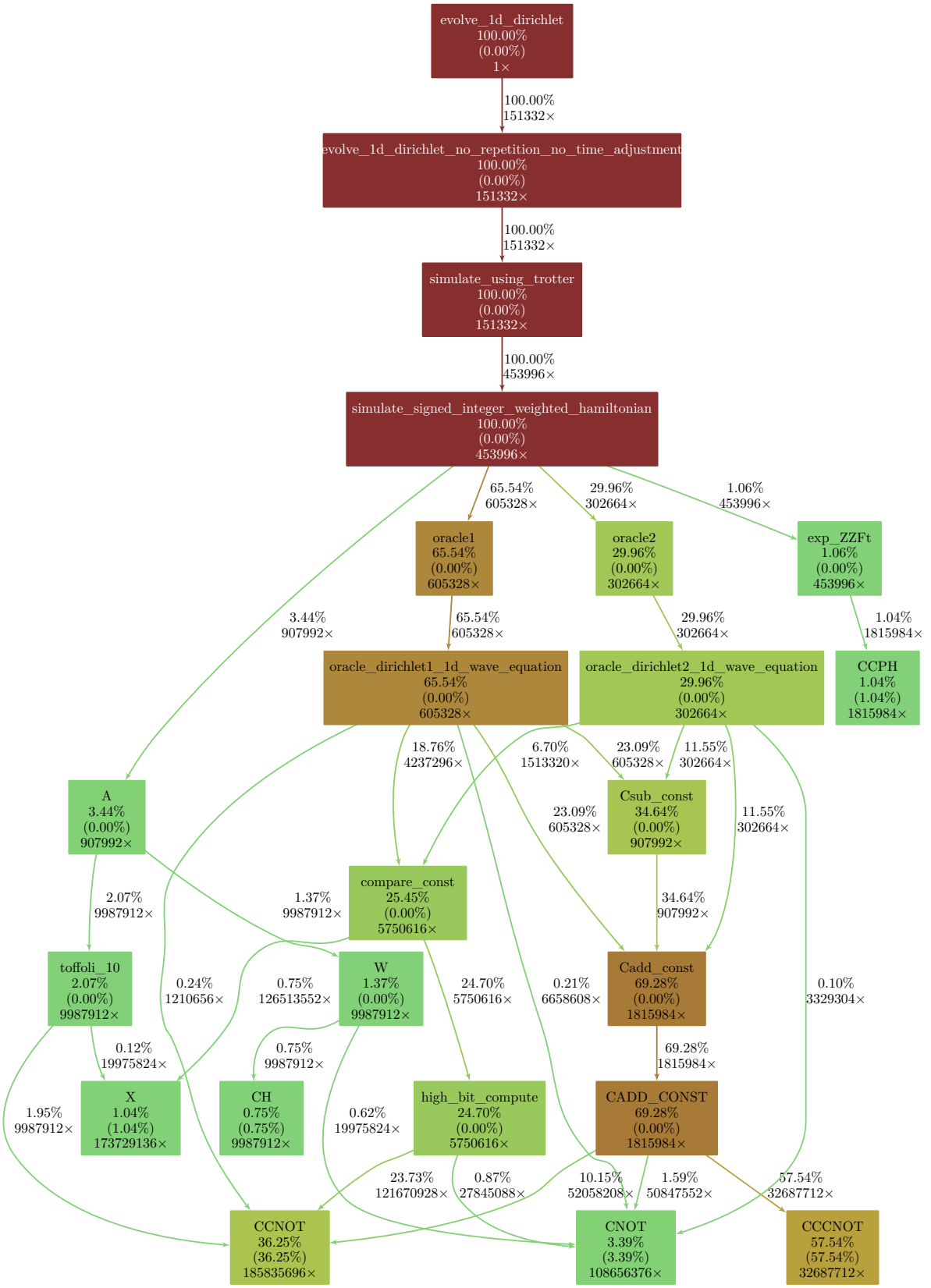


Figure 4.16: *Call-graph of the quantum wave equation solver. Nodes (i.e. quantum routines) that account for less than* 0.5% *of the total execution time are not plotted. Edges (i.e. subroutine calls) that account for* 0.1% *or less of the total execution time are also discarded for readability purposes.*

4.6 Discussion

Now that we have described qprof internals and how to use it on quantum circuits, we can compare the insights it provides with the current state-of-the-art. We also discuss the current limitations of the tool and potential improvements that could be added in the future.

4.6.1 Comparison with the state-of-the-art

A description of the profiling or resource estimate capabilities of several widely used quantum computing frameworks have been provided in Section 4.2.2.

One of the first advantages provided by qprof comparatively with the frameworks presented in Section 4.2.2 is its framework agnostic interface. As explained in Section 4.3.2 and shown in Listings 4.1 to 4.3, qprof can handle nearly transparently different quantum computing frameworks and provide a standardised report. The fact that qprof has been architectured as shown in Figure 4.2 allows it to decouple entirely the framework used to represent the profiled quantum circuit from the output format. It means that if a new exporter is implemented in the future, it will be available for all the implemented frameworks. Conversely, if a new framework adapter is added to qcw, qprof will directly be able to generate reports using all the already existing exporters. This decoupling, crucial due to the increasing number of quantum computing frameworks, has not been implemented by any of the existing resource estimation features listed in Section 4.2.2, each framework providing features that are only compatible with its own quantum circuit representation.

Additionally qprof already provides a more detailed report than most of the quantum computing frameworks listed in Section 4.2.2. The Q# Flame graph exporter provides the same type of information by using a different visualisation format (Flame graphs [146]) but seems to be less flexible than qprof with respect to the quantities that can be profiled.

4.6.2 qprof and quantum circuit compilation

qprof might be used to understand the impact of quantum compilation on a given quantum circuit provided that the compilation tool-chain used does not destroy the call-graph structure of the quantum circuit.

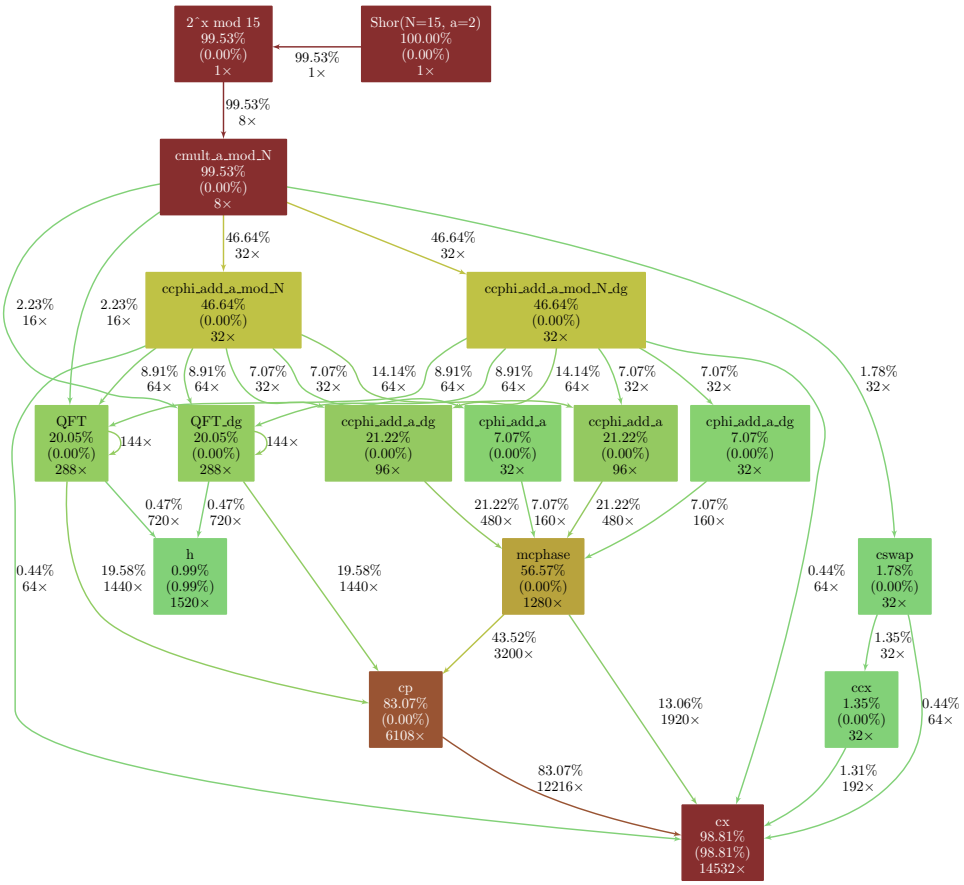
One of the only strong requirement of the qprof tool is that the quantum circuit provided can be explored using the unified interface provided by qcw. But in order for qprof to generate a *useful* report, a few other requirements should be checked.

First, routine names should be informative and human-readable. This requirement seems trivial at first sight, but quantum program compilers might generate routines, for example using quantum circuit synthesis algorithms [147–149], and the name attached to the generated quantum circuit might not be informative at all.

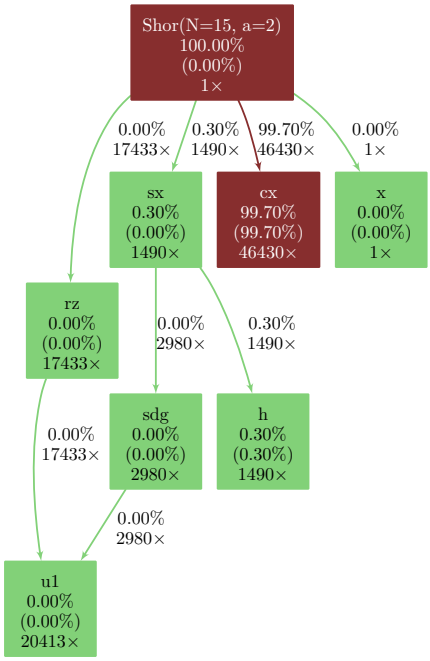
Secondly, and even more importantly, the profiled quantum program should contain enough information about the routines and subroutines used. Some compilers such as the one used by Qiskit at the time of writing (version `0.32.1`) start the compilation process by flattening the quantum circuit and unrolling all the quantum gates that are not in the basis provided. As soon as the quantum circuit has been flattened, all the call-graph information is lost and cannot be retrieved by qprof anymore, making its report less useful when the profiled circuit has been flattened.

Figure 4.17 illustrates this issue with an implementation of Shor's algorithm trying to factorise the number 15: qprof report before the transpilation provides enough information to plot a meaningful call-graph as shown in Figure 4.17a whereas qprof report for the exact same circuit but after calling Qiskit transpiler (Figure 4.17b) contains nearly no useful information.

This means that qprof will only interact nicely with compilers if and only if the compiler used is able to keep relatively untouched the structure of the call-graph. Currently, only a few



(a) *Call-graph obtained on the circuit generated using Qiskit implementation of Shor's algorithm trying to factorise* 15.



(b) *Call-graph obtained on the circuit generated using Qiskit implementation of Shor's algorithm trying to factorise* 15. `qiskit.transpile` *has been used on the generated quantum circuit with* `ibm_cairo` *as the backend and* `optimisation_level=2`.

Figure 4.17: *Effect of using Qiskit transpiler on qprof reports. Using Qiskit transpiler flattens the quantum circuit and effectively replace every routine call that is not in the transpilation basis by equivalent calls to gates in the transpilation basis, making qprof report less informative and useful. Note that Qiskit transpiler inserted CX gates in order to make the quantum circuit compliant with* `ibm_cairo` *topology, which is why the gate cost of the* `cx` *gate became even more predominant in the transpiled circuit.*

compilers are able to do so but projects like QCOR [150] may help democratising this approach. For compilers that check this property, qprof will be able to help visualising the effect of compiler on the circuit costs by plotting the call-graphs of the original and compiled circuits side by side and comparing the different costs computed.

4.6.3 qprof and hardware-aware timings

The fact that most of the current compilers are flattening the compiled circuit makes qprof reports less meaningful and informative as shown in Section 4.6.2. Not being able to use compilers restrict the class of quantum circuits that might be sent to qprof: hardware-compliant circuits are not likely to be analysed for the moment. This is due to the fact that to get an hardware compliant circuit, one should either use a compiler, which is not possible yet as discussed earlier, or build a hardware-compliant circuit directly, which is an exceedingly complex task for large circuits.

Because hardware-compliant circuits are, for the moment, unlikely to be studied with qprof, the tool is not yet capable of adapting the costs of a given gate depending on the qubits it is applied on.

4.6.4 Limitations of the `gprof` exporter

The main output format for qprof reports are based on the output format of `gprof` [140, 141] for several reasons: standard format, widely used during decades, human-readable, availability of external tools to get visual representations from the textual format, etc. But this output format is inherently limited to sequential programs, which impose a strong limitation on what it can represent. When exporting using the `gprof`-based format, qprof will not take into account gate parallelism, i.e. as if quantum gates were executed sequentially, one at a time. Trying to take into account gate parallelism using the `gprof`-based format leads to percentages not adding up to 100% which was deemed too confusing to be worth implementing.

4.6.5 qprof and NISQ circuits

qprof is currently only using a limited set of information on the profiled quantum routines. In particular, even though the information is available through qcw for some frameworks, qprof ignores on which qubits a particular routine is applied on for the moment.

By extending qcw public interface in Figure 4.7 to include a way to access qubits the routine is applied on and modifying slightly Algorithm 2 (see comment above line 16) to allow non-additive quantities to be profiled, qprof would be able to include gate error or topology in its reports.

The gate error *estimation* would be a nice addition for NISQ algorithms, even though only providing a *lower bound* on the real error that would be observed on hardware due to the presence of other source of errors such a decoherence, cross-talk or "SPAM" (state preparation and measurement) errors.

Reporting on topology has its own challenges, one of them being to find a good format for qprof report as the `gprof` format is not adapted to include such information.

4.6.6 qprof and dynamical circuits

qprof being a static analyser, it does not support dynamical circuits that may use the result of a previous quantum operation to determine which is the next quantum gate to execute. Moreover, the features related to dynamic circuits are still not introduced in a lot of quantum computing frameworks and, for the frameworks that do implement some of them, are relatively new. As such, the companion package qcw and the unique interface it provides has not been updated to include information about dynamic circuits.

4.7 Conclusion

In this chapter we introduced qprof, an open-source and, to the best of our knowledge, novel tool that is able to generate profiling reports in well-known formats from a quantum circuit implementation. Our library is able to natively read quantum circuits from multiple frameworks — currently Qiskit, myQLM, OpenQASM 2.0 and XACC — and can be easily extended to support more quantum computing libraries. It generates consistent reports independently of the underlying framework used. qprof opens new optimisation opportunities for quantum scientists and programmers by allowing them to view their quantum circuit implementation in a well-known, synthetic and visual representation.

In this chapter, we presented the main concepts used in the internals of qprof: how is qprof able to be framework-agnostic thanks to a unique interface provided by qcw, the processing performed by qprof in order to compute quantities of interest to profile and how exporters are used to output the profiling report in a usable and convenient format. We then analysed qprof runtime performance by providing asymptotic complexity estimates, examples of worst- and best-case quantum circuits, and benchmarked execution times on several well-known quantum circuit implementations. We also used qprof on three different quantum circuit implementations of increasing complexity to demonstrate its features: simplicity of use, adaptability and consistency of the interface and generated reports.

Finally, we discussed potential improvements and limitations of qprof, opening the way for more development on the library. In the future, we plan to extend the set of supported quantum computing frameworks. The number of exporters can also be improved to handle different output formats such as a `perf_event` [151] compatible format or a Flame graph [146] compatible one, allowing to easily use new visualisations such as Flame graphs [146].

Supplementary material

The qprof tool is available at `https://gitlab.com/qcomputing/qprof/qprof`. The different qcw packages are available at `https://gitlab.com/qcomputing/qcw`.

Part IV

Targetting NISQ

Chapter

5 Hardware aware compiler

In this chapter, we study a critical part of the quantum computing stack: the compiler. We present an improved algorithm to solve the qubit mapping problem by taking into account the last calibrations of the targeted quantum chip, making the algorithm "hardware-aware" or "noise-aware".

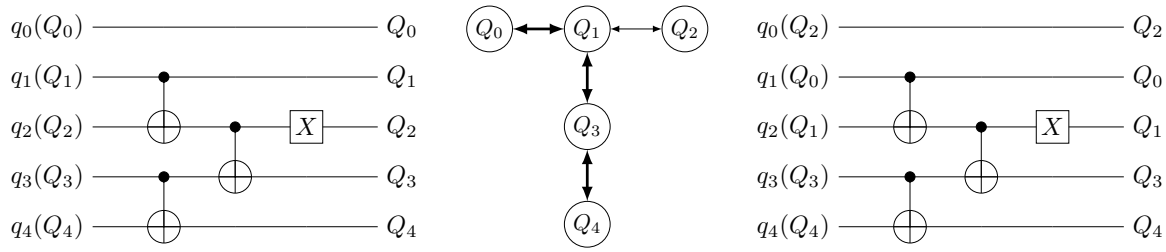
Contents

5.1 Introduction

In the gate-based model of quantum computations presented in Section 1.2.3, a *computation* is represented by a quantum circuit (see Section 1.4.2). In order to perform any computation on a quantum chip, the quantum circuit describing the computation have to be constructed. The quantum circuit construction, also known as *implementation* of the computation, is a task that is performed manually by programmers with the help of programming languages and libraries. The task of implementing an algorithm targeting a specific hardware is extremely complex as it requires to solve a multi-objective optimisation problem manually: the code should be correct and have to check an often daunting list of requirements imposed by the hardware used, but also have to be efficient, readable and ideally re-usable easily on other hardware.

This multi-objective optimisation problem often imposes too much constraints on the implementation to be realistically solvable by a human in a reasonable time. In order to circumvent this issue and ease the task of programmers (as well as improving their productivity), most of the constraints have been offloaded to others specialised programs. Classical computing programmers of today only have to write a correct and readable code in a given programming language and the tasks of adapting and optimising this code for a given hardware is mostly left to the *compiler*.

As what is witnessed in classical computing, quantum chips also have requirements on the type of computations that can be executed. A quantum circuit can only be executed on a given quantum chip if it exclusively uses quantum gates natively implemented by the hardware.



(a) *Implemented quantum circuit*

(b) `ibmq_valencia` *topology*

(c) *Compiled quantum circuit*

Figure 5.1: *A first motivational example for the qubit mapping problem. Here, the targeted hardware (*`ibmq_valencia`*) has the required topology (highlighted in bold) to execute the implemented quantum computation without any change to the circuit. Small cased labels represent* logical *qubits and upper cased labels represent* physical *qubits (see Definitions 16 and 17).*

This simple restriction in appearance has in fact deep implications, effectively making most of the quantum computations in-executable on hardware. This is due to the fact that the set of natively implemented quantum gates effectively imposes a hardware "topology".

Definition 15 (Hardware topology)**.** Any quantum chip has a *topology* that is induced by the set of native multi-qubit quantum gates implemented by the hardware. The topology of a given quantum chip is the set of all the native inter-connections between hardware qubits. In the special and most widely seen case of hardware providing only 1- and 2-qubit quantum gates, the topology is defined as the set of pairs of qubits that are natively connected.

The problem of respecting the target hardware topology is archetypal of the kind of problems that have to be offloaded to an automated third-party program: as each quantum hardware might have very different topologies, a valid quantum computation on a given hardware is likely to be invalid on any other quantum hardware due to a topology mismatch, making the implementation non-portable across chips.

5.1.1 Motivational examples

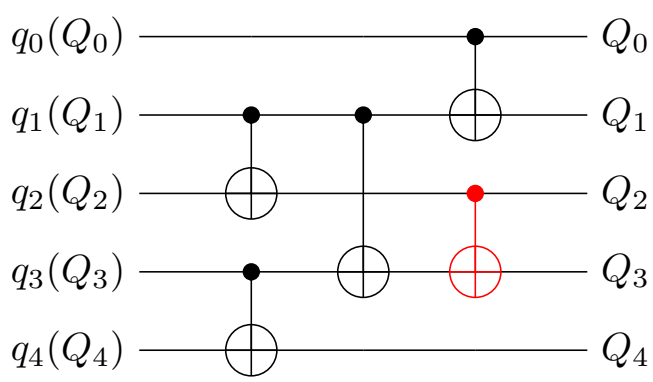
Simple motivational example

A small and quite simple quantum circuit is depicted in Figure 5.1a. This circuit is composed of three `CNOT` gates and one `X` gate. The three 2-qubit gates are defining the topology required by the circuit in order to run natively on a given quantum hardware: 4 qubits have to be linearly connected (or chained) together. This is the case of the quantum chip `ibmq_valencia` depicted in Figure 5.1b with the qubits labelled 0, 1, 3 and 4.

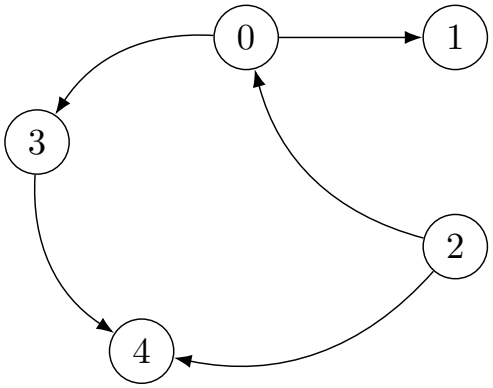
Definition 16 (Physical qubit)**.** A piece of hardware that is part of a quantum chip and implementing a qubit.

Definition 17 (Logical qubit)**.** In the context of this chapter, a *logical* qubit is represented by classical computer data (often a quantum register name and an index) and is an abstract representation of a qubit, not linked with any physical qubit. This abstraction is used to implement hardware-agnostic quantum computations, leaving all the hardware-aware part to the compiler.

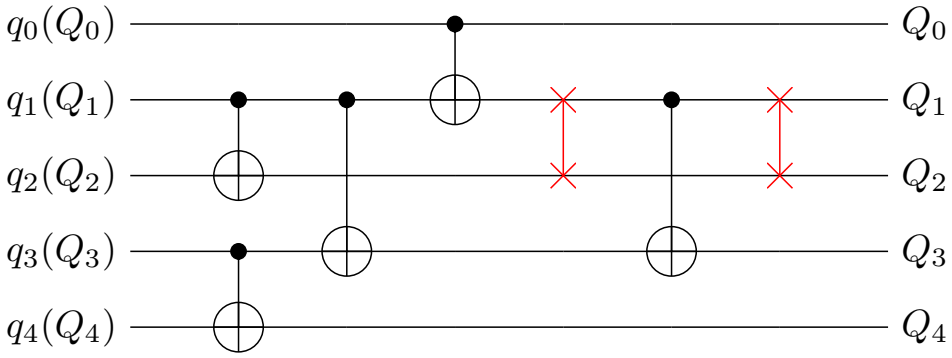
Note 13. Definition 17 introduces the concept of *logical qubit* but the naming convention interferes with a most widespread definition in the quantum error correction field. In this chapter, except if explicitly denoted otherwise, a *logical qubit* should be read as explained in Definition 17 and does not mean an error-corrected qubit.



(a) *Implemented quantum circuit. The last* `CNOT` *gate between* Q_2 *and* Q_3 *is not native and as such cannot be executed directly.*



(b) *Topology required by the quantum circuit. This graph is not a sub-graph of the hardware topology from Figure 5.1b.*



(c) *One possible output of the compiler. The quantum circuit is now only using native* `CNOT` *gates.*

Figure 5.2: *A second motivational example for the qubit mapping problem. Here, the targeted hardware (*`ibmq_valencia`*) does not have the required topology to natively execute the implemented quantum computation. Small cased labels represent* logical *qubits and upper cased labels represent* physical *qubits (see Definitions 16 and 17).*

As the hardware topology is compatible with the circuit topology, we only have to find a *mapping* between logical and physical qubits that makes the physical quantum circuit executable.

Definition 18 (Mapping between logical and physical qubits)**.** In this context, a mapping is a bijection that associates to each logical qubit in the circuit a physical qubit from the targeted hardware.

Finding a valid mapping can be reformulated as an instance of the sub-graph isomorphism problem where we are trying to find a sub-graph of the hardware topology that is isomorph to the graph representing the topology required by the quantum computation. The sub-graph isomorphism is a NP-complete problem [152] but there exist algorithms that are efficient for some instances [153].

More complex motivational example

A more complex motivational example is depicted in Figure 5.2a. In this example, the topology required by the quantum circuit (depicted in Figure 5.2b) is not compatible with `ibmq_valencia` topology (see Figure 5.1b).

Because quantum hardware topology is set and cannot be changed easily, we are left with only one way of performing the computation represented in Figure 5.2a on the targeted hardware (`ibmq_valencia` here): change the computation to an *equivalent* computation that checks the

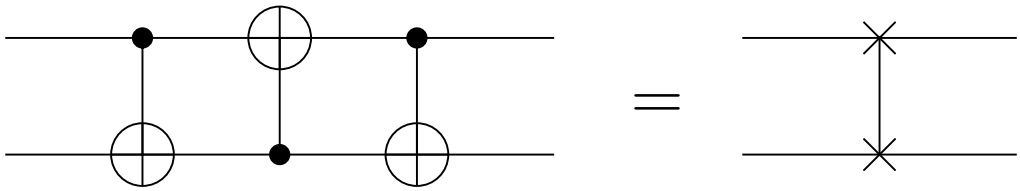


Figure 5.3: *SWAP gate. This gate exchange the state of* 2 *qubits.*

topology requirements imposed by the hardware. A valuable tool to reach this goal is the `SWAP` gate shown in Figure 5.3. The `SWAP` gate can be used to exchange 2 qubits states, allowing to change the mapping between logical and physical qubits at the time of execution.

An equivalent quantum circuit that checks the hardware topology is depicted in Figure 5.2c.

5.1.2 Automatically adapting any quantum computation to a given topology

The problem of automatically adapting a quantum computation (i.e., a quantum circuit) to a given topology has already been studied in several research papers and is a vibrant and active domain of research.

Two main types of methods have been used in the literature to solve this problem. The first method consists in reformulating the problem as a mathematically equivalent problem that can then be solved using a specialised solver. The target mathematically equivalent problem often uses the formalisms of Integer Linear Programming (ILP) [154–157], Satisfiability Modulo Theory (SMT) [158, 159], or even Constraint Programming (CP) [160, 161]. These approaches have the huge advantage of building over well known and studied optimisation problems for which highly efficient solvers are available. Moreover, most of the solvers available are able to compute the exact optimal solution to the problem, minimising a cost function that is often the number of gates added to the quantum circuit to adapt it. However, all these methods are difficult to scale for larger quantum circuits, either containing more qubits or containing more quantum gates, and as such suffer from very long runtime even for medium-sized problems. The second type of methods use heuristic algorithms to adapt the quantum circuit, starting from the first quantum gate and transforming the circuit sequentially by making each gate one after the other hardware-compliant.

Most of the previous works [162–166] using the second method are specialised to a nearest-neighbour connectivity and cannot be directly applied to actual quantum architectures such as `ibmq_valencia` or `ibmq_almaden` that do not have a nearest-neighbour connectivity (Figures 5.1b and 5.4). More recent work [167–174] introduced approaches and algorithms that are not restricted to a specific topology. For example, the algorithm presented in [167] uses an heuristic approach to find the best permutation at each step of the quantum circuit transformation. Instead of representing a quantum circuit as a fixed sequence of layers like most of the previous works, [175] introduced a Directed Acyclic Graph (DAG) representation that takes into account commutativity of quantum gates and their time dependencies. A major improvement has been shown by [169] which uses a "forward-backward-forward" algorithm. Moreover, a "look-ahead" strategy has been introduced in the heuristic cost function for further optimisation in some existing works, notably [167, 169–171, 174]. Most of the methods and algorithms cited only use `SWAP` gates to adapt the original quantum circuit. A notable exception is [171] that considered both `Bridge` and `SWAP` gates. Finally, most of these works aim at minimising the number of inserted gates and do not consider the impact a imperfect quantum gates and the error-rate variations that can be witnessed between different qubits.

The algorithms used in [155, 158, 176–178] try to use calibration data of the targeted quantum chip in order to insert additional 2-qubit (`SWAP`) gates between strongly linked qubits, i.e., qubits that are linked with a low 2-qubit gate error rate. However, these works do not consider a holistic

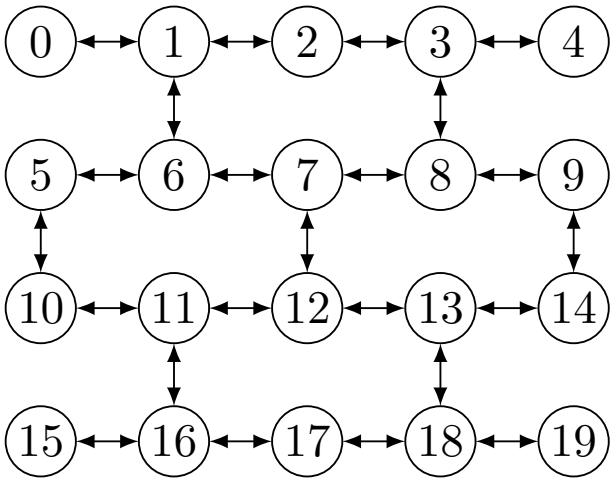


Figure 5.4: *`ibmq_almaden` topology. Qubits are represented as circles and indexed from* 0 *to* 19*. A connection between two qubits is represented by an edge between the two qubits.*

Qubit number	20
Single qubit error rate	$2.655e^{-4}$ to $9.569e^{-4}$
CNOT error rate	$8.136e^{-3}$ to $3.403e^{-2}$
`id` gate length	35.56 ns
`u1` gate length	0 ns
`u2` gate length	35.56 ns
`u3` gate length	71.11 ns
`CNOT` gate length	248.88 ns to 860.44 ns
T1	34.66 μs to 139.46 μs
T2	12.16 μs to 200.25 μs

Table 5.1: *`ibmq_almaden` characteristics. Note that the exact hardware characteristics are not constant and change at each re-calibration of the chip.*

view of the problem, either lacking a good and automatic way of "seeding" the solver or using heuristic methods that are not efficient enough to select the best candidates in some common cases.

In this work, we follow the second type of methods that consist in developing a heuristic algorithm to choose the best `SWAP` to insert based on calibration data. We propose a Hardware-Aware (HA) heuristic mapping transition algorithm to address the drawbacks mentioned above. First, we present a mapping transition algorithm that takes into account the hardware topology and the calibration data to improve the overall output state fidelity and reduce the total execution time. Second, to reduce the number of additional gates required to map the quantum circuit to the quantum chip, our algorithm is able to select between a `SWAP` or `Bridge` gate. Finally, we ran our HA algorithm on real quantum hardware and compared it with various other algorithms from the literature.

5.1.3 Examples of quantum hardware

Figure 5.4 shows the topology, also called coupling graph, of IBM Quantum's `ibmq_almaden`, a 20-qubit system. Each vertex represents a qubit and each edge represents the coupling interconnect between the two qubits it links. Table 5.1 lists a summary of the calibration data that have been extracted from [179] for `ibmq_almaden`. It includes `CNOT` error rates, single qubit error rates, energy relaxation and decoherence characteristic times T1 and T2, and execution time

(gate length). The calibration data show that the error of the only 2-qubit gate is one order of magnitude higher than their 1-qubit counterparts. This is also the case for gate execution times: 2-qubit gates are approximately an order of magnitude slower than 1-qubit gates. For simplicity and because of the relatively low error rates and execution times of 1-qubit gates when compared to 2-qubit gates, we focus on 2-qubit gates in this paper.

Moreover, it is important to note that all the interconnects between qubits are not equal. When looking at the `CNOT` gate error rate or execution time on `ibmq_almaden`, the best `CNOT` gate has an error rate 4.18 times lower than the worst `CNOT` and the maximum execution time is 3.46 times longer than the minimum one. Therefore, as rightly noted in [176], we cannot treat each qubit equally, and we have to consider the topology as well as their error rate. `CNOT` gates can be applied in either direction by conjugating with H gates. As we do not consider 1-qubit gates in this study, we do not have to consider any "native" `CNOT` direction.

5.2 Proposed solution

Our algorithm improves over the SABRE algorithm presented in [169], which is a `SWAP`-based heuristic algorithm to reduce the number of additional `CNOT` gates. We propose a Hardware-Aware `SWAP`- and `Bridge`-based heuristic search algorithm. Compared to the SABRE algorithm, which only aims at reducing the number of additional gates, we improve the final circuit fidelity as well as reduce the number of additional gates by introducing a new distance matrix that takes into account both the hardware connectivity and the last calibration data available for the targeted chip. Moreover, SABRE algorithm only uses `SWAP` gate whereas our algorithm is able to decide between `SWAP` and `Bridge` gates to further reduce the number of additional gates. Finally, we also develop an initial mapping algorithm called Hardware-aware Simulated Annealing (HSA) in order to evaluate the mapping transition algorithm of different flavours.

The introduced algorithm takes as input a quantum program written in the OpenQASM 2.0 language [180] and the calibration data of a specific IBM quantum device. During the compilation process, it considers the hardware constraints such as hardware topology, gate availability and error rates. Then, the quantum circuit transformation algorithm is applied. It contains two main parts: a initial mapping algorithm and a mapping transition algorithm. In the mapping transition step, some optimisations are done to generate a circuit with a better performance in terms of final state fidelity. The source code is publicly available at `https://github.com/peachnuts/HA`.

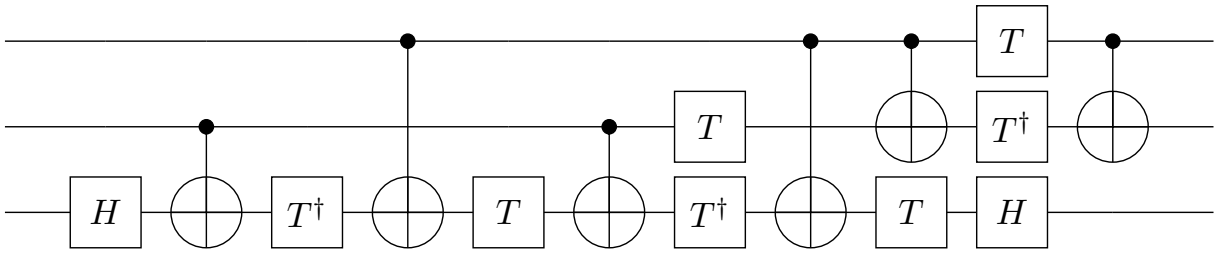
We start by explaining our HA algorithm in Section 5.2.1. In Section 5.2.2, we describe the hardware-aware simulated annealing (HSA) method for initial mapping. Finally, Section 5.2.3 presents the metrics used to evaluate our algorithm.

5.2.1 Hardware-aware `SWAP`- and `Bridge`-based heuristic search

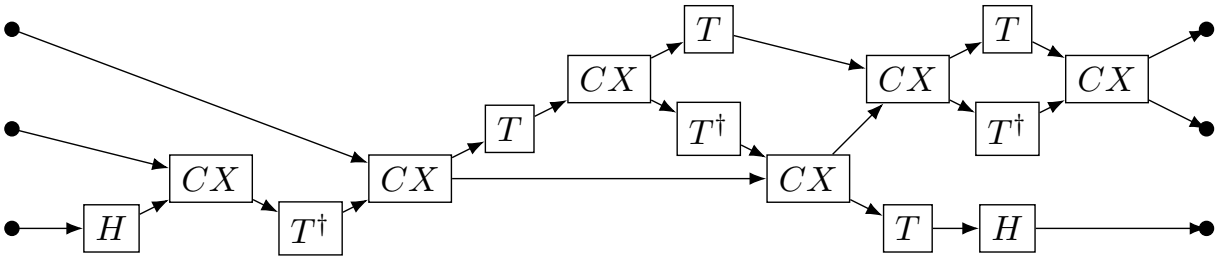
Initialisation of the algorithm

The first step of the algorithm is to process the input quantum circuit in order to reformulate it in a more convenient data format. Starting from the input quantum circuit, we can obtain a Directed Acyclic Graph (DAG) circuit which represents the operation dependencies in the quantum circuit without considering the hardware constraints. The DAG is constructed such that quantum gates are represented by the graph nodes and the directed edge (i, j) between nodes i and j represents a dependency from gate i to j, i.e., gate i should be executed before j. Figure 5.5 shows an example of a quantum circuit that is then transformed into a DAG.

Once the DAG is constructed, graph nodes (i.e., quantum gates) can be ordered following gate dependencies. For example if gate j depends on gate i, then gate i will be ordered before gate j. One possible ordering that fulfil this property is the well known topological ordering. Depending on the quantum circuit, a topological ordering might not be unique.

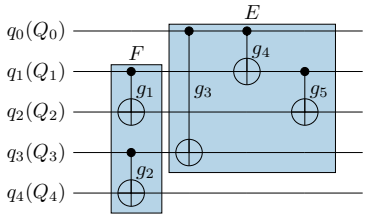


(a) *An example of a quantum circuit.*

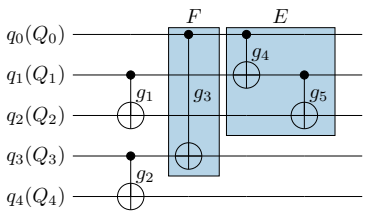


(b) *The DAG representation of the quantum circuit example.*

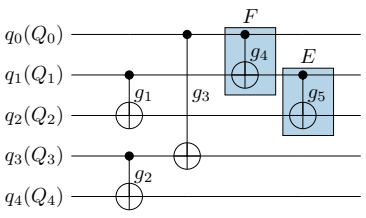
Figure 5.5: *Transformation of a quantum circuit to a Directed Acyclic Graph (DAG).*



(a) *Beginning of the HA mapping algorithm. g_1 and g_2 do not overlap and are the first gates in the circuit so they are in the front layer F. The other gates are not executable and are consequently in the extended layer E.*



(b) *g_1 and g_2 are compliant with the hardware topology. They are executed and removed from F. The gate g_3 is pushed into F as it is now "executable". g_4 overlaps with (i.e., depends on) g_3 and as such cannot be inserted in F.*



(c) *After* `Bridge` *insertion, g_3 is executed and removed from F. g_4 no longer overlap with a gate in F and is added to the front layer. g_5 overlaps with g_4 and so should stay in E.*

Figure 5.6: *Evolution of the front (resp. extended) layer F (resp. E) on a simple circuit with a detailed explanation at each step.*

Quantum gates can then be divided into three groups: the executed gates, the executable gates, and to be executed future gates. Executed gates are quantum gates that have already been mapped by the algorithm. Executable gates constitute the front layer, denoted F. A gate is considered executable when all the gates it depended on are in the executed gates group. Finally, gates that are not yet executed nor executable are included in the extended layer E. An illustration of layers E and F is shown in Figure 5.6.

The HA algorithm

The core of the HA algorithm then starts by checking sequentially the quantum gates of the circuit, following a topological ordering. If the quantum gate currently being checked is natively *executable* (i.e., all its predecessors are *executed* and the gate topology is compatible with the hardware topology) the algorithm marks this gate as *executed* and go on to the next gate. Else, if the gate is not natively executable, it is added to the front layer F. By following a topological order, the algorithm ensures that for each explored gate, all its predecessors are either executed or in the front layer. This process is then repeated until no more quantum gate can be added to the front layer F, i.e., when the currently explored quantum gate has predecessors in the front layer or when no gate is left to explore.

When the front layer F is full, the HA algorithm calls a heuristic function to choose the best `SWAP` or `Bridge` gate to insert in order to make some of the gates in the front layer executable. If any gate becomes executable after a `SWAP` insertion (or if it is executed by inserting a `Bridge`), it is removed from the front layer and marked as executed. The algorithm then iterates, adding gates to the front layer and inserting `SWAP` or `Bridge` gate until the quantum circuit is fully mapped.

The skeleton of the main algorithm can be found in [169, Algorithm 1] with only some minor adjustments to allow the insertion of `Bridge` gates that will not change the current mapping, denoted as π in the algorithm.

The algorithm used to choose what is the best `SWAP` gate to insert at a given point in the algorithm as well as select the best candidate between a `SWAP` and a `Bridge` gate is based on a heuristic cost function described in Algorithm 3. In order for this heuristic to work best, the most recent calibration data should be retrieved through the IBM Quantum Experience or Qiskit API before each usage of the HA algorithm to ensure that the algorithm has access to the most accurate and up-to-date information possible.

The heuristic method to insert a `SWAP` or `Bridge` starts by constructing a list of all the candidate `SWAP` gates, named `swap_candidate_list` in Algorithm 3, from the quantum gates in the front layer F and the hardware coupling graph G. Then, for each `SWAP` candidate a temporary mapping π_{temp} is computed with the `Map_Update` function. The final cost of the candidate `SWAP` is computed following Equation (5.5). The `SWAP` with the minimum score is selected and called SWAP_{min}.

The last step is to choose between a `SWAP` gate or a `Bridge` gate. A `SWAP` gate can always be used, whereas a `Bridge` gate can only be inserted if a gate in the front layer F becomes executable from the mapping obtained after applying the SWAP_{min} gate. If the conditions to insert a `Bridge` gate are not met, HA algorithm inserts a `SWAP` gate. Else, the algorithm decides the gate (`SWAP` or `Bridge`) to insert based on the *effect* of the `SWAP` gate on the extended layer E. The `SWAP` gate effect is computed with Equation (5.6). A negative effect implies that changing the mapping with a `SWAP` gate will have a short-term negative impact in the future, making the gates in the extended layer harder to adapt. In the case of a negative effect, a `Bridge` gate, which does not change the current mapping, is inserted. Otherwise, if adding a `SWAP` gate has a positive effect on the extended layer, the algorithm inserts a `SWAP` gate.

Algorithm 3: Heuristic algorithm for selecting additional gate candidate

input : Circuit DAG, Coupling graph G, Current mapping π_c, Distance matrix D, Swap matrix S, front layer F, Extended layer E, Weight parameter W

output: New mapping π_n, Inserted gate g_{add}

```
1  begin
2      Set score to empty list;
3      Set effect to empty list;
4      swap_candidate_list ← FindSwapPairs(F, G);
5      for swap ∈ swap_candidate_list do
6          π_temp ←Map_Update (swap);
7          H_basic ← 0;
8          for gate ∈ F do
9              H_basic ← H_basic + D(gate, π_temp) ;
10         end
11         H_extended ← 0;
12         for gate ∈ E do
13             H_extended ← H_extended + D(gate, π_temp) ;
14             effect_cost ← effect_cost + D(gate, π_c) − D(gate, π_temp);
15         end
16         H ← (1/|F|) H_basic + (W/|E|) H_extended ;
17         score.append(H);
18         effect.append(effect_cost);
19     end
20     Find the swap with minimum score: swap_min;
21     Find the gate in F that become executable by applying swap_min: g_s;
22     if effect [swap_min] < 0 and S(g_s,π_c) = 2 then
23         π_n ← π_c;
24         g_add ← g_B;
25     else
26         π_n ← Map_Update (swap_min);
27         g_add ← swap_min;
28     end
29     return π_n, g_add;
30 end
```

Heuristic cost function for SWAP pairs

A heuristic cost function H is introduced to estimate the cost of each possible (i.e., natively executable) SWAP pairs at a given step of the iterative algorithm. Its objective is to quantify the quality of the possible SWAP pairs according to the distance considered and to select the best SWAP pair.

When inserting a SWAP gate, the circuit is divided into two layers: the front layer F and the extended layer E. Note that inserting a SWAP gate will not only influence the gates in the front layer F but also the gates in the extended layer E. The approach of considering the SWAP pair's impact on the extended layer is referred as the look-ahead ability. It can contribute to an overall better performance of the algorithm and depends on the size of the extended layer E.

In order to build the HA algorithm presented in this chapter, we devised several metrics that can be used to estimate the cost of a SWAP pair. We considered three different distance matrices: swap matrix S, swap error matrix $\mathcal{E}$ and swap execution time matrix T. Because S, $\mathcal{E}$, and T contain entries with incompatible units and different scales, we update T to make it dimensionless and each matrix is normalised. Moreover, we introduce weights (α_1, α_2, and α_3 for S, $\mathcal{E}$, and T, respectively) to allow to choose the importance of each of the parameters: number of SWAPs, SWAP gate error and SWAP gate execution time.

Matrix S is constructed such that the entry (i, j) stores the distance on the real hardware between qubit i to a neighbour of qubit j, which is also equal to the minimum number of SWAP gates needed to move qubit i to qubit j. The matrix is efficiently constructed by using the Floyd-Warshall algorithm [181].

Matrix $\mathcal{E}$ stores in its entry (i, j) the minimum error rate attainable to move the qubit i to a neighbour of qubit j. The error rate of each possible SWAP is computed based on the calibration data of the native CNOT gates and the decomposition shown in Figure 5.3.

The success rate of a CNOT between the physical qubits Q_i and Q_j, denoted by $S(Q_i, Q_j)$, is computed from the error rates given in the calibration data. Equation (5.1) computes the error rate of a SWAP gate between two connected physical qubits Q_i and Q_j while taking into account that the swap operation is symmetric. The final $\mathcal{E}$ matrix is constructed by using the Floyd-Warshall algorithm on the graph $G_{\mathcal{E}}$ with the computed errors as edge weights.

$$G_{\mathcal{E}}(Q_i, Q_j) = 1 - S(Q_i, Q_j) \times S(Q_j, Q_i) \times \max(S(Q_i, Q_j), S(Q_j, Q_i)) \tag{5.1}$$

Matrix T is computed, similarly as S and $\mathcal{E}$, with the Floyd-Warshall algorithm applied on graph G_T but by using the SWAP execution time. This execution time is computed with Equation (5.2) where $t(Q_i, Q_j)$ is the execution time of the CNOT gate with Q_i as control and Q_j as target, extracted from the calibration data.

$$G_T(Q_i, Q_j) = t(Q_i, Q_j) + t(Q_j, Q_i) + \min(t(Q_i, Q_j), t(Q_j, Q_i)) \tag{5.2}$$

The summation of the three matrices forms a new matrix called distance matrix D (shown in Equation (5.3)). The distance matrix represents the "distance" between each pair of qubits in the quantum chip. Here, the "distance" means the combination of swap distance, overall error rate and execution time of the shortest path.

$$D = \alpha_1 \times S + \alpha_2 \times \mathcal{E} + \alpha_3 \times T \tag{5.3}$$

Inserting a SWAP gate will have an impact on the current mapping π_c, changing it to π_{temp}. We compute the cost of this SWAP on the front layer F with the cost function H_{basic} shown in Equation (5.4). A small score means that the mapping π_{temp} makes the hardware topology close the topology required by the gates in the front layer F. The SWAP pair with the minimum score

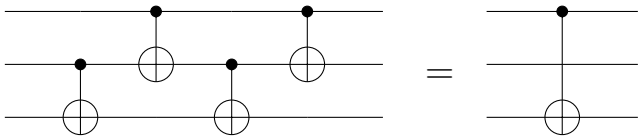


Figure 5.7: *Implementation of the* `Bridge` *gate. This gate can be useful to implement a* `CNOT` *between two qubits that share a common neighbour without changing the current mapping.*

is selected as the best candidate.

$$H_{basic} = \sum_{g \in F} D[\pi_{\text{temp}}(g.q_1)][\pi_{\text{temp}}(g.q_2)] \tag{5.4}$$

We also consider the impact of the `SWAP` pair on the extended layer E. The impact of a `SWAP` on the front layer is prioritised over its impact on the extended layer. As a result, a weight parameter W is added to the extended layer cost to scale its impact. Moreover, the impacts on the front layer and extended layer are normalised by dividing them with their respective number of gates. The complete heuristic function including the extended layer E with look-ahead ability is shown in Equation (5.5). Even though Equation (5.4) and Equation (5.5) are similar to equations in [169], it is important to note that the distance matrix D is different.

$$H = \frac{H_{\text{basic}}}{|F|} + \frac{W}{|E|} \sum_{g \in E} D[\pi_{\text{temp}}(g.q_1)][\pi_{\text{temp}}(g.q_2)] \tag{5.5}$$

Heuristic cost function to estimate the effect of `SWAP` versus `Bridge` gates

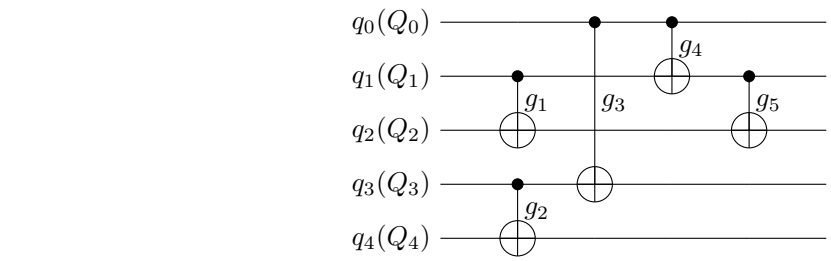
An equally important metric of the HA algorithm is the heuristic cost function that estimates the usefulness of a `SWAP`. In some situations (that happen on real topologies and circuits), even the best `SWAP` may have a negative impact on the overall circuit. In that case, inspired by [171], our heuristic function decides to insert a `Bridge` gate instead of a `SWAP` gate if the topology allows it. The decomposition of the `Bridge` gate with four `CNOT`s is shown in Figure 5.7. The `Bridge` gate allows executing a `CNOT` between two qubits that share a common neighbour.

It is important to note that using a `Bridge` gate does not induce an additional cost in term of gate number when compared to a `SWAP` gate. Actually, both gates require 4 `CNOT` gates in total (for the `SWAP` gate: 3 gates to implement the `SWAP` and 1 additional gate to perform the `CNOT`). The `Bridge` gate can only be used to replace a `CNOT` if the distance between the control and target qubits (i.e., the minimum number of links between the two qubits) is exactly two.

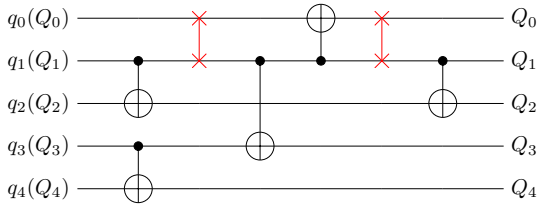
Figure 5.8a shows an example of quantum circuit that is mapped to `ibmq_valencia` with the topology described in Figure 5.1b. The quantum gates g_1 and g_2 comply with the topology of the chip, but g_3 does not. By evaluating the heuristic cost function H, the `SWAP` between q_0 and q_1 is selected. But as shown in Figure 5.8b, the chosen `SWAP` has a negative impact on the extended layer: gate g_5 is no longer executable and another `SWAP` gate is required to execute it.

Such situations can be solved by using a `Bridge` gate instead of a `SWAP` gate as shown in Figure 5.8c. Since the distance between the control qubit q_0 and the target qubit q_3 of gate g_3 is 2, we can insert a `Bridge` gate instead. Using a `Bridge` gate allows to execute the `CNOT` gate g_5 without changing the current mapping. Moreover, by using a `Bridge` gate, we only add three `CNOT`s to map the entire circuit, instead of six if only `SWAP` gates were used.

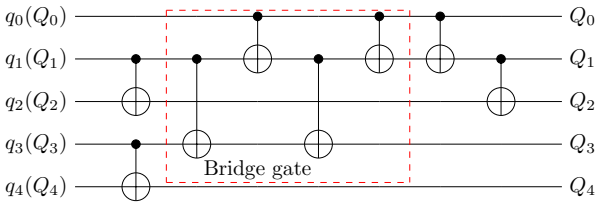
The decision to insert either a `SWAP` or a `Bridge` gate happens only when the cost H of each `SWAP` pair is computed and the best `SWAP` pair has been chosen. Then, two different mappings are considered: π_c, the currently used mapping (i.e., before executing the best `SWAP` pair or equivalently the mapping obtained after inserting a `Bridge` gate), and π_{temp}, the new mapping that would be obtained after inserting the best `SWAP` gate.



(a) *Original quantum circuit that should be compiled in order to be executable on* `ibmq_valencia`.



(b) *Compiled circuit using* `SWAP` *gates.*



(c) *Compiled circuit using* `Bridge` *gate.*

Figure 5.8: *An example of a quantum circuit showing the difference between* `SWAP` *and* `Bridge` *transformations. Because Q_0 and Q_3, the two qubits involved in the gate g_3, have Q_1 as a common neighbour, both transformations are applicable. Comparing the obtained circuits, the* `SWAP` *transformation is more costly due to the fact that the effect of the first* `SWAP` *on the current mapping have to be undone to execute g_5. In this example, the* `Bridge` *transformation avoids 3 additional* `CNOT` *gates.*

The overall effect of the `SWAP` gate on the extended layer E is computed according to Equation (5.6):

$$\text{Effect}\,(\pi_c, \pi_{\text{temp}}) = \sum_{g \in E} D[\pi_c(g.q_1)][\pi_c(g.q_2)] - D[\pi_{\text{temp}}(g.q_1)][\pi_{\text{temp}}(g.q_2)]. \tag{5.6}$$

If the effect of the best `SWAP` gate is negative, this means that the considered `SWAP` pair has an overall negative impact on the extended layer E. In this case, we consider that it is better to keep the current mapping so, if the hardware topology allows it, a `Bridge` gate is inserted instead of a `SWAP` gate.

Runtime analysis

The HA algorithm outperforms SABRE algorithm thanks to several modifications while retaining its low asymptotic complexity. The mapping procedure is separated into two steps: an initialisation step that is independent of the mapped quantum circuit and a mapping step.

The initialisation step computes the distance matrix that is used afterwards in the mapping step. In our algorithm, the distance matrix is computed according to Equation (5.3). Each of S, $\mathcal{E}$ and T appearing in the distance matrix D require to use the Floyd-Warshall algorithm once on the hardware graph G. This means that we need to perform three calls to an algorithm that scales as $O(n^3)$, n being the number of qubits of the targeted quantum chip. Moreover, the weights used by the Floyd-Warshall algorithm for the matrices $\mathcal{E}$ and T should be retrieved online with Qiskit API. This retrieval is an operation that theoretically takes $O(n^2)$ time in the worst case as we need to retrieve `CNOT` error rates and execution time for each link. Note that the current quantum chips only have $O(n)$ links and so the asymptotic complexity of this step is $O(n)$. Overall, the initialisation step is dominated by the cost of applying the Floyd-Warshall algorithm, that takes $O(n^3)$ time.

After the initialisation step, the actual mapping procedure is applied. Let n be the number of qubits, g the number of `CNOT` gates in the mapped quantum circuit and d the diameter of the chip, i.e., the minimum `SWAP` distance between the two farthest qubits on the quantum chip. In

the worst case scenario, all the CNOT gates should be mapped because none of them comply with the hardware topology. Moreover, all the CNOT gates might need up to d SWAPs in order to become executable. Finally, for each SWAP insertion we need to execute the heuristic cost function. This function will need to explore at most n^2 links (in the case of an all-to-all connected chip, this number improves to $O(n)$ on practical quantum chips with a nearest-neighbour connectivity), where exploring one link might take a time of $O(g)$ if all the CNOT gates are included in either F or E. In summary, the mapping step takes $O(g^2dn^2)$ time in the worst case, which can be improved to $O(gn^{2.5})$ under reasonable assumptions (nearest-neighbour chip connectivity, i.e., $d \in O(\sqrt{n})$, and an extended layer E with at most $O(n)$ CNOT gates).

It is important to note that the initialisation step only needs to be repeated when the calibration data change but that requires to recover data from the Internet which can be a slow operation (in the order of several seconds).

5.2.2 Initial mapping

Heuristic-based mapping transition algorithms rely crucially on a good initial mapping to achieve the best results. A well-known algorithm when trying to approximate the global minimum of a scalar function with a discrete search space is simulated annealing. Simulated annealing is a meta-heuristic designed to explore the search space by randomly selecting neighbours of the current state, evaluating them with the provided cost function and evolving in such a way that the algorithm will not be trapped into local minimums. The simulated annealing algorithm is depicted in Algorithm 4.

A modified version of simulated annealing has already been applied in [170] where a repetition parameter R is used to explore several neighbours at each temperature step. The authors consider a simple `get_neighbour` function that modifies randomly the current mapping π to a neighbouring mapping $\pi_{\text{neighbour}}$. However, `get_neighbour` function is limited as it is not aware of the underlying hardware. This means that from the set of mappings generated by this function and evaluated by the simulated annealing procedure, several mappings can be excluded even before evaluating the mapping cost.

We aim to improve the initial mapping generated with the simulated annealing procedure by designing a Hardware-aware Simulated Annealing (HSA) algorithm using a hardware-aware `get_neighbour` method to explore the neighbouring mappings. To explore different mappings, we separate the `get_neighbour` procedure in three algorithms governed by a top execution policy. This top layer policy decides which one of the three algorithms the `get_neighbour` method should execute to obtain a new mapping. The policy we used randomly chooses which algorithm to use from the value of a random number.

The first algorithm, called `shuffle`, does not change the physical qubits involved in the current mapping but changes how they are mapped to logical qubits. The most straightforward algorithm that can be used for this task is a random shuffle: we list the physical qubits involved in the mapping, randomly shuffle them, and obtain a new arbitrary mapping with the same physical qubits.

The second algorithm, `expand`, does not change the mapping between physical qubits and logical ones but replaces one of the physical qubits involved in the mapping by another physical qubit that is not part of the mapping. Instead of a hardware-unaware `expand`, we use an `expand` algorithm that tries to avoid separating the physical qubits in the current mapping into two disconnected groups. Moreover, the algorithm encourages re-arrangement of qubits based on the figure of merit chosen (i.e., final state fidelity, circuit depth, execution time). In this algorithm, we consider that strongly connected qubits have high fidelity. The hardware-aware implementation aims to identify the qubits with the least and most connections. Finally, based on tests related to qubit measurement operations and their communicated error rate, we found them to be a non-negligible source of errors. To account for theses errors, we add a weight to

Algorithm 4: Simulated annealing

input : Initial mapping π_0, Cost function `C`, Neighbour computation function `get_neighbour`, Initial temperature T_{init}, Final temperature T_f, Temperature evolution constant Δ

output: Best initial mapping found π_{opt}

```
1  begin
2      π() ← π_0();
3      π_opt() ← π_0();
4      T() ← T_init();
5      cost() ← C(π);
6      cost_opt() ← cost;
7      while T() ⩾ T_f do
8          π_neighbour() ← get_neighbour(π);
9          cost_neighbour() ← C(π_neighbour);
10         if cost_neighbour() < cost_opt then
11             cost_opt() ← cost_neighbour;
12             π_opt() ← π_neighbour;
13         end
14         if cost_neighbour() < cost then
15             cost() ← cost_neighbour;
16             π() ← π_neighbour;
17         else
18             if rand() < exp((cost()-cost_neighbour())/T) then
19                 cost() ← cost_neighbour;
20                 π() ← π_neighbour;
21             end
22         end
23         T() ← T() × Δ;
24     end
25     return π_opt;
26 end
```

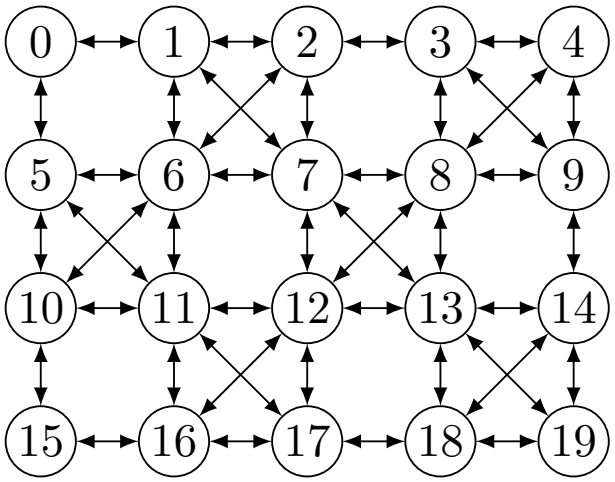


Figure 5.9: *Topology of the* `ibmq_tokyo` *quantum chip.*

each qubit accounting for its measurement error-rate.

The third algorithm used in the `get_neighbour` algorithm is called `reset`. Its purpose is to give the possibility to the simulated annealing algorithm to escape local-minimums. This algorithm is needed because the first two algorithms `shuffle` and `expand` will likely explore only the close neighbourhood of the current mapping and may not be able to escape a local minimum. To avoid being stuck, the `reset` algorithm tries to find a potentially good new initial mapping from a randomly chosen qubit, without considering the previously explored mappings. The algorithm starts with a random qubit and expands the mapping by iteratively weighting all the qubits and adding the best qubit to the new mapping.

5.2.3 Metrics

In order to evaluate the efficiency of HA algorithm over the state of the art, metrics have to used.

The first metric used to compare the algorithms is the success rate of the different mapped quantum circuit on a given hardware. The success rate of a quantum circuit is the frequency of correct outputs over a large number of repetitions. This metric can only be used when the quantum circuit used for the benchmark has a pure quantum state as expected output. We computed the success rate over 8192 repetitions.

The second metric chosen is the additional number of `CNOT` gates. This metric is tightly linked with the total number of `SWAP`/`Bridge` gates inserted and is often a good and easy to compute proxy metric for the success rate due to the high error rate of `CNOT` gates.

The third metric is the total execution time of the circuit. As the execution time of each `CNOT` gate can be extracted from [179], we can estimate the overall execution time of a given circuit. This metric is important for several reasons. First, it shows the ability of the mapping algorithm to schedule gates in parallel when possible and how good is the algorithm at doing this. Secondly, it allows us to have an idea of the importance of decoherence noise in the computed fidelity. For each qubit, the execution time is computed by adding the total execution time of gate operations acting on it. The longest qubit execution time is selected to represent the total execution time of the quantum circuit.

5.3 Evaluation and comparison of the proposed HA Algorithm

5.3.1 Methodology

We collected quantum circuits from previous works that aimed at building a collection of circuits for benchmarking purpose [167, 169, 182]. These quantum circuits include several implementations taken from RevLib [183] as well as implementations of quantum algorithms from a variety of domains including optimisation, simulation, quantum arithmetic, etc. These benchmarks are well known in the community and given as quantum circuits written in the OpenQASM 2.0 language [180].

We chose to evaluate our algorithm on two quantum chips, `ibmq_almaden` and `ibmq_valencia`, available from the IBM Quantum experience website. Additionally, we used the `ibmq_tokyo` chip that is not accessible anymore but has been widely used as benchmark by state-of-art algorithms. `ibmq_almaden` is a 20-qubit quantum chip. Its topology and characteristics are summarised in Figure 5.4 and Table 5.1. `ibmq_valencia` is a 5-qubit chip depicted in Figure 5.1b. `ibmq_tokyo` is a 20-qubit virtual chip depicted in Figure 5.9.

Our algorithm is implemented in Python and the Qiskit version is 0.19.1. To empirically evaluate our algorithm, we use a personal computer equipped with 1 Intel i5-5300U CPU and 8 GB memory. The Operating System is Ubuntu 18.04.

At the time of writing, several algorithms already exist and are available, as discussed in Section 5.1. The SABRE algorithm [169] seems to be among the best performers when using the second metric described in Section 5.2.3, the number of additional gates inserted in order to make the given quantum circuit hardware-compliant. The authors also provide a good initial mapping method. Among the iterative improvements based on the SABRE algorithm, the DL [174] (Dynamic Look-ahead) algorithm seems to be a good candidate for inclusion in our benchmark as it shows an improvement of the number of additional gates required. Moreover, the mapping method presented in [158] uses hardware calibration data to try to find a good mapping and as such is interesting to include in the benchmarked algorithms.

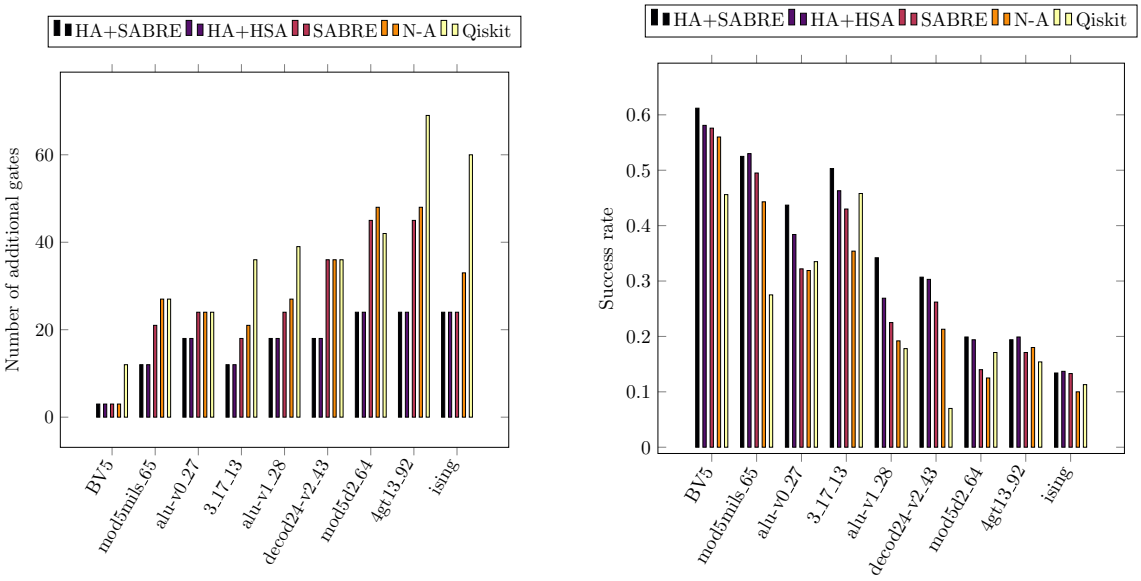
We compare the HA algorithm to all the algorithms cited above. The source code of SABRE has been provided by the authors of the algorithm, and the mapping method presented in [158], called Noise-Adaptive (N-A) Compiler, has been integrated into Qiskit as a transpiler pass. Finally, we also include the default transpiler included in Qiskit as baseline. We execute our HA mapping transition algorithm with two different initial mapping algorithm: SABRE initial mapping algorithm and our Hardware-aware Simulated Annealing (HSA) algorithm.

Summarising, five different algorithms are included in the benchmarks: (1) our HA mapping algorithm with SABRE initial mapping; (2) our HA mapping algorithm with HSA initial mapping; (3) SABRE mapping algorithm with SABRE initial mapping; (4) N-A Compiler and (5) Qiskit transpiler. For a fair comparison, we set the `optimisation_level` parameter of the Qiskit transpiler to `0`[1] and make sure that the circuits obtained from the five methods are all executed with the same calibration data. Moreover, when using the N-A Compiler, the routing method is set to "look-ahead" to make sure that it uses its look-ahead ability.

To evaluate our algorithm with the different initial mapping methods, we allow each of them to call the mapping algorithm at most 100 times. The number of calls to the mapping algorithm is a natural parameter of the simulated annealing-based method, but the SABRE initial mapping method only requires 2 calls. To let the SABRE algorithm take advantage of a larger number of calls, we repeat the algorithm on several random initial mappings until no more calls are allowed and choose the best mapping found. The whole process is repeated 10 times to obtain 10 initial mappings.

We divide benchmarks by size according to their number of gates. We only execute small size

[1]The `optimisation_level` is set to zero to only use the mapping algorithm and avoid other modifications of the quantum circuit that do not happen with the other benchmarked algorithms.



(a) *Average number of additional gates (`CNOT`) over* 10 *repetitions.*

(b) *Average success rate of each quantum circuit over* 10 *repetitions.*

Figure 5.10: *Comparison of the average number of additional gates and success rate on `ibmq_valencia`. HA algorithm has been used with* $\alpha_1 = 0.5$, $\alpha_2 = 0.5$ *and* $\alpha_3 = 0$.

benchmarks on real quantum hardware, because the larger benchmarks results are exhibiting too much noise to obtain any meaningful results. Moreover, the initial mapping generation process described above is applied on small and medium sized benchmarks. Large benchmarks suffer from long run time, so we generate 10 initial random mappings and use them with different algorithms. When using `ibmq_tokyo` virtual chip, we select the best results out of 5 attempts, which is a similar approach applied in the SABRE and DL research papers.

When testing the HSA algorithm we used the random policy described in Section 5.2.2 to choose which one of the three subroutines to execute. The `shuffle` procedure is executed with a probability of 0.9, the `expand` algorithm is chosen with a probability 0.08 and the `reset` procedure is executed when the two previous algorithms are not used (i.e., with a probability of 0.02).

First, we compare the number of additional gates and success rate. The weight parameter α_1 associated with the `SWAP` matrix S is set to 0.5, the weight parameter α_2 associated with the `CNOT` error matrix $\mathcal{E}$ is set to 0.5 and α_3, the weight associated with the `CNOT` execution time matrix T is set to 0.

In a second time, we compare the number of additional gates and total execution time. Weight are set as $(\alpha_1, \alpha_2, \alpha_3) = (0.5, 0, 0.5)$.

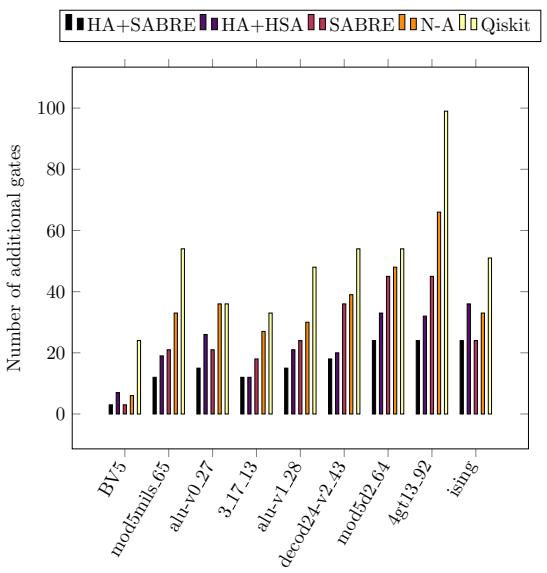
Finally, we compare the number of additional gates for circuits that are not executable on the real quantum device with the weights $(\alpha_1, \alpha_2, \alpha_3) = (1, 0, 0)$.

For these three scenarii, the weight parameter W in the cost function Equation (5.5) is set to 0.5 and the extended layer size is set to $|E| = 20$.

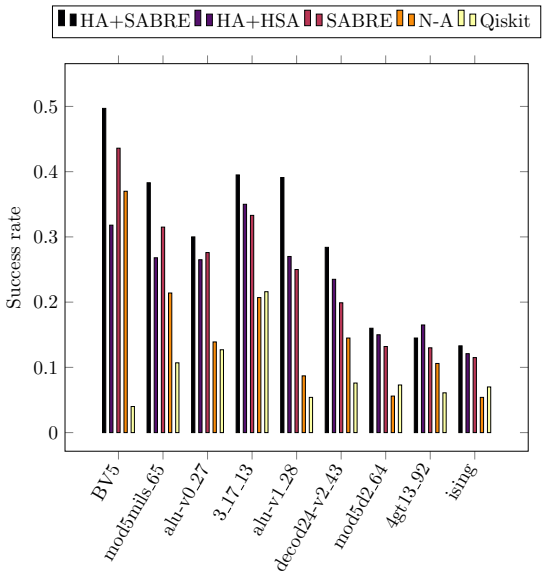
5.3.2 Experimental results

As stated in Section 5.3.1, we compiled several circuits with the 5 mapping algorithms to benchmark. Both `ibmq_valencia` (see Figure 5.10) and `ibmq_almaden` (see Figure 5.11) have been used as target backend.

We compare both the average number of additional gates (see Figure 5.10a and Figure 5.11a) and average success rate (see Figure 5.10b and Figure 5.11b) among the 10 initial mappings for



(a) *Average number of additional gates (`CNOT`) over* 10 *repetitions.*



(b) *Average success rate of each quantum circuit over* 10 *repetitions.*

Figure 5.11: *Comparison of the average number of additional gates and success rate on `ibmq_almaden`. HA algorithm has been used with* $\alpha_1 = 0.5$, $\alpha_2 = 0.5$ *and* $\alpha_3 = 0$.

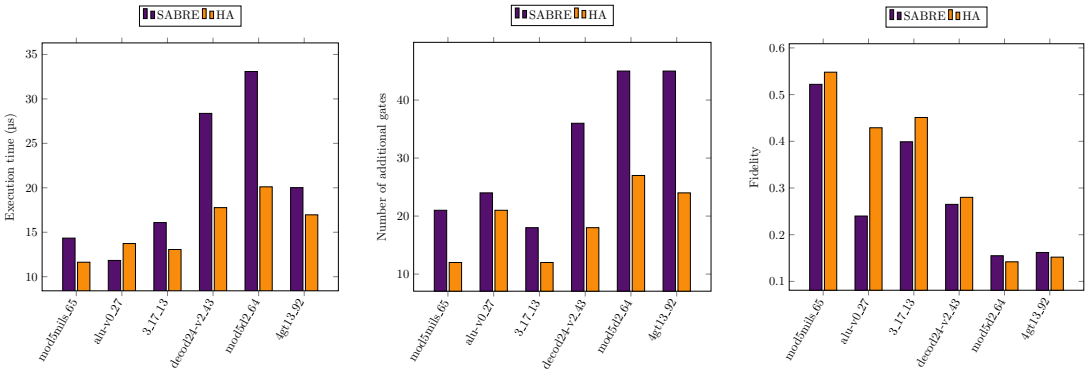
each of the five benchmarked methods. The complete experimental results are listed in Table 5.4 and Table 5.5.

The `Qiskit` default qubit mapping algorithm is nearly always the worst one in terms of additional gates, which translates in most of the cases to the worst output state fidelity. Although `N-A` compiler takes into account the calibration data and uses a look-ahead strategy, results show that it does not outperform the SABRE mapping algorithm with SABRE initial mapping (labelled as `SABRE` in the plots). Our HA mapping algorithm with SABRE initial mapping (labelled as `HA+SABRE` in the plots) seems to be the best combination as, on average, it achieves the best success rate. Moreover, `HA+SABRE` gives the minimum number of additional gates. HA mapping algorithm with HSA initial mapping (labelled as `HA+HSA` in the plots) is also good, but its results are less consistent than `HA+SABRE` due to its random nature. Although, in many test cases, it outperforms `SABRE`.

We also tried to map and execute the `qft_10` circuit which implements a Quantum Fourier Transform (QFT) on 10 qubits. We found that its success rate is less than 0.01 for all the methods tested in the benchmark. Because the base success rate is too low to perform a meaningful comparison, we only compare the number of additional gates as summarised in Table 5.2 and Table 5.3 for quantum circuits with a medium-to-large number of gates.

Figure 5.12 shows the result of comparing the execution times, number of additional gates and success rates of the HA algorithm with SABRE algorithm on `ibmq_valencia`. The execution time is reduced by 19% on average. Even though the weight parameter α_2 of `CNOT` error matrix $\mathcal{E}$ is set to 0, the success rate is improved by 8%. The number of additional gates is reduced by 38%.

Table 5.2 lists the result of the number of additional gates on `ibmq_almaden`. Using the selection of `SWAP` and `Bridge` gate, the HA algorithm can outperform SABRE on circuits with different sizes. For medium circuits, HA and SABRE give similar results, with HA improving the result from SABRE for only one circuit among the eight circuits tested. For large circuits, HA outperforms SABRE and consistently reduces the number of additional gates by 28% on average. Table 5.3 shows the number of additional gates on `ibmq_tokyo` when comparing the HA algorithm with SABRE and DL. DL outperforms SABRE on nearly all the benchmarked



(a) *Execution time* **(b)** *Number of additional gates* **(c)** *Success rate*

Figure 5.12: *Comparison of execution time, number of additional gates and fidelity on* `ibmq_valencia`*. HA has been used with* $\alpha_1 = 0.5$*,* $\alpha_2 = 0$ *and* $\alpha_3 = 0.5$*.*

quantum circuits and the HA algorithm is able to further reduce the number of additional gates by 14% on average.

The SABRE and DL authors only provide the runtime of their algorithm on `ibmq_tokyo`. As such, a comparison of of the three algorithms runtime on this specific chip topology is shown in Table 5.3. Note that DL is written in C++ and tested on a normal personal computer. SABRE is written in Python and tested on a server with 2 Intel Xeon E5-2680 CPUs (48 logical cores) and 378GB memory. Since there is an intrinsic speed difference between C++ and Python as well as the different devices used, the runtime data in this table are for reference rather than for comparison.

5.4 Conclusion

In this chapter we presented a classical algorithm that can be used to adapt quantum circuits to a specific hardware topology by taking into account the calibrations of the chip. We benchmarked the proposed algorithm against several algorithms that were considered the state-of-the-art at the time the research was performed and showed that our algorithm is able to improve very consistently over these algorithms, both with respect to the number of additional quantum gates and to the success rate of the compiled quantum circuits.

Table 5.2: *Number of additional gates on* `ibmq_almaden` *for large circuits. HA has been used with* $\alpha_1 = 1$, $\alpha_2 = 0$ *and* $\alpha_3 = 0$. $\mathbf{n}$*: number of qubits.* $\mathbf{g_{all}}$*: total number of gates.* $\mathbf{g}$*: average number of additional gates.* $\mathbf{g_{min}}$*: minimum number of additional gates.* $\mathbf{t}$*: runtime in seconds.* $\mathbf{\Delta g}$*: comparison of average number of additional gates between HA and SABRE.* $\mathbf{\Delta g_{min}}$*: comparison of minimum number of additional gates between HA and SABRE.*

Original Circuit				SABRE		HA			Comparison	
type	name	n	g_{all}	g	g_{min}	g	g_{min}	t	$\Delta g\%$	$\Delta g_{min}\%$
medium	qaoa	6	270	30	27	30	27	0.008	0	0
medium	ising_model_10	10	480	0	0	0	0	0.02	0	0
medium	ising_model_13	13	633	0	0	0	0	0.03	0	0
medium	ising_model_16	16	786	3	0	9	0	0.10	-200	0
medium	qft_10	10	200	93	81	66	42	0.04	29	48.1
medium	qft_13	13	403	192	177	195	171	0.07	-1.6	3.4
medium	qft_16	16	512	425	372	450	375	0.24	-5.9	-0.8
large	adr4_197	13	3439	2973	2856	2136	2004	2.13	28.2	29.8
large	radd_250	13	3213	2742	2655	2040	1926	1.62	25.6	27.5
large	z4_268	11	3073	2628	2559	1872	1815	1.44	28.8	29.1
large	sym6_145	14	3888	3024	2982	2022	1965	2.18	33.1	34.1
large	misex1_241	15	4813	3999	3831	2892	2630	3.04	27.7	31.3
large	rd73_252	10	5321	4539	4428	3261	3090	3.73	28.2	30.2
large	cycle10_2_110	12	6050	5127	5043	3795	3576	4.87	26	29.1
large	square_root_7	15	7630	6477	6324	4851	4707	7.00	25.1	25.6
large	sqn_258	10	10223	8679	8580	6012	5736	13.92	30.7	33.1
large	rd84_253	12	13658	11889	11673	8721	8574	24.54	26.6	26.5
large	co14_215	15	17936	16710	16368	13071	12426	37.81	21.8	24.1
large	sym9_193	10	34881	30558	30027	21900	21168	160.19	28.3	29.5
large	9symml_195	11	34881	30471	30129	21949	21168	151.84	28	29.7

Table 5.3: *Number of additional gates on* `ibmq_tokyo` *for large circuits. HA has been used with* $\alpha_1 = 1$, $\alpha_2 = 0$ *and* $\alpha_3 = 0$. $\mathbf{n}$*: number of qubits.* $\mathbf{g_{all}}$*: total number of gates.* $\mathbf{g}$*: minimum number of additional gates.* $\mathbf{t}$*: runtime in seconds.* $\mathbf{\Delta g}$*: comparison of minimum number of additional gates between HA and DL.*

Original Circuit				SABRE		DL		HA		Comparison
type	name	n	g_{all}	g	t	g	t	g	t	$\Delta g\%$
medium	ising_model_10	10	480	0	0.004	0	0	0	0.005	0
medium	ising_model_13	13	633	0	0.007	0	0	0	0.01	0
medium	ising_model_16	16	786	0	0.01	0	0	0	0.02	0
medium	qft_10	10	200	54	0.103	39	0.015	36	0.015	7.7
medium	qft_13	13	403	93	0.036	96	0.031	78	0.043	18.8
medium	qft_16	16	512	186	0.084	192	0.062	174	0.09	9.4
large	adr4_197	13	3439	1614	0.49	1224	0.218	882	1.41	27.9
large	radd_250	13	3213	1275	0.48	1047	0.186	840	1.24	19.8
large	z4_268	11	3073	1365	0.44	855	0.202	801	1.13	6.3
large	sym6_145	14	3888	1272	0.56	1017	0.202	786	1.71	22.7
large	misex1_241	15	4813	1251	0.89	1098	0.249	942	2.57	14.2
large	rd73_252	10	5321	2133	0.94	2193	0.343	1635	3.19	25.4
large	cycle10_2_110	12	6050	2622	1.35	1968	0.348	1719	4.02	12.7
large	square_root_7	15	7630	2598	1.5	1788	0.406	828	5.66	53.7
large	sqn_258	10	10223	4344	3.52	3057	0.563	2712	11.7	11.3
large	rd84_253	12	13658	6147	5.39	5697	0.892	3843	21.8	32.5
large	co14_215	15	17936	8982	9.51	5061	1.062	6429	36	-27
large	sym9_193	10	34881	16653	30.17	13746	2.091	11553	138.3	16

Table 5.4: *Comparison of number of additional gates and fidelity on* `ibmq_valencia`*. HA has been used with* $\alpha_1 = 0.5$, $\alpha_2 = 0.5$ *and* $\alpha_3 = 0$. $\mathbf{n}$*: number of qubits.* $\mathbf{g_{all}}$*: total number of gates.* $\mathbf{g}$*: average number of additional gates.* $\mathbf{g_{min}}$*: minimum number of additional gates.* $\mathbf{S}$*: mean of success rate.* $\mathbf{S_{max}}$*: maximum of success rate.* $\mathbf{\Delta g}$*: comparison of average number of additional gates between HA+SABRE and SABRE.* $\mathbf{\Delta g_{min}}$*: comparison of minimum number of additional gates between HA+SABRE and SABRE.* $\mathbf{\Delta S}$*: comparison of mean of success rate between HA+SABRE and SABRE.* $\mathbf{\Delta S_{max}}$*: comparison of maximum of success rate between HA+SABRE and SABRE.* $\mathbf{t}$*: runtime of HA+SABRE in seconds.*

Original Circuit			SABRE				HA + SABRE					HA + HSA				Qiskit		N-A		Comparison			
name	n	g_{all}	g	g_{min}	S	S_{max}	g	g_{min}	S	S_{max}	t	g	g_{min}	S	S_{max}	g	S	g	S	$\Delta g\%$	$\Delta g_{min}\%$	$\Delta S\%$	$\Delta S_{max}\%$
BV5	5	15	3	3	0.576	0.639	3	3	0.612	0.639	0	3	3	0.581	0.63	12	0.456	3	0.56	0	0	6.3	0
mod5mils_65	5	35	21	21	0.495	0.515	12	12	0.525	0.559	0.003	12	12	0.53	0.559	27	0.275	27	0.443	42.9	42.9	6.1	8.5
alu-v0_27	5	36	24	24	0.322	0.329	18	18	0.437	0.437	0.002	18	18	0.384	0.431	24	0.335	24	0.319	25	25	35.7	32.8
3_17_13	3	36	18	18	0.43	0.476	12	12	0.503	0.546	0.004	12	12	0.463	0.542	36	0.458	21	0.354	33.3	33.3	17	14.7
alu-v1_28	5	37	24	24	0.225	0.233	18	18	0.342	0.384	0.004	18	18	0.269	0.384	39	0.178	27	0.192	25	25	52	64.8
decod24-v2_43	4	52	36	36	0.262	0.396	18	18	0.307	0.37	0.004	18	18	0.303	0.372	36	0.07	36	0.213	50	50	17.2	-6.6
mod5d2_64	5	53	45	45	0.14	0.208	24	24	0.199	0.207	0.005	24	24	0.194	0.207	42	0.171	48	0.125	46.7	46.7	42.1	-0.4
4gt13_92	5	66	45	45	0.171	0.191	24	24	0.194	0.206	0.006	24	24	0.199	0.22	69	0.154	48	0.18	46.7	46.7	13.5	7.9
ising	5	90	24	24	0.133	0.145	24	24	0.134	0.141	0.007	24	24	0.137	0.143	60	0.113	33	0.1	0	0	0.8	-2.8

Table 5.5: *Comparison of number of additional gates and fidelity on* `ibmq_almaden`*. HA algorithm has been used with* $\alpha_1 = 0.5$*,* $\alpha_2 = 0.5$ *and* $\alpha_3 = 0$*.* $\mathbf{n}$*: number of qubits.* $\mathbf{g_{all}}$*: total number of gates.* $\mathbf{g}$*: average number of additional gates.* $\mathbf{g_{min}}$*: minimum number of additional gates.* $\mathbf{S}$*: mean of success rate.* $\mathbf{S_{max}}$*: maximum of success rate.* $\mathbf{\Delta g}$*: comparison of average number of additional gates between HA+SABRE and SABRE.* $\mathbf{\Delta g_{min}}$*: comparison of minimum number of additional gates between HA+SABRE and SABRE.* $\mathbf{\Delta S}$*: comparison of mean of success rate between HA+SABRE and SABRE.* $\mathbf{\Delta S_{max}}$*: comparison of maximum of success rate between HA+SABRE and SABRE.* $\mathbf{t}$*: runtime of HA+SABRE in seconds.*

Original Circuit			SABRE				HA + SABRE					HA + HSA				Qiskit		N-A		Comparison			
name	n	g_{all}	g	g_{min}	S	S_{max}	g	g_{min}	S	S_{max}	t	g	g_{min}	S	S_{max}	g	S	g	S	$\Delta g\%$	$\Delta g_{min}\%$	$\Delta S\%$	$\Delta S_{max}\%$
BV5	5	15	3	3	0.436	0.624	3	3	0.497	0.651	0.002	7	6	0.318	0.508	24	0.04	6	0.37	0	0	14	4.3
mod5mils_65	5	35	21	21	0.315	0.47	12	12	0.383	0.481	0.003	19	15	0.268	0.439	54	0.107	33	0.214	42.9	42.9	21.6	2.3
alu-v0_27	5	36	21	21	0.276	0.413	15	15	0.3	0.483	0.002	26	19	0.265	0.408	36	0.127	36	0.139	28.6	28.6	8.7	16.9
3_17_13	3	36	18	18	0.333	0.469	12	12	0.395	0.519	0.002	12	12	0.35	0.502	33	0.216	27	0.207	33.3	33.3	18.6	10.7
alu-v1_28	5	37	24	24	0.25	0.359	15	15	0.391	0.478	0.002	21	21	0.27	0.408	48	0.054	30	0.087	37.5	37.5	56.4	33.1
decod24-v2_43	4	52	36	36	0.199	0.334	18	18	0.284	0.401	0.006	20	18	0.235	0.387	54	0.076	39	0.145	50	50	42.7	20.1
mod5d2_64	5	53	45	45	0.132	0.198	24	24	0.16	0.266	0.003	33	33	0.15	0.263	54	0.073	48	0.056	46.7	46.7	21.2	34.3
4gt13_92	5	66	45	45	0.13	0.249	24	24	0.145	0.312	0.007	32	27	0.165	0.347	99	0.061	66	0.106	46.7	46.7	11.5	25.3
ising	5	90	24	24	0.115	0.177	24	24	0.133	0.191	0.01	36	30	0.121	0.235	51	0.07	33	0.054	0	0	15.7	7.9

Chapter

6 Variational quantum linear solver

Following the results obtained in Chapters 3 and 4 and the research path of Chapter 5, an interesting approach to check if scientific computing problem solvers can be of interest on NISQ chips is to actually implement an algorithm that is tailored to noisy and small quantum chips. In this chapter we perform a study of a variational algorithm that can be used to solve linear systems of equations: the Variational Quantum Linear System algorithm introduced in [48]. The study performed considers several linear systems of interest and executions on NISQ chips from IBM.

Contents

6.1 Introduction

The quantum computing field has been evolving at an increasing rate in the past few years and is currently gaining more traction. Several quantum chips, the underlying hardware that enable researchers and companies to run quantum algorithms, have been announced by different research teams. The error rates and number of qubits provided by these chips greatly improved, with quantum hardware that have up to 127 qubits in the end of 2021 [115]. This increase in qubit number also comes with steady improvements of the qubits quality as the techniques around chip engineering, qubit control or software are improving.

This improvement of the quantum computing hardware is to be compared with the advances in the very active field of quantum algorithms. Quantum algorithms applied to linear algebra, and more particularly solvers for linear system of equations, are of particular interest as they

promise to solve an ubiquitous problem in the world of scientific computing: finding efficiently the solution to a given linear system of equation.

The HHL algorithm [13] was the first quantum algorithm devised to solve the quantum alternative of the linear system of equation problem. However, a careful study of the theoretical requirements [38] and practical resources [77] to be able to use it on a given linear system raise the concern that the HHL algorithm might not be interesting in practical use-cases. Additionally, due to the large number of gates required to execute the algorithm, the HHL algorithm is not suitable for NISQ hardware (see Definition 11).

The issue raised by the high error rates characterising NISQ hardware is major and impact the whole field of quantum computing. As such, a large variety of approaches have been explored to mitigate the negative impact of noise on computations. The following sections present two of the most promising approaches.

6.1.1 Quantum error correction

One of the first approaches that has been theoretically explored and is based on classical computing ideas is error correction. The goal of error correction is to embrace the fact that the underlying hardware is faulty and to reliably correct the errors that might happen. Most of the classical error correction techniques are based on adding redundant information in order to detect and/or correct any error that might happen during the communication or computation. This technique of encoding redundantly an information before performing a potentially faulty operation to "protect" it from errors is easily applicable to classical bits that can be read and copied at will, but quantum bits and quantum computing in general are subject to what is called the no-cloning theorem that forbids the direct translation of classical error-correction techniques to quantum computers.

Theorem 2 (No-cloning theorem)**.** Let H be a Hilbert space, $|\phi\rangle \otimes |\psi\rangle \in H \otimes H$ two quantum states from the same state space H. Then, there is no unitary U acting on $H \otimes H$ such as for all $|\phi\rangle$ and $|\psi\rangle$ in H

$$U\left(|\phi\rangle \otimes |\psi\rangle\right) = |\phi\rangle \otimes |\phi\rangle \tag{6.1}$$

up to a global phase $e^{i\alpha}$.

The no-cloning theorem forbids the exact cloning of unknown quantum states, which in turns forbids the adaptation of certain classical error correction schemes to quantum computing.

Rather than copying quantum states when needed, quantum error-correction schemes [104] encode a *logical qubit*, i.e., a perfect, noise-free qubit, using several *physical qubits*, i.e., noisy qubits. A crucial point of quantum error-correction schemes is their *threshold* ϵ, i.e., the maximum error-rate the physical qubits can experience before the error-correcting scheme starts failing [104] and stops correcting all errors. The order of magnitude of the threshold ϵ depends a lot on the error-correction scheme used, but it is typically in the order of magnitude of 10^{-3} to 10^{-2} [184, Table 1]. For most quantum error correction schemes, the threshold ϵ is the *maximum* error-rate at which they can theoretically operate, but the number of physical qubits that will be used to encode one logical qubit close to the threshold is very high and impractical for current quantum systems. For example, [104] shows with a numerical study that with an error rate of $p = 10^{-3}$ per elementary gate, implementing a logical qubit with a probability of error below 10^{-15} requires more than 10^4 physical qubits.

6.1.2 Quantum error mitigation

Because *correcting* quantum errors is not currently feasible due to the error-rates of current quantum hardware being too high and the large number of qubits required to encode one logical qubit, another approach to lower down the negative impact of hardware errors is to only try to

mitigate them, i.e., try to reduce to a minimum their effect on the result of a quantum computation knowing that some errors will still pass through in the final result. There are a variety of mitigation methods, some of them only considering the quantum computation performed while others are also using classical processing to mitigate the results obtained from the noisy quantum chip by using pre- and post-processing.

Within the mitigation methods that do not require any classical post-processing, one can cite dynamical decoupling that insert specific operations when a qubit is idle to avoid the effect of decoherence [185–190], quantum circuit optimisation [132–136] that modify the quantum computation performed in order to lower down the effect of potential errors or their rate of apparition or pulse-shape optimisation [137–139] that changes the low-level implementation of quantum gates to try to lower down their error rate.

Hybrid quantum-classical methods are also well represented with, for example, probabilistic error cancellation [191–193], Clifford data regression [194, 195], measurement error mitigation [196] or zero-noise extrapolation [191, 197, 198].

6.2 Variational quantum algorithms

While quantum error correction and quantum error mitigation are mostly independent of the quantum computation performed (i.e., they can be used for any quantum computation), the use of quantum variational algorithms is a paradigm shift that changes the underlying algorithms used to solve a given problem with the goal of having a NISQ-compatible algorithm from the ground up.

6.2.1 General idea

Variational algorithms are a specific type of hybrid quantum-classical algorithms that, by following a specific template first introduced in [199], aim at being NISQ-friendly. Creating a variational algorithm for a given problem is a matter of carefully encoding the solution of the problem into the minimum of a cost function that can be written as

$$C\left(\vec{\alpha}\right) = \left\langle \psi\left(\vec{\alpha}\right)\right| H \left|\psi\left(\vec{\alpha}\right)\right\rangle \tag{6.2}$$

where $\left|\psi\left(\vec{\alpha}\right)\right\rangle$ is a classically-parameterised quantum state that is prepared with a classically-parameterised unitary $U\left(\vec{\alpha}\right)$ (also called *ansatz*, see Section 6.2.2) and H is a Hamiltonian matrix whose ground-state encodes the solution to the problem at hand.

The particular cost function expression shown in Equation (6.2) is motivated by the fact that Postulate 4 and the special case of Projection-Valued Measurement (PVM) used when the quantum system measured is considered isolated match exactly the cost function, meaning that the cost function C can be computed on a quantum computer by measuring a specific quantum state $\left|\psi\left(\vec{\alpha}\right)\right\rangle$ according to the measurement represented by the hermitian matrix H.

Most of the time the measurement matrix H will not be trivially implementable on a quantum computer, leading to a measurement that will be expensive to implement in practice. One of the key realisations of variational algorithms is that under specific conditions the computation of C can be split into several simpler and independent computations. Such conditions can for example be that the matrix H should be a linear combination of measurement matrices:

$$H = \sum_i \beta_i H_i \tag{6.3}$$

where each H_i represent an easily implementable measurement on a quantum computer. Examples of "simple to implement" measurement matrices include the H_i that can be written as

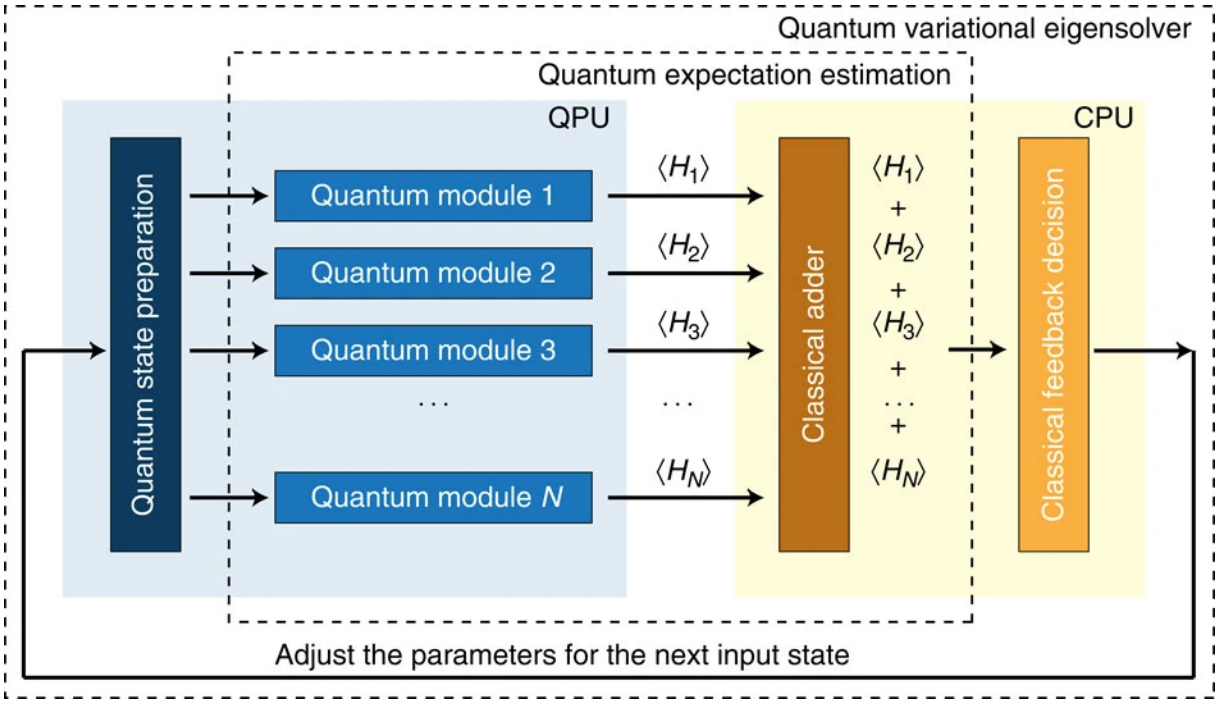


Figure 6.1: *Schematic representation of the computations performed by the variational quantum eigensolver. Any variational quantum algorithm follow the same principles with a problem-dependent Hamiltonian H and decomposition $H = \sum_i \alpha_i H_i$. Graph obtained from [199].*

the tensor product of Pauli matrices $H_i = \bigotimes_{j=0}^{n-1} \sigma_j$ where $\sigma_j \in \{I, \sigma_X, \sigma_Y, \sigma_Z\}$. Under this condition, the cost function becomes

$$C(\vec{\alpha}) = \sum_i \beta_i \langle \psi(\vec{\alpha}) | H_i | \psi(\vec{\alpha}) \rangle = \sum_i \beta_i C_i(\vec{\alpha}) \tag{6.4}$$

where each C_i can be computed efficiently and independently on a quantum computer and a classical computer computes the weighted sum (see Figure 6.1).

Once function C has been devised and can be computed using a quantum computer, its cost is optimised using classical optimisers, either using gradient-free methods or, if the gradient or higher-order derivatives of C can be computed efficiently, using more sophisticated optimisation methods using this additional information.

6.2.2 Ansatz

As briefly explained in Section 6.2.1, a classically parameterised quantum state $|\psi(\vec{\alpha})\rangle$ is prepared by applying the classically parameterised unitary $U(\vec{\alpha})$. The unitary $U(\vec{\alpha})$ is commonly called an *ansatz*. Using the right ansatz (or parameterised quantum circuit) for a given problem instance is a crucial problem that may have deep consequences on the convergence of the variational algorithm used.

Choosing the best parameterised quantum circuit for a given problem instance is an open question and will depends on several factors such as the problem at hand, the desired precision, the hardware noise level, etc. In general, ansatzes can be grouped into two groups, depending on whether the ansatz is problem-inspired or problem-agnostic.

Problem-inspired ansatzes use prior knowledge about the problem at hand to guide the exploration of trial states and avoid as many "invalid" states (with respect to the problem constraints) as possible. The Unitary Coupled Cluster (UCC) ansatz is a good example of such a problem-inspired ansatz [199].

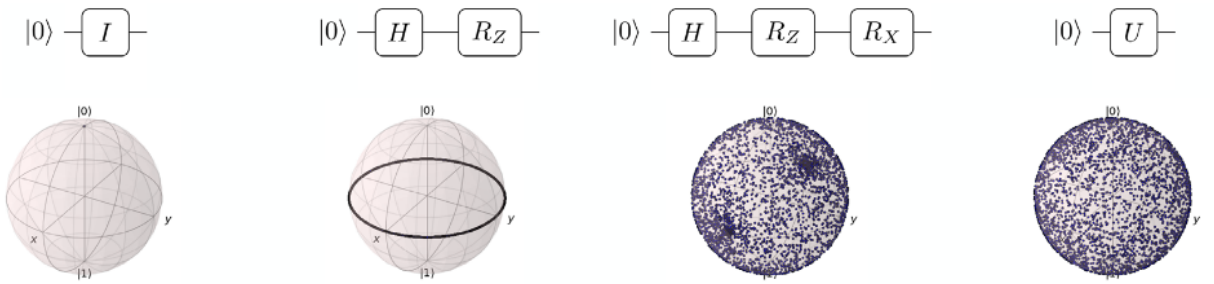


Figure 6.2: *Illustration of ansatz expressibility for a* 1*-qubit ansatz. As can be seen, the two left-most circuits have a low expressibility as they are not able to cover the space of all the* 1*-qubit states. The ansatz composed of successive applications of the* H*,* $R_z(\theta_1)$ *and* $R_x(\theta_2)$ *is able to cover the whole* 1*-qubit state space, but does not cover it uniformly. Finally, the right-most circuit, composed of a random unitary, is the definition of the maximally expressive ansatz as it is able to cover the whole space uniformly. Figure obtained from [200].*

But not all problems have a structure that imposes specific constraints on the possible solutions. Solving general systems of linear equations is exactly one of such problem as the normalised solution vector $|x\rangle$ can be *any* efficiently preparable quantum state. This claim is easily proven by taking the trivial linear system $A|x\rangle = |b\rangle$ with $A = I$, the identity matrix. In this case, the solution of the linear system is $|b\rangle$, which can be any efficiently preparable quantum state. Rephrasing, it is not possible to devise a problem-inspired ansatz for all the problems, which calls for problem-agnostic ansatzs.

With such a problem, the ansatzs chosen should be as generic as possible. The capability of a given ansatz acting on n qubits to cover uniformly the space of all the n-qubit quantum states is quantified by a quantity known as its expressibility [200] and illustrated in Figure 6.2. Expressibility of an ansatz $A\left(\vec{\theta}\right)$ is often computed as the deviation between the ensemble of Haar-random state and the distribution of quantum states that can be generated by A when $\vec{\theta}$ is sampled uniformly.

Hardware-efficient ansatzs is a class of problem-agnostic ansatzs that try to be as close as possible to the underlying hardware to reduce their execution time. These ansatzs often adapt their entangling gate to the hardware topology to avoid any `SWAP` gate insertion during the qubit mapping phase and the associated execution time increase due to the `SWAP` gate execution. Figure 6.3 shows a possible hardware-efficient ansatzs using the controlled-X as entangling gate on a random topology using 4 qubits.

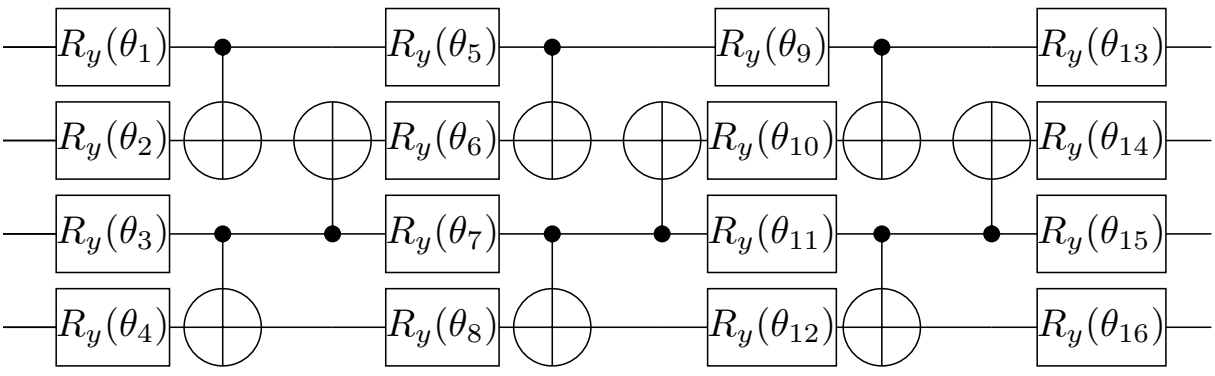


Figure 6.3: *One example of the hardware-efficient ansatz used in this study. Represented here is a depth of* 3 *with only* R_y *rotations as* 1*-qubit gates. The controlled* X *gates are used to generate entanglement and are applied following the native connectivity of the targeted hardware (here randomly selected).*

Note that the hardware-efficient ansatz depicted in Figure 6.3 is slightly "problem-inspired"

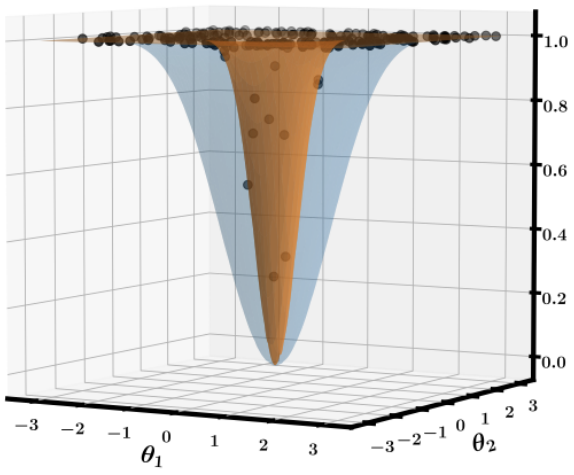


Figure 6.4: *Illustration of the effect of Barren plateau by using the (global) cost function* $C_n(\vec{\theta}) = 1 - \prod_{i=1}^{n} \cos^2(\theta_i)$ *for* $n = 4$ *(blue) and* $n = 24$ *(orange) and fixing all variables except* θ_1 *and* θ_2*. The cost function* C_n *ends up being constant nearly everywhere for large* n*. Image obtained from [201].*

and is a good illustration of optimisations that rely on knowledge about the problem that will be solved. In this case, the ansatz in Figure 6.3 does not, by design, prepare quantum states with complex amplitudes. This is due to the fact that the controlled-X quantum gate unitary matrix (shown in Equation (1.18)) and the R_y gate

$$R_y(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix} \tag{6.5}$$

only have real coefficients and that the initial quantum state, supposed to be $\bigotimes_i |0\rangle$, does not contain any complex amplitude either.

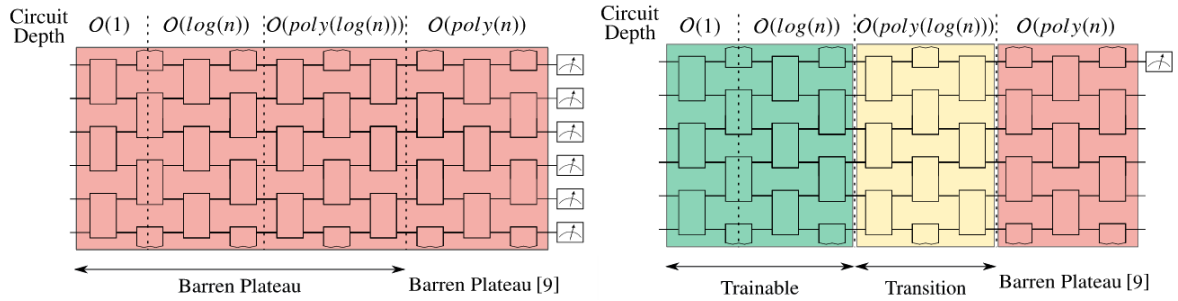
In order to limit the number of varying parameters for the upcoming analysis and because our ultimate goal is to understand how the VQLS algorithm can behave on real hardware, we chose to only use the hardware-efficient ansatz presented in Figure 6.3.

6.2.3 Barren plateaus

The phenomenon of Barren plateau is a major limitation of variational quantum algorithm and the reason why a careful study of the cost function and ansatz used is necessary to ensure an appropriate convergence rate. It essentially states that cost functions used within the framework of variational quantum algorithms may have derivatives that, on average, vanish exponentially with the system size (i.e., the number of qubits n). From the cost function point of view, this implies that the optimisation landscape will appear flat nearly everywhere except on exponentially small regions around the minima as illustrated in Figure 6.4.

Due to the Barren plateau phenomenon, the cost of estimating expectations values with a sufficient precision to be able to use information about their difference for a local perturbation of the inputs grows exponentially. This exponential cost obviously impede the efficiency of gradient-based optimisation method that rely on an estimation of the gradient, but also impacts gradient-free optimisation as shown in [202].

It has been shown in [201] that local cost functions that only rely on the computation of expectations values over a constant number of qubits are not as impacted as global cost functions. The authors also show that the ansatz depth plays a crucial role in the apparition of Barren plateau and summarise their findings in a graphical representation that is duplicated in Figure 6.5.



(a) *Apparition of the Barren plateau phenomenon for global cost function.*

(b) *Apparition of the Barren plateau phenomenon for local cost function.*

Figure 6.5: *Illustration of the Barren plateau apparition depending on the type of cost function used (global or local) and the ansatz depth. Global cost functions are bound to eventually experience a Barren plateau phenomenon whereas local cost functions might, under some conditions on the ansatz depth, be free from any Barren plateau. Image obtained from [201].*

6.3 The Variational Quantum Linear Solver

Due to the nature of NISQ hardware, quantum algorithms should take into account the high error-rate associated with each gate execution. This means that in order to avoid being impeded by noise, quantum implementations should refrain from executing large circuits to the quantum chips. As shown in Section 6.2, , variational quantum algorithms offer the advantage of being able to decompose larger problems into a series of smaller ones, thus limiting circuit size and gate requirements making them ideal candidates for NISQ hardware.

6.3.1 Cost functions

The Variational Quantum Linear Solver (VQLS) algorithm [48] is a variational algorithm that has been introduced to solve linear systems of equations. For a linear system $Ax = b$ with a given matrix A and right-hand side b, the VQLS algorithm may use one out of the two cost functions introduced in [48] and repeated below.

The first cost function introduced in [48] is the *global* cost function C_G described in Equations (6.6) and (6.7):

$$C_G(\vec{\alpha}) = \langle x(\vec{\alpha})| H_G |x(\vec{\alpha})\rangle \tag{6.6}$$

where

$$H_G = A^\dagger \left(I - |b\rangle \langle b|\right) A. \tag{6.7}$$

The cost function C_G has the advantage of being simple to understand: the matrix A is "applied" to the trial quantum state that is supposed to encode the solution. This operation results in a *vector* that does not necessarily represent a valid quantum state since A is not restricted to be a unitary matrix but can be any $2^n \times 2^n$ matrix. The *vector* $\vec{v} = A|\psi(\vec{\alpha})\rangle$ is compared to the right-hand side b by estimating the "proportion" of $|\vec{v}\rangle = \frac{\vec{v}}{||\vec{v}||}$ that does not overlap with $|b\rangle$. The normalised version of the global cost function C_G illustrates very nicely this interpretation:

$$\hat{C_G}(\vec{\alpha}) = \frac{C_G(\vec{\alpha})}{\langle x(\vec{\alpha})| A^\dagger A |x(\vec{\alpha})\rangle} = 1 - |\langle \vec{v}|b\rangle|^2. \tag{6.8}$$

But the global cost function $C_G(\alpha)$ has a crucial downside: it is vulnerable to the phenomenon called "Barren plateau" explained in Section 6.2.3. In order to avoid the Barren plateau issue, a *local* cost function that is less vulnerable has also been devised in [48] and is presented in Equations (6.9) and (6.10):

$$C_L(\vec{\alpha}) = \langle x(\vec{\alpha})| H_L |x(\vec{\alpha})\rangle \tag{6.9}$$

Linear system name	Decomposition of A	Initialisation matrix V	Equation
`identity`	$\bigotimes_i I_i$	$\bigotimes_i H_i$	Equation (6.13)
`trivial`	$\bigotimes_i X_i$	$\bigotimes_i H_i$	Equation (6.14)
`varying_condition`	$A(k)$	$\bigotimes_i H_i$	Equation (6.18)
`heat_opti`	B	$\bigotimes_i H_i$	Equation (6.26)

Table 6.1: *List of linear systems studied in this chapter.*

where

$$H_L = A^\dagger V \left(I - \frac{1}{n} \sum_{j=1}^{n} |0_j\rangle \langle 0_j| \otimes I_{\bar{j}} \right) V^\dagger A \tag{6.10}$$

and V is the matrix representing the right-hand side quantum state preparation procedure such as $V|0\rangle = |b\rangle$. A normalised version of the local cost function is also introduced as

$$\hat{C}_L(\vec{\alpha}) = \frac{\langle x(\vec{\alpha})| H_L |x(\vec{\alpha})\rangle}{\langle x(\vec{\alpha})| A^\dagger A |x(\vec{\alpha})\rangle}. \tag{6.11}$$

In order any of the cost functions presented above to be efficiently computable by a quantum computer, additional conditions should be verified by the matrix A. Primarily, A should be decomposable into a weighted sum of "easy-to-implement" (see Definition 19) unitary matrices:

$$A = \sum_{i=0}^{m-1} \alpha_i U_i \tag{6.12}$$

with A the matrix of the linear system at hand, α_i any complex number and U_i unitary matrices that can be efficiently implemented on a quantum computer.

Definition 19 (Easy-to-implement unitary matrices)**.** A unitary matrix U on n qubits is considered "easy-to-implement" if there exist a quantum circuit C implementing this unitary matrix with a number of quantum gates growing as $\mathcal{O}(\text{poly}(n))$.

6.3.2 Linear systems of interest

In order to study the VQLS algorithm behaviour and convergence, we need to define several representative and interesting linear systems to test the algorithm on. For the VQLS algorithm, each linear system $Ax = b$ is defined by:

1. The decomposition of the matrix A into a weighted sum of unitary matrices: $A = \sum_i \alpha_i U_i$.

2. The unitary V that initialises the right-hand side: $V|0\rangle = \sum_i b_i |i\rangle$.

Even though the choice of the ansatz $U(\vec{\alpha})$ that initialise the trial state $|x(\vec{\alpha})\rangle = U(\vec{\alpha})|0\rangle$ is highly problem-dependent or even instance-dependent, we chose to not include the ansatz in what *defines* a particular linear system problem. This choice is justified by the fact that, for a given linear system instance, any generic ansatz can theoretically be used to initialise the trial state.

We listed four different linear systems in order to highlight different behaviours of the VQLS algorithm with respect to different parameters. Each linear system used in this chapter is summarised in Table 6.1 and explained in more details in the following sections.

Identity linear system

The first linear system of interest consists in the trivial

$$\begin{aligned} \left(\bigotimes_{i=0}^{n-1} I_i\right)\mathbf{x} &= \left(\bigotimes_{i=0}^{n-1} H_i\right)|0\rangle \\ \Leftrightarrow I^{\otimes n}\mathbf{x} &= \frac{1}{\sqrt{2^n}}\sum_{i=0}^{2^n-1}|i\rangle \end{aligned} . \tag{6.13}$$

Having such a trivial linear system is important to establish a baseline. Apart from being the most simple instance of the linear system problem, the identity matrix that defines this system has several desirable properties. First, the identity matrix has a constant condition number $\kappa = 1$ with respect to the number of qubits. Moreover, due to the simplicity of the system, it is possible to easily choose V and a specific ansatz $U(\vec{\alpha})$ such that the ansatz is able to encode *exactly* the solution of the linear system. Finally, $A_I = \left(\bigotimes_{i=0}^{n-1} I_i\right)$ is a tensor product of identity matrices and as such checks the "easy-to-implement" condition from Definition 19.

In other words, the linear system in Equation (6.13) is particularly interesting as it allows the study of the VQLS algorithm in an ideal setting, removing several factors that might impact the convergence such as the underlying linear system hardness (quantified with the condition number $\kappa(A_I)$), the imprecisions due to the approximability of the solution with the trial states $|x(\alpha)\rangle$ at hand and the potentially larger number of quantum gates and quantum circuits that would be needed to implement any linear system more complex than the identity.

This `identity` linear system is used as the baseline for comparison.

Pauli-X

The second linear system of interest, presented in Equation (6.14), is also trivially solvable but actually requires more quantum gates and circuit submissions than the `identity` linear system from Section 6.3.2.

$$\left(\bigotimes_{i=0}^{n-1} X_i\right)\mathbf{x} = \left(\bigotimes_{i=0}^{n-1} H_i\right)|0\rangle \tag{6.14}$$

Theoretically, due to the fact that the matrix

$$A_X = \bigotimes_{i=0}^{n-1} X_i \tag{6.15}$$

representing this linear system is a permutation of the matrix A_I, this linear system should be as easy to solve as the `identity` linear system. Nevertheless, implementing in practice the VQLS algorithm for the matrix A_X requires more quantum gates than for the identity matrix. This leads to a linear system of equivalent hardness, but with an overhead due to its representation.

Another reason to include the `pauli-x` linear system from Equation (6.14) in the study is the fact that it can be seen as a poor quantum formulation of the `identity` linear system. Indeed the `pauli-x` linear system can be solved by first finding the permutation of columns that would change the A_X matrix into the identity A_I matrix and then solving the simpler `identity` linear system with a modified (permuted) right-hand side $|b\rangle$. Finding the permutation is trivial and re-organising the solution only requires the application n `X` gates (one on each qubit).

Varying condition number

When dealing with linear systems, the condition number κ of the matrix representing the linear system is a crucial quantity of interest that has huge implications on the inherent hardness of

the linear system. The condition number is defined as

$$\kappa(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)} \tag{6.16}$$

where $\sigma_{\max}$ (resp. $\sigma_{\min}$) is the maximum (resp. minimum) singular value of A. For normal matrices (i.e. matrices that commute with their complex conjugate), the condition number can also be computed by using the maximum and minimum eigenvalues $\lambda_{\max}$ and $\lambda_{\min}$:

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}. \tag{6.17}$$

The importance of the condition number κ on the linear system hardness makes it a parameter of choice for a study of the VQLS algorithm when this condition number κ vary. In this chapter we use the matrix

$$A(k) = \frac{k+1}{2} \bigotimes_{i=0}^{n-1} I_i + \left(1 - \frac{k+1}{2}\right) \bigotimes_{i=0}^{n-1} Z_i. \tag{6.18}$$

For any value of k, $A(k)$ is diagonal and the entries in its diagonal are drawn from $\{1, k\}$ meaning that its eigenvalues are $\{1, k\}$. $A(k)$ being diagonal and real, it is a normal matrix. As such its condition number for any parameter $k \geqslant 1$ is

$$\kappa(A(k)) = \frac{|\lambda_{\max}(A(k))|}{|\lambda_{\min}(A(k))|} = \frac{k}{1} = k. \tag{6.19}$$

The case $k < 1$ is less interesting for this study as our goal to have a linear system with a variable condition number is already fulfilled by $\geqslant 1$.

Discretised, periodic, implicit heat equation

The final test-case is inspired from a simple partial differential equation that is the normalised and periodic heat equation, shown in Equation (6.20).

$$\begin{aligned} \frac{\partial}{\partial t} f &= \frac{\partial^2}{\partial x^2} f \\ f(0,t) &= f(1,t) \\ f(x,0) &= f_0(x) \end{aligned} \tag{6.20}$$

Several methods exist to solve partial differential equations numerically. One of the most widespread is probably to reformulate the partial differential equation as a linear system by using some well-known classical discretisation schemes, for example finite differences.

Using a finite difference scheme, we can discretise the time dimension using steps of k (i.e. $t_n = nk$) and the space dimension using steps of h (i.e. $x_i = ih$). This discretisation allows to use a finite difference scheme to solve the heat equation presented in Equation (6.20). Depending on the schemes used to approximate time and space derivatives, solving Equation (6.20) can result in an explicit or an implicit method of resolution.

Let $f(x_i, t_n) = f_i^n$, the explicit method

$$\frac{f_i^{n+1} - f_i^n}{k} = \frac{f_{i+1}^n - 2f_i^n + f_{i-1}^n}{h^2} \tag{6.21}$$

is obtained when using a forward difference for the time derivative and a second-order central difference for the spatial derivative. The approximated value at time $n+1$, f_i^{n+1}, can be directly computed from Equation (6.21) using the formula

$$f^{n+1} = (1 - 2r) f_i^n + r f_{i+1}^n + r f_{i-1}^n \tag{6.22}$$

where $r = k/h^2$. This explicit method has the advantage of being simple to solve, but is known to be numerically unstable or non-convergent when $r > 1/2$, which might be an issue.

A more complex but resilient numerical scheme is obtained when using a backward difference for the time derivative and a second-order central difference for the spatial derivative:

$$\frac{f_i^{n+1} - f_i^n}{\delta k} = \frac{f_{i+1}^{n+1} - 2f_i^{n+1} + f_{i-1}^{n+1}}{\delta h^2}. \tag{6.23}$$

This implicit method leads to the resolution of a linear system of equations given by

$$(1 + 2r)\, f_i^{n+1} - r f_{i+1}^{n+1} - r f_{i-1}^{n+1} = f_i^n \tag{6.24}$$

or, in its matrix form:

$$B\,\mathbf{f^{n+1}} = \begin{pmatrix} 1+2r & -r & 0 & \cdots & 0 & -r \\ -r & \ddots & \ddots & \ddots & & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & & \ddots & \ddots & \ddots & -r \\ -r & 0 & \cdots & 0 & -r & 1+2r \end{pmatrix} \mathbf{f^{n+1}} = \mathbf{f^n} \tag{6.25}$$

where $\mathbf{f^n}$ is the vector of the space-discretised values of f at time $t = nk$. The matrix B can be written as a weighted sum of "easy-to-implement" (see Definition 19) unitary matrices as shown in Equation (6.26):

$$B = (1 + 2r) \left(\bigotimes_{i=0}^{n-1} I_i \right) - r\,(\mathtt{add}_1) - r\,(\mathtt{add}_1)^\dagger \tag{6.26}$$

were

$$\mathtt{add}_1 = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & & & \ddots & 1 \\ 1 & 0 & \cdots & \cdots & 0 \end{pmatrix} \tag{6.27}$$

is the unitary matrix representing the quantum operation that adds 1 to a quantum register. This decomposition is usable with the VQLS algorithm because there exist efficient implementations of the $\mathtt{add}_1$ gate [203] (see requirements in Section 6.3).

Note that it is also possible to implement efficiently the non-periodic version of the linear system with Dirichlet boundary conditions, but this requires to implement 2-level unitaries as a quantum circuit. The implementation of 2-level unitary matrices into a quantum circuit is *efficient* [204], i.e., requires a number of gates that grows logarithmically with the linear system size, but is not NISQ-compatible as it requires several multi-controlled X gates.

6.4 Results of the study

We ran the different linear systems listed in Section 6.3.2 using the hardware-aware ansatz presented in Section 6.2.2.

6.4.1 Global versus local cost function

As noted in Section 6.3.1, the VQLS algorithm can use either a global or a local cost function. These cost function both have different characteristics and result in different quantum circuits being submitted to the quantum chip or simulator.

Let

$$|\Psi(\vec{\alpha})\rangle = \frac{A|x(\vec{\alpha})\rangle}{\langle x(\vec{\alpha})|A^\dagger A|x(\vec{\alpha})\rangle}, \tag{6.28}$$

both cost functions can be re-written as

$$C_G(\vec{\alpha}) = \langle\Psi(\vec{\alpha})|\Psi(\vec{\alpha})\rangle - |\langle b|\Psi(\vec{\alpha})\rangle|^2 \tag{6.29}$$

and

$$C_L(\vec{\alpha}) = \frac{1}{2}\left(\langle\Psi(\vec{\alpha})|\Psi(\vec{\alpha})\rangle - \frac{1}{n}\sum_{j=1}^{n}\langle\Psi(\vec{\alpha})|UZ_jU^\dagger|\Psi(\vec{\alpha})\rangle\right) \tag{6.30}$$

by using the equality

$$I - \frac{1}{n}\sum_{i=1}^{n}|0_j\rangle\langle 0_j| \otimes I_{\bar{j}} = \frac{1}{2}\left(I + \frac{1}{n}\sum_{i=1}^{n} Z_j \otimes I_{\bar{j}}\right). \tag{6.31}$$

By replacing A with its decomposition as shown in Equation (6.12), developing the expression and counting the number of terms that should be computed on the quantum computer, we can compute the number of circuit evaluations needed to compute either the global or the local cost functions. The global cost function C_G needs $2m^2 - m$ circuit evaluations, where m is the number of terms in the decomposition of A (see Equation (6.12)). The local cost function C_L requires $(n+1)m^2 - nm$ circuits, where m as been defined previously and n is the number of qubits.

The number of quantum circuit executions needed seems to indicate that the global cost function C_G is always the best choice as it always require less quantum circuit execution per cost function evaluation. But as written in Section 6.2.3, global cost functions suffer from the Barren plateau phenomenon, which means that the cost function C_G will eventually become untrainable due to vanishing gradients. The $\mathcal{O}(n)$ overhead in the number of quantum circuit evaluations required by the local cost function C_L is in fact a necessary condition to avoid the apparition of the Barren plateau phenomenon.

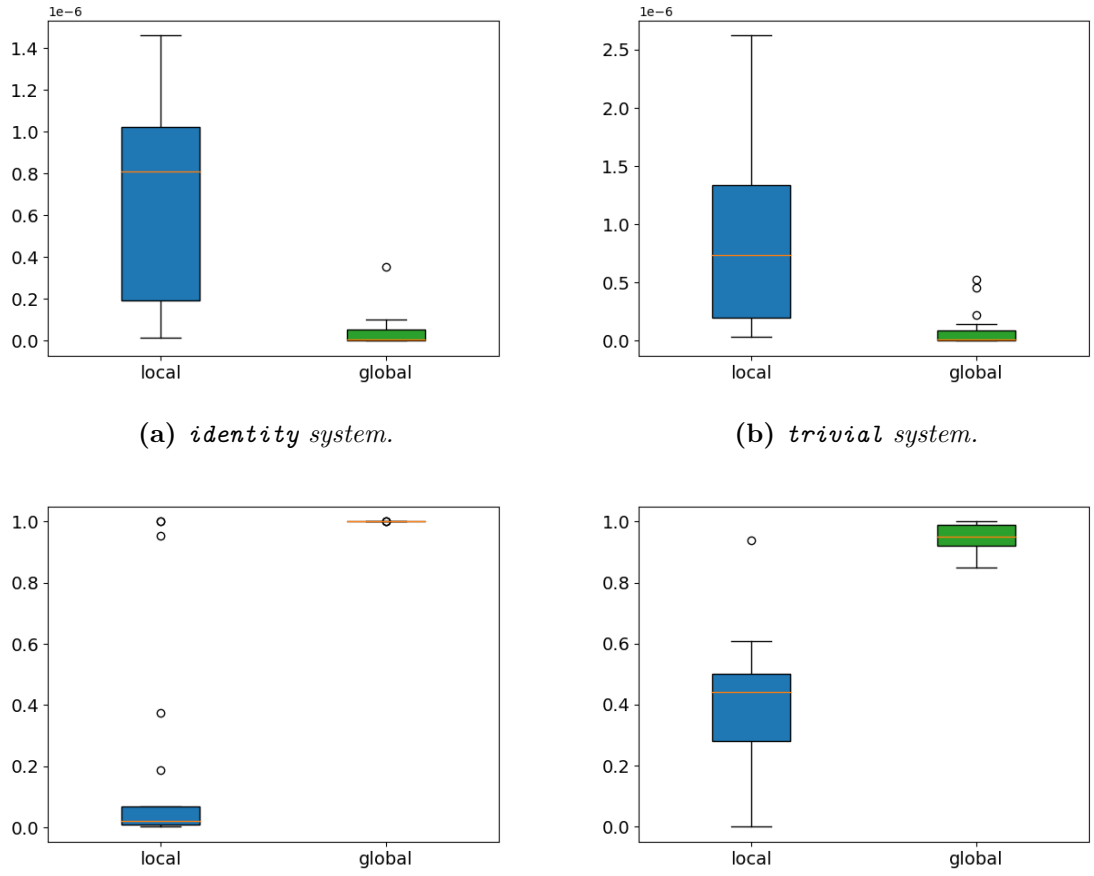
Figure 6.6 shows that, for easy problems such as `identity` or `trivial`, the global cost function achieve better results than the local cost function. Nevertheless, the global cost function completely fails to obtain any meaningful results on harder problems such as `varying_condition` or `heat_opti`. This behaviour can also be observed with all the other optimisers used in this study: SPSA, COBYLA and POWELL.

6.4.2 Dependence on the condition number κ

The condition number κ as defined in Equations (6.16) and (6.17) is an important parameter of any system of linear equations as it roughly describes the impact of a small change in b on the solution x. A system with a low condition number κ is said to be *well-conditioned* whereas *ill-conditioned* systems have a high condition number.

Figures 6.7 and 6.8 show the convergence of the COBYLA, POWELL, SPSA and SLSQP optimisation algorithms when used with the VQLS algorithm on the system defined in Equation (6.16) with increasing condition numbers.

The expected scaling is $\epsilon \in \mathcal{O}(\kappa)$, i.e., the precision ϵ that can be obtained numerically on the solution of a system of linear equations that has a condition number κ scales linearly in κ [205].



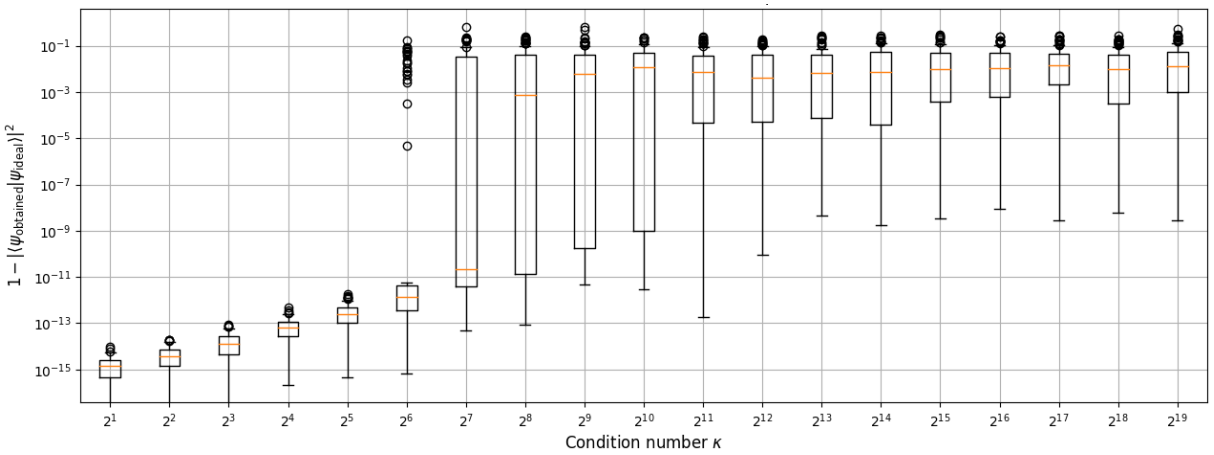
(a) *identity system.*

(b) *trivial system.*

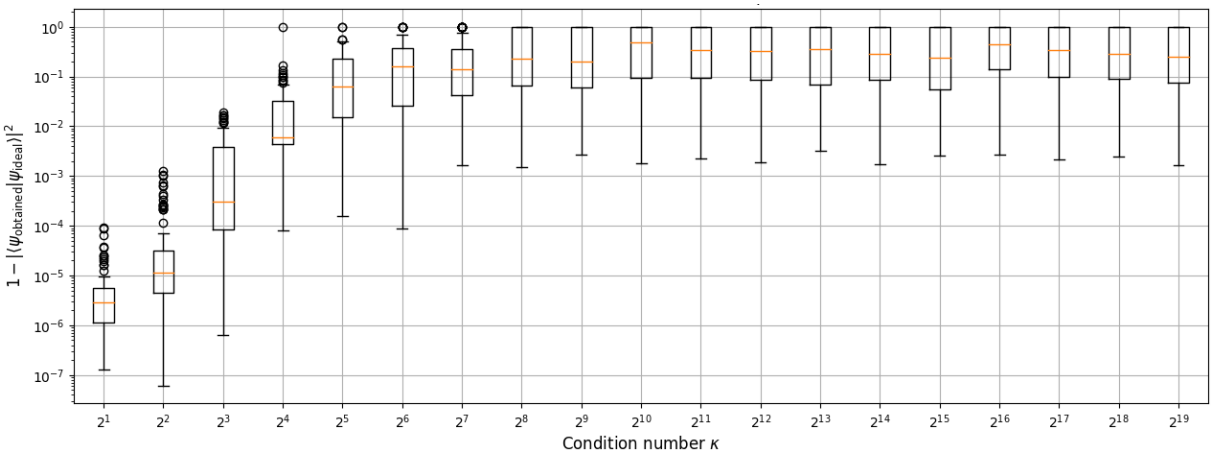
(c) *varying_condition system with* $\kappa = 1024$.

(d) *heat_opti system.*

Figure 6.6: *Plots representing the distribution of the error obtained on the final solution with the global and local cost functions by using a perfect statevector simulator and SLSQP optimiser on* 5*-qubit problems. The error on the final solution is obtained by computing the quantum state fidelity between the exact solution (obtained by classical algorithms) and the solution obtained with the VQLS algorithm.*



(a) *Convergence of the COBYLA optimiser.*

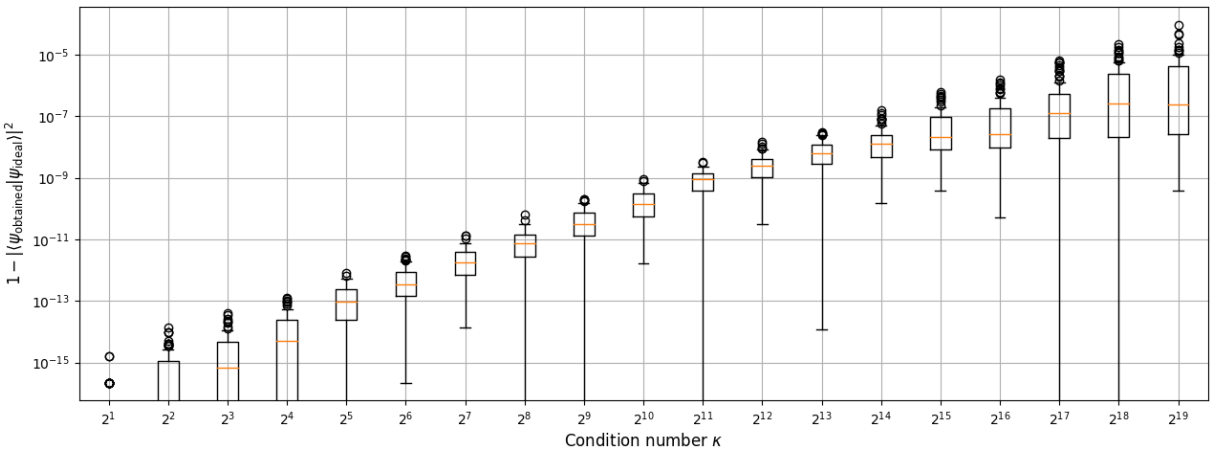


(b) *Convergence of the SLSQP optimiser.*

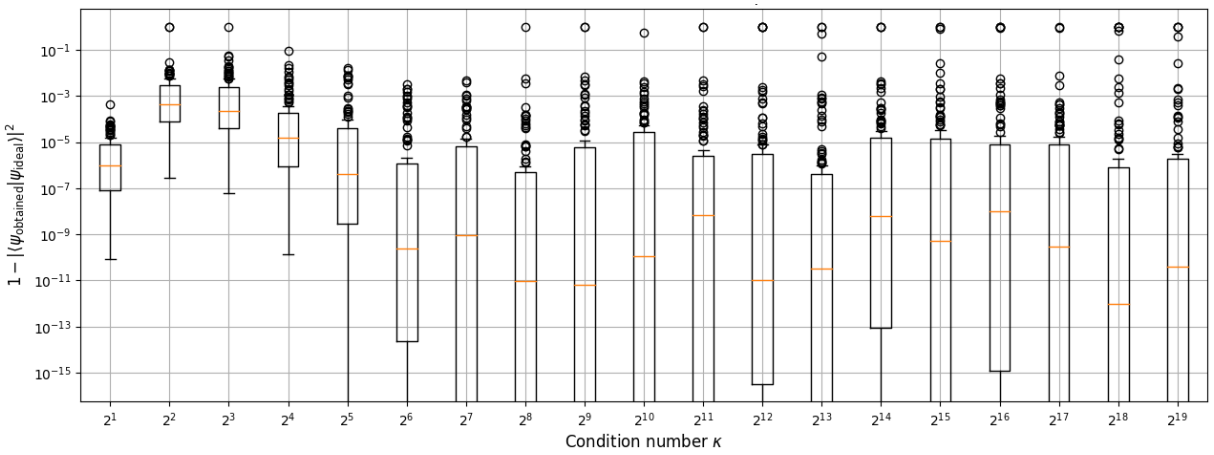
Figure 6.7: *Plot of the VQLS algorithm convergence with different values of κ, the condition number of the linear system matrix as defined in Equation (6.19), for the linear system defined by the matrix from Equation (6.18) and a right-hand side $b = \left(\bigotimes_{i=0}^{2} H_i\right)|000\rangle$ and with an ideal simulator. For each of the optimisers, the condition number plays an important role in the convergence to the correct solution. Similar results have been obtained for the same setup but by using 10 qubits. Each optimiser was given 1000 iterations and each box-plot contains 100 independent optimisation run.*

This expected linear scaling can be seen in Figures 6.7a, 6.7b and 6.8a with the POWELL optimiser that follows exactly a linear scaling and the SLSQP and COBYLA optimiser that, for low values of κ, also exhibit the linear scaling $\epsilon \in \mathcal{O}(\kappa)$. Nevertheless, both the SLSQP and COBYLA optimisers fail to get a correct solution for higher values of κ. An interesting observation is that the COBYLA optimiser obtains systematically a very high-precision solution for $\kappa \leqslant 2^5$. Then, it starts to experience a few bad solutions that does not change significantly the distribution of the solution between the first and third quartiles at $\kappa = 2^6$. The optimiser struggle even more at $\kappa = 2^7$, with the third quartile going from $\approx 10^{-11}$ for $\kappa = 2^6$ to $> 10^{-2}$ for $\kappa = 2^7$. Finally, the distribution of final state infidelities show that the COBYLA optimiser fails to find a good approximate solution in most of the optimisation runs for $\kappa \geqslant 2^9$ with more than half of the obtained quantum states that have an infidelity above 10^{-2}.

The convergence observed in Figure 6.8b when using the SPSA optimiser does not follow the expected linear scaling, even at the regime for low values of κ. More interestingly, the SPSA optimiser does not seem sensitive to an increase in the value of κ and succeed in optimising successfully most of the optimisation runs to obtain an output quantum state very close to the solution of the linear system, with median infidelities below 10^{-8} for $\kappa \geqslant 2^6$ and most of the



(a) *Convergence of the POWELL optimiser.*



(b) *Convergence of the SPSA optimiser.*

Figure 6.8: *Plot of the VQLS algorithm convergence with different values of κ, the condition number of the linear system matrix as defined in Equation (6.19), for the linear system defined by the matrix from Equation (6.18) and a right-hand side $b = \left(\bigotimes_{i=0}^{2} H_i\right)|000\rangle$ and with an ideal simulator. For each of the optimisers, the condition number plays an important role in the convergence to the correct solution. Similar results have been obtained for the same setup but by using 10 qubits. Each optimiser was given 1000 iterations and each box-plot contains 100 independent optimisation run.*

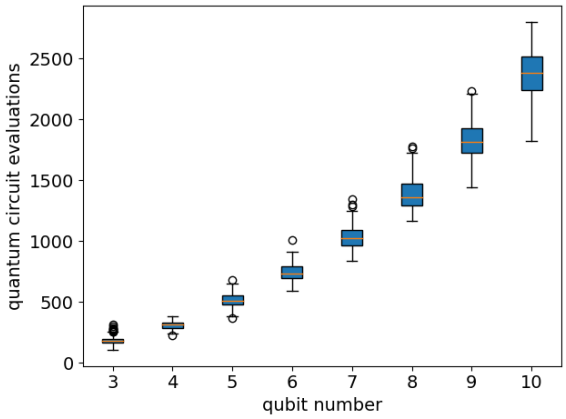
time more than $\frac{1}{4}$ of the runs ending up at the ideal solution. This hints us that the SPSA optimiser might be a good candidate to solve systems of linear equations with a high condition number κ.

6.4.3 Dependence on the size of the linear system

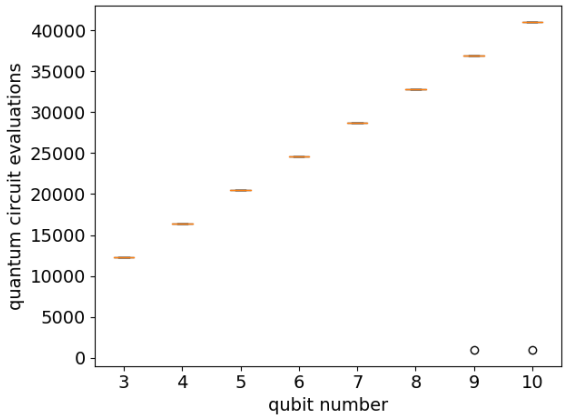
The size of the linear system to be solved is another variable of interest as our end goal will be to scale up the linear system to sizes that are not manageable on current classical hardware.

Figure 6.9 shows the number of quantum circuits required for the COBYLA, SLSQP and POWELL[1] optimisers to obtain a solution with an error (infidelity with respect to the ideal solution) below 10^{-5}. These plots indicate that the VQLS algorithm can converge to a desired precision in a number of circuit evaluations that scale as $\mathcal{O}\left(\text{poly}\log\left(n\right)\right)$, where n is the number of qubits.

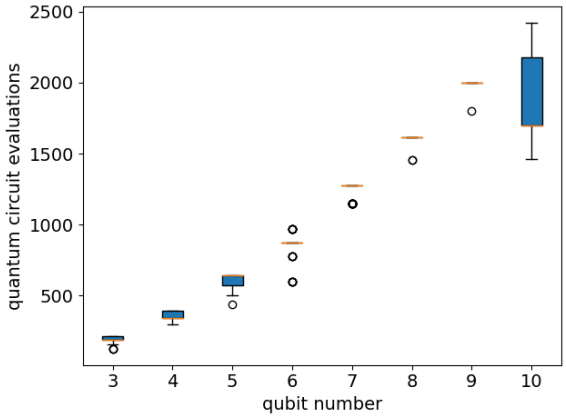
[1]The SPSA optimiser should be excluded from comparison here, see the main caption of Figure 6.9.



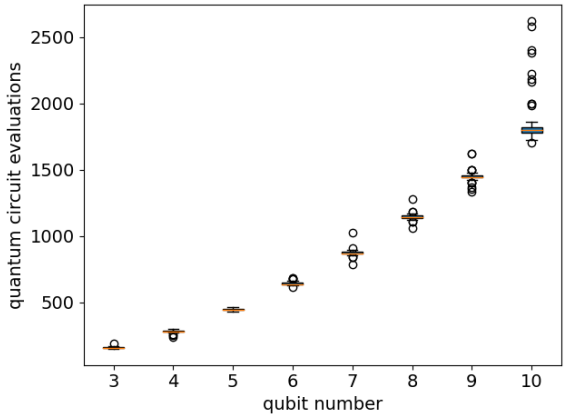
(a) *COBYLA optimiser.*



(b) *SPSA optimiser. See note in main caption.*

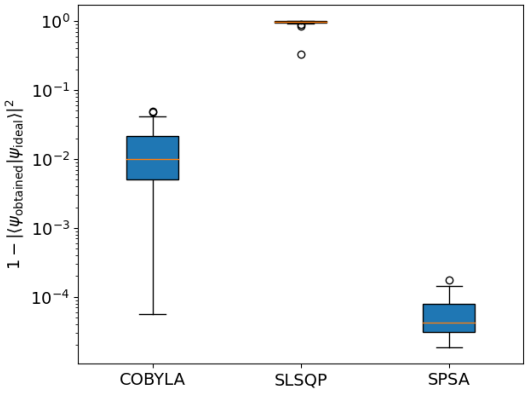


(c) *SLSQP optimiser.*

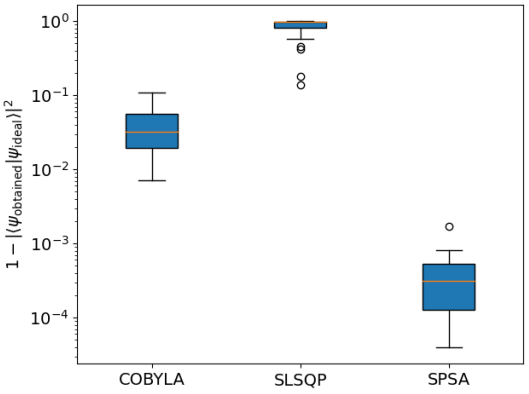


(d) *POWELL optimiser.*

Figure 6.9: *Plot of the number of quantum circuit evaluations required with each optimiser to obtain a precision of* $\epsilon = 10^{-5}$ *on the obtained solution of the* `identity` *system of linear equations with respect to the number of qubits (i.e., the size of the problem). Each box is obtained by repeating the optimisation* 20 *times with a random initialisation point. Note that the SPSA implementation used does not compute the value of the current optimisation point in order to save a few circuits evaluations. As such, Figure 6.9b shows the number of quantum circuit evaluations that have been performed for the entirety of the optimisation run and does not reflect the true performance of SPSA.*



(a) *Convergence of the COBYLA, SLSQP and SPSA optimisers on a noisy simulator* mimicking *the* `ibm_lagos` *backend.*



(b) *Convergence of the COBYLA, SLSQP and SPSA optimisers on a noisy simulator* mimicking *the* `ibmq_bogota` *backend.*

Figure 6.10: *Plot of the VQLS algorithm convergence on the* `identity` *system using Qiskit noisy simulation capabilities,* mimicking *the hardware noise of* `ibm_lagos` *and* `ibmq_bogota` *respectively. POWELL optimiser performs as bad as SLSQP and is not plotted. Each box plot has been obtained from* 20 *independent optimisation runs with a random initial point.*

6.4.4 Running VQLS on noisy hardware

Executing the VQLS algorithm on noisy hardware brings a lot of challenges. One of the most prominent challenge is due to the different sources of noise present in NISQ quantum hardware (finite sampling, decoherence, gate errors, state preparation and measurement errors, ...). The backend noise directly influence the cost function, that suffers from high errors and as such becomes harder to optimise correctly.

Figure 6.10 shows the convergence of each optimiser (except POWELL that does not converge and perform as the SLSQP optimiser) on a noisy simulator mimicking the noise of real quantum hardware. In this noisy setting, the SPSA optimiser clearly outperforms the other optimisers, achieving good accuracies even with 2-qubit gate error-rate close to 1%.

It should be noted that the SPSA optimiser requires a lot of quantum circuit executions to achieve the precision shown in Figure 6.10, and for small budgets (i.e., when one can only execute a low number of quantum circuits) the COBYLA optimiser seems to obtain better results.

Figure 6.11 show the results obtained on real quantum hardware. Only the COBYLA optimiser has been benchmarked due to the relatively low budget of quantum circuit execution imposed by IBMQ queue system to run on real hardware. The precision obtained on real quantum systems is still insufficient as most of the optimisation runs do not achieve an error below 10^{-3}.

6.5 Conclusion

In this chapter we studied a variational algorithm to solve systems of linear equations. After a careful choice of problems, we ran multiple simulation to benchmark the dependence of the algorithm with respect to several parameters such as the condition number of the linear system κ or its size.

We used our implementation of the VQLS algorithm to run on noisy simulators and real quantum hardware, that showed that the errors of current NISQ hardware (due to finite sampling and errors from hardware imperfections) impact significantly the optimisers ability to obtain a good approximation of the solution. Simulations on noisy simulators showed that the SPSA optimiser is the most efficient if one has a high budget (i.e., is able to evaluate a high number of

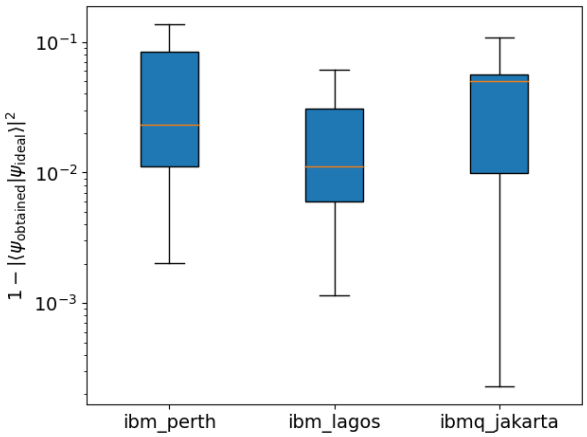


Figure 6.11: *Precision of the solution obtained by solving the* `identity` *system of linear equations on* 3 *qubits on real quantum hardware by using the COBYLA optimiser.*

quantum circuits). If only a low budget of quantum circuit evaluations is available, the COBYLA optimiser is a good candidate as it is able to converge quicker than the SPSA optimiser.

Running the VQLS algorithm on NISQ devices is still a challenge nowadays and several improvements will need to be performed to the current quantum hardware in order to hope using this algorithm in the future. First and foremost, the hardware error rates should be drastically decreased. An initial amelioration can be made by using error mitigation techniques, but the challenge remains even with such improvements. Due to the number of quantum circuits needed to evaluate once the local cost function C_L that scales as $\mathcal{O}\left((n+1)m^2\right)$, and the variational nature of the algorithm, efficient classical processing and fast circuit submissions techniques should be employed to avoid large overheads. The Qiskit Runtime feature, that allows one to submit a full variational algorithm execution without the need to perform communications between the local machine and IBM servers at each cost function evaluation, has been introduced to solve one of these problems.

In conclusion, successfully running the VQLS algorithm to solve interesting linear systems on real quantum hardware is still a challenge and would require several improvements to the software and hardware stack to start becoming usable on interesting problems.

Part V

Noise characterisation

Chapter

7 Single qubit tomography visualisation

Dealing with real quantum hardware is challenging as seen in Chapter 6. Hardware calibrations are changing rapidly and in an apparently unpredictable manner. Moreover, the hardware calibrations provided by most vendors are only representing a part of the noise experienced by the hardware and the noise models available are known to be lacking details due to observed discrepancies between the noise predicted by the model and the actual hardware behaviour.

As already said in Chapter 4, any optimisation process has to start by measuring extensively the quantities to optimise. As such, whether our aim is to improve noise models or to lower down hardware error-rates, we have to start by measuring extensively hardware noise and get a better understanding of what is happening. The research presented in this chapter is a novel attempt at measuring the noise experienced by a qubit in order to discover new ways of characterising it and in turn devise new ways of mitigating noise.

This work has been accepted at the IEEE International Conference on Quantum Computing and Engineering 2022 as a technical short paper.

Contents

7.1 Introduction

The emergence of commercial Quantum Computing (QC) hardware has made tools for Quantum Characterisation, Verification, and Validation (QCVV) [206] more important than ever. Especially in the era of Noisy Intermediate-Scale Quantum (NISQ) [109] devices, QCVV methods provide the means for QC users to measure and quantify the performance of quantum hardware platforms and enables consistent comparisons across different hardware architectures. The scope of QCVV is broad and ranges from testing individual quantum operations (e.g., error rates of one- and two-qubit gates [207]), verifying small circuits (e.g., Quantum State Tomography [208], Randomised Benchmarking [209, 210], Gate Set Tomography [211]), to full system-level protocols (e.g., quantum volume estimation [212], random quantum circuits [213]). Over the years these

QCVV tools have become an invaluable foundation for benchmarking and measuring progress of quantum processors [214], culminating in multiple quantum supremacy demonstrations [215, 216].

Quantum state tomography provides a gold standard for QCVV in that it can theoretically provide an exact reconstruction of the full quantum state of QC hardware, with sufficient data. At its most basic level, quantum state tomography provides a protocol for combining multiple observations to uniquely identify the state of a quantum system (i.e., a density matrix, see Section 1.3.1). On small quantum systems, where data collection and result computations are feasible, quantum state tomography provides a precise measure of how accurately a quantum hardware device can execute a desired quantum computation. The strength of quantum state tomography for QCVV is that it can provide a comprehensive picture of hardware performance. The drawback is that it requires a prohibitively large amount of data and interpreting the results can be complex. To mitigate this, many other QCVV protocols (e.g. Randomised Benchmarking [209, 210], Gate Set Tomography [211]) choose to provide trade offs in data collection and result details resulting in more scalable QCVV alternatives to quantum state tomography.

This work explores the use of quantum state tomography to conduct QCVV on commercially available QC platforms. We focus on quantum state tomography of single-qubits to minimise the data collection requirements and to develop a protocol that can be executed in parallel for all of the qubits in a given hardware platform. In addition to the reduced data collection, the limitation to single-qubit states allows to reduce significantly the complexity when interpreting the results. The core contributions of this work are a *Vector Field Visualisation* of single-qubit quantum state tomography and an open-source software tool for data collection, state reconstruction and result presentation. Through experiments on QC hardware available in IBM's Q-Hub, we show that the proposed method can identify qubit performance features that are not easily identified and captured with a single value and provide clear signatures that distinguish qubit performance from both perfect and noisy simulators of this hardware.

This work begins by introducing the foundations of quantum state tomography for a single-qubit in Section 7.2 and reviews the maximum likelihood estimation method [208] for reconstructing a quantum state from a finite number of quantum measurements. Section 7.3 then proposes the vector field visualisation for presenting the single-qubit quantum state tomography results and illustrates how this visualisation can be leveraged to provide unique insights into qubit performance. Section 7.4 investigates how the results from different quantum state tomography algorithms can be combined with the vector field visualisation to identify signatures of data corruption. The details of the open-source software are provided in Section 7.5 and the paper concludes with a discussion of the usefulness of the proposed protocol and future work in Section 7.6.

7.2 Single-Qubit State Tomography

As explained in Section 1.3.1, the state of a quantum system composed of n qubits is fully described by a $2^n \times 2^n$ hermitian matrix ρ, the so-called density matrix. Density matrices are positive semi-definite, normalised matrices, i.e., $\langle\phi|\,\rho\,|\phi\rangle \geq 0$ for all $|\phi\rangle \in \mathbb{C}^{2^n}$, and $\mathrm{Tr}\,[\rho] = 1$.

The task of reconstructing density matrices from repetitive observations of these projections is called Quantum State Tomography (QST). Since the space of density matrices has $4^n - 1$ real parameters, exact QST can only be done if one records at least $4^n - 1$ different projections. Therefore, general QST remains prohibitive for quantum systems of moderate to large size due to the exponential growth in the number of required observations. However, for single-qubits, QST is tractable and provides a complete description of the quantum system, making it a powerful tool for conducting QCVV of individual qubits in QC hardware.

7.2.1 Maximum-Likelihood Quantum State Tomography

There exists multiple methods (i.e., statistical estimators) that can be used for QST. Each of these estimators comes with its own advantages and disadvantages such as improved reconstruction quality with respect to specific metrics, ease of implementation, or computational benefits. Some popular choices for QST include linear regression based methods [217], nuclear norm constrained reconstructions [218] and the Maximum-Likelihood Estimator (MLE) [219]. In this work, we adopted the MLE method as it is widely used in practice, leverages fundamental statistical theory principles and can be easily implemented for small quantum systems. However, a sensitivity study on simulated data suggested that all of these methods produce similar results under the specific data collection settings used in this work.

The Maximum-Likelihood approach for QST consists in finding the density matrix ρ that will maximise the probability of realised measurements. Our observables are measures k described by an ensemble of projectors $\{P_k\}$ that is the union of all measurement bases or more precisely the Projection-Valued Measures (PVMs) that we consider. What is recorded, and what serves as an input to the MLE algorithm, are the number of times, n_k, that a particular measure k has been observed. The log-likelihood function is then expressed in terms of our statistics as

$$\ln \mathbb{P}\left[\{n_k\} \mid \rho\right] = \sum_k n_k \ln \mathrm{Tr}\left[\rho P_k\right]. \tag{7.1}$$

The reconstructed density matrix ρ_{out} is the output of the following concave maximisation problem on the positive semi-definite cone,

$$\rho_{\text{out}} = \underset{\substack{\rho \succcurlyeq 0 \\ \mathrm{Tr}[\rho]=1}}{\arg\max} \sum_k n_k \ln \mathrm{Tr}\left[\rho P_k\right]. \tag{7.2}$$

Due to the limitations of off-the-shelf optimisation software, in practice, enforcing the positive semi-definite (PSD) constraint, $\rho \succcurlyeq 0$, may require a specialised optimisation algorithm. One possible approach to solving Equation (7.2) consists in using local gradient ascent algorithm interleaved with eigendecomposition based projections onto the SDP cone. However, we will see that for single-qubit QST, the SDP constraint has a convenient simplification.

7.2.2 Specialising to Single-Qubit State Tomography

The Bloch vector representation of quantum states enables us to significantly simplify QST for an individual qubit. In this representation, density matrices are encoded as vectors $\vec{a} \in \mathbb{R}^3$ following the decomposition

$$\rho = \frac{1}{2}\left(I + \vec{a} \cdot \vec{\sigma}\right) \tag{7.3}$$

where $\vec{\sigma}$ is the vector of Pauli matrices. The PSD constraint that ρ must satisfy is enforced through the requirement that its corresponding Bloch vector lies within the unit sphere, i.e., $\|\vec{a}\| \leq 1$. Projectors are described in a similar fashion by

$$P_{\vec{u}} = \frac{1}{2}\left(I + \vec{u} \cdot \vec{\sigma}\right) \tag{7.4}$$

where $\|\vec{u}\| = 1$. Any 1-qubit PVM is composed of only two projectors $P_{\vec{u}}$ and $P_{-\vec{u}} = I - P_{\vec{u}}$, and therefore can be identified by a single unit vector $\vec{u}$. Since we choose to perform the same number of measurements N in each PVMs, we only need to record the empirical probability $p_{\vec{u}} = n_{\vec{u}}/N$ of measuring the observable associated with $\vec{u}$. The MLE estimator from Equation (7.2) is then simplified into the following program,

$$\vec{a}_{\text{out}} = \underset{\|\vec{a}\| \leq 1}{\arg\max} \sum_{\vec{u}} p_{\vec{u}} \ln\left(1 + \vec{a} \cdot \vec{u}\right) + \left(1 - p_{\vec{u}}\right) \ln\left(1 - \vec{a} \cdot \vec{u}\right). \tag{7.5}$$

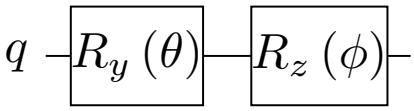


Figure 7.1: *The state preparation procedure used to initialise a single-qubit quantum state where the elementary rotations are defined by* $R_y(\theta) = \exp(-iY\theta/2)$ *and* $R_z(\phi) = (-iZ\phi/2)$.

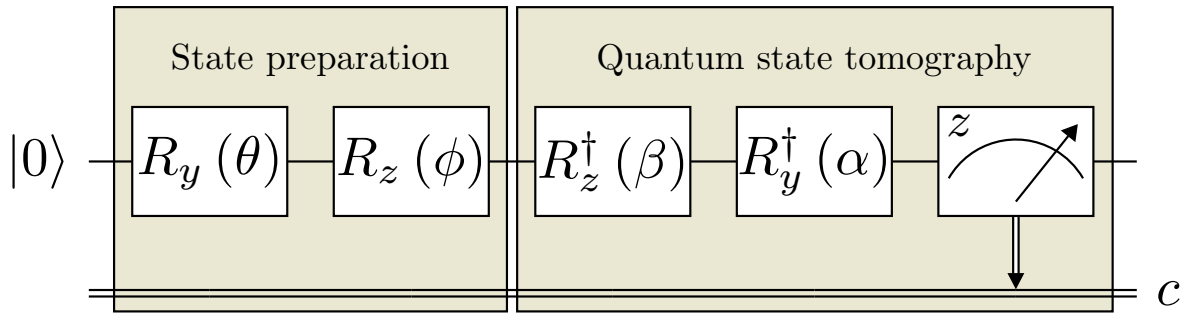


Figure 7.2: *The state preparation and tomography procedure used to initialise a single-qubit quantum state.*

The maximisation problem in Equation (7.5) is amenable to standard optimisation software for it is a simple concave non-linear problem in $\mathbb{R}^3$ with the cumbersome SDP constraint from Equation (7.2) replaced with a unit sphere constraint.

7.2.3 Single-Qubit State Tomography Experiment Design

Our main goal is to reconstruct the state of a single qubit programmed to be in the pure state

$$\rho_{\text{in}} := R_{\theta,\phi} |0\rangle\langle 0| R_{\theta,\phi}^\dagger \tag{7.6}$$

where the rotation matrix $R_{\theta,\phi}$ is implemented through elementary rotations with respect to the y and z axis $R_{\theta,\phi} = R_z(\phi)R_y(\theta)$ as depicted in Figure 7.1. The programmed density matrix ρ_{in} is represented on the Bloch sphere with the unit vector

$$\vec{a}_{\text{in}}(\theta,\phi) = \begin{pmatrix} \sin(\theta)\cos(\phi) \\ \sin(\theta)\sin(\phi) \\ \cos(\theta) \end{pmatrix}. \tag{7.7}$$

Using the MLE estimator from Equation (7.5), we will assess the quality of the state preparation for a specific qubit by comparing $\vec{a}_{\text{in}}(\theta,\phi)$ with the corresponding $\vec{a}_{\text{out}}$ for multiple values of ϕ and θ.

In our experiments, we are limited to measurements in the computational basis, that is to say, to the PVM associated with the vector $(0,0,1)$. To remedy to this issue, we rotate the qubit before measuring it with the matrix $R_{\alpha,\beta}^\dagger$ to emulate a PVM on

$$\vec{u} = \begin{pmatrix} \sin(\alpha)\cos(\beta) \\ \sin(\alpha)\sin(\beta) \\ \cos(\alpha) \end{pmatrix}. \tag{7.8}$$

The rotation matrix is again implemented with elementary rotations $R_{\alpha,\beta}^\dagger = R_y(-\alpha)R_z(-\beta)$. The complete circuit is shown in Figure 7.2.

Conducting QST on a single qubit requires measurements from at least 3 different linearly independent PVMs. One natural choice are the Pauli PVMs along canonical axes, represented by the set of values

$$(\alpha,\beta) \in \{(0,0),(\pi/2,0),(\pi/2,0)\}. \tag{7.9}$$

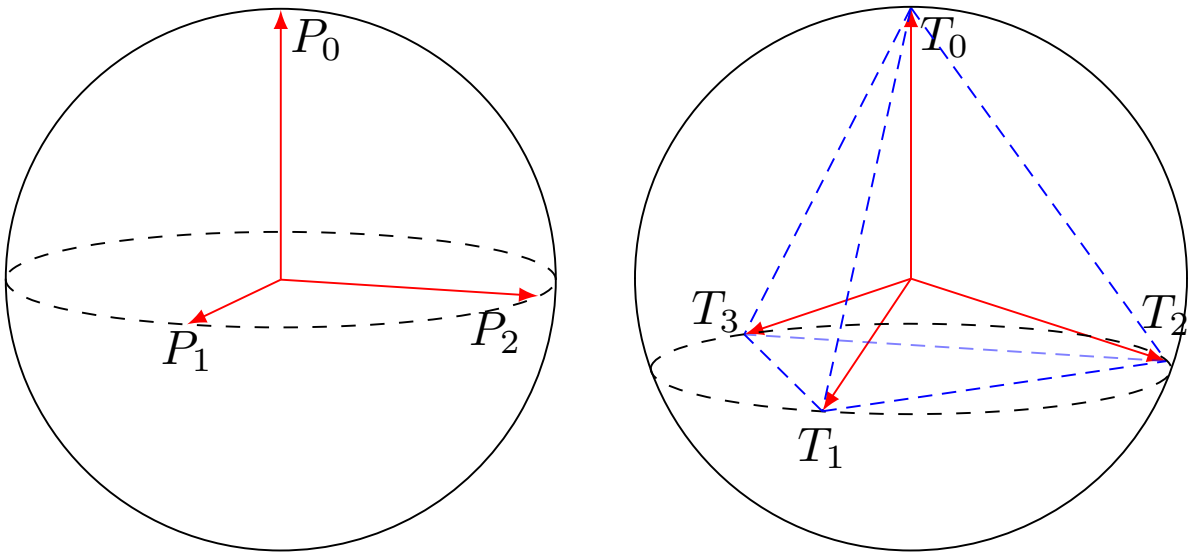


Figure 7.3: *The Bloch sphere representation of PVMs that can be considered for single-qubit state tomography. The Pauli operators (left) provide a minimal set of PVMs while the tetrahedral PVMs used in this work (right) provides data redundancy and improved reconstruction accuracy. In this figure,* $P_0 = |0\rangle$, $P_1 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $P_2 = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$, $T_0 = |0\rangle$ *and* $T_k = \frac{1}{\sqrt{3}}|0\rangle + \sqrt{\frac{2}{3}}e^{i\frac{2}{3}k\pi}|1\rangle$ *for* $k \in \{1, 2, 3\}$.

However, having more than the minimum number of PVMs can be beneficial in reducing the statistical error in state reconstruction. In this work we leverage a tetrahedral set of PVMs using

$$(\alpha, \beta) \in \left\{(0,0), \left(\cos^{-1}\left(\frac{-1}{3}\right), 0\right), \left(\cos^{-1}\left(\frac{-1}{3}\right), \frac{2\pi}{3}\right), \left(\cos^{-1}\left(\frac{-1}{3}\right), \frac{-2\pi}{3}\right)\right\} \tag{7.10}$$

to provide a balance between mitigating statistical error and data collection requirements. The Bloch sphere vectors representing both the Pauli and tetrahedral PVMs are depicted in Figure 7.3.

The experimental procedure for conducting single-qubit QST proposed in this work uses the tetrahedral PVMs with the MLE model from Equation (7.5) as follows. For a given state $\vec{a}_{\text{in}}(\theta, \phi)$ on the Bloch sphere, four variants of the tomography circuit (see Figure 7.2) are executed, one for each (α, β) combination in the tetrahedral PVMs. State measurement statistics are collected for each of these circuits and converted into empirical probabilities $p_{\vec{u}}$ providing the information required for posing the MLE problem from Equation (7.5). The optimal solution to Equation (7.5), $\vec{a}_{\text{out}}$, encodes the most likely density matrix that was implemented by the qubit at the state $\vec{a}_{\text{in}}(\theta, \phi)$.

An important subtlety in using a QST workflow in practice is to quantify how the accuracy of the empirical probability impacts the solution quality of model Equation (7.5). On a quantum hardware platform one only has access to a finite number of measurements, N (a.k.a., *shots*), to estimate the empirical probabilities $p_{\vec{u}}$. If the number of measurements is not sufficient, finite sample errors can yield significant artifacts in $\vec{a}_{\text{out}}$. We performed a sensitivity study to quantify this statistical error on simulated data. For a given prepared state $\vec{a}_{\text{in}}(\theta, \phi)$, we sample $N = 20,000$ observations for each of the tetrahedral PVMs, and then reconstruct $\vec{a}_{\text{out}}$ with our QST MLE algorithm. We repeat this procedure 10^4 times to accumulate an empirical histogram of the statistical error for each state, $a_{\text{in}}(\theta, \phi)$, and estimate the 99-th percentile of the Euclidean distance error ϵ (i.e., $\epsilon \in \mathbb{R}^+$ such as $\mathbb{P}[\|\vec{a}_{\text{out}} - \vec{a}_{\text{in}}(\theta, \phi)\| \leq \epsilon] = 0.99$). We find empirically that $\epsilon \leq 0.02$ for all angles (θ, ϕ). Therefore, throughout this work we standardised on $N = 20,000$ observations per circuit to ensure that statistical fluctuations will only contribute to an error in the second digit, with high confidence.

7.3 Vector Field Visualisation of Single-Qubit State Tomography

To illustrate the potential usefulness of single-qubit QST, this work investigates two questions arising in QCVV: (1) how accurate is single-qubit state preparation; (2) how consistent is the state preparation quality throughout the Bloch sphere. The first question amounts to quantifying the difference between the *ideal* quantum state, $\vec{a}_{\text{in}}(\theta, \phi)$, and the *reconstructed* quantum state resulting from the QST protocol, $\vec{a}_{\text{out}}$. The second question consists of repeating the state tomography protocol for a wide variety of possible state preparations and investigating how the reconstructed states vary. Throughout this section we develop the idea of a vector field visualisation for presenting the results of single-qubit QST to visually investigate these two questions. A variety of examples are then used to illustrate the usefulness of the proposed approach.

For a given single-qubit state $\vec{a}_{\text{in}}(\theta, \phi)$, the tomography procedure described in Section 7.2 yields a density matrix ρ_{out} in the form of a vector in $\vec{a}_{\text{out}} \in \mathbb{R}^3$. Many metrics can be computed from $\vec{a}_{\text{out}}$ (or equivalently ρ_{out}) such as the quantum state purity

$$\gamma = \text{Tr}\left[\rho_{\text{out}}^2\right] \tag{7.11}$$

or its fidelity with respect to the ideal pure quantum state ρ_{in}

$$F\left(\rho_{\text{in}}, \rho_{\text{out}}\right) = \text{Tr}\left[\sqrt{\sqrt{\rho_{\text{in}}}\rho_{\text{out}}\sqrt{\rho_{\text{in}}}}\right]. \tag{7.12}$$

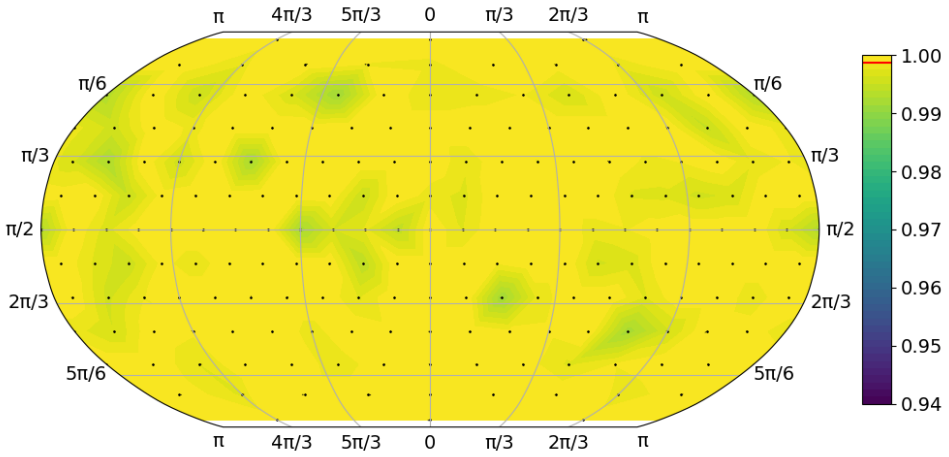
A natural way to visualise $\vec{a}_{\text{out}}$ is a point within the Bloch sphere using the relation in Equation (7.3). However, this representation is difficult to interpret without an interactive visualisation as many points within the sphere are co-located in standard orthographic projections, such as Figure 7.3. In particular, it is hard to distinguish if a given point is "on the front" or "on the back" of the sphere, it is hard to distinguish a pure state, that is represented on the surface of the Bloch sphere, from a mixed state, that is represented strictly within the Bloch sphere.

This work carefully combines three visualisation tools to address the challenge of presenting this data. The first idea is to embed the single qubit states on the two-dimensional plane using established spherical projection methods. In this work we leverage the *Robinson projection* to place the QST results on a two-dimensional plot.[1] The second idea is to leverage a heat-map on this 2-dimensional representation to visualise a key metric of interest, such as the reconstructed state's purity or fidelity. Throughout this work the colour of the plot is used to present the state's purity. The third idea is to use arrows to illustrate the locations difference between the *ideal* and *reconstructed* states, with each arrow starting at the *ideal* state and ending at the corresponding *reconstructed* state, projected onto the surface of the Bloch sphere. This communicates the rotational error that occurs in the *reconstructed* state. Overall, we call this representation of single-qubit QST the Vector Field Visualisation (VFV).

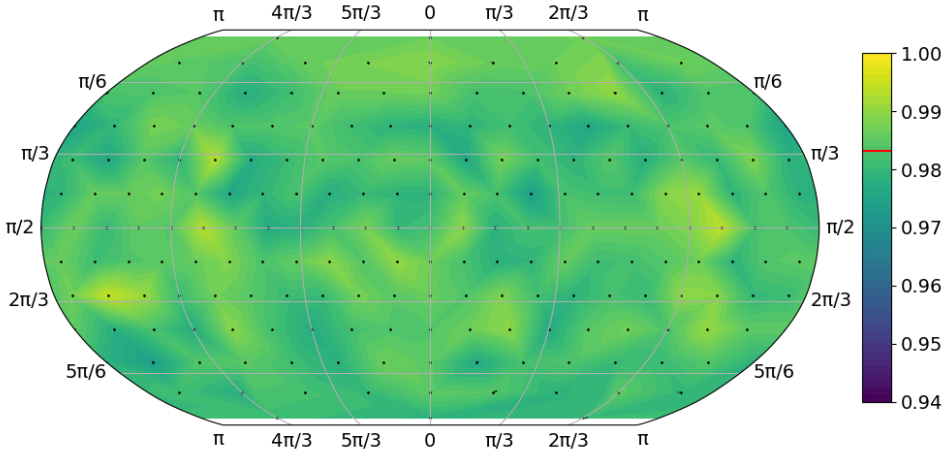
7.3.1 Vector Field Visualisation Examples

To verify the correctness of the proposed VFV approach, and to illustrate its usefulness, we begin with a series of QST studies of 200 quantum states spaced approximately equidistantly around the Bloch sphere using the `ibm_lagos` quantum chip. The results are presented in Figure 7.4. As a validation exercise, we begin by performing the full QST protocol on an ideal quantum simulator in Figure 7.4a. This procedure yields reconstructed states with very high

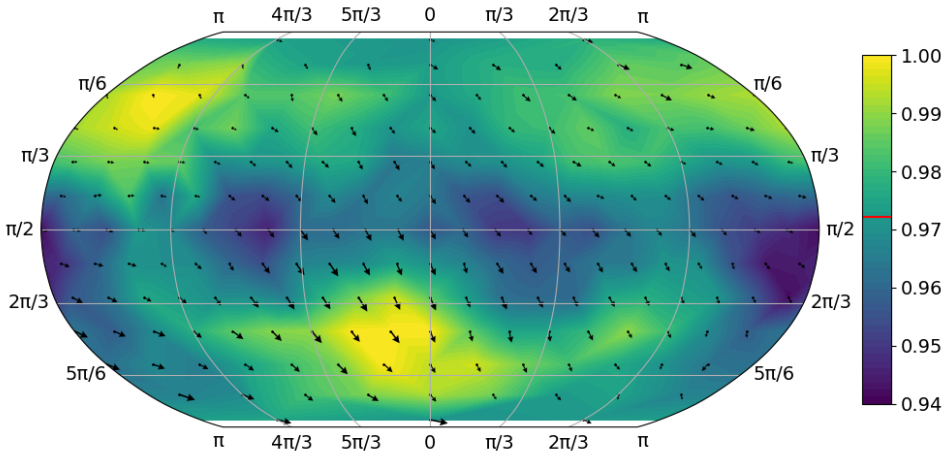
[1] Note that any 2-dimensional projection of a 3-dimensional sphere is bound to imperfectly represent some of the features. The Robinson projection is a compromise and is neither angle-preserving nor equal-area, but a balance designed to reduce the overall perceived distortion.



(a) *Data obtained on an ideal simulator.*



(b) *Data obtained on a noisy simulator configured with the calibration data of* `ibm_lagos` *at the time of hardware data collection.*



(c) *Data obtained on* `ibm_lagos`.

Figure 7.4: *Vector Field Visualisations of the reconstructed states using the single-qubit state tomography protocol from Section 7.2 using* $20,000$ *shots on qubit* 1 *of* `ibm_lagos`. *The heat-map indicates the purity of the reconstructed quantum state and the horizontal red line in the colour scale indicates the average purity of the states.*

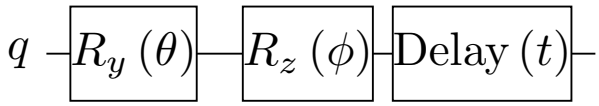


Figure 7.5: *Quantum state-preparation circuit used to visualise the effect of state degradation over time. The original protocol is obtained by setting the delay time $t = 0$. Increasing t shows the errors that arise in idle open-quantum systems. On IBM Quantum chips, t is given as a multiple of a specific time* dt. *On* `ibm_lagos`, $\mathrm{dt} = \frac{2}{9}\mathrm{ns}$.

quality (purity of 0.999 on average) with no rotational errors, confirming the correctness of the workflow's implementation. The slight variability in these results illustrates the small errors that occur due to finite samples.

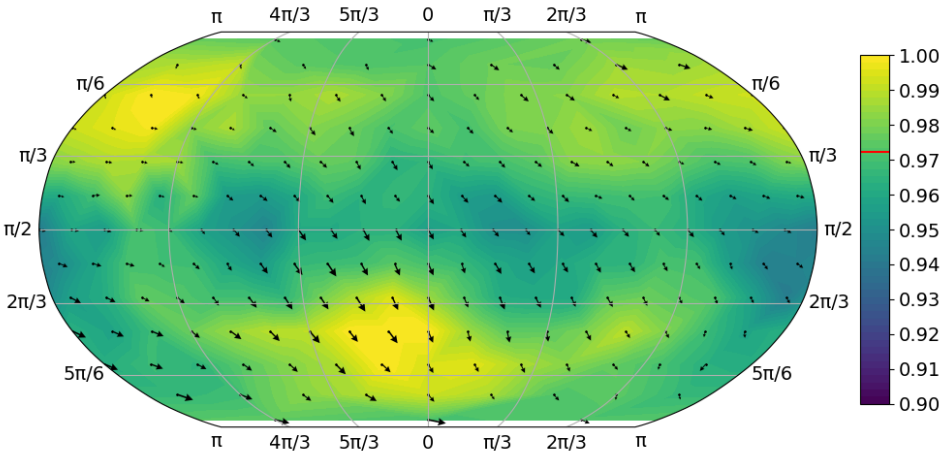
The second experiment, Figure 7.4b, consists in performing QST on data obtained from a simulator of a noisy quantum computer. IBM's Q-Hub tracks various performance properties of their qubits (e.g., gate and measurement errors) and provides an interface to perform noisy simulations using these properties. As expected, this procedure yields less consistent reconstructed states (purity 0.983, on average). Most notably, the simulated noisy qubit performance shows no rotational errors and is very homogeneous regardless of the quantum state that is being inspected.

The third and most interesting experiment, Figure 7.4c, consists in performing the state tomography protocol with real data from the `ibm_lagos` quantum chip. Three interesting observations can be made when comparing this result to the two simulations: (1) This is the first data-set where the vector field arrows are visible, indicating a type of rotational error that is not captured by the simulations; (2) The purity of the reconstructed states is more heterogeneous than the simulators; (3) the purity of the reconstructed states is similar in average to the noisy simulator (0.983 compared to 0.972) but has a notably higher standard deviation (0.004 compared to 0.013). All of these observations indicate the potential for a state-dependent error model to better capture this qubit's performance. Overall, the value of the VFV approach is highlighted by the distinct motifs that are clear in each of these figures and may provide inspiration for new qubit performance measures and noise mitigation schemes.

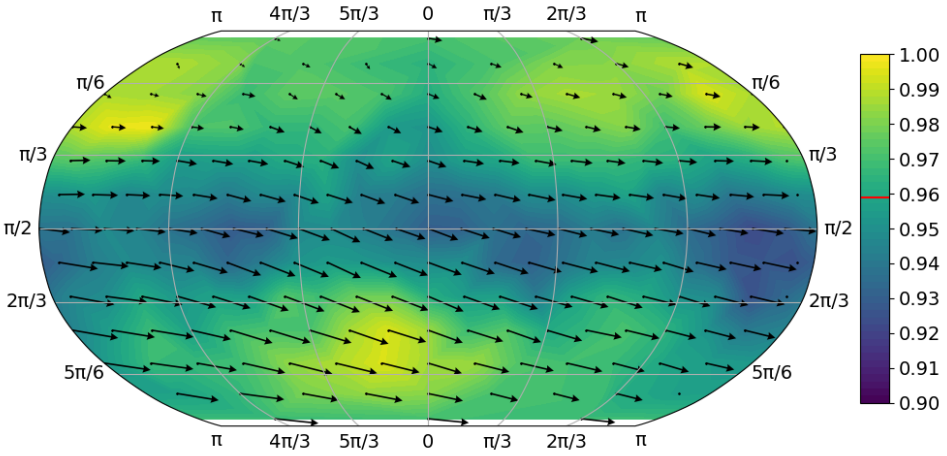
7.3.2 Visualisation of State Degradation

One of the primary uses of data visualisation tools like the VFV is to examine features of qubit performance that are not captured by current QC simulators. This can provide inspiration for which features could make simulators more faithful proxies of real-world QC hardware. To illustrate this point, this section explores the impact of adding a `delay(t)` instruction into the state preparation protocol, as shown in Figure 7.5. In this circuit the state preparation procedure used in the original protocol is now followed by a delay during which the qubit experiences the effects of an open-quantum system before the QST measurements are performed. Note that this delayed QST experiment would produce identical results in the case of *closed-system* simulations that are considered in Figure 7.4. This type of experiment is only interesting on QC hardware or simulations of open-quantum systems [220, 221].

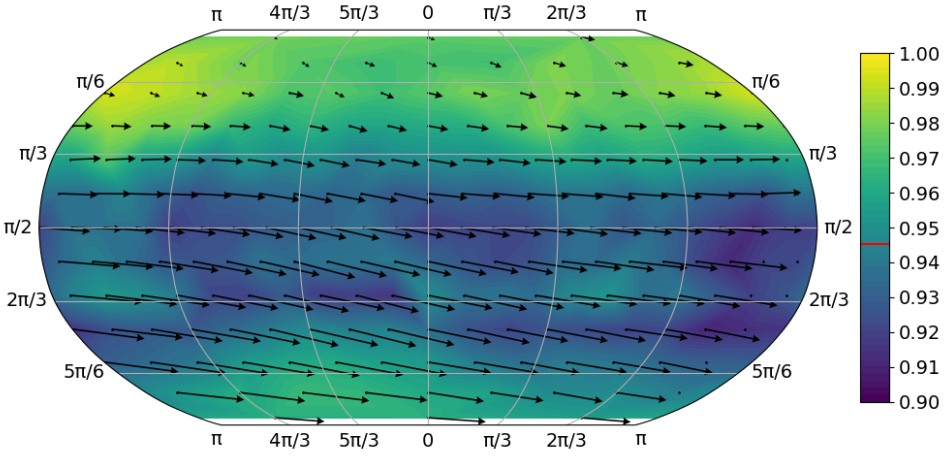
Results of the time delay experiments are shown in Figure 7.6. Two notable observations can be made from these results. First we can see that the average of the state's purity degrades steadily, starting with an average value of 0.972, degrading to 0.959 and then 0.945 as the delay duration is increased. The second observation is that the rotational error of the state is also increasing steadily and non-uniformly with time; notice how the rotational bias at the top of the VFV (i.e., close to the $|0\rangle$ state) is very different from the bottom (i.e., close to the $|1\rangle$ state). It is also surprising to see a systematic rotational shift from left to right that appears to be state-dependent. Although this study is only a proof-of-principle, it highlights the potential usefulness of VFV for designing models of open-quantum systems and provides some intuition for what decoherence *looks like* on this QC hardware.



(a) *Data obtained on* `ibm_lagos` *with a delay of* $t = 0$ dt.



(b) *Data obtained on* `ibm_lagos` *with a delay of* $t = 800$ dt.



(c) *Data obtained on* `ibm_lagos` *with a delay of* $t = 1600$ dt.

Figure 7.6: *Vector Field Visualisations of the reconstructed states of the state preparation with delay circuit using* $20,000$ *shots on qubit* 1 *of* `ibm_lagos` *using three different delay values. The heat-map indicates the purity of the reconstructed quantum state and the horizontal red line in the colour scale indicates the average purity of the states.*

7.4 Signatures of Single-Qubit Data Corruption

In this section we explore how different QST reconstruction methods can be combined to identify additional issues with the performance of individual qubits. In particular, we compare reconstructed states using the MLE method with those produced by the Linear Regression (LR) method [217]. In short, LR QST identifies a density matrix that minimises the difference between the observed and predicted measurement probabilities. For a single qubit system, the LR method solves the following optimisation task on the Bloch sphere,

$$\vec{a}_{\text{out}}^{\text{LR}} = \underset{\|\vec{a}\|\leq 1}{\arg\min} \sum_{\vec{u}} (1 + \vec{u}\cdot\vec{a} - 2p_{\vec{u}})^2. \tag{7.13}$$

Using the statistical error analysis from Section 7.2.3, we find that the 99-th percentile of the Euclidean distance error of LR is less than 0.02 for 20,000 observations, which is comparable to the MLE method.

The key insight of this section is that the difference in purity between the reconstructed states of MLE and LR should be very small, but in practice is it not always the case. Specifically, when using identical input statistics, the 99-th percentile of the Euclidean distance between the state estimates of MLE and LR is also less than 0.02 with 20,000 observations (i.e., $\mathbb{P}\left[\left|\|\vec{a}_{\text{out}}^{\text{LR}}\| - \|\vec{a}_{\text{out}}^{\text{MLE}}\|\right| \leq 0.02\right] \geq 0.99$). However, in practice, we observe that some qubits produce data where the differences between the quantum states reconstructed by MLE and LR cannot be reasonably explained by statistical fluctuations, as shown in Figure 7.7. This suggests that the the measured statistics of these qubits are corrupted after the state preparation occurs. It is likely that the elementary rotations $R_y(-\alpha)$ and $R_z(-\beta)$ used for changing measurement basis introduce *state-dependant* errors that are incompatible with the QST models considered in this work. Nevertheless, the analysis in Figure 7.7 indicates that combining multiple tomography methods can be a useful and effective tool for identifying signatures of data corruption.

7.5 Open-Source Software Implementation

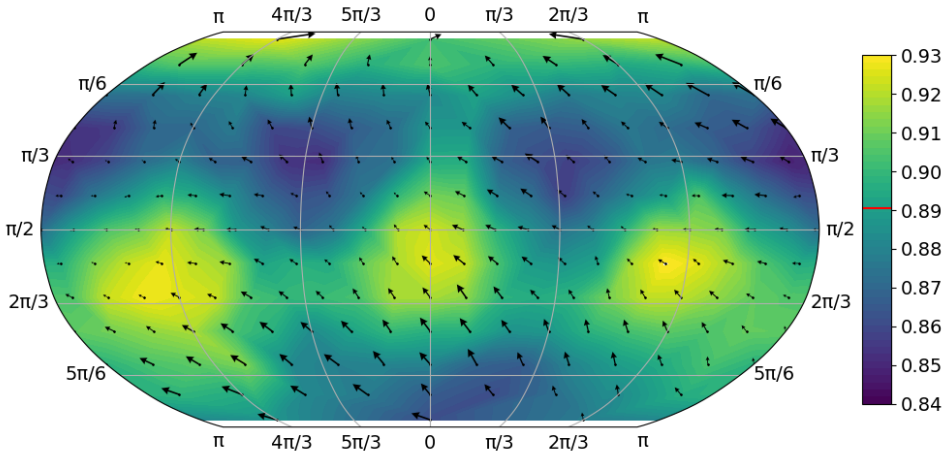
In the last couple of years, IBM's Qiskit Python package has emerged as a de facto standard for gate-based QC and can be used to access a wide variety of hardware platforms, even beyond those provided by IBM. As such, the libraries developed in this work leverage Qiskit for data collection on QC hardware. Even though Qiskit is the only supported framework for the moment, particular attention has been given to enable extending to other frameworks in the future.

The single-qubit QST protocol presented in this work is organised into two packages: `sqt` and `sqmap`. The first package, `sqt` (short for Single-Qubit Tomography), implements all the functions and interfaces to perform a single-qubit QST efficiently. In particular, it provides various single-qubit tomography basis and quantum state reconstruction methods. `sqt` also provides ways to parallelise a single-qubit experiment over all the available qubits in QC hardware for a quick assessment of the qubit performance across a full-chip. The second package, `sqmap` (short for Single-Qubit Map), provides various ways of visualising single-qubit tomography results obtained with `sqt`, such as the VFV figures presented in this work.

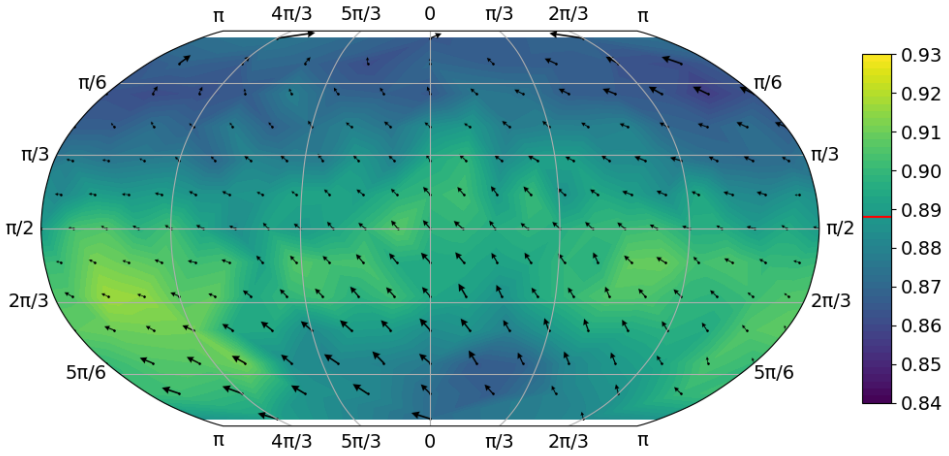
All of the QST procedures and plotting facilities used in this paper are available as open-source at `https://github.com/nelimee/sqt` and `https://github.com/nelimee/sqmap` and can be used freely by anyone to benchmark their QC platform. Installation of these packages can be done with the official Python package manager `pip`.

7.6 Conclusion

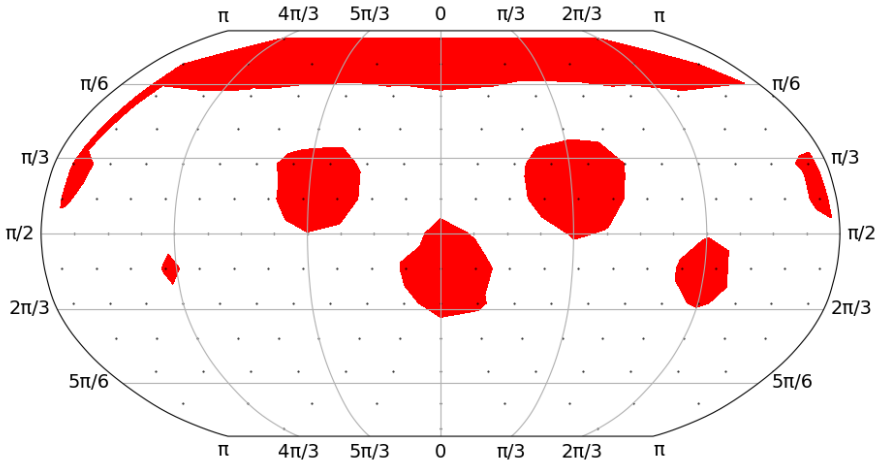
As the variety of quantum computing platforms continues to increase so does the need of tools to inspect their performance characteristics. In this work we have demonstrated that quantum



(a) *Data obtained on* `ibmq_belem` *and reconstructed with MLE.*



(b) *Data obtained on* `ibmq_belem` *and reconstructed with LR.*



(c) *Reconstructed quantum states where the absolute purity difference between MLE and LR reconstruction methods is above the* 0.02 *threshold.*

Figure 7.7: *A comparison of the quantum state reconstruction methods MLE and LR on qubit* 3 *of* `ibmq_belem`*, where the reconstruction methods do not entirely agree. Each post-processing method is given the same raw data from* `ibmq_belem`*.*

state tomography of individual qubits is a viable approach for inspecting qubit performance, although similar procedures are unlikely to scale to much larger systems due to the notable data collection requirements. Through the careful design of data collection, state reconstruction and result visualisation, this work illustrates that the proposed QCVV procedures can highlight elusive patterns in qubit performance that are difficult to capture with simpler metrics. The results indicate that there is room for improvement on the simulation models that are currently used and highlight the importance of modelling open-quantum system effects at medium time scales. A careful comparison different tomography methods also indicates that more general models for quantum state tomography should be considered to better capture the exotic effects that can be observed is current QC platforms. We hope that the proposed QCVV procedure and vector field visualisation will be a valuable tool for the quantum computing community in evaluating qubit performance and have provided the implementation as open-source to support that aim.

Part VI

Outlooks and conclusion

Chapter

8 Conclusion

Since the introduction of the Harrow-Hassidim-Lloyd (HHL) algorithm in 2008, the field of quantum computing has seen a lot of work targeting scientific computing problems. From new algorithms to drastic improvements of the software stack, the state of the quantum computing field has experienced tremendous changes. We conclude this document by restating some of the most important results obtained in this thesis and pointing out promising works and important problems related to the application of quantum computing to the field of scientific computations.

8.1 Important results

New software able to profile quantum programs. In Chapter 3 we saw that quantum circuits resulting from the implementation of quantum algorithms can contain more quantum gates than could potentially be printed in this manuscript (several billions). Such implementations often result from the nested calls of several quantum algorithms. Moreover, modern software development "best-practices" encourage the use of self-contained and short "functions" which, if followed when implementing quantum algorithms, lead to even more levels of nesting and larger call-graphs. The task of debugging and optimising these often very large implementations is made unnecessarily complex by the lack of efficient, synthetic and human-readable visualisation. We have fixed this problem by providing qprof, an open-source, multi-framework, and efficient profiler able to quickly analyse complex quantum circuits and output reports in different human-readable formats.

Quantum program compilation. We showed in Chapter 5 that quantum program compilers can be substantially improved by taking into account the most up to date hardware calibrations and using gates that do not change the qubit ordering as the `Bridge` gate. New work tackling the problem of compiling quantum circuits introduced problem-specific compilation algorithms, able to generate optimised quantum circuits for specific applications. But as noted in Section 4.6.2, most of the compilers targeting quantum programs are only able to work on "flattened" quantum circuits which (1) is radically different of the approach took by classical compilers and (2) will lead to large inefficiencies for large quantum circuits.

Improved quantum hardware characterisation. Chapter 7 showed a new method to reliably visualise errors happening on single-qubit operations. In this work, we show that there is a measurable and consistent noise affecting the qubit state that is not accounted for in hardware calibration and hardware noise models. Such works are the first step for an improved understanding of the errors a quantum hardware can be subject to. It has the potential to lead to improvements of the error models used to simulate quantum noise and can help devising ways to correct these errors directly at the hardware level, either by tuning each qubit or by fixing the physical process causing these errors.

Implementation of quantum "oracles". When analysing the quantum wave equation solver implementation presented in Chapter 3 we realised that "oracles", quantum circuits that encodes

classical information such as a matrix entries in a format usable by a quantum computer, were the major source of inefficiencies of the implementation. Aside from the fact that the tasks of prior analysis, implementation and debugging leading to a correct oracle implementation were very time consuming and extremely complex, our analysis with qprof showed that our initial implementation was inefficient. The implementation only reached a "reasonably optimised" state after a few manual iterations, which increased again the time spent implementing the oracle. These observations indicate that automatic generation approaches might be worth exploring. As noted in Section 3.5, a few work (HODL [112] and the Classiq start-up [113]) have already started to explore these approaches, and more work should be performed to evaluate the quality of the generated implementations.

Improvements in variational quantum algorithms. As shown in Chapter 6, variational quantum algorithms are still facing challenges preventing them from scaling to bigger problem sizes. Part of these challenges are due to hardware imprecisions and can be partly mitigated with the help of quantum error mitigation techniques such as dynamical decoupling or measurement error mitigation. But the Barren plateau phenomenon is imposing drastic requirements to both the cost function and the ansatz used, as well as good initialisation strategies. The Barren plateau phenomenon implications on the ability of variational quantum algorithms to solve large problem is a crucial research direction to continue improving our understanding of variational algorithms.

8.2 Research perspectives

The work presented in this manuscript improved over the state of the art in several domains, from compilation of quantum programs to the analysis of complex quantum programs. This section present a few of the research perspectives that we consider to be interesting paths to explore in the future.

"Hierarchical" quantum compilation. Most of the state of the art compilers for quantum programs have been designed to compile a "flattened" quantum circuit, only containing hardware-native quantum gates. In order to perform the compilation, they start the process by "unrolling" the quantum subroutines, a step also called "inlining" in classical computing and that consist in replacing a call to a given subroutine by its implementation.

This behaviour is beneficial for very small quantum circuits as it allows the compiler to have a global vision of all the quantum gates present in the circuit and enable the possibility to perform even more optimisations. But such an approach for larger circuits can lead to compiling the exact same sequence of gates several times (i.e., missed opportunities) and makes it impossible to use some tools (such as qprof) to analyse the compiled program.

We think that a hierarchical compiler able to compile quantum programs without "flattening" them unconditionally, inspired from what is done in classical compiler with the processes of selective inlining of functions, will improve drastically the performance of current quantum compilers and may even allow these compilers to improve the overall optimisation by selectively spending more computational time optimising some high-cost routines.

New initiatives such as the Quantum Intermediate Representation (QIR) or OpenQASM 3.0 that try to introduce new standards to represent quantum computation will help standardising compilers and might even be able to leverage existing compiler infrastructures like LLVM to compile quantum programs.

Improvements to qprof. The qprof and qcw tools got a few very positive feedback from the quantum computing community but have some limitations. One of the first and most evident

improvement that would benefit to the whole community would be to increase the number of frameworks supported by qcw, the quantum circuit wrapper. Implementing a plugin to support OpenQASM 3.0 would allow qprof to support all the quantum computing frameworks capable of exporting to OpenQASM 3.0, which we think will be the case of most frameworks as the OpenQASM 3.0 language will probably become a standard.

Adding new exporters to qprof is also a very interesting as it would be directly usable for any quantum computing framework already implemented. We particularly think that Flamegraphs would be a very good improvement.

Finally, we would like to improve the internals of qprof to allow the benchmark of non-additive quantities such as an estimate of the error-rate of each subroutine or the topology required to execute each subroutine.

Improvement of the development ecosystem. When performing the different implementations presented and analysed in Chapters 3 and 6, we found that a lot of the classical tools made to help developers were missing to the quantum ecosystem. For example, there is currently no standard way to distribute implementations, analogous to libraries in classical computing. This is mostly due to the fact that there is no standard interface or language in the quantum computing field. The introduction and universal recognition of a standard will take time, but will help tremendously the field and community.

Answering the question of inputs and outputs. The problem of constructing oracles is the visible face of a greater and more fundamental problem in quantum computing: how can we input classical data into a quantum computer? Here classical data can be anything from the coefficients of a matrix (as studied in Chapter 3) or the right-hand side of a system of linear equations (as needed in the VQLS algorithm from Chapter 6).

A very related problem consist in performing the opposite operation: how can we "output" data from a quantum computer? Or more precisely, what classical problems only requires the type of information that can be extracted from a quantum computer at a reasonable cost?

We think that these questions of input and output are crucial to answer for the future of quantum computing, even more importantly for its potential applications to scientific computing in which one needs to encode right-hand sides of systems of linear equations, initial or boundary conditions of partial differential equations or even constraints for optimisation problems.

Bibliography

References for Chapter 1: Introduction to Quantum Computing

[1] Max Planck and Morton Masius. *The theory of heat radiation*. "Author's bibliography": p. 216-217. Philadelphia: P. Blakiston's Son & Co., 1914, xiv p., 1 l., 225 p. *Cited on page 5.*

[2] W. Heisenberg. "Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik". In: *Zeitschrift fur Physik* 43.3-4 (Mar. 1927), pp. 172–198. DOI: 10.1007/BF01397280 *Cited on page 6.*

[3] Paul Adrien Maurice Dirac. *The principles of quantum mechanics*. English. Oxford: Clarendon Pr., 1930 *Cited on page 6.*

[4] John von Neumann. "Mathematical Foundations of Quantum Mechanics". In: 1955 *Cited on pages 6, 8.*

[5] A. M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265. DOI: https://doi.org/10.1112/plms/s2-42.1.230. eprint: https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/plms/s2-42.1.230 *Cited on page 6.*

[6] 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* 22.5 (May 1980), pp. 563–591. DOI: 10.1007/bf01011339 *Cited on page 6.*

[7] Paul Benioff. "Quantum mechanical hamiltonian models of turing machines". In: *Journal of Statistical Physics* 29.3 (Nov. 1982), pp. 515–546. DOI: 10.1007/bf01342185 *Cited on page 6.*

[8] Richard P. Feynman. "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21.6 (June 1982), pp. 467–488. ISSN: 1572-9575. DOI: 10.1007/BF02650179 *Cited on page 6.*

[9] David Deutsch. "Quantum theory, the Church-Turing principle and the universal quantum computer". In: *Proceedings of the Royal Society A* 400.1818 (1985), pp. 97–117. DOI: 10.1098/rspa.1985.0070. eprint: https://royalsocietypublishing.org/doi/10.1098/rspa.1985.0070 *Cited on pages 6, 7.*

[10] David Deutsch and Richard Jozsa. "Rapid solution of problems by quantum computation". In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439.1907 (Dec. 1992), pp. 553–558. DOI: 10.1098/rspa.1992.0167 *Cited on page 6.*

[11] Lov K. Grover. "A fast quantum mechanical algorithm for database search". In: (May 1996). eprint: quant-ph/9605043v3 *Cited on page 6.*

[12] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM J. Sci. Statist. Comput. 26 (1997) 1484* (Sept. 1995). doi: https://doi.org/10.1137/S0097539795293172. eprint: quant-ph/9508027v2 *Cited on page 6.*

[13] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. "Quantum Algorithm for Linear Systems of Equations". In: *Physical Review Letters* 103 (15 Oct. 2009). Phys. Rev. Lett. vol. 15, no. 103, pp. 150502 (2009). DOI: 10.1103/PhysRevLett.103.150502. eprint: 0811.3171v3 *Cited on pages 6, 26, 31, 122.*

[14] Edward Farhi et al. "A Quantum Adiabatic Evolution Algorithm Applied to Random Instances of an NP-Complete Problem". In: *Science* 292.5516 (2001), pp. 472–475. DOI: 10.1126/science.1057726. eprint: https://www.science.org/doi/pdf/10.1126/science.1057726 *Cited on page 7.*

[15] 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://doi.org/10.1137/080734479 *Cited on page 7.*

[16] W. van Dam, M. Mosca, and U. Vazirani. "How powerful is adiabatic quantum computation?" In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. 2001, pp. 279–287. DOI: 10.1109/SFCS.2001.959902 *Cited on page 7.*

[17] Andris Ambainis and Oded Regev. *An Elementary Proof of the Quantum Adiabatic Theorem*. 2004. DOI: 10.48550/ARXIV.QUANT-PH/0411152 *Cited on page 7.*

[18] Robert Raussendorf and Hans J. Briegel. "A One-Way Quantum Computer". In: *Phys. Rev. Lett.* 86 (22 May 2001), pp. 5188–5191. DOI: 10.1103/PhysRevLett.86.5188 *Cited on page 7.*

[19] Hans J. Briegel and Robert Raussendorf. "Persistent Entanglement in Arrays of Interacting Particles". In: *Phys. Rev. Lett.* 86 (5 Jan. 2001), pp. 910–913. DOI: 10.1103/PhysRevLett.86.910 *Cited on page 7.*

[20] Robert Raussendorf, Daniel E. Browne, and Hans J. Briegel. "Measurement-based quantum computation on cluster states". In: *Phys. Rev. A* 68 (2 Aug. 2003), p. 022312. DOI: 10.1103/PhysRevA.68.022312 *Cited on page 7.*

References for Chapter 2: Scientific computing and quantum computing

[13] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. "Quantum Algorithm for Linear Systems of Equations". In: *Physical Review Letters* 103 (15 Oct. 2009). Phys. Rev. Lett. vol. 15, no. 103, pp. 150502 (2009). DOI: 10.1103/PhysRevLett.103.150502. eprint: 0811.3171v3 *Cited on pages 6, 26, 31, 122.*

[21] Peter Lynch. "The origins of computer weather prediction and climate modeling". In: *Journal of Computational Physics* 227.7 (2008). Predicting weather, climate and extreme events, pp. 3431–3444. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2007.02.034 *Cited on page 18.*

[22] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Jan. 2003. DOI: 10.1137/1.9780898718003 *Cited on pages 21, 22.*

[23] Stefan Heinz. "A review of hybrid RANS-LES methods for turbulent flows: Concepts and applications". In: *Progress in Aerospace Sciences* 114 (2020), p. 100597. ISSN: 0376-0421. DOI: https://doi.org/10.1016/j.paerosci.2019.100597 *Cited on page 21.*

[24] TOP500 ranking. *Supercomputer Fugaku*. https://www.top500.org/system/179807/. Accessed: 2022-09-15. Sept. 2022 *Cited on page 24.*

[25] Dominic W. Berry et al. "Efficient Quantum Algorithms for Simulating Sparse Hamiltonians". In: *Communications in Mathematical Physics* 270 (2 Jan. 2007). Communications in Mathematical Physics 270, 359 (2007), pp. 359–371. DOI: 10.1007/s00220-006-0150-x. eprint: quant-ph/0508139v2 *Cited on pages 24, 25, 32–35, 51, 52.*

[26] Andrew M. Childs and Robin Kothari. "Simulating Sparse Hamiltonians with Star Decompositions". In: *Theory of Quantum Computation, Communication, and Cryptography*. Ed. by Wim van Dam, Vivien M. Kendon, and Simone Severini. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 94–103. ISBN: 978-3-642-18073-6 *Cited on page 25.*

[27] Stuart Hadfield and Anargyros Papageorgiou. "Divide and conquer approach to quantum Hamiltonian simulation". In: *New Journal of Physics* 20.4 (Apr. 2018), p. 043003. DOI: 10.1088/1367-2630/aab1ef *Cited on page 25.*

[28] Andrew M. Childs, Aaron Ostrander, and Yuan Su. "Faster quantum simulation by randomization". In: *Quantum* 3 (Sept. 2019), p. 182. ISSN: 2521-327X. DOI: 10.22331/q-2019-09-02-182 *Cited on page 25.*

[29] Andrew M. Childs et al. "Toward the first quantum simulation with quantum speedup". In: *Proceedings of the National Academy of Sciences* 115.38 (Sept. 2018), pp. 9456–9461. ISSN: 1091-6490. DOI: 10.1073/pnas.1801723115 *Cited on pages 25, 70.*

[30] Dominic W. Berry and Andrew M. Childs. "Black-box Hamiltonian Simulation and Unitary Implementation". In: *Quantum Info. Comput.* 12.1-2 (Jan. 2012). Quantum Information and Computation 12, 29 (2012), pp. 29–62. ISSN: 1533-7146. DOI: 10.26421/QIC12.1-2. eprint: 0910.4157v4 *Cited on pages 25, 32, 34, 35.*

[31] Dominic W. Berry et al. "Exponential Improvement in Precision for Simulating Sparse Hamiltonians". In: *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*. STOC '14. New York, New York: Association for Computing Machinery, 2014, pp. 283–292. ISBN: 9781450327107. DOI: 10.1145/2591796.2591854 *Cited on pages 25, 34, 35.*

[32] Dominic W. Berry et al. "Simulating Hamiltonian Dynamics with a Truncated Taylor Series". In: *Physical Review Letters* 114 (9 Mar. 2015). Phys. Rev. Lett. 114, 090502 (2015). DOI: 10.1103/PhysRevLett.114.090502. eprint: 1412.4687v1 *Cited on pages 25, 32, 34, 35.*

[33] Dominic W. Berry, Andrew M. Childs, and Robin Kothari. "Hamiltonian Simulation with Nearly Optimal Dependence on all Parameters". In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. Proceedings of the 56th IEEE Symposium on Foundations of Computer Science (FOCS 2015), pp. 792-809 (2015). Oct. 2015, pp. 792–809. DOI: 10.1109/FOCS.2015.54. eprint: 1501.01715v3 *Cited on pages 25, 33–35.*

[34] Guang Hao Low and Isaac L. Chuang. "Optimal Hamiltonian Simulation by Quantum Signal Processing". In: *Physical Review Letters* 118 (1 Jan. 2017). Phys. Rev. Lett. 118, 010501 (2017). DOI: 10.1103/PhysRevLett.118.010501. eprint: 1606.02685v2 *Cited on pages 25, 32, 34, 35.*

[35] Guang Hao Low and Isaac L. Chuang. "Hamiltonian Simulation by Qubitization". In: *Quantum* 3 (None July 2019). DOI: 10.22331/q-2019-07-12-163 *Cited on page 25.*

[36] Andris Ambainis. "Variable time amplitude amplification and quantum algorithms for linear algebra problems". In: *STACS'12 (29th Symposium on Theoretical Aspects of Computer Science)*. Ed. by Thomas Wilke Christoph Dürr. Vol. 14. Paris, France: LIPIcs, Feb. 2012, pp. 636–647 *Cited on page 26.*

[37] B. D. Clader, B. C. Jacobs, and C. R. Sprouse. "Preconditioned quantum linear system algorithm". In: (Jan. 2013). Phys. Rev. Lett. 110, 250504 (2013). DOI: 10.1103/PhysRevLett.110.250504. eprint: 1301.2340v4 *Cited on page 26.*

[38] Scott Aaronson. "Read the fine print". In: *Nature Physics* 11.4 (Apr. 2015), pp. 291–293. DOI: 10.1038/nphys3272 *Cited on pages 26, 122.*

[39] Andrew M. Childs and Nathan Wiebe. "Hamiltonian Simulation Using Linear Combinations of Unitary Operations". In: (Feb. 2012). Quantum Information and Computation 12, 901-924 (2012). DOI: 10.26421/QIC12.11-12. eprint: 1202.5822v1 *Cited on pages 26, 34, 35.*

[40] Leonard Wossnig, Zhikuan Zhao, and Anupam Prakash. "Quantum Linear System Algorithm for Dense Matrices". In: *Phys. Rev. Lett.* 120 (5 Jan. 2018), p. 050502. DOI: 10.1103/PhysRevLett.120.050502 *Cited on page 26.*

[41] Iordanis Kerenidis and Anupam Prakash. "Quantum Recommendation Systems". In: *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*. Ed. by Christos H. Papadimitriou. Vol. 67. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 49:1–49:21. ISBN: 978-3-95977-029-3. DOI: 10.4230/LIPIcs.ITCS.2017.49 *Cited on page 26.*

[42] Yi ğit Suba ş ı, Rolando D. Somma, and Davide Orsucci. "Quantum Algorithms for Systems of Linear Equations Inspired by Adiabatic Quantum Computing". In: *Phys. Rev. Lett.* 122 (6 Feb. 2019), p. 060504. DOI: 10.1103/PhysRevLett.122.060504 *Cited on page 26.*

[43] Almudena Carrera Vazquez, Ralf Hiptmair, and Stefan Woerner. "Enhancing the Quantum Linear Systems Algorithm Using Richardson Extrapolation". In: *ACM Transactions on Quantum Computing* 3.1 (Jan. 2022). ISSN: 2643-6809. DOI: 10.1145/3490631 *Cited on page 26.*

[44] Wentao Qi et al. *Quantum algorithms for matrix operations and linear systems of equations*. 2022. DOI: 10.48550/ARXIV.2202.04888 *Cited on page 26.*

[45] Dong An and Lin Lin. "Quantum Linear System Solver Based on Time-Optimal Adiabatic Quantum Computing and Quantum Approximate Optimization Algorithm". In: *ACM Transactions on Quantum Computing* 3.2 (Mar. 2022). ISSN: 2643-6809. DOI: 10.1145/3498331 *Cited on page 26.*

[46] Xiaosi Xu et al. "Variational algorithms for linear algebra". In: *Science Bulletin* 66.21 (2021), pp. 2181–2188. ISSN: 2095-9273. DOI: https://doi.org/10.1016/j.scib.2021.06.023 *Cited on page 26.*

[47] Hsin-Yuan Huang, Kishor Bharti, and Patrick Rebentrost. "Near-term quantum algorithms for linear systems of equations". In: (Sept. 2019). arXiv: https://arxiv.org/abs/1909.07344v1. eprint: 1909.07344v1 *Cited on page 26.*

[48] Carlos Bravo-Prieto et al. *Variational Quantum Linear Solver*. 2020. arXiv: 1909.05820 [quant-ph] *Cited on pages 26, 121, 127.*

[49] Arthur Pesah. *Quantum Algorithms for Solving Partial Differential Equations*. https://arthurpesah.me/assets/pdf/case-study-quantum-algorithms-pde.pdf. Accessed: 2022-09-15. Mar. 2020 *Cited on page 27.*

[50] Sarah K. Leyton and Tobias J. Osborne. "A quantum algorithm to solve nonlinear differential equations". In: (2008). DOI: 10.48550/ARXIV.0812.4423 *Cited on pages 27, 32.*

[51] Dominic W Berry. "High-order quantum algorithm for solving linear differential equations". In: *Journal of Physics A: Mathematical and Theoretical* 47.10 (Feb. 2014), p. 105301. DOI: 10.1088/1751-8113/47/10/105301 *Cited on pages 27, 32.*

[52] Dominic W. Berry et al. "Quantum Algorithm for Linear Differential Equations with Exponentially Improved Dependence on Precision". In: *Communications in Mathematical Physics* 356 (3 Dec. 2017). Communications in Mathematical Physics 356, 1057-1081 (2017), pp. 1057–1081. DOI: 10.1007/s00220-017-3002-y. eprint: 1701.03684v2 *Cited on page 28.*

[53] Yudong Cao et al. "Quantum algorithm and circuit design solving the Poisson equation". In: *New Journal of Physics* 15.1 (Jan. 2013), p. 013021. DOI: 10.1088/1367-2630/15/1/013021 *Cited on pages 28, 32.*

[54] Andrew M. Childs, Jin-Peng Liu, and Aaron Ostrander. "High-precision quantum algorithms for partial differential equations". In: (Feb. 2020). eprint: 2002.07868v1 *Cited on page 28.*

[55] Shengbin Wang et al. "Quantum fast Poisson solver: the algorithm and complete and modular circuit design". In: *Quantum Information Processing* 19.6 (Apr. 2020). DOI: 10.1007/s11128-020-02669-7 *Cited on page 28.*

[56] Pedro C. S. Costa, Stephen Jordan, and Aaron Ostrander. "Quantum algorithm for simulating the wave equation". In: *Physical Review A* 99 (1 Jan. 2019). Phys. Rev. A 99, 012323 (2019). DOI: 10.1103/PhysRevA.99.012323. eprint: 1711.05394v1 *Cited on pages 28, 33, 37, 38, 52, 54, 58, 59.*

[57] I. Y. Dodin and E. A. Startsev. *On applications of quantum computing to plasma simulations*. 2020. DOI: 10.48550/ARXIV.2005.14369 *Cited on page 28.*

[58] Alexander Engel, Graeme Smith, and Scott E. Parker. "Quantum algorithm for the Vlasov equation". In: *Phys. Rev. A* 100 (6 Dec. 2019), p. 062315. DOI: 10.1103/PhysRevA.100.062315 *Cited on page 28.*

[59] Frank Gaitan. "Finding flows of a Navier-Stokes fluid through quantum computing". In: *npj Quantum Information* 6.1 (July 2020). DOI: 10.1038/s41534-020-00291-0 *Cited on page 28.*

[60] KP Griffin et al. *Investigation of quantum algorithms for direct numerical simulation of the Navier-Stokes equations*. 2019 *Cited on page 28.*

[61] Javier Gonzalez-Conde et al. *Simulating option price dynamics with exponential quantum speedup*. 2021. DOI: 10.48550/ARXIV.2101.04023 *Cited on page 28.*

[62] Ashley Montanaro and Sam Pallister. "Quantum algorithms and the finite element method". In: (Dec. 2015). Phys. Rev. A 93, 032324 (2016). DOI: 10.1103/PhysRevA.93.032324. eprint: 1512.05903v2 *Cited on page 28.*

[63] Andrew M. Childs and Jin-Peng Liu. "Quantum spectral methods for differential equations". In: (Jan. 2019). eprint: 1901.00961v1 *Cited on page 28.*

[64] Paula García-Molina, Javier Rodríguez-Mediavilla, and Juan José García-Ripoll. "Quantum Fourier analysis for multivariate functions and applications to a class of Schrödinger-type partial differential equations". In: *Phys. Rev. A* 105 (1 Jan. 2022), p. 012433. DOI: 10.1103/PhysRevA.105.012433 *Cited on page 28.*

[65] Michael Lubasch et al. "Variational quantum algorithms for nonlinear problems". In: *Phys. Rev. A* 101 (1 Jan. 2020), p. 010301. DOI: 10.1103/PhysRevA.101.010301 *Cited on page 28.*

[66] Fernando G. S. L. Brandao and Krysta Svore. *Quantum Speed-ups for Semidefinite Programming*. 2016. DOI: 10.48550/ARXIV.1609.05537 *Cited on page 28.*

[67] Joran van Apeldoorn et al. "Quantum SDP-Solvers: Better upper and lower bounds". In: (May 2017). In 58th IEEE Symposium on Foundations of Computer Science (FOCS 2017), pp.403-414. DOI: 10.1109/FOCS.2017.44. eprint: 1705.01843v3 *Cited on page 28.*

[68] Fernando G. S. L. Brandão et al. *Quantum SDP Solvers: Large Speed-ups, Optimality, and Applications to Quantum Learning*. 2017. DOI: 10.48550/ARXIV.1710.02581 *Cited on page 28.*

[69] Joran van Apeldoorn and András Gilyén. "Improvements in Quantum SDP-Solving with Applications". In: *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Ed. by Christel Baier et al. Vol. 132. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 99:1–99:15. ISBN: 978-3-95977-109-2. DOI: 10.4230/LIPIcs.ICALP.2019.99 *Cited on page 28.*

[70] Joran van Apeldoorn et al. "Quantum SDP-Solvers: Better upper and lower bounds". In: *Quantum* 4 (Feb. 2020), p. 230. ISSN: 2521-327X. DOI: 10.22331/q-2020-02-14-230 *Cited on page 28.*

[71] Iordanis Kerenidis and Anupam Prakash. "A Quantum Interior Point Method for LPs and SDPs". In: *ACM Transactions on Quantum Computing* 1.1 (Oct. 2020). ISSN: 2643-6809. DOI: 10.1145/3406306 *Cited on page 28.*

[72] Brandon Augustino et al. *Quantum Interior Point Methods for Semidefinite Optimization*. 2021. DOI: 10.48550/ARXIV.2112.06025 *Cited on page 28.*

[73] Andrew Lucas. "Ising formulations of many NP problems". In: *Frontiers in Physics* 2 (2014). ISSN: 2296-424X. DOI: 10.3389/fphy.2014.00005 *Cited on page 28.*

[74] Sheir Yarkoni et al. *Quantum Annealing for Industry Applications: Introduction and Review*. 2021. DOI: 10.48550/ARXIV.2112.07491 *Cited on page 28.*

References for Chapter 3: PDE solver

[13] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. "Quantum Algorithm for Linear Systems of Equations". In: *Physical Review Letters* 103 (15 Oct. 2009). Phys. Rev. Lett. vol. 15, no. 103, pp. 150502 (2009). DOI: 10.1103/PhysRevLett.103.150502. eprint: 0811.3171v3 *Cited on pages 6, 26, 31, 122.*

[25] Dominic W. Berry et al. "Efficient Quantum Algorithms for Simulating Sparse Hamiltonians". In: *Communications in Mathematical Physics* 270 (2 Jan. 2007). Communications in Mathematical Physics 270, 359 (2007), pp. 359–371. DOI: 10.1007/s00220-006-0150-x. eprint: quant-ph/0508139v2 *Cited on pages 24, 25, 32–35, 51, 52.*

[30] Dominic W. Berry and Andrew M. Childs. "Black-box Hamiltonian Simulation and Unitary Implementation". In: *Quantum Info. Comput.* 12.1-2 (Jan. 2012). Quantum Information and Computation 12, 29 (2012), pp. 29–62. ISSN: 1533-7146. DOI: 10.26421/QIC12.1-2. eprint: 0910.4157v4 *Cited on pages 25, 32, 34, 35.*

[31] Dominic W. Berry et al. “Exponential Improvement in Precision for Simulating Sparse Hamiltonians”. In: *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*. STOC ’14. New York, New York: Association for Computing Machinery, 2014, pp. 283–292. ISBN: 9781450327107. DOI: 10.1145/2591796.2591854 *Cited on pages 25, 34, 35.*

[32] Dominic W. Berry et al. “Simulating Hamiltonian Dynamics with a Truncated Taylor Series”. In: *Physical Review Letters* 114 (9 Mar. 2015). Phys. Rev. Lett. 114, 090502 (2015). DOI: 10.1103/PhysRevLett.114.090502. eprint: 1412.4687v1 *Cited on pages 25, 32, 34, 35.*

[33] Dominic W. Berry, Andrew M. Childs, and Robin Kothari. “Hamiltonian Simulation with Nearly Optimal Dependence on all Parameters”. In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. Proceedings of the 56th IEEE Symposium on Foundations of Computer Science (FOCS 2015), pp. 792-809 (2015). Oct. 2015, pp. 792–809. DOI: 10.1109/FOCS.2015.54. eprint: 1501.01715v3 *Cited on pages 25, 33–35.*

[34] Guang Hao Low and Isaac L. Chuang. “Optimal Hamiltonian Simulation by Quantum Signal Processing”. In: *Physical Review Letters* 118 (1 Jan. 2017). Phys. Rev. Lett. 118, 010501 (2017). DOI: 10.1103/PhysRevLett.118.010501. eprint: 1606.02685v2 *Cited on pages 25, 32, 34, 35.*

[39] Andrew M. Childs and Nathan Wiebe. “Hamiltonian Simulation Using Linear Combinations of Unitary Operations”. In: (Feb. 2012). Quantum Information and Computation 12, 901-924 (2012). DOI: 10.26421/QIC12.11-12. eprint: 1202.5822v1 *Cited on pages 26, 34, 35.*

[50] Sarah K. Leyton and Tobias J. Osborne. “A quantum algorithm to solve nonlinear differential equations”. In: (2008). DOI: 10.48550/ARXIV.0812.4423 *Cited on pages 27, 32.*

[51] Dominic W Berry. “High-order quantum algorithm for solving linear differential equations”. In: *Journal of Physics A: Mathematical and Theoretical* 47.10 (Feb. 2014), p. 105301. DOI: 10.1088/1751-8113/47/10/105301 *Cited on pages 27, 32.*

[53] Yudong Cao et al. “Quantum algorithm and circuit design solving the Poisson equation”. In: *New Journal of Physics* 15.1 (Jan. 2013), p. 013021. DOI: 10.1088/1367-2630/15/1/013021 *Cited on pages 28, 32.*

[56] Pedro C. S. Costa, Stephen Jordan, and Aaron Ostrander. “Quantum algorithm for simulating the wave equation”. In: *Physical Review A* 99 (1 Jan. 2019). Phys. Rev. A 99, 012323 (2019). DOI: 10.1103/PhysRevA.99.012323. eprint: 1711.05394v1 *Cited on pages 28, 33, 37, 38, 52, 54, 58, 59.*

[75] Adrien Suau, Gabriel Staffelbach, and Henri Calandra. “Practical Quantum Computing: Solving the Wave Equation Using a Quantum Approach”. In: *ACM Transactions on Quantum Computing* 2.1 (Feb. 2021). ISSN: 2643-6809. DOI: 10.1145/3430030. arXiv: 2003.12458 [quant-ph] *Cited on pages 31, 86, 89.*

[76] M.J. Werner and P.D. Drummond. “Robust Algorithms for Solving Stochastic Partial Differential Equations”. In: *Journal of Computational Physics* 132.2 (1997), pp. 312–326. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1996.5638 *Cited on page 32.*

[77] Artur Scherer et al. “Concrete resource analysis of the quantum linear-system algorithm used to compute the electromagnetic scattering cross section of a 2D target”. In: *Quantum Information Processing* 16 (3 Mar. 2017). Quantum Inf Process (2017) 16: 60. DOI: 10.1007/s11128-016-1495-5. eprint: 1505.06552v2 *Cited on pages 32, 34, 122.*

[78] Andrew M. Childs et al. “Toward the first quantum simulation with quantum speedup”. In: *Proceedings of the National Academy of Sciences* 115 (38 Sept. 2018). Proceedings of the National Academy of Sciences 115, 9456-9461 (2018), pp. 9456–9461. DOI: 10.1073/pnas.1801723115. eprint: 1711.10980v1 *Cited on pages 32–34, 36, 56, 58, 66.*

[79] Graeme Robert Ahokas. “Improved Algorithms for Approximate Quantum Fourier Transforms and Sparse Hamiltonian Simulations”. mastersthesis. University of Calgary, 2004. DOI: 10.11575/PRISM/22839 *Cited on pages 32–36, 40, 41, 48, 51, 62, 65.*

[80] Guang Hao Low. “Hamiltonian simulation with nearly optimal dependence on spectral norm”. In: (July 2018). eprint: 1807.03967v1 *Cited on pages 32, 34, 35.*

[81] Andrew M. Childs and Robin Kothari. “Simulating Sparse Hamiltonians with Star Decompositions”. In: *Theory of Quantum Computation, Communication, and Cryptography*. Theory of Quantum Computation, Communication, and Cryptography (TQC 2010), Lecture Notes in Computer Science 6519, pp. 94-103 (2011). Springer Berlin Heidelberg, Mar. 2011, pp. 94–103. DOI: 10.1007/978-3-642-18073-6_8. eprint: 1003.3683v2 *Cited on pages 32, 34–36.*

[82] Andrew M. Childs, Aaron Ostrander, and Yuan Su. *Faster quantum simulation by randomization*. misc. Only arXiv eprint available. May 2018. eprint: 1805.08385v1 *Cited on page 32.*

[83] Jeongwan Haah et al. “Quantum Algorithm for Simulating Real Time Evolution of Lattice Hamiltonians”. In: *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. Oct. 2018, pp. 350–360. DOI: 10.1109/FOCS.2018.00041 *Cited on page 32.*

[84] Stuart Hadfield and Anargyros Papageorgiou. “Divide and conquer approach to quantum Hamiltonian simulation”. In: *New Journal of Physics* 20 (4 Apr. 2018). DOI: 10.1088/1367-2630/aab1ef *Cited on page 32.*

[85] Maria Kieferova, Artur Scherer, and Dominic Berry. *Simulating the dynamics of time-dependent Hamiltonians with a truncated Dyson series*. misc. Only eprint on arXiv. May 2018. eprint: 1805.00582v1 *Cited on pages 32, 34, 35.*

[86] Guang Hao Low and Isaac L. Chuang. *Hamiltonian Simulation by Qubitization*. misc. Only available as eprint, no journal publication. Oct. 2016. eprint: 1610.06546v2 *Cited on pages 32, 34, 35.*

[87] Guang Hao Low and Isaac L. Chuang. *Hamiltonian Simulation by Uniform Spectral Amplification*. misc. Only available as eprint. No journal publication. July 2017. eprint: 1707.05391v1 *Cited on pages 32, 34, 35.*

[88] Leonardo Novo and Dominic W. Berry. “Improved Hamiltonian simulation via a truncated Taylor series and corrections”. English. In: *Quantum Information and Computation* 17.7-8 (Nov. 2016). Quantum Information and Computation 17, 0623 (2017), pp. 623–635. ISSN: 1533-7146. eprint: 1611.10033v1 *Cited on page 32.*

[89] Patrick J. Coles et al. “Quantum Algorithm Implementations for Beginners”. In: (Apr. 2018). eprint: 1804.03719v1 *Cited on pages 33, 34.*

[90] *Hamiltonian simulation implementation in qiskit-aqua*. https://github.com/Qiskit/qiskit-aqua/blob/master/qiskit/aqua/operators/weighted_pauli_operator.py#L837. Accessed: 2020-03-27. 2019 *Cited on page 34.*

[91] *Quantum algorithms for the simulation of Hamiltonian dynamics*. https://github.com/njross/simcount. Accessed: 2020-03-27. 2019 *Cited on page 34.*

[92] Almudena Carrera Vazquez. "Quantum Algorithm for Solving Tri-Diagonal Linear Systems of Equations". mastersthesis. ETH Zürich, Nov. 2018 *Cited on page 35.*

[93] Masuo Suzuki. "Fractal decomposition of exponential operators with applications to many-body theories and Monte Carlo simulations". In: *Physics Letters A* 146 (6 June 1990), pp. 319–323. DOI: 10.1016/0375-9601(90)90962-N *Cited on page 36.*

[94] Masuo Suzuki. "Quantum statistical monte carlo methods and applications to spin systems". In: *Journal of Statistical Physics* 43 (5-6 June 1986), pp. 883–909. DOI: 10.1007/BF02628318 *Cited on page 36.*

[95] Himanshu Thapliyal and Nagarajan Ranganathan. "Design of Efficient Reversible Logic Based Binary and BCD Adder Circuits". In: (Dec. 2017). J. Emerg. Technol. Comput. Syst. 9 (2013) 17:1-17:31. DOI: 10.1145/2491682. eprint: 1712.02630v1 *Cited on page 44.*

[96] Steven A. Cuccaro et al. "A new quantum ripple-carry addition circuit". In: (Oct. 2004). eprint: quant-ph/0410184v1 *Cited on pages 44, 45.*

[97] Thomas G. Draper. "Addition on a Quantum Computer". In: (Aug. 2000). eprint: quant-ph/0008033v1 *Cited on pages 44, 45.*

[98] Adriano Barenco et al. "Approximate Quantum Fourier Transform and Decoherence". In: (Jan. 1996). DOI: 10.1103/PhysRevA.54.139. eprint: quant-ph/9601018v1 *Cited on pages 44, 45.*

[99] Richard Cleve and John Watrous. "Fast parallel circuits for the quantum Fourier transform". In: (June 2000). eprint: quant-ph/0006004v1 *Cited on pages 44, 45.*

[100] Thomas Häner, Martin Roetteler, and Krysta M. Svore. "Factoring using 2n+2 qubits with Toffoli based modular multiplication". In: (Nov. 2016). Quantum Information and Computation, Vol. 17, No. 7 & 8 (2017). eprint: 1611.07995v2 *Cited on pages 45, 65.*

[101] *Constructing Large Controlled Nots.* https://algassert.com/circuits/2015/06/05/Constructing-Large-Controlled-Nots.html. Accessed: 2020-03-27. 2015 *Cited on pages 46, 65.*

[102] *14-qubit backend: IBM Q team, "IBM Q 16 Melbourne backend specifications V1.3.0" (2019).* Retrieved from https://quantum-computing.ibm.com. 2019 *Cited on page 51.*

[103] *Melbourne hardware operation execution time.* https://github.com/Qiskit/ibmq-device-information/blob/master/backends/melbourne/V1/version_log.md#gate-specification. Accessed: 2020-03-27. 2019 *Cited on pages 51, 57.*

[104] Austin G. Fowler et al. "Surface codes: Towards practical large-scale quantum computation". In: *Phys. Rev. A 86, 032324 (2012)* (Aug. 4, 2012). DOI: 10.1103/PhysRevA.86.032324. arXiv: 1208.0928v2 [quant-ph] *Cited on pages 54, 58, 66, 122.*

[105] *Melbourne gate specification.* https://github.com/Qiskit/ibmq-device-information/tree/master/backends/melbourne/V1#gate-specification. Accessed: 2020-03-27. 2019 *Cited on page 57.*

[106] Neil J Ross and Peter Selinger. "Optimal ancilla-free Clifford+ T approximation of z-rotations". In: *arXiv preprint arXiv:1403.2975* (2014) *Cited on pages 58, 65.*

[107] Vlatko Vedral, Adriano Barenco, and Artur Ekert. "Quantum networks for elementary arithmetic operations". In: *Physical Review A* 54.1 (July 1996), pp. 147–153. ISSN: 1094-1622. DOI: 10.1103/physreva.54.147 *Cited on pages 62, 65.*

[108] Andrew M. Childs et al. "A Theory of Trotter Error". In: (Dec. 2019). eprint: 1912.08854v1 *Cited on pages 56, 58, 66.*

[109] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79 *Cited on pages 58, 141.*

[110] Vivek V. Shende and Igor L. Markov. *On the CNOT-cost of TOFFOLI gates*. 2008. arXiv: 0803.2316 [quant-ph] *Cited on page 58.*

[111] Taewan Kim and Byung-Soo Choi. "Efficient decomposition methods for controlled-Rnusing a single ancillary qubit". In: *Scientific Reports* 8.1 (Apr. 2018), p. 5445. ISSN: 2045-2322. DOI: 10.1038/s41598-018-23764-x *Cited on pages 58, 66.*

[112] Ayush Tambde. *A Programming Language For Quantum Oracle Construction*. 2021. DOI: 10.48550/ARXIV.2110.12487 *Cited on pages 64, 156.*

[113] KP Griffin et al. *Investigation of quantum algorithms for direct numerical simulation of the Navier-Stokes equations*. Accessed: 2022-09-10. 2022 *Cited on pages 64, 156.*

References for Chapter 4: qprof

[29] Andrew M. Childs et al. "Toward the first quantum simulation with quantum speedup". In: *Proceedings of the National Academy of Sciences* 115.38 (Sept. 2018), pp. 9456–9461. ISSN: 1091-6490. DOI: 10.1073/pnas.1801723115 *Cited on pages 25, 70.*

[75] Adrien Suau, Gabriel Staffelbach, and Henri Calandra. "Practical Quantum Computing: Solving the Wave Equation Using a Quantum Approach". In: *ACM Transactions on Quantum Computing* 2.1 (Feb. 2021). ISSN: 2643-6809. DOI: 10.1145/3430030. arXiv: 2003.12458 [quant-ph] *Cited on pages 31, 86, 89.*

[114] Adrien Suau, Gabriel Staffelbach, and Aida Todri-Sanial. "Qprof: A Gprof-Inspired Quantum Profiler". In: *ACM Transactions on Quantum Computing* (Mar. 2022). Just Accepted. ISSN: 2643-6809. DOI: 10.1145/3529398 *Cited on page 69.*

[115] Philip Ball. *First quantum computer to pack 100 qubits enters crowded race*. Nature 599, 542 (2021). 2021. URL: https://www.nature.com/articles/d41586-021-03476-5 (visited on 01/31/2022) *Cited on pages 70, 121.*

[116] Héctor Abraham et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: 10.5281/zenodo.2562110 *Cited on page 70.*

[117] Microsoft Quantum team. *The Q# User Guide*. 2021. URL: https://docs.microsoft.com/en-us/azure/quantum/user-guide/ (visited on 05/31/2021) *Cited on page 70.*

[118] Rigetti Computing. *PyQuil documentation*. 2021. URL: https://pyquil-docs.rigetti.com/en/stable/ (visited on 05/31/2021) *Cited on page 70.*

[119] Cirq Developers. *Cirq*. 2021. DOI: 10.5281/ZENODO.4062499 *Cited on page 70.*

[120] Atos Quantum Computing team. *myQLM documentation*. 2021. URL: https://myqlm.github.io/ (visited on 05/31/2021) *Cited on page 70.*

[121] Scott Aaronson and Daniel Gottesman. "Improved simulation of stabilizer circuits". In: *Phys. Rev. A* 70 (5 Nov. 2004), p. 052328. DOI: 10.1103/PhysRevA.70.052328 *Cited on page 70.*

[122] Craig Gidney. "Stim: a fast stabilizer circuit simulator". In: *Quantum* 5 (July 2021), p. 497. ISSN: 2521-327X. DOI: 10.22331/q-2021-07-06-497 *Cited on page 70.*

[123] Guifré Vidal. "Efficient Classical Simulation of Slightly Entangled Quantum Computations". In: *Physical Review Letters* 91.14 (Oct. 2003). ISSN: 1079-7114. DOI: 10.1103/physrevlett.91.147902 *Cited on page 70.*

[124] Ulrich Schollwöck. "The density-matrix renormalization group in the age of matrix product states". In: *Annals of Physics* 326.1 (Jan. 2011), pp. 96–192. ISSN: 0003-4916. DOI: 10.1016/j.aop.2010.09.012 *Cited on page 70.*

[125] Joseph Emerson, Robert Alicki, and Karol Życzkowski. "Scalable noise estimation with random unitary operators". In: *Journal of Optics B: Quantum and Semiclassical Optics* 7.10 (Sept. 2005), S347–S352. ISSN: 1741-3575. DOI: 10.1088/1464-4266/7/10/021 *Cited on page 70.*

[126] E. Knill et al. "Randomized benchmarking of quantum gates". In: *Physical Review A* 77.1 (Jan. 2008). ISSN: 1094-1622. DOI: 10.1103/physreva.77.012307 *Cited on page 70.*

[127] Jay M. Gambetta et al. "Characterization of Addressability by Simultaneous Randomized Benchmarking". In: *Phys. Rev. Lett.* 109 (24 Dec. 2012), p. 240504. DOI: 10.1103/PhysRevLett.109.240504 *Cited on page 70.*

[128] Andrew W. Cross et al. "Scalable randomised benchmarking of non-Clifford gates". In: *npj Quantum Information* 2.1 (Apr. 2016), p. 16012. ISSN: 2056-6387. DOI: 10.1038/npjqi.2016.12 *Cited on page 70.*

[129] David C. McKay et al. "Three-Qubit Randomized Benchmarking". In: *Phys. Rev. Lett.* 122 (20 May 2019), p. 200502. DOI: 10.1103/PhysRevLett.122.200502 *Cited on page 70.*

[130] Ryan LaRose et al. *Mitiq: A software package for error mitigation on noisy quantum computers*. 2020. DOI: 10.48550/arXiv.2009.04417. arXiv: 2009.04417 [quant-ph] *Cited on page 70.*

[131] Sergey Bravyi et al. "Mitigating measurement errors in multiqubit experiments". In: *Phys. Rev. A* 103 (4 Apr. 2021), p. 042605. DOI: 10.1103/PhysRevA.103.042605 *Cited on page 70.*

[132] Raban Iten et al. "Exact and Practical Pattern Matching for Quantum Circuit Optimization". In: *ACM Transactions on Quantum Computing* 3.1 (Jan. 2022). ISSN: 2643-6809. DOI: 10.1145/3498325 *Cited on pages 70, 123.*

[133] D. Maslov et al. "Quantum Circuit Simplification and Level Compaction". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.3 (Mar. 2008), pp. 436–444. ISSN: 1937-4151. DOI: 10.1109/tcad.2007.911334 *Cited on pages 70, 123.*

[134] Yunseong Nam et al. "Automated optimization of large quantum circuits with continuous parameters". In: *npj Quantum Information* 4.1 (May 2018). DOI: 10.1038/s41534-018-0072-4 *Cited on pages 70, 123.*

[135] Thomas Fösel et al. *Quantum circuit optimization with deep reinforcement learning*. 2021. DOI: 10.48550/arXiv.2103.07585. arXiv: 2103.07585 [quant-ph] *Cited on pages 70, 123.*

[136] J.-H. Bae et al. "Quantum circuit optimization using quantum Karnaugh map". In: *Scientific Reports* 10.1 (Sept. 2020), p. 15651. ISSN: 2045-2322. DOI: 10.1038/s41598-020-72469-7 *Cited on pages 70, 123.*

[137] Yunong Shi et al. "Optimized Compilation of Aggregated Instructions for Realistic Quantum Computers". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '19. Providence, RI, USA: Association for Computing Machinery, 2019, pp. 1031–1044. ISBN: 9781450362405. DOI: 10.1145/3297858.3304018 *Cited on pages 70, 123.*

[138] Pranav Gokhale et al. "Optimized Quantum Compilation for Near-Term Algorithms with OpenPulse". In: *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2020, pp. 186–200. DOI: 10.1109/MICRO50266.2020.00027 *Cited on pages 70, 123.*

[139] Nathan Earnest, Caroline Tornow, and Daniel J. Egger. *Pulse-efficient circuit transpilation for quantum applications on cross-resonance-based hardware*. 2021. DOI: 10.1103/PhysRevResearch.3.043088. arXiv: 2105.01063 [quant-ph] *Cited on pages 70, 123.*

[140] Susan L. Graham, Peter B. Kessler, and Marshall K. Mckusick. "Gprof: A Call Graph Execution Profiler". In: *SIGPLAN Not.* 17.6 (June 1982), pp. 120–126. ISSN: 0362-1340. DOI: 10.1145/872726.806987 *Cited on pages 70, 71, 95.*

[141] Free Software Foundation. *GNU gprof*. 2020. URL: https://sourceware.org/binutils/docs/gprof/index.html (visited on 05/31/2021) *Cited on pages 70, 95.*

[142] *Unix Programmer's Manual, 4th Edition*. `prof` manual can be found in the file `manx/prof.1`. 1973. URL: http://www.tuhs.org/Archive/Distributions/Research/Dennis_v4/v4man.tar.gz *Cited on page 71.*

[143] OProfile maintainers. *OProfile website*. 2020. URL: https://oprofile.sourceforge.io/news/ (visited on 01/26/2022) *Cited on page 71.*

[144] Ali Javadi-Abhari et al. "ScaffCC: A Framework for Compilation and Analysis of Quantum Computing Programs". In: *Proceedings of the 11th ACM Conference on Computing Frontiers*. CF '14. Cagliari, Italy: Association for Computing Machinery, 2014. ISBN: 9781450328708. DOI: 10.1145/2597917.2597939 *Cited on page 71.*

[145] Jonathan M. Smith et al. *Quipper: Concrete Resource Estimation in Quantum Algorithms*. 2014. DOI: 10.48550/arXiv.1412.0625. arXiv: 1412.0625 [cs.PL] *Cited on page 73.*

[146] Brendan Gregg. "The Flame Graph". In: *Commun. ACM* 59.6 (May 2016), pp. 48–57. ISSN: 0001-0782. DOI: 10.1145/2909476 *Cited on pages 73, 93, 96.*

[147] Ketan N. Patel, Igor L. Markov, and John P. Hayes. "Optimal Synthesis of Linear Reversible Circuits". In: *Quantum Info. Comput.* 8.3 (Mar. 2008), pp. 282–294. ISSN: 1533-7146 *Cited on page 93.*

[148] Arianne Meijer-van de Griend and Ross Duncan. *Architecture-aware synthesis of phase polynomials for NISQ devices*. 2020. DOI: 10.48550/arXiv.2004.06052. arXiv: 2004.06052 [quant-ph] *Cited on page 93.*

[149] Timothée Goubault de Brugière et al. "Quantum CNOT Circuits Synthesis for NISQ Architectures Using the Syndrome Decoding Problem". In: *Reversible Computation: 12th International Conference, RC 2020, Oslo, Norway, July 9-10, 2020, Proceedings*. Oslo, Norway: Springer-Verlag, 2020, pp. 189–205. ISBN: 978-3-030-52481-4. DOI: 10.1007/978-3-030-52482-1_11 *Cited on page 93.*

[150] Alexander Mccaskey et al. "Extending C++ for Heterogeneous Quantum-Classical Computing". In: *ACM Transactions on Quantum Computing* 2.2 (July 2021). ISSN: 2643-6809. DOI: 10.1145/3462670 *Cited on page 95.*

[151] The Linux Foundation. *perf_event tutorial*. 2020. URL: https://perf.wiki.kernel.org (visited on 05/31/2021) *Cited on page 96.*

References for Chapter 5: Hardware aware compiler

[152] Donatello Conte et al. "Thirty Years Of Graph Matching In Pattern Recognition". In: *International Journal of Pattern Recognition and Artificial Intelligence* 18.3 (2004), pp. 265–298. DOI: 10.1142/S0218001404003228 *Cited on page 101.*

[153] Vincenzo Carletti et al. "Introducing VF3: A New Algorithm for Subgraph Isomorphism". In: *Graph-Based Representations in Pattern Recognition*. Ed. by Pasquale Foggia, Cheng-Lin Liu, and Mario Vento. Cham: Springer International Publishing, 2017, pp. 128–139. ISBN: 978-3-319-58961-9 *Cited on page 101.*

[154] Debjyoti Bhattacharjee and Anupam Chattopadhyay. *Depth-Optimal Quantum Circuit Placement for Arbitrary Topologies*. arXiv : https://arxiv.org/abs/1703.08540. 2017. arXiv: 1703.08540 [cs.ET] *Cited on page 102.*

[155] Debjyoti Bhattacharjee et al. "MUQUT: Multi-constraint quantum circuit mapping on NISQ computers". In: *38th IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2019*. doi: https://doi.org/10.1109/ICCAD45719.2019.8942132. Institute of Electrical and Electronics Engineers Inc. 2019, p. 8942132 *Cited on page 102.*

[156] Lingling Lao et al. "Mapping of quantum circuits onto NISQ superconducting processors". In: *arXiv e-prints*, arXiv:1908.04226v1 (Aug. 2019). arXiv : https://arxiv.org/abs/1908.04226v1. arXiv: 1908.04226v1 [quant-ph] *Cited on page 102.*

[157] Alexandre A. A. de Almeida, Gerhard W. Dueck, and Alexandre C. R. da Silva. "Finding Optimal Qubit Permutations for IBM's Quantum Computer Architectures". In: *Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design*. SBCCI '19. doi: https://doi.org/10.1145/3338852.3339829. São Paulo, Brazil: Association for Computing Machinery, 2019. ISBN: 9781450368445 *Cited on page 102.*

[158] Prakash Murali et al. "Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '19. doi: https://doi.org/10.1145/3297858.3304075. Providence, RI, USA: Association for Computing Machinery, 2019, pp. 1015–1029. ISBN: 9781450362405 *Cited on pages 102, 114.*

[159] Prakash Murali et al. *Full-Stack, Real-System Quantum Computer Studies: Architectural Comparisons and Design Insights*. doi: https://doi.org/10.1145/3307650.3322273. 2019. arXiv: 1905.11349 [quant-ph] *Cited on page 102.*

[160] Kyle E. C. Booth et al. "Comparing and Integrating Constraint Programming and Temporal Planning for Quantum Circuit Compilation". In: *International Conference on Automated Planning and Scheduling*. arXiv: https://arxiv.org/abs/1803.06775. 2018, pp. 366–374. arXiv: 1803.06775 [quant-ph] *Cited on page 102.*

[161] Davide Venturelli et al. "Quantum Circuit Compilation : An Emerging Application for Automated Reasoning". In: (2019) *Cited on page 102.*

[162] Mehdi Saeedi, Robert Wille, and Rolf Drechsler. "Synthesis of quantum circuits for linear nearest neighbor architectures". In: *Quantum Information Processing* 10.3 (2011). doi: https://doi.org/10.1007/s11128-010-0201-2, pp. 355–377 *Cited on page 102.*

[163] Mohammad Alfailakawi, Imtiaz Ahmad, and Suha Hamdan. "LNN Reversible Circuit Realization Using Fast Harmony Search Based Heuristic". In: *Asia-Pacific Conference on Computer Science and Electrical Engineering*. Nov. 2014 *Cited on page 102.*

[164] Robert Wille et al. “Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits”. In: *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. doi: https://doi.org/10.1109/ASPDAC.2016.7428026. IEEE. 2016, pp. 292–297 *Cited on page 102.*

[165] Ritu Ranjan Shrivastwa, Kamalika Datta, and Indranil Sengupta. “Fast qubit placement in 2D architecture using nearest neighbor realization”. In: *2015 ieee international symposium on nanoelectronic and information systems*. doi: https://doi.org/10.1109/iNIS.2015.59. IEEE. 2015, pp. 95–100 *Cited on page 102.*

[166] Abhoy Kole, Kamalika Datta, and Indranil Sengupta. “A Heuristic for Linear Nearest Neighbor Realization of Quantum Circuits by SWAP Gate Insertion Using N-Gate Lookahead”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 6.1 (2016). doi: https://doi.org/10.1109/JETCAS.2016.2528720, pp. 62–72 *Cited on page 102.*

[167] Alwin Zulehner, Alexandru Paler, and Robert Wille. “An efficient methodology for mapping quantum circuits to the IBM QX architectures”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.7 (2018). doi: https://doi.org/10.1109/TCAD.2018.2846658, pp. 1226–1236 *Cited on pages 102, 114.*

[168] Marcos Yukio Siraichi et al. “Qubit Allocation”. In: *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. CGO 2018. doi: https://doi.org/10.1145/3168822. Vienna, Austria: Association for Computing Machinery, 2018, pp. 113–125. ISBN: 9781450356176 *Cited on page 102.*

[169] Gushu Li, Yufei Ding, and Yuan Xie. “Tackling the qubit mapping problem for NISQ-era quantum devices”. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. doi: https://doi.org/10.1145/3297858.3304023. 2019, pp. 1001–1014 *Cited on pages 102, 104, 106, 109, 114.*

[170] Xiangzhen Zhou, Sanjiang Li, and Yuan Feng. “Quantum Circuit Transformation Based on Simulated Annealing and Heuristic Search”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020). doi: https://doi.org/10.1109/TCAD.2020.2969647 *Cited on pages 102, 111.*

[171] Toshinari Itoko et al. “Optimization of quantum circuit mapping using gate transformation and commutation”. In: *Integration* 70 (2020). doi: https://doi.org/10.1016/j.vlsi.2019.10.004, pp. 43–50 *Cited on pages 102, 109.*

[172] Alexander Cowtan et al. “On the Qubit Routing Problem”. In: *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*. Ed. by Wim van Dam and Laura Mancinska. Vol. 135. Leibniz International Proceedings in Informatics (LIPIcs). doi: https://doi.org/10.4230/LIPIcs.TQC.2019.5. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 5:1–5:32. ISBN: 978-3-95977-112-2. DOI: 10.4230/LIPIcs.TQC.2019.5 *Cited on page 102.*

[173] Gian Giacomo Guerreschi. “Scheduler of quantum circuits based on dynamical pattern improvement and its application to hardware design”. In: *arXiv e-prints*, arXiv:1912.00035 (Nov. 2019). arXiv: https://arxiv.org/abs/1912.00035, arXiv:1912.00035. arXiv: 1912.00035 [quant-ph] *Cited on page 102.*

[174] P. Zhu, Z. Guan, and X. Cheng. “A Dynamic Look-ahead Heuristic for the Qubit Mapping Problem of NISQ Computers”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020). doi: https://doi.org/10.1109/TCAD.2020.2970594. ISSN: 1937-4151. DOI: 10.1109/TCAD.2020.2970594 *Cited on pages 102, 114.*

[175] Gian Giacomo Guerreschi and Jongsoo Park. "Two-step approach to scheduling quantum circuits". In: *Quantum Science and Technology* 3.4 (July 2018). doi: https://doi.org/10.1088/2058-9565/aacf0b, p. 045003. DOI: 10.1088/2058-9565/aacf0b *Cited on page 102.*

[176] Swamit S Tannu and Moinuddin K Qureshi. "Not all qubits are created equal: a case for variability-aware policies for NISQ-era quantum computers". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. doi: https://doi.org/10.1145/3297858.3304007. 2019, pp. 987–999 *Cited on pages 102, 104.*

[177] Abdullah Ash-Saki, Mahabubul Alam, and Swaroop Ghosh. "Qure: Qubit re-allocation in noisy intermediate-scale quantum computers". In: *Proceedings of the 56th Annual Design Automation Conference 2019*. doi: https://doi.org/10.1145/3316781.3317888. 2019, pp. 1–6 *Cited on page 102.*

[178] Will Finigan et al. "Qubit allocation for noisy intermediate-scale quantum computers". In: *arXiv preprint arXiv:1810.08291* (2018). arXiv: https://arxiv.org/abs/1810.08291 *Cited on page 102.*

[179] *IBMQ backends information*. https://github.com/Qiskit/ibmq-device-information. Accessed: 2020-03-27. 2019 *Cited on pages 103, 113.*

[180] Andrew W Cross et al. "Open quantum assembly language". In: *arXiv preprint arXiv:1707.03429* (2017). arXiv: https://arxiv.org/abs/1707.03429 *Cited on pages 104, 114.*

[181] Robert W Floyd. "Algorithm 97: shortest path". In: *Communications of the ACM* 5.6 (1962). doi: https://doi.org/10.1145/367766.368168, p. 345 *Cited on page 108.*

[182] Ang Li and Sriram Krishnamoorthy. "QASMBench: A Low-level QASM Benchmark Suite for NISQ Evaluation and Simulation". In: *arXiv preprint arXiv:2005.13018* (2020). arXiv: https://arxiv.org/abs/2005.13018 *Cited on page 114.*

[183] R. Wille et al. "RevLib: An Online Resource for Reversible Functions and Reversible Circuits". In: *Int'l Symp. on -Valued Logic*. RevLib is available at http://www.revlib.org. 2008, pp. 220–225 *Cited on page 114.*

References for Chapter 6: Variational quantum linear solver

[13] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. "Quantum Algorithm for Linear Systems of Equations". In: *Physical Review Letters* 103 (15 Oct. 2009). Phys. Rev. Lett. vol. 15, no. 103, pp. 150502 (2009). DOI: 10.1103/PhysRevLett.103.150502. eprint: 0811.3171v3 *Cited on pages 6, 26, 31, 122.*

[38] Scott Aaronson. "Read the fine print". In: *Nature Physics* 11.4 (Apr. 2015), pp. 291–293. DOI: 10.1038/nphys3272 *Cited on pages 26, 122.*

[48] Carlos Bravo-Prieto et al. *Variational Quantum Linear Solver*. 2020. arXiv: 1909.05820 [quant-ph] *Cited on pages 26, 121, 127.*

[77] Artur Scherer et al. "Concrete resource analysis of the quantum linear-system algorithm used to compute the electromagnetic scattering cross section of a 2D target". In: *Quantum Information Processing* 16 (3 Mar. 2017). Quantum Inf Process (2017) 16: 60. DOI: 10.1007/s11128-016-1495-5. eprint: 1505.06552v2 *Cited on pages 32, 34, 122.*

[104] Austin G. Fowler et al. "Surface codes: Towards practical large-scale quantum computation". In: *Phys. Rev. A 86, 032324 (2012)* (Aug. 4, 2012). DOI: 10.1103/PhysRevA.86.032324. arXiv: 1208.0928v2 [quant-ph] *Cited on pages 54, 58, 66, 122.*

[115] Philip Ball. *First quantum computer to pack 100 qubits enters crowded race*. Nature 599, 542 (2021). 2021. URL: https://www.nature.com/articles/d41586-021-03476-5 (visited on 01/31/2022) *Cited on pages 70, 121.*

[132] Raban Iten et al. "Exact and Practical Pattern Matching for Quantum Circuit Optimization". In: *ACM Transactions on Quantum Computing* 3.1 (Jan. 2022). ISSN: 2643-6809. DOI: 10.1145/3498325 *Cited on pages 70, 123.*

[133] D. Maslov et al. "Quantum Circuit Simplification and Level Compaction". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.3 (Mar. 2008), pp. 436–444. ISSN: 1937-4151. DOI: 10.1109/tcad.2007.911334 *Cited on pages 70, 123.*

[134] Yunseong Nam et al. "Automated optimization of large quantum circuits with continuous parameters". In: *npj Quantum Information* 4.1 (May 2018). DOI: 10.1038/s41534-018-0072-4 *Cited on pages 70, 123.*

[135] Thomas Fösel et al. *Quantum circuit optimization with deep reinforcement learning*. 2021. DOI: 10.48550/arXiv.2103.07585. arXiv: 2103.07585 [quant-ph] *Cited on pages 70, 123.*

[136] J.-H. Bae et al. "Quantum circuit optimization using quantum Karnaugh map". In: *Scientific Reports* 10.1 (Sept. 2020), p. 15651. ISSN: 2045-2322. DOI: 10.1038/s41598-020-72469-7 *Cited on pages 70, 123.*

[137] Yunong Shi et al. "Optimized Compilation of Aggregated Instructions for Realistic Quantum Computers". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '19. Providence, RI, USA: Association for Computing Machinery, 2019, pp. 1031–1044. ISBN: 9781450362405. DOI: 10.1145/3297858.3304018 *Cited on pages 70, 123.*

[138] Pranav Gokhale et al. "Optimized Quantum Compilation for Near-Term Algorithms with OpenPulse". In: *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2020, pp. 186–200. DOI: 10.1109/MICRO50266.2020.00027 *Cited on pages 70, 123.*

[139] Nathan Earnest, Caroline Tornow, and Daniel J. Egger. *Pulse-efficient circuit transpilation for quantum applications on cross-resonance-based hardware*. 2021. DOI: 10.1103/PhysRevResearch.3.043088. arXiv: 2105.01063 [quant-ph] *Cited on pages 70, 123.*

[184] Earl T. Campbell, Barbara M. Terhal, and Christophe Vuillot. "Roads towards fault-tolerant universal quantum computation". In: *Nature* 549.7671 (Sept. 2017), pp. 172–179. ISSN: 1476-4687. DOI: 10.1038/nature23460 *Cited on page 122.*

[185] Bibek Pokharel et al. "Demonstration of Fidelity Improvement Using Dynamical Decoupling with Superconducting Qubits". In: *Phys. Rev. Lett.* 121 (22 Nov. 2018), p. 220502. DOI: 10.1103/PhysRevLett.121.220502 *Cited on page 123.*

[186] Alexandre M Souza, Gonzalo A Álvarez, and Dieter Suter. "Robust dynamical decoupling". In: *Philos. Trans. A Math. Phys. Eng. Sci.* 370.1976 (Oct. 2012), pp. 4748–4769 *Cited on page 123.*

[187] Jacob R. West et al. "High Fidelity Quantum Gates via Dynamical Decoupling". In: *Phys. Rev. Lett.* 105 (23 Dec. 2010), p. 230503. DOI: 10.1103/PhysRevLett.105.230503 *Cited on page 123.*

[188] Lu-Ming Duan and Guang-Can Guo. "Suppressing environmental noise in quantum computation through pulse control". In: *Physics Letters A* 261.3 (1999), pp. 139–144. ISSN: 0375-9601. DOI: https://doi.org/10.1016/S0375-9601(99)00592-7 *Cited on page 123.*

[189] Lorenza Viola, Emanuel Knill, and Seth Lloyd. "Dynamical Decoupling of Open Quantum Systems". In: *Phys. Rev. Lett.* 82 (12 Mar. 1999), pp. 2417–2421. DOI: 10.1103/PhysRevLett.82.2417 *Cited on page 123.*

[190] Lorenza Viola and Seth Lloyd. "Dynamical suppression of decoherence in two-state quantum systems". In: *Phys. Rev. A* 58 (4 Oct. 1998), pp. 2733–2744. DOI: 10.1103/PhysRevA.58.2733 *Cited on page 123.*

[191] Kristan Temme, Sergey Bravyi, and Jay M. Gambetta. "Error Mitigation for Short-Depth Quantum Circuits". In: *Phys. Rev. Lett.* 119 (18 Nov. 2017), p. 180509. DOI: 10.1103/PhysRevLett.119.180509 *Cited on page 123.*

[192] Suguru Endo, Simon C. Benjamin, and Ying Li. "Practical Quantum Error Mitigation for Near-Future Applications". In: *Phys. Rev. X* 8 (3 July 2018), p. 031027. DOI: 10.1103/PhysRevX.8.031027 *Cited on page 123.*

[193] Shuaining Zhang et al. "Error-mitigated quantum gates exceeding physical fidelities in a trapped-ion system". In: *Nature Communications* 11.1 (Jan. 2020), p. 587. ISSN: 2041-1723. DOI: 10.1038/s41467-020-14376-z *Cited on page 123.*

[194] Piotr Czarnik et al. "Error mitigation with Clifford quantum-circuit data". In: *Quantum* 5 (Nov. 2021), p. 592. ISSN: 2521-327X. DOI: 10.22331/q-2021-11-26-592 *Cited on page 123.*

[195] Angus Lowe et al. "Unified approach to data-driven quantum error mitigation". In: *Phys. Rev. Research* 3 (3 July 2021), p. 033098. DOI: 10.1103/PhysRevResearch.3.033098 *Cited on page 123.*

[196] Paul D. Nation et al. "Scalable Mitigation of Measurement Errors on Quantum Computers". In: *PRX Quantum* 2 (4 Nov. 2021), p. 040326. DOI: 10.1103/PRXQuantum.2.040326 *Cited on page 123.*

[197] Ying Li and Simon C. Benjamin. "Efficient Variational Quantum Simulator Incorporating Active Error Minimization". In: *Physical Review X* 7 (2 June 2017). DOI: 10.1103/PhysRevX.7.021050 *Cited on page 123.*

[198] Abhinav Kandala et al. "Error mitigation extends the computational reach of a noisy quantum processor". In: *Nature* 567.7749 (Mar. 2019), pp. 491–495. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1040-7 *Cited on page 123.*

[199] Alberto Peruzzo et al. "A variational eigenvalue solver on a photonic quantum processor". In: *Nature Communications* 5.1 (July 2014). DOI: 10.1038/ncomms5213 *Cited on pages 123, 124.*

[200] Sukin Sim, Peter D. Johnson, and Alán Aspuru-Guzik. "Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms". In: *Advanced Quantum Technologies* 2.12 (2019), p. 1900070. DOI: https://doi.org/10.1002/qute.201900070. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/qute.201900070 *Cited on page 125.*

[201] M. Cerezo et al. "Cost function dependent barren plateaus in shallow parametrized quantum circuits". In: *Nature Communications* 12.1 (Mar. 2021), p. 1791. ISSN: 2041-1723. DOI: 10.1038/s41467-021-21728-w *Cited on pages 126, 127.*

[202] Andrew Arrasmith et al. "Effect of barren plateaus on gradient-free optimization". In: *Quantum* 5 (Oct. 2021), p. 558. ISSN: 2521-327X. DOI: 10.22331/q-2021-10-05-558 *Cited on page 126.*

[203] *Constructing Large Increment Gates*. https://algassert.com/circuits/2015/06/12/Constructing-Large-Increment-Gates.html. Accessed: 2022-09-10. 2015 *Cited on page 131.*

[204] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000 *Cited on page 131.*

[205] University of Texas Austin Ward Cheney and David Kincaid. *Numerical mathematics and computing*. 6th ed. Belmont, CA: Wadsworth Publishing, Aug. 2007 *Cited on page 132.*

References for Chapter 7: Single qubit tomography visualisation

[109] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79 *Cited on pages 58, 141.*

[206] Jens Eisert et al. "Quantum certification and benchmarking". In: *Nature Reviews Physics* 2.7 (July 2020), pp. 382–390. ISSN: 2522-5820. DOI: 10.1038/s42254-020-0186-4 *Cited on page 141.*

[207] K. Wright et al. "Benchmarking an 11-qubit quantum computer". In: *Nature Communications* 10.1 (Nov. 2019), p. 5464. ISSN: 2041-1723. DOI: 10.1038/s41467-019-13534-2 *Cited on page 141.*

[208] Z. Hradil. "Quantum-state estimation". In: *Phys. Rev. A* 55 (3 Mar. 1997), R1561–R1564. DOI: 10.1103/PhysRevA.55.R1561 *Cited on pages 141, 142.*

[209] Easwar Magesan, J. M. Gambetta, and Joseph Emerson. "Scalable and Robust Randomized Benchmarking of Quantum Processes". In: *Phys. Rev. Lett.* 106 (18 May 2011), p. 180504. DOI: 10.1103/PhysRevLett.106.180504 *Cited on pages 141, 142.*

[210] E. Knill et al. "Randomized benchmarking of quantum gates". In: *Phys. Rev. A* 77 (1 Jan. 2008), p. 012307. DOI: 10.1103/PhysRevA.77.012307 *Cited on pages 141, 142.*

[211] Erik Nielsen et al. *Gate Set Tomography*. 2020. eprint: arXiv:2009.07301 *Cited on pages 141, 142.*

[212] Andrew W. Cross et al. "Validating quantum computers using randomized model circuits". In: *Phys. Rev. A* 100 (3 Sept. 2019), p. 032328. DOI: 10.1103/PhysRevA.100.032328 *Cited on page 141.*

[213] Sergio Boixo et al. "Characterizing quantum supremacy in near-term devices". In: *Nature Physics* 14.6 (June 2018), pp. 595–600. ISSN: 1745-2481. DOI: 10.1038/s41567-018-0124-x *Cited on page 141.*

[214] Jay Gambetta and Sarah Sheldon. *Cramming More Power Into a Quantum Device*. Published online at https://www.ibm.com/blogs/research/2019/03/power-quantum-device/. Accessed: 03/28/2021. 2019 *Cited on page 142.*

[215] Frank Arute et al. "Quantum supremacy using a programmable superconducting processor". In: *Nature* 574.7779 (Oct. 2019), pp. 505–510. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1666-5 *Cited on page 142.*

[216] Yulin Wu et al. "Strong Quantum Computational Advantage Using a Superconducting Quantum Processor". In: *Phys. Rev. Lett.* 127 (18 Oct. 2021), p. 180501. DOI: 10.1103/PhysRevLett.127.180501 *Cited on page 142.*

[217] Bo Qi et al. "Quantum state tomography via linear regression estimation". In: *Scientific reports* 3.1 (2013), pp. 1–6 *Cited on pages 143, 150.*

[218] David Gross et al. "Quantum State Tomography via Compressed Sensing". In: *Phys. Rev. Lett.* 105 (15 Oct. 2010), p. 150401. DOI: 10.1103/PhysRevLett.105.150401 *Cited on page 143.*

[219] Z. Hradil. "Quantum-state estimation". In: *Phys. Rev. A* 55 (3 Mar. 1997), R1561–R1564. DOI: 10.1103/PhysRevA.55.R1561 *Cited on page 143.*

[220] Huo Chen and Daniel A. Lidar. *HOQST: Hamiltonian Open Quantum System Toolkit.* 2020. eprint: arXiv:2011.14046 *Cited on page 148.*

[221] J.R. Johansson, P.D. Nation, and Franco Nori. "QuTiP: An open-source Python framework for the dynamics of open quantum systems". In: *Computer Physics Communications* 183.8 (2012), pp. 1760–1772. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2012.02.021 *Cited on page 148.*

References for Chapter 8: Conclusion

[112] Ayush Tambde. *A Programming Language For Quantum Oracle Construction.* 2021. DOI: 10.48550/ARXIV.2110.12487 *Cited on pages 64, 156.*

[113] KP Griffin et al. *Investigation of quantum algorithms for direct numerical simulation of the Navier-Stokes equations.* Accessed: 2022-09-10. 2022 *Cited on pages 64, 156.*