



UNIVERSIDADE D
COIMBRA

Diogo Miguel Ferreira Gonçalves

**HYBRID QUANTUM-CLASSICAL
COMPUTATION**
HYBRID DIRAC QUANTUM WALKS

**Dissertation in the context of the Master in Engineering Physics,
advised by Doctor Professor Yasser Omar and Doctor Professor
Pedro Vieira Alberto and presented to the Department of Physics
of the Faculty of Sciences and Technology of the University of
Coimbra.**

September 2023



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTMENT OF PHYSICS

Diogo Miguel Ferreira Gonçalves

Hybrid Quantum-Classical Computation

Hybrid Dirac Quantum Walks

Dissertation in the context of the Master in Engineering Physics, advised by Doctor Professor Yasser Omar and Doctor Professor Pedro Vieira Alberto and presented to the Department of Physics of the Faculty of Sciences and Technology of the University of Coimbra.

September 2023



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE
COIMBRA

DEPARTAMENTO DE FÍSICA

Diogo Miguel Ferreira Gonçalves

Computação Quântica-Clássica Híbrida

Passeios quânticos de Dirac híbridos

Dissertação no âmbito do Mestrado em Engenharia Física, orientada pelo Professor Doutor Yasser Omar e Professor Doutor Pedro Vieira Alberto e apresentada ao Departamento de Engenharia Física da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro 2023

Acknowledgements

Começo por agradecer, aos meus orientadores Prof. Yasser Omar e Prof. Pedro Vieira Alberto por tornarem possível explorar esta área, que imensa curiosidade me gerou, com engenho e arte.

À minha família, destacando os meus pais, Maria da Conceição e Fernando, irmãs, Ana e Raquel, e cunhados, Alexandre e Emanuel, pelo carinho, apoio incondicional e orientação durante toda a minha vida.

À minha namorada Maria, pela motivação e carinho.

Por fim, serve também de apreço a todos os que de forma direta ou indireta contribuíram para o desenvolvimento não só deste trabalho, mas de todo o meu percurso académico.

Abstract

Quantum Computing has emerged as a technology capable of changing paradigms on multiple fields. In the other hand, current development of such applications is tampered by noisy quantum hardware.

Hybrid Quantum-Classical Computation leverages the strengths of both Quantum and Classical Computers (such as High Performance Computers (HPC)) to create a bridge that seamlessly integrates and exploits the advantages of each. This field is still overlooked, being under the shadow of the great advancement in the field of quantum hardware. Currently, there is a noticeable void in the efforts to develop algorithms that use the power of both technologies.

In this work we will explore hybrid computing as a whole, understand all of its fundamental parts and its limitations. We began with an introduction of the necessary theoretical background and a study of the state of the art related to hybrid algorithm developments. Then we will present the framework for the quantum walk algorithm, exploring some of its capabilities. Next we will propose an hybrid architecture and application on an unexplored domain, the simulation of Dirac free particle. Then we will do a test of our hybrid algorithm for simple problems, using real quantum hardware, exposing the limitations of it. As a final approach we will use a simulator of quantum hardware ran on a classical High Performance Computer provided by the Laboratory for Advanced Computing of the University of Coimbra, to execute the quantum walk algorithm of the Dirac free-particle trapped in a square potential.

Keywords

quantum computing, hybrid classical-quantum algorithms, high performance computing, quantum walk, dirac free particle

Resumo

A computação quântica emergiu como uma tecnologia capaz de mudar paradigmas em vários campos. Por outro lado, o desenvolvimento atual de tais aplicações é limitado por hardware quântico ainda muito ruidoso.

A Computação Quântica-Clássica Híbrida usa as capacidades de ambos os Computadores Quântico e Clássico (como High Performance Computers (HPC) ou supercomputadores) para criar uma ponte que integra e explora as vantagens de cada um. Este campo ainda é relativamente ignorado, ficando na sombra de grandes avanços do hardware quântico. Atualmente, existe um vazio considerável no que diz respeito aos esforços para desenvolver algoritmos que utilizam as capacidades de ambas as tecnologias.

Neste trabalho, exploraremos a computação híbrida como um todo, compreendendo todas as suas partes fundamentais e as suas limitações. Começaremos com uma introdução teórica sendo necessário para contextualizar este trabalho, e um estudo do estado da arte relacionado ao desenvolvimos de algoritmos híbridos. Depois iremos apresentar a estrutura para o algoritmo de passeios quânticos, propondo uma arquitetura híbrida e aplicação num domínio inexplorado, a simulação da partícula livre de Dirac. De seguida, iremos realizar um teste ao nosso algoritmo híbrido, utilizando hardware quântico real, e expor as limitações deste. Como abordagem final, utilizaremos um simulador de hardware quântico instalado num HPC localizado no Laboratório para Computação Avançada da Universidade de Coimbra, para calcular a evolução de uma partícula livre de Dirac aprisionada num potencial quadrado, utilizando como base o algoritmo de passeios quânticos (quantum walk).

Palavras-Chave

computação quântica, algoritmos híbridos clássicos quânticos , computação de alta performance, caminhos quânticos, partícula livre de Dirac

Contents

List of Figures	xvii
List of Tables	xxi
1 Introduction	1
1.1 Quantum computation history	1
1.2 Classical Random Walks and Quantum Walks	2
1.3 Framing, Motivation and Structure	3
2 Theoretical Foundation	7
2.1 Notions on Quantum Information and Quantum Computing	9
2.1.1 Principles of quantum information	9
2.1.2 Principles of quantum computation	10
2.1.3 Quantum computers: physical realization	13
2.1.4 Notable quantum algorithms	14
2.2 Hybrid Quantum-Classical Algorithms	15
2.2.1 Variational Quantum Eigensolver	15
2.2.2 Quantum Approximate Optimization Algorithm	17
2.2.3 Hybrid Quantum Monte Carlo	18
2.2.4 Hybrid Machine Learning	19
2.3 High Performance Computing	20
2.3.1 High performance computing and quantum computing	22
3 Quantum Walks	25
3.1 Coined Quantum Walk	27
3.1.1 Dirac quantum walk	30
3.1.2 Potential Well	32
3.2 Algorithm for quantum walk	34
3.2.1 Simplified and hybrid quantum walks	38
3.2.2 Quantum walk with position-dependent coins	42
3.2.3 Dirac free-particle trapped in a square potential well	43
3.2.4 Visualizing time evolution	44
3.2.5 Qiskit implementation	45
4 Algorithm Benchmarking	49
4.1 Quantum walk algorithm advantages	51
4.2 Algorithm execution on classical hardware	55
4.2.1 Classical cluster simulation methodology	55
4.2.2 Laboratory for Advanced Computing - Navigator+	55

4.2.3	Simulator architecture	56
4.2.4	Simulation parameters tuning and validation	57
4.2.5	Results and analysis between analytical results	61
5	Discussion and Conclusion	67
	References	69
	Appendix A The Postulates of Quantum Mechanics	77
	Appendix B Analytical overview on the one dimensional Dirac Quantum Walk	81
B.1	Evolution Operator	81
B.2	Spacial Grid	83
B.3	Wave packet	83
	Appendix C ibm_perth calibration data	85
	Appendix D Open sourced Quantum Walk Code	87

Glossary

HPC High Performance Computers.

Hybrid Quantum-Classical Computation algorithm architecture focused on combining quantum and classical computation by leveraging their strengths to obtain an advantage on current state of the art classical and quantum computers.

Noisy Intermediate-Scale Quantum term used to describe the current quantum computers, as machines prone to considerable error rates and limited in size by the number of logical qubits..

qiskit digital open-source framework to develop algorithms for quantum computers, provided by IBM. Development is done using `python` programming language..

Quantum Walk quantum equivalent to random walks. Describes the coherent propagation of quantum particles in networks. It serves as a foundation for many quantum algorithms and applications..

Quantum Computing multidisciplinary field comprising aspects of computer science, physics, and mathematics that utilizes quantum mechanics to solve specific complex problems faster than on classical computers..

List of Figures

2.1	Graphical representation of the Bloch sphere. This method allows for a detail visualization of a qubit state.	10
2.2	Graphical representation of the entanglement circuit we'll use as a practical example. Our horizontal lines represent either qubits (q_0, q_1) or bits (c_2). Gates are applied to the qubits which can affect one or more of them. The CNOT gate performs operations that involve more than one qubit, similar to the classical NOT gate. The Hadamard gate is one of many one qubit gates. Finally, the measurement blocks on the qubits store the results to classical bits.	12
3.1	Probability distribution for different initial coin states, "tails" $ 0\rangle$ and "heads" $ 1\rangle$, respectively.	29
3.2	Symmetrical probability distribution using 8 qubits and $t = 2^{8-1}$ steps simulated on quantum computer simulator <code>qasm_simulator</code> from IBMQ.	29
3.3	Graph networks for a cycle network (left) and for a limited line (right).	30
3.4	Gate for for the position-dependent coin.	32
3.5	Circuit for position dependent coin. Important to note, that for the sake of simplicity we omitted $ \psi_c\rangle$ from the slice annotations.	33
3.6	Quantum circuits for different resolutions on the position space.	33
3.7	Quantum Circuit for the Quantum Walk. For it we want to implement the coin operator and shift operator.	34
3.8	Left and Right shifts, respectively.	35
3.9	Initial position state preparation circuit.	35
3.10	Initial coin state preparation circuit.	36
3.11	Circuit preparation for the generic cycle quantum walk.	36
3.12	Circuit fraction for the S (shift) gate	37
3.13	In the left side we have a graphical representation of the Quantum circuit complexity with the number of qubits n . In the right side we have a graphical representation of the Quantum circuit depth with the number of qubits n	38
3.14	Circuit Schematic for the Quantum Fourier Transform	39
3.15	Shift operator circuit (V) using Fourier space simplification.	40
3.16	Graph for the Unitary Evolution Operator	40
3.17	Quantum Walk Simulation, using 10 qubits for position resolution (2^{10}). The simulation was done using <code>qasm_simulator</code>	41
3.18	Quantum walk circuit in the Fourier space, with the basis transformation at the beginning and at the end of the unitary evolution.	41

3.19	Quantum Circuit with schematic of the Dirac free-particle coin. . . .	44
3.20	Schematic visually showing how the data in the heatmap plot refers to the individual histograms.	45
3.21	Array of all the available, on cloud, IBMQ quantum hardware devices. This list has layed out the number of qubits, quantum volume (QV) and Circuit Layer Operations Per Second (CLOPS) which refer to the performance metrics of the hardware.	46
3.22	Qiskit's task flow chart evolution.	47
4.1	<code>ibm_perth</code> qubit layout. Its architecture can be visualized through different tonalities or shades across its layout. Herein, unique shades correspond to distinct $T1$ times, while variations in connection tonalities represent different extents of CNOT errors. A gradient effect is applied to these tonalities, transitioning from dark to light. It's important to note that this gradation symbolizes a change from lower to higher values, respectively. All the details for this quantum backend can be found on Appendix C.	51
4.3	Graphs from the results ran on ' <code>ibm_perth</code> '. On the top row the results relate to the non-hybrid architecture and the bottom ones to the hybrid architecture.	52
4.2	Graphs from the results ran on ' <code>ibmq_qasm_simulator</code> ' for 2 step quantum walk. On the left is the probability distribution scatter ad on the right the counts histogram.	52
4.4	Plot with the overlap obtain from the 2 step quantum walk simulation on the ideal quantum computer (' <code>ibmq_qasm_simulator</code> ') in red (or dashed) with the ones obtained from ' <code>ibm_perth</code> ', purple (or solid line). On the left is the graph for the non-hybrid architecture and on the right for the hybrid architecture.	53
4.5	Graphs from the results ran on ' <code>ibmq_qasm_simulator</code> ' for 1 step quantum walk. On the left is the probability distribution scatter ad on the right the counts histogram.	53
4.6	Plot with the overlap obtain from the 1 step quantum walk simulation on the ideal quantum computer (' <code>ibmq_qasm_simulator</code> ') in red (or dashed) with the ones obtained from ' <code>ibm_perth</code> ', purple (or solid line). On the left is the graph for the non-hybrid architecture and on the right for the hybrid architecture.	54
4.7	Plots for the algorithm memory usage using the statistics provided by the SLURM <code>sacct -format MaxRSS</code> command. On the left is displayed the memory usage for different number of steps with number of qubits fixed on 6. On the right is displayed the memory usage for different number of qubits, with number of steps fixed on 64.	57

4.8	Memory profiler (using the <code>mprof</code> library for python) for a 6 qubits 64 steps algorithm simulation. From 0 to around 2 seconds the circuits are processed, then until the 82 seconds marks is the transpiler phase and the rest is dedicated to the run phase which is the one that should be affected by this parameter. On the left is the graph for the <code>double</code> precision and on the right for the <code>single</code> precision. Even though the graph curves on the running phase are not identical, both simulations reach the same peak of memory usage and take the same time to simulate.	58
4.9	Schematic for the batching diagram. Q[1-10] relates to the all quantum circuits, distributed by the 7 available batches. The red block is the region until the max memory per pool can be.	58
4.10	Memory profiler (using the <code>mprof</code> library for python) for the <code>max_parallel_experiments</code> (left) and <code>max_parallel_shots</code> (right) optioned for the maximum number of available threads (80). Unfortunately there are not noticeable changes in the memory curve between these two different settings. The running time as slightly better for the <code>max_parallel_shots=80</code> (2.8595 seconds versus 2.6087seconds, respectively).	59
4.11	Memory profiler (using the <code>mprof</code> library for python) for a 6 qubits 64 steps algorithm simulation, comparing the values 5 and 10 for the setting <code>max_parallel_experiments</code> . On the left is for 5 parallel experiments and on the right for 10 parallel experiments. The curves are similar and these differences do not impose changes on the behaviour of the simulation.	60
4.12	Plots for the algorithm time (left) and memory consumption (right) for different batch sizes using the statistics provided by the SLURM <code>sacct -format MaxRSS</code> command. For all the combinations created, all execution took the same time. On the left the average was 10087.8461 ± 59.4051 (0.5889%) [s] and on the right 92570.0 ± 1589.6051 (1.717%) [Mb].	60
4.13	Memory profiler plot for a 6 qubits and 64 (top) and 128 steps (bottom) algorithm simulation, comparing the memory performance by changing the parameter <code>optimization_level</code> , = 0 (left) and = 1 (right). For 64 steps, the transpile times were similar being 363,74 seconds and 362,57 seconds and the execution times were also similar being 356,15 seconds and 355,59 seconds. For 128 steps, we got a shorter transpile times of 1670,056 seconds and 1761,844 seconds. Surprisingly it also got a shorter run time 1391,53 seconds versus 1462,07 seconds.	61
4.14	High definition simulation of a Dirac free-particle trapped in a squared potential well. The available exploratory space was $N = 2^9 = 512$ discrete positions and were done 954 steps.	63
4.15	Scatter plots for the simulation of a Dirac free-particle trapped in a squared potential well for separated in time snippets. (1/3)	64
4.16	Scatter plots for the simulation of a Dirac free-particle trapped in a squared potential well for separated in time snippets. (2/3)	65
4.17	Scatter plots for the simulation of a Dirac free-particle trapped in a squared potential well for separated in time snippets. (3/3)	66

B.1 Overlap between analytical solution from the $\Psi(x, t)$ wave packet and
free-Dirac particle Quantum Walk algorithm execution. 84

List of Tables

2.1	Six common one-qubit gates and the associated unitary transformations	11
2.2	List of the most popular multi-qubit quantum gates.	12
3.1	Probability of finding the quantum particle in position x at time t , assuming that the walk starts at the origin with the quantum coin in “tails” state, $ 0\rangle$	29
4.1	List of the used partitions and specifications for benchmark testing and simulation.[70]	55
4.2	Results for the batching test.	59
A.1	List of Hermitian operators.	78
C.1	Calibration data for the <code>ibm_perth</code> quantum backend.	86

Chapter 1

Introduction

1.1 Quantum computation history

Humanity consequently been profoundly changed by the advent of computing systems. Each evolution of the computing devices to perform numerical operations marked a significant leap in society evolution.

computation

noun [C or U]

the act or process of calculating an answer or amount by using a machine

The concept of an universal machine (later named Turing machine) that could simulate any other computing machine given the proper inputs and programming was created and described by Alan Turing. It paved the way in the mid 1930s for computer science as we know today, giving a more pronounced function to the word computation. In 1945, John von Neumann demonstrated a concept of computer architecture that laid the foundation for most modern computers, even though it was not designed with the Turing machine in mind.

After this theoretical development, it was followed by engineering feats, first by the use of vacuum tube circuits for logic circuitry and soon after, most notably the invention of the transistor by John Bardeen, Walter Brattain, and William Shockley in the Bell Labs. These components truly revolutionized the field of electronics, paving the way for more highly capable electronic devices in consequently smaller foot prints.

This unique moment for rapid and effervescent technology advancement, helped to grasp some particular new ideas in the computation field, namely one crucial for this work to exist. In 1982, Feynman gave a lecture at the MIT Computer Science and Artificial Intelligence Laboratory, proposing the idea of quantum computation. The intuition behind quantum computing is to use quantum mechanics (or quantum states of quantum bits) as a tool to solve complex problems - ones that may be overly complex for classical computers. Thus, he proposed a basic model for a quantum computer that would be capable of such simulations. With this, he outlined the

possibility to exponentially outpace classical computers. However, it took more than 10 years until a special algorithm was created to change the view on quantum computing, the Shor algorithm.

In 1994, Peter Shor developed an algorithm that enables quantum computers to factorize large integers exponentially faster than the most efficient classical algorithm on traditional machines. These classical algorithms would take millions of years to factor 300-digit numbers, whereas theoretically, Shor's algorithm can accomplish this task much more quickly. Capable of breaking many of today's cryptosystems, the prospect of quantum computers doing in hours what would have taken millions of years sparked significant interest in quantum computing and its applications.

In 1996, Lov Grover invented a quantum database search algorithm that presented a quadratic speedup for a variety of problems (classical computers takes $\mathcal{O}(n)$ steps whereas a quantum computer can search in $\mathcal{O}(\sqrt{n})$, for an unsorted and unstructured list).

Similarly, the Quantum Fourier Transform — an integral part of Shor's factoring algorithm — stands out as well. Additionally, algorithms like Quantum Phase Estimation and Harrow-Hassidim-Lloyd (HHL) algorithm present promising applications in the sectors of quantum physics and linear algebra respectively.

These innovations led as influence and even as an encouragement to continuously tackle this domain of computer science. This precedent forms the foundation for our work, as we intend to further explore and contribute to this scientific field a little more.

1.2 Classical Random Walks and Quantum Walks

The simplest example of a random walk is the classical motion of a particle in a line. It can move left or right with this behaviour being determined by a non-biased coin. For example, when tossing the coin, if it tails the particle will move one unit rightward, if it is heads the particle will move one unit leftward, with this process being repeated every time unit. Being a probabilistic process, when the quantum walk ends it will result in an array of probabilities of where the particle is likely to be. The final probability distribution of finding the particle at a given position — as well as questions like “how long does it take the particle to reach a particular position?” — are interesting and useful quantities to calculate.

Quantum Walks, the quantum analogous to the classical random walks, were first introduced by Aharonov et al. [1] in 1993. Classical Random Walks are stochastic processes used to model random motion in discrete space time. In a classical random walk, a walker moves randomly from one location to another in a step wise fashion, with each step being independent of the previous steps, obeying to classical probability theory. The degrees of freedom for space time can be seen as a discrete N -dimensional medium.

Quantum Walks (QW) make use of quantum computing advantages to achieve a performance benefit of the classical counterpart. While classical random walks evolve

randomly over each step, QW exploits interference effects to enhance the probability of finding the walker in certain locations while suppressing it in others, leading to a faster convergence towards the target state [1]. The evolution of isolated quantum systems is unitary, there is no room for randomness, which lead to dropping *random* for the term Quantum Random Walks [2].

Since Aharonov et al. introduced the concept many have suggested different models of Quantum Walks for quantum annealers [3, 4], for circuit based quantum computers in continuous time [5] or discrete time [6, 7] and even classical HPC [8, 9].

Quantum walks emerge as an algorithmic framework for different and interesting domains but also with an interesting modularity in terms of algorithm architecture design. This encouraged us to explore then with more detail, becoming an essential part of this work. More importantly, the quantum walk's architecture allows to implement hybridization.

1.3 Framing, Motivation and Structure

Quantum computation is an emerging science that uses laws from quantum mechanics with the purpose of solving problems, complex for classical computers, in a more efficient way. With transformative applications in areas from chemistry and materials science to finance, logistics operations, and cybersecurity, it began to receive a lot of attention in the last decade, as the possibility of dramatically accelerating the execution time of current supercomputers becomes increasingly realistic.

Faced with these facts, this technology is strongly funded by the United States of America (with the "National Quantum Initiative" being founded in 2018, with a budget of 200 million dollars per year), the European Union (with the "Quantum Technologies Flagship" being founded, which expects to gather one billion euros in ten years) and especially China (having invested estimated amounts of ten billion euros). In addition, it also has the participation of private companies with high skills in this area such as IBM, Google, and Microsoft.

On the other hand, its projection requires cutting-edge engineering solutions. With current technology, it is impossible so far to develop products and services that are useful for the industry and society. The number of qubits in current hardware is very low (for context, the IBM Osprey's number of qubits, one of the most advanced quantum computers, has 433 qubits), and hundreds of thousands of qubits will be necessary to develop some of the most promising areas as well as to significantly reduce the error rate in each qubit.

Moreover, hybrid quantum-classical computing has seen significant advancements in both the hardware and software components, as well as in the development of new algorithms. Despite the hardware limitations, it has not constrained the proposal of new algorithms and efforts to create a good relation between classical and quantum domains. It allows to use NISQ era quantum hardware in real world applications.

In an European perspective, ATOS and IQM [10, 11] have made significant efforts to increase the usability of hybrid quantum-classical computing by partnering to

deliver end-to-end quantum computing technologies as part of their hybrid computing strategy. The EuroHPC JU [12] has a total planned investment in 100 million euros for hybrid quantum computing. In the international landscape, NVIDIA [13] aims to make quantum computing more accessible by creating a coherent hybrid quantum-classical programming model. There are investments for hybrid computation to become a reality but there is still research to be done on the algorithm side of things.

Given these constraints, in this work we want to explore the capabilities that the High Performance Computers (HPC) world can give to the quantum world.

This will be done by developing a hybrid algorithm that provides a possible significant speed-up in performance by effectively harness the benefits of both noise intermediate-scale quantum (NISQ) computers, despite their limited availability and efficiency, and classical computers. To complement this we will also understand the importance of simulating real quantum hardware on HPC to help on systems and algorithm development.

As a framework for this algorithm we will be employing the quantum walks algorithm. Theoretically it shows that quantum walks are quadratically faster than the classical counter part when ran in quantum computers and as we discovered though out this work they can be design to attack many sets of applications. More importantly this framework allows for hybridization, we will be testing with current publicly available quantum hardware.

Personally, working with quantum computing in my master thesis was an extremely important ambition. It came after working on the field via a scholarship under the program "Talents in Quantum Computing" given by the Calouste Gulbenkian Foundation, where a deep curiosity for the field widen and finally sprouted in this course's last hurrah. Supplementing this, was to create something that felt my own, from start to finish, that transmitted my personality and way of thinking but also to have the freedom to execute my science.

Secondarily, but equality important, is to create such endeavor that can be malleable in the various domains it touches. In other words, being an approachable work for any scientific domain and more importantly bringing the worlds of quantum computing and high performance computing closer together. I want the reader to fell a grasp of the passion and intellectual curiosity that fueled this research.

All chapters will include at the beginning some highlights and personal introspection of its upcoming sections. In Chapter 2 we want to settle the main parts for this work, beginning with a brief overview through the fundamental elements of quantum computation and its state of the art in the hardware and software (more in the algorithm) realms. We will also introduce some concepts on high performance computing and the existing hybrid algorithms. Moreover, in Chapter 3 the problems thought process is introduced, unraveling not only the theoretical but the practical aspects, describing the mathematical background of the created algorithms. *Pseudo-code* and quantum circuits for the developed algorithms will be shown and explained. Finally, in Chapter 4 our created algorithms are put to the test, using real quantum hardware and quantum simulators ran on a high performance computer. We close this work in Chapter 5 with the work discussion, summarizing our key findings,

successes and disappointments and new directions for future research work.

Appended to this work, Appendix A lays down the "game rules", the postulates of quantum mechanics, then Appendix B focus on comparing the analytical result from the free wave packet for a free Dirac particle with an execution of the quantum walk algorithm for the same type of problem. Furthermore, Appendix C presents calibration data for the `ibm_perth`, one of IBM's quantum hardware, which was used for our hybrid algorithm execution but also allows to display real values for coherence and gate processing times. Lastly, Appendix D gives the details to access the code of our algorithm and framework versions.

Chapter 2

Theoretical Foundation

In the inaugural chapter (2), we aim to lightly touch the central aspects from both the quantum computing and information domain, as well as the sphere of high-performance computing. From this study, was difficult to understand on how a HPC could have a drastic presence on the development of hybrid algorithms. All of the referred hybrid algorithms do not discuss the potential of use of the capabilities of the HPC, but more on the difficulties of the QPU (short for Quantum Processing Unit) and how can these be minimized by moving some of the quantum processing to a classical computer. In the other hand, HPC is more heavily mentioned as an important part for the simulators of quantum hardware. This will provide the requisite knowledge to fuel curiosity to propel throughout the remainder of this work.

We begin in Section 2.1.1 by showing the most important aspects of quantum information and progressively we will introduce the computation aspect. The bend between quantum mechanics and quantum computing was a detail we tried to always have present, since this is a work done under the Department of Physics of the University of Coimbra.

Then, in Section 2.2.4 a study on the most integral and recent hybrid algorithms is covered. The shortness of this section, shows the need to developed further this field. This also serves as a motivation to explore more fields where hybrid algorithms make sense and can create a substancial advantage.

Finally, in Section 2.3.1 are explained the goals for high performance computing and architecture, such in the software and hardware departments. We will also have a slight look at the available quantum simulator to be ran on HPC.

2.1 Notions on Quantum Information and Quantum Computing

2.1.1 Principles of quantum information

In the field of information theory, the term *bit* refers to two related but distinct things. The term is also used to describe a physical system with two distinct physical states. A qubit is a quantum system having two distinct (orthogonal) states [14]. These states can be labeled as $|0\rangle$ or $|1\rangle$. A qubit can hold one bit of information by virtue of it being possible to prepare it in either of these states. Due to the combination of quantum theory and information theory, a qubit differs from its classical counterpart allowing for three main special properties: **Superposition**, **Entanglement** and **Tunneling**. We will only focus on the first since it is the main property used on this work. The quantum mechanics postulates can be accessed on Appendix A.

The superposition principle tells us that the qubit can be prepared in any superposition of the states $|0\rangle$ and $|1\rangle$, that is:

$$|\psi\rangle = a_0 |0\rangle + a_1 |1\rangle \quad (2.1)$$

where a_0 and a_1 are complex numbers – sometimes called amplitude. $|a_0|^2$ and $|a_1|^2$ give the probability of measuring each state, with $|a_0|^2 + |a_1|^2 = 1$

A qubit can be represented as a two-dimensional complex Hilbert space, \mathbb{C}^2 , as we will see in the next sections. The state of the qubit at any given time can be represented by a vector in this complex Hilbert space:

$$|\psi\rangle = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} \quad (2.2)$$

We can represent the states $|0\rangle$ and $|1\rangle$ as vector as shown:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (2.3)$$

Finally, one of the most common ways to represent a qubit is on the Bloch sphere (figure 2.1). This geometric representation consists of a sphere of unit radius where each point on the sphere's surface corresponds to a qubit state. Opposite points represent a pair of mutually orthogonal states, therefore, being the north pole defined as $|0\rangle$, the south will be $|1\rangle$. The following is its mathematical representation which depends on the two spherical coordinates θ and φ :

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right) |0\rangle + e^{i\varphi} \sin\left(\frac{\theta}{2}\right) |1\rangle \quad (2.4)$$

Since the qubits are independent from each other is possible to create composite states connected via a tensor product. Considering two qubits, for example, this

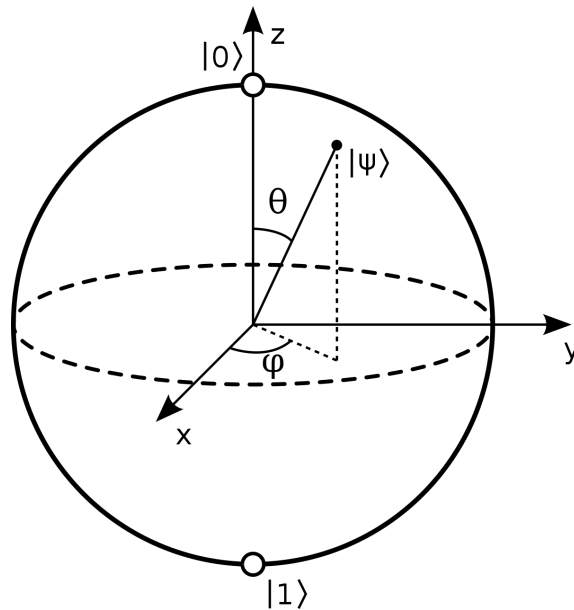


Figure 2.1: Graphical representation of the Bloch sphere. This method allows for a detail visualization of a qubit state.

allows to create computational basis states denoted by $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$, which can also be written in decimal notation $|0\rangle$, $|1\rangle$, $|2\rangle$ and $|3\rangle$. This consideration will be important for the algorithm development. Furthermore, the total number of possible combinations is determined by (2^n) , where (n) corresponds to the number of qubits. A pair of qubits can also exist in superposition of these four states, so the quantum state of two qubits involves associating a complex coefficient with each computational basis state, such that the state vector describing the two qubits is:

$$|\psi\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle \quad (2.5)$$

2.1.2 Principles of quantum computation

The fundamental goal of any quantum information processor would be to enact the unitary transformation:

$$|a\rangle \rightarrow \hat{U}|a\rangle = |f(a)\rangle \quad (2.6)$$

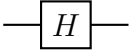
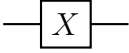
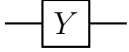

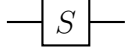

With a being any desired binary number (limited by our number of qubits n , $2^n - 1$) encoded into a state $|a\rangle$, and $f(a)$ any Boolean function of a (where $0 \leq f(a) \leq 2^n - 1$)[14].

There are different ways on how this computation can be performed, with the most common approach being the circuit model [15], which we will be using in this work.

Quantum Circuits

In computer science, circuits are models of computation in which information is carried by wires through a network of gates, which represent operations that transform the information carried by the wires. Quantum circuits are just one example

Table 2.1: Six common one-qubit gates and the associated unitary transformations

Gate	Diagram	Operator
Hadamard		$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
Pauli-X		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli-Y		$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli-Z		$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Phase		$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
$\frac{\pi}{8}$		$\begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix}$

of a model of computation based on this more general concept, implementing new characteristics as they operate according to the principles of quantum mechanics, introducing unique properties.

Quantum circuits, comprising a finite sequence of events, beginning with an initial quantum state ($|a\rangle$) and aim to transform it into a desired state ($|b\rangle$). It results from the successive application of quantum operators to all or specific pairs of qubits. These can be decomposed into combinations of **CNOT** (quantum equivalent of XOR classical gate) and single qubit gates, in a similar way to the boolean classical logic gates, important architectural feat as we will see in the DiVincero criteria in the upcoming pages. Quantum gates, acting on one or multiple qubits in a single operation, facilitate this transformation, provided they satisfy the condition of being unitary matrices ($GG^\dagger = \mathbb{I}$). As the number of qubits increases, the dimensionality of the matrix operators representing these gates expands exponentially.

In the table 2.1 are detailed the most popular one qubit gates.

The most commonly encountered two-qubit gate is the controlled-NOT or CNOT gate. This gate acts on the state of two qubits, known as control qubit and target qubit. For short, if the control qubit is $|0\rangle$ the target qubit does not change but if it is $|1\rangle$ the Pauli-X matrix is applied to the target qubit. The CNOT shares in common the classical XOR gate from classical logic:

$$|A\rangle \otimes |B\rangle \rightarrow |A\rangle \otimes |B \oplus A\rangle \tag{2.7}$$

Controlled gates can have n amount of control qubits and can have any type of gate applied to the target qubit. For these types of gates to be *activated*, all the n control qubits need to be in state $|1\rangle$, or more precisely, be on state $|1\rangle^{\otimes n}$.

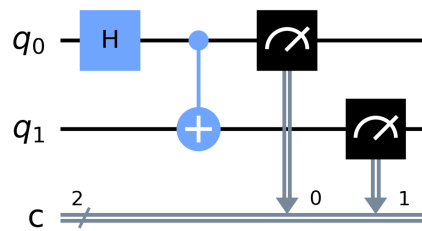


Figure 2.2: Graphical representation of the entanglement circuit we’ll use as a practical example. Our horizontal lines represent either qubits (q_0, q_1) or bits (c_2). Gates are applied to the qubits which can affect one or more of them. The CNOT gate performs operations that involve more than one qubit, similar to the classical NOT gate. The Hadamard gate is one of many one qubit gates. Finally, the measurement blocks on the qubits store the results to classical bits.

Table 2.2: List of the most popular multi-qubit quantum gates.

Gate	Diagram	Operator
CNOT (CX)		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
Toffoli (CCNOT)		$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$
Controlled-U (Unitary Gate)		$\begin{pmatrix} I_2 & 0 \\ 0 & \hat{U} \end{pmatrix}$

The graphical representation of quantum circuits is general and quite intuitive. More details can be found on the figure 2.2.

Finally, opposite to the classical counterpart, the act of measurement that seeks to obtain classical information about a quantum state has a profound effect on its observation. According to the Measurement Postulate (Appendix A), a projective measurement is described by a Hermitian operator \mathcal{O} , called observable in the state space of the system being measured. The measurement in the computational basis of space \mathbb{C} is the set $\{|0\rangle, |1\rangle\}$ and therefore, for one qubit, the measurement should be in the Pauli-Z basis. In the circuit model the symbol for measurement is described in

the end node of figure 2.2. This element converts a single qubit into a probabilistic classical bit, denoted by the two parallel lines. Given the inherent uncertainty—since we can't directly observe the quantum state, as our retrieved information resides in classical data—the execution of a specific circuit must be repeated multiple times. This repetition allows us to infer a probability distribution, furnishing us with a more comprehensive understanding of the quantum system's behavior.

One last element is subject to this information-processing can be imperfect with its operations leading, inevitably, to errors. In classical computations this is not such of a problem because quantum systems tend to be significantly smaller than their classical counterparts and are far more susceptible to environmental influences. Secondly a classical bit only has two possible states, where a qubit can be prepared in an infinitely number of quantum states and also can be modified in a wide variety of ways by interaction with its environment. The most predominant protocols for error-correction are the Shor's[16] and Steane's[17] error-correcting protocols. These require extra qubits (nine or seven respectively), to protect a single logical qubit.

Having explored the intricacies of the circuit model in quantum information processing, it is enlightening to consider alternative approaches that offer unique advantages. Such approach is the one-way-measurement [18] where a large entangled state, called the resource state, is prepared and measurements are performed on individual qubits to generate the unitary evolution of the computation. This approach is different from the circuit model as it relies on a sequence of measurements to generate the desired quantum state. It is *one-way* because the resource state is destroyed by the measurements.

Farhi et al.[19] introduced Adiabatic quantum computation (AQC) in which the evolution of a quantum system is controlled by slowly varying a Hamiltonian over time. The final state of the system encodes the solution to a computational problem. AQC is based on the idea that if a system is prepared in its ground state at the start of the evolution and if the Hamiltonian is slowly varied, the system will stay in its ground state throughout the evolution and will reach the ground state of the final Hamiltonian.

2.1.3 Quantum computers: physical realization

To implement a qubit, any type of quantum system with two quantum states can be used to provide a physical implementation of a qubit. Is important to emphasize that the qubit is a conceptual framework used to explain and calculate quantum information processing and communication — it is not tied to any specific physical realization. Also a quantum computer has to be well isolated in order to retain its quantum properties, but at the same time qubits have to be accessible so that they can be manipulated to perform a computation and to read out the results.

The decay of the quantum states produced by the quantum computer establish a limitation when performing complex quantum algorithm. Quantum decoherence is usually characterized by measuring two constants: The $T1$ and $T2$ times, commonly known as thermal relaxation time and dephasing time respectively. These often allow to quantify the noise. $T1$ is defined as the time needed for a qubit to move from the

excited state $|1\rangle$ to the ground state $|0\rangle$. $T2$ is defined as the elapsed time before a qubit's resonance frequency becomes unidentified. $T2$ could be thought of as the loss of quantum coherence over time. In Appendix C is provided a table with values for $T1$ and $T2$ times of real quantum hardware, plus some more details that define its fidelity, which we will not have time to discuss.

The DiVincenzo criteria, proposed by DiVincenzo[20], establish the conditions required to realize a quantum information processor. There are five criteria, providing the means by which we can compare developments in competing systems and measure progress towards a quantum computer:

- Well-defined state space - scalable system and have well-defined qubits;
- Initialization - capability of initializing the system of qubits in an unique pure state;
- Long coherence times - preventing qubits from decoherence, causing data loss and introducing error to computations;
- Universal set of quantum gates - similar to classical gates, be possible to have a gate or a set of gates that can, in turn, create any other possible configuration of gates. This is possible to an extremely good approximation, result of the Solovay-Kitaev theorem [21];
- Qubit-specific measurements - obtain the state of the qubit accurately.

Some of the current examples of physical realizations of a quantum computer are trapped ions, nuclear magnetic resonance, cavity quantum electrodynamics, coupled quantum dots and most notably superconducting Josephson junctions (also popularly referred as superconducting qubits). The latter is responsible for the most advanced and complex quantum computers, or more precisely Noisy Intermediate-Scale Quantum (Noisy Intermediate Scale Quantum) computers. As of the date of publish of this work, the most advanced NISQ quantum processing unit is the IBM Osprey [22], with 433 qubits. Another breakthrough has been published by Zhong et al.[23] who admit to have accomplished quantum supremacy (capability of a quantum computer to solve a problem any other device can) by using a quantum computer based on photons, the first of its kind to reach such goal.

Another great achievement is the availability of some real quantum hardware, product of the revolution in cloud computing, allowed with a single internet connection.

2.1.4 Notable quantum algorithms

Quantum algorithms have the potential to revolutionize various industries with their innovative applications. Two of the most notable quantum algorithms are the Shor's Algorithm and Grover Algorithm.

The Shor's Factoring Algorithm [24] aims to find the prime factors of an integer by identifying a number, m , that divides it exactly. Given an integer N , the al-

gorithm’s task is to find this number m . It provides an efficient method for factoring and presents a significant threat to public-key cryptosystems which resulted in widespread interest in quantum computation. Its efficiency is derived from the Quantum Fourier Transform as a method for determining the period of a function.

The Grover Algorithm [25] is a quantum search algorithm that allows us to find a specific item in an unsorted/unstructured database with a quadratic speedup, $\mathcal{O}(\sqrt{N})$, over classical algorithms, $\mathcal{O}(N)$. For example, given a function that returns 1 for the desired item and 0 for all other items, the algorithm’s task is to find the input that produces an 1. It provides an efficient method for searching. The main advantage of Grover’s algorithm is that it doesn’t require any additional information about the structure of the data, it only requires the ability to evaluate the function that encodes the problem.

2.2 Hybrid Quantum-Classical Algorithms

Hybrid quantum-classical computing has seen significant advancements in both the hardware and software components, as well as in the development of new algorithms. Despite the hardware limitations, it has not constrained the proposal of new algorithms and efforts to create a good relation between classical and quantum domains.

Many of the algorithms that will be presented, have proven slightly worse or equal results when compared with the classical computations using state of the art HPCs. These results are the foundations for the true potential of hybrid quantum-classical computations and can draw a path for the quantum supremacy.

It is also important to note than some of the work done on this field has been problem related which does not trace a clear path for modular and flexible algorithms, which should be the goal for the NISQ era.

2.2.1 Variational Quantum Eigensolver

In the spectrum of hybrid quantum computing, the Variational Quantum Eigensolver [26] is seen as one of the most influential algorithms given the the practical abilities it provides combining classical computation with NISQ computers.

In its essence, this algorithm finds variational solutions to eigenvalue problems, parameterized by quantum states. The latter is computed in a quantum system whereas the classical computer *guides* the obtained solutions to a general minima/maxima by optimizing the state parameters.

Mathematically, we want to obtain the minimal energy (E_0) for a given Hamiltonian (\mathcal{H}):

$$\mathcal{H} |GS\rangle = E_0 |GS\rangle \quad (2.8)$$

Where $|GS\rangle$ is the state wave function which corresponds to the ground state.

The variational method [27] tells us there is a state wave function that dictates an upper, or in the best of cases, equal limit to the minimal energy of a given Hamiltonian.

$$E_{GS} \leq \lambda_\theta \equiv \langle \psi(\theta) | \mathcal{H} | \psi(\theta) \rangle \equiv \langle \mathcal{H} | \mathcal{H} \rangle_\theta \quad (2.9)$$

Considering a test wave function $|\psi(\theta)\rangle$, with a given number of adjustable parameters (e.g. θ), calculating repeatedly $\langle H | H \rangle$ for successive parameters values is possible to converge to a result close or even equal to the E_{GS} .

The generation and transformation of wave functions is relatively straightforward in a quantum computer:

$$|\psi(\theta)\rangle = U(\theta) |0\rangle$$

where:

- $|0\rangle$ - initial state;
- $U(\theta)$ - responsible gate for the input state manipulation. This gate should be modeled to the dimension and complexity of the problem in question;
- $|\psi(\theta)\rangle$ - new (candidate) state.

The next step is in regard of the optimizer architecture. This will operate in a classical computer, as the hybrid quantum-classical algorithm. Peruzzo et al.[26] has implemented the **Nelder-Mean (NM)**[28], a numerical optimization method for function minimization and which is unconstrained.

Following the proposal of Peruzzo et al.[26], many new *ansatzes* have been developed to achieve higher quantum efficiency [29]. These mainly focus on imposing symmetries associated with particle number, spin and time-reversal symmetries. Additionally, these are based on the capabilities of the hardware and perform state preparation by combining parameterized gates available on the processor. However, despite these efforts, hardware noise remains a major limiting factor for the success of this approach [30]. Examples of more efficient *ansatzes* include the qubit coupled cluster (QCC) [31] and Quantum subspace expansion (QSE) [32].

2.2.2 Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm [33] (QAOA) is an algorithm dedicated to combinatorial optimization problems ¹ in NISQ computers.

This algorithm uses the unitary matrix $F(\beta, \gamma)$ characterized by the parameters (β, γ) to prepare the state $|\psi(\beta, \gamma)\rangle$. The goal of this algorithm is to find values of $(\beta_{opt}, \gamma_{opt})$ that minimize the state, $|\psi(\beta_{opt}, \gamma_{opt})\rangle$.

To achieve $|\psi(\beta_{opt}, \gamma_{opt})\rangle$, we begin by defining an initial test state $|s\rangle$ that corresponds, for example, to a superposition of states: $|s\rangle = \frac{1}{\sqrt{2^n}} |z\rangle$.

$$\langle C|C\rangle = \sum_{z \in \{0,1\}^n} f(z) |a_z|^2 = \frac{1}{2^n} \sum_{z \in \{0,1\}^n} f(z) \quad (2.10)$$

Thus, we have a given probability of obtaining the state that maximizes (or minimizes), in this example $\frac{1}{2^n}$ of obtaining it. On the other hand, we can apply a rotation to $|s\rangle$ in order to obtain more interesting results, that is, to maximize the probability of measuring the state corresponding to the maximum (or minimum) eigenvalue.

We can define a rotation given by:

$$U(C, \gamma) = e^{-i\gamma C} = e^{-i\gamma \sum_{\alpha=1}^m C_{\alpha}(z)} = \prod_{\alpha=1}^m e^{-i\gamma C_{\alpha}} \quad (2.11)$$

This rotation translates to that whenever our condition C_{α} is satisfied, there is a rotation of $e^{-i\gamma}$. $\gamma \rightarrow [0, 2\pi]$ can be seen as the weight of C_{α} relative to a given input. Otherwise, the state remains the same. However, by just applying this rotation, there is still no change in probability, so we will introduce the operator $U(B, \beta)$.

We can define the rotation operator $B = \sum_{j=1}^n \sigma_j^x$ and β (which has a similar role to that of γ since represents the *weight* with which B is relative to a given input):

$$U(B, \beta) = e^{-i\beta B} = \prod_{j=1}^n e^{-i\beta \sigma_j^x} \quad (2.12)$$

Knowing that the x-axis rotation operator has the following aspect:

$$R_x(\theta) = e^{-i\frac{\theta}{2\sigma^x}} \quad (2.13)$$

¹Combinatorial optimization is the process of searching for the maximum (or minimum) of a given objective function. It is an NP-complete problem. These types of problems are specified by n bits and m clauses, which are simply the rules for the limits of the subset of bits. For example, $C(z) = \sum_{\alpha=1}^m C_{\alpha}(z)$, where $C_{\alpha}(z) = 1$ or $= 0$ if it satisfies clause α or not, respectively. We can also define it through $C|z\rangle = \sum_{k=1}^m C_k(\langle) |z\rangle = f(z) |z\rangle$, where we want to find the eigenvalue, C_k that maximizes (or minimizes) our function.

Applying the two rotations in succession does not guarantee that it is possible to obtain the state corresponding to the maximum (or minimum) eigenvalue. To do so, these are applied $2p$ times in a single operation.

This calculation takes advantage of the capabilities of a NISQ and the parameters γ and β are then optimized using a classical computer, in similarity to the VQE 2.2.1. This process is repeated until it converges to a value close enough or equal to the maximum (or minimum).

As of the likes of the VQE 2.2.1 these algorithms have also been worked on to enhance its quantum efficiency so their more resilient to quantum noise. Recent suggestions to change the circuit model [34] have shown benefits in terms of improved performance when benchmarking using the Max-K-problem.

2.2.3 Hybrid Quantum Monte Carlo

Until now, hybrid quantum classical computation has been tackled as an optimization problem, but this method is far from being the only capable of bringing applications to the NISQ era.

An innovative idea was presented by Huggins et al.[35] which departs from the optimization problems and allocates both the quantum and classical computers to different tasks.

First, Quantum Monte Carlo (QMC) is a powerful method to study the properties of many-body quantum systems, such as electrons in a solid or atoms in a gas, by simulating the quantum system using a classical computer. It is based on the idea of stochastically sampling the many-body wave functions using a random walk algorithm [36]. Mathematically, we will have i $|\phi_i\rangle$ oversimplified descriptions which in a weighted average represent a given state. Unfortunately this method will run into the fermionic sign problem [37]. This can be solved by overlapping the $|\phi_i\rangle$ with a trial wavefunction $|\psi_T\rangle$ (which should be an estimator of the ground state energy) and demanding they remain positive, if not they should be excluded.

Since there is no efficient classical algorithm to estimate the overlap and any wavefunction that can be prepared with a quantum circuit is a candidate for a trial wavefunction $|\psi_T\rangle$ on a quantum computer, there is an advantage in evaluating the overlap $\langle\psi_T|\phi_i(\tau)|\psi_T|\phi_i(\tau)\rangle$ in a quantum computer.

Huggins et al.[35] obtained experimental results to practically demonstrate this algorithm. For the H_4 in a square geometry, the results were competitive with classical state of the art methods. This experiment also showed better results when comparing with the VQE counter part.

It is still to be seen a fundamental work to generalize this procedure.

2.2.4 Hybrid Machine Learning

Quantum Machine Learning Models

Quantum Machine Learning (QML) can show advantages over the classical counterpart, but is wrong to assume that it can leverage *any* classical ML algorithm [38]. In this section we will look at the work done on Quantum Machine Learning Models, namely the foundation that helped to develop this module.

Schuld et al.[39] present that it is possible to build a parameterized (θ) quantum model ($f_\theta(x)$), by using quantum gates as a sum of partial Fourier Transformations:

$$f_\theta(x) = \sum_{\omega \in \Omega} c_\omega(\theta) e^{i\omega x} \quad (2.14)$$

Which translates to an expectation value of some observable (M), our measurement operator ², with respect to a state prepared via a parameterized quantum circuit ($U(x, \theta)$):

$$f_\theta(x) = \langle 0 | U^\dagger(x, \theta) M U(x, \theta) | 0 \rangle \quad (2.15)$$

Focusing on $U(x, \theta)$, this composite gate has L layers each one with a data-encoding block $S(x)$ and trainable circuit block ($\mathcal{W}^{(i)}(\theta)$). In each layer the $\mathcal{W}^{(i)}(\theta)$ can have different parameters but $S(x)$ is always the same. $S(x)$ consists of the gate \mathcal{G} , responsible for encoding our classical input (x). Encoding procedure used (that can be applied for full-scale quantum computers [40] and also for NISQ [41]) follows the presentation in a n -qubit $x \rightarrow |01011\rangle$ where x is mapped in an angle $\phi(x) = [\phi_1, \dots, \phi_n]$ where the rotation values represent the qubit values.

Regarding expressivity of such quantum models, in other words the ability of the model's architecture to represent and approximate a broader array of functions, Schuld et al.[39] establish that the given algorithm just needs to be scaled in multiple blocks, in parallel \downarrow (with dimension d) or in serial \rightarrow (with dimension L). There are limitations in both hardware and also due to the model's architecture. The first related to the number of available qubits, coherence time and CNOT error. The latter is referred to the frequency spectrum that the model can support and the control of the Fourier coefficients. The size of the frequency spectrum is upper-bounded by the size of the encoding gate, and the scaling of the degrees of freedom needed to control the Fourier coefficients is proportional to the size of the frequency spectrum. Therefore, the accessible frequency spectrum should be asymptotically rich enough so these models can be universal function approximators.

²The measurement operator determines the properties of the system that will be observed and provides the output of the model $f_\theta(x)$.

Quantum Optimization

Stochastic gradient descent (SGD) is an important technique in quantum machine learning that allows for the efficient optimization of quantum circuits using a form of gradient descent optimization that utilizes stochastic measurement outcomes. This method, known as hybrid quantum-classical optimization, is used to optimize a wide range of quantum algorithms such as VQE 2.2.1, QAOA 2.2.2, and certain quantum classifiers. It provides a framework for the derivation of rigorous optimization results in the context of near-term quantum devices (NISQ) and can lead to state-of-the-art results with significantly fewer circuit executions and measurements. This can result in a practical advantage over classical counterparts. In a recent work, Sweke et al.[42] formalize and solidify the approach of optimization using quantum systems with the Stochastic Gradient Descent for both hybrid quantum-classical algorithms and generic loss functions, such as the mean-square-error classifier. These also show good convergence for VQE, QAOA, and MSE quantum classifiers.

Hybrid Decision Trees

Similarly to the presented in 2.2.4 a proactive field of research are Hybrid Decision Trees, a popular and powerful tool for classification or regression algorithms, using quantum-classical machines. Sun and Zheng[43] explore the relationship between the quantum time (depth of the quantum circuit) and the accuracy of the decision tree by demonstrating that longer quantum time leads to a strictly more powerful decision tree, or in other words, a decision tree with greater accuracy and efficiency. It also serves as a foundation on how this type of problems should be documented.

Similarly, Chia et al.[44] had already proven the need of large quantum depth even in hybrid solutions.

2.3 High Performance Computing

High Performance Computing (HPC) is a collection of technologies of hardware and software whose aim is to solve computationally challenging problems in areas of science, technology and sociology. With its advanced hardware and software technologies, HPC has proven to be a formidable tool in addressing computationally challenging tasks. Unlike quantum computers, HPC systems have been developed and refined over many years, resulting in a high degree of reliability and stability.

Additionally, HPC systems are highly scalable and can be easily expanded to meet the demands of even the largest computational problems. The software used in HPC systems is also well established, with a vast library of algorithms and tools that can be leveraged to solve problems. Moreover, HPC systems are more cost-effective than quantum computers and more widely available, making them accessible to a broader range of users and organizations. This is especially important for many scientific

and engineering communities that rely on large-scale simulations to advance their research.

In the following paragraphs will detail some of the most critical technologies that are important to the functioning of HPCs:

Probably the most meaningful and impactful is Parallel Computing, the execution of many operations at a single instance in time [45]. It allows to handle big data problems (up to millions of millions of degrees of freedom) or even problems that were previously unsolvable. In addition, it is more energy, time and cost efficient when compared to the serial process (single computational core performance). The *Task Farming* approach is a parallel computing method that processes large-scale computational problems by breaking them down into smaller tasks, distributed among multiple processing units. Therefore, it allows for a brutal speed-up for data analysis. Is important to mention that there is a limit for parallel computing described by the Amdahl's Law [46]. Amdahl[46] states that no matter how fast we make the parallel part of the code, we will always be limited by the serial portion:

$$\text{SpeedUp}(N) = \frac{1}{S + \frac{P}{N}} \quad (2.16)$$

where,

P	parallel fraction of the code
S	serial fraction
and	$P + S = 1$
N	number of processors

Supercomputing software offers various paradigms for developing parallel programs, including shared memory systems with OpenMP and distributed memory systems with MPI. OpenMP uses compiler directives and is easier to program for parallel processing, but is limited to a single (or in other words a single server) from the computing cluster. On the other hand, MPI, as a software library, is crucial for tasks that require large amounts of memory and allows for distributed memory systems, allowing for the combination of memory from multiple servers. In practice, a combination of MPI and OpenMP is often used to achieve maximum performance.

The advancement of high-bandwidth dedicated interconnects with low latency facilitates scalability in high performance computing and enables the creation of distributed memory systems, also known as clusters (and the nodes are each processing unit). This leads to the improvement of C/C (computation-to-communication) ratio [47] by maximizing the computation time and minimizing communication time, where the latter can slow down the process. The standard for interconnects architecture is InfiniBand [48] being the NDR 400G InfiniBand [49] the latest generation of the architecture providing communication speeds up to 400 Gb/s.

GPGPUs (General Purpose Graphics Processing Units) [50] are hardware components designed to tremendously boost the performance of computing tasks. They offer high memory bandwidth and computational power and feature many programmable units that are capable of handling complex mathematical operations, including vector operations and IEEE floating-point precision. This allows for faster and more efficient processing of data, making GPGPUs an useful technology in a wide range of fields. These hardware component are usually benchmarked in terms of FLOPS (Floating Point Operations Per Second) which relates, as the name suggests, to x amount of floating-point operations per second. NVIDIA Tensor Cores [51] are some of the most advanced hardware accelerators, with the NVIDIA H100 Tensor Core GPU [52] their flagship GPU. This hardware is complemented with software development tools such as CUDA and OpenACC.

The most advanced and powerful HPC in the world is the *Frontier* [53], with 9408 CPUs (602 112 cores) and 37632 GPUs totalling 1,1 exaFLOPS. At the European level, the EuroHPC JU [12], a joint initiative between the EU, European countries and private partners are responsible for the LUMI [54] a pre-exascale HPC with a peak performance of 550 petaFLOPS. This initiative is also working on the first European exascale supercomputer, JUPITER [55]. The European Union also guaranteed a funding programme [56] focused on *digital technologies to businesses, citizens and public administrations* where HPC technologies and EuroHPC are included.

2.3.1 High performance computing and quantum computing

When mentioning both HPC and Quantum Computing in the same sentence, the most popular result are the use of highly capable computers to simulate quantum hardware, i. e., as quantum simulators.

These platforms have the goal to implement and simulate quantum algorithms without the need for access to quantum machines. These mostly use CUDA, MPI and OpenMP allows programs to run on a wide variety of systems. It not only helps on testing proof of concepts for quantum algorithms but also as a verification of results when running on the real-quantum counterpart.

Some of these frameworks are the following:

- Qiskit-Aer [57] - developed by IBM's Qiskit team, qiskit aer works as a complementary for the qiskit (main framework), providing the simulation of quantum algorithms, designed and transpiled using the IBM's software, on classical computers. It provides high-performance quantum computing simulators with realistic noise models, designed for HPC.
- NWQ-Sim [58] - NWQ-Sim is implemented in C++/CUDA/HIP for general full-state quantum circuit simulation. It uses internal gate representations for advanced optimization and profiling. The statevector simulator (SV-Sim) is designed for high-performance ideal simulation and the density matrix simulator (DM-Sim) for noise-aware simulation.

- hybridq [59] - HybridQ is a highly extensible platform designed to provide a common framework to integrate multiple state-of-the-art techniques to simulate large scale quantum circuits on a variety of hardware. HybridQ provides tools to manipulate, develop, and extend noiseless and noisy circuits for different hardware architectures. HybridQ also supports large-scale high-performance computing (HPC) simulations, automatically balancing workload among different processor nodes and enabling the use of multiple backends to maximize parallel efficiency.
- quantuloop - the Quantuloop Quantum Simulator Suite for HPC is a collection of high-performance quantum computer simulators for the Ket programming language (open-source platform that provides dynamic interaction between classical and quantum data at the programming level, streamlining classical-quantum development).

Chapter 3

Quantum Walks

For this chapter, we will focus on the coined discrete-time model (DTQW) [2, 7]. A basic discrete time quantum walk is specified by a particle walking on a lattice, at each time step moving from its current site position to another in its vicinity, and scattering through self-interaction. The versatility of DTQW extends its utility to numerous problem domains, including the simulation of the free Dirac particle equation. Therefore, despite the existence of the Continuous Time Quantum Walk (CTQW), employing DTQW may often prove more advantageous due to its broad applicability and easy implementation in discrete time environments. Such use will be with the Dirac equation as we will see in the upcoming sections. We will consider the simplest of cases, an one dimensional walk on a finite lattice. Usually this is called a (1+1D) which relates to the sum of space dimension (1D) with time dimension (1).

Therefore, we will begin by focusing on describing the coined quantum walk, in its discrete form, (section 3.1), highlighting some important aspects that will enable us to use this tool in developing new algorithms. Still inside this topic we will bring to light the Dirac quantum walk which tends to implement the simulation of a Dirac free particle using the architecture provided by the coined discrete time quantum walk. It considers the fundamental correlations between the discrete time quantum walk and Dirac free particle equation. Next we will explore the implementation of a potential well also using this tool. In the last chapter (4) we will demonstrate a use case to create an algorithm that simulates the Dirac-free particle inside a potential well, using quantum walks.

Subsequently, we will explain thoughtfully the algorithm development for the quantum walk, with consecutive architecture iterations to seek efficiency, workflow that was part of this project. Also in this section, we will present an implementation of a hybrid algorithm concept whose results will be shown and discussed in the f chapter. Besides we will also present the method to implement potential wells using position-dependent coins. Coherent data visualization for the generated product from the quantum walk algorithm will be discussed and covered. In the last segment, section 3.2.5, we will present the framework used to create and code the algorithm, its pros and cons and the rationale to use this instead of the other competitive frame-

works available. The available resources, such as classical simulator and quantum hardware, for algorithm execution have additionally been discussed briefly in this section.

3.1 Coined Quantum Walk

The walked position x can be represented as a vector in a N -dimensional Hilbert space \mathcal{H}_P , where the computational basis of which is $\{|x\rangle : x \in \mathbb{Z}\}$ and is encoded in base 2. The number of qubits, n , defines the total number of possible positions $N = 2^n$. The position state can therefore be represented as:

$$|x\rangle = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} (0) \\ (1) \\ \vdots \\ (x) \\ \vdots \end{matrix} \quad (3.1)$$

Each step of the walk moves one position to the "left" $|x\rangle \rightarrow |x - 1\rangle$ or to the "right" $|x\rangle \rightarrow |x + 1\rangle$ where this evolution depends on a quantum "coin". For example when one obtains "heads" after tossing the "coin" the walker moves to the "right" and if it is "tails" it moves to the "left". The "coin" can be represented in two-dimensional Hilbert space \mathcal{H}_C , the computational basis of which is $\{|0\rangle, |1\rangle\}$, respectively.

The Hilbert space of the system should be:

$$\mathcal{H} = \mathcal{H}_C \otimes \mathcal{H}_P \quad (3.2)$$

$$\mathcal{H} = \mathbb{C}^2 \otimes \mathbb{C}^N \quad (3.3)$$

The movement of the walker is modeled by the so called *shift operator* S which should operate as follows:

$$S |0\rangle |n\rangle = |0\rangle |n + 1\rangle \quad (3.4)$$

$$S |1\rangle |n\rangle = |1\rangle |n - 1\rangle \quad (3.5)$$

Given that the shift is restricted to neighboring zones with a step length of one, the operator can be translated to the general form:

$$S = |0\rangle \langle 0|_C \otimes \sum_{n \in \mathbb{Z}} \langle n - 1|_P \langle n|_P + |1\rangle \langle 1|_C \otimes \sum_{n \in \mathbb{Z}} \langle n + 1|_P \langle n|_P \quad (3.6)$$

The coin toss operator can be described by a three-parameter $SU(2)$ [60] unitary matrix:

$$C_{\xi, \theta, \zeta} = \begin{bmatrix} e^{i\xi} \cos \theta & e^{i\zeta} \sin \theta \\ -e^{-i\zeta} \sin \theta & e^{-i\xi} \cos \theta \end{bmatrix} \quad (3.7)$$

sufficient to describe the most general form of the discrete-time quantum walk.

A step in a quantum walk consists in applying the following unitary operator:

$$U = S (C \otimes \mathbb{1}) \tag{3.8}$$

In the quantum case, if we measure the particle position after the first step, we destroy the correlations between different positions, typical of quantum systems. Therefore we should apply U (3.8) successively allowing for the quantum correlations between different positions generating a different behaviour to the classical counterpart. After those t steps we make our measurement to uncover the final position of the particle:

$$|\psi(t)\rangle = U^t |\psi(0)\rangle \tag{3.9}$$

The methodology described here can be used as a framework to create quantum walks in two different types of spaces: infinite and cycled. The latter result in an algorithm architecture that we will discuss in depth in the following section.

Consider using a Hadamard gate as the "coin" operator C :

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.10}$$

On an initial "balanced" state:

$$|\text{coin}(t = 0)\rangle = \frac{|0\rangle + i |1\rangle}{\sqrt{2}} \tag{3.11}$$

When we apply the evolution operator, terms with the imaginary unit are not converted into terms without the imaginary unit and vice versa. There will be no cancellation of terms of the walk that goes rightward with the terms of the walk that goes leftward. At the end, the probability distributions are added and we should obtain a symmetrical distribution (Figure 3.2).

The formula for the standard deviation of the probability distribution is:

$$\sigma(t) = \sqrt{\sum_{n=-\infty}^{\infty} n^2 p(t, n)} \tag{3.12}$$

where $p(t, n)$ is the probability distribution of the quantum walk.

The standard deviation in a classical random walk, which can be seen as the non-quantum counterpart to a quantum walk, typically exhibits a square-root relationship with time. However, in a quantum walk, when calculating 3.12 numerically [2], the standard deviation presents an unique characteristic: it shows a direct, linear dependence on time. This distinctive behavior highlights the more expansive spread inherent in quantum walks when compared to their classical equivalents.

To end this section with an interesting perspective, quantum walk position space can be modelled in a network graph , where each node represents a possible position.

Table 3.1: Probability of finding the quantum particle in position x at time t , assuming that the walk starts at the origin with the quantum coin in “tails” state, $|0\rangle$.

$t \backslash n$	-5	-4	-3	-2	-1	0	1	2	3	4	5
0						1					
1					1/2		1/2				
2				1/4		1/2		1/4			
3			1/8		1/8		5/8		1/8		
4		0		1/8		1/8		5/8		0	
5	0		1/6		1/8		1/8		1/2		0

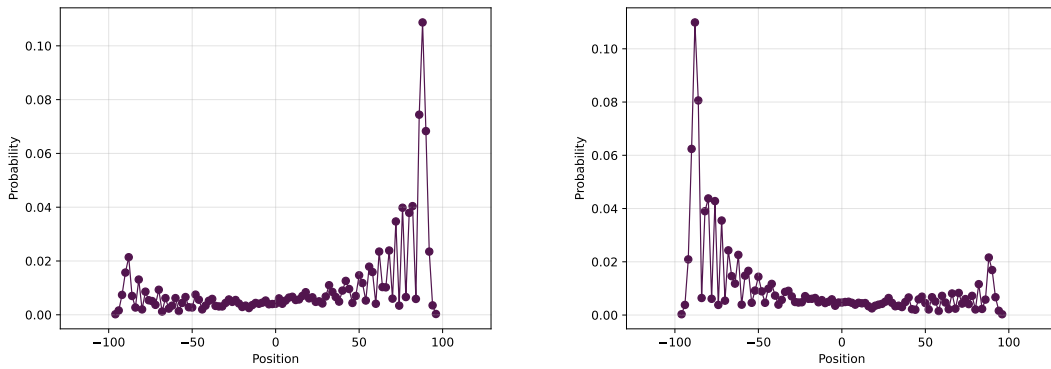


Figure 3.1: Probability distribution for different initial coin states, "tails" $|0\rangle$ and "heads" $|1\rangle$, respectively.

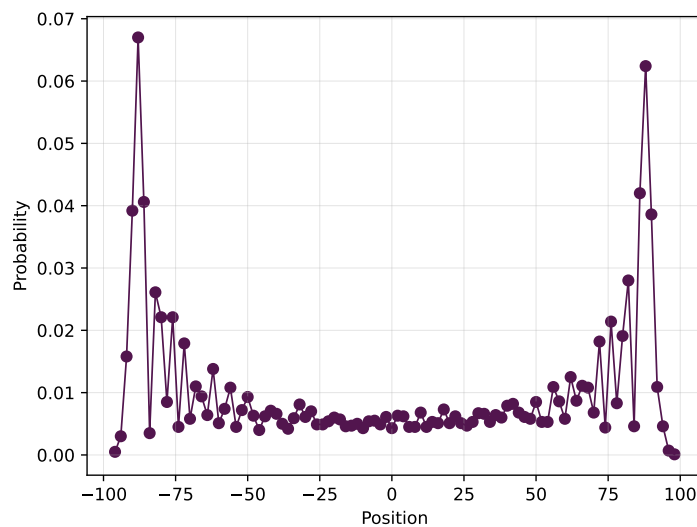


Figure 3.2: Symmetrical probability distribution using 8 qubits and $t = 2^8 - 1$ steps simulated on quantum computer simulator `qasm_simulator` from IBMQ.

This helps to open new algorithmic frontiers to new or already existing applications, allowing to solve and tackle complex network graphs plus for possible increments in efficiency. In figure 3.3 we can observe the network graph for the cycled quantum walk and for the quantum walk with frontiers on its ends (where we will focus more deeply on the potential well implemented into the quantum walk).

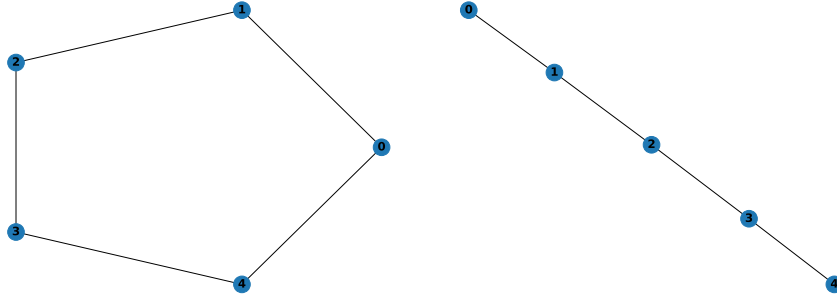


Figure 3.3: Graph networks for a cycle network (left) and for a limited line (right).

3.1.1 Dirac quantum walk

One particular important application for quantum walks is the relation between them and the Dirac Equation. The Dirac Equation is a relativistic wave equation that describes the behavior of fermions, particles that follows Fermi–Dirac statistics and obey the Pauli exclusion principle (i.e., no two fermions can occupy the same quantum state). The equation was first proposed by British physicist Paul Dirac in 1928 and has since become a cornerstone of modern physics.

The connection between quantum walks and the Dirac equation is rooted in the fact that both describe the behavior of quantum particles. Specifically, it has been shown that the Dirac Equation can be expressed in terms of a quantum walk on a lattice [61–63]. This means that by studying the behavior of quantum walks, we can gain new insights into the behavior of fermions and the properties of the Dirac Equation.

The applications of this connection are far-reaching. Such application is in research of properties of the fermions in a material or system such as conductivity, magnetism, and superconductivity. Also Cellular Automata [64] or even as using transformations for the Dirac Equation on fluid dynamics [65].

The free Dirac Hamiltonian operator for a particle with zero momentum along the y and z directions is:

$$\mathcal{H} = c\alpha_z\hat{p}_z + mc^2\beta \quad (3.13)$$

$$\text{with, } \hat{p} = -i\hbar\frac{d}{dx} \quad (3.14)$$

And also the matrices:

$$\alpha_z = \begin{bmatrix} 0 & \sigma_z \\ \sigma_z & 0 \end{bmatrix} \quad \beta = \begin{bmatrix} \mathbb{1}_2 & 0 \\ 0 & \mathbb{1}_2 \end{bmatrix} \quad (3.15)$$

where σ_i , $i = 1, 2, 3$ are the usual Pauli matrices and $\mathbb{1}_2$ is the 2×2 unit matrix.

Demonstrated by Bracken et al. (Sec. II) [62] we can consider an Hamiltonian in the effective form:

$$\mathcal{H}_{\text{eff}} = \sigma_3 \hat{p} + \sigma_2 \quad (3.16)$$

Assuming:

$$m = \hbar = c = 1 \quad (3.17)$$

Considering a fixed small time interval $\Delta t \ll 1/E_0$, the (effective) unitary time evolution operator for the Dirac particle is:

$$e^{-i\mathcal{H}\Delta t} = SC + O([E_0\Delta t]^2) \quad (3.18)$$

$$S = e^{-i\Delta t\sigma_3\hat{p}}, \quad C = e^{-i\Delta t\sigma_2} \quad (3.19)$$

Where SC represent the operator for an one-dimensional Discrete Time Quantum Walk, with S the Shift Operator with a step of length Δt along the x axis and C our Coin operator. Considering an unitary time space, $\Delta t = 1$ (step length) and reorganizing the Pauli operator, σ_3 , and \hat{p} :

$$\begin{aligned} S &= e^{-i(|0\rangle\langle 0| - |1\rangle\langle 1|)} \otimes (\Delta t = 1) \hat{p} \\ &= |0\rangle\langle 0| \otimes e^{-i\hat{p}} + |1\rangle\langle 1| \otimes e^{i\hat{p}} \\ &= |0\rangle\langle 0|_C \otimes \sum_{n \in \mathbb{Z}} \langle n-1|n\rangle_P + |1\rangle\langle 1|_C \otimes \sum_{n \in \mathbb{Z}} \langle n+1|n\rangle_P \end{aligned} \quad (3.20)$$

For multiple steps $t = n\Delta t$ we have:

$$e^{-i\mathcal{H}t} = (SC)^n + O([E_0\Delta t]^2) \quad (3.21)$$

By letting $n \rightarrow \infty$ and $\Delta t \rightarrow 0$ we can eliminate $O([E_0\Delta t]^2)$:

$$\lim_{n \rightarrow \infty, \Delta t \rightarrow 0} e^{-i\mathcal{H}t} = (SC)^n \quad (3.22)$$

For the DTQW, $\Delta t = 1$, therefore in order to minimize the intrinsic error the number of steps, n should be much greater than Δt . As we will see in the next sections, the optimal number of steps n_{optimal} is a trade-off between minimizing the error and exploring the entire space, which is limited by the available number of qubits. In other words, by increasing the number of qubits, that consecutively increases the explorable space, we need to perform an increasing number of steps to gain significant simulation knowledge. In addition, the simulation time will also escalate due to the increasing number of steps.

In Appendix B the reader can find a more detailed study on the one dimensional Dirac quantum walk.

3.1.2 Potential Well

The problem of a relativistic spin 1/2 particle confined to an one-dimensional box can be emulated via a relativistic quantum walk by introducing barriers or better, a squared potential well.

The behaviour of the particle trapped inside a square potential with infinite walls can be replicated by introducing a conditional coin, or in other words, position-dependent coin operator, $C^{(n)}$. This will allows to have the normal behaviour of the particle in between our barriers and more importantly to implement a barrier that *repulses* our particle, maintaining it inside the enclosure.

Following the suggestion by Nzongani et al. [66] implementing a position-dependent coin operator implies creating coin operators for each position, C_x . If implemented in a naive way, only using one qubit to handle the coin state, it places all the coin operators C_x sequentially, i.e., one after the other, along the coin wire. This already implies that the depth of the circuit will be exponential with the number of position qubits.

It terms of Dirac notation, the position-dependent coin's operator can be described as:

$$C^{(n)} = \sum_{k=0}^{N-1} |k\rangle \langle k| \otimes C_k \quad (3.23)$$

Implementing this condition for the circuit model requires some additional steps. First, $C^{(n)}$, takes the n -controlled gate form $G_n(C_x)$, as can be seen in figure 3.4. As for now and for the upcoming sections, we will assume $G_n()$ as a gate function that return a n controlled gate. It can be represented as an iterative matrix:

$$G_1(C_k) := I_2 \oplus C_k = \begin{bmatrix} I_2 & 0 \\ 0 & C_k \end{bmatrix} \quad (3.24a)$$

$$G_n(C_k) := I_{2^n} \oplus G_{n-1}(C_k) = \begin{bmatrix} I_{2^n} & 0 \\ 0 & G_{n-1}(C_k) \end{bmatrix} \quad (3.24b)$$

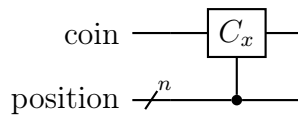


Figure 3.4: Gate for for the position-dependent coin.

Since to activate a controlled gate, all the control qubits need to be in $|1\rangle^{\otimes n}$, for a given positional state $|\psi_p\rangle$ to be activated we will need to make conditional changes, so the conditional coin can be applied to it. In other words, for a coin to be activated upon the desired position we will use X gates to change the position state, $|\psi_p\rangle$, to the $|1\rangle^{\otimes n}$. For example, with $n = 3$ and $|\psi_p\rangle = |000\rangle$ we will need to flip all the qubits before the gate, so it can be activated and also after so the position state $|\psi_p\rangle$ remains unaffected. It can be visualized in figure 3.5.

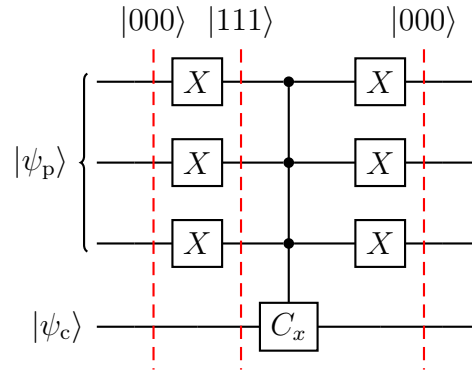


Figure 3.5: Circuit for position dependent coin. Important to note, that for the sake of simplicity we omitted $|\psi_c\rangle$ from the slice annotations.

Applying this to all the possible positions, can be done sequentially, covering all the conditional coins and without affecting the final state. Nzongani et al. refer this process as a "tower of X gates", T_n , to progressively obtain the possible positions. For example, for $n = 3$ we will have $N = 2^3$ possible position states: $|0\rangle, |1\rangle, |2\rangle, \dots, |7\rangle$ or $|000\rangle, |001\rangle, |010\rangle, \dots, |111\rangle$. We can cover all positions by progressively applying X gates. By induction we can define $T_n(i)$:

$$T_1(0) := X \otimes I_2 \tag{3.25a}$$

$$T_n(i) := M_i \otimes T_{n-1}(i \bmod 2^{n-2}) \tag{3.25b}$$

with

$$M_i := \begin{cases} X & \text{if } i = 0, \\ I_2 & \text{otherwise} \end{cases} \tag{3.26}$$

This will lead to the updated circuit evolution:

$$U^{(n)} = \prod_{k=0}^{N-1} G_n(C_k) T_n \tag{3.27}$$

In figure 3.6, we can see the circuit for $n = 1, 2$ and 3 qubits.

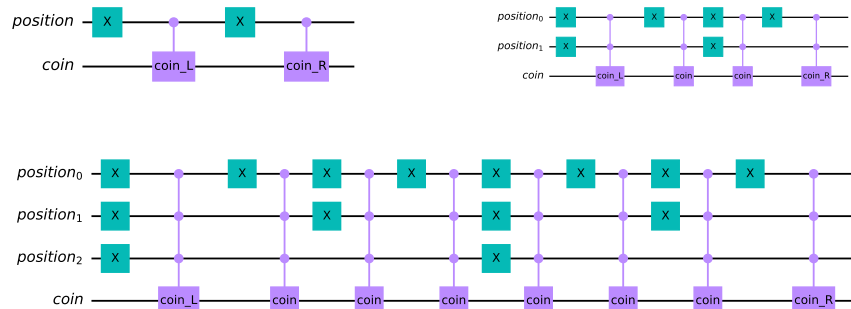


Figure 3.6: Quantum circuits for different resolutions on the position space.

3.2 Algorithm for quantum walk

Let us begin by laying out the pseudocode for the generic (1D +1) cycled quantum walk:

Algorithm 1 (1D+1) Quantum Walk

- 1: Define *position_state* as an integer lattice of n nodes
 - 2: Define *coin_state* as a two-level quantum system (qubit)
 - 3: Define *coin_operation* as a 2×2 quantum operator
 - 4: Initialize *walk_state* using *position_state* and *coin_state*
 - 5: **for** each time-step **do**
 - 6: Apply the *coin_operation* by some $SU(2)$ unitary operation on the *coin_state* of *walk_state*
 - 7: Apply the shift operator on the *walk_state* which conditionally shifts the *position_state* depending upon the *coin_state*
 - 8: **end for**
 - 9: Measure the state of the system to obtain the current *walk_state*
-

And also the generic one step quantum walk circuit:

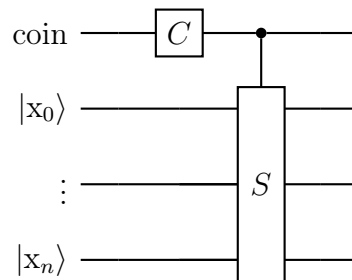


Figure 3.7: Quantum Circuit for the Quantum Walk. For it we want to implement the coin operator and shift operator.

For the sake of simplicity, the position space vector uses its natural ordering, induced on the basis by values of x , i. e. the position space is a vector with dimensions $1 \times N$, as we have seen given by the number of allocated qubits, n , for the position space, $N = 2^n$.

$$|101\rangle = |1\rangle_0 \otimes |0\rangle_1 \otimes |1\rangle_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} (0) \\ (1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7) \end{matrix} \quad (3.28)$$

Next, the right shift can be represented by a $N \times N$ matrix T :

$$T = \begin{bmatrix} 0 & 0 & 0 & \dots & 1 \\ 1 & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 & 0 \\ \vdots & \ddots & 1 & 0 & 0 \\ 0 & \ddots & 0 & 1 & 0 \end{bmatrix} \quad (3.29)$$

Therefore, $T |n\rangle = |n + 1\rangle$. For the left shift it corresponds to the transpose matrix of T , T^\top . For a self shift, our coin should be in state $|1\rangle$ and for a right shift in state $|0\rangle$. When implementing this operators, these should be dependent on the coin position, therefore we build the controlled operators:

$$T_{\text{controlled}} := G_1(T) = \begin{bmatrix} \mathbb{I}_2 & \mathbb{0} \\ \mathbb{0} & T \end{bmatrix} \quad T_{\text{controlled}}^\top := G_1(T) = \begin{bmatrix} \mathbb{I}_2 & \mathbb{0} \\ \mathbb{0} & T^\top \end{bmatrix} \quad (3.30)$$

Since the right shift is associated with the coin state $|0\rangle$, we need to add X gates to flip the space.

$$\text{Right Shift} = (X \otimes \mathbb{I}_n) G_1(T) (X \otimes \mathbb{I}_n) \quad (3.31)$$

These should take the following shape:

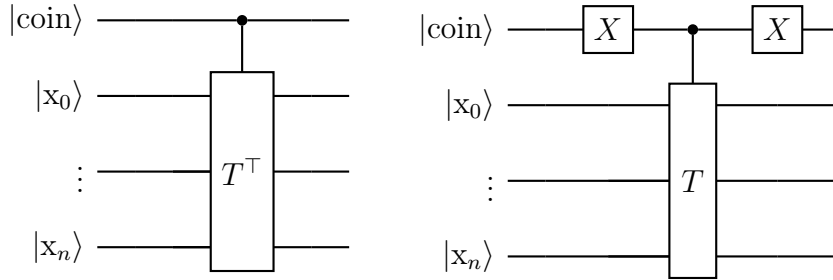


Figure 3.8: Left and Right shifts, respectively.

The shift operator can be written in the compact operator [67]:

$$\sigma = \begin{bmatrix} T & 0 \\ 0 & T^\top \end{bmatrix} \quad (3.32)$$

The time evolution repeats t times coin and shift operators. Besides this we need to prepare the initial state for the position qubits and coin qubits. The first is prepared such that represent the center point of the grid, $2^{N/2}$ (this can be achieved by simply adding an X gate to the n th qubit):

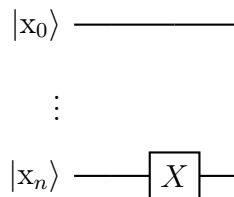


Figure 3.9: Initial position state preparation circuit.

For the coin space, since we usually want to start the state in a "balanced" state, where interference doesn't bias our distribution towards only one side:

$$|i\rangle = \frac{|0\rangle + i|1\rangle}{\sqrt{2}} \tag{3.33}$$

It can be prepared by applying an Hadamard gate, H , and a Phase gate of $\pi/2$ (usually called S gate):

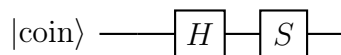


Figure 3.10: Initial coin state preparation circuit.

Our final quantum circuit should have the shape, as seen in figure 3.11:

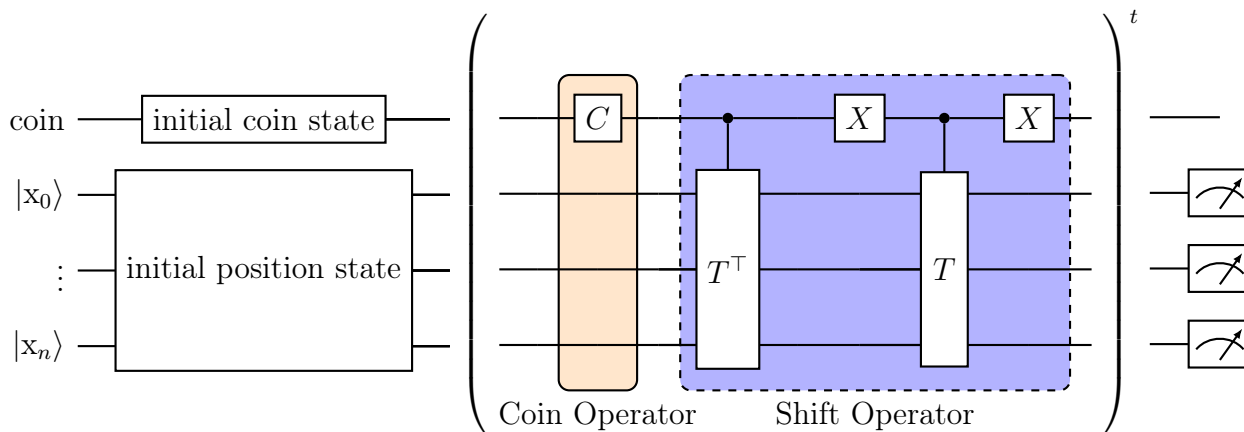


Figure 3.11: Circuit preparation for the generic cycle quantum walk.

Sadly, the simplification of the Shift gate, S , in fundamental gates reveals an extremely complex operation, evolving entangling multiple qubits which not only increases the depth of the circuit, but when simulated in real quantum hardware produces poor results due to the current low coherence times, i.e., performance quality drops with subsequent operations. In the other hand, for the general quantum walk's coin implementation is an one qubit gate and is easy to implement with sufficient high fidelity.

Fundamentally, when describing the Shift gate from the matricial form to the available gates, this is composed by multi controlled NOT gates, called CNx or n-Toffoli gates:

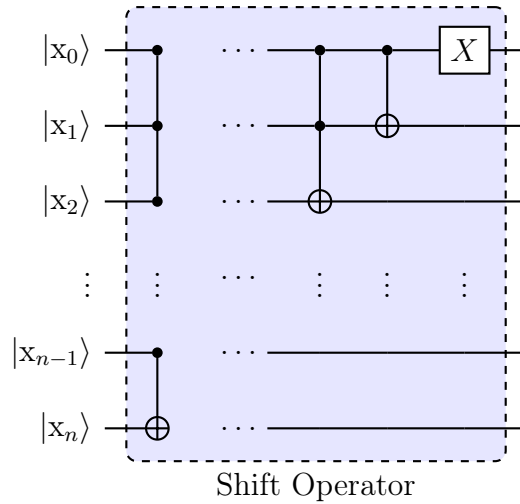


Figure 3.12: Circuit fraction for the S (shift) gate

Each CNx (for a minimum of three qubits) as $2n^2 - 6n + 5$ circuit complexity and $8n - 20$ of circuit depth, with n the number of qubits. For a S gate applied to a subspace of at least two position qubits and one coin qubit (three qubits in total), the circuit complexity follows the expression [67]:

$$\left[\sum_{i=3}^n 2n^2 - 6n + 5 \right] + 2 \tag{3.34a}$$

Where the plus two is related to the added complexity of CNOT gate and NOT gate needed for the development of the gate. Since this is an arithmetic progression, it can be simplified to the following equation

$$= n(2n^2 - 6n + 7)/3 \tag{3.34b}$$

The same can be done for the circuit depth:

$$\left[\sum_{i=3}^n 8n - 20 \right] + 2 \tag{3.35a}$$

$$= 2(2n^2 - 8n + 9) \tag{3.35b}$$

This leads to a polynomial increase with the number of qubits n when using the architecture, restraining the efficiency of such algorithm in current quantum hardware. Therefore we propose the use of an simplified architecture (3.2.1) whose complexity and depth increase linearly with the number of qubits n , whose gains can be seen in the graph 3.13:

$$2(n - 1) \tag{3.36}$$

We will also explore the possibility of implementing an hybrid architecture to maximize the usage of real quantum hardware.

This topic becomes even more important when we want to implement our conditional coins 3.1.2.

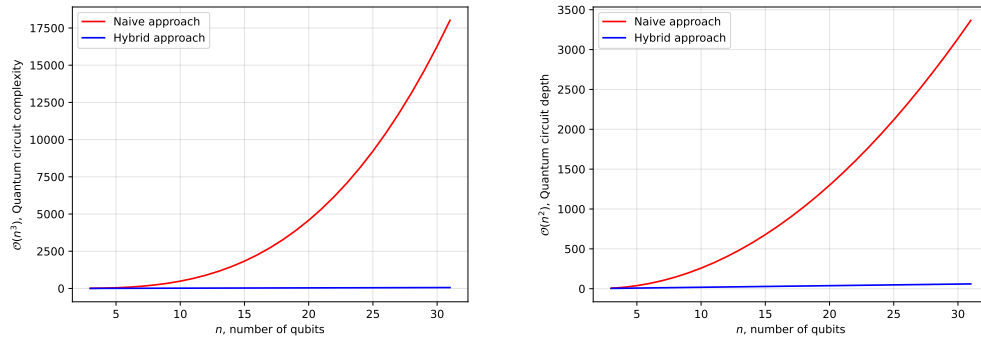


Figure 3.13: In the left side we have a graphical representation of the Quantum circuit complexity with the number of qubits n . In the right side we have a graphical representation of the Quantum circuit depth with the number of qubits n .

3.2.1 Simplified and hybrid quantum walks

As we have seen, the efficiency of the walker’s implementation depends mainly on the shift operator, S , which is a multi-qubit-controlled gate since the coin operator, being an one-qubit gate, is easy to implement in practice, with sufficiently high fidelity. The presented architecture that uses the T operator (eq. 3.29), without any type of simplification, relies on multi-qubit gates, namely $C_N X$ gates, which require excessive computational power to enhance spacial resolution. This is because achieving greater spatial resolution requires an increase in the accessible space, which can be accomplished by utilizing more qubits and a greater number of steps or circuit depth to cover more distance. To address this issue, it is crucial to implement a more efficient solution to simulate quantum walks.

One promising and effective approach is to work in the Fourier space. The Fourier transform acts on the spacial part of the computational basis \mathcal{H}_P as follows. A recent work has proposed working with Quantum Fourier Transforms [67] which deeply simplifies the quantum circuit.

The Fourier Discrete Transform (FT) diagonalizes the shift operator S [67]. In fact, FT diagonalizes any unitary circulant matrix, i.e., any convolution matrix [68].

Lets assume V and V' as the new simplified formats of the right and left shift, respectively:

$$V = T \tag{3.37a}$$

$$V' = T^\top \tag{3.37b}$$

The Fourier transform diagonalizes our (right) shift operator:

$$V = \mathcal{F}\Omega\mathcal{F}^{-1} \tag{3.38}$$

where

$$\mathcal{F}_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}kn}, \quad k = 0, 1, \dots, N - 1$$

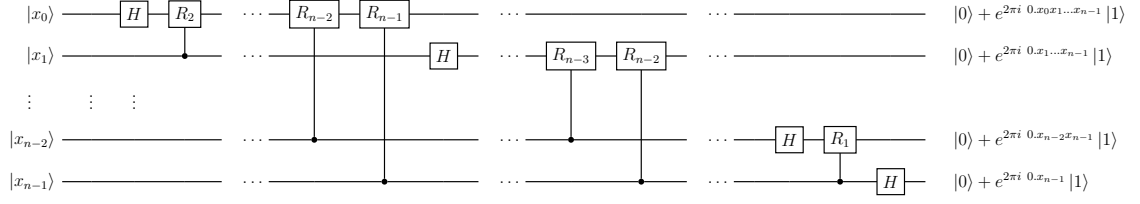


Figure 3.14: Circuit Schematic for the Quantum Fourier Transform

Which produces a diagonal matrix Ω :

$$\Omega = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & \omega & \ddots & \ddots & \vdots \\ 0 & 0 & \omega^2 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & \omega^{N-1} \end{bmatrix} \quad (3.39)$$

This can be simplified in a tensor product that can be seen as controlled phase changes in each individual qubit:

$$\Omega = \begin{bmatrix} 1 & 0 \\ 0 & \omega \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & \omega^2 \end{bmatrix} \otimes \cdots \otimes \begin{bmatrix} 1 & 0 \\ 0 & \omega^{N-1} \end{bmatrix} \quad (3.40)$$

where $\omega = e^{2\pi i/2^n}$. Is important to note that in order to have the output of the results ordered correctly due to the effects of the Fourier Transform we can simply express Ω as:

$$\Omega = R_n \otimes R_{n-1} \otimes R_{n-2} \otimes \cdots \otimes R_1 \quad (3.41)$$

where:

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix} \quad (3.42)$$

For the left shift:

$$\bar{\Omega} = \bar{R}_n \otimes \bar{R}_{n-1} \otimes \bar{R}_{n-2} \otimes \cdots \otimes \bar{R}_1 \quad (3.43)$$

where:

$$\bar{R}_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{-2\pi i/2^k} \end{bmatrix} \quad (3.44)$$

For the shift to be coin dependent the matrices should be displayed in their controlled form:

$$C(R_k) = \begin{bmatrix} \mathbb{1}_2 & 0 \\ 0 & R_k \end{bmatrix} \quad (3.45a)$$

$$C(\bar{R}_k) = \begin{bmatrix} \mathbb{1}_2 & 0 \\ 0 & \bar{R}_k \end{bmatrix} \quad (3.45b)$$

Since for the right shift, we need to invert the coin state, $|0\rangle$ to $|1\rangle$ so the control qubits can target the gate, we need to surround the control element by X gates:

$$C^{(1)} = (X \otimes I_2)C^{(0)}(R_j)(X \otimes I_2) \tag{3.46}$$

We finally can display the controlled rotation as:

$$C(\Omega) = \prod_{k=1}^n C(R_k) \tag{3.47a}$$

$$C(\bar{\Omega}) = \prod_{k=1}^n C(\bar{R}_k) \tag{3.47b}$$

The simplified shift operator V , should be:

$$V = (\mathbb{1}_2 \otimes \mathcal{F}^\dagger)C(R_k, \bar{R}_k)(\mathbb{1}_2 \otimes \mathcal{F}) \tag{3.48}$$

The circuit for the shift operator, V , gains the following form:

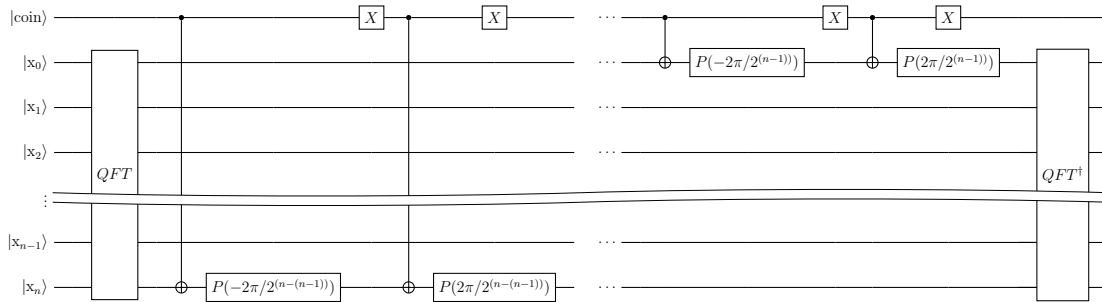


Figure 3.15: Shift operator circuit (V) using Fourier space simplification.

And the full unitary evolution is:

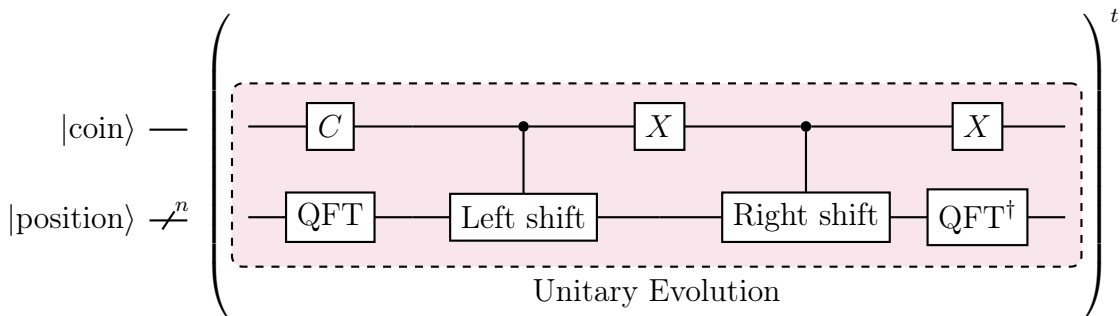


Figure 3.16: Graph for the Unitary Evolution Operator

An example for a quantum walk is shown on figure 3.17 using 10 qubits.

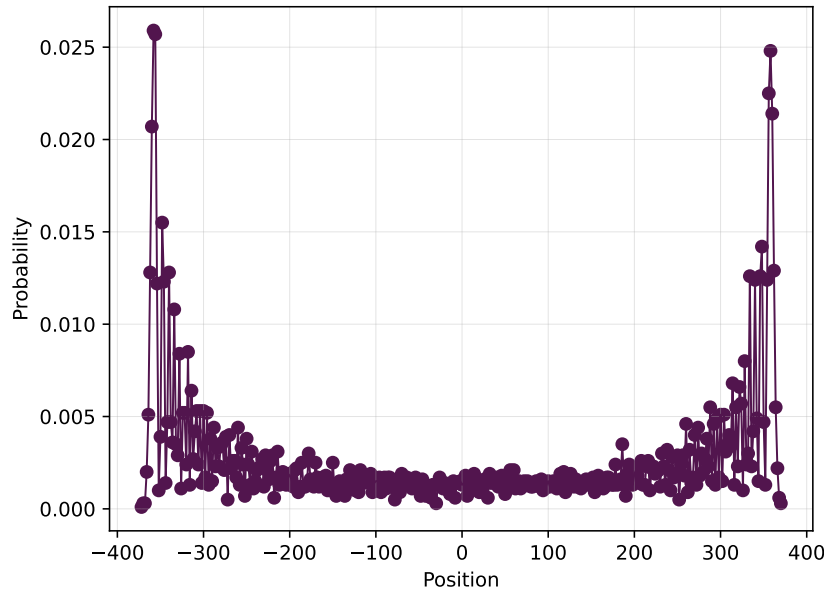


Figure 3.17: Quantum Walk Simulation, using 10 qubits for position resolution (2^{10}). The simulation was done using `qasm_simulator`.

In this work we went even further, is possible to assume that the position space is on the Fourier space, i.e. we drop the quantum Fourier transformation at the beginning and at end of each shift and only apply them at the beginning and at end of the algorithm. When this is possible to apply, we significantly reduce the number of circuit depth, since we do not need two QFTs for every single shift operation, only two QFTs for all the shift operations.

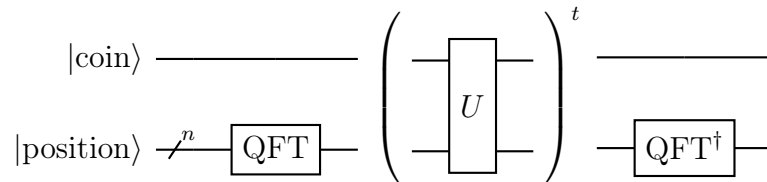


Figure 3.18: Quantum walk circuit in the Fourier space, with the basis transformation at the beginning and at the end of the unitary evolution.

The interesting part is for our simple case, (i.e. preparing the particle in a central position of the line grid) we can create the initial state and apply the Classical Fourier transformation to this state, all in a classical computer [69]! This allows for a decreasing (slightly!) of the depth and complexity of the circuit. More specifically we can reduce the circuit by 2^n (QFT depth) with the downside of the increased complexity, since instead of one rotation in one qubits we need to apply n rotation for all n qubits. The depth of the initial state preparation remains the same.

In the next chapter 4 we will delve a little bit in the possible implications of this three different architectures for the coined quantum walk.

Algorithm 2 Hybrid (1D+1) quantum walk

-
- 1: Define *position_state* as an integer lattice of n nodes in the fourier space
 - 2: Define *coin_state* as a two-level quantum system (qubit)
 - 3: Define *coin_operation* as a 2×2 quantum operator
 - 4: Initialize *walk_state* using *position_state* and *coin_state*
 - 5: **for** each time-step **do**
 - 6: Apply the *coin_operation* by some SU(2) unitary operation on the *coin_state* of *walk_state*
 - 7: Apply the shift operator on the *walk_state* which conditionally shifts the *position_state* depending upon the *coin_state*
 - 8: **end for**
 - 9: Apply QFT[†]
 - 10: Measure the state of the system to obtain the current *walk_state*
-

3.2.2 Quantum walk with position-dependent coins

The implementation of the position-dependent coin in a quantum walk, unfortunately, limits the use of some of the techniques we looked at. In a controlled gate, the control qubits, to correctly perform as intended, i.e., activate the target gate, must be in the computational basis $\{|0\rangle, |1\rangle\}$. If we implement the hybrid quantum walk architecture in this case, is impossible to have an ordinary control of the target gates due to the different computational base. Thus, we can only implement the simplified shift operator 3.48.

Also, given the exponential increase of coins due to the need for one coin for every possible position, regular personal hardware is not capable of developing highly detailed graphs, which lead us to dedicate a big part of this work on exploring the benefits of using high performance computing to simulate quantum hardware, aspect that is reserved to the next chapter 4.

Redirecting our focus to the aspects of the position-dependent coin, we want to implement a square potential well, where the barriers are *perfectly* repulsive. For this to happen in practice, in our quantum algorithm, it's crucial to tailor our position-dependent coins to behave like this:

$$C^{(n)} = \begin{cases} \text{Repulsion, if } n = \text{barrier position} \\ \text{Standard coin (ex. Hadamard, Dirac Particle), otherwise} \end{cases} \quad (3.49)$$

In practice the algorithm should work as follows:

Algorithm 3 (1D+1) Quantum Walk in potential square well

```

1: Define position_state as an integer lattice of  $n$  qubits
2: Define coin_state as a two-level quantum system (one qubit)
3: Define barrier_position_left and barrier_position_right in the available position space
4: Define main_coin_operation as a  $2 \times 2$  quantum operator, with  $n$  control qubits

5: Define coin_barrier_left and coin_barrier_right as a  $2 \times 2$  quantum controlled operator, with  $n$  control qubits
6: Initialize walk_state using position_state and coin_state
7: for each time-step do
8:   for each position do
9:     if position = barrier_position_left then
10:      Apply the coin_barrier_left on the coin_state of walk_state
11:     else if position = barrier_position_right then
12:      Apply the coin_barrier_right on the coin_state of walk_state
13:     else
14:      Apply the coin_operation on the coin_state of walk_state
15:     end if
16:   end for
17:   Apply the shift operator on the walk_state which conditionally shifts the position_state depending upon the coin_state
18: end for
19: Measure the state of the system to obtain the current walk_state

```

To display the correct coin behaviour in each of the two limits of the position space, when tried to reach the left limit, the coin needs to force a right shift and vice versa. This implies that the last coin state, before reaching the barrier was technically $|1\rangle$ for the left side and $|0\rangle$ for the right side. Therefore we want to flip that state and lead the coin to be in the left limit in $|0\rangle$ and in the right limit in the $|1\rangle$. To exhibit this behaviour using the circuit model, it can be accomplished by setting *coin_barrier_left* and *coin_barrier_right* as C_nX gates.

3.2.3 Dirac free-particle trapped in a square potential well

One of the objectives of this work is to integrate the various distinctive characteristics of the Quantum Walk algorithm and combine them to develop something interesting and with scientific importance.

Such product is the simulation of a Dirac free-particle trapped inside a square potential well. This perfectly combines one of the most attractive applications of the quantum walks algorithms, the relationship between quantum walks and Dirac free-particle equation, with the position-dependent coin.

The algorithm implementation follows a similar architecture to the one depicted in section 3.2.2. The most important and notorious change is the need to develop a coin specific to model the Dirac free-particle behaviour, while we maintain the

"barrier" coins in the ends of our lattice, modelled by the $C_n X$ gates. The "main" coin will be the $R Y(\theta)$ with a $\theta = 2$:

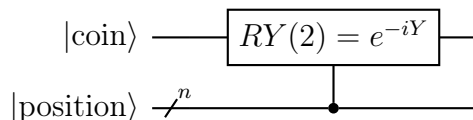


Figure 3.19: Quantum Circuit with schematic of the Dirac free-particle coin.

with $R Y(\theta)$ given by:

$$R Y(\theta) = e^{-i\frac{\theta}{2}Y} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & \sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad (3.50)$$

3.2.4 Visualizing time evolution

Until now, we displayed the data representation of the 1D coined quantum walk as a single circuit execution, where its result product is in regard to the particle state after a certain amount to time steps given by the particle. Although already quite interesting, would be far more practical to observe all the consecutive time steps, or in better words, the time evolution of the particle's state. Following the 3rd quantum mechanics postulate A, after a measurement of $|\Psi\rangle$, it collapses into one of its eigenstate, i. e., we end up with a different initial state. This is to say that we cannot simply consecutively gather information from the quantum walk by adding measurement operators after each time step, since that would fundamentally disrupt the evolution of the quantum walk. Each measurement would collapse the state of the system and change the initial conditions for the next time step, making it impossible to continuously gather information about the initial quantum walk without interfering with its progression.

To avoid this problem and effectively visualize the evolution of a quantum walk with l steps, we will need to create l different quantum walk circuits. Each individual circuit in this series will represent a quantum walk incrementally increasing in the number of steps, commencing with a single step in the first circuit and culminating with an l -step quantum walk in the final circuit:

Algorithm 4 Incremental Evolution Visualization in (1D+1) Quantum Walk

- 1: Define *position_state* as an integer lattice of n nodes
- 2: Define number of steps l
- 3: Define *coin_state* as a two-level quantum system (qubit)
- 4: Define *coin_operation* as a 2×2 quantum operator
- 5: Initialize *walk_state* using *position_state* and *coin_state*
- 6: **for** *step* in range from 0 to l **do**
- 7: **for** each time-*step* **do**
- 8: Generate Circuit
- 9: Apply the *coin_operation* using some SU(2) unitary operation on the *coin_state* of *walk_state*
- 10: Apply the shift operator on the *walk_state* which conditionally shifts the *position_state* depending on the *coin_state*
- 11: **end for**
- 12: Measure the state of the system to obtain the current *walk_state*
- 13: **end for**
- 14: Save results in (l, n) array

Following is a graph schematic of how we intend to display part of the data:

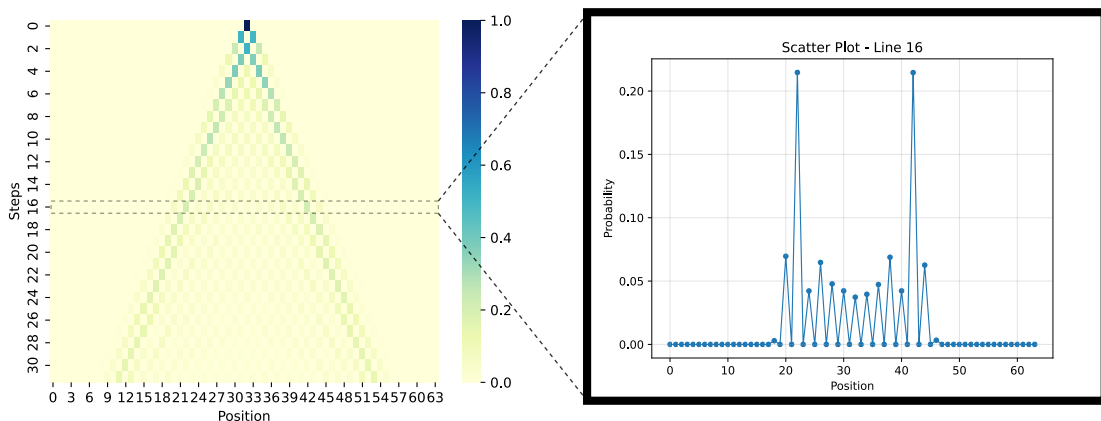


Figure 3.20: Schematic visually showing how the data in the heatmap plot refers to the individual histograms.

3.2.5 Qiskit implementation

For this work, the framework used to work on the quantum information and computing domain is the Qiskit framework. Established by IBM as an open-source software development kit it is based on Python programming language and is intended to work with the quantum circuit model, providing a broad array of core components for the creation, processing, simulation and visualization on quantum computing tasks. The component Qiskit Aer will be of high interest in the next chapter, because it provides high-performance quantum computing simulators capable of running in CPUs or GPU and multiple options to tune and maximize the simu-

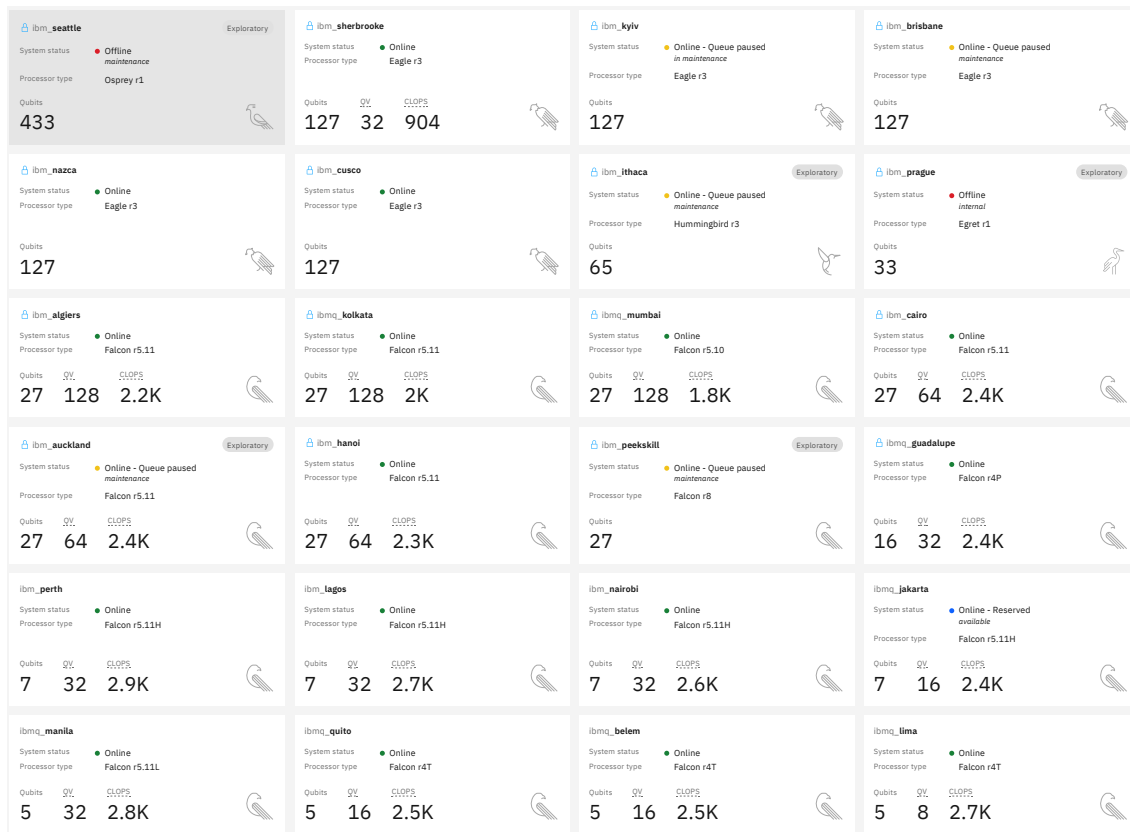


Figure 3.21: Array of all the available, on cloud, IBMQ quantum hardware devices. This list has layed out the number of qubits, quantum volume (QV) and Circuit Layer Operations Per Second (CLOPS) which refer to the performance metrics of the hardware.

lation efficiency. It also allows to easily run quantum algorithms in real quantum hardware, but with a lot of limited resources and availability.

Moreover, IBM has invested heavily in educational components as well as providing extensive documentation for its framework. Unfortunately this produces tones of available documents, whereas some parts of this vast repository are repeated or even deprecated, which introduces some expected challenges, something we experienced in the course of this work. The highly vibrant community to help on debugging our code and also some existing experience on using this framework add to the reasons to work with this tool.

Following, we will explaining on how the algorithm is implemented into this framework:

In Qiskit there are about 3 main stages to correctly implement a given quantum circuit 3.22. First we generate the quantum circuit(s) in respect to the desired task or algorithm. It allows to create our unique operators and easily initialize quantum states. Then we need to choose the type of backend to use. Backend represents either a simulator or a real quantum computer and are responsible for running quantum circuits, running pulse schedules, and returning results. The last main stage is responsible for compiling and running our tasks. In this stage first comes the transpiler process which involves converting the operations in the circuit



Figure 3.22: Qiskit's task flow chart evolution.

to those supported by the device and/or to optimize the circuit. This parameter can be highly tuned so it can be adapted to specific demands. Also in real quantum hardware, not all the qubits have direct connections, needed to performance controlled actions, therefore this process is capable of reorganizing qubits to overcome this connectivity limit. The transpile process is followed by the run process where it executes the quantum circuit(s). Each one runs a `shots` amount of times to obtain the desired probability distribution of results. Lastly, the result processing operation reformulates the data outputted by the execution process into the form of a dictionary. Given the presumption that the execution process yields a 'job' Python object, invoking `job.result().get_counts()` is expected to provide a Python dictionary. This dictionary organizes the count of occurrences for each quantum state. It's important to note that each state is a single observed outcome from one execution of the quantum circuit. The repeated execution of the Quantum circuit, regulated by the `shots` variable, enables us to construct a probability distribution. This distribution is critical for comprehending the potential final states of our quantum system.

Also, if we need to simulate multiple circuits, it is possible and also recommended since it allows for efficiency improvements.

The error between experiments, i.e., not between each `shots`, but between two separated executions with the same number of `shots` for the same quantum circuit is usually given by the Hellinger Fidelity which equivalent to the standard classical fidelity:

$$F(Q, P) = \left(\sum_i \sqrt{p_i q_i} \right) \quad (3.51)$$

This equation comes handy when comparing the runs in different real quantum backends or even between a classical quantum simulator and a real quantum backend. When comparing the results from different runs ran in the same classical simulator, the fidelity even for just 1000 `shots` is extremely high, around 99.98%, which is expected since our probability distributions cannot always be exactly the same.

Chapter 4

Algorithm Benchmarking

The ultimate goal is to understand how far-reaching our algorithm can be. Understanding the current limits from the publicly available NISQ quantum computers and classical quantum hardware simulators is important to perceive what can be done, i.e., what is capable of being implemented on useful applications using new, hybrid, and non-hybrid quantum algorithms. Unfortunately, at the time of test, the public available hardware is still too limited in resources and accessibility. In the other hand, the software was successfully implemented showing a clear path to future iterations.

In this chapter, we began with section 4.1, where we explore the available features and resources from NISQ quantum hardware, available via the IBMQ cloud service. In addition to showcasing the obtained results, we also discuss the current state of available hardware and how it impacts the execution of quantum algorithms. This also serves to display the importance of hybrid quantum algorithms, as a bridge to the full quantum hardware and how noise effects completely jeopardize the result.

Furthermore, we dedicate the next section 4.2 to adapt our algorithm to the high performance computing (HPC) by executing our algorithm using a quantum computer simulator. Such an approach not only helped on bringing to light the importance of running code on classical hardware, since it gives a greater perspective on how algorithms can be developed to use some of its unique capabilities. We tried to maximize efficiency so, in the same time frame, we could execute the quantum walk algorithm, for a problem specific case, with more spacial resolution and step count.

We end this chapter by demonstrating the substantial impact of executing HPC algorithms at high resolutions with a high step count. Using the quantum walk algorithm framework, we were capable of simulating the behaviour of a Dirac free-particle trapped inside a potential square well. This produces highly complex and depth quantum circuits, executions that could only be accomplished, without noise errors, on a HPC.

4.1 Quantum walk algorithm advantages

As we seen through out section 3.2, we discussed some architectures to run our quantum walk algorithm. Here we want to analyse in more detail the implications these have when ran in different computing hardware: real quantum hardware (in a hybrid approach) and classical quantum simulator.

First, in this test we tackled the architectures on section 3.2.1, applying the quantum Fourier transform on a real quantum computer (i.e. using the Fourier transform) vs applying it on a classical computer. We performed tests using the IBMQ cloud computers: `'ibmq_qasm_simulator'`, classical quantum computer simulator (represent a fault free, environment intolerant, quantum computer) and on `'ibmq_perth'`. This backend has available 7 qubits, connected in an H shape as we can see in the figure 4.1. On Appendix C are available all the details regarding this quantum backend.

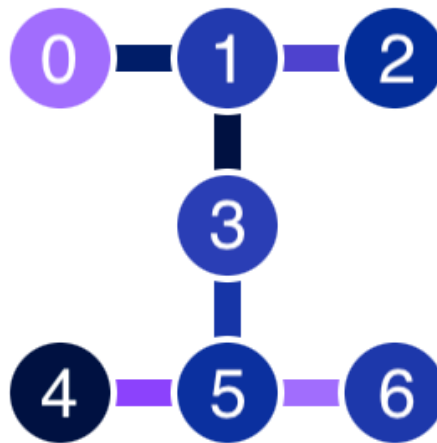


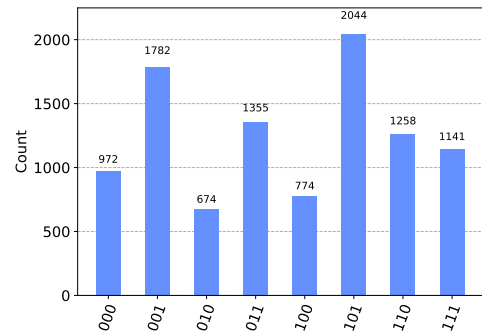
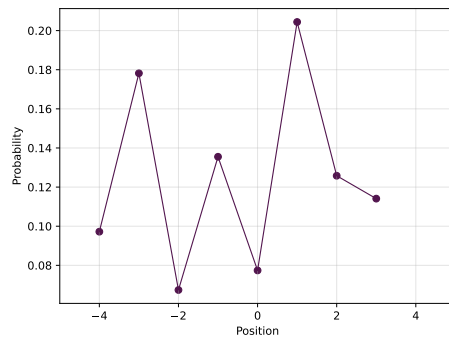
Figure 4.1: `ibmq_perth` qubit layout. Its architecture can be visualized through different tonalities or shades across its layout. Herein, unique shades correspond to distinct $T1$ times, while variations in connection tonalities represent different extents of CNOT errors. A gradient effect is applied to these tonalities, transitioning from dark to light. It's important to note that this gradation symbolizes a change from lower to higher values, respectively. All the details for this quantum backend can be found on Appendix C.

The algorithm will use 3 qubits and perform 2 steps (we will see that even this low number will display an exaggerated uncertainty in our results). We performed 10000 shots for each circuit. The initial position will be in the middle of the lattice $|100\rangle$ and the coin will be an Hadamard coin starting in a "balanced" position.

In this experiment, `'ibmq_qasm_simulator'` displays the desired result, the one we want the algorithms ran on quantum computers to converge to. This result comes in the shape of a probabilistic distribution and the disparity between these results is given by the Hellinger Fidelity (eq. 3.51).

First, the results from running the code on `'ibmq_qasm_simulator'` can be found on figure 4.2. Here we only made use of the Hybrid architecture since it produces the same result:

non-Hybrid



Hybrid

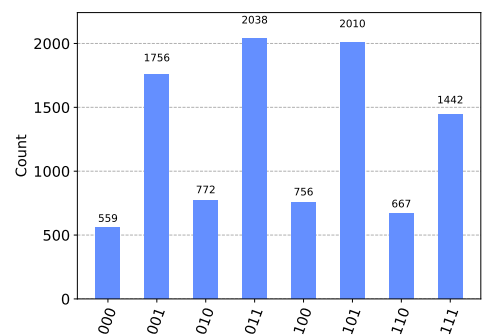
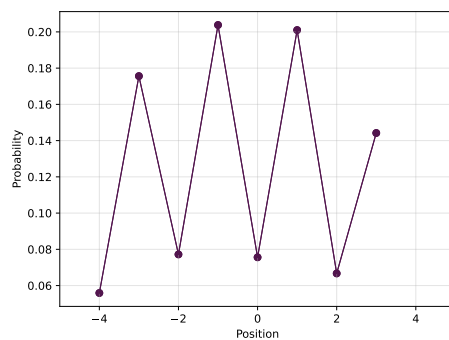


Figure 4.3: Graphs from the results ran on 'ibm_perth'. On the top row the results relate to the non-hybrid architecture and the bottom ones to the hybrid architecture.

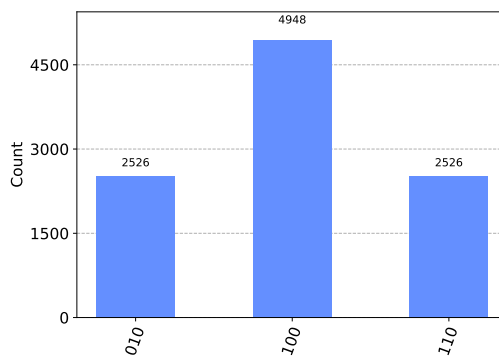
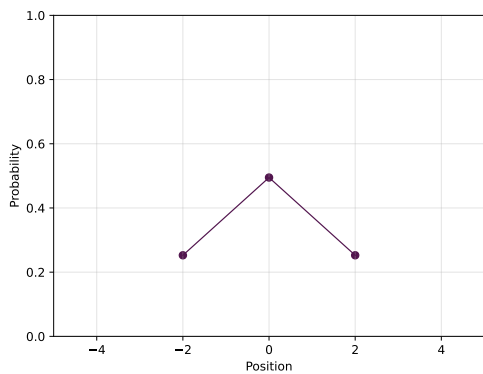


Figure 4.2: Graphs from the results ran on 'ibmq_qasm_simulator' for 2 step quantum walk. On the left is the probability distribution scatter ad on the right the counts histogram.

Next the results from running the code on 'ibm_perth' using the non-hybrid and hybrid architectures, can be found on figure 4.3.

To better visualize the difference between the ideal result ('ibmq_qasm_simulator') and the result from real quantum hardware ('ibm_perth') we built the following graph (figure 4.4).

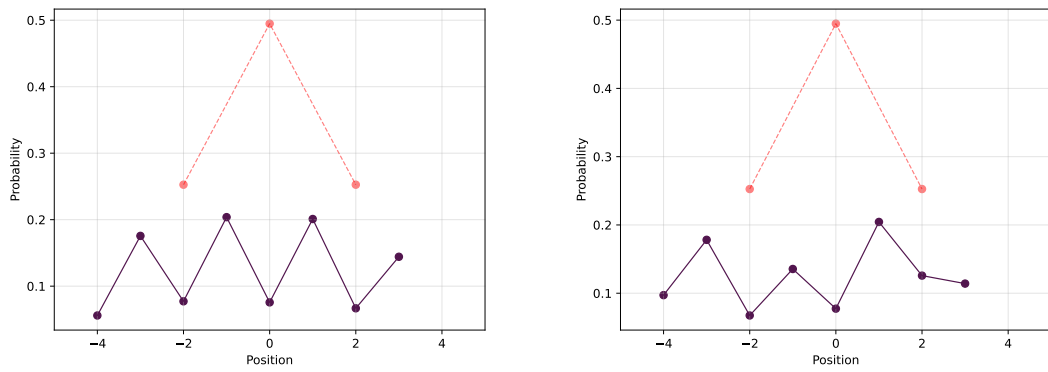


Figure 4.4: Plot with the overlap obtain from the 2 step quantum walk simulation on the ideal quantum computer ('ibmq_qasm_simulator') in red (or dashed) with the ones obtained from 'ibmq_perth', purple (or solid line). On the left is the graph for the non-hybrid architecture and on the right for the hybrid architecture.

The Hellinger Fidelity (eq. 3.51) was, using as term of comparison the result obtained by the ideal quantum computer, *ibmq_qasm_simulator*, for the hybrid architecture 0.254459 and for the non-hybrid architecture 0.214234.

This process was repeated for 1 step quantum walk to see if we could obtain a better result. First the ideal results, as on figure 4.5:

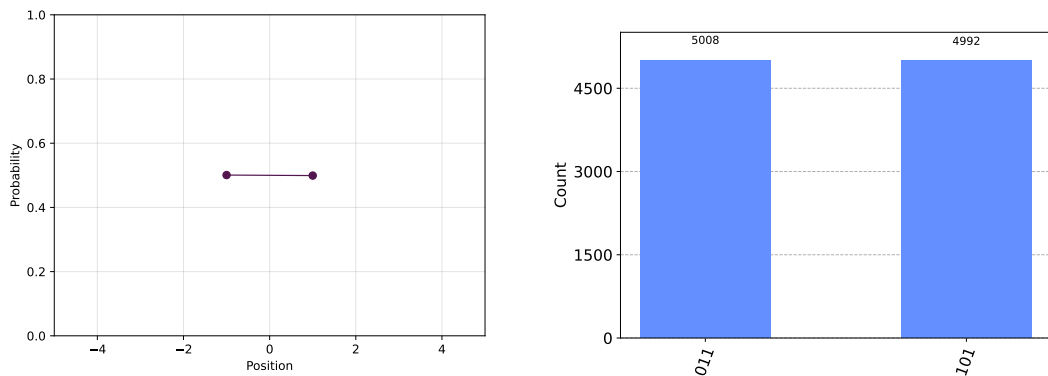


Figure 4.5: Graphs from the results ran on 'ibmq_qasm_simulator' for 1 step quantum walk. On the left is the probability distribution scatter ad on the right the counts histogram.

And the overlap with the obtain results on real quantum hardware, 'ibm_perth':

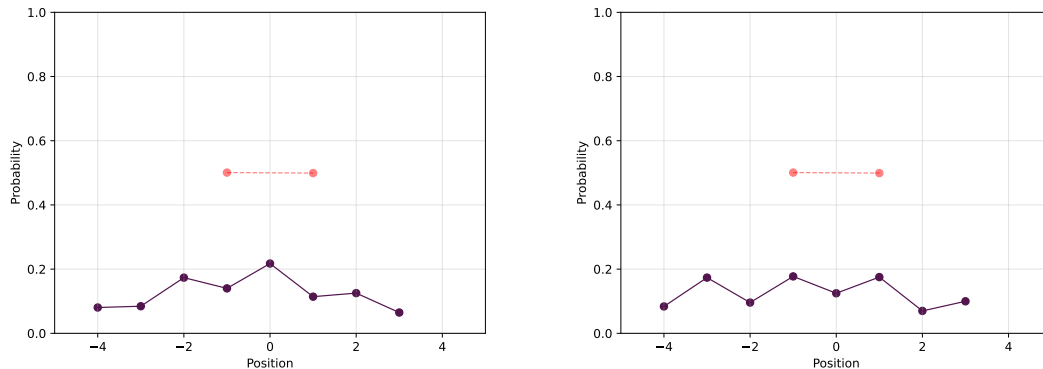


Figure 4.6: Plot with the overlap obtain from the 1 step quantum walk simulation on the ideal quantum computer ('ibmq_qasm_simulator') in red (or dashed) with the ones obtained from 'ibm_perth', purple (or solid line). On the left is the graph for the non-hybrid architecture and on the right for the hybrid architecture.

For the hybrid architecture we obtained a fidelity of 0.352498 and for the non-hybrid architecture 0.253564. This result is quite more interesting, showing a more significant difference between these two architectures.

Unfortunately, is difficult to assume that the hybrid architecture is straight out better. Each circuit execution in a quantum computer is always different due to the ever changing environment conditions. Ideally, to perform this tests with much more accuracy to extract substantial statistical data, we would need multiple repeated circuit executions. As of today, we could not submit jobs with multiple circuits in IBMQ Provider component, which in theory is possible, but in its absence implies to create separate submissions to each circuit. This leads to a difficult situation where the amount of data generated, that cannot be automatically sorted, would lead to a tedious task, to have marginally better results.

Not just only from the visual data, but from the statistical data we can easily understand the current hardware limitations to perform any kind of quantum walk. At this stage is difficult to suggest any type of modification to enhance results. Theoretically we can separate all the qubit computations by multiple backends, when using the hybrid architecture. In the other hand, this is more difficult in practice. Real-quantum hardware is still limited with the few available backends with characteristics that point to our needs coupled with long waiting times (just to give an example one of the executions stood in queue to be executed by three days). To add to this, separating our qubits evolution would expose different environments to our qubits what would create inconsistencies and random errors.

4.2 Algorithm execution on classical hardware

4.2.1 Classical cluster simulation methodology

As we saw in the previous section (4.1) the execution of the algorithm in real quantum hardware is far from a viable solution. The generated quantum circuits to output a sufficient high resolution image for the quantum walks algorithm are too complex. Therefore we focus now on executing the code on a quantum classical simulator. It is important to clarify that a classical simulator and hybrid algorithms relate to different aspects.

Classical simulators of quantum hardware have ever since been an important component in the quantum computing realm. They not only help on the development of quantum algorithms by being a more accessible mean to debug and allow for a proof of concept for the algorithm, with testing and tuning in a fast and reliable way, without changing the underlying architecture of the quantum algorithm.

Running our algorithm in a classical quantum simulator is quite different from simply sending the quantum circuits for a cloud-available quantum computer. The goal is to maximize the capabilities of the available hardware by maximizing the efficiency at which the results of the quantum walk algorithm are produced.

To achieve this, we will use the resources provided by the Laboratory of Advanced Computing at the University of Coimbra. Running our code on a HPC allows for more higher memory, higher number of threads and large number of nodes when compared with a typical personal computer.

4.2.2 Laboratory for Advanced Computing - Navigator+

Navigator+ is the latest deployment of the HPC infrastructure of the Laboratory for Advanced Computing (in Portuguese *Laboratório para Computação Avançada*) of the University of Coimbra. Following is a table of the available partitions and respective specifications:

Table 4.1: List of the used partitions and specifications for benchmark testing and simulation.[70]

Partition	Name Prefix	# Nodes	MEM	# vCPUs	Local Disk
cpu1	barinel	158	96GB	48	0
cpu2	caravela	13	96GB	80	239GB
hmem1	nau	7	384GB	80	239GB
hmem2	caraca	1	3TB	144	3.8TB
gpu	galeao	4	96GB	80	239GB

The Navigator+ supports many of most important and used compilers, tools and message passing interface libraries. A complete list can be accessed in the reference: [70].

4.2.3 Simulator architecture

For the simulation of our algorithm we will be using the Qiskit Aer component, more specifically the Aer-Simulator [57] v0.12.1 framework, a high-performance quantum computing simulator, supporting many of the available technologies on the Navigator+ to improve performance. Given its easy accessibility and functionality it is contrasted by an absence of comprehensive community testing. To ensure we took the maximum capabilities of the Aer we did a series of validation and benchmark tests.

The Aer-Simulator offers a selection of simulation methods, in other words, classical quantum computing simulators tailored to specific simulation needs. For this work the following was used:

- **statevector** - statevector simulator that can sample measurement outcomes from *ideal* circuits with all measurements at end of the circuit. Is also capable of introducing noise to obtain results similar to the ones on current, state of the art, quantum computing hardware. Since we are studying our quantum state evolution (expressed by the position of the particle in the mesh), this is the indicated simulator.

It is worth mentioning that throughout this work, we made use of other simulators such as `qasm_simulator`, `aer_simulator` and `aer_statevector_simulator`. However due to the ever-evolving nature of the framework's documentation, these simulators have been deprecated or dropped in favour for the newer implementations.

In addition, the Aer-Simulator offers developers a range of advanced options to optimize the software's performance. This allows for tailoring the simulator software to the algorithm and classical hardware needs, however this results in hundreds of possible combinations, a time-consuming and occasionally frustrating task, particularly considering long wait times in the queue and during simulations. On top of that, given the novelty of qiskit-aer, we found a bug on the code, helping the community on developing the framework, detail we will see latter.

Each set of options has effects in the Memory and CPU use on the Navigator+, being the goal to maximize the efficiency of both. The set of advanced options tested are the following:

- **max_job_size** - divides the array of quantum circuits into smaller sub processes, which should allow for better efficiency.
- **max_parallel_experiments** - number of maximum quantum circuits executed in parallel. This option is coupled with **max_parallel_threads**, responsible to set the number of available threads.
- **max_parallel_shots** - number of maximum shots to be executed in parallel up to the **max_parallel_threads**.

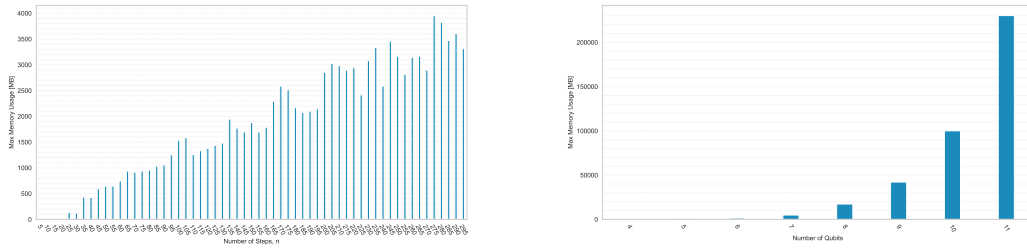


Figure 4.7: Plots for the algorithm memory usage using the statistics provided by the SLURM `sacct -format MaxRSS` command. On the left is displayed the memory usage for different number of steps with number of qubits fixed on 6. On the right is displayed the memory usage for different number of qubits, with number of steps fixed on 64.

- **precision:** Set the floating point precision for certain simulation methods to either "single" or "double" precision (default: "double").

Also, within the Qiskit framework, the `transpile()` [71] function is a crucial process to tune efficiency and extract the maximum performance of the classical cluster. In this work we tested the option `optimization_level`, which runs in levels from 0 to 3, with the first being where there is little to no optimization, and 3 to have heavy optimization. These levels have an effect on how the simulation reacts to the quantum circuit, therefore was also an important variable to test.

4.2.4 Simulation parameters tuning and validation

Memory Consumption

We begin this test by one of the cornerstones of qiskit-aer. Such limitation is the unexpected use of memory by the Aer Simulator. Only the state vector memory usage is clearly defined (n -qubits use 2^n complex values each occupying 16Bytes), when in reality the circuit depth also has a slightly well defined memory consumption even though not specified or correctly documented.

For our benchmark test we wanted to evaluate how by changing the number of steps and qubits could influence the memory consumption. By gaining an insight into the predicted behavior of memory consumption, in theory we could have a more predictable and consistent memory usage from our algorithm, thus allowing us to plan according the circuit execution in compliance to the available resources in the Navigator+.

In figure 4.7 are displayed the memory measurements for the different number of steps and qubits. As expected by the algorithm characteristics with the linear increase of number of steps, the increase in memory usage is also linear. However there is a slightly randomness in the maximum memory used. For the qubits, since the circuit depth increases exponentially with the number of qubits, the memory usage doubles with qubits increments.

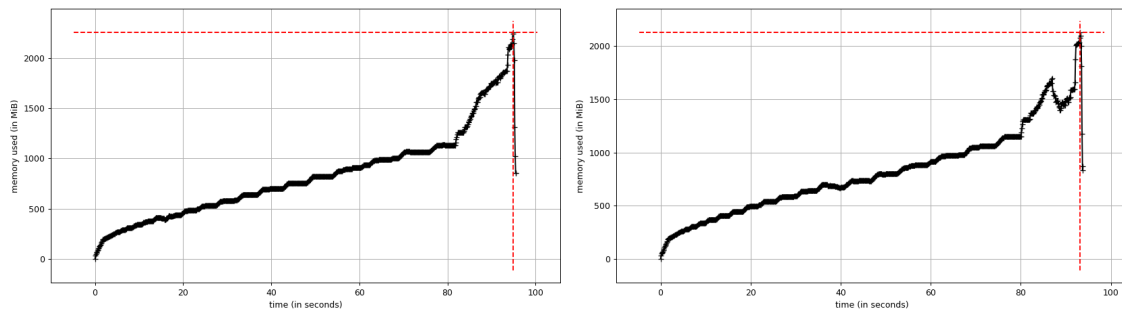


Figure 4.8: Memory profiler (using the `mprof` library for python) for a 6 qubits 64 steps algorithm simulation. From 0 to around 2 seconds the circuits are processed, then until the 82 seconds marks is the transpiler phase and the rest is dedicated to the run phase which is the one that should be affected by this parameter. On the left is the graph for the `double` precision and on the right for the `single` precision. Even though the graph curves on the running phase are not identical, both simulations reach the same peak of memory usage and take the same time to simulate.

The Aer-simulator also has the `precision` option which should reduce by half the memory usage when running the simulation and, in certain cases, enhance the efficiency. In our testes was not possible to see any benefits of running our code in the `'single'` option instead of the `'double'`, as is visually explained by the memory profiler plot in figure 4.8.

With this information we tried to design a sort of batching system. To maximize the memory usage of the simulator, we defined a pool, or batch, where the maximum size of it would be roughly limited by the depth of biggest circuit. The method can be seen in more detail on the schematic 4.9:

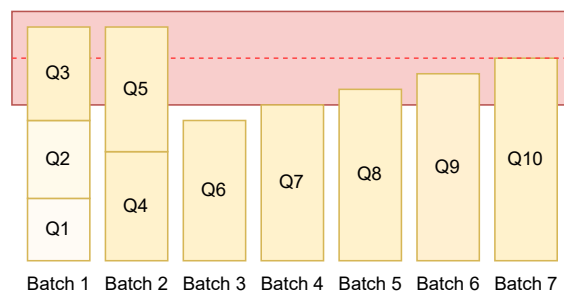


Figure 4.9: Schematic for the batching diagram. `Q[1-10]` relates to the all quantum circuits, distributed by the 7 available batches. The red block is the region until the max memory per pool can be.

The results from this test showed that this batching method required a higher memory usage, when compared to the default method. In the other hand, we obtained a small speed-up in terms of job execution time. The added complexity of the method and the invisible different of tuning the parameters `max_parallel_experiments` lead us to drop its usage.

Table 4.2: Results for the batching test.

Method	Job Wall-clock time	Memory Usage
Batched	01:46:53	9.51 GB
Default	02:07:00	2.17 GB

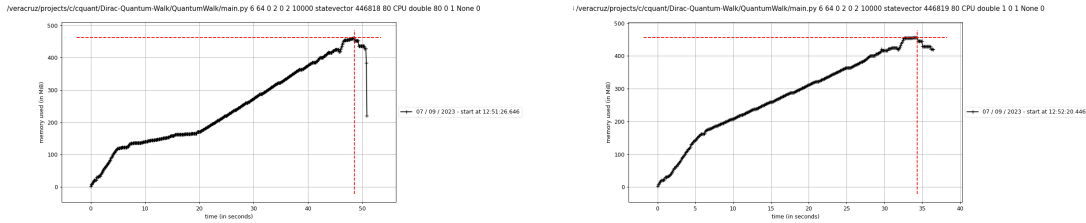


Figure 4.10: Memory profiler (using the `mprof` library for python) for the `max_parallel_experiments` (left) and `max_parallel_shots` (right) optioned for the maximum number of available threads (80). Unfortunately there are not noticeable changes in the memory curve between these two different settings. The running time as slightly better for the `max_parallel_shots=80` (2.8595 seconds versus 2.6087seconds, respectively).

CPU Time Usage Efficiency

A key principle of supercomputing is to enhance computational speeds through the parallelization of as many aspects of the running code as possible (as discussed in Section 2.3). The metric to evaluate the efficiency will be given by Slurm’s `seff` command, which indicates the number of threads used divided by those made available. It essentially provides a measure of how efficiently a job is using the allocated CPU resources. `max_parallel_experiments` and `max_parallel_shots` are the two options capable of implementing parallelization on our simulator, that can only be used individually, i.e., if `max_parallel_experiments > 1` then `max_parallel_shots = 1` and vice versa. However, during our experiments when tuning this parameters for different scenarios never outputted a noticeable change in performance and efficiency. When optioned `max_parallel_experiments` and `max_parallel_shots` to the maximum number of available threads (80), individually, the running times for running a Hadamard coin quantum walk of 6 qubits and 64 steps were just slightly different, (2.8595 seconds versus 2.6087 seconds, respectively). This lead us to creating an issue ticket in the Qiskit’s Github [72], which fortunately helped to unveil a bug in the source framework, being label as issue # 1880. At the time of submission of the work, the fix has not still been implemented in a qiskit release.

The experiment depicted in Figure 4.12 was designed to analyze the performance impact of the `max_parallel_experiments` setting in an environment saturated with a high number of circuits. We were particularly concerned that the intense circuit activity could undermine the efficacy of this setting. To deal with this, we implemented an exploratory approach wherein we divided the data into evenly sized batches, being these governed by the `job_size` parameter. Unfortunately, this did not result in any benefits, performance-wise or efficiency-wise.

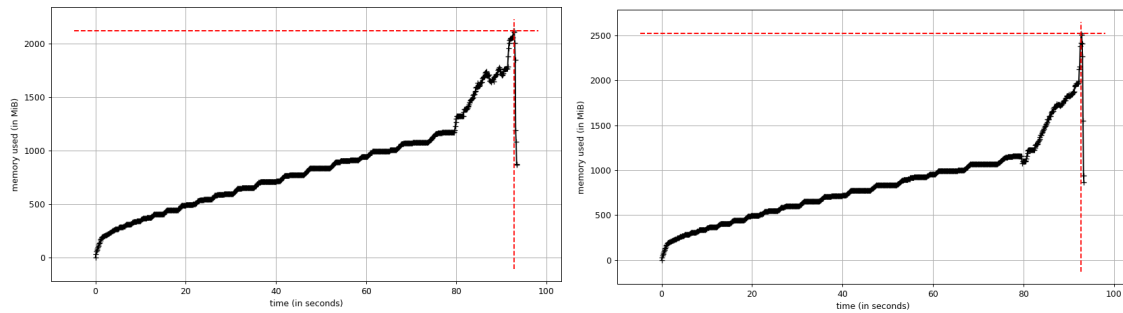


Figure 4.11: Memory profiler (using the `mprof` library for python) for a 6 qubits 64 steps algorithm simulation, comparing the values 5 and 10 for the setting `max_parallel_experiments`. On the left is for 5 parallel experiments and on the right for 10 parallel experiments. The curves are similar and these differences do not impose changes on the behaviour of the simulation.

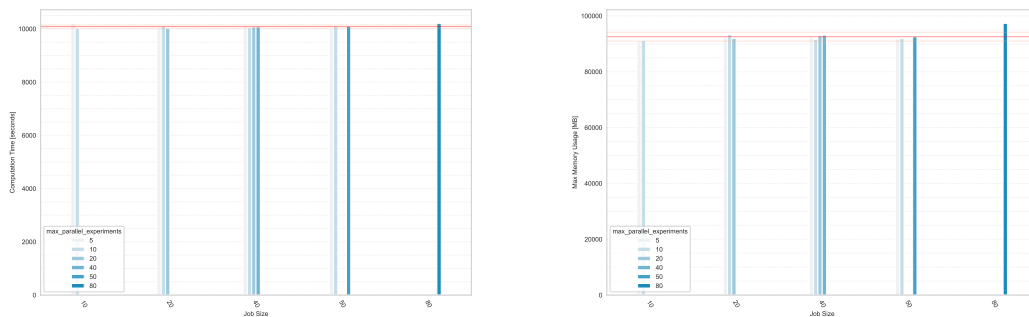


Figure 4.12: Plots for the algorithm time (left) and memory consumption (right) for different batch sizes using the statistics provided by the SLURM `sacct -format MaxRSS` command. For all the combinations created, all execution took the same time. On the left the average was 10087.8461 ± 59.4051 (0.5889%) [s] and on the right 92570.0 ± 1589.6051 (1.717%) [Mb].

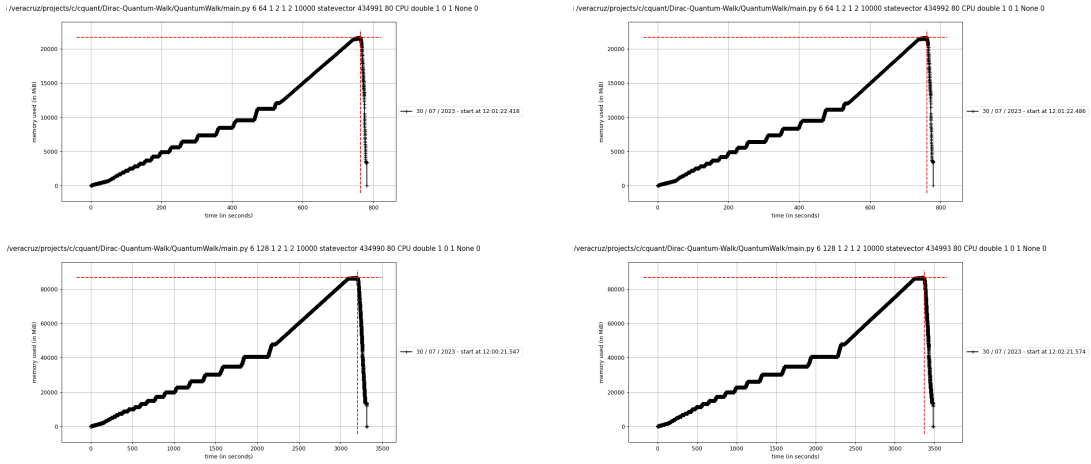


Figure 4.13: Memory profiler plot for a 6 qubits and 64 (top) and 128 steps (bottom) algorithm simulation, comparing the memory performance by changing the parameter *optimization_level*, = 0 (left) and = 1 (right). For 64 steps, the transpile times were similar being 363,74 seconds and 362,57 seconds and the execution times were also similar being 356,15 seconds and 355,59 seconds. For 128 steps, we got a shorter transpile times of 1670,056 seconds and 1761,844 seconds. Surprisingly it also got a shorter run time 1391,53 seconds versus 1462,07 seconds.

Transpiler Optimization

For the transpiler phase, the optimization parameter (*optimization_level* =) controls the amount of simplification of the circuit. Higher levels of optimization provide better run times of circuits but require more time to transpile, therefore is necessary to find a sweet spot to maximize performance. During our tests, for our algorithm, *optimization_level* = 2, 3 could not compile our circuit, which lead us to only test for values of optimization 0 and 1.

optimization_level = 0 only does the required transpile method for the circuit to run in the desired backend, *optimization_level* = 1 does light circuit simplification.

We tested for 6 qubits and for 64 and 128 steps. Detail results can be analysed in figure 4.13. Given these results, we opted to default the *optimization_level* = 0.

4.2.5 Results and analysis between analytical results

After an extensive range of tests, we were left with a distinct aftertaste, as none of the elements we experimented with, resulted in any significant benefits.

We were, therefore, faced with two options, or run the code as is, in the Aer-simulator, or *hard-code* some sort of parallelization on the available resources. To achieve this sort of "parallelization" we would distribute all the quantum circuits by the available partition nodes, drastically reducing the global execution time. We opted for the *hard-code* method, allowing for full control on the parallelization process.

The partition used was the *hmem1* due to the large available memory. We used all the available threads (80) to give access to the max memory (384Gb). It was ensured

that each job had its memory totally reserved to minimize possible errors within the cluster framework.

The options used were the following:

- `max_job_size = 1`
- `max_parallel_experiments = 1` `max_parallel_threads`, responsible to set the number of available threads.
- `max_parallel_shots = 80`
- `precision = double`

The selected problem to simulate on a quantum algorithm was the simulation of a Dirac free-particle trapped in a square well potential. To achieve good spacial resolution to minimize the error, we used 9 qubits. This would also allow to explore deeply the available space. During roughly thirty days, we executed 954 quantum circuits distributed by 7 nodes. In figure 4.14 is displayed a graph of the temporal evolution and in figures 4.15, 4.16, 4.17 histograms of specific time snippets.

In the plots we wanted to mainly visualize the correct propagation of the particle in the available space but also the reflection on the barriers, defined as the ends of the available space (positions 0 and 511).



Figure 4.14: High definition simulation of a Dirac free-particle trapped in a squared potential well. The available exploratory space was $N = 2^9 = 512$ discrete positions and were done 954 steps.

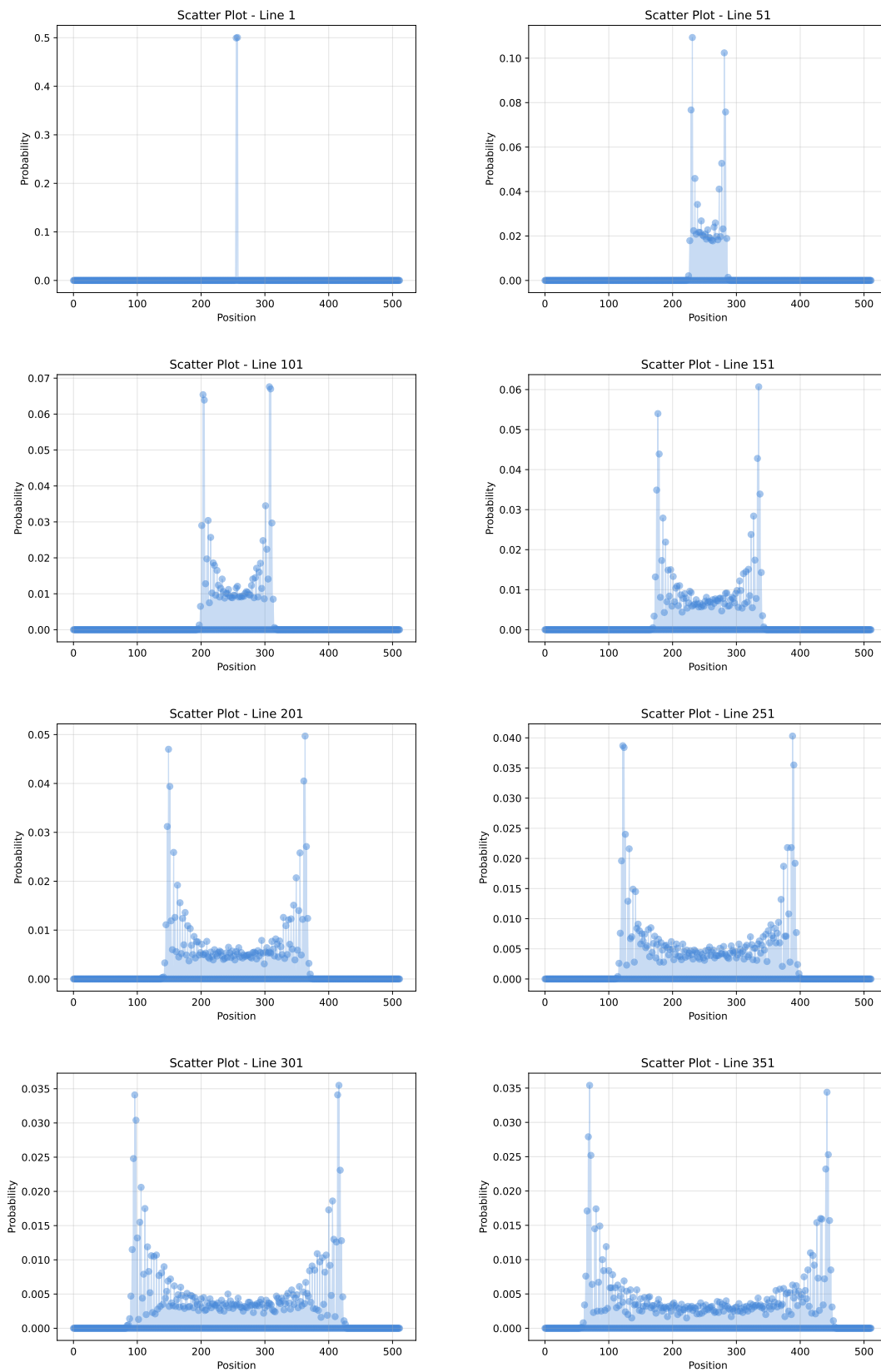


Figure 4.15: Scatter plots for the simulation of a Dirac free-particle trapped in a squared potential well for separated in time snippets. (1/3)

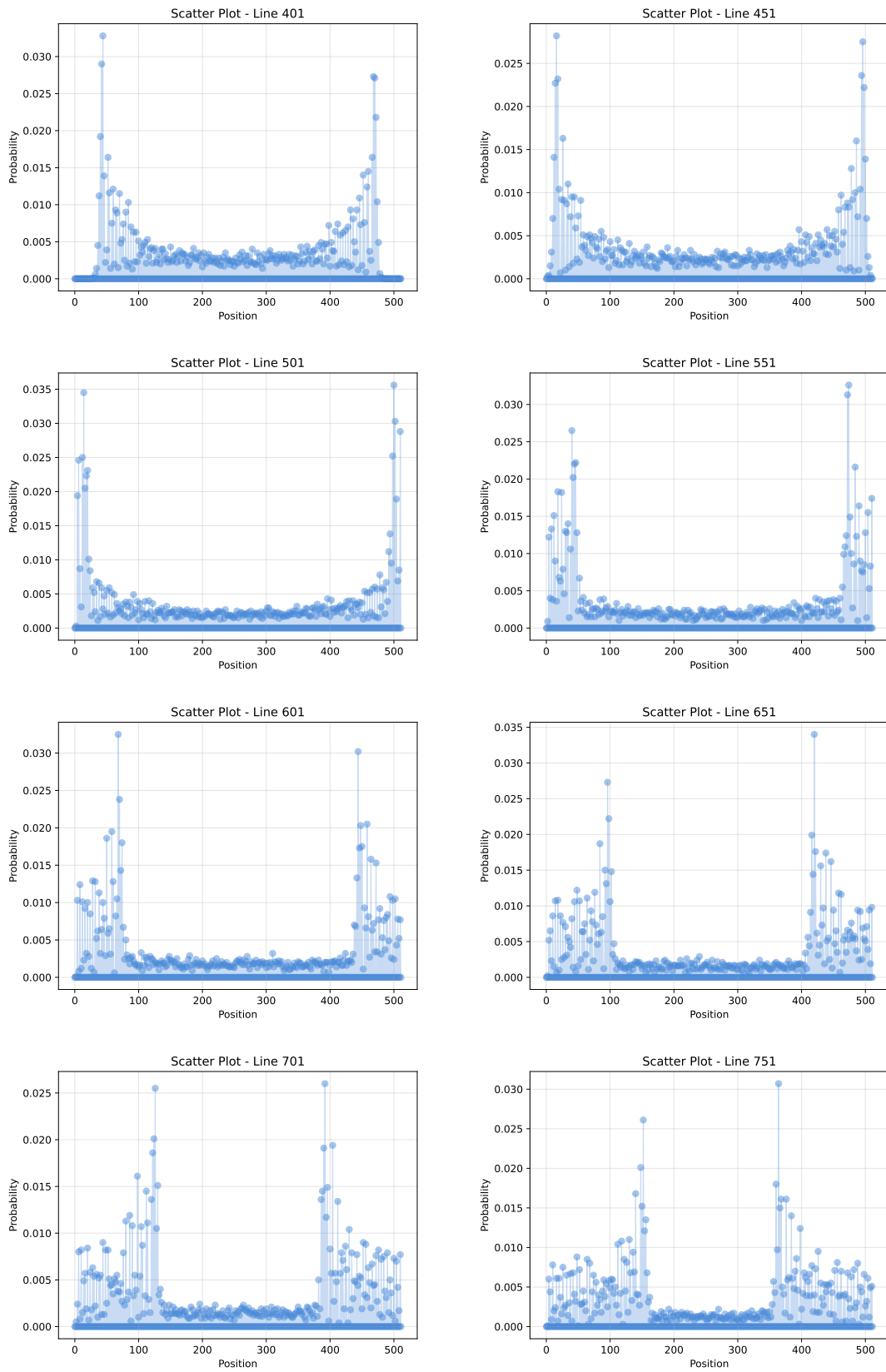


Figure 4.16: Scatter plots for the simulation of a Dirac free-particle trapped in a squared potential well for separated in time snippets. (2/3)

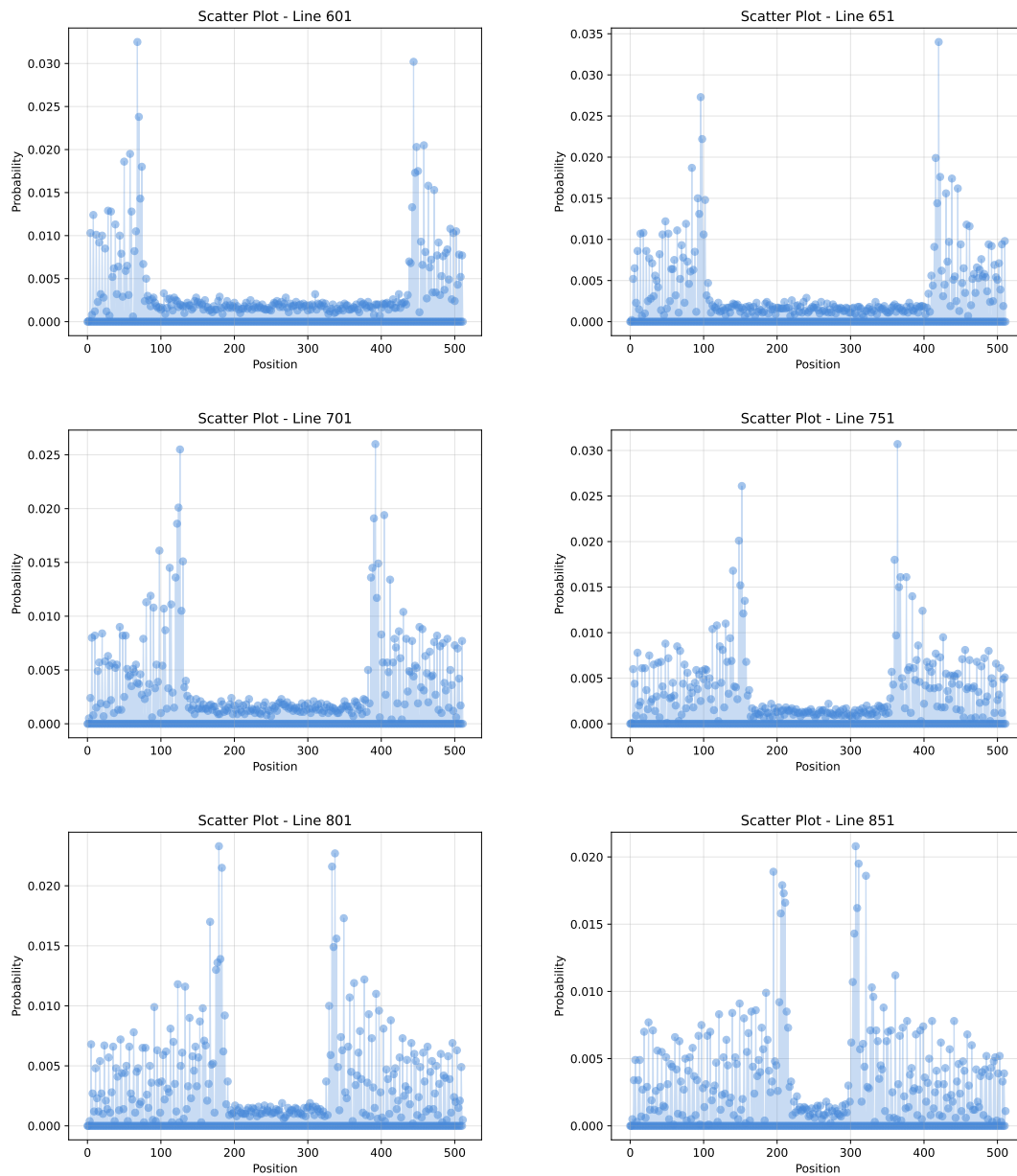


Figure 4.17: Scatter plots for the simulation of a Dirac free-particle trapped in a squared potential well for separated in time snippets. (3/3)

Chapter 5

Discussion and Conclusion

Great new technologies and innovations usually open up vast new opportunities leading to a succession of events until it is possible to achieve its true potential. The idea of quantum computing led to the investment in quantum hardware, but soon it was realized that to achieve *quantum supremacy* in between steps would need to be made. Hybrid quantum-classical computing has the possibility to fill but also further develop for this in between void.

The main goal for this thesis was to develop an hybrid algorithm and to attribute a specific problem for it to solve. Secondly was to approach the HPC and quantum computing worlds to more easily introduce hybridization in algorithms. We believe these goals were accomplished even though in ways we did not expect.

In Chapter 2 we not only laid down all the prerequisites for developing such algorithm but also showed the potential and existing work that has been made on the field, which contributed with ideas and motivation.

More importantly, in Chapter 3 the framework for our hybrid algorithm was presented, the quantum walk algorithm. Its hybridization was focused on reducing the depth and complexity of algorithm's quantum circuit by passing the quantum Fourier transform (QFT) to a classical computer (exactly by 2^n , with n the number of qubits). This can be an interesting and promising architecture for quantum walk based algorithms that require a relatively low step count, since the depth decrease would have a bigger impact. It also allows for a parallelization of our qubits since these are independent from each other, important element given the constraints in the number of qubits in current state of the art quantum computers. In a more broader perspective, it allows for any quantum algorithm that uses simply the QFT for a change of computational basis, to perform it in a classical computer.

Equally significant, in Appendix B we demonstrated that the execution of the free-Dirac particle in a quantum walk algorithm matches the free wave packet propagation for the same particle, something that had not been previously done. It followed the pure theoretical work done by Strauch [61] and gives confidence to develop more complex research in the same domain.

Chapter 4, focused on the more experimental component. Even though we im-

plemented successfully the algorithm's connection between classical and quantum computers, the latter still compromised a lot, resulting in noisy experiments. There is still a lot of compromise, at least in the public available hardware to get decent results, even in light computational tasks. In the other hand, we feel that there are still some techniques that seek to improve the performance of quantum hardware, such as multiple sequenced experiments, algorithm changes on the transpile function to further optimize the gates used and also qubit error correcting algorithms.

We then sought to further combine HPC and quantum computing by conducting a simulation of our algorithm on an HPC using a quantum computer simulator. As shown in section 4.2, although the simulation process was sub-optimal, it made software validation and verification possible. Moreover, despite the difficulty in extracting the maximum efficiency from the available hardware, the simulation showcased the potential of such simulators, given the high depth and resolution of the algorithm execution. Additionally, it's worth noting that many of the frameworks used (qiskit, qiskit aer) are still ongoing projects, primarily for educational purposes. On a positive note, our commitment to extracting every possible aspect of efficiency from qiskit aer led us to identify a software bug, which allowed us to contribute to the qiskit community.

For future work, we would want to demonstrate the match between the analytical result of the free-Dirac particle trapped in a square potential well and quantum algorithm, so we can validate the results from section 4.2.5. Besides this is to simulate our developed algorithm on more reliable quantum hardware and maximize the techniques to mitigate and minimize the environmental noise, element responsible for deprecating the quality of our qubits. For example using the state of the art computers privately available on IBMQ, such as the `ibm_seattle` that uses the Osprey QPU, being one of the most resource capable at the write of this work. This would allow us to truly understand the state of the art in hybrid algorithm performance (for our own algorithm) and perceive the needed steps for it to be competitive with the full-stack HPC. Besides this, exploring different quantum simulators such as the QuantuLOOP[73], HybridQ [59] or NWQ-Sim[58] so we can obtain results with even higher spacial resolution and step count. Finally we will also want this work to result in a scientific paper.

In sum, we certainly accomplished our main goals: we developed a hybrid algorithm that can reduce the workload of quantum computers but also spark ideas to adapt existing algorithm or create new ones; this research helped to contribute to the bridge between HPC and quantum computing, that even tough we did not specifically create an hybrid algorithm for HPC we helped to lay down the elements for it to happen. Also, in the beginning of March, we had the opportunity to share some details from this work on a workshop promoted by the University of Coimbra and IBM, focused on Quantum Computing, showing the interest of many to work on this field. Globally we were satisfied with the end product of this research work, assuming gains in both educational and experience levels.

References

- [1] Y. Aharonov, L. Davidovich, and N. Zagury. Quantum random walks. *Phys. Rev. A*, 48:1687–1690, Aug 1993. doi: 10.1103/PhysRevA.48.1687. URL <https://link.aps.org/doi/10.1103/PhysRevA.48.1687>.
- [2] Renato Portugal. *Quantum walks and search algorithms*. Quantum Science and Technology. Springer, New York, NY, 2013 edition, February 2013.
- [3] A. Ramezanzpour. Quantum walk in a reinforced free-energy landscape: Quantum annealing with reinforcement. *Physical Review A*, 106(1), jul 2022. doi: 10.1103/physreva.106.012418. URL <https://doi.org/10.1103/physreva.106.012418>.
- [4] James G Morley, Nicholas Chancellor, Sougato Bose, and Viv Kendon. Quantum search with hybrid adiabatic-quantum walk algorithms and realistic noise. September 2017.
- [5] Edward Farhi and Sam Gutmann. Quantum computation and decision trees. *Phys. Rev. A*, 58:915–928, Aug 1998. doi: 10.1103/PhysRevA.58.915. URL <https://link.aps.org/doi/10.1103/PhysRevA.58.915>.
- [6] M Szegedy. Quantum speed-up of markov chain based algorithms. In *45th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2004.
- [7] Dorit Aharonov, Andris Ambainis, Julia Kempe, and Umesh Vazirani. Quantum walks on graphs, 2002.
- [8] P Lara, A Leão, and R Portugal. Simulation of quantum walks using HPC. *J. Comput. Interdiscip. Sci.*, 6(1), 2016.
- [9] Marek Sawerwain and Roman Gielerek. GPGPU based simulations for one and two dimensional quantum walks. In *Computer Networks*, pages 29–38. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-13861-4_3. URL https://doi.org/10.1007/978-3-642-13861-4_3.
- [10] A quantum accelerator to solve your computational needs. <https://meetiqm.com/products/for-hpc-centers/>, . Accessed: 2022-01-23.
- [11] Atos and iqm partner up in quantum simulation. <https://www.meetiqm.com/articles/press-releases/atos-and-iqm-partner-up-in-quantum-simulation/>, . Accessed: 2022-01-21.

- [12] The european high performance computing joint undertaking (eurohpc ju), . URL https://eurohpc-ju.europa.eu/index_en. Accessed: 2022-02-01.
- [13] Nvidia announces hybrid quantum-classical computing platform. <https://nvidianews.nvidia.com/news/nvidia-announces-hybrid-quantum-classical-computing-platform>, . Accessed: 2022-01-23.
- [14] Stephen M Barnett. *Quantum Information*. Oxford Master Series in Physics. Oxford University Press, London, England, May 2009.
- [15] Richard P Feynman. Quantum mechanical computers. *Found. Phys.*, 16(6): 507–531, June 1986.
- [16] Multiple-particle interference and quantum error correction. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 452(1954):2551–2577, nov 1996. doi: 10.1098/rspa.1996.0136. URL <https://doi.org/10.1098/rspa.1996.0136>.
- [17] Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52:R2493–R2496, Oct 1995. doi: 10.1103/PhysRevA.52.R2493. URL <https://link.aps.org/doi/10.1103/PhysRevA.52.R2493>.
- [18] Vincent Danos, Elham Kashefi, and Prakash Panangaden. The measurement calculus. April 2007.
- [19] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. January 2000.
- [20] David P. DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik*, 48(9-11):771–783, sep 2000. doi: 10.1002/1521-3978(200009)48:9/11<771::aid-prop771>3.0.co;2-e. URL [https://doi.org/10.1002/1521-3978\(200009\)48:9/11<771::aid-prop771>3.0.co;2-e](https://doi.org/10.1002/1521-3978(200009)48:9/11<771::aid-prop771>3.0.co;2-e).
- [21] A Yu Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191–1249, 1997. ISSN 0036-0279. doi: 10.1070/rm1997v052n06abeh002155.
- [22] Ibm unveils 400 qubit-plus quantum processor and next-generation ibm quantum system two. <https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two>. Accessed: 2022-01-21.
- [23] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Yuxuan Li, Xiao Jiang, Lin Gan, Guangwen Yang, Lixing You, Zhen Wang, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, December 2020.

- [24] P W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc. Press, 2002.
- [25] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*, New York, New York, USA, 1996. ACM Press.
- [26] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1):4213, July 2014.
- [27] David J Griffiths and Darrell F Schroeter. *Introduction to Quantum Mechanics*. Cambridge University Press, Cambridge, England, 3 edition, August 2018.
- [28] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 01 1965. ISSN 0010-4620. doi: 10.1093/comjnl/7.4.308. URL <https://doi.org/10.1093/comjnl/7.4.308>.
- [29] Dmitry A Fedorov, Bo Peng, Niranjan Govind, and Yuri Alexeev. VQE method: a short survey and recent developments. *Materials Theory*, 6(1):2, January 2022.
- [30] Enrico Fontana, M Cerezo, Andrew Arrasmith, Ivan Rungger, and Patrick J Coles. Non-trivial symmetries in quantum landscapes and their resilience to quantum noise. November 2020.
- [31] Ilya G Ryabinkin, Tzu-Ching Yen, Scott N Genin, and Artur F Izmaylov. Qubit coupled cluster method: A systematic approach to quantum chemistry on a quantum computer. *J. Chem. Theory Comput.*, 14(12):6317–6326, December 2018.
- [32] Nobuyuki Yoshioka, Hideaki Hakoshima, Yuichiro Matsuzaki, Yuuki Tokunaga, Yasunari Suzuki, and Suguru Endo. Generalized quantum subspace expansion. July 2021.
- [33] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. November 2014.
- [34] Bence Bakó, Adam Glos, Özlem Salehi, and Zoltán Zimborás. Near-optimal circuit design for variational quantum optimization. September 2022.
- [35] William J Huggins, Bryan A O’Gorman, Nicholas C Rubin, David R Reichman, Ryan Babbush, and Joonho Lee. Unbiasing fermionic quantum monte carlo with a quantum computer. *Nature*, 603(7901):416–420, March 2022.
- [36] Paulo H. Acioli. Review of quantum monte carlo methods and their applications. *Journal of Molecular Structure: THEOCHEM*, 394(2):75–85, 1997. ISSN 0166-1280. doi: [https://doi.org/10.1016/S0166-1280\(96\)04821-X](https://doi.org/10.1016/S0166-1280(96)04821-X). URL <https://www.sciencedirect.com/science/article/pii/S016612809604821X>. Proceedings of the Eighth Brazilian Symposium of Theoretical Chemistry.

- [37] Matthias Troyer and Uwe-Jens Wiese. Computational complexity and fundamental limitations to fermionic quantum monte carlo simulations. *Phys. Rev. Lett.*, 94:170201, May 2005. doi: 10.1103/PhysRevLett.94.170201. URL <https://link.aps.org/doi/10.1103/PhysRevLett.94.170201>.
- [38] Hsin-Yuan Huang, Michael Broughton, Masoud Mohseni, Ryan Babbush, Sergio Boixo, Hartmut Neven, and Jarrod R McClean. Power of data in quantum machine learning. *Nature Communications*, 12(1):2631, May 2021.
- [39] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. The effect of data encoding on the expressive power of variational quantum machine learning models. August 2020.
- [40] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, England, December 2010.
- [41] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. February 2018.
- [42] Ryan Sweke, Frederik Wilde, Johannes Meyer, Maria Schuld, Paul K Faehrmann, Barthélemy Meynard-Piganeau, and Jens Eisert. Stochastic gradient descent for hybrid quantum-classical optimization. October 2019.
- [43] Xiaoming Sun and Yufan Zheng. Hybrid decision trees: Longer quantum time is strictly more powerful. November 2019.
- [44] Nai-Hui Chia, Kai-Min Chung, and Ching-Yi Lai. On the need for large quantum depth. September 2019.
- [45] Robert Robey and Yuliana Zamora. *Parallel and high performance computing*. Manning Publications, New York, NY, June 2021.
- [46] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67 (Spring)*, New York, New York, USA, 1967. ACM Press.
- [47] M. Crovella, R. Bianchini, T. LeBlanc, E. Markatos, and R. Wisniewski. Using communication-to-computation ratio in parallel program design and performance prediction. In *[1992] Proceedings of the Fourth IEEE Symposium on Parallel and Distributed Processing*, pages 238–245, 1992. doi: 10.1109/SPDP.1992.242738.
- [48] Introduction to infinibandTM, . URL https://network.nvidia.com/pdf/whitepapers/IB_Intro_WP_190.pdf. Accessed: 2022-02-02.
- [49] Introducing ndr 400gb/s infinibandTM, . URL <https://www.nvidia.com/en-us/on-demand/session/supercomputing2020-sc2018/>. Accessed: 2022-02-02.

-
- [50] David Luebke, Mark Harris, Naga Govindaraju, Aaron Lefohn, Mike Houston, John Owens, Mark Segal, Matthew Papakipos, and Ian Buck. S07—GPGPU. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing - SC '06*, New York, New York, USA, 2006. ACM Press.
- [51] Nvidia tensor cores: Versatility for hpc and ai, . URL <https://www.nvidia.com/en-us/data-center/tensor-cores/>. Accessed: 2022-02-01.
- [52] Nvidia h100 tensor core gpu, . URL <https://www.nvidia.com/en-us/data-center/h100/>. Accessed: 2022-02-01.
- [53] Top 500, the list. URL <https://top500.org/lists/top500/2022/11/>. Accessed: 2022-02-01.
- [54] Lumi - pre-exascale eurohpc supercomputer located in kajaani, finland. URL <https://www.lumi-supercomputer.eu>. Accessed: 2022-02-01.
- [55] One step closer to exascale: Eurohpc ju and forschungszentrum jülich sign the hosting agreement for exascale supercomputer jupiter. URL <https://eurohpc-ju.europa.eu/one-step-closer-exascale-eurohpc-ju-and-forschungszentrum-julich-sign-hosting-en>. Accessed: 2022-02-01.
- [56] The digital europe programme, . URL <https://digital-strategy.ec.europa.eu/en/activities/digital-programme>. Accessed: 2022-02-01.
- [57] IBM. Aer simulator; qiskit aer 0.12.1 documentation — qiskit.org. https://qiskit.org/ecosystem/aer/stubs/qiskit_aer.AerSimulator.html#qiskit_aer.AerSimulator. [Accessed 06-Jul-2023].
- [58] Northwest quantum simulator (nwq-sim) framework. <https://www.pnnl.gov/publications/scalable-simulation-quantum-circuits>. [Accessed 19-08-2023].
- [59] Github - nasa/hybridq. <https://github.com/nasa/hybridq>. [Accessed 19-08-2023].
- [60] C. M. Chandrashekar, R. Srikanth, and Raymond Laflamme. Optimizing the discrete time quantum walk using a $su(2)$ coin. *Phys. Rev. A*, 77:032326, Mar 2008. doi: 10.1103/PhysRevA.77.032326. URL <https://link.aps.org/doi/10.1103/PhysRevA.77.032326>.
- [61] Frederick W. Strauch. Relativistic quantum walks. *Phys. Rev. A*, 73:054302, May 2006. doi: 10.1103/PhysRevA.73.054302. URL <https://link.aps.org/doi/10.1103/PhysRevA.73.054302>.
- [62] A. J. Bracken, D. Ellinas, and I. Smyrnakis. Free-dirac-particle evolution as a quantum random walk. *Phys. Rev. A*, 75:022322, Feb 2007. doi: 10.1103/PhysRevA.75.022322. URL <https://link.aps.org/doi/10.1103/PhysRevA.75.022322>.

-
- [63] C. M. Chandrashekar, Subhashish Banerjee, and R. Srikanth. Relationship between quantum walks and relativistic quantum mechanics. *Phys. Rev. A*, 81: 062340, Jun 2010. doi: 10.1103/PhysRevA.81.062340. URL <https://link.aps.org/doi/10.1103/PhysRevA.81.062340>.
- [64] Xingyou Song. Quantum cellular automata models for general dirac equation, 2019.
- [65] Julien Zylberman, Giuseppe Di Molfetta, Marc Brachet, Nuno F. Loureiro, and Fabrice Debbasch. Hybrid quantum-classical algorithm for hydrodynamics, 2022.
- [66] Ugo Nzongani, Julien Zylberman, Carlo-Elia Doncecchi, Armando Pérez, Fabrice Debbasch, and Pablo Arnault. Quantum circuits for discrete-time quantum walks with position-dependent coin operator, 2023.
- [67] Asif Shakeel. Efficient and scalable quantum walk algorithms via the quantum fourier transform. December 2019.
- [68] Gene H Golub and Charles F Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 4 edition, February 2013.
- [69] Julien Zylberman, Giuseppe Di Molfetta, Marc Brachet, Nuno F Loureiro, and Fabrice Debbasch. Hybrid quantum-classical algorithm for hydrodynamics. February 2022.
- [70] Universidade de Coimbra. Navigator cluster. URL <https://www.uc.pt/lca/computing-resources/navigator-cluster/>.
- [71] IBM. Transpiler (qiskit.transpiler). qiskit 0.43.2. <https://qiskit.org/documentation/apidoc/transpiler.html>, 2023. [Accessed 17-Jul-2023].
- [72] Pull request # 1880: Increased accuracy of simulator calculation, 2022. URL <https://github.com/Qiskit/qiskit-aer/pull/1880>. Accessed: 2022-08-01.
- [73] Quantum simulation architectures in hpc systems. <https://quantumspain-project.es/en/quantum-simulation-architectures-in-hpc-systems/>. [Accessed 19-08-2023].

Appendices

Appendix A

The Postulates of Quantum Mechanics

There are six postulates of quantum mechanics [27] being the foundation of the theory and provide a mathematical framework for describing the behavior of matter and energy at the atomic and subatomic level.

Postulate 1

The state of a quantum mechanical system is completely specified by the function $\Psi(\mathbf{r}, t)$ that depends on the coordinates of the particle, \mathbf{r} and the time, t . This function is called the wavefunction or state function and has the property that $\Psi^*(\mathbf{r}, t)\Psi(\mathbf{r}, t)d\tau$ is the probability that the particle lies in the volume element $d\tau$ located at \mathbf{r} and time t .

This follows as the *probabilistic* or *statistical* interpretation of the wavefunction. As a results the wavefunction must satisfy the condition that finding the particle *somewhere* in space is 1 and this gives us the normalisation condition,

$$\int_{-\infty}^{+\infty} \Psi^*(\mathbf{r}, t)\Psi(\mathbf{r}, t)d\tau \quad (\text{A.1})$$

The other conditions that arise from this interpretation are that is must be single-valued, continuous and finite.

Postulate 2

To every observable in classical mechanics there corresponds a linear, Hermitian operator in quantum mechanics.

This postulate comes from the observation that the expectation value of an operator that corresponds to an observable must be real and therefore the operator must be Hermitian. Some examples of Hermitian operators are:

Observable	Classical Symbol	Quantum Operator	Operation
position	\mathbf{r}	\hat{r}	multiply by \mathbf{r}
momentum	\mathbf{p}	\hat{p}	$-i\hbar(\hat{i}\frac{\partial}{\partial x} + \hat{j}\frac{\partial}{\partial y} + \hat{k}\frac{\partial}{\partial z})$
kinetic energy	T	\hat{T}	$-\frac{\hbar^2}{2m}(\hat{i}\frac{\partial}{\partial x^2} + \hat{j}\frac{\partial}{\partial y^2} + \hat{k}\frac{\partial}{\partial z^2})$
potential energy	$V(\mathbf{r})$	$\hat{V}(\mathbf{r})$	multiply by $V(\mathbf{r})$
total energy	E	\mathcal{H}	$-\frac{\hbar^2}{2m}(\hat{i}\frac{\partial}{\partial x^2} + \hat{j}\frac{\partial}{\partial y^2} + \hat{k}\frac{\partial}{\partial z^2}) + V(\mathbf{r})$

Table A.1: List of Hermitian operators.

Postulate 3

In any measurement of the observable associated with operator \hat{A} , the only values that will ever be observed are the eigenvalues, a , that satisfy the eigenvalue equation:

$$\hat{A}\Psi = a\Psi \quad (\text{A.2})$$

This shows that the values of dynamical variables are quantized in quantum mechanics (although it is possible to have a continuum of eigenvalues in the case of unbound states). If the system is in an eigenstate of \hat{A} with eigenvalue a then any measurement of the quantity A will always yield the value a .

Although measurement will always yield a value, the initial states does not have to be an eigenstate of \hat{A} . An arbitrary state can be expanded in the complete set of eigenvectors of \hat{A} , $\hat{A}\Psi_i = a_i\Psi_i$, as

$$\Psi = \sum_i^n c_i\Psi_i \quad (\text{A.3})$$

where n may go to infinity. In this case, measurement of A will yield *one* of the eigenvalues, a_i , but we do not know which one. The *probability* of observing the eigenvalue a_i is given by the absolute value of the square of the coefficient, $|c_i|^2$. This postulate also implies that, after the measurement of Ψ yields some value, a_i , the wavefunction *collapses* into the eigenstate, Ψ_i , that corresponds to a_i . If a_i is degenerate Ψ collapses onto the degenerate subspace. This the act of measurement affects the state of the system.

Postulate 4

If a system is in a state described by the normalised wavefunction, Ψ , then the average value of the observable corresponding to \hat{A} is given by:

$$\langle \hat{A} \rangle = \int_{-\infty}^{+\infty} \Psi^* \hat{A} \Psi \, d\tau \quad (\text{A.4})$$

Postulate 5

The wavefunction or state function of a system evolves in time according to the time-dependent Schrödinger equation:

$$\mathcal{H}\Psi(\mathbf{r}, t) = i\hbar \frac{\partial \Psi}{\partial t} \quad (\text{A.5})$$

Postulate 6

The total wavefunction must be anti symmetric with respect to the interchange of all coordinates of one fermion with those of another. Electronic spin must be included in this set of coordinates.

The Pauli exclusion principle is a direct result of this *antisymmetry* postulate.

Appendix B

Analytical overview on the one dimensional Dirac Quantum Walk

B.1 Evolution Operator

Let's consider the Hamiltonian for a free Dirac particle in the dimension configuration (1+1):

$$\mathcal{H} = \sigma_1 pc + \sigma_3 mc^2 \quad (\text{B.1})$$

where $p = -i\hbar(d/dx)$ and σ_1 and σ_3 are Pauli matrices. In the representation where σ_1 is diagonal (Weyl representation) we have:

$$\sigma_1^w = \sigma_3 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \sigma_3^w = \sigma_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad (\text{B.2})$$

Formerly, the Hamiltonian gains the shape:

$$\mathcal{H} = \sigma_3 pc + \sigma_1 mc^2 \quad (\text{B.3})$$

The time propagator for a state ψ is given by:

$$U(t) = e^{(-i/\hbar)Ht} \quad U(\Delta t)\psi(t) = \psi(t + \Delta t) \quad (\text{B.4})$$

As σ_1 and σ_3 do not commute, in general, the exponential of the Hamiltonian cannot be written as products of exponentials, except when the exponents are small, that is, for small Δt , i.e.:

$$e^{(-i/\hbar)H\Delta t} = e^{-i/\hbar(\sigma_3 pc + \sigma_1 mc^2)\Delta t} \sim I - \left(\frac{i}{\hbar} \sigma_3 pc + i\sigma_1 \frac{mc^2}{\hbar} \right) \Delta t \quad (\text{B.5})$$

$$\sim e^{-i/\hbar \sigma_3 pc \Delta t} e^{-i\sigma_1 mc^2 / \hbar \Delta t} = e^{-\sigma_3 c \Delta t (d/dx)} e^{-i\sigma_1 mc^2 / \hbar \Delta t} \quad (\text{B.6})$$

retaining terms up to 1st order in Δt . More explicitly, this happens when:

$$\frac{\langle p \rangle c}{\hbar} \Delta t \sim \frac{mc^2}{\hbar} \Delta t \ll 1 \quad (\text{B.7})$$

with $\langle p \rangle$ the *expectation value* of the linear momentum. One operator of type $e^{\Delta x(d/dx)}$ makes a translation in space, i.e., applied to the function $f(x)$ we have:

$$e^{\Delta x(d/dx)} f(x) = \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} \left(\frac{d}{dx} \right)^n f(x) = \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} f(x)^{(n)} = f(x + \Delta x) \quad (\text{B.8})$$

as one can verify from the Taylor series expansion of $f(x + \Delta x)$ around x . The operator $e^{\sigma_3 \Delta x(d/dx)}$ acts in the space of 2-dimensional spinors in the 1+1 space (1 spatial dimension + one temporal dimension). We can decompose a spinor of this type in the form

$$\Psi(x, t) = \begin{bmatrix} \psi_+(x, t) \\ \psi_-(x, t) \end{bmatrix} = \psi_+(x, t) \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \psi_-(x, t) \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (\text{B.9})$$

Since the spinors $\begin{bmatrix} 1 & 0 \end{bmatrix}^T$ and $\begin{bmatrix} 0 & 1 \end{bmatrix}^T$ are spinors eigenstates of σ_3 with eigenvalues +1 and -1 respectively, it is obtained (with $\Delta x = c \Delta t$)

$$e^{-\sigma_3 \Delta x(d/dx)} \Psi(x, t) = e^{-\sigma_3 \Delta x(d/dx)} \psi_+(x, t) \begin{bmatrix} 1 \\ 0 \end{bmatrix} + e^{-\sigma_3 \Delta x(d/dx)} \psi_-(x, t) \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \quad (\text{B.10})$$

$$= e^{-\Delta x(d/dx)} \psi_+(x, t) \begin{bmatrix} 1 \\ 0 \end{bmatrix} + e^{-\Delta x(d/dx)} \psi_-(x, t) \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \psi_+(x - \Delta x, t) \\ \psi_-(x + \Delta x, t) \end{bmatrix} \quad (\text{B.11})$$

The action of $e^{-i\sigma_1 mc^2/\hbar \Delta t}$ on the spinor $\Psi(x, t)$ is

$$e^{-i\sigma_1 mc^2/\hbar \Delta t} \Psi(x, t) \sim \left(I - i\sigma_1 \frac{mc^2}{\hbar} \Delta t \right) \begin{bmatrix} \psi_+(x, t) \\ \psi_-(x, t) \end{bmatrix} = \begin{bmatrix} \psi_+(x, t) \\ \psi_-(x, t) \end{bmatrix} - i \frac{mc^2}{\hbar} \Delta t \begin{bmatrix} \psi_-(x, t) \\ \psi_+(x, t) \end{bmatrix} \quad (\text{B.12})$$

The action of the evolution operator comes as:

$$U(\Delta t) \Psi(x, t) = e^{-\sigma_3 c \Delta t(d/dx)} e^{-\sigma_1 (mc^2/\hbar) \Delta t} \Psi(x, t) = \quad (\text{B.13})$$

$$= \left(\begin{bmatrix} \psi_+(x, t) \\ \psi_-(x, t) \end{bmatrix} - i \frac{mc^2}{\hbar} \Delta t \begin{bmatrix} \psi_-(x, t) \\ \psi_+(x, t) \end{bmatrix} \right) \quad (\text{B.14})$$

$$= \begin{bmatrix} \psi_+(x - \Delta x, t) \\ \psi_-(x + \Delta x, t) \end{bmatrix} - i \frac{mc^2}{\hbar} \Delta t \begin{bmatrix} \psi_+(x - \Delta x, t) \\ \psi_-(x + \Delta x, t) \end{bmatrix} \rightarrow \Psi(x, t + \Delta t) \quad (\text{B.15})$$

The evolution for an instance $t = n\Delta t$ is given by

$$U(t) \Psi(x, t) = U(n\Delta t) \Psi(x, 0) = U^n(\Delta t) \Psi(x, t) \underset{\tau \ll 1}{\sim} \left(e^{\sigma_3 c \Delta t(d/dx)} \right)^n \left(e^{-i\sigma_1 (mc^2/\hbar) \Delta t} \right) \Psi(x, 0) \quad (\text{B.16})$$

where $\tau = (mc^2/\hbar)$. The first operator is the shift operator and the second the coin. These only commute until the first order in Δt and therefore the algorithm application depends from this approximation.

B.2 Spacial Grid

In a box of size L , we can define a grid of point $x_i = i\Delta x - L/2$, $i = 0, 1, \dots, N$, with N even, $\Delta x = L/N$. If we want to identify the instance where discrete increases on the index $n = 0, 1, \dots$

If we also identify the instant using discrete increases of the instants by an index $n = 0, 1, \dots$ such that $t_n = n\Delta t$, the wave function is described as $\Psi(x_i, t_n)$ or simply $\Psi(i, n)$. Thus, the action of the evolution operator is given by:

$$U(\Delta t)\Psi(i, n) = \Psi(i, n+1) \sim U(\Delta t) \begin{bmatrix} \psi_+(i, n) \\ \psi_-(i, n) \end{bmatrix} = \begin{bmatrix} \psi_+(i-1, n) \\ \psi_-(i+1, n) \end{bmatrix} - i \frac{mc^2}{\hbar} \Delta t \begin{bmatrix} \psi_+(i, n) \\ \psi_-(i, n) \end{bmatrix} \quad (\text{B.17})$$

B.3 Wave packet

For an eigenstate of \mathcal{H} with well defined momentum p and positive energy $E = \sqrt{m^2c^4 + p^2c^2}$ is given by:

$$\Psi_p(x, t) = \frac{N}{E} e^{-(i/\hbar)px} e^{(i/\hbar)Et} \begin{bmatrix} E + pc + mc^2 \\ E - pc + mc^2 \end{bmatrix} \quad (\text{B.18})$$

where N is a normalization constant

A wave packet with these characteristics is given by

$$\Psi(x, t) = \int dp f(p) \Psi_p(x, t) \quad (\text{B.19})$$

If we choose:

$$f(p) = e^{-aE/(\hbar c)} \quad (\text{B.20})$$

$a > 0$, the integral

$$\Psi(x, t) = \int_{-\infty}^{\infty} dp \frac{N}{E} e^{-aE/(\hbar c)} e^{(i/\hbar)px} e^{(i/\hbar)Et} \begin{bmatrix} E + pc + mc^2 \\ E - pc + mc^2 \end{bmatrix} \quad (\text{B.21})$$

can be calculated analytically. Depending on whether the remaining terms are even or odd, the exponential $e^{(i/\hbar)px} = \cos(px/\hbar) + i \sin(px/\hbar)$ contributes with the cos or sin term, respectively.

Thus, one has (formulas 3.914 from Gradstein and Ryzhik)

$$\Psi(x, t) = Nmc \begin{bmatrix} K_0(sm c/\hbar) + s^{-1}[a + i(ct + x)]K_1(sm c/\hbar) \\ K_0(sm c/\hbar) + s^{-1}[a + i(ct - x)]K_1(sm c/\hbar) \end{bmatrix} \quad s = \sqrt{(a + ict)^2 + x^2} \quad (\text{B.22})$$

with K_0 and K_1 modified Bessel function of degree 0 and 1. This expression coincides with the one from Phys. Rev. A **73** 054302. Since this is an exact result:

$$\Psi(x, t + \Delta t) = U(\Delta t)\Psi(x, t) \quad (\text{B.23})$$

for any t . The probability at the end of Δn is given by $|\Psi(x, n\Delta t)|^2$

Finally, the goal is to understand if the results from the free-Dirac particle Quantum Walk algorithm execution can reproduce a good estimate of the analytical result. This allows us to understand the fidelity of such algorithm since the result of this specific problem has a well known solution.

The figure B.1 depicts an overlap between the analytical solution from the $\Psi(x, t)$ wave packet (with parameters $a = 0.311$, $t = 50$, $c = m = 1$) with the free-Dirac particle Quantum Walk algorithm execution (using 7 qubits and 100 unitary time steps and as starting state a single particle in the middle of the lattice). The only parameters we tuned was a for the wave packet and the algorithm time steps.

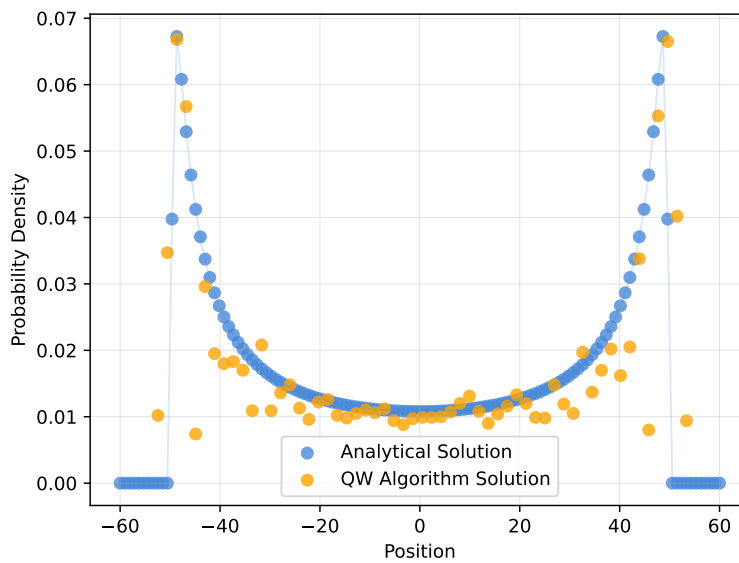


Figure B.1: Overlap between analytical solution from the $\Psi(x, t)$ wave packet and free-Dirac particle Quantum Walk algorithm execution.

As we could see, this roughly demonstrates a good match between the two results from different calculations. A perfect match was not expected due to the sampling nature of quantum algorithms.

Appendix C

ibm_perth **calibration data**

In the next page we have available a table with all the calibration data for the `ibm_perth` quantum backend. This quantum information processor has 7 qubits and uses the processor architecture Falcon r5.11H. At the time of this work it was in version 1.2.8.

The errors value should be interpreted as a specified amount of erroneous values out of the total measurement, i.e., for a gate error with $4.40 \cdot 10^{-2}$, in 1000 measurements 44 of those values are going to be faulty. It is also needed to take into consideration the gate times with the respective T_1 and T_2 times.

Table C.1: Calibration data for the `ibm_perth` quantum backend.

Qubit	T1 (us)	T2 (us)	Frequency (GHz)	Anharmonicity (GHz)	Readout assignment error
0	235.0038292025969	96.96857445392642	5.157566039436631	-0.3415245174784648	0.027599999999999958
1	164.2272579571736	57.2221209113439	5.033542120092729	-0.3443687022163673	0.024399999999999977
2	146.75142160474985	104.76198839620311	4.862642920006381	-0.3472724723347956	0.024699999999999944
3	168.75993143015742	196.9594262775911	5.125101581016992	-0.3404418992142746	0.015900000000000025
4	115.74326190732062	113.48692383727457	5.159209394867257	-0.33336653711831216	0.025800000000000045
5	150.63356258873085	70.15126534253005	4.978591251015995	-0.3460220316533784	0.030699999999999995
6	160.85396029791193	145.46188568978997	5.156638027876555	-0.34045439072670325	0.011099999999999999
Qubit	Prob meas0 prep1	Prob meas1 prep0	Readout length (ns)	ID error	\sqrt{x} (sx) error
0	0.03159999999999996	0.0236	721.7777777777777	0.0001700321078925373	0.0001700321078925373
1	0.022399999999999975	0.0264	721.7777777777777	0.00034339798964320886	0.00034339798964320886
2	0.0204	0.029000000000000026	721.7777777777777	0.00022249783838757602	0.00022249783838757602
3	0.0142	0.017599999999999995	721.7777777777777	0.00023220211560893606	0.00023220211560893606
4	0.0278	0.023800000000000043	721.7777777777777	0.0004769326352302657	0.0004769326352302657
5	0.030800000000000005	0.0306	721.7777777777777	0.00026359173269770256	0.00026359173269770256
6	0.01419999999999999	0.008	721.7777777777777	0.0003474534930657497	0.0003474534930657497
Qubit	Pauli-X error	CNOT error	Gate time (ns)		
0	0.000170032107892537	0_ 1:0.00536823611917	0_ 1:391.111		
1	0.000343397989643208	1_ 3:0.0044514000269; 1_ 2:0.009209922939; 1_ 0:0.005368236119	1_ 3:369.776; 1_ 2:640; 1_ 0:426.666		
2	0.000222497838387576	2_ 1:0.00920992293940	2_ 1:604.445		
3	0.000232202115608936	3_ 5:0.00734669232179; 3_ 1:0.004451400027	3_ 5:284.444; 3_ 1:334.223		
4	0.000476932635230265	4_ 5:0.01066779916082	4_ 5:590.222		
5	0.000263591732697702	5_ 6:0.01277198089002; 5_ 4:0.010667799160; 5_ 3:0.007346692322	5_ 6:640; 5_ 4:625.778; 5_ 3:320		
6	0.000347453493065749	6_ 5:0.01277198089002	6_ 5:604.445		

Appendix D

Open sourced Quantum Walk Code

The code we developed is made available in the following GitHub web link: <https://github.com/dup0nt/Dirac-Quantum-Walk>

All code developed for this work, ran in the software versions:

- Python: 3.9.12
- Numpy: 1.23.1
- Qiskit: 0.44.0
 - qiskit-aer: 0.12.2
 - qiskit-ibmq-provider: 0.20.2
 - qiskit-terra: 0.25.0

