Memory Management in Software Quantum Computing Simulators

Gilberto Javier Díaz Toro

Doctoral thesis to qualify for the title of Doctor in Computer Science

Advisors:
Ph.D. Carlos Jaime Barrios Hernández
Ph.D. Luiz Angelo Steffenel
Ph.D. Jean-françois Couturier

Universidad Industrial de Santander
Facultad de Ingenierías Físico Mecánicas
Escuela de Ingeniería de Sistemas e Informática

Bucaramanga

2024

Abstract

Title: Memory Management in Software Quantum Computing Simulators. [1]

Author: Gilberto Javier Díaz Toro. [2]

keywords: Quantum Computing, High-Performance Computing, Quantum Simulators.

Description: Quantum computing promises unprecedented computational power by leveraging quantum mechanical principles such as superposition and entanglement. However, current quantum hardware is limited by the number of qubits and high error rates, making it challenging to test complex quantum algorithms at scale. As a result, software-based quantum simulators running on classical computers have become essential tools for exploring quantum algorithms and principles. These simulators, while powerful, require significant memory resources due to the exponential growth of quantum state sizes. For example, representing an n-qubit state demands 2^n complex numbers, placing an immense burden on classical memory systems.

This research investigates memory management strategies to optimize resource usage in quantum computing simulators. Techniques explored include state pruning, full-state data structures, quantum gate application optimizations, data compression, and distributed memory approaches. The goal is to reduce memory consumption while preserving the accuracy of quantum simulations. The study implements these strategies in a quantum simulator, providing a detailed comparison of their performance across various quantum algorithms. The results highlight the trade-offs between computational efficiency and memory usage, offering insights into the most effective approaches for scaling quantum simulations on classical hardware.

[1] Doctoral thesis

[2] Facultad de Ingeniería Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática

Advisors: Ph.D. Carlos Jaime Barrios Hernández (cbarrios@uis.edu.co), Ph.D. Luiz Angelo Steffenel (luiz-angelo.steffenel@univ-reims.fr), Ph.D. Jean-françois Couturier (jean-francois.couturier@univ-reims.fr)

Resumen

Título: Gestión de memoria en simuladores de computación cuántica por software. [3]

Autor: Gilberto Javier Díaz Toro. [4]

Palabras clave: Computación cuántica, computación de alto rendimiento, simuladores cuánticos.

Descripción: La computación cuántica promete una potencia computacional sin precedentes al aprovechar principios mecánicos cuánticos como la superposición y el entrelazamiento. Sin embargo, el hardware cuántico actual está limitado por la cantidad de cúbits y las altas tasas de error, lo que dificulta la prueba de algoritmos cuánticos complejos a escala. Como resultado, los simuladores cuánticos basados en software que se ejecutan en computadoras clásicas se han convertido en herramientas esenciales para explorar algoritmos y principios cuánticos. Estos simuladores, aunque potentes, requieren recursos de memoria significativos debido al crecimiento exponencial de los tamaños de los estados cuánticos. Por ejemplo, representar un estado de n cúbits requiere 2^n números complejos, lo que supone una carga inmensa para los sistemas de memoria clásicos.

Este trabajo investiga estrategias de gestión de memoria para optimizar el uso de recursos en simuladores de computación cuántica. Las técnicas exploradas incluyen la eliminación de estados, las estructuras de datos de estado completo, las optimizaciones de aplicaciones de puertas cuánticas, la compresión de datos y los enfoques de memoria distribuida. El objetivo es reducir el consumo de memoria al tiempo que se preserva la precisión de las simulaciones cuánticas. El estudio implementa estas estrategias en un simulador cuántico, proporcionando una comparación detallada de su rendimiento en varios algoritmos cuánticos. Los resultados resaltan las compensaciones entre la eficiencia computacional y el uso de la memoria, ofreciendo información sobre los enfoques más efectivos para escalar simulaciones cuánticas en hardware clásico.

[3] Tesis de doctorado.

[4] Facultad de Ingeniería Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática

Directores: Ph.D. Carlos Jaime Barrios Hernández (cbarrios@uis.edu.co), Ph.D. Luiz Angelo Steffenel (luiz-angelo.steffenel@univ-reims.fr), Ph.D. Jean-françois Couturier (jean-francois.couturier@univ-reims.fr)

Content

Introduction **1**

1 Theoretical Framework **8**
1.1 Quantum Information Theory 9
1.1.1 Logical Qubit 10
1.1.2 Measurement of Single Qubit Quantum State 13
1.1.3 Entanglement 14
1.2 Quantum Computing 15
1.2.1 Building Blocks 16
1.2.2 Quantum Computing Models 17
1.2.3 Quantum Gates 18
1.2.4 Quantum Algorithms 23
1.3 Quantum Computing Simulators 27
1.3.1 Basis of Quantum Computing Simulation 27
1.4 State of the Art 33
1.4.1 Leading Open-Source Simulators for Quantum Computing 35
1.4.2 Simulators Comparison 42

2 Memory Management Strategies **45**
2.1 Proper Data Structures 45
2.2 Remove the least Probable Quantum States 46
2.3 Full State Data Structures 50
2.4 Applying Quantum Gates to a Quantum State 50
2.4.1 Applying single-qubit Gates 51
2.4.2 Applying two-qubits Gates 52
2.5 Data Compression 53
2.6 Distribute the Vector State Among Different Computing Nodes 55

3 Implementing a Simulator **57**
3.1 Representation of Fundamental Concepts 58
3.1.1 Single Processor Case 58

3.1.2 State Pruning . . . 62
3.1.3 Shared Memory Case . . . 63
3.2 Distributed Memory Case . . . 63
3.2.1 Compressing The State Vector . . . 65
3.3 Simulator Verification . . . 66
3.3.1 Quantum Gates Tests . . . 66

4 Results **67**
4.1 Algorithm Selected for Testing . . . 67
4.2 Test platform . . . 67
4.3 States Pruning . . . 67
4.4 Full-State Quantum Register . . . 68
4.5 Data Compression . . . 70
4.6 Distribute the Quantum Register Across Multiple Nodes . . . 71
4.7 Combination of Distributed Memory and Shared Memory Approaches . . . 72
4.8 Combination of Distributed Memory and Data Compression Approaches . 73
4.9 Quantum Simulators Comparison . . . 74

5 Conclusions **76**
5.1 Further Work . . . 78

A Quantum Gates **88**
A.1 Single Qubit Quantum Gates . . . 88
A.2 Multiple Qubits Quantum Gates . . . 91
A.2.1 Controlled Quantum Gates . . . 91

B Quantum Algorithms **95**

C TMFQS User Guide **102**
C.1 Requirements . . . 102
C.2 Cloning the Source Code . . . 102
C.3 Compiling TMFQS . . . 103

List of Figures

1.1 Quantum Information Theory Inheritance 9
1.2 Bits vs Qubits 10
1.3 Bloch Sphere 11
1.4 Transistor 16
1.5 Josephson junction 17
1.6 Classical Gates 19
1.7 The general outline of variational hybrid algorithms 26
1.8 Intel-QS Simulator 37
1.9 Quantum++ Simulator 38
1.10 qsim Simulator 39
1.11 QuEST Simulator 40
1.12 QuEST Simulator 41
1.13 Comparison of the quantum Fourier transform using different simulators and optimization techniques. 42

2.1 Basic Data Structures 46
2.2 Improved Data Structures 46
2.3 Full State Data Structures 51

3.1 Class Diagram of the Prototype 59
3.2 Linearized state vector 59
3.3 States pairs affected by qubit operation 61
3.4 State vector order using dynamic memory approach 62
3.5 Simulator profiling 63
3.6 Pairwise calculation scheme for a 5-qubit state vector 64
3.7 Data exchange between process 65
3.8 Test Quantum Circuit 66

4.1 QFT Performance of Dynamic Memory Management vs Full-State Approach 68
4.2 QFT Distributed Memory Performance 69
4.3 QFT Full State with ZFP (log_{10}) 70
4.4 QFT Full State with ZFP Data Size 71
4.5 QFT Distributed Memory Performance 72

4.6 Shared Memory and Distributed Memory Parallelism 73
4.7 QFT Full State with MPI with Data Compression 74
4.8 Quantum Simulators Comparison 75

B.1 The 5 complex 5th roots of 1 96
B.2 Quantum Circuit for QFT 97
B.3 Deutsch Jozsa Algorithm 98
B.4 Quantum circuit implementing the general Deutsch–Jozsa algorithm. . . . 101

*

Glossary

AMD Advanced Micro Devices (AMD), a company specializing in developing processors and related technologies for computers, including CPUs and GPUs. 34

CPU Central Processing Unit (CPU), the primary component of a computer responsible for executing instructions and performing calculations. 24

GPU Graphics Processing Unit (GPU), a specialized processor initially designed for rendering graphics, now widely used in scientific and computational fields to perform parallel processing of large datasets, particularly useful in machine learning and scientific simulations. 34

MPI Message Passing Interface (MPI), a standardized communication protocol used to enable parallel processing in distributed computing environments. 35

OpenMP An API for parallel programming in shared memory environments, enabling developers to write code that takes advantage of multi-core processors by distributing tasks among multiple threads. 35

QFT Quantum Fourier Transform (QFT), a quantum algorithm used for Fourier transformation, essential in many quantum algorithms, such as Shor's algorithm for factoring large numbers. 27

QPU Quantum Processing Unit (QPU), the quantum equivalent of a CPU, specifically designed to perform quantum computations by leveraging qubits and quantum gates. 24

RAM Random Access Memory (RAM), a type of computer memory that can be accessed randomly; essential for storing data temporarily during computational tasks. 5

ZFP A compression algorithm designed for reducing the storage size of floating-point arrays, often used in simulations where storage efficiency is critical. 51

Introduction

An innovative model was developed in the second half of the 20th century by scientists who combined two remarkable theories: Information Theory [1] and Quantum Mechanics [2].

The first is about studying the transmission, processing, extraction, and utilization of information. The second one is about the behavior of matter and its interactions with energy on the scale of atoms and subatomic particles. The result was a new point of view of computation and information, the Quantum Information Theory [3, 4, 5, 6, 7, 8, 9, 10].

Information theory has abstracted away the physical part of the devices used for computation and communication so that it is possible to talk about the efficiency of an algorithm or the robustness of a communication protocol without understanding the details of the underlying physics. Quantum information theory applies all interesting quantum mechanics concepts directly to computing, like superposition and entanglement. The contribution of quantum mechanics to developing new computing devices, from the transistor to the latest advances in hardware, has been highly significant. Until a few recent decades, the influence of quantum mechanics was only in low-level implementation; it did not affect how the computation or communication was made. However, the Quantum Information Theory provides a new paradigm of computation, where the model of quantum mechanics is not only limited to hardware but is applied directly to processing and transmitting information. In other words, Quantum Information Theory is the research area of computer science that applies the principles of quantum mechanics to how calculations are performed.

New technological advances have allowed an increasing development of real quantum devices available through the cloud, with up to hundreds of qubits on universal quantum computers (UQC) and thousands of qubits on annealer (QA) devices. However, they are expected to be very limited in the near term in the number and quality of their fundamental component, the qubit. It enables the possibility of using real quantum hardware to solve elementary problems for the first time. Enormous challenges such as qubit connectivity limitations, high noise levels, and full error-correction overhead are the main obstacles to quantum hardware's ability to incorporate the proper amount of qubits to deliver the theoretical speedup many quantum algorithms have promised. Therefore, using them for practical applications that require thousands of qubits [11] is challenging.

The early stages in the development of quantum hardware and the growing interest in quantum computing have caused many organizations to focus on developing software

quantum simulators that run on classical computers. These simulators are a popular tool suitable for testing quantum computing concepts on ideal conditions, avoiding dealing with hardware challenges like the limited number and quality of physical qubits and quantum error correction, among others. A list of the most important developments can be found in [12, 13, 14]. This large number of projects reflects the area's growth and makes it difficult for researchers to decide which tool to use in their research.

However, the simulation of quantum computing models in classical computers requires exponential time and involves highly complex memory management. The problem is that using conventional techniques to simulate an arbitrary quantum process significantly more prominent than any of the existing quantum prototypes would soon require considerable memory on a classical computer. Therefore, researchers try to reduce such challenges by proposing efficient simulators.

Several initiatives are trying to reduce the consumption of classical resources by quantum simulators for example, Jianxin Chen et al. [15] work on a new technique based on Google's model for variable elimination in the line graph that implements a single-amplitude simulator; Aidan Dang et al. [16] studies how the entanglement structure of Shor's algorithm [17] is suitable for a particular matrix product state representation that quantifiably reduces the computational requirements for simulating it in a classical computer and Xin-Chuan Wu et al. [18] implements a lossy compression algorithm to decrease the amount of memory usage. The aim is to re-design quantum simulators using at least one of these techniques or a mix of them to test quantum algorithms with significant dimensions. These works comprise different efforts using distinct techniques to improve the consumption of classical resources. However, none of them represent a standard approach.

The following section describes the motivations for the rising interest in quantum computing due to the promise of accelerating the solution of several problems that are very difficult to solve using classical computers.

Motivation

One of the primary reasons to develop quantum computing is that, theoretically, it has been demonstrated that it allows efficient solutions to some complex problems whose best-known solution has an exponential cost for the input size. Quantum superposition, quantum uncertainty, and quantum entanglement are powerful resources that we can use to encode, decode, transmit, and process information in a highly efficient way that is impossible in the classical world. Richard Feynman, Yuri Manin, and others showed that a Turing machine could not efficiently simulate quantum phenomena related to entangled particles [19].

Another of the primary motivations for working in the science of quantum information is the possibility of fast quantum algorithms to solve essential computing problems. There has been considerable technical progress, but we need to go deeper into knowing what problems can be expected to surpass classic computers. Although, In theory, a quantum

computer can perform any task a classical computer can execute, this does not necessarily mean that a quantum computer exceeds a classical computer for all functions. If we use our classical algorithms in a quantum computer, we will similarly perform the calculation in a classical computer.

For a quantum computer to demonstrate its superiority, it must leverage algorithms designed to exploit the ability to simultaneously operate on all possible states, a concept known as quantum parallelism. However, while quantum parallelism enables the system to evaluate multiple possibilities simultaneously through superposition, this capability alone does not guarantee computational speedup. To truly outperform classical algorithms, quantum interference is required. Quantum interference allows for the constructive and destructive combination of probability amplitudes, enhancing correct outcomes while suppressing incorrect ones. This interference interaction enables quantum algorithms to solve problems efficiently, as seen in algorithms such as Grover's and Shor's.

These algorithms are not easy to design, but once they have been made, they produce spectacular results. One example of these algorithms is the quantum factorization algorithm created by Peter Shor of AT & T Bell Laboratories [17]. The algorithm addresses the problem of factoring large numbers into their prime factors. This task is classically challenging to solve; it is so difficult that it forms the basis of RSA encryption. Shor's algorithm uses the effects of quantum parallelism to quickly give the results of the main factorization problem, whereas a classical computer would take years. The following section describes the problem we will treat in this work.

Quantum computing simulators play a crucial role in the development, testing, and validation of quantum algorithms before they are implemented on actual quantum hardware. One of the primary advantages of quantum simulators is their accessibility. Unlike quantum computers, which are still relatively scarce and often require significant resources and expertise to operate, simulators can be run on conventional computers. This accessibility allows a broader range of researchers and developers to explore quantum algorithms and concepts without the need for physical quantum computing resources.

Quantum simulators offer a controlled environment for designing and refining quantum algorithms. They can simulate ideal quantum systems without the noise and error rates present in current quantum hardware, providing clearer insights into the theoretical performance of an algorithm. This is particularly useful for educational purposes and theoretical research, where understanding the principles of quantum computation and algorithm design is the main focus.

Problem

Even though some quantum computers have come onto the market, they represent prototypes that need to be more scalable and sufficient to test complex quantum algorithms. Quantum computer prototypes are currently very small and cannot overcome classical com-

puters in terms of exceeding capacity. For example, Atom Computing has 1180 qubits in the quantum circuit model, and IBM Osprey has 433 qubits. In the adiabatic model, the D-Wave 2000Q has 2000 qubits. These quantum computers represent prototypes that are not scalable and sufficient to test complex quantum algorithms. Constructing a full-scale quantum computer comprising millions of qubits is a longer-term prospect. Programming this prototype still lacks the compiler support modern programming languages enjoy today. Programmers of this machine must design low-level circuits; in particular, they must map logical qubits into physical qubits that must obey connectivity constraints. This task resembles the early days of programming, in which software was built in machine languages [20].

The capability to simulate quantum algorithms at this level is essential to learning how a quantum computer will physically operate, how the software can work, and what sort of problems it can solve. However, it's important to note that quantum simulators have limitations. The complexity and resource requirements of simulating quantum systems increase exponentially with the number of qubits. Therefore, simulators on classical computers are constrained by the computational resources available and can only simulate relatively small quantum systems accurately. This limitation underscores the complementary nature of simulators and actual quantum hardware in the development of quantum computing. Despite these limitations, quantum simulators remain indispensable tools in the quantum computing ecosystem. They enable the preliminary testing and validation of algorithms, facilitate learning and experimentation in quantum computing, and help identify promising algorithms and approaches that could be further explored using real quantum hardware. As quantum computing continues to evolve, the role of simulators in algorithm development and education is likely to expand, bridging the gap between classical and quantum computing.

Simulation of quantum computing models in classical computers requires exponential time and involves highly complex memory management. The problem is that using conventional techniques to simulate an arbitrary quantum process significantly more prominent than any of the existing quantum prototypes would soon require considerable memory on a classical computer. For instance, to simulate a 50-qubit quantum state, the process would take about 16 petabytes [21] ($8Bytes(double) \times 2(real, imag) \times 2^{50} = 16PB$).

In the quantum circuit model, the transformation of a quantum state is performed using quantum gates, which are modeled as matrices in a software quantum simulator. To apply a quantum gate on a n-qubit state, we need to build a $2^n \times 2^n$ matrix. For instance, to apply a quantum gate to a 50-qubit register, the matrix needs 16 QB (1024^{10}) of RAM. Therefore, researchers try to reduce such challenges by proposing efficient simulators.

In the next section, we establish the objectives of this work, which are focused on investigating how to implement a novel strategy to run quantum simulations efficiently to reduce classical resource consumption.

Research Aim

To develop memory management strategies for quantum state representation and quantum gate operations. These strategies are designed to optimize memory utilization when handling qubit superposition states and quantum gate circuits, enabling efficient implementation of algorithms within software-based quantum computing simulators.

Research Objectives

Accomplish the research aim implies to fulfill the following specific objectives:

- To review the structure of recent quantum software simulators to identify their main characteristics and memory management techniques (section 1.4).
- To design a method for qubits and quantum gates representation in classical computers to reduce memory consumption (chapter 2).
- To incorporate the proposed method into a quantum software simulator and test the performance to evaluate its effectiveness in reducing memory consumption (chapter 3).
- To execute selected case study applications in the original and modified versions of simulators and real quantum computers to validate the results of the proposed method (chapter 4).

To accomplish the objectives of this work, several key tasks were undertaken. First, the fundamental elements of quantum computing required for this study were examined and are detailed in the theoretical framework (Chapter 1). This chapter also includes the state of the art, where leading quantum simulators were evaluated based on criteria and characteristics that establish them as benchmarks in the field. The memory management techniques employed by these simulators were analyzed, and their efficiency was assessed through simulations of the quantum Fourier transform. This algorithm was chosen for its advantages, such as scalability, the ability to compare results with analytical methods, and more.

The core of this work is presented in Chapter 2, titled Memory Management Strategies. This chapter introduces a series of approaches designed to achieve efficient memory management and reduce memory usage in the representation of quantum states. These states are stored in data structures called quantum registers and are transformed by applying quantum gates to these registers. The strategies explored in this chapter include:

- Elimination of less probable states (State Pruning): While this technique significantly reduces memory usage, it introduces substantial drawbacks to the precision of quantum algorithms, leading to notable errors in the results.

- Full-state vector representation: Although this approach requires more memory to store all states, optimizing data structures can save up to $2^{Qubits} \times 4$ Bytes of memory by avoiding storing states and only retaining their amplitudes.

- State vector compression: The two most common compression algorithms, lossy and lossless, were evaluated. Lossy compression was selected for its superior compression rate. Additionally, two popular compression libraries, SZ and ZFP, were analyzed. ZFP was chosen as the most suitable for this work, offering potential memory savings of up to 75%.

- Adoption of Doan Trieu's [22] strategy for applying quantum gates: This approach eliminates the need to construct the matrix $U = \underbrace{\mathbb{1}_2 \otimes ... \otimes \mathbb{1}_2}_{n-i-1} \otimes G \otimes \underbrace{\mathbb{1}_2 \otimes ... \otimes \mathbb{1}_2}_{i}$ to apply the quantum gate G to a quantum state, resulting in memory savings of $2^{Qubits} * 2^{Qubits} \times 2 \times 8$ Bytes.

- Parallelism techniques: Shared-memory and distributed-memory parallelism were implemented to enhance performance when applying memory management strategies, particularly in conjunction with data compression techniques.

Chapter 3 outlines the design and implementation of a prototype software quantum simulator developed to test the memory management strategies proposed in the previous chapter. The prototype, written in C++, includes only the essential elements of quantum computing needed for this work. It supports one-qubit quantum gates and controlled two-qubit quantum gates.

Chapter 4 presents the results of simulations using the proposed memory management strategies. The findings reveal the following:

- State pruning while reducing memory usage introduces significant drawbacks, including poor performance when managing a disordered quantum state vector.

- Using a full-state vector requires more memory but performs substantially better.

- Implementing shared-memory parallelism results in a significant acceleration in computational performance.

- Applying data compression achieves considerable memory savings but negatively impacts performance. This drawback is mitigated by combining amplitude vector compression with distributed-memory parallelism, permitting both memory efficiency and improved performance.

Finally, we present the conclusions of this work in the chapter six 5, where we summarizes the key findings of the research, highlighting the effectiveness of memory management strategies for quantum simulators. While reducing memory usage, state pruning introduced

significant accuracy drawbacks. The use of optimized data structures improved efficiency. Adopting ZFP compression achieved considerable memory savings, although it also introduced performance overhead, mitigated by distributed-memory parallelism. Techniques for efficient quantum gate application and parallel computing frameworks enhanced scalability and resource utilization. The study concludes that combining compressed full-state vector representation with distributed-memory parallelism best balances memory efficiency and computational performance.

Chapter 1

Theoretical Framework

Quantum Mechanics, also known as quantum theory, quantum physics, matrix mechanics, or the wave mechanical model is a theory that describes the world at the scale of the energy level of atoms and other subatomic particles. The early proposed theory about some physical phenomena that could not be explained using classical physics was formulated by Max Planck (black-body radiation problem, 1900) [23] and Albert Einstein (photoelectric effect, 1905) [24]. This theory was completed and reformulated by Erwin Schrödinger, Werner Heisenberg, Max Born, and others. Several mathematical formalisms use the wave function to provide information about the probability amplitude of a particle's position, momentum, and other physical properties (observables). Several postulates constitute the basis of that formalism; however, some authors consider the following list to be the main ones [25]:

- **State Space**: the state of an isolated physical system can be characterized by a vector (**state vector**) in a complex, separable Hilbert space of infinite dimension and with an inner product.
- **Evolution**: The temporal evolution of the state of an isolated physical system can be described by an *unitary transformation* (operator) that act on the state vector that describes the system. That is, the state $|\Psi\rangle$ of the system at time t_1 is related to the state $|\Psi'\rangle$ of the system at time t_2 by a unitary operator U, which depends only on the times t_1 and t_2 ($|\psi'\rangle = U\,|\psi\rangle$).
- **Measurement** Upon measurement, a quantum system collapses to an eigenstate of the measured observable. The probability of obtaining a specific eigenvalue is given by the square of the amplitude of the corresponding state in the system's wave function.
- **Observable Postulate** Physical quantities (observables) are represented by operators on the Hilbert space. The possible outcomes of measuring an observable correspond to the eigenvalues of its operator.

Information Theory studies information transmission, processing, extraction, and utilization. It is based on the studies of Claude Shannon (1948) [1], who worked on the ideas of Ralph Hartley (1928) [26] and Harry Nyquist. Hartley tried to define a measure of information as: [27].

$$I = k \log_b(s) \tag{1.1}$$

Where:

- I is the amount of information.
- s is the number of possible symbols or messages.
- k is a constant.
- b is the base of the logarithm (often 2 when measuring information in bits).

1.1 Quantum Information Theory

Quantum Information Theory is the study of the use of quantum mechanics theory to model information, process it, and transmit it when it is stored in quantum particles [19] (Figure 1.1). Quantum Information Theory includes quantum computing, quantum cryptography and quantum communications, among others. [28] [29] [25]. The concept of quantum information comes from classical information.

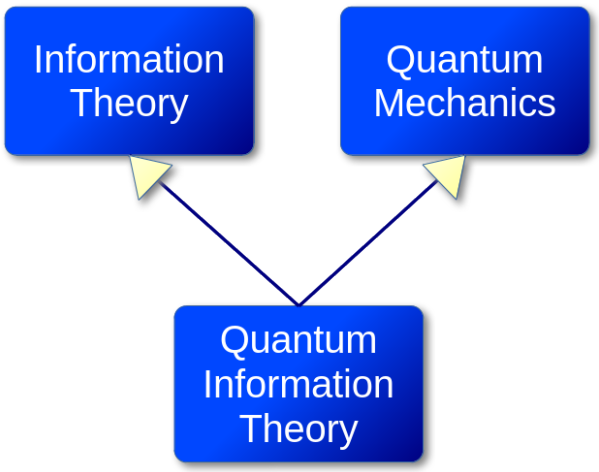


Figure 1.1: Quantum Information Theory Inheritance

1.1.1 Logical Qubit

The basic unit used in classical information is the bit, which can have two values: 0 or 1. The corresponding unit in quantum information is the qubit. A logical qubit is a unitary vector in a two-dimensional Hilbert space.

$$|\psi\rangle \in H \text{ where } ||\psi|| = \sqrt{\langle\psi|\psi\rangle} = 1 \text{ and } dimH = 2 \tag{1.2}$$

The prescribed pair of normalized and mutually orthogonal quantum states denoted using Dirac's notation $|0\rangle$ and $|1\rangle$ represents the Boolean states 0 and 1 [30].

The two states form a "computational basis" and any other (pure) state of the qubit can be written as a superposition $\alpha|0\rangle + \beta|1\rangle$ [31]. The bra/ket notation is independent of the basis and the order of the basis elements. Once this notation becomes familiar, it is easier to read and faster to use. Matrix notation is convenient for performing calculations and could be more familiar to readers; however, it always requires the choice of a basis and an ordering of that basis. Mathematically, the Dirac's notation (*ket*) is a shorthand notation for column vectors:

$$|0\rangle \equiv \begin{pmatrix}1\\0\end{pmatrix} : \ |1\rangle \equiv \begin{pmatrix}0\\1\end{pmatrix} \tag{1.3}$$

In Quantum Information processing, classical bits values of 0 and 1 are represented using distinguished states $|0\rangle$ and $|1\rangle$. With this representation we can compare directly bits and qubits, where bits can take on only two values, 0 and 1, while qubits not only can take on those previous values but, any superposition of them. See the section 1.1.1. The figure 1.2 depicts this comparison.

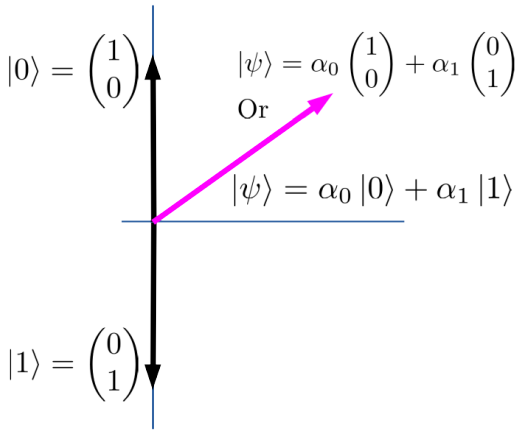


Figure 1.2: Bits vs Qubits

The Bloch sphere is commonly used to depict a qubit. Two angles represent the state, $0 < \theta < \pi$ and $0 <= \phi <= 2\pi$. Thus, the state $|\psi\rangle$ can be rewritten as:

$$|\psi\rangle = e^{i\gamma}\left(\cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle\right) \tag{1.4}$$

The vector from the origin to the point representing the state makes an angle of θ with the z-axis and its component in the x-y plane make an angle of ϕ with the x-axis. γ is the global phase, which does not affect the measurable probabilities of the quantum state (it only introduces a uniform phase shift to the whole state). The state $|0\rangle$ is the North Pole of the sphere, and the state $|1\rangle$ is the South Pole. The Bloch sphere is often helpful in illustrating how a quantum map changes the state of a qubit. Figure 1.3 depicts the Bloch sphere of a qubit.

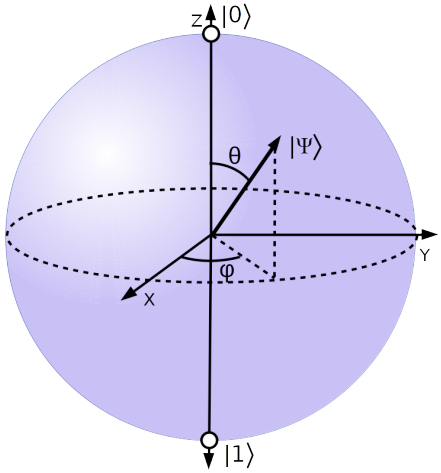


Figure 1.3: Bloch Sphere

Quantum State

A quantum state is simply something that encodes the state of a quantum system and provides a probability distribution for the outcomes of every possible measurement in the system.

Formally, a quantum state is a vector $|v\rangle$ representing a superposition of basis elements $\{|\beta_1\rangle, |\beta_2\rangle\}$ if it is a nontrivial linear combination of $|\beta_1\rangle$ and $|\beta_2\rangle$, if $|v\rangle = a_1|\beta_1\rangle + a_2|\beta_2\rangle$ where a_1 and a_2 are non-zero. For the term *superposition* to be meaningful, a basis must be specified [19].

Knowledge of the quantum state and the rules for the system's evolution in time determine everything that can be predicted about the system's behavior. A mixture of quantum states is again a quantum state.

The state of a bit is a particular case of a two-dimensional vector; that is to say, there are only two vectors in the whole two-dimensional vector space with real meaning; these are the two orthogonal vectors $|0\rangle$ and $|1\rangle$. Conversely, qubits do not suffer from this limitation. The state $|\psi\rangle$ associated with a qubit can be any unit vector in the two-dimensional vector space spanned by $|0\rangle$ and $|1\rangle$ over the complex numbers [32]. The general state of a qubit is:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle \tag{1.5}$$

Where α_0 and α_1 are two complex numbers constrained only by the requirement that $|\psi\rangle$, like $|0\rangle$ and $|1\rangle$, should be a unit vector in the complex vector space, in other words, only by the normalization condition:

$$|\alpha_0|^2 + |\alpha_1|^2 = 1 \tag{1.6}$$

The state of n qubits is spanned by the tensor product basis.

$$\begin{aligned} |0\rangle \otimes ... |0\rangle \otimes |0\rangle &= |0...00\rangle \\ |0\rangle \otimes ... |0\rangle \otimes |1\rangle &= |0...01\rangle \\ &. \\ &. \\ |1\rangle \otimes ... |1\rangle \otimes |1\rangle &= |1...11\rangle \end{aligned} \tag{1.7}$$

The general equation of a n-qubit state is

$$|\Psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \tag{1.8}$$

As we can see, a single complex number can specify a single-qubit state, so n complex numbers can specify any tensor product of n individual single-qubit states. Three-valued units are called **qutrits**, and n-valued units are called **qudits**. Since qudits can be modeled using multiple qubits, a model of quantum information based on qudits has the same computational power as one based on qubits. For this reason, qudits are not commonly utilized.

Superposition

Quantum states allow the system to be in a few states simultaneously; this is called *superposition* [33].

Quantum bits are not constrained to be wholly 0 or wholly 1 at a given instant. In quantum physics, if a quantum system can be found to be in one of a discrete set of states, which we'll write as $|0\rangle$ or $|1\rangle$, then, whenever it is not being observed it may also exist in a superposition, or a blend of those states simultaneously [34].

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \equiv \begin{pmatrix}\alpha\\ \beta\end{pmatrix} \tag{1.9}$$

where α and β are complex numbers having the following property:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{1.10}$$

The coefficients α and β are called the **amplitude** of component $|0\rangle$ and component $|1\rangle$ respectively. The requirement of the equation 1.10 is to ensure the qubit is normalized. This guarantees that when we read the qubit, it will be found, with probability of $|\alpha|^2$ to be in the state $|0\rangle$ or, with probability of $|\beta|^2$ to be in the state $|1\rangle$.

The superposition property in quantum mechanics is fundamentally different from classical probabilism. In classical systems, probabilities are always non-negative real numbers and describe the likelihood of specific outcomes. In contrast, quantum mechanics uses probability amplitudes, which can be positive, negative, or complex numbers. The square of the absolute value of these amplitudes gives the probability of observing a particular outcome upon measurement [35].

How Big are Quantum States?

In Classical Information Theory, the amount of information a specific state contains using n bits is n; there will be only one combination of n 0s and 1s. In Quantum Information Theory, a state of n qubits will be a union of all possible combinations of n 0s and 1s; that is, the information's size is 2^n. For example, if we are using 3 bits, we will have just one of the 2^3 possibilities whose length is 3, for instance, 010. If we use 3 qubits, we will have not only one but all combinations: 001, 010, 011... 111, each multiplied by the corresponding amplitude. If we increase the number of bits by one, the size will be $n + 1$, but if we increase the number of qubits, we get double the size, that is, 2^{n+1}.

1.1.2 Measurement of Single Qubit Quantum State

We now have a mathematical model with which to describe quantum bits. In addition, we need a mathematical model for measuring devices and their interaction with quantum bits. According to postulate 3 of quantum mechanics, measuring a quantum state changes it. If a state $|v\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ is measured and the outcome is $|0\rangle$, then the state $|v\rangle$ changes to $|0\rangle$. A second measurement for the same basis will return $|0\rangle$ with probability 1. To understand this, it is necessary to think of a superposition $|v\rangle$ as a state that could be in both state $|0\rangle$ and state $|1\rangle$ at the same time, that is to say, the quantum state $|v\rangle$ is a combination of $|0\rangle$ and $|1\rangle$ in similar proportions but with different amplitudes. For example, $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ and $\frac{1}{\sqrt{2}}(|0\rangle + i\,|1\rangle)$, represent different states that behave differently in many situations.

Because a qubit can take on any one of infinitely many states, a single qubit could store lots of classical information. However, the properties of quantum measurement severely restrict the amount of data that can be extracted from a qubit. Measuring is the only way to obtain information about a quantum bit, and any measurement results in one of only two states, the two basis states associated with the measuring device; thus, a single measurement yields, at most, a single classical bit of information.

A fundamental fact about this measurement process (measurement in the computational basis) is that it disturbs the quantum state $|v\rangle$. The effect of the measurement is that the new state is precisely the outcome of the measurement. The final result of this process is a classical bit, which can be 0 or 1. The information about the amplitudes of the state $|v\rangle$ is lost after the measurement. Therefore, it is impossible to determine the original state from any sequence of measurements [19].

1.1.3 Entanglement

The "entanglement" describes a correlation between different parts of a quantum system that surpasses anything classically possible. It happens when the subsystems interact so that the resulting state of the whole system can not be expressed as the tensor product of the states of its parts. When a quantum system is in such a tangled state, the actions performed in one subsystem will have a side effect in another subsystem, even if it does not act directly on that subsystem [34]. States that cannot be written as the tensor product of n single-qubit states are called **entangled states**. If we can write the tensor product of those states, they are said to be **separate states** [19].

Let's see an example. If we have two quantum states represented by the following two qubits

$$\begin{aligned} 3/5\,|0\rangle + 4/5\,|1\rangle \\ 1/\sqrt{2}\,|0\rangle - 1/\sqrt{2}\,|1\rangle \end{aligned} \tag{1.11}$$

Then, the joint state of the two qubits is

$$\frac{3}{5\sqrt{2}}\,|00\rangle - \frac{3}{5\sqrt{2}}\,|01\rangle + \frac{4}{5\sqrt{2}}\,|10\rangle - \frac{4}{5\sqrt{2}}\,|11\rangle \tag{1.12}$$

Every qubit must be in some state like $\alpha\,|0\rangle + \beta\,|1\rangle$, and the state of two qubits must be the product. However, there are some states that cannot be decomposed in the form of a state of the first qubit and that of the second qubit. For example:

$$\left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{1.13}$$

Such a state (one of the Bell's bases) is entangled because we cannot determine the state of each qubit separately. That is to say, this state cannot be decomposed because it is impossible to find a_1, a_2, b_1, b_2 such that:

$$(a_1 |0\rangle + b_1 |1\rangle) \otimes (a_2 |0\rangle + b_2 |1\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{1.14}$$

since

$$(a_1 |0\rangle + b_1 |1\rangle) \otimes (a_2 |0\rangle + b_2 |1\rangle) = a_1 a_2 |00\rangle + a_1 b_2 |01\rangle + b_1 a_2 |10\rangle + b_1 b_2 |11\rangle \tag{1.15}$$

If $a_1 b_2 = 0$ implies that either $a_1 a_2 = 0$ or $b_1 b_2 = 0$. Therefore, this cannot be expressed in separate states.

At the end of this section, we can say the state of the qubits has as much to do with the relationship of the two qubits as it does with their states. Many different types of information can be accommodated within quantum mechanics, including classical information, coherent quantum information, and entanglement. Exploring the wide variety of capabilities allowed by these types of data is the subject of quantum information theory.

1.2 Quantum Computing

Quantum computing represents a fundamentally new approach to information processing, leveraging the unique properties of quantum systems such as superposition and entanglement. Unlike classical bits, which can only represent 0 or 1, quantum bits (qubits) can exist in superpositions of both states, enabling quantum computers to process vast amounts of information simultaneously. This capacity opens up possibilities for solving complex problems in areas such as cryptography, optimization, and material science, where classical approaches fall short.

Quantum computing is still a relatively new discipline that we can compare with the early days of classic computers in the 1950s. Like the manual programming of a classic computer using punched cards or assembler, the user needs to specify a quantum algorithm as a sequence of fundamental quantum logic gates in today's quantum computers. Therefore, it is necessary to perform several steps in different layers of abstraction to implement a quantum algorithm in real quantum hardware [36].

One could use two approaches to incorporate the quantum mechanical effects into computing machinery. The first one is to suppress the quantum effects and preserve a semblance of classical computing even though the computational elements are very small. The second approach is a deeper integration of the quantum effects and trying to find clever ways to enhance and sustain them to achieve old computational goals in new ways. Quantum computing attempts to pursue the latter strategy by leveraging quantum effects. [34].

After this, we can define **Quantum computing** as the area of study focused on developing computer technology based on the principles of quantum-mechanical phenomena, which explains the nature and behavior of energy and matter on the quantum (atomic and subatomic) level, such as superposition and entanglement. In other words, quantum

computing is the research area of computer science that applies the principles of quantum mechanics to how to perform calculations.

Quantum computing is not about changing the physical substrate on which computation is done from classical to quantum, but rather changing the notion of computation itself. The change starts at the most basic level: the fundamental unit of computation is no longer the bit but rather the quantum bit or qubit. Placing computation on a quantum mechanical foundation led to the discovery of faster algorithms, novel cryptographic mechanisms, and improved communication protocols [19].

1.2.1 Building Blocks

The most common computing model today is based on bits that can take on only two values, 0 and 1. The transistor is the electronic device used by excellence to implement 0s and 1s in any modern computer.

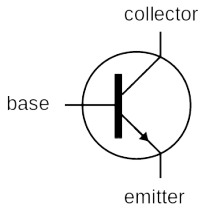


Figure 1.4: Transistor

A transistor is an electronic device whose operation can be considered as a switch that is electrically operated. A current is entered through the collector; this will pass to the emitter only if there is a current in the base. The absence of current in the base cut the output at the emitter. Figure 1.4 depicts the diagram of a transistor.

Physical Qubit

Just as the transistor is the fundamental mechanism for implementing bits and the logic gates that transform the state of those bits, a fundamental device is required to implement the qubits. A physical Qubit is a two-state (or two-level) quantum system [1]. To comply with the basic postulates of quantum computing, for example, superposition and entanglement, it is necessary to use elements capable of satisfying those properties. These mechanisms must have the capacity to represent the qubits, operate with them, define the initial value of the qubits at the beginning of the calculation, and measure the final result. That is, the qubit must correspond to some physical property of a particle, and all manipulations on the qubits must be able to be done by altering said property in some way. In

[1] `https://en.wikipedia.org/wiki/Quantum_system`

general, the qubits can be implemented by quantum mechanical objects like atoms. Several strategies are used to implement the qubit; basically, any two-level quantum mechanical system can be used as a qubit. However, it is impossible to determine which is the most promising technology because much of the "cutting-edge" research in quantum computing is proprietary or classified. Among the most popular options we have [37]:

Technique	**Physical Support**	$\|0\rangle$	$\|1\rangle$
Polarization Encoding	Photons	horizontal	vertical
Electronic Spin	Electrons	up	down
Superconducting flux qubit	Josephson junction	Clockwise current	Counterclockwise current
Superconducting phase qubit	Josephson junction	Ground state	First excited state
Superconducting charge qubit	Josephson junction	Uncharged superconducting island	Charged superconducting island

Table 1.1: Qubits Implementations

The IBM Q system uses transmon superconducting qubits based on the Josephson junction. The Josephson effect occurs when two superconductors are separated by a layer of an insulating medium or a non-superconducting metal of a few nanometers; the Cooper pairs can cross the barrier by tunneling, an effect characteristic of quantum mechanics. The Cooper pairs are two electrons linked because, in the superconducting state, both particles behave as if they were attracted. Figure 1.5 shows a Josephson junction.

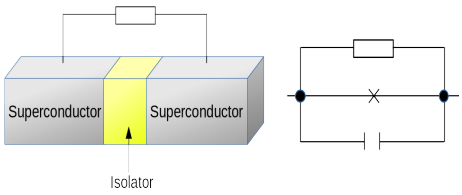


Figure 1.5: Josephson junction

Like any quantum system, qubits have all these properties (superposition, entanglement, among others) and the advantages they offer.

1.2.2 Quantum Computing Models

A quantum computing model describes the different scientific approaches to formalize the transformations over inputs to compute outputs using quantum resources. The essential elements in which the computation is decomposed define a model. [29] [38]. The five main models of practical importance are:

- **Quantum Gate Array or Quantum Circuit**: The computation is decomposed into sequence of few qubit quantum gates. This is the best well-known model of quantum computation.
- **One-way quantum computer**: The computation is decomposed into sequence of one-qubit measurements applied to a highly entangled initial state or cluster state.
- **Adiabatic quantum computer**: This model of quantum computation is motivated by ideas in quantum many-body theory. The computation is decomposed into a slow continuous transformation of an initial Hamiltonian into a final Hamiltonian, whose ground states contain the solution.
- **Topological quantum computer**: The computation is decomposed into the braiding of anyons in a 2D lattice.
- **Quantum Turing Machine**: A quantum Turing machine, also known as universal quantum computer, is an abstract machine used to model a quantum computer proposed by David Deutsch [9]

In this study, we will work with the quantum circuit model because of its popularity.

1.2.3 Quantum Gates

In order to understand quantum computation, we must understand which sorts of transformations nature allows and which it does not. This section describes the transformations of a closed quantum system, which map the quantum system's state space to itself.

A quantum gate is a basic quantum circuit operating on a small number of qubits. It is the building block of quantum circuits like classical logic gates for conventional digital circuits. The most common quantum gates operate on spaces of one or two qubits, just like the standard classical logic gates operate on one or two bits. The first significant difference from classical computing is the information representation; quantum bits (qubits) are used instead of bits. The type of the computer, quantum or classical, will depend on how the information is represented and manipulated in a quantum or classical way. Classical computing relies on principles expressed by Boolean algebra, and the state of bits is changed using the traditional 7-mode logic gate principle: AND, OR, XOR, NOT, NAND, NOR, and XNOR. By contrast, quantum computing can work with a two-mode logic gate XOR. Conventionally, a logic gate is thought of as a physical device that takes one or more Boolean values as inputs and returns a single Boolean value as output. The Boolean values (FALSE and TRUE) are often synonymously with the bit values 0 and 1, respectively [34]. Figure 1.6 depicts the designs of classical logic gates based on transistors.

The main difference between classical and quantum logic gates is that the former manipulates the classical bit values, 0 or 1, and the latter can manipulate arbitrary multi-partite quantum states, including arbitrary superpositions of computational base states, which

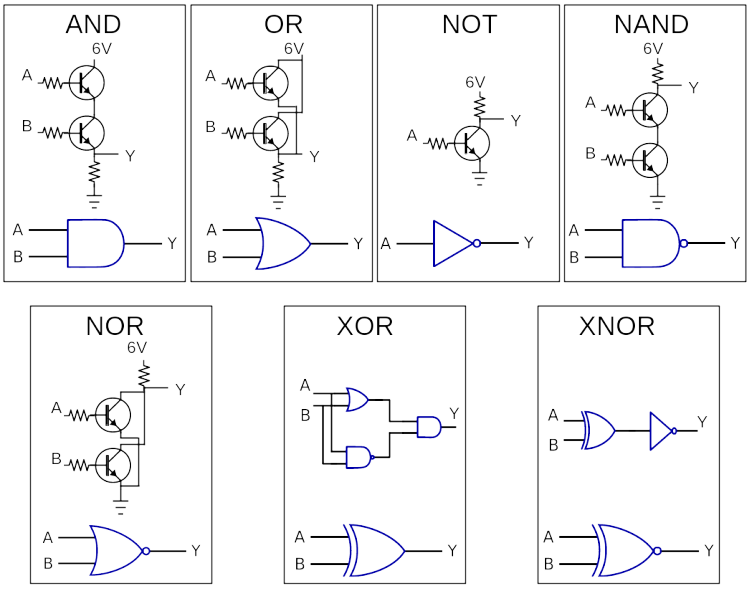


Figure 1.6: Classical Gates

are often interlaced. Thus, the logical gates of quantum computing are considerably more varied than the logical gates of classical computation.

Reversibility

The reversibility is a property of some operations that consists of obtaining unique inputs for all outputs of said operations. Reversibility is one of the most valuable mechanisms in quantum computing. Reversible operations change the initial state of the qubits into its final form using only processes whose action can be inverted. There is only one irreversible component to the operation of a quantum computer, the measurement, which is the only way to extract useful information from the qubits after their state has acquired its final form [32]. In a reversible operation, every final state arises from a unique initial state. Before studying quantum gates, let us briefly describe this property of the classical logic gates. The AND gate is logically irreversible because if the output is 0, we cannot tell whether the input values were 00, 01, or 10. It "erases" some information when it acts whenever the output from the AND gate is 0. Similarly, the OR gate is irreversible because when its output is 1 it is impossible to say whether the inputs were 01, 10, or 11. The simplest example of a reversible logic gate is the NOT gate; when the output is 1, the only possible input is 0, and vice-versa.

Unitary Transformations

Nature does not allow arbitrary transformations of a quantum system. Therefore, these transformations must satisfy the properties related to quantum measurement and quantum superposition. The transformations must be linear transformations of the vector space associated with the state space. Linearity means that for any quantum transformation U

$$U(a_1 |\psi_1\rangle + ... + a_k |\psi_k\rangle) = a_1 U |\psi_1\rangle + ... + a_k U |\psi_k\rangle \tag{1.16}$$

On any superposition $a_1 |\psi_1\rangle + ... + a_k |\psi_k\rangle$ [19]. That means unit length vectors must go to unit length vectors, which implies that orthogonal subspaces go to orthogonal subspaces. These properties ensure that measuring and applying a transform to the outcome gives the same result as first applying the transform and then measuring on the transformed basis. This affirmation is valid only if U preserves the inner product; for any $|\psi\rangle$ and $|\phi\rangle$, the inner product of their images, $U |\psi\rangle$ and $U |\phi\rangle$, must be the same as the inner product between $|\psi\rangle$ and $|\phi\rangle$, that is to say:

$$\langle\phi| U^\dagger U |\psi\rangle = \langle\phi|\psi\rangle \tag{1.17}$$

Mathematically, it can be demonstrated that this condition is valid for all $|\psi\rangle$ and $U |\phi\rangle$ only if $U^\dagger U = I$. That is to say, for any quantum transformation U, its adjoint $U^\dagger$ must be equal to its inverse, precisely the condition, $U^\dagger = U^-1$, for a linear transformation to be **unitary**. The unitarity condition ensures that the operator does not violate any general principles of quantum theory.

The No-Cloning Principle

A very important consequence of the unitary condition is tha unknown quantum states cannot be copied or cloned. Suppose U is a unitary transformation that clones, such that

$$U(|a\rangle |0\rangle) = |a\rangle |a\rangle \tag{1.18}$$

for all quantum states $|a\rangle$. Let $|a\rangle$ and $|b\rangle$ be two orthogonal quantum states. Let U a transformation that clones a state, such that

$$\begin{aligned} U(|a\rangle |0\rangle) &= |a\rangle |a\rangle \\ U(|b\rangle |0\rangle) &= |b\rangle |b\rangle \end{aligned} \tag{1.19}$$

Now, let consider

$$|c\rangle = \frac{1}{\sqrt{2}}(|a\rangle + |b\rangle) \tag{1.20}$$

Applying the linearity we have

$$\begin{aligned} U(|c\rangle |0\rangle) =& \frac{1}{\sqrt{2}}(U(|a\rangle |0\rangle) + U(|b\rangle |0\rangle)) \\ =& \frac{1}{\sqrt{2}}(|a\rangle |a\rangle + |b\rangle |b\rangle) \end{aligned} \tag{1.21}$$

But if U is a cloning transformation then

$$U(|c\rangle |0\rangle) = |c\rangle |c\rangle = 1/2(|a\rangle |a\rangle + |a\rangle |b\rangle + |b\rangle |a\rangle + |b\rangle |b\rangle) \tag{1.22}$$

Which is not equal to the second part of the equation 1.21. Thus, no unitary operation can reliably clone all quantum states.

Quantum Gates Representation

Quantum gates are based on reversible logic. Because a quantum gate changes the state of a qubit, which is ruled by the quantum mechanics principles, a quantum gate must be implemented physically as the quantum mechanical evolution of an isolated quantum system [34]. The transformation it achieves is governed by Schrödinger's equation:

$$\frac{i\hbar\partial |\psi\rangle}{\partial t} = \mathcal{H} |\psi\rangle \tag{1.23}$$

Where, $\mathcal{H}$ is the Hamiltonian, specifying the physical fields and forces at work.

Unitary matrices represent quantum logic gates. A gate that acts on n qubits is represented by a $2^n \times 2^n$ unitary matrix. If a measurement process is executed after applying a gate on a quantum state, the possible outcomes are the base vectors. The number of qubits in the input and output of the gate must be equal [39].

The unitary matrices describing quantum gates are related to the physical processes by which they are achieved via the equation:

$$U = e^{\frac{-i\mathcal{H}t}{\hbar}} \tag{1.24}$$

Here $\mathcal{H}$ is the Hamiltonian, which specifies the interactions in the physical system. The quantum mechanical evolution induced by this equation is unitary as long as no measurements are made and no interactions with the environment occur. In this particular case, from a initial state $|\psi(0)\rangle$, the quantum system will evolve, in time t, into the state

$$|\psi(t)\rangle = e^{\frac{-i\mathcal{H}t}{\hbar}} |\psi(0)\rangle = U |\psi(0)\rangle \tag{1.25}$$

Where U is some unitary matrix. Thus, the evolution in time t of an isolated quantum system is described by a unitary transformation of an initial state $|\psi(0)\rangle$ to a final state as follows:

$$|\psi(t)\rangle = U\,|\psi(0)\rangle \tag{1.26}$$

This means that a quantum logic gate acting on an isolated quantum computer, will transform that state unitarily up until the point at which an observation is made. Therefore, quantum gates are described, mathematically, by unitary matrices, and their action is always logically reversible [34]. For example, let $|ab\rangle$ be the vector representation of two qubits

$$|ab\rangle = |a\rangle \otimes |b\rangle = a_{00}\,|00\rangle + a_{01}\,|01\rangle + a_{10}\,|10\rangle + a_{11}\,|11\rangle \equiv \begin{pmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{pmatrix} \tag{1.27}$$

The action of the gate on a specific quantum state is found by multiplying the vector $|ab\rangle$, which represents the state, by the matrix U representing the gate.

$$U\,|ab\rangle \tag{1.28}$$

Properties of the Quantum Gates Derived from the Unitarity

The fundamental properties of quantum gates come from how unitary matrices describe them. A matrix, U, is unitary if and only if its inverse equals its conjugate transpose, if and only if $U^{-1} = U^{\dagger}$. If U is unitary, the following facts are valid:

- $U^{\dagger}$ is unitary.
- U^{-1} is unitary.
- $U^{-1} = U^{\dagger}$
- $U^{\dagger}U = \mathbb{1}$
- $|det(U)| = 1$
- The columns or rows of U form an orthonormal set of vectors.
- For a fixed column, $\sum_{i=1}^{2^n} |U_{ij}|^2 = 1$
- For a fixed row, $\sum_{j=1}^{2^n} |U_{ij}|^2 = 1$
- $U = e^{\frac{-i\mathcal{H}t}{\hbar}}$ where $\mathcal{H}$ is an hermitian matrix such that $\mathcal{H} = \mathcal{H}^{\dagger}$

The property $U^{\dagger}U = \mathbb{1}$ ensures that we can always undo a quantum gate, that is to say, a quantum gate is logically reversible [34].

Universal Quantum Gates

A set of gates, S, is *universal* if we can achieve any essential operation in a circuit that uses solely gates from S. That is to say, we can express any other unitary operation as a finite sequence of gates from the set. Quantum gates are very different from classical gates in terms of universality. While in classical reversible computing, there is no 2-bit gate that is both reversible and universal, in quantum computing, almost all 2-qubit gates are universal [34].

For example, TOFFOLI and FREDKIN gates are universal for classical reversible computing. A single gate set of universal quantum gates can also be formulated using the three-qubit Deutsch gate $D(\theta)$ [40].

A list of quantum gates can be found in the appendix C.

1.2.4 Quantum Algorithms

A quantum algorithm is made in one of the quantum computing models; quantum circuits are the most used model. A classical (or non-quantum) algorithm is a finite sequence of instructions or a step-by-step procedure for solving a problem, where each step or instruction can be performed on a classical computer. Similarly, a quantum algorithm is a step-by-step procedure where each step can be performed on a quantum computer. Although all classical algorithms can also be performed on a quantum computer, the term quantum algorithm is generally used for those algorithms that incorporate some essential feature of quantum computing, such as superposition or entanglement [41].

David Deutsch is the precursor of the quantum algorithms field. His work went from quantum information to quantum computation [42], raising the question of whether there is a quantum extension of the Church-Turing idea that any computation that runs on a classic computer can be efficiently simulated on a universal Turing machine [43]. The quantum Strong Church-Turing thesis states that the quantum circuit model can efficiently simulate any realistic model of computation [44]. Deutsch proposes that in order to see the Church-Turing hypothesis as a physical principle, not only must computer science be turned into a branch of physics, but also part of experimental physics into a branch of computer science [45].

Several quantum computation models have been developed (see the section 1.2.2). However, they can be efficiently simulated by quantum circuits. Quantum circuits are very similar to most of the approaches currently used to try to build scalable quantum computers.

Contrary to the popular belief that quantum computers have few applications, the field of quantum algorithms has become a sufficiently large area of study. We can see how websites like "Quantum Algorithms Zoo" [2] [46] cite almost 400 articles in this area.

[2] `https://quantumalgorithmzoo.org/`

P	A deterministic classical computer can solve it in polynomial time
BPP	A probabilistic classical computer can solve it in polynomial time
BQP	A quantum computer can solve it in polynomial time
NP	A deterministic classical computer can check the solution in polynomial time
QMA	A quantum computer can check the solution in polynomial time

Table 1.2: Computational Complexity Classes of Algorithms

When referring to an algorithm, the **computational complexity**, or just complexity, is a measure of the resources used by the algorithm, usually measured as a function of the algorithm's input size. The complexity for the input size n is taken as the cost of the algorithm in a more unfavorable case entry for the size n problem. When referring to a problem, the complexity is the minimum amount of resources required by any algorithm to solve the problem [44].

In the computational complexity theory, asymptotic scales of complexity measures such as execution time or problem size are generally considered. In classical and quantum computing, we measure the execution time using the number of elementary operations an algorithm uses. In the case of quantum computing, we measure this using the quantum circuit model, where a quantum circuit is a sequence of quantum operations called quantum gates, each applied to a small number of qubits. To compare the performance of the algorithms, the notation $O(f(n))$ of the computing style is used, which is interpreted as "asymptotically delimited by $f(n)$" [47]. In these cases, it is convenient to use the basic ideas of the computational complexity theory [48], especially the notion of complexity classes, which are groupings of problems by difficulty. Table 1.2 contains the informal descriptions of some essential complexity classes. If a problem is said to be complete for a complexity class, it is one of the "most difficult" problems within that class.

Three classes of quantum algorithms have clear advantages over known classical algorithms.

- Algorithms based on quantum versions of the Fourier transform are used in classical algorithms. Some algorithms in this category are Deutsch–Jozsa algorithm and Shor's algorithms for factoring.
- Quantum search algorithms.
- Quantum simulation. A quantum computer is used to simulate a quantum system.

Richard Feynman pointed out that simulating quantum systems on a classical computer is complicated. Besides, other physicists believe that all aspects of the world around us, including classical logic circuits, can ultimately be explained using quantum mechanics. However, quantum circuits cannot be used to directly simulate classical circuits because unitary quantum logic gates are inherently reversible, whereas many classical logic gates,

such as the NAND gate, are inherently irreversible [25]. To overcome this obstacle, we can replace the original classical circuit with an equivalent circuit containing only reversible gates like *Toffoli* gate.

Quantum Parallelism

Parallel algorithms are designed to perform multiple calculations or tasks simultaneously by breaking down a problem into separate pieces that can be addressed concurrently. Each piece is processed by a different processor or core, with the combined results producing the overall solution. This approach contrasts traditional sequential algorithms, where tasks are executed one after the other. There are two main parallel programming models: shared memory and distributed memory. The first describes a computer architecture where all processors directly access shared physical memory. The second refers to network-based memory access for non-common physical memory.

The key benefit of parallel algorithms is their efficiency in drastically reducing the time needed to tackle complex issues by distributing tasks across multiple processors. This proves particularly valuable in fields requiring the handling of extensive data sets or undertaking computationally intensive tasks, such as conducting scientific simulations, analyzing vast quantities of data, and processing images. In quantum computing, parallelism is implicit.

One of the main features of quantum computing is taking advantage of quantum mechanics effects, like superposition and entanglement, to speed up calculations. In 1985, Deutsch [42] found a computational problem that could be solved on a quantum computer in a manner that is impossible classically. In 1992, Deutsch and Jozsa [49] simplified and extended the earlier result [50].

The modern formulation of the problem is the following. Suppose a classical algorithm that computes some function $f : \pm 1 \to f : \pm 1$ [43]. There are exactly four such functions:

$$\begin{aligned} f_1(x) &= x \\ f_2(x) &= -x \\ f_3(x) &= +1 \\ f_4(x) &= -1 \end{aligned} \tag{1.29}$$

If we evaluate $f(-1)$ and $f(+1)$, we obtain two bits of classical information and know which of our four functions; this requires two evaluations of the function. All those functions can be categorized as follows:

- Balanced: for example: $f(-1) + f(+1) = 0$
- Unbalanced: or constant.

Classically, it is necessary to acquire a classic bit of information to find out what class the function is in, which requires two function evaluations. Quantum mechanically, we can determine the class with only a single evaluation or "measurement." The trick is to put x into a superposition of $+1$ and -1; then, a single function evaluation can determine the class. However, it is impossible to determine what function in the class is because the measurement returns only one bit of classical information. This idea is the starting point to investigate other quantum algorithms.

Hybrid Approach

Current quantum computers cannot execute many principal quantum algorithms with asymptotic speedups, such as the Shor algorithm, for practical size problems; this is because the small number of qubits and the limited number of gates that can be executed before the accumulation of errors makes the output useless due to decoherence. Several quantum-classical algorithms have been developed to overcome this inconvenience. The general method is to decompose the problem statically or dynamically and solve the problematic subproblems on QPU, and combine them on the CPU to obtain a global solution. One of the most famous hybrid algorithms is the Variational Quantum Eigensolver (QVE). This algorithm combines a small QPU and CPU to find a Hamiltonian problem's ground state. The trial state (ansatz) is prepared by applying a series of parameterized gates on the QPU, and its energy is measured.

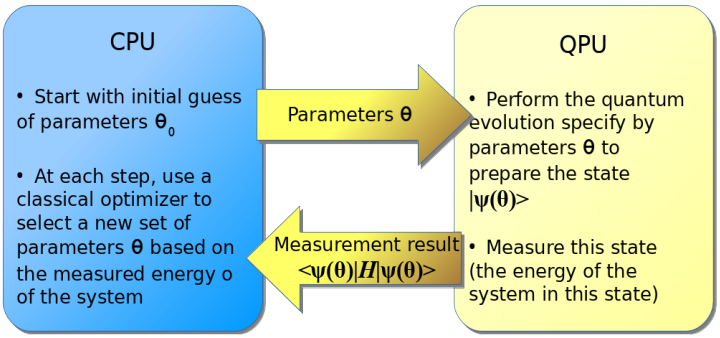


Figure 1.7: The general outline of variational hybrid algorithms

This process is outlined in the figure 1.7. The advantage of this kind of algorithm is that the trial state can be chosen. Therefore, the required gates are small enough to run feasibly on a small QPU or noisy intermediate-scale quantum computer (NISQ). Shaydulin et al. [11] describe in their work a hybrid approach for solving practical size problems about quantum local search (QLS) [51] [52]. The QLS is inspired by the success of numerous local-search heuristics applied to various computational complex problems like traveling

salesman problems.

The appendix B describes several quantum algorithms.

1.3 Quantum Computing Simulators

A quantum simulator is an object able to execute quantum computations. They can be classified into two categories [53]:

- **A Quantum System** that can perform very specific quantum computations
- **Software Packages** that can reproduce most of the fundamental aspects of a general universal quantum computer on a general-purpose classical computer.

In this work, we are interested in the latter, which are powerful tools to understand and develop the universal quantum computer.

1.3.1 Basis of Quantum Computing Simulation

This section shows how to represent the fundamental concepts of Quantum Computing on classical computers [21].

Single Qubit Representation

To store $|\psi\rangle$ on a classical computer, it is more suitable to use two orthonormal vectors to represent $|0\rangle$ and $|1\rangle$.

$$|\psi\rangle = \alpha_0 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \alpha_1 \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} \tag{1.30}$$

Operations On Single Qubits

A complex unitary matrix represents any operation on the qubit of the equation 1.30. For example, applying a NOT gate (X gate) is equivalent to executing a matrix-vector multiplication.

$$X|\psi\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_0 \end{pmatrix} = \alpha_1 |0\rangle + \alpha_0 |1\rangle \tag{1.31}$$

The table A.1 describes other single-qubit quantum gates.

Two Qubits Representation

The state of two qubits system $|\phi\rangle$ is represented by four complex coefficients which provide the contribution of all possible classical states of two bits.

$$|\phi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle = \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} \tag{1.32}$$

Operations On Two Qubits

Complex unitary matrices represent the operations that act on the entire state of two qubits of dimension 4×4, for example, the controlled Z gate or CZ.

$$CZ \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{1.33}$$

To build the 4×4 matrix acting on the whole system, applying a single-qubit operation to just one qubit, one performs a Kronecker product with a 2×2 identity matrix. Applying an X-gate to the first qubit (with bit-index 0) can be achieved as follows [21]:

$$\begin{aligned} X_0|\phi\rangle &= \mathbb{1}_2 \otimes X|\phi\rangle \\ &= \alpha_{01}|00\rangle + \alpha_{00}|01\rangle + \alpha_{11}|10\rangle + \alpha_{10}|11\rangle \\ &= \alpha_{00}|01\rangle + \alpha_{01}|00\rangle + \alpha_{10}|11\rangle + \alpha_{11}|10\rangle \end{aligned} \tag{1.34}$$

n-Qubits Representation

A n-qubits quantum state is represented by the equation 1.8. Its expanded form is as follows.

$$|\Psi\rangle = \alpha_0|0...00\rangle + \alpha_1|0...01\rangle + ... + \alpha_{2^n-1}|1...11\rangle \tag{1.35}$$

We need an array of complex numbers to model this state in a quantum simulator. This is also known as a **quantum register**. For instance, for double-precision values, just storing the state vector for 50 qubits would already require 16 petabytes of memory ($8Bytes(double) \times 2(real, imag) \times 2^{50} = 16PB$).

Operations On n-Qubits

In the Quantum Circuits model, the fundamental transformation of a quantum state is carried out using quantum gates, which are the basic components of quantum circuits. Quantum gates are analogous to classical logic gates but operate on qubits instead of classical bits.

There are many different types of quantum gates, each performing a specific transformation into a quantum state. For example, the Hadamard gate is a commonly used quantum gate that transforms a qubit from the standard basis to the superposition basis. Another example is the Pauli-X gate, which is a quantum NOT gate that inverts the state of a qubit from 0 to 1 or vice versa. Furthermore, the most used gates are single-qubit and two-qubit gates (controlled gates).

Quantum gates can be combined to create more complex transformations in a quantum state. For example, the quantum Fourier transform (QFT), a fundamental operation in quantum computing, can be constructed using a combination of Hadamard and phase change gates. QFT is used in many quantum algorithms, including Shor's algorithm.

To transform the state of the equation 1.35, we need $2^n \times 2^n$ unitary matrices. Applying a single-qubit gate G to the i-th qubit of an n-qubit quantum state amounts to multiplying the state vector of coefficients α_i by the matrix.

$$\underbrace{\mathbb{1}_2 \otimes ... \otimes \mathbb{1}_2}_{n-i-1} \otimes G \otimes \underbrace{\mathbb{1}_2 \otimes ... \otimes \mathbb{1}_2}_{i} \tag{1.36}$$

The tensor product of the identity matrix is carried out until reaching the position of the target qubit, then the tensor product is made using the gate, and the tensor product is continued using the identity matrix until the proper size is obtained. In the end, a matrix of dimension $2^n \times 2^n$ is obtained. This is a complex sparse matrix-vector multiplication.

$$U = \mathbb{1}_2^{\otimes n-i-1} \otimes G \otimes \mathbb{1}_2^{\otimes i} \quad where \quad G = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} \tag{1.37}$$

Thus, applying the gate G to the i-th qubit of the state $|\psi\rangle$ is

$$U|\psi\rangle = \mathbb{1}_2^{\otimes n-i-1} \otimes G \otimes \mathbb{1}_2^{\otimes i} |\psi\rangle \tag{1.38}$$

U is a complex sparse matrix of dimension $2^n \times 2^n$.

Single Qubit Quantum Gates

A single-qubit quantum gate is a transformation that acts on a single qubit. A single-qubit quantum gate can be represented by a unitary matrix, meaning it preserves the length of the qubit's state vector. A single-qubit quantum gate can also be visualized as a rotation around a certain axis of the Bloch sphere.

To illustrate these transformations, we show how to apply, for example, a Hadamard gate to the first qubit of the state $|\Psi\rangle = |00\rangle$. But first, let's look at the definition of this quantum gate.

$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ H|0\rangle &= |+\rangle \\ &\quad And \\ H|1\rangle &= \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\ H|1\rangle &= |-\rangle \end{aligned} \tag{1.39}$$

The matrix representation of Hadamard gate is

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{1.40}$$

Now let's apply Hadamard on the first qubit of the state $|\Psi\rangle = |00\rangle$. The operation is as follows:

$$
\begin{aligned}
H\,|00\rangle =& (H \otimes \mathbb{1})(|0\rangle \otimes |0\rangle) \\
=& \left[\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \otimes \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}\right]\left[|0\rangle \otimes |0\rangle\right] \\
=& \begin{pmatrix} \frac{1}{\sqrt{2}}\cdot\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} & \frac{1}{\sqrt{2}}\cdot\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} \\ \frac{1}{\sqrt{2}}\cdot\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} & -\frac{1}{\sqrt{2}}\cdot\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}\end{pmatrix}\begin{pmatrix}1\cdot\begin{pmatrix}1\\ 0\end{pmatrix} \\ 0\cdot\begin{pmatrix}1\\ 0\end{pmatrix}\end{pmatrix} \\
=& \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}}\end{pmatrix}\begin{pmatrix}1\\0\\0\\0\end{pmatrix} \\
=& \frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\\1\\0\end{pmatrix} \\
=& \frac{1}{\sqrt{2}}\left[\begin{pmatrix}1\\0\\0\\0\end{pmatrix} + \begin{pmatrix}0\\0\\1\\0\end{pmatrix}\right] \\
=& \frac{1}{\sqrt{2}}\big(\,|00\rangle + |10\rangle\,\big)
\end{aligned}
\tag{1.41}
$$

Controlled Quantum Gates

Controlled gates act on two or more qubits, where one or more qubits act as a control for some operation. Let U be a gate that operates on single qubits with matrix representation.

$$
U = \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11}\end{bmatrix} \tag{1.42}
$$

Then, the controlled-U operates on two qubits, so the first qubit serves as a control. It maps the basis states as follows.

$$\begin{aligned}
&|00\rangle \mapsto |00\rangle \\
&|01\rangle \mapsto |01\rangle \\
&|10\rangle \mapsto |1\rangle \otimes U\,|0\rangle = |1\rangle \otimes (u_{00}\,|0\rangle + u_{10}\,|1\rangle) \\
&|11\rangle \mapsto |1\rangle \otimes U\,|1\rangle = |1\rangle \otimes (u_{01}\,|0\rangle + u_{11}\,|1\rangle)
\end{aligned} \tag{1.43}$$

Thus, the matrix representation of the controlled-U is as follows.

$$C(U) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix} \tag{1.44}$$

To illustrate the use of these types of quantum gates, we show how to apply, for example, a Controlled Phase Shift gate to the first qubit of the state. But before that, let's look at the definition of this quantum gate. This gate maps the four possible combinations of two qubits into the following states.

$$\begin{aligned}
&|00\rangle \mapsto |00\rangle \\
&|01\rangle \mapsto |01\rangle \\
&|10\rangle \mapsto |10\rangle \\
&|11\rangle \mapsto e^{i\phi}\,|11\rangle
\end{aligned} \tag{1.45}$$

Formally, this is.

$$\begin{aligned}
U_{CPS}(\phi)\,|ab\rangle =& e^{i(a\cdot b)\phi} \quad Where \\
a \cdot b =& a \wedge b
\end{aligned} \tag{1.46}$$

The control qubit is a, and the target qubit is b. The operation $\wedge$ from the equation 1.46 is the classical logical AND operation. Thus, if $a = 0$, $a \cdot b = 0$; if $a = 1$, $a \cdot b = b$. That implies applying the phase-shift quantum gate to qubit b.

The matrix representation of the Controlled Phase Shift quantum gate (CPS) is as follows.

$$CPS \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix} \tag{1.47}$$

It can be said that applying the controlled phase shift gate is equivalent to applying the phase shift gate to the target qubit only if the control qubit is equal to 1. Then, applying

the CPS gate, for example, to the second qubit of the state $|11\rangle$ controlled by the first qubit is as follows.

$$\begin{aligned} U_{CPS}\,|11\rangle =& (U_{CPS})(|1\rangle \otimes |1\rangle) \\ =& \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix} \left[|1\rangle \otimes |1\rangle \right] \\ =& \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{pmatrix} \begin{pmatrix} 0 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 1 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} \\ =& \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \\ =& e^{i\phi} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \\ =& e^{i\phi}\,|11\rangle \end{aligned} \tag{1.48}$$

This result corresponds to the definition of how to apply a two-qubit quantum gate described in the equation 1.43.

1.4 State of the Art

As we study the limit of the non-simulable, it is important to verify that quantum computers behave as intended. Furthermore, it is of fundamental interest to classify that limit beyond which the quantum computers are doing something genuinely unreachable classically (quantum supremacy). Thus, pushing this boundary as far as possible is important to make quantum supremacy meaningful. The interest in general quantum simulation has been increased to fulfill this aim. Already, there are more than 100 classical simulators for various types of quantum systems available [13, 14, 12, 54]. Different results have been obtained in various areas of interest within this wide range of simulators. There are two major amplitude-wise approaches to simulate a quantum circuit of N qubits and [15] depth d:

- The first approach stores the entire state vector in memory; therefore, memory is the principal issue here.

- The second approach calculates the amplitude α_x for any N-bit string $x \in \{0,1\}^N$.

In the first approach, memory is the main concern. Considering the criticality of this resource, the simulation of quantum circuits has been limited by its dimensions. More than one decade ago, the largest simulation was a 42-qubit quantum circuit on the Jülich supercomputer [55]. Mikhail Smelyanskiy et al. used an Intel simulator to simulate up to 42 qubits of quantum supremacy circuits. A simulation of 45 qubits was carried out on the Cori II supercomputing system using 0.5 petabytes of memory and 8,192 nodes in 2017 [56]. In 2018, a 7×7 grid of depth 39 quantum circuit was simulated on the Sunway TaihuLight supercomputer [57]. In 2020, on the same supercomputer, a 49 qubits circuit of 55 depth was performed [58].

Several works implement different techniques to make better use of memory. Jianxin Chen et al. [15] works on a new technique, based on Google's model for variable elimination in the line graph, that implements a single-amplitude simulator that computes $\langle x| \mathcal{C} |0...0\rangle$ for an arbitrary quantum circuit $\mathcal{C}$. Jianxin tested the simulator for a randomly generated $\mathcal{C}$ from a restricted circuit class that produces a sampling problem classified as intractable [59]. The algorithm is based on tensor network contraction [60], where the treewidth is the dominant factor in determining the time and space complexity.

Aidan Dang et al. [16] studies how the entanglement structure of Shor's algorithm [17] is suitable for a particular matrix product state representation that quantifiably reduces the computational requirements for simulating it in a classical computer. The matrix product states of tensor networks [61] were originally used for simulating one-dimensional quantum many-body systems [62, 63], however, they have been adapted for simulating quantum circuits [64, 65] A traditional state vector representation of an n-qubit system requires the storage of 2^n complex scalars, no matter the state being stored. While simulations of quantum circuits may be performed by operating on this collection of scalars, the exponential space complexity limits the size of the systems that can reasonably be simulated. Nevertheless, by using the matrix product state representation of a quantum state, the space requirements scale according to the amount of entanglement present in the system.

Thomas Häner et al. [66] introduce a quantum computer emulator that makes use of an abstract, high-level quantum code which is available by directly utilizing classical emulation for quantum subroutines at the level of their mathematical description instead of compiling them into quantum gates before applying them using a series of sparse matrix-vector multiplications. This technique improves the notion of a simulator where a compilation framework for quantum programs is being used [67]. This enables an entirely new class of optimizations useful for simulating quantum computers.

The simulation of quantum systems carried out in classical computers has been done for a long time. However, the recent boom in real quantum devices has fueled the development of these simulators. It has increased pressure to determine the capabilities of what can be achieved in classical computers. This is important to know if quantum computers behave as predicted. Likewise, determining the limits. The fact that a quantum computer can

do things that cannot be achieved in a classical computer (quantum supremacy) is of fundamental interest.

1.4.1 Leading Open-Source Simulators for Quantum Computing

We must first select the proper platform to implement the techniques designed in this research. In that sense, we must consider the fundamental aspects and characteristics of the available alternatives. Many initiatives are working on quantum software, specifically quantum simulators, from academic research groups to big companies. A list of the very recent developments is maintained in several websites [13, 14, 12, 54]. They present several quantum computing software packages developed by different organizations.

LaRose et al. [68] reviewed some important general-purpose projects operating at the quantum gates level. Guzik [69] did a study on the appropriate approach to implement different models of quantum computing. Fingerhuth et. al. [36] evaluated a wide range of open source software for quantum computing, including all stages of the quantum toolchain from quantum hardware interfaces through quantum compilers to implementations of quantum algorithms, as well as several quantum computing models: quantum annealing and discrete and continuous-variable gate-model. The criteria used by this team to select the projects involve aspects like approved licenses, maturity, number of contributors, repository availability, etc.

From those reviews, we take some quantum simulators that currently lead the field to characterize their main properties, performance, execution mode, and simulation results to provide comparison and analysis. To facilitate this task, we work with open-source simulators. These simulators are considered state-of-the-art due to several factors [70]:

- **Innovative Features:** Each simulator offers unique capabilities that set them apart, such as optimized algorithms, integration with widely-used programming frameworks, or novel approaches to handling quantum state representations. For example, qsim's integration with Cirq and its ability to simulate up to 40 qubits on a high-performance workstation make it a significant tool for developers and researchers.

- **Adoption and Partnerships:** Some of these simulators are backed by major tech companies and have extensive partnerships within the industry, increasing their influence and credibility.

- **Academic and Commercial Use:** These tools are not only used in academic research but are also increasingly adopted by industries for practical applications, which demonstrates their effectiveness and robustness.

- **Recent Updates and Community Support:** The continual updates, community support, and documentation available for these tools contribute to their status as leaders in the field. This ongoing development ensures they remain relevant and useful as quantum computing technology evolves.

- **Open Collaboration:** Open-source projects encourage open collaboration among developers, researchers, and users. Ensuring the source code is available for modification and redistribution fosters a community-driven development approach. This can lead to rapid improvements and innovations, as a diverse group of contributors can work on the software.

The combination of these factors makes these simulators outstanding in the current world of quantum computing, pointing towards their innovativeness and leadership in technological advancement.

Algorithm Selected for Testing

The quantum Fourier transform was selected to carry out the simulations because it offers several advantages: it is a well-studied quantum algorithm with known properties, making it a reliable benchmark for validating the accuracy and efficiency of quantum simulators on classical hardware. QFT's performance scales predictably with the number of qubits, allowing researchers to analyze how the simulator handles increasing complexity. Performing QFT simulations helps estimate the computational resources (memory, processing power) required for larger, more complex quantum algorithms. Finally, QFT is a crucial component in many quantum algorithms, such as Shor's algorithm for factoring large integers. Simulating QFT provides a foundation for testing and understanding these more complex algorithms. We can find the details of QFT in appendix B.

Test platform

To evaluate the performance of the selected simulators, the following platforms were used:

- **Platform 1:** One of the nodes of the cluster Guane of Supercomputing Center of Universidad Industrial de Santander with the following configuration: two AMD EPYC 9554 64-Core (two threads per core) @ 3.1 GHz Processors and 375 GB of RAM memory.

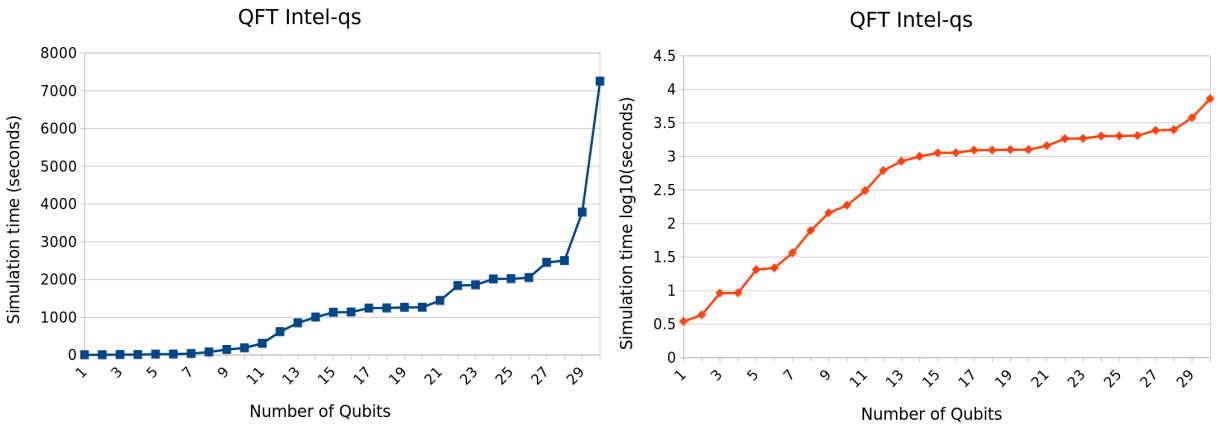
- **Platform 2:** A workstation with One Intel(R) Xeon(R) E-2136 CPU 6-Core, (two threads per core) @ 3.30GHz processor with 32 GiB of RAM and a NVIDIA Corporation GP106GL Quadro P2000 5GB. This node is used only for GPU-capable simulators.

Intel-QS

It is an open-source quantum circuit simulator implemented in C++. It uses multiprocessing and has an intuitive Python interface. It is a full-state vector simulator using arbitrary single-qubit gates and gates controlled by two qubits. [71]. The Intel Quantum Simulator

leverages the full capabilities of an HPC system through its shared and distributed memory implementation. The first scenario involves multiple processors or a processor with multiple computing cores accessing the same memory, allowing operations to be performed without a specific sequential order. This type of parallelism is most effectively harnessed using OpenMP. The implementation on a single node incorporates enhancements such as vectorization, threading, and cache optimization through the process of gate fusion. The second scenario for parallelism occurs when either a relatively small amount of memory requires extensive computation or, as in the case of storing quantum states, a large amount of memory exceeds the capacity of a single machine or node. In this situation, it is essential to explicitly manage the communication patterns between different processes, making using the Message Passing Interface (MPI) crucial. In the distributed version of the implementation, the state vector is partitioned evenly across the nodes. Each node retains its segment divided into two halves and also includes an additional temporary vector. When a gate operation necessitates communication, halves are exchanged pairwise between nodes. Subsequently, the gate operation is executed using the temporary vector, after which the halves are swapped back to their original nodes. The Intel-QS simulator uses the technique introduced by [22] to avoid constructing the matrix of the equation 1.36 when a quantum gate is to be applied to the state vector. The primary object in the Intel Quantum Simulator (IQS) is the QubitRegister, representing the quantum state of the qubits in the system of interest. When declaring a QubitRegister, the number of qubits must be specified to allocate enough memory to describe their state. The state can then be initialized to any computational basis state, uniquely identified by its index.

Simulation Results The performance and scalability of the Intel-QS simulator were evaluated on the **platform 1**. The performance for a QFT was reported varying the number of qubits from 1 to 30 using shared memory programming model with OpenMP. Figures 1.8a and 1.8b shows the QFT performance with Intel-QS.



(a) QFT performance (b) QFT performance log10

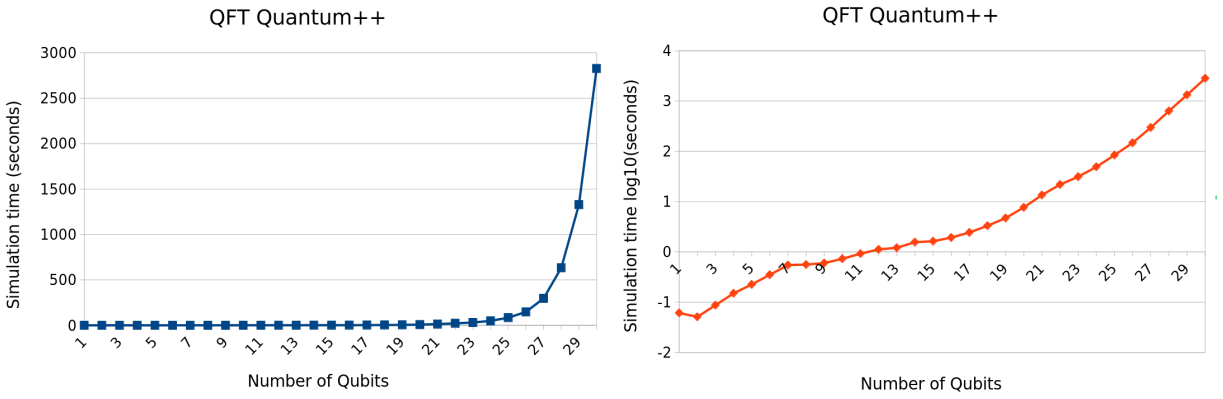
Figure 1.8: Intel-QS Simulator

Quantum++

Is a general-purpose multi-threaded quantum simulator with high performance. The library is not restricted to qubit systems or specific quantum information processing tasks, being capable of simulating arbitrary quantum processes [72]. Quantum++ is developed using standard C++17 and has minimal external dependencies. It primarily utilizes the Eigen 3 linear algebra template library, which is header-only. Additionally, when available, it employs the OpenMP library to facilitate multi-processing.

Quantum++'s design avoids a complex class hierarchy and instead focuses on a functional-style approach that is more suitable for a relatively small API. This allows users to focus on quantum algorithm design rather than object-oriented design. Furthermore, there is no need for explicit memory allocations or raw pointers. The library handles all resource allocations, initializations, and freeing, eliminating the risk of forgetting to deallocate memory, using uninitialized objects, or overflowing buffers, which are the common, dangerous, and difficult-to-diagnose errors in C and C++ programming. Although many quantum computing libraries and simulators are available in various programming languages, what sets Quantum++ apart is its ease of use, portability, and high performance. The primary data types are complex vectors and complex matrices, such as complex dynamic matrices, double dynamic matrices, complex dynamic column vectors, complex dynamic row vectors, etc.

Simulation Results The performance evaluation of the Quatum++ simulator was performed on the **platform 1**. In the same way, the range of qubits was varied from 1 to 30 using shared memory with OpenMP. Figures 1.9a and 1.9b shows the QFT performance with Quantum++.



(a) QFT performance (b) QFT performance log10

Figure 1.9: Quantum++ Simulator

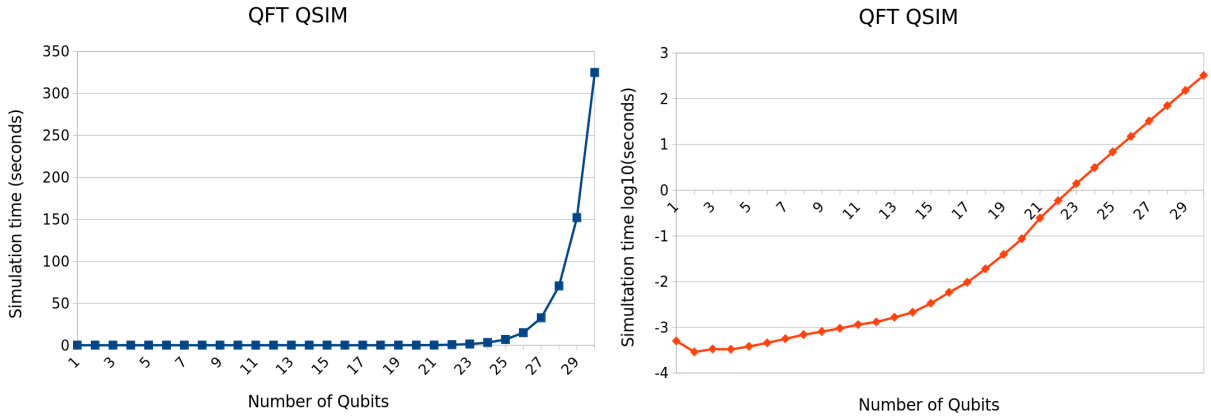
qsim

Developed by Google, qsim is an optimized quantum circuit simulator that uses gate fusion and vectorized instructions to simulate up to 40 qubits on a powerful workstation [73]. Integrated with Cirq, it provides a robust environment for developing and testing quantum algorithms. To achieve cutting-edge simulations of quantum circuits, it uses gate fusion, AVX/FMA vectorized instructions, and openMP multi-threading. This relies on cuQuantum to integrate GPU support.

cuQuantum

NVIDIA's cuQuantum SDK is another leading tool, designed to accelerate quantum circuit simulations on GPUs. This toolkit is essential for developers looking to leverage the power of GPUs to enhance simulation performance and scalability. It provides an integrated programming model tailored for a hybrid environment, enabling the combined operation of CPUs, GPUs, and QPUs.

Simulation Results The simulation of QFT using qsim was performed on the **platform 2**. Figures 1.10a and 1.10b shows the corresponding performance values.



(a) QFT performance (b) QFT performance log10

Figure 1.10: qsim Simulator

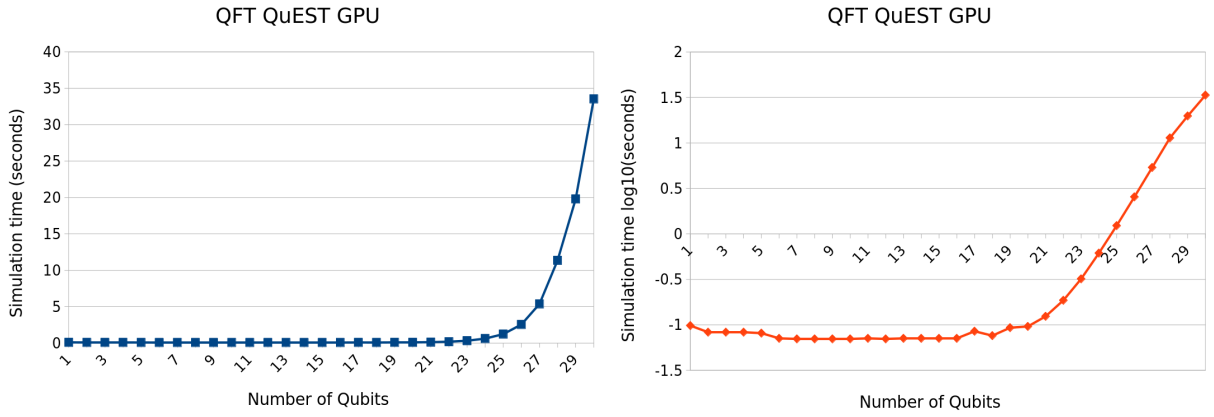
QuEST

QuEST, or the Quantum Exact Simulation Toolkit, is a high-performance open-source quantum computing simulator designed for simulating quantum circuits, state-vectors, and density matrices. Developed by the Quantum Technology Theory Group at the University of Oxford, QuEST is distinguished by its ability to utilize multithreading, GPU acceleration, and distribution, making it highly effective across various computing environments, from laptops to networked supercomputers. The toolkit is capable of simulating both

pure quantum states and mixed states with precision, and supports a wide array of quantum operations. It allows for simulations that are extensible and adaptable, thanks to its open-source nature and support for various back-end hardware via its simple and flexible interface [74]. QuEST supports complex quantum operations such as density matrices for simulating noisy quantum environments, general unitaries, and decoherence channels of any dimension, which makes it particularly powerful for advanced quantum computing simulations. It also offers tools for analyzing quantum states, such as probability, fidelity, and expected value calculations, enhancing its utility in research and educational settings.

QuEST represents a pure state for a system of n qubits using 2^n complex floating-point numbers, with each real and imaginary component having double precision by default. However, QuEST can be configured to use single or quad precision if desired. The simulator stores the state using C/C++ primitives, which means that by default, the state vector alone consumes 16×2^n bytes of memory.

Simulation Results The simulation of QFT using QuEST with GPU support was performed on the **platform 2**. Figures 1.11a and 1.11b illustrate the simulations results.



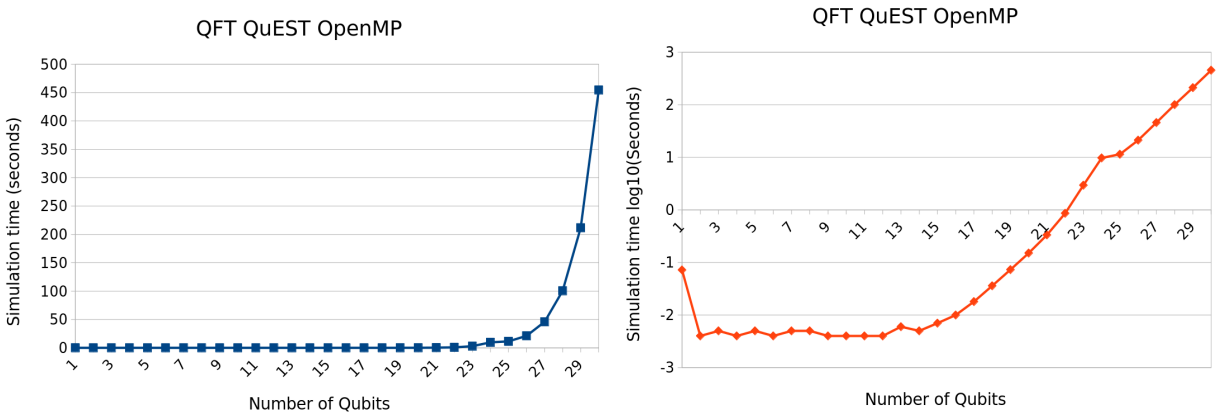
(a) QFT performance (b) QFT performance log10

Figure 1.11: QuEST Simulator

QFT simulation using QuEST with OpenMP support was performed on the **platform 1**. Figures 1.12a and 1.12b illustrate the simulations results.

Qrack

Qrack is a high-performance quantum computer simulator that is written in C++ and supports OpenCL and CUDA [75] [76]. It is particularly notable for its ability to simulate arbitrary numbers of entangled qubits, limited only by system resources. Qrack is designed to be embedded in other projects and includes a comprehensive suite of standard quantum gates, along with variations suitable for register operations and arbitrary rotations. The simulator is integrated with other quantum computing frameworks like ProjectQ and



(a) QFT performance (b) QFT performance log10

Figure 1.12: QuEST Simulator

Qiskit, enhancing its versatility and application. Qrack also features optimizations for noiseless pure state simulations and includes tools that aid in the control, extension, and visualization of data from quantum circuits.

Qrack's core component, called QEngine, stores a set of complex number coefficients in a permutation basis and operates on them using logic gates and register-like methods. The state vector indicates the probability and phase of all possible pure bit permutations, numbered from 0 to $2^n - 1$, via simple binary counting.

Qrack is implemented with float precision complex numbers by default. Using double precision is optional and essentially costs the equivalent of one additional qubit, resulting in twice as many potential bit permutations on the same system. However, double precision complex numbers naturally align with the width of SIMD intrinsics. It is up to the developer to decide whether precision and alignment with SIMD or having one additional qubit on a system is more important.

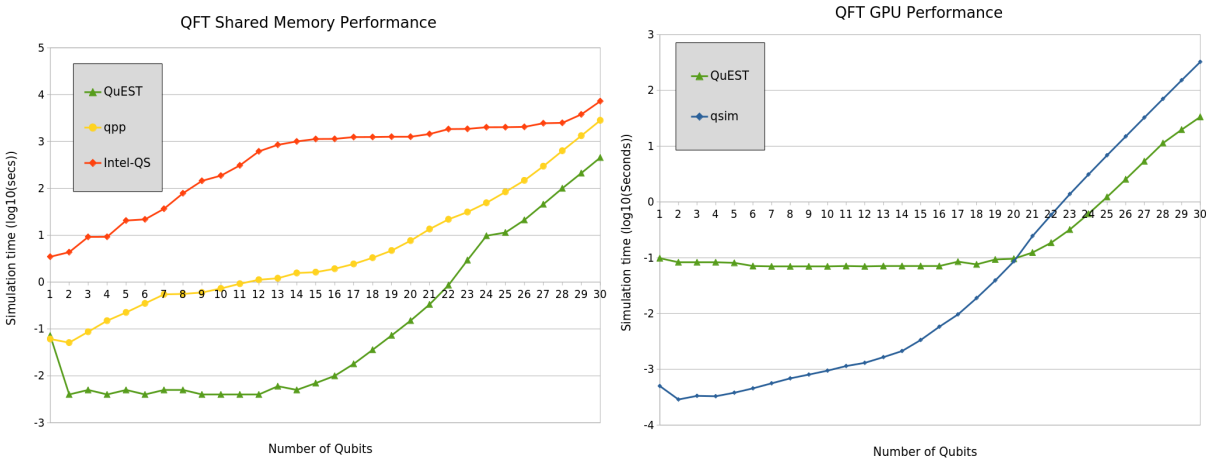
Qrack maintains the state representation in a factorized form to enhance simulation efficiency. A general ket state $|\psi\rangle$ of n qubits is described by $O(2^n)$ complex amplitudes. However, if the state $|\psi\rangle$ can be factorized as the tensor product of m local states.

$$|\psi\rangle = |\psi_{S_1}\rangle \otimes |\psi_{S_1}\rangle \dots |\psi_{S_m}\rangle \tag{1.49}$$

Where $S_1, \dots, S_m$ represent disjoint subsets of the n qubits, the number of complex amplitudes required to represent $|\psi\rangle$ can be significantly reduced. For example, in the extreme case where $m = n$, meaning the qubits are not entangled, $O(n)$ complex amplitudes are sufficient.

1.4.2 Simulators Comparison

Observing all simulator's results, we can notice as the number of qubits increases, the execution time increases exponentially, especially beyond 25 qubits. This exponential increase is typical due to the growing complexity and state space with more qubits. The performance results indicate efficient handling of quantum processes, but highlight the challenges posed by scaling. QuEST and qsim, with GPU support, performs significantly better for larger qubit sizes due to its utilization of GPU acceleration and multithreading. Figures 1.13a and 1.13b show the comparison of the performance of the simulators.



(a) Shared Memory Performance (b) GPU Performance

Figure 1.13: Comparison of the quantum Fourier transform using different simulators and optimization techniques.

Regarding academic, community and industry support for these simulators, the continual updates, active support, and documentation for these tools contribute to their status as leaders in the field. This ongoing development ensures they remain relevant and valuable as quantum computing technology evolves. Each of these simulators offers unique features and optimizations, making them suitable for various aspects of quantum computing research and development. Their continual evolution is critical as the quantum computing field strives to solve more complex problems and improve algorithm efficiency. Table 1.3 shows a comparison of the evaluated simulators of their design properties and optimization mechanisms.

Table 1.3: Leading Open-Source Simulators for Quantum Computing Comparison

Features	Intel-QS	Quantum++	QuEST	Qrack	qsim
Optimization Techniques	Uses MPI for distributed computing, optimizes for multi-core and multi-nodes, supports MKL for mathematical operations	Employs template metaprogramming for compile-time optimizations, supports OpenMP for parallelization	Utilizes GPU acceleration, multithreading, and can be distributed across networked supercomputers	Leverages gate fusion and OpenCL for parallel execution across different hardware platforms	Optimizes simulations with gate fusion, AVX/FMA vectorized instructions, and OpenMP
Hardware Support	Optimized for high-performance computing systems, can be deployed on cloud infrastructures	Compatible with various architectures via C++ standardization, no specific hardware acceleration mentioned	Supports laptops to supercomputers, compatible with GPUs and distributed systems	Designed for broad compatibility with OpenCL-supporting GPUs and CPUs	Runs efficiently on high-core-count CPUs and potentially on any system supported by Cirq
Programming Model	Provides C++ and Python interfaces, supports state vector simulation	C++ library with emphasis on flexibility and ease of integration	Offers a C library that's easy to integrate and extend, with optional Python bindings	C++ based, with a focus on integrating with other quantum computing frameworks like Qiskit	Integrated with Cirq, emphasizes ease of use in Python for simulating quantum circuits
Design Properties	Focuses on scalability and performance across different computational environments	Prioritizes modular, generic programming for ease of adaptation and maintenance	Designed for precision and versatility in quantum state manipulation	Prioritizes rapid prototyping and flexibility for embedding in various applications	Designed to simulate large quantum circuits with high precision
Unique Features	Supports dynamic circuit simulation and state manipulation during runtime	Highly adaptable to various quantum computing models due to generic programming approach	Extensible and supports detailed state analysis tools like fidelity and entanglement measures	Integrates classical computing elements within quantum simulations for enhanced functionality	Deeply integrated with Google's quantum computing framework, providing extensive simulation capabilities

Other projects, like XACC and Qiskit, provide a full-stack approach to quantum computing, including a simulator and compilers and the possibility to run the program on real quantum processors.

In this work, several strategies to reduce memory consumption are studied: selecting correct data structures to use less memory, eliminate less probable quantum states, data compression, and the compressed state vector distribution among multiple computing nodes. The details of these strategies are shown in the next chapter. Besides, a prototype has been developed that allows us to test the strategies proposed in the next chapter.

Chapter 2

Memory Management Strategies

In order to obtain meaningful results from simulations, it is crucial to use a reasonable number of qubits. Consequently, efficient memory management is of utmost importance.

This chapter discusses several memory management strategies for optimizing quantum simulations on classical architectures. As quantum simulations become more complex and larger, efficient memory management becomes crucial to handle the exponential growth of resource requirements. The goal is to identify and implement techniques to reduce memory usage while significantly maintaining simulation accuracy and performance. These approaches range from careful selection of appropriate data structures to data compression. Of course, simulation performance remains a critical consideration in each scenario.

2.1 Proper Data Structures

Remember that the Quantum Register and the Quantum gates are the main elements we must build to create a minimum environment for carrying out simulations using the quantum circuit model. We need to store the states and amplitudes in the equation 1.35, where those amplitudes are complex numbers.

The first approach to represent the binary form of elements of the state vector in Figure 2.1 is to use character strings, similar to how [77] does it. However, for programming languages such as C or C++, storing the state vector of a 50-qubit quantum register requires 50 PB of memory.

A better approach is to replace the binary notation of the state vector elements with their integer equivalent. This way, we only need 4PB to store a quantum register using an `unsigned int` data type.

The data structures in figure 2.1 can be replaced by a C++ object such as **std::map**. Maps are associative containers that store elements formed by a combination of a key value and a mapped value, following a specific order. The key value could represent states, and the assigned value can be another native C++ object such as **std::complex** to represent amplitudes. Both objects implement methods that facilitate quantum register modeling.

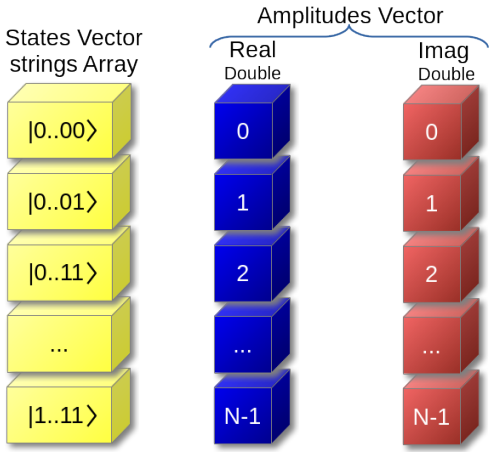


Figure 2.1: Basic Data Structures

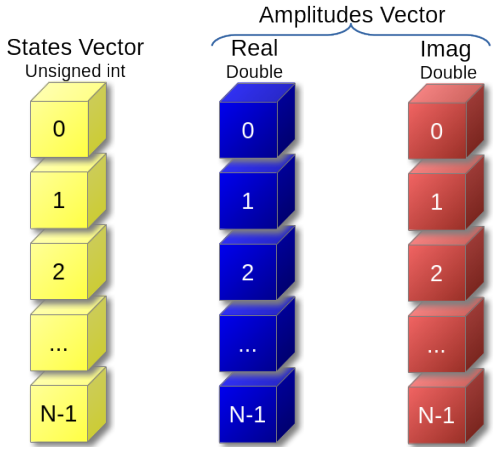


Figure 2.2: Improved Data Structures

The key value could be strings. However, as we mentioned before, it is more efficient to use integers. Despite these advantages, if we want to use the data compression approach, the available libraries do not support these native C++ structures.

2.2 Remove the least Probable Quantum States

Removing the least probable quantum states, a process often referred to as "state pruning" or "truncation," can save substantial amounts of memory. In this case, dynamic memory management is necessary to maintain the quantum register data. Using this strategy, it is possible to use $1\,|00..00\rangle$ as the initial state, occupying only 20 bytes. The vector of states

and amplitudes will grow dynamically as operations with quantum gates are performed on the quantum register. For example, we will obtain the following result after applying the Hadamard gate on the first qubit on $1\,|00\rangle$ as the initial quantum register.

$$H\,|00\rangle = \frac{1}{\sqrt{2}}\,|00\rangle + \frac{1}{\sqrt{2}}\,|10\rangle \tag{2.1}$$

The state vector will grow in a non-sequential manner. This approach adds the need to search for each state and its amplitude within the state vector each time it is used. This negatively impacts the performance of the simulations.

On the other hand, this technique also entails significant risks and potential drawbacks, implying a negative impact on several aspects such as:

- **Loss of Fidelity**: The fidelity of a quantum state refers to how closely the actual quantum state matches the intended state. Removing less probable states reduces the overall fidelity of the quantum register, degrading the quality of the quantum computation and increasing the likelihood of errors. Quantum states, even those with low probability, can interfere in ways that are non-intuitive but crucial to the outcomes of quantum algorithms.

- **Impact on Algorithm Accuracy**: Certain quantum algorithms rely on the collective behavior of multiple quantum states, including those with lower probabilities. Algorithms that depend on the superposition and entanglement of all possible states may yield incorrect results if some states are pruned.

- **Error Accumulation**: In longer or more complex quantum circuits, the error introduced by pruning less probable states can accumulate, potentially leading to significant deviations from the correct quantum behavior as the simulation progresses.

- **Threshold Sensitivity**: Deciding the threshold below which states are considered "improbable" and can be safely discarded is challenging. Setting this threshold too high might result in the loss of important states, whereas too low a threshold may not offer sufficient memory savings. A practical approach is to set the threshold at the smallest number that the utilized architecture can accurately represent.

- **Impact on Quantum Entanglement**: Quantum entanglement can cause a previously improbable state to become significant due to the interactions with other states. Removing certain quantum states can disrupt quantum entanglement, reducing the effectiveness of algorithms that rely on these correlations for computational advantage.

Let's take into account the following statistical considerations to better understand this strategy's disadvantages in terms of obtaining accurate results. First, let's remember

that a quantum state of n qubits is represented by the equation 1.8. The probability of measuring the state $|i\rangle$ is $|\alpha_i|^2$. If $|\alpha_i|^2$ is small, the state $|i\rangle$ is considered "least probable".

To eliminate the least probable states, we set a threshold ϵ and remove all states $|i\rangle$ for which $|\alpha_i|^2 < \epsilon$ The elimination of the least probable states introduces an error in the representation of the quantum state.

A quantum state $|\psi\rangle$ of an n-qubit system can be expressed as a superposition of 2^n basis states:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \tag{2.2}$$

Where the sum of the squares of the magnitudes of the amplitudes must equal 1

$$\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1 \tag{2.3}$$

Let the pruned state be denoted as $|\psi_p\rangle$, where:

$$|\psi_p\rangle = \frac{1}{\sqrt{N}} \sum_{i \in S} \alpha_i |i\rangle \tag{2.4}$$

Where

- S is the set of indices for the basis states that are retained after pruning.
- N is a normalization factor to ensure that the pruned state remains a valid quantum state (i.e., $\langle\psi_p| |\psi_p\rangle = 1$)

$$N = \sum_{i \in S} |\alpha_i|^2 \tag{2.5}$$

The fidelity F between the original state $|\psi\rangle$ and the pruned state $|\psi_p\rangle$ is given by the square of the absolute value of the inner product of the two states [78, 79, 80, 81]:

$$F(\psi, \psi_p) = | \langle\psi|\psi_p\rangle |^2 \tag{2.6}$$

Substituting the expressions for $|\psi\rangle$ and $|\psi_p\rangle$:

$$\langle\psi|\psi_p\rangle = \frac{1}{\sqrt{N}} \sum_{i \in S} |\alpha_i|^2 \tag{2.7}$$

Thus, the fidelity becomes:

$$F(\psi, \psi_p) = \frac{1}{N} \left(\sum_{i \in S} |\alpha_i|^2 \right)^2 \tag{2.8}$$

Substituting the expressions for $|\psi\rangle$ and $|\psi_p\rangle$:

$$\langle \psi | \psi_p \rangle = \frac{1}{\sqrt{N}} \sum_{i \in S} |\alpha_i|^2 \tag{2.9}$$

Thus, the fidelity becomes:

$$F(\psi, \psi_p) = \frac{1}{N} \left(\sum_{i \in S} |\alpha_i|^2 \right)^2 \tag{2.10}$$

Fidelity loss can be quantified as the difference between the ideal fidelity (which would be 1 if there were no pruning) and the actual fidelity after pruning:

$$\text{Fidelity Loss} = 1 - F(\psi, \psi_p) \tag{2.11}$$

Substituting the expression for $F(\psi, \psi_p)$:

$$\text{Fidelity Loss} = 1 - \frac{1}{N} \left(\sum_{i \in S} |\alpha_i|^2 \right)^2 \tag{2.12}$$

Given that N is the sum of the retained probabilities:

$$N = \sum_{i \in S} |\alpha_i|^2 \tag{2.13}$$

we can rewrite the fidelity loss as:

$$\text{Fidelity Loss} = 1 - \left(\sum_{i \in S} |\alpha_i|^2 \right) \tag{2.14}$$

Since $\sum_{i \in S} |\alpha_i|^2$ is less than or equal to 1, the fidelity loss indicates how much of the original state's information is discarded during pruning.

Since $\sum_{i \in S} |\alpha_i|^2$ is less than or equal to 1 (depending on the amount of pruning), the fidelity loss indicates how much of the original state's information is discarded during pruning.

Let's consider a simple example with a 2-qubit system, where the quantum state $|\psi\rangle$ is:

$$|\psi\rangle = 0.8\,|00\rangle + 0.4\,|01\rangle + 0.2\,|10\rangle + 0.1\,|11\rangle \tag{2.15}$$

Suppose we prune the states with $|\alpha_i| < 0.15$, i.e., we remove the state $|11\rangle$ (since its amplitude is 0.1). The pruned state $|\psi_p\rangle$ becomes:

$$|\psi_p\rangle = \frac{1}{\sqrt{0.8^2 + 0.4^2 + 0.2^2}} (0.8\,|00\rangle + 0.4\,|01\rangle + 0.2\,|10\rangle) \tag{2.16}$$

Calculating the normalization factor N:

$$N = 0.8^2 + 0.4^2 + 0.2^2 = 0.64 + 0.16 + 0.04 = 0.84 \tag{2.17}$$

Now, the fidelity between the original and pruned states is:

$$F(\psi, \psi_p) = \left(\frac{1}{\sqrt{0.84}} \cdot (0.8^2 + 0.4^2 + 0.2^2) \right)^2 = \left(\frac{0.84}{\sqrt{0.84}} \right)^2 = 0.84 \tag{2.18}$$

The fidelity loss is:

$$\text{Fidelity Loss} = 1 - 0.84 = 0.16 \tag{2.19}$$

This means that pruning this state has resulted in a 16% loss in fidelity, indicating that the pruned state has deviated from the original state by this amount.

Thus, the fidelity decreases as the number of removed states increases, quantifying the trade-off between memory savings and state accuracy. Eliminating the least probable quantum states can save memory and computational resources but introduces approximation errors. The mathematical analysis of the error norm and fidelity provides a framework to quantify and understand these risks. The decision to eliminate states should consider the specific requirements of the quantum algorithm and the acceptable level of error.

2.3 Full State Data Structures

If we start the simulations with full-state vector states and amplitude vectors, It is possible to eliminate the storage of the state vector and manage the states using the indices of the amplitudes vector. On the other hand, linearizing the amplitude vector from the beginning of the simulations allow the allocation of continuous memory, which will positively impact the execution performance.

Organizing all states in a sequence within a quantum register can streamline the process of locating specific quantum states, thereby minimizing overhead introduced by the search of a specific state inside a not-ordered array. This approach also avoids the necessity for an additional data structure to hold the states, as it leverages the indices of the amplitude vector for managing the quantum states.

2.4 Applying Quantum Gates to a Quantum State

Quantum gates are the fundamental operations in quantum circuits, analogous to logic gates in classical computing. In the quantum circuits model, these gates are used to

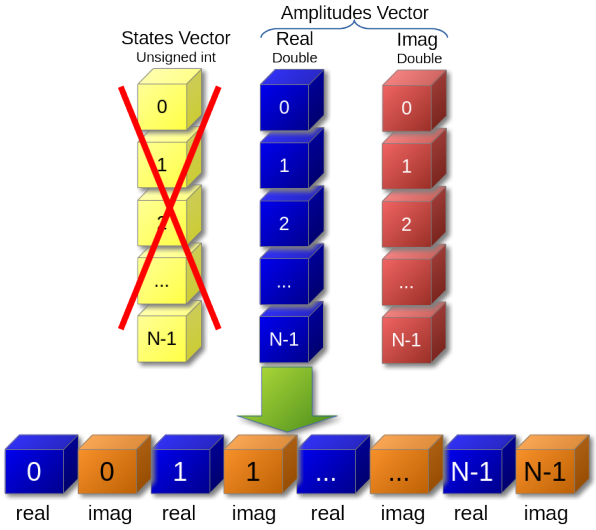


Figure 2.3: Full State Data Structures

manipulate the quantum states of qubits. The most commonly used gates include single-qubit and two-qubit gates, often controlled gates.

As the number of qubits in a quantum state increases, the size of the unitary matrix that represents a quantum gate also grows exponentially. Specifically, to apply a quantum gate to an n-qubit quantum register, the matrix of the equation 1.36 is required, whose size is $2^n \times 2^n$.

However, according to research [22] and [82], it is not always necessary to explicitly construct such large matrices. Instead, the quantum state can be transformed using more efficient methods, bypassing the need for full matrix representation. The following subsections explain how quantum gates are applied to single and two-qubit systems.

2.4.1 Applying single-qubit Gates

A single-qubit gate transforms one qubit while leaving the rest of the quantum register unchanged. A traditional way to manage the application of such gates is through sparse matrix methods, which reduce the memory requirements significantly. According to [22] and [82], applying a single-qubit gate G_k to the k-th qubit of an N-qubit register can be computed by processing pairs of amplitudes whose indices differ by the k-th bit of their binary representation.

The action of a single-qubit gate G_k, represented as:

$$G_k = \begin{pmatrix} g_{00} & g_{01} \\ g_{10} & g_{11} \end{pmatrix} \tag{2.20}$$

results in updating the amplitudes of the quantum state as follows:

$$\begin{aligned}\alpha'_{*...*0_k*...*} &= g_{11} \cdot \alpha_{*...*0_k*...*} + g_{12} \cdot \alpha_{*...*1_k*...*} \\ \alpha'_{*...*1_k*...*} &= g_{21} \cdot \alpha_{*...*0_k*...*} + g_{22} \cdot \alpha_{*...*1_k*...*}\end{aligned} \tag{2.21}$$

This efficient method allows for the transformation of quantum states without explicitly constructing large matrices, improving both time and memory performance.

To show how this method works, let's continue with the example of equation 1.41, let's see how to apply the Hadamard gate to the first qubit of the state $|00\rangle$. For the values: $k = 0$, $*... * 0_k * ...* = 00$, $*... * 1_k * ...* = 10$, $\alpha_{00} = 1 + 0i = 1$, $\alpha_{10} = 0$ and Hadamard gate.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.22}$$

Replacing these values in equation 2.21, we obtain the following results, which are the same results in equation 1.41.

$$\begin{aligned}\alpha'_{00} &= \frac{1}{\sqrt{2}} \cdot 1 + \frac{1}{\sqrt{2}} \cdot 0 = \frac{1}{\sqrt{2}} \\ \alpha'_{10} &= \frac{1}{\sqrt{2}} \cdot 1 - \frac{1}{\sqrt{2}} \cdot 0 = \frac{1}{\sqrt{2}}\end{aligned} \tag{2.23}$$

2.4.2 Applying two-qubits Gates

In the case of two-qubit gates, such as controlled gates, the process is slightly more complex. A controlled gate modifies the state of one qubit (the target qubit) based on the value of another qubit (the control qubit). For a controlled operation between qubit c (control) and qubit t (target), the transformation of amplitudes can be described as follows:

$$\begin{aligned}\alpha'_{*..*1_c*..*0_t*..*} &= g_{11} \cdot \alpha_{*..*1_c*..*0_t*..*} + g_{12} \cdot \alpha_{*..*1_c*..*1_t*..*} \\ \alpha'_{*..*1_c*..*1_t*..*} &= g_{21} \cdot \alpha_{*..*1_c*..*0_t*..*} + g_{22} \cdot \alpha_{*..*1_c*..*1_t*..*}\end{aligned} \tag{2.24}$$

This method is critical in quantum algorithms such as the Quantum Fourier Transform and Grover's algorithm, which rely heavily on the use of controlled two-qubit gates. By applying these gates efficiently, the simulator can manage larger quantum states while maintaining accuracy in the transformations.

Continuing with the example of equation 1.48, let's see how to apply the CPS gate to the second qubit of the state $|11\rangle$ controlled by the first qubit. All amplitudes are equal to 0 except α_{11} which is equal to 1. Replacing these values in the equation 2.24 we have:

$$\begin{aligned}\alpha'_{10} &= 1 \cdot 0 + 0 \cdot 0 = 0 \quad ' \\ \alpha'_{11} &= 0 \cdot 1 + e^{i\phi} \cdot 1 = e^{i\phi}\end{aligned} \tag{2.25}$$

Thus, we obtain the amplitude values for the states $|10\rangle$ and $|11\rangle$

2.5 Data Compression

The large volumes of data produced by extreme-scale scientific research and applications, especially in quantum computing, have driven various data compression techniques to be developed for years. These techniques have been proposed to reduce the storage space of scientific data by using compression methods optimized for floating point data. However, they require additional calculations to compress and decompress data before working with it. Scientific researchers often use two kinds of compressors: lossless compressors that preserve all the data or error-bounded lossy compressors that allow some controlled distortion.

Lossless compressors typically use variable length encoding algorithms such as Huffman encoding [83] and arithmetic encoding [84], as well as dictionary encoders such as LZ77/78 [85]. Despite their effectiveness in various scenarios, widely used lossless compressors such as gzip and bzip2, among others, face challenges regarding scientific data. This is mainly due to the random nature of the final mantissa bits in floating point values, which makes it difficult to identify consistent patterns in the data flow [86].

In work [87], authors point out that traditional lossless compressors often produce low compression ratios, typically around 2:1, which does not meet the demands of today's extreme-scale high-performance computing (HPC) needs. Consequently, the scientific community has widely adopted error-bounded lossy compression as a superior solution to address the challenges posed by large scientific data sets. This approach significantly reduces data size and allows control of data distortion based on user-defined requirements.

Error-bounded lossy compressors come in various designs and implementations, making the selection of an appropriate compression technique crucial to research efforts. These compressors can be broadly classified into data prediction-based and domain transformation-based models, each described in more detail below [86].

- The data-prediction-based compression model aims to achieve precise predictions for each data point by considering its neighboring points in spatial or temporal dimensions. Subsequently, the data size is reduced through coding algorithms like data quantization [88] and bit-plane truncation. A notable example of this model is SZ [89], which encompasses four compression steps: data prediction, linear-scaling quantization, entropy encoding, and lossless compression. Another example of this model is FPZIP [90].

- In contrast, the domain-transform-based compression model transforms all original data values into a non-orthogonal coefficient domain for decorrelation. The data size is then reduced through optimized algorithms like embedded coding [91]. An exemplary compressor in this category is ZFP [91]. Which executes classic texture compression using three techniques within each 4D block (where d represents the dimensions): exponent alignment, (non)orthogonal block transform, and embedded

coding. Other examples adopting the Wavelet transform in the domain transform step are VAPOR [92] and Sasaki et al.'s compressor [93].

Due to the limitations of conventional compression methods in handling scientific data, specialized (lossy) compressors like ZFP [91] and SZ [87] have been developed. These compressors exhibit remarkable performance in terms of both accuracy and reduction in data size, enabling a substantial decrease in storage requirements by 10 to 20 percent without sacrificing significant relevant information. This reduction in data footprint proves instrumental in minimizing the storage and communication needs for scientific data. However, it has to be pointed out that the computational demands escalate as the data must undergo decompression before processing and subsequent recompression. This increased computational resources required for these operations is a critical consideration [94].

Some of the most recognized and effective floating point compression libraries for each of the models described above are SZ and ZFP. The table 2.1 shows the main characteristics of them [95] [96].

To test the data compression strategy, we have selected ZFP as a suitable library to store the amplitude vector. ZFP supports both lossy compression and lossless compression. The ZFP lossy compression approach involves partitioning a d-dimensional array into discrete blocks containing $4d$ values. For instance, in three dimensions, each block may consist of $4 \times 4 \times 4$ values. Notably, the compression or decompression of each block occurs in complete isolation from all other blocks. In this regard, the zfp method shares similarities with existing hardware texture compression schemes employed in image coding on graphics cards and mobile devices. The reversible (lossless) compression algorithm closely resembles the procedure of the lossy algorithm, considering some parts as sources of error, where loss may occur during the conversion to zfp's block-floating-point representation; the reversible algorithm incorporates a safeguard to verify the losslessness of this conversion. This verification involves reverting the values to the source format and checking for bitwise equality with the uncompressed data. If this test is successful, a modified decorrelating transform is executed, utilizing only reversible integer subtraction operations. Additionally, ZFP exhibits several advantages over SZ that make this library most suitable for this work.

- The compression or decompression of each block occurs in complete isolation from all other blocks.
- Supports both lossy compression and lossless compression.
- Very simple interface.
- Block-based approach combined with an orthogonal transform and embedded encoding achieve efficient compression.
- Its key strength lies in the fine control it provides over the trade-off between compression ratio and precision.

Feature	SZ	ZFP
Type of Compression	Error-bounded Lossy	Lossy (fixed-precision, fixed-rate)
Primary Use Case	Scientific data compression (e.g., simulations)	Scientific data compression (e.g., climate models)
Compression Mode	User-defined error bound	User-defined precision or rate
Error Control	Absolute and relative error bounds	Fixed-point precision, fixed-rate control
Compression Ratio	High (depending on error bound)	High (depending on precision or rate)
Supported Data Types	Floating-point (32-bit and 64-bit), integers	Floating-point arrays (single/-double precision)
Data Dimensionality	Multidimensional arrays	1D, 2D, 3D, and 4D arrays
Speed	Fast, optimized for high compression ratios	Fast, optimized for both speed and compression
Decompression Speed	Fast	Fast
Lossless Compression	Not primarily, but has some support	Not primarily, but has reversible transform support
Parallelization	Supported (OpenMP, MPI)	Supported (OpenMP, CUDA)
Library Language	C/C++, Python wrappers	C/C++, Python, Fortran wrappers
Maturity & Community	Mature with a strong scientific community	Mature with a strong scientific community
License	BSD-3-Clause	BSD-3-Clause
Use Cases	Large-scale simulations, climate data, astrophysics	Climate modeling, fluid dynamics, signal processing
Customizability	High (user-defined parameters, custom filters)	High (configurable precision and rate)

Table 2.1: Comparison between SZ and ZFP compression libraries

ZFP library has several compression modes. The table 2.2 shows a brief description of them. Since memory saving is a central goal of this work, fixed-rate compression mode represents the most suitable use case.

2.6 Distribute the Vector State Among Different Computing Nodes

Using a distributed memory programming model like a message-passing interface allows us to leverage the resources of several nodes configured in cluster architecture to store the state vector with a more significant amount of RAM memory. Thus, we can increase the number of qubits.

To use this strategy, certain conditions must be taken into account. Assuming that the nodes have a uniform amount of memory M, we could divide the number of states

Mode	Purpose	Use Case
Expert	Provides fine-grained control over compression settings.	Advanced users optimizing for specific applications.
Fixed Precision	Maintains a specified precision for each value.	Important when preserving significant bits is crucial.
Fixed Accuracy	Controls the maximum absolute error in compressed data.	Common in streaming I/O operations for limiting absolute error.
Reversible	Achieves lossless compression with exact data reconstruction.	Critical when no data loss can be tolerated (e.g., legal, medical records).
Fixed Rate	Achieves a specific bit rate per value.	**When storage or bandwidth constraints are strict.**

Table 2.2: ZFP compression modes

$2^{numQubits}$ present in the state vector by the number of nodes N. The distribution of the state vector between the nodes must be done so that each node contains the same number of states; that is, the division of $2^{numQubits}$ by the number of nodes must be exact.

This way, we could have a memory capacity $M \times N$ where $2^{numQubits} < M \times N$. On the other hand, details such as the impact of communications and adequate partitioning must be considered.

In this chapter, we study different strategies to reduce memory usage in quantum circuit simulations carried out on classical computers. These strategies range from using appropriate data structures to represent the fundamental elements of quantum computing to state elimination and the distribution of the state vector in a compressed form. To test these strategies, it was decided to develop a prototype that facilitates simple modifications for testing each strategy. This approach was considered to be more expeditious than modifying an existing simulator. The following chapter provides detailed information on the implementation of this prototype.

Chapter 3

Implementing a Simulator

To gain a deeper understanding of the fundamental operations of quantum computing and to test the various memory management approaches discussed in the previous chapter, a software quantum simulator prototype was developed in C++. (The Memory eFficient Quantum Simulator, TMFQS) [97]. This prototype was designed to switch strategies easily through minimal modifications. It allows us to easily adjust the data structures to represent the fundamental concepts of quantum computing and the use of compression libraries. On the other hand, the scope of this work was limited to optimization techniques using shared memory and distributed memory. The implementation using GPUs is left for further work.

It has to be pointed out that this prototype does not implement all the concepts of quantum computing, such as quantum error correction, entanglement, measurement and an extended set of quantum gates. The primary purpose of this prototype is to provide a minimal platform for carrying out memory consumption tests. Since these techniques do not inherently depend on measurement processes, implementing a measurement function was not essential to achieving the research goals. However, the measurement operation could be incorporated in future iterations to extend the simulator's capabilities for practical quantum algorithm execution. Several scenarios were implemented to carry out the tests.

- Dynamic memory management. The primary purpose is to test the strategy of removing less probable states.
- Full State: The objective is to accelerate the simulations, avoiding the overhead introduced by the search of the states.
- Full State with OpenMP: The intention is to accelerate the simulations of the previous version.
- Full State with data compression: The purpose is to test a lossy compression library like ZFP.
- Full State with MPI: The main objective of this scenario is to distribute the amplitude vector among different computing nodes, allowing for a greater number of qubits.

- Full State with MPI and data compression: Here, data compression was incorporated into the previous scenario.

The only public version of this prototype is the second one because the authorship details are not yet defined for the rest of them. Readers can request any of the other versions on demand.

3.1 Representation of Fundamental Concepts

As we saw in section 1.3.1, the basic simulation concepts include the following elements.

- Quantum Register: comprised of states vector and amplitudes vector.
- Quantum Gates: matrix representation of quantum gates. Only one-qubit and two-qubit gates were considered.
- Applying quantum gates to a quantum register.

We have used an array of double-precision floating point numbers to store the amplitudes. No data structure has been used to represent the states, since the vector indices are used to refer to them. To implement the method to apply a quantum gate to a quantum register (`QuantumRegister::applyGate`), we use the technique represented in equation 2.21 proposed by [22]. In the figure 3.1, we can observe the main classes of the prototype.

3.1.1 Single Processor Case

State Vector

The state vector is a linear combination of states represented by the equation 1.35. As we said previously, the amplitudes are complex numbers, so we need two `float` or two `double` numbers to represent them in the code. Of course, the state vector must fit in the local memory. If we use binary notation we can rewrite the state vector as follows:

$$|\Psi\rangle = \begin{pmatrix} \alpha_{0\ldots00} \\ \alpha_{0\ldots01} \\ \vdots \\ \alpha_{1\ldots10} \\ \alpha_{1\ldots11} \end{pmatrix} \tag{3.1}$$

The amplitudes of the states are implemented using a single-dimension double-precision array stored in a continuous memory space. To increase performance, a single array was used to store both the real and the imaginary parts of each amplitude; that is, the state vector was linearized. The real parts are placed in the odd positions of this arrangement,

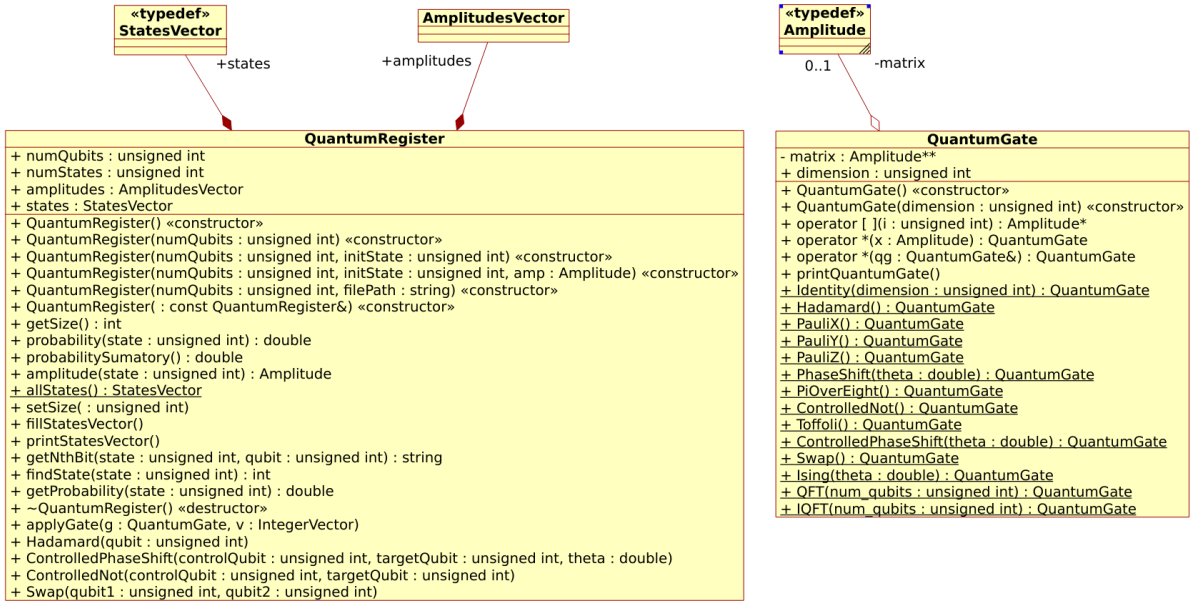


Figure 3.1: Class Diagram of the Prototype

and the imaginary parts are placed in the even positions. This strategy avoids jumping between two arrays, one for the real part and one for the imaginary part. Figure 3.2 depicts this data structure.

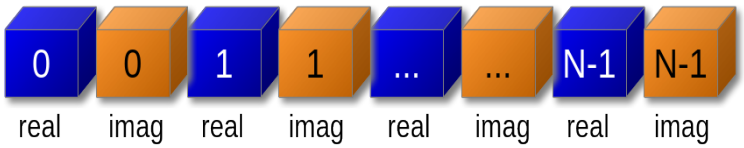


Figure 3.2: Linearized state vector

Quantum Gates

Like other simulators such as Intel-QS, this prototype only implements single-qubit gates and controlled two-qubit gates. The minimum list of quantum gates developed to implement the Quantum Fourier Transform algorithm are:

- Identity
- Hadamard
- ControlledPhaseShift

- ControlledNot

- Swap

All these quantum gates are implemented as two-dimensional double-precision arrays. This reduced set of quantum gates limits the simulation of algorithms that require additional gates. However, adding new single-qubit and controlled two-qubit gates is very easy. Just insert the corresponding matrix into the **quantumGate.cpp** source file.

Applying a Quantum Gate to a Quantum Register

Binh in [22] introduced the notation $\alpha_{*\ldots*j_k*\ldots*}$, where the asterisks refer to the bits that do not change while $j_k \in \{0, 1\}$. If we apply a quantum gate G_k to the $k-th$ qubit of this state vector we have the following result.

$$G_k \left|\Psi\right\rangle = \left|\Psi'\right\rangle = \begin{pmatrix} \alpha'_{0\ldots00} \\ \alpha'_{0\ldots01} \\ \vdots \\ \alpha'_{1\ldots10} \\ \alpha'_{1\ldots11} \end{pmatrix} \tag{3.2}$$

To obtain the vector $\left|\Psi'\right\rangle$, we use the equation 2.21, which implies that all state vector elements must be processed in pairs.

Some simulators, like **qiskit**, reverse the order of the qubits such that qubit 0 corresponds to the least significant bit of the binary representation of the state. In this case, the distance between $\alpha'_{*\ldots*0_k*\ldots*}$ and $\alpha'_{*\ldots*1_k*\ldots*}$ is equal to 2^k.

In this work, we maintain the natural order of the qubits. For example, in state $\left|011\right\rangle$, qubit 0 is the leftmost, qubit 1 is in the middle, and qubit 2 is the rightmost. Therefore, the distance between $\alpha'_{*\ldots*0_k*\ldots*}$ and $\alpha'_{*\ldots*1_k*\ldots*}$ is equal to $2^{(numQubits-1)-(k-th\ qubit)}$. To illustrate this, figure 3.3 shows the distance between the states of a 4-qubit state vector.

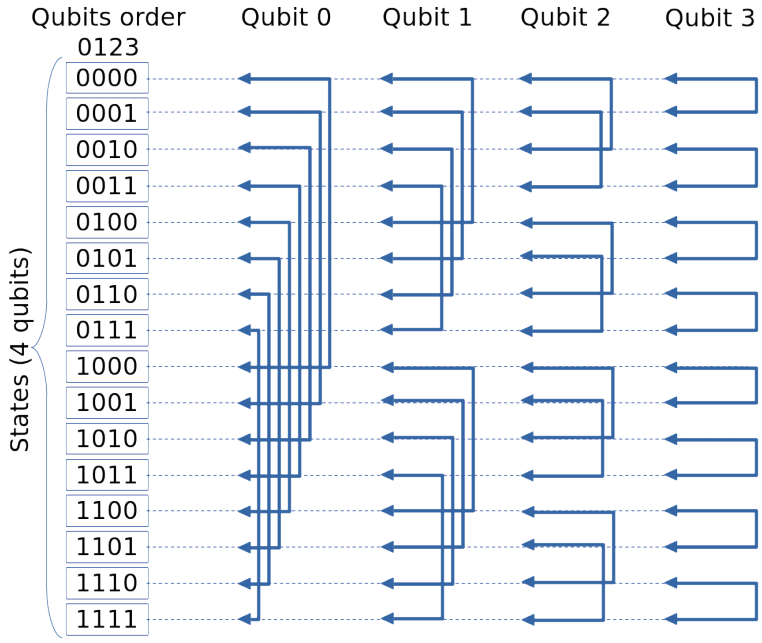


Figure 3.3: States pairs affected by qubit operation

Generally, a single-qubit gate can be applied to a quantum register performing the following pseudo-code.

```
for each amplitude in the state vector
do
   Calculate the new amplitudes for the current state.

        Find the pair corresponding to the current quantum state

   Calculate new values for amplitude of the new state.

done
```

In summary, calculating the amplitudes for the current state and the new affected state is done as follows: Determine the value of the current state's amplitude using equation 2.21. Then, find the pair corresponding to the current state, and finally, calculate the value of the latter using that same equation.

To find the pair corresponding to the current state, we can use two methods: the first calculates the distance using the relation $2^{(numQubits-1)-(k-th\ qubit)}$, as we explained before. The second method applies an XOR operation between the binary representation of the current state and a sequence of 0s with a 1 placed in the k-th position corresponding to the qubit we are working on. For example, applying a quantum gate on the 0th qubit on a for 4-qubits state $|0101\rangle$ we can find the corresponding pair using the following operation.

$$\begin{array}{r} 0101 \\ \oplus\ 1000 \\ \hline 1101 \end{array} \tag{3.3}$$

This result can be corroborated in figure 3.3. C++ offers binary operations to execute this operation efficiently and effortlessly.

```
unsigned int pos = numQubits - qubit - 1 ;
unsigned pairState = currentState ^ ((uint)1 << pos);
```

3.1.2 State Pruning

In the version of the simulator where the least probable states are pruned, we use a dynamic memory management because the states are stored non-sequentially in memory. This arrangement results from their computation via equation 2.21. Consequently, a state search method was developed to facilitate access to a specific state for calculations in subsequent iterations. However, performance is negatively impacted because a lot of time has to be spent searching for a state's values before they are used in a calculation. Figure 3.4 depicts the order of a 3-qubits state vector after applying a single-qubit gate (Hadamard) on the qubit 0.

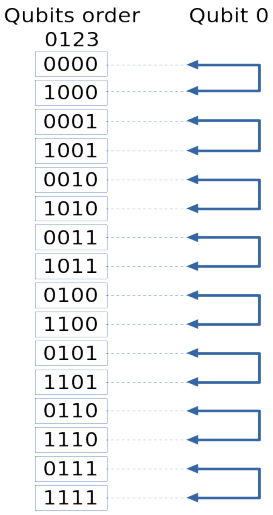


Figure 3.4: State vector order using dynamic memory approach

Because of this, the less probable states elimination approach was discarded early, therefore, we focus on pure states, which imply that the state vector contains the complete information about the quantum state; and this approach was adopted for the rest of this simulator's design.

3.1.3 Shared Memory Case

In order to improve performance, parallelizing the code is necessary. The first method is to apply a shared memory programming model. This was done using OpenMP.

We use **valgrind** to run program profiling and determine the sections of code that consume the most resources. Afterward, it was determined that the `QuantumRegister::applyGate` method is the component of the simulator where we had to focus on increasing performance. Figure 3.5 shows the profiling results.

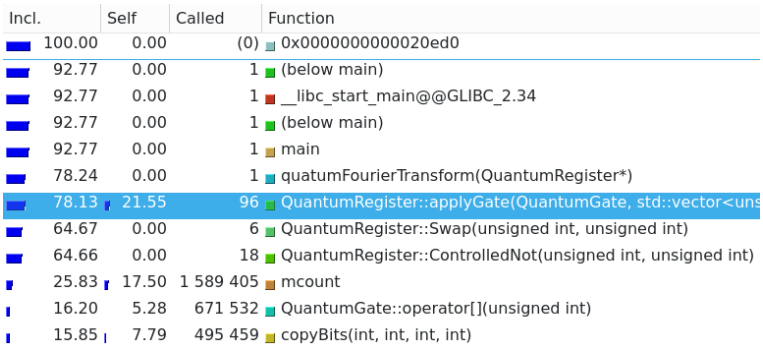


Figure 3.5: Simulator profiling

The `QuantumRegister::applyGate` method iterates through the state vector, implementing equation 2.21. To enhance performance, we partition the data and execute instructions on segments of the state vector, thereby speeding up the simulation. It is crucial to carefully manage the method's internal variables to prevent race conditions.

3.2 Distributed Memory Case

In the distributed memory model, the state vector needs to be divided among $numProcs$ processes. On the other hand, the equation 2.21, proposed in [22], indicates that the calculation of the amplitudes of the states must be done in pairs, therefore, we must guarantee that the number of amplitudes per process is even. To achieve this, we use the relationship

$$numProcs = \frac{2^{numQubits}}{2^m} \tag{3.4}$$

Where 2^m is the number of states per process. In this case we can face two cases:

- The pair corresponding to the current state is locally stored.
- The pair corresponding to the current state is located in other process. In this case it is neccessary to communicate values and results.

Figure 3.6 shows the pairwise calculation scheme for a 5-qubit state vector, applying each qubit. Partitioning with 2, 4, and 8 processes is also shown to visualize the communication process easily when we use the distributed programming model.

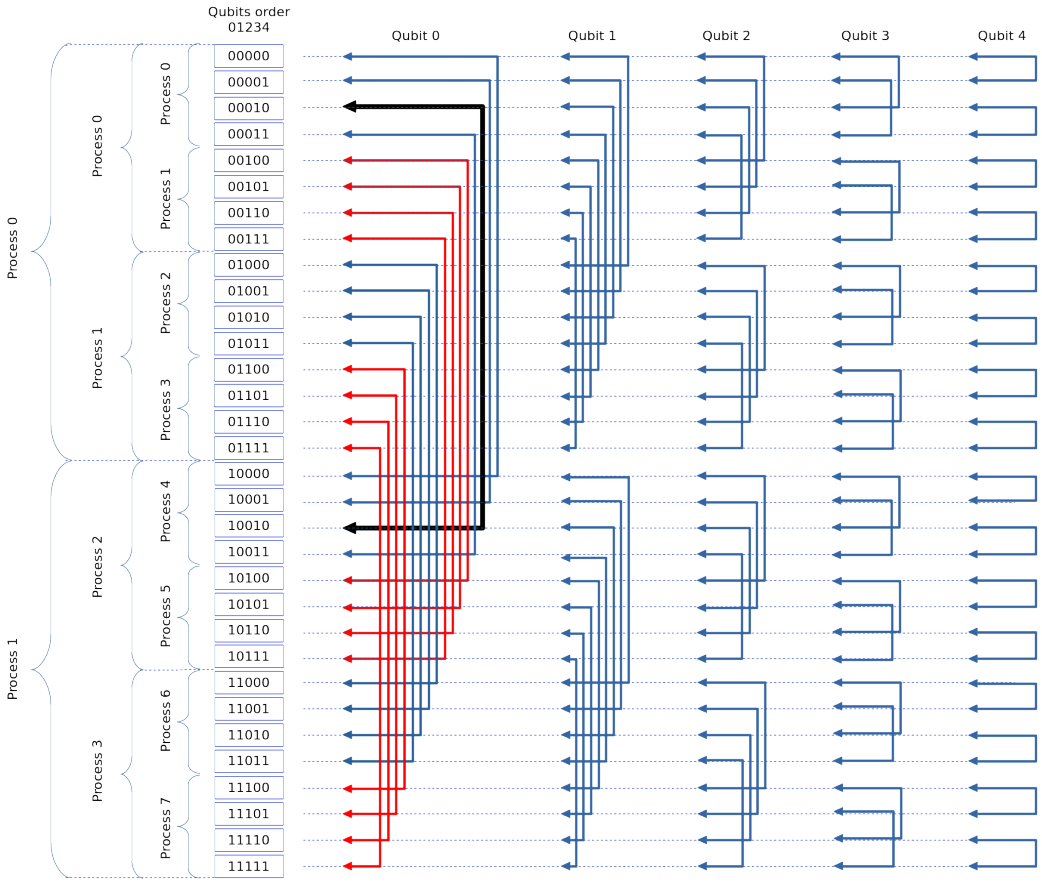


Figure 3.6: Pairwise calculation scheme for a 5-qubit state vector

For instance, consider performing a calculation on qubit 0 of the state $|00010\rangle$; the corresponding pair would be $|10010\rangle$. If two processes are used, communication should be established with process 1. If four processes are utilized, the remote process is process 2. Lastly, if eight processes are employed, the remote process will be process 4.

We use the following expression to calculate the process's identifier where the corresponding pair is located.

$$remoteProcID = \frac{pairState}{2^m} \tag{3.5}$$

In Figure 3.6, it is evident that for 2 processes, specifically regarding qubit 0, the number of communications required is $2^{\text{numQubits}/2}$. This substantially degrades performance. To mitigate the overhead caused by the extensive number of communications, the entire

segment of the state vector is exchanged between the peer processes involved, as outlined in equation 2.21. The calculations are then made locally, and the results are communicated back to the original process.

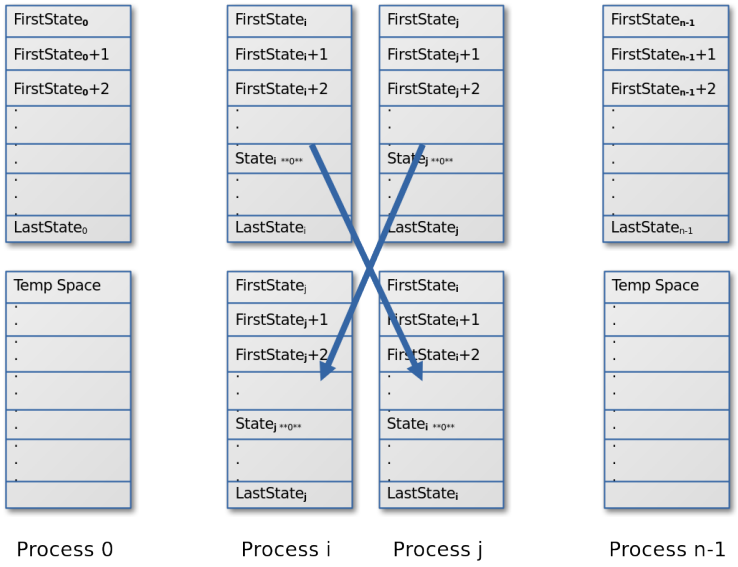


Figure 3.7: Data exchange between process

For this reason, we are unable to use the the total sum of local memory of each node to augment the number of qubits, and can only utilize half of the combined memory from all nodes. Figure 3.7 depicts this idea.

3.2.1 Compressing The State Vector

To compress the amplitude vector, we use the ZFP library [91] as it provides significant performance in accuracy and data size reduction. For the development of this prototype, the lossy compression method was selected to achieve a higher level of data size reduction. To go from a vector of amplitudes using traditional data types to a compressed vector, change the corresponding line in the **types.h** source file from `typedef std::vector<double> AmplitudesVector;` to `typedef zfp::array1<double> AmplitudesVector;` Of course, the corresponding header file from the ZFP library must be included.

Combining amplitude vector compression with amplitude vector distribution across multiple processes is an approach that can be effective both in terms of efficient memory usage and overall simulation performance. The version where a compressed vector is used to store the amplitudes was parallelized to achieve this. To obtain effective performance, the state vector portions are transmitted in a compressed manner. This makes communications faster because the message size is reduced.

To achieve compressed messaging, the compressed portions of the state vector must be serialized, and a custom MPI data type must be used in send and receive functions.

3.3 Simulator Verification

In order to validate the accuracy of our quantum simulator, we have excecuted different tests and compare the outputs with intel-qs and quantum++.

3.3.1 Quantum Gates Tests

To test the superposition principle we apply Hadamard gate to a quantum register of 4-qubits. The following quantum circuit was used:

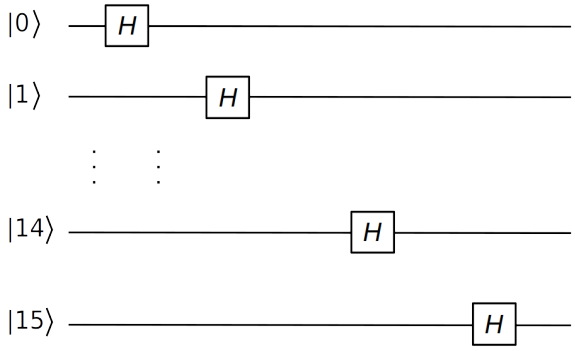


Figure 3.8: Test Quantum Circuit

The test was executed by initializing the first state with a probability equal to one, that is to say, $1 \times |0000\rangle$. Then, we repeat the experience with $1 \times |0001\rangle$ and so on until executing the test with the last state $1 \times |1111\rangle$.

The results of excecuting this quantum circuit with intel-qs, quantum++ and TMFQS were the same.

In this chapter, we review the details of a prototype implementation to provide a platform for testing the memory management strategies discussed in the previous chapter. Different scenarios were developed to facilitate testing.

In the next chapter, we present the results of the tests that implemented the memory management strategies. There, we will see the results taking into account not only the memory management but also the impact of each scenario and its performance.

Chapter 4

Results

This chapter presents the results of several quantum simulation tests performed using the prototype software quantum simulator developed in C++. These tests aim to evaluate the effectiveness of different memory management strategies discussed in chapter 2. By simulating fundamental quantum operations, we can assess how well these strategies reduce memory consumption and improve the efficiency of quantum computing simulations on classical hardware.

Throughout the chapter, we compare the simulator's performance with and without the proposed memory management techniques, highlighting the improvements achieved. By providing a comprehensive evaluation of these memory management strategies, this chapter aims to contribute to ongoing efforts to make quantum computing simulations more efficient and scalable, ultimately advancing the field of quantum computing.

4.1 Algorithm Selected for Testing

TMFQS was evaluated using the quantum Fourier transform, as in assessing the simulators presented previously in section 1.4.

4.2 Test platform

To run the simulations, we use two high-performance nodes from the scientific computing center of the Universidad Industrial de Santander (SC3-UIS) as described in section 1.4.1.

4.3 States Pruning

We can free up memory that is not needed by eliminating the quantum states that have the smallest chance of occurring, that is, eliminating those states whose amplitude is close to zero. However, in addition to all the points against this approach raised in the section 2.2,

this requires dynamic memory management, which introduces a lot of extra work because before applying a quantum gate to a state, we need to search for it in the array due the states are not ordered. That is to say, to apply every quantum gate, we need to execute 2^n search operations.

The quantum register contains all the states at the end of executing the Quantum Fourier transform algorithm. Due to the initial superposition process, the quantum register also has all the states in the first stage of Grover's algorithm. Therefore, this approach does not work well for these algorithms.

For these reasons, along with the risks outlined previously, we have decided to discard this approach because its numerous disadvantages outweigh its benefits.

4.4 Full-State Quantum Register

A quantum register with all the states arranged in a sequence can reduce the overhead of searching for quantum states. This also eliminates the need for an extra data structure to store the states and uses the indices of the amplitude vector to handle the quantum states. The total amount needed using this strategy $2^{numQubits} * 16$ Bytes for double precision floating point numbers. However, we save $2^{numQubits} * 4$ bytes, avoiding the state vector array.

The graph of figure 4.1 shows the performance of QFT applying state pruning (dynamic memory) vs full-state strategies. Simulations using dynamic memory involving more than 20 qubits were discarded due to their execution time exceeding one day.

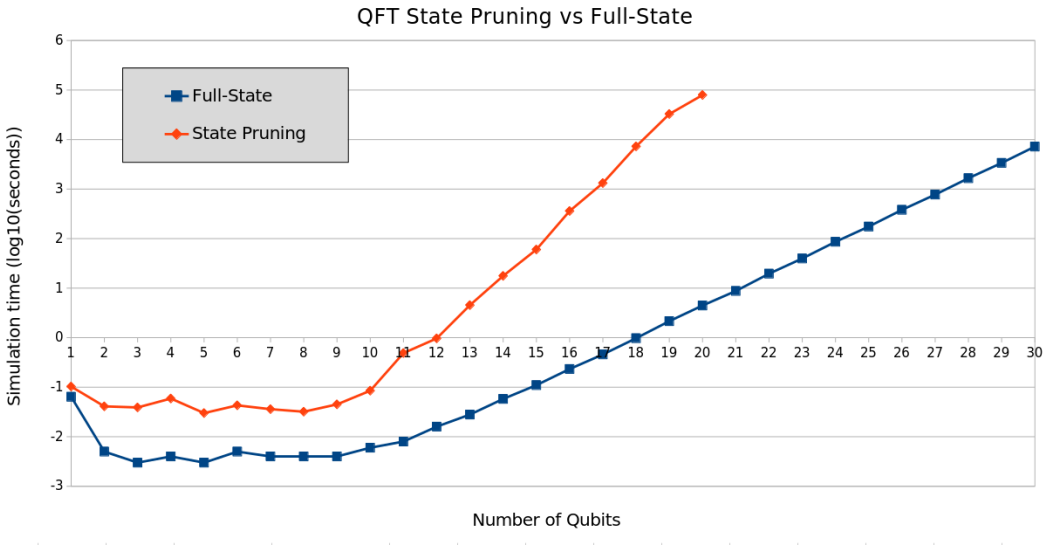


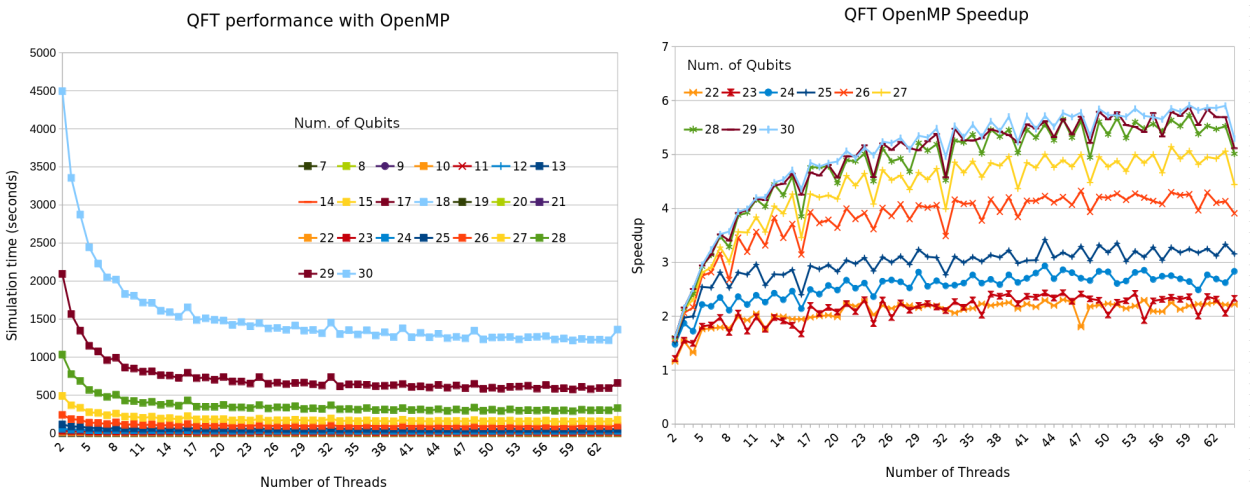
Figure 4.1: QFT Performance of Dynamic Memory Management vs Full-State Approach

The relatively constant execution time of up to 10 qubits in both approaches indicates

that the memory management strategies are efficient within this range. The full-state and dynamic memory approaches handle quantum states efficiently without requiring additional computational resources until the quantum state size becomes large enough to demand more from the memory system. This behavior suggests that both approaches' overhead and computational complexity scale similarly up to this point, maintaining constant execution time. Then, exponential growth is observed in both strategies from an 18-qubit state vector; however, this growth is more significant in the dynamic memory approach. This is a consequence of a substantial increase in the overhead introduced by searching the state every time we need to use it.

As can be seen by comparing these strategies, the workload overhead using dynamic memory is significant.

We have parallelized the full-state version to increase performance using the shared memory model with OpenMP.



(a) QFT Full State with MPI

(b) QFT Full State MPI Speedup

Figure 4.2: QFT Distributed Memory Performance

The graph in figure 4.2a shows the simulation time using OpenMP for parallelization. As the number of threads increases beyond a certain point (around 32 threads for most qubit counts), the simulation time does not decrease significantly. No substantial benefit from parallelism is observed for a small number of qubits. The 30-qubit line shows the longest execution time across all thread counts. The graph in figure 4.2b shows the speedup of OpenMP to observe the benefit of parallelism. Here, we show only the speedups for the bigger number of qubits because parallelism does not significantly benefit small numbers of qubits. For all qubit sizes, speedup generally increases with the number of threads, reflecting the scalability of the implementation. The speedup flattens after approximately 32 threads. Larger qubit sizes (e.g., 28–30) achieve higher speedup compared to smaller ones (e.g., 22–24). This is because larger problems offer more computational work, allowing

better utilization of parallel threads.

4.5 Data Compression

We have selected one of the most widely used C++ libraries for data compression, ZFP, to test this approach. We modified the full-state version of the simulator to compress the amplitude vector. The graph of figure 4.3 shows the performance comparison between full-state vs full-state using ZFP.

We can observe that the overhead introduced by the compression and decompression procedure is significant.

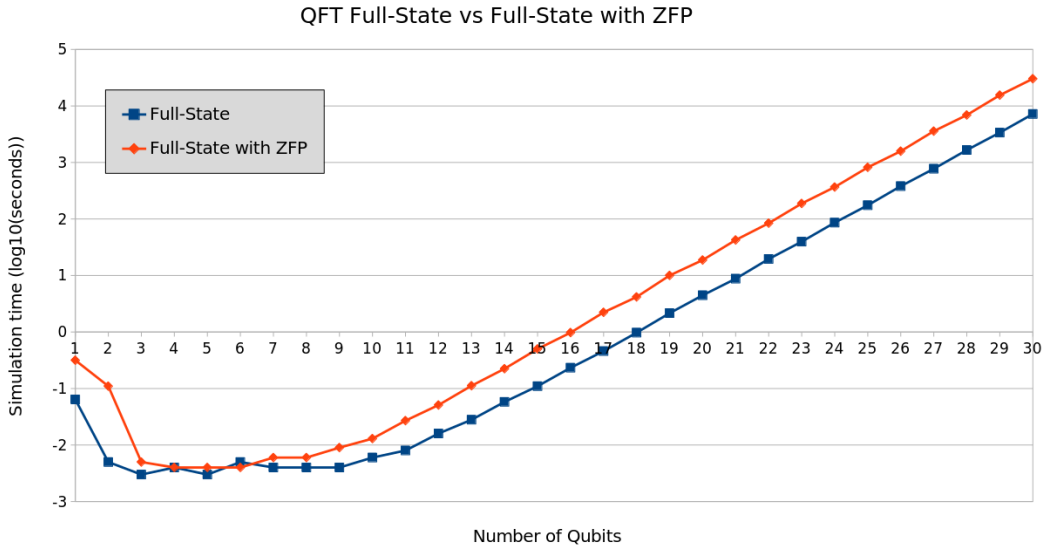


Figure 4.3: QFT Full State with ZFP (log_{10})

The graph of figure 4.4 shows the amount of memory used by both simulator versions. We can observe that the compression approach is highly efficient. This enables the possibility of increasing the number of qubits in simulations. The difference in memory usage between the full-state and full-state with ZFP compression is due to the fixed-rate compression mode of ZFP. This mode was specifically chosen because it ensures a predictable and consistent compression ratio, which is crucial in scenarios where storage constraints are critical. In the fixed-rate mode, ZFP maintains a constant number of bits per value, independent of the actual number of qubits. This results in a consistent compression percentage across different qubit counts, as observed in this graph. The choice of fixed-rate compression is ideal for use cases where the available memory or storage must be carefully managed, as it allows for precise control over the amount of data being stored. While the compression percentage remains constant, the total memory savings increase with the

number of qubits because the absolute size of the quantum state grows exponentially. This highlights the effectiveness of ZFP's fixed-rate mode in reducing memory usage in large-scale quantum simulations, ensuring predictable memory savings in memory-constrained environments.

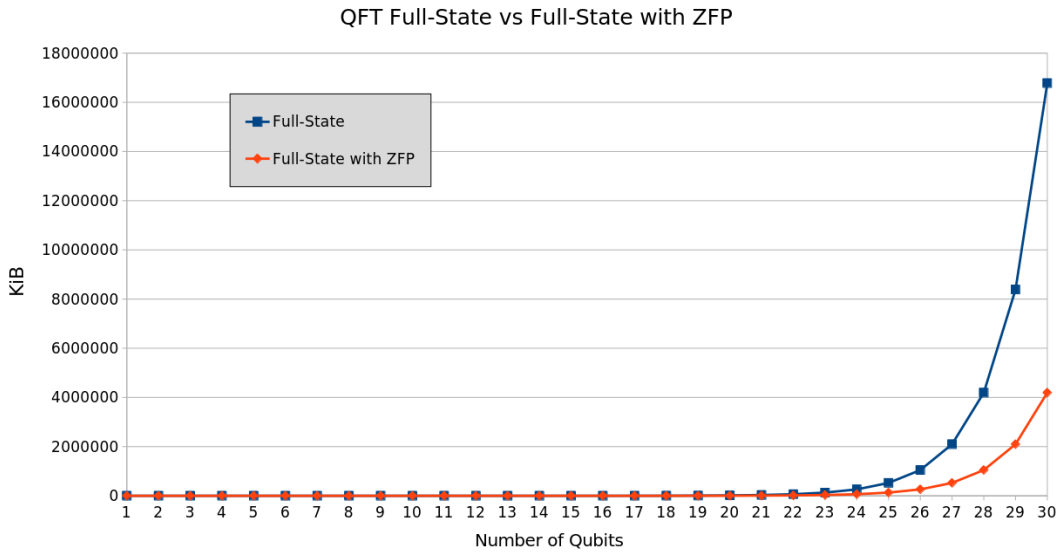


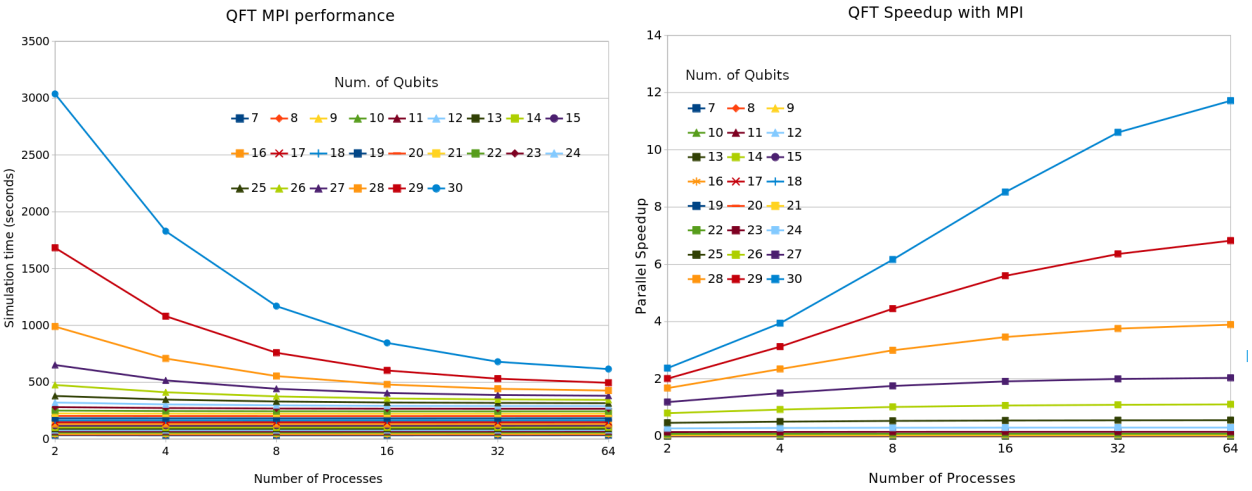
Figure 4.4: QFT Full State with ZFP Data Size

4.6 Distribute the Quantum Register Across Multiple Nodes

We have developed a simulation version employing MPI to increase memory capacity by leveraging the RAM of additional computer nodes. To uphold computational efficiency, it is essential to underscore the necessity of maintaining an optimal ratio between the number of processes and the allocation of qubits per process. That is, preserve the relation of the equation 3.4.

The graph shown in Figure 4.5a demonstrates the performance of the Quantum Fourier Transform across a range of qubit counts from 7 to 30, using 2, 4, 8, 16, 32, and 64 processes. The reason for starting at seven qubits is to preserve the relation described in equation 3.4.

In addition to achieving better performance, we can see that by increasing the number of processes, we can increase the number of qubits and reduce the size of messages required to exchange partial results between processes. We can see also that parallelism is helpful for a number greater than 26 qubits.



(a) QFT Full State with MPI (b) QFT Full State MPI Speedup

Figure 4.5: QFT Distributed Memory Performance

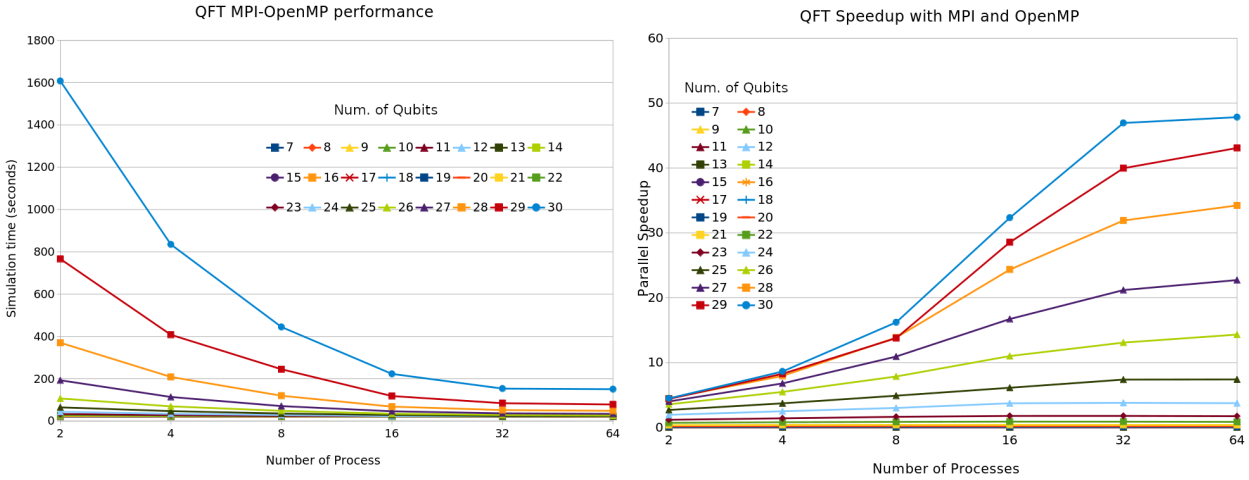
As we can see in the graph 4.6b, we obtain a more significant acceleration as the number of qubits increases. This behavior starts from qubit 25, where an acceleration of 2 is obtained for 64 processes. In the case of 30 qubits, the acceleration is 12 when the simulation is performed with 64 processes. We observe an almost linear acceleration at low process counts (from 1 to 32 processes). At 64 processes, the acceleration starts to slow down.

4.7 Combination of Distributed Memory and Shared Memory Approaches

We have developed a simulator version that combines MPI with OpenMP to achieve better performance. The graph in figure 4.6a illustrates the performance of Quantum Fourier Transform using this hybrid approach.

Comparing the results of the graphs in Figures 4.5a and 4.6a, we see that the combination of MPI and OpenMP increases the performance, especially for cases where the size of the state vector portion at each node is large.

In the case of the hybrid shared memory and distributed memory approach, we see in Figure 4.6b that the speedup is much higher than the speedup obtained with distributed memory alone. Here, we also get a significant speedup for 25 qubits or more; for example, in the case of 30 qubits, the speedup is close to 50. At 32 processes, the speedup curve begins to flatten, approaching an asymptotic limit. Despite increasing the process count, the speedup gain becomes minimal.



(a) QFT Full State with MPI-OpenMP (b) QFT Full State MPI-OpenMP Speedup

Figure 4.6: Shared Memory and Distributed Memory Parallelism

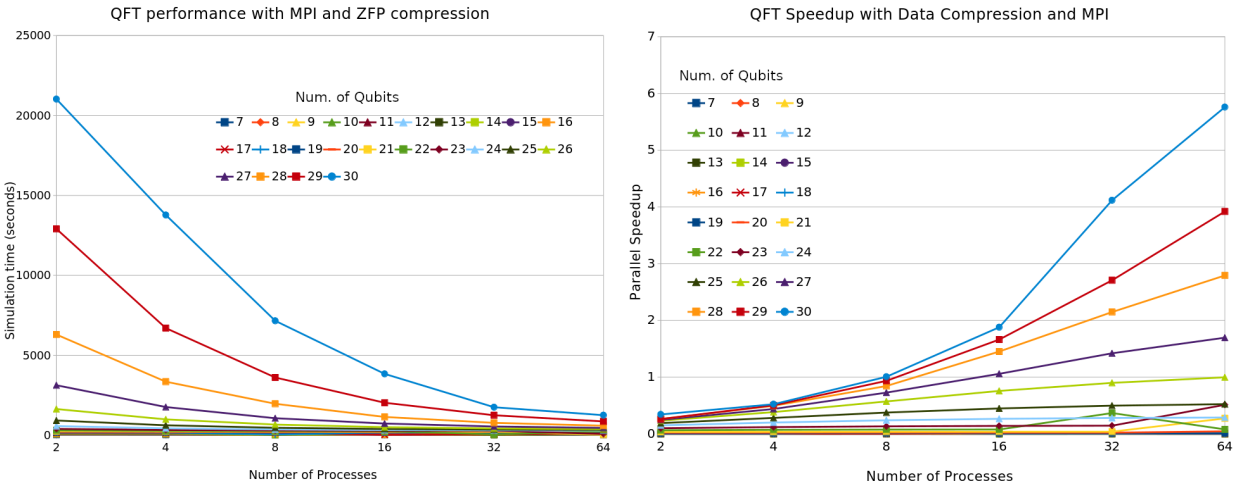
4.8 Combination of Distributed Memory and Data Compression Approaches

Taking advantage of distributed resources to have more memory available, combined with data compression, makes it possible to perform simulations with a larger number of qubits. We have already seen that there is a processing overhead introduced by the compression process, however, the transmission of compressed data contributes positively to overall performance.

We have modified the prototype to test this approach. Figure 4.7a shows the performance of the distributed memory approach with data compression, for a range of 7 to 30 qubits with a variation in the number of processes equal to 2, 4, 8, 16, 32, 64. Recall that the selection of the number of processes obeys the relation of the equation 3.4.

In this graph, we can see that performance has decreased; however, the reduction of the required memory is significant because the same strategy used in the section on data compression is adopted here. It has to be pointed out that this strategy is valid only if portions of the quantum register are transmitted in a compressed form.

In Figure 4.7b we can see that the speedup is not as significant as in the previous cases. For 30 qubits, a speedup close to 6 is achieved using 64 processes. However, the gain does not diminish starting from 32 processes, as in the previous speedup graphs.



(a) QFT Full State with MPI-ZFP (b) QFT Full State with MPI-ZFP Speedup

Figure 4.7: QFT Full State with MPI with Data Compression

4.9 Quantum Simulators Comparison

To validate the results obtained with our prototype, a comparison is made with other simulators. First, specific conditions must be established to allow a fair comparison of the simulators studied. The common conditions were using a single computing node with a shared memory model. Simulators that use GPUs are excluded because their performance is much higher than the others, but their scaling is limited. The case of using distributed memory is also excluded because only some include this capability.

The graph in Figure 4.8 shows the performance of the quantum Fourier transform for intel-qs, quantum++, QuEST, and TMFQS. The selected simulators use the C++ complex numbers data type for memory management. They use a full-state vector scheme, which allows better performance but does not reduce memory consumption.

As can be seen in the graph in Figure 4.8, the Intel-QS simulator performs lower than the other simulators. QuEST exhibited the best performance. Our prototype TMFQS performs acceptably compared to these mature tools that have been optimized, for example, by using libraries such as MKL in the case of Intel-QS.

It has to be pointed out that the open-source simulators studied only use the special method for applying quantum gates to a quantum register discussed in section 2.4.1 and the full-state vector representation.

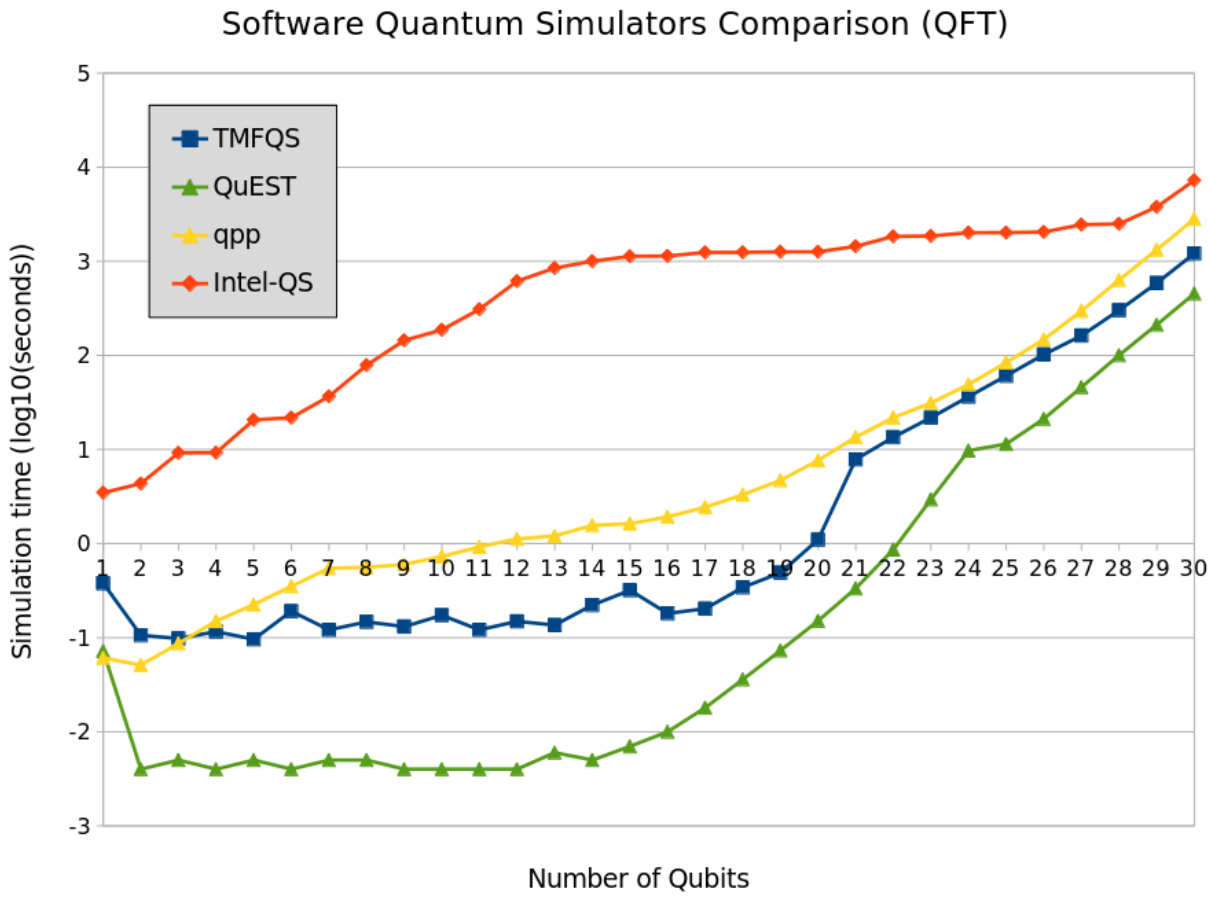


Figure 4.8: Quantum Simulators Comparison

After testing the strategies studied in this work, described in detail in the previous chapter, we can comment that the combination of feasible approaches contributes to adequate memory management and achieves reasonable performance. This consists of working with a full-state vector in a compressed way and using distributed memory to provide the possibility of using a more significant number of qubits. On the other hand, although eliminating the least probable states promises a significant reduction in memory usage, it presents many drawbacks, including a negative impact on the reliability of the results and the performance of the simulations.

In the conclusions chapter, the most relevant considerations are made after the completion of this study.

Chapter 5

Conclusions

This chapter summarizes the most important research findings and discusses their implications. It also presents the study's limitations and suggests possible directions for future research.

Upon reviewing some of the most prominent open-source simulators in the field, we have identified several approaches to achieve reasonable performance. These include shared memory parallelism, distributed memory, and the use of GPUs. Among these, GPUs offer the best performance. However, the memory limitations of current GPUs restrict the number of qubits that can be managed. In contrast, using distributed memory allows access to more memory resources, enabling simulations with a higher number of qubits. However, none of them perform data compression to reduce memory consumption. Our proposal about combining distributed memory with data compression represents a critical point in memory consumption.

The objective of this work was to establish a memory management method that would allow the efficient representation of the fundamental elements of the quantum circuit model. In this sense, the most relevant considerations of this study are shown below.

1. **State Pruning:** A notable strategy explored in this study is removing the least probable quantum states. Although this approach could significantly reduce memory consumption by discarding states with amplitudes near zero, this technique introduces important drawbacks, such as potential fidelity loss and reduced accuracy of quantum algorithms. Careful consideration of the error introduced and the impact on algorithm performance is essential when applying this method. Besides, the inherent nature of certain quantum algorithms, like Grover's search and Quantum Fourier Transform, requires constructing the entire state vector in the initial simulation phases. On the other hand, this approach involves dynamic memory allocation, which disrupts the order of state and amplitude vectors, resulting in significant overhead and reduced performance. Consequently, initiating simulations with a fully loaded quantum register in memory proves superior, markedly enhancing overall performance.

2. **Data Structure Optimization:** Implementing optimized data structures has proven critical for memory conservation. The transition from traditional binary state representation to more suitable data structures improve the memory usage. This optimization supports larger simulations by reducing the memory footprint per qubit state. Strategic selection of appropriate data structures enables memory savings by avoiding storing quantum register states, which would require substantial memory ($2^{Qubits} \times 4$ Bytes with unsigned integers).

3. **Special Method to Apply Quantum Gates:** The method detailed in [22] for applying a quantum gate to a quantum register plays a crucial role in memory conservation, steering clear of matrices sized at $2^{Qubits} * 2^{Qubits} \times 2 \times 8$ Bytes when employing double-precision floating-point data types. This represents one of the main strategies to improve the memory usage.

4. **Compression Techniques:** The integration of compression techniques, specifically the use of ZFP within quantum simulation frameworks, has introduced memory savings of up to 75% in certain cases. Although this introduces additional computational overhead, the trade-off is justifiable for long-term simulations where memory resources are a limiting factor.

5. **Parallel Computing Frameworks:** The adaptation of simulations to distributed computing frameworks like MPI and OpenMP has dispersed the memory load effectively across multiple nodes. This scalability shows that the hybrid use of these technologies can lead to improvements in simulation times and reduction in memory overhead.

6. **Recommended Strategy:** The best approach is to implement a compressed full-state vector distributed across multiple computing nodes, where communications are based on compressed messages. Parallelism is the mechanism to mitigate the overhead introduced by the compression process.

During the development of this work, several articles were published to disseminate key findings and contributions to the field of memory management in quantum computing simulators. The first article, published in Dyna journal [98], describes how to model the main elements of quantum computing in a classical computer and depicts resource consumption using two popular quantum simulators. The second article, included in a Springer conference proceedings [99], shows a survey of different implementations to simulate quantum computing supported by classical computing, highlighting important considerations for implementing and developing solutions. The third article, also included in a Springer conference proceedings (reference is not available yet), proposes several strategies to save memory to increase the number of qubits in the simulations. These publications represent significant milestones in advancing efficient resource management for quantum computing

simulators, offering valuable contributions to both academic and applied quantum computing communities.

5.1 Further Work

While this thesis has explored several aspects of memory management in software quantum computing simulators, there are numerous areas for further research.

- **Advanced Memory Compression Techniques**: Investigating more sophisticated data compression algorithms tailored specifically for quantum state vectors, and exploring hybrid compression techniques combining lossy and lossless methods to balance between accuracy and memory usage.
- **Optimization of Quantum Gate Application**: Developing efficient algorithms for applying higher-order multi-qubit gates.
- **Error Mitigation and Fault Tolerance**: Develop strategies for incorporating error mitigation techniques within the simulators to emulate real quantum hardware better.
- **Distributed Quantum Computing**: Developing protocols for efficient communication and synchronization between distributed nodes to minimize overhead.
- **Energy-efficient Quantum Simulations**: Researching into energy-efficient memory management techniques to reduce the power consumption of large-scale quantum simulations.

By addressing these topics, future research can further enhance the efficiency, scalability, and practicality of software quantum computing simulators, bringing them closer to the capabilities required for real-world quantum computing applications.

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication," *BELL SYST TECH*, vol. 27, no. 3, pp. 379–423, Jul. 1948. [Online]. Available: http://bstj.bell-labs.com/BSTJ/images/Vol27/bstj27-3-379.pdf

[2] M. Born and P. Jordan, "Zur quantenmechanik," *Zeitschrift für Physik*, vol. 34, no. 1, pp. 858–888, Dec 1925. [Online]. Available: https://doi.org/10.1007/BF01328531

[3] R. P. Feynman, "Feynman and computation," A. J. G. Hey, Ed. Cambridge, MA, USA: Perseus Books, 1999, ch. There's Plenty of Room at the Bottom, pp. 63–76. [Online]. Available: http://dl.acm.org/citation.cfm?id=304763.305682

[4] A. S. Holevo, "Bounds for the quantity of information transmitted by a quantum communication channel," *Problems of Information Transmission*, vol. 9, no. 3, pp. 177–183, 1973. [Online]. Available: http://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=ppi&paperid=903&option_lang=eng

[5] R. P. Poplavskiĭ, "Thermodynamic models of information processes," *Soviet Physics Uspekhi*, vol. 18, no. 3, pp. 222–241, mar 1975. [Online]. Available: https://doi.org/10.1070%2Fpu1975v018n03abeh001955

[6] R. Ingarden, *Quantum Information Theory*, ser. Preprint - Instytut Fizyki Uniwersytetu Mikołaja Kopernika. PWN, 1975. [Online]. Available: https://books.google.com.co/books?id=7CYhtwAACAAJ

[7] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines," *Journal of Statistical Physics*, vol. 22, no. 5, pp. 563–591, May 1980. [Online]. Available: https://doi.org/10.1007/BF01011339

[8] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," in *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing*, India, 1984, p. 175.

[9] D. Deutsch, "Quantum theory, the church-turing principle and the universal quantum computer," vol. 400, pp. 97–117, 1985.

[10] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, Oct. 1997. [Online]. Available: http://dx.doi.org/10.1137/S0097539795293172

[11] R. Shaydulin, H. Ushijima-Mwesigwa, C. Negre, I. Safro, S. Mniszewski, and Y. Alexeev, "A hybrid approach for solving optimization problems on small quantum computers," *Computer*, vol. 52, pp. 18–26, 06 2019.

[12] M. Fingerhuth, "Open-source quantum software projects," https://github.com/qosf/os_quantum_software, March 2019.

[13] Q. C. Report, "Qbit count," https://quantumcomputingreport.com/scorecards/qubit-count/, January 2019.

[14] Quantiki, "List of qc simulators," https://www.quantiki.org/wiki/list-qc-simulators, February 2019.

[15] J. Chen, F. Zhang, C. Huang, M. Newman, and Y. Shi, "Classical simulation of intermediate-size quantum circuits," 2018.

[16] A. Dang, C. D. Hill, and L. C. L. Hollenberg, "Optimising Matrix Product State Simulations of Shor's Algorithm," *Quantum*, vol. 3, p. 116, Jan. 2019. [Online]. Available: https://doi.org/10.22331/q-2019-01-25-116

[17] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, p. 1484–1509, Oct 1997. [Online]. Available: http://dx.doi.org/10.1137/S0097539795293172

[18] X.-C. Wu, S. Di, F. Cappello, H. Finkel, Y. Alexeev, and F. T. Chong, "Memory-efficient quantum circuit simulation by using lossy data compression," 2018.

[19] R. Eleanor and P. Wolfgang, *Quantum Computing, A Gentle Introduction.* The MIT Press, 2011.

[20] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, ser. CGO 2018. New York, NY, USA: ACM, 2018, pp. 113–125. [Online]. Available: http://doi.acm.org/10.1145/3168822

[21] T. Häner and D. S. Steiger, "0.5 petabyte simulation of a 45-qubit quantum circuit," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: ACM, 2017, pp. 33:1–33:10. [Online]. Available: http://doi.acm.org/10.1145/3126908.3126947

[22] D. B. Trieu, "Large-Scale Simulations of Error-Prone Quantum Computation Devices," Dr. (Univ.), Universität Wuppertal, Jülich, 2009, record converted from VDB: 12.11.2012; Universität Wuppertal, Diss., 2009. [Online]. Available: https://juser.fz-juelich.de/record/7578

[23] M. Planck, "Zur Theorie des Gesetzes der Energieverteilung im Normalspektrum. (German) [On the theory of distribution of energy in the normal spectrum]," *j-VERH-DTSCH-PHYS-GES*, vol. 2, no. 17, pp. 237–245, Dec. 1900. [Online]. Available: http://web.ihep.su/dbserv/compas/src/planck00b/eng.pdf;http://web.ihep.su/owa/dbserv/hw.part2?s_c=PLANCK+1900B;https://hdl.handle.net/2027/coo.31924056107224?urlappend=%3Bseq=551

[24] A. Einstein, "Über einen die Erzeugung und Verwandlung des Lichtes betreffenden heuristischen Gesichtspunkt. (German) [On the production and transformation of light from a heuristic viewpoint]," *j-ANN-PHYS-1900-4*, vol. 322, no. 6, pp. 132–148, 1905. [Online]. Available: http://www.gsjournal.net/Science-Journals/Essays/View/2490;http://www.gsjournal.net/Science-Journals/Essays/View/2491;http://www.zbp.univie.ac.at/einstein/einstein1.pdf

[25] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, 10th ed. New York, NY, USA: Cambridge University Press, 2011.

[26] R. V. L. Hartley, "Transmission of information," *The Bell System Technical Journal*, vol. 7, no. 3, pp. 535–563, July 1928.

[27] R. V. L. Hartley, "Transmission of information," *Bell System Technical Journal*, vol. 7, pp. 535–563, 1928.

[28] H. Barnum, S. Wehner, and A. Wilce, "Introduction: Quantum information theory and quantum foundations," *Foundations of Physics*, vol. 48, no. 8, pp. 853–856, Aug 2018. [Online]. Available: https://doi.org/10.1007/s10701-018-0188-6

[29] Wikipedia, "Quantum Computing," https://en.wikipedia.org/wiki/Quantum_computing, Jan 2019.

[30] J. A. Bergou and M. Hillery, *Introduction to the Theory of Quantum Information Processing*. Springer Publishing Company, Incorporated, 2013.

[31] P. H. Artur Ekert and H. Inamori, "Basic concepts in quantum computation," *Coherent atomic matter waves*, pp. 661–701, Jan. 2001.

[32] N. D. Mermin, *Quantum Computer Science: An Introduction*. Cambridge University Press, 2007.

[33] B. Shoshany, "In layman's term, what is a quantum state?" https://www.quora.com/In-laymans-term-what-is-a-quantum-state, April 2018.

[34] C. P. Williams, *Explorations in Quantum Computing, Second Edition*, ser. Texts in Computer Science. Springer, 2011. [Online]. Available: https://doi.org/10.1007/978-1-84628-887-6

[35] I. Research and the IBM QX team, "General questions about quantum information science," https://quantumexperience.ng.bluemix.net/qx/tutorial?sectionId=full-user-guide&page=000-FAQ~2F000-Frequently_Asked_Questions, April 2017.

[36] M. Fingerhuth, T. Babej, and P. Wittek, "Open source software in quantum computing," *PLOS ONE*, vol. 13, no. 12, pp. 1–28, 12 2018. [Online]. Available: https://doi.org/10.1371/journal.pone.0208561

[37] Wikipedia, "Qubits Physical implementations," https://en.wikipedia.org/wiki/Qubit#Physical_implementations, March 2019.

[38] Imanuel, "What is quantum computing? top 18 quantum computing companies," https://www.predictiveanalyticstoday.com/what-is-quantum-computing/, February 2018.

[39] Wikipedia, "Quantum Logic Gate," https://en.wikipedia.org/wiki/Quantum_logic_gate, March 2019.

[40] D. E. Deutsch and R. Penrose, "Quantum computational networks," *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 425, no. 1868, pp. 73–90, 1989. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1989.0099

[41] Wikipedia, "Quantum Computing," https://en.wikipedia.org/wiki/Quantum_algorithm, Jan 2019.

[42] D. Deutsch, "Quantum theory, the church-turing principle and the universal quantum computer," *Proceedings of the Royal Society of London*, vol. 400, pp. 97–117, 1985.

[43] S. M. Girvin, "Basic concepts in quantum information," 2013.

[44] M. Mosca, "Quantum algorithms," 2008.

[45] T. S. Humble and E. P. DeBenedictis, "Quantum realism," *Computer*, vol. 52, no. 6, pp. 13–17, June 2019.

[46] S. Jordan, "Quantum Algorithm Zoo," https://quantumalgorithmzoo.org/, Jun 2018.

[47] A. Montanaro, "Quantum algorithms: an overview," 2015.

[48] C. H. Papadimitriou, *Computational complexity.* Addison-Wesley, 1994.

[49] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," University of Bristol, Bristol, UK, UK, Tech. Rep., 1992.

[50] E. Rosinger, "Basics of quantum computation (part 1)," *arXiv: Quantum Physics*, 2004.

[51] R. Shaydulin, H. Ushijima-Mwesigwa, I. Safro, S. Mniszewski, and Y. Alexeev, "Community detection across emerging quantum architectures," 2018.

[52] ——, "Network community detection on small quantum computers," 2018.

[53] I. Karafyllidis, G. C. Sirakoulis, and P. Dimitrakis, "Representation of qubit states using 3d memristance spaces: A first step towards a memristive quantum simulator," in *Proceedings of the 14th IEEE/ACM International Symposium on Nanoscale Architectures*, ser. NANOARCH '18. New York, NY, USA: ACM, 2018, pp. 163–168. [Online]. Available: http://doi.acm.org/10.1145/3232195.3232197

[54] Q. O. S. F. Team, "Quantum open source foundation," https://qosf.org/, April 2019.

[55] K. De Raedt, K. Michielsen, H. De Raedt, B. Trieu, G. Arnold, M. Richter, T. Lippert, H. Watanabe, and N. Ito, "Massively parallel quantum computer simulator," *Computer Physics Communications*, vol. 176, no. 2, p. 121–136, Jan 2007. [Online]. Available: http://dx.doi.org/10.1016/j.cpc.2006.08.007

[56] M. Smelyanskiy, N. P. D. Sawaya, and A. Aspuru-Guzik, "qhipster: The quantum high performance software testing environment," *CoRR*, vol. abs/1601.07195, 2016.

[57] R. Li, B. Wu, M. Ying, X. Sun, and G. Yang, "Quantum supremacy circuit simulation on sunway taihulight," 2018.

[58] R. Li, B. Wu, M. Ying, X. Sun, and G. Yang, "Quantum supremacy circuit simulation on sunway taihulight," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 805–816, 2020.

[59] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, "Characterizing quantum supremacy in near-term devices," *Nature Physics*, vol. 14, no. 6, p. 595–600, Apr 2018. [Online]. Available: http://dx.doi.org/10.1038/s41567-018-0124-x

[60] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, and H. Neven, "Simulation of low-depth quantum circuits as complex undirected graphical models," 2017.

[61] R. Orús, "A practical introduction to tensor networks: Matrix product states and projected entangled pair states," *Annals of Physics*, vol. 349, pp. 117 – 158, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0003491614001596

[62] G. Vidal, "Classical simulation of infinite-size quantum lattice systems in one spatial dimension," *Phys. Rev. Lett.*, vol. 98, p. 070201, Feb 2007. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.98.070201

[63] U. Schollwöck, "The density-matrix renormalization group in the age of matrix product states," *Annals of Physics*, vol. 326, no. 1, pp. 96 – 192, 2011, january 2011 Special Issue. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0003491610001752

[64] D. S. Wang, C. D. Hill, and L. C. L. Hollenberg, "Simulations of shor's algorithm using matrix product states," 2015.

[65] G. Vidal, "Efficient classical simulation of slightly entangled quantum computations," *Phys. Rev. Lett.*, vol. 91, p. 147902, Oct 2003. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.91.147902

[66] T. Häner, D. S. Steiger, M. Smelyanskiy, and M. Troyer, "High performance emulation of quantum circuits," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2016, pp. 866–874.

[67] T. Häner, D. S. Steiger, K. Svore, and M. Troyer, "A software methodology for compiling quantum programs," *Quantum Science and Technology*, vol. 3, no. 2, p. 020501, Feb 2018. [Online]. Available: http://dx.doi.org/10.1088/2058-9565/aaa5cc

[68] R. LaRose, "Overview and Comparison of Gate Level Quantum Software Platforms," *Quantum*, vol. 3, p. 130, Mar. 2019. [Online]. Available: https://doi.org/10.22331/q-2019-03-25-130

[69] V. Guzik, S. Gushanskiy, M. Polenov, and V. Potapov, "Models of a quantum computer, their characteristics and analysis," in *2015 9th International Conference on Application of Information and Communication Technologies (AICT)*, Oct 2015, pp. 583–587.

[70] Q. Insider, "Top 63 quantum computer simulators for 2024," https://thequantuminsider.com, 2024, accessed: 2024-05-12.

[71] G. G. Guerreschi, J. Hogaboam, F. Baruffa, and N. Sawaya, "Intel quantum simulator: A cloud-ready high-performance simulator of quantum circuits," 2020.

[72] V. Gheorghiu, "Quantum++: A modern c++ quantum computing library," 2014.

[73] Q. A. team and collaborators, "qsim," Sep. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4023103

[74] T. Jones, A. Brown, I. Bush, and S. C. Benjamin, "Quest and high performance simulation of quantum computers," *Scientific Reports*, vol. 9, no. 1, p. 10736, 2019. [Online]. Available: https://doi.org/10.1038/s41598-019-47174-9

[75] D. Strano, "Qrack," https://vm6502q.readthedocs.io/en/latest/, April 2019.

[76] D. Strano, B. Bollay, A. Blaauw, N. Shammah, W. J. Zeng, and A. Mari, "Exact and approximate simulation of large quantum circuits on a single gpu," 2023.

[77] J. T. Iosue, "Quantum computer simulator with algorithms," https://github.com/jtiosue/Quantum-Computer-Simulator-with-Algorithms, 2022.

[78] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018. [Online]. Available: https://doi.org/10.22331/q-2018-08-06-79

[79] T. Jones, J. D. Whitfield, P. L. McMahon, M.-H. Yung, and A. Aspuru-Guzik, "Efficient quantum algorithms for simulating sparse hamiltonians," *Quantum Information & Computation*, vol. 10, no. 3, pp. 241–272, 2010. [Online]. Available: https://arxiv.org/abs/0911.3895

[80] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, M. Kieferová, I. D. Kivlichan, T. Menke, B. Peropadre, N. P. D. Sawaya, S. Sim, L. Veis, and A. Aspuru-Guzik, "Quantum algorithms for quantum chemistry and quantum computing," in *Chemical Reviews*, 2019. [Online]. Available: https://doi.org/10.1021/acs.chemrev.8b00803

[81] J. Zeng, O. Higgott, T. Booth, B. Green, and A. Gilliam, "Quantum error mitigation: A scalable approach to reducing errors in quantum computations," *PRX Quantum*, vol. 2, no. 4, p. 040342, 2021. [Online]. Available: https://doi.org/10.1103/PRXQuantum.2.040342

[82] M. Smelyanskiy, N. P. D. Sawaya, and A. Aspuru-Guzik, "qhipster: The quantum high performance software testing environment," *CoRR*, vol. abs/1601.07195, 2016.

[83] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[84] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, p. 520–540, jun 1987. [Online]. Available: https://doi.org/10.1145/214762.214771

[85] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[86] X.-C. Wu, S. Di, E. M. Dasgupta, F. Cappello, H. Finkel, Y. Alexeev, and F. T. Chong, "Full-state quantum circuit simulation by using data compression," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2019. [Online]. Available: http://dx.doi.org/10.1145/3295500.3356155

[87] S. Di and F. Cappello, "Fast error-bounded lossy hpc data compression with sz," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016, pp. 730–739.

[88] D. Tao, S. Di, Z. Chen, and F. Cappello, "Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization," *CoRR*, vol. abs/1706.03791, 2017. [Online]. Available: http://arxiv.org/abs/1706.03791

[89] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello, "Error-controlled lossy compression optimized for high compression ratios of scientific datasets," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 438–447.

[90] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, 2006.

[91] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[92] J. Clyne, P. Mininni, A. Norton, and M. Rast, "Interactive desktop analysis of high resolution simulations: application to turbulent plume dynamics and current sheet formation," *New Journal of Physics*, vol. 9, no. 8, p. 301, aug 2007. [Online]. Available: https://dx.doi.org/10.1088/1367-2630/9/8/301

[93] N. Sasaki, K. Sato, T. Endo, and S. Matsuoka, "Exploration of lossy compression for application-level checkpoint/restart," in *2015 IEEE International Parallel and Distributed Processing Symposium*, 2015, pp. 914–922.

[94] M. Martel, "Compressed matrix computations," in *2022 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)*, 2022, pp. 68–76.

[95] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[96] ——, "High-throughput decoding for floating-point compression," in *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2020, pp. 10–21.

[97] G. Díaz, "Prototype Quantum Computing Simulator," https://github.com/diaztoro/TMFQSfullstate.git, Jan 2024.

[98] G. J. Díaz, L. A. Steffenel, and C. J. Barrios Hernández, "On the resource consumption of software quantum computing simulators," *DYNA*, vol. 88, no. 218, p. 72–80, jul. 2021. [Online]. Available: https://revistas.unal.edu.co/index.php/dyna/article/view/90781

[99] G. J. Díaz T, C. J. Barrios H., L. A. Steffenel, and J. F. Couturier, "Nearly quantum computing by simulation," in *High Performance Computing*, P. Navaux, C. J. Barrios H., C. Osthoff, and G. Guerrero, Eds. Cham: Springer International Publishing, 2022, pp. 205–219. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-23821-5_15

[100] J. Roell, "Demystifying quantum gates; one qubit at a time," https://towardsdatascience.com/demystifying-quantum-gates-one-qubit-at-a-time-54404ed80640, February 2018.

[101] N. S. Yanofsky, "An introduction to quantum computing," in *Proof, Computation and Agency - Logic at the Crossroads.*, 2011, pp. 145–180. [Online]. Available: https://doi.org/10.1007/978-94-007-0080-2_10

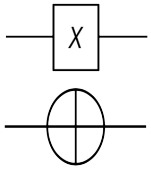
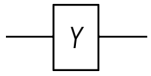
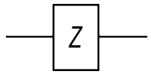
Appendix A

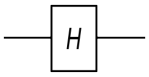
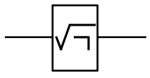
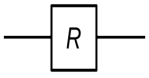
Quantum Gates

A.1 Single Qubit Quantum Gates

This section describes several quantum gates. The table A.1 shows some of them which act on only one qubit and the table A.3 shows gates that act on multiple qubits.

Table A.1: Single Qubit Quantum Gates

Quantum Gate	Diagram	Description
Pauli X Gate (X, NOT, bit flip, σ_x)		This gate acts linearly and it takes the state $\alpha\|0\rangle + \beta\|1\rangle$ to the corresponding state in which the role of $\|0\rangle$ and $\|1\rangle$ have been interchanged, that is to say, $\alpha\|1\rangle + \beta\|0\rangle$ [25]. The Pauli-X gate acts on a single qubit. It is the quantum equivalent of the NOT gate for classical computers [39]. The matrix representation is $$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{(A.1)}$$ If the quantum state is written in vector notation, applying the Pauli X gate we obtain $$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} \quad \text{(A.2)}$$ Using Dirac's notation, the transformation is as follows. $$\|0\rangle \rightarrow \|1\rangle \text{ and } \|1\rangle \rightarrow \|0\rangle \quad \text{(A.3)}$$
Pauli Y Gate, (Y, σ_y)		It equates to a rotation around the Y-axis of the Bloch sphere by π radians. It maps $\|0\rangle$ to $i\|1\rangle$ and $\|1\rangle$ to $-i\|0\rangle$. The Pauli-Y gate also acts on a single qubit. Its matrix representation is as follows. $$Y \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad \text{(A.4)}$$
Pauli Z Gate (Z, R_π, phase plip, σ_z)		It equates to a rotation around the Z-axis of the Bloch sphere by π radians. It leaves the basis state $\|0\rangle$ unchanged and maps $\|1\rangle$ to $-\|1\rangle$. This gate also acts on a single qubit. This is the matrix representation. $$Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{(A.5)}$$

Hadamard Gate		The Hadamard gate is the first authentic quantum gate because can generate superposition states. It maps the basis states as follows. $$\begin{aligned} \|0\rangle &\rightarrow \frac{\|0\rangle + \|1\rangle}{\sqrt{2}} \\ \|1\rangle &\rightarrow \frac{\|0\rangle - \|1\rangle}{\sqrt{2}} \end{aligned} \tag{A.6}$$ This implies that a measurement will have equal probabilities to become 1 or 0. It represents a rotation of π about the axis $(\hat{x} + \hat{z})/\sqrt{2}$. This is the combination of two rotations: π about the Z-axis followed by $\pi/2$ about the Y-axis. Its matrix representation is $$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{A.7}$$ The Hadamard gate is the one-qubit version of the quantum Fourier transform [39]. This is extremely useful for performing the first computation in any quantum program because it transforms initialized qubits back into their natural fluid state to leverage their full quantum powers [100].
Square Root NOT Gate ($\sqrt{NOT}$)		This gate maps the basis states as follows. $$\begin{aligned} \|0\rangle &\rightarrow \frac{(1+i)\|0\rangle + (1-i)\|1\rangle}{2} \\ \|1\rangle &\rightarrow \frac{(1-i)\|0\rangle + (1+i)\|1\rangle}{2} \end{aligned} \tag{A.8}$$ The matrix representation of this gate is as follows. $$\sqrt{NOT} = \sqrt{X} = \frac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix} \tag{A.9}$$
Phase Shift R_ϕ gate		**Definition:** This is a single qubit gate that leaves the basis state $\|0\rangle$ unchanged and maps the state $\|1\rangle$ to $e^{i\phi}\|1\rangle$. Def: $\|0\rangle \rightarrow \|0\rangle$ Applying this gate does not change the probability of measuring a $\|0\rangle$ or $\|1\rangle$, however it modifies the phase of the quantum state. This is equivalent to tracing a horizontal circle (a line of latitude) on the Bloch sphere by ϕ radians. The matrix representation is as follows. $$R_\phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix} \tag{A.10}$$ where ϕ is the phase shift.

A.2 Multiple Qubits Quantum Gates

The table A.3 shows the quantum gates that act on two or more qubits. Some of these gates are controlled gates, which implement *if-then-else* operations. These latter are useful to change the operation applied to one set of qubits depending upon the values of some other set of qubits.

A.2.1 Controlled Quantum Gates

Controlled gates act on two or more qubits, where one or more qubits act as a control for some operation. Let U be a gate that operates on single qubits with matrix representation.

$$U = \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix} \tag{A.11}$$

Then, the controlled-U operates on two qubits so that the first qubit serves as a control. It maps the basis states as follows.

$$\begin{aligned} |00\rangle &\mapsto |00\rangle \\ |01\rangle &\mapsto |01\rangle \\ |10\rangle &\mapsto |1\rangle \otimes U\,|0\rangle = |1\rangle \otimes (u_{00}\,|0\rangle + u_{10}\,|1\rangle) \\ |11\rangle &\mapsto |1\rangle \otimes U\,|1\rangle = |1\rangle \otimes (u_{01}\,|0\rangle + u_{11}\,|1\rangle) \end{aligned} \tag{A.12}$$

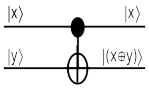
Thus, the matrix representation of the controlled-U is as follows

$$C(U) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix} \tag{A.13}$$

Table A.2: Controlled Quantum Gates

Quantum Gate	**Diagram**	**Description**

Controlled NOT Gate (CNOT or CX)

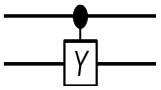


This gate has two inputs and two outputs. The top input is the control qubit, which controls what the output will be. If $|x\rangle = |0\rangle$, then the output of $|y\rangle$ will be the same value as the input. The output will be the opposite value if $|x\rangle = |1\rangle$. If the top qubit is written first and then the bottom qubit, then the controlled-not gate takes $|x, y\rangle$ to $|x, x \otimes y\rangle$ where $\otimes$ is the binary exclusive or operation [101].
The corresponding matrix to this reversible gate is as follows.

$$\begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

This is gate is used in quantum computing to generate entangled states.

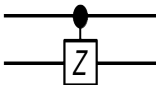
Controlled Pauli Y (CY)



Its matrix representation is

$$CY \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{bmatrix} \tag{A.14}$$

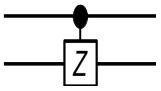
Controlled Pauli Z (CZ)



This gate adds (-1)-phase to the $|11\rangle$ basis state. This gate is symmetric in terms of qubits, that is to say, it does not matter which is the control or the target qubit. Its matrix representation is

$$CZ \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{A.15}$$

Controlled Phase Shift (CPS)



Definition:
This gate left unchanged all 2-qubits states except $|11\rangle$. That is to say:

Def: $|00\rangle \to |00\rangle$; $|01\rangle \to |01\rangle$; $|10\rangle \to |10\rangle$; $|11\rangle \to e^{i\phi}|11\rangle$;

We can rewrite the definition as follows

$$U_{cps}(\phi)|ab\rangle = e^{i(a\cdot b)\phi}$$

where $a \cdot b = a \wedge b$

a	b	result
0	0	0
0	1	0
1	0	0
1	1	1

The controlling qubit is a (most significative bit), and the target qubit is b. If the controlling qubit is 0 $a \cdot b$ is 0
The matrix representation of the controlled phase shift is

$$CPS \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix} \tag{A.16}$$

In general, applying a controlled phase shift gate to the following state

$$|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$$

Imply to do the following operation.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix} \times \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ e^{i\phi}\delta \end{bmatrix} \tag{A.17}$$

Therefore, we obtain the new state.

$$|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + e^{i\phi}\delta|11\rangle$$

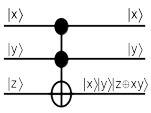
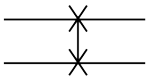
Toffoli Gate (Controlled CNOT or Deutsch $D(\pi/2)$)		This gate acts on the computational basis to flip the state of the third (target) qubit if and only if, the states of both of the first two (control) qubits are 1. This gate is the CNOT gate version for three qubits. If we limit input qubits to only $\|0\rangle$ and $\|1\rangle$, then if the first two qubits are in the state $\|1\rangle$ it applies Pauli X on the third qubit. Otherwise, it does nothing. The Toffoli gate is universal when combined with the single qubit Hadamard gate. The matrix representation is as follows. $$CCNOT = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (A.18)$$

Table A.3: Other Multiple Qubits Quantum Gates

Quantum Gate	Diagram	Description
SWAP Gate		This gate operates on two qubits. It swaps the state of the two qubits involved in the operation concerning the basis $\|00\rangle, \|01\rangle, \|10\rangle$ and $\|11\rangle$. Its matrix representation is $$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (A.19)$$

Appendix B

Quantum Algorithms

The Quantum Fourier Transform

The Discreet Fourier Transform (DFT) or Fourier Analysis is very used in the context of signal processing, linear algebra and many other branches of science. We can say that Fourier analysis useful because transformed version of a problem is often easier than the original problem. Quantum Fourier Transform (QFT) is a quantum implementation of the discreet Fourier transform [25].

The Fourier quantum transformation is a generalization of the Hadamard transformation. The difference is that QFT introduces phase. The specific types of phases introduced by QFT are the primitive roots of the unit, ω. Let's remind that in the complex numbers, the equation $z^n = 1$ has n solutions, for example: for $n = 2$ z could be 1 or -1, for $n = 4$ z could be 1, i, -1 or $-i$. This roots can be written as power of $\omega = e^{2\pi i}/n$. This number ω is called a primitive nth root of unity.

The figure B.1 shows that ω is in the unit circle, therefore, $|\omega| = 1$. The line from the origin to ω makes the angle $\phi = 2\pi/M$ with the real line. If ω is raised to the jth power, ω^j has phase angle $\phi = 2j\pi/M$ and is still Mth root of unity.

DFT is a transformation of a set $x_0, ...x_{N-1}$ of N complex numbers into a set of complex numbers $y_0, ...y_{N-1}$ defined by the equation B.1.

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

To build the quantum version of DFT let's define a linear transformation U on n qubits that acts on computational basis states $|j\rangle$ where $0 \leq j \leq 2^n - 1$.

$$|j\rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi ijk/2^n} |k\rangle \tag{B.2}$$

If we consider its action on superpositions we note that it corresponds to a vector notation for the Fourier transform (B.1) for the case $N = 2^n$.

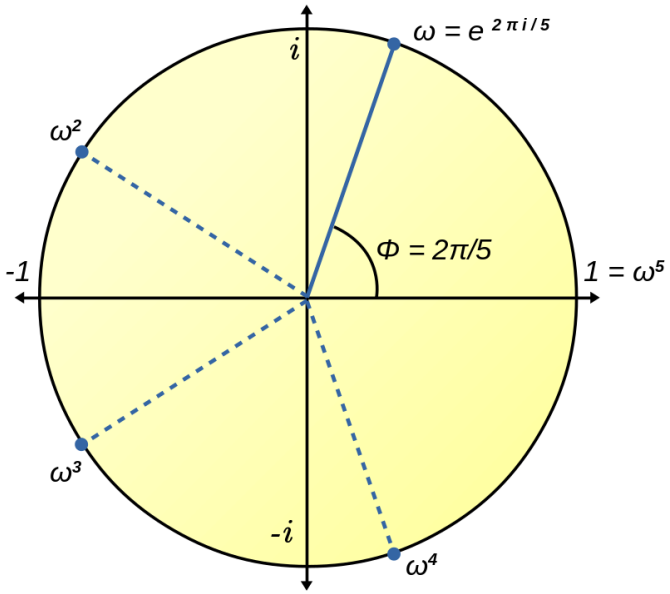


Figure B.1: The 5 complex 5th roots of 1

$$\sum_{k=0}^{2^n-1} x_j \left|j\right\rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \left[\sum_{j=0}^{2^n-1} e^{2\pi ijk/2^n} x_j\right] \left|k\right\rangle = \sum_{k=0}^{2^n-1} y_k \left|k\right\rangle \tag{B.3}$$

Considering the action of QFT on an orthonormal basis $\left|0\right\rangle, ..., \left|N-1\right\rangle$ we can define it as a linear operator with the following transformation on the basis states.

$$\left|j\right\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N} \left|k\right\rangle \tag{B.4}$$

That action on an arbitrary state can be written as

$$\sum_{j=0}^{N-1} x_j \left|j\right\rangle \rightarrow \sum_{k=0}^{N-1} y_k \left|k\right\rangle \tag{B.5}$$

Where the amplitudes y_k are the discrete Fourier transform of the amplitudes x_j. It can be checked that this transformation is a unitary transformation, and thus can be implemented as a quantum circuit.

Let's make $N = 2^n$ where n is some integer and the basis $\left|0\right\rangle, ..., \left|2^n-1\right\rangle$ is the computational basis for an n qubit quantum computer.

Let's write the state $\left|j\right\rangle$ using the binary representation $j = j_1 2^{n-1} + j_2 2^{n-2} + ... + j_n 2^0$. And let's use the notation $0.j_l j_{l+1} ... j_m$ to represent the binary fraction $j_l/2 + j_{l+1}/4 + ... + j_m/2^{m-l+1}$. Thus, the QFT can be written with the following product representation [25].

$$|j_1, ...j_n\rangle \rightarrow \frac{(|0\rangle + e^{2\pi i 0.j_n}|1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle)...(|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_n}|1\rangle)}{2^{n/2}} \tag{B.6}$$

The figure B.2 shows the quantum circuit for QFT. The gate R_k is the phase gate with the following matrix representation.

$$R_\phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix} \tag{B.7}$$

Applying the Hadamard gate on the first qubit produces the state

$$\frac{1}{2^{1/2}}(|0\rangle + e^{2\pi i 0.j_1}|1\rangle)\,|j_2 j_n\rangle \tag{B.8}$$

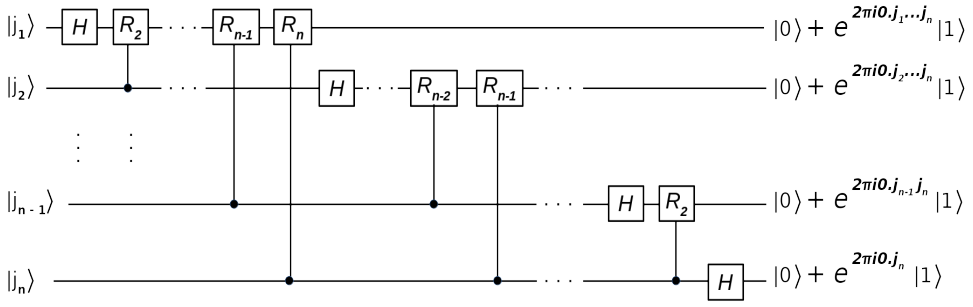


Figure B.2: Quantum Circuit for QFT

In terms of performance DFT takes $N\log(N) = n2^n$ steps to transform $N = 2^n$ numbers. On a quantum computer the transform can be accomplished using $\log^2(N) = 2^n$.

It seems that quantum computers can be used to very quickly calculate the Fourier transform of a vector of 2^n complex numbers. However, the Fourier transformation is performed on the "hidden" information in the amplitudes of the quantum state. This information is not directly accessible in the measurement process. The problem, of course, is that if the output status is measured, each qubit will collapse in the state $|0\rangle$ or $|1\rangle$, preventing us from learning the result of the transformation y_k directly.

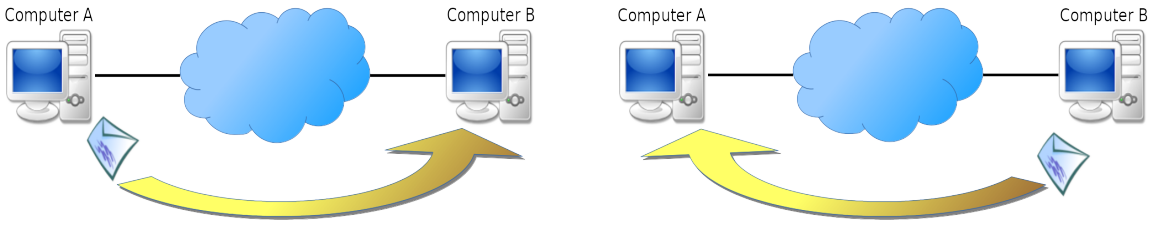
The Deutsch–Jozsa Algorithm

Let's suppose there a two computers A and B. Computer A selects a number from 0 to 2n -1 and sends it to computer B as a query (network packet). The latter calculates some function f(x) and replies the result, which is either 0 or 1, to computer B in another network packet.

The function f(x) is one of the following categories:

- $f(x)$ $f(x)$ is constant for all values of x.
- $f(x)$ is balanced, that is to say, is equal to 0 for exactly half of all the possible x, and 1 for the other half.

[49]



(a) Computer A choose a number x from 0 to $2^n - 1$ and sends it to Computer B

(b) Computer B computes $f(x)$ and returns the result to Computer A

Figure B.3: Deutsch Jozsa Algorithm

Classically, the computer A send only one value of x in each network packet (query). In the worst case, computer A needs to send at least $2^n/2 + 1$ queries, since it may receive $2^n/2$ 0s before finally receive a 1, which imply that computer B used a balanced function. Therefore, best deterministic classical algorithm that can be used requires $2^n/2+1$ queries. It have to be pointed out that every query contains n bits of information. Furthermore, $f(x)$ may be inherently difficult to calculate.

If we use qubits instead of bits and a unitary transformation U_f instead of $f(x)$ we can achieve the goal in just one query, using the following algorithm.

- **Inputs:** A black box U_f which performs the transformation

$$|x\rangle |y\rangle \rightarrow |x\rangle |y \otimes f(x)\rangle \text{ , for x} \in \{0, .., 2^n - 1\} \tag{B.9}$$

  $f(x)$ is constant for all values of x, of else $f(x)$ is balanced, that is to say, equal to 1 for exactly half of the values of x, and 0 for the other half.

- **Outputs:** 0 if and only if $f(x)$ is constant.
- **Runtime:** One evaluation of U_f
- **Procedure:** The algorithm is in the table B.1

The input state

$$|\psi_0\rangle = |0\rangle^{\otimes n} |1\rangle \tag{B.10}$$

1	Initialize state	$\|0\rangle^{\otimes n} \|1\rangle$
2	Generate superposition with Hadamard gates	$\rightarrow \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} \|x\rangle \left[\frac{\|0\rangle - \|1\rangle}{\sqrt{2}}\right]$
3	Calculate function f using U_f	$\rightarrow \sum_x (-1)^{f(x)} \|x\rangle \left[\frac{\|0\rangle - \|1\rangle}{\sqrt{2}}\right]$
4	Perform Hadamard transformation	$\rightarrow \sum_z \sum_x \frac{(-1)^{z\cdot x + f(x)} \|z\rangle}{\sqrt{2^n}} \left[\frac{\|0\rangle - \|1\rangle}{\sqrt{2}}\right]$
5	Measure to obtain the output z	$\rightarrow z$

Table B.1: Deutsch Jozsa Algorithm

Describes the state of n qubits all prepared in the $|0\rangle$ state. Then, after applying the Hadamard transform on the query register and the Hadamard gate on the answer register we obtain

$$|\psi_1\rangle = \sum_{x\in\{0,1\}^n} \frac{|x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \tag{B.11}$$

The query register is now a superposition of all values, and the answer register is in a evenly weighted superposition of 0 and 1. Then, the function f is evaluated in the computer B using $U_f\colon\ |x, y\rangle \rightarrow |x, y \otimes f(x)\rangle$, giving

$$|\psi_2\rangle = \sum_x \frac{(-1)^{f(x)} |x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \tag{B.12}$$

The computer A now has a set of qubits in which the result of the function evaluation, in the computer B, is stored in the amplitude of the qubit superposition state. Computer B interferes terms in the superposition using a Hadamard transform on the query register. It is useful to first calculate the effect of the Hadamard transformation in an $|x\rangle$ state,

in order to determine the result of the Hadamard transformation. By verifying the cases $x = 0$ and $x = 1$ separately, we observe that for a single qubit we have the following relation.

$$H|x\rangle = \sum_z \frac{(-1)^{xz}|z\rangle}{\sqrt{2}} \tag{B.13}$$

Thus

$$H^{\otimes n}|x_1, ..., x_n\rangle = \frac{\sum_{z_1,...,z_n}(-1)^{x_1z_1+...+x_nz_n}|z_1, ..., z_n\rangle}{\sqrt{2^n}} \tag{B.14}$$

This can be rewritten in a more succinctly and useful equation

$$H^{\otimes n}|x\rangle = \frac{\sum_z (-1)^{x\cdot z}|z\rangle}{\sqrt{2^n}} \tag{B.15}$$

Where $x\cdot z$ is the bitwise inner product of x and z, modulo 2. By combining this equation and the equation B.12 we can obtain $|\psi_3\rangle$

$$|\psi_3\rangle = \sum_z \sum_x \frac{(-1)^{z\cdot x+f(x)}|z\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \tag{B.16}$$

Note that the amplitude for the state $|0\rangle^{\otimes n}$ is $\sum_x (-1)^{f(x)}/2^n$. If f is constant the amplitude for $|0\rangle^{\otimes n}$ is $+1$ or -1, depending of the constant value $f(x)$ takes. Because $|\psi_3\rangle$ is of unit length it follows that all the other amplitudes must be zero, and an observation will yield 0s for all qubits in the query register. If f is balanced then the positive and negative contributions to the amplitude of $|0\rangle^{\otimes n}$ cancel, leaving an amplitude of zero, and a measurement must yield a result other than 0 on at least one qubit in the query register. In conclusion, if the computer A measures all 0s then the function is constant; otherwise the function is balanced.

This demonstrates that a quantum computer can solve the problem of Deutsch with only one evaluation of the function f compared to the classical requirement for $2^n/2 + 1$ evaluations. Although this seems much faster, there are several aspects against it. First, the problem of Deutsch is not a particularly important problem; it has no known applications. Second, the comparison between classical and quantum algorithms is somehow a comparison of apples and oranges, since the method to evaluate the function is quite different in the two cases. Third, if computer A is a probabilistic classic computer, by asking computer B to evaluate $f(x)$ for some randomly chosen xs, it can quickly determine with high probability whether f is constant or balanced. This probabilistic scenario is perhaps more realistic than the deterministic scenario we have been considering. The figure B.4 depicts the corresponding quantum circuit.

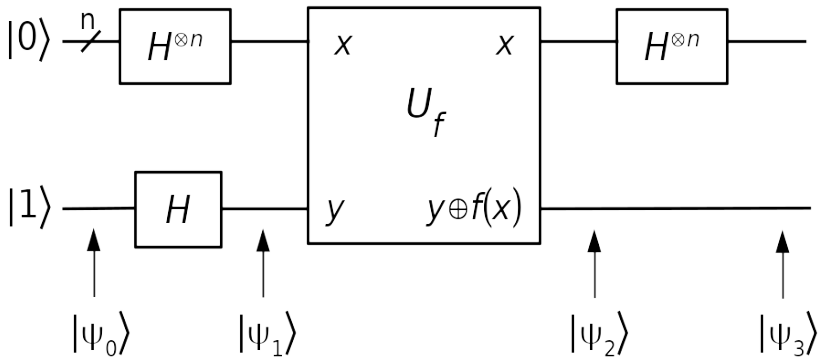


Figure B.4: Quantum circuit implementing the general Deutsch–Jozsa algorithm.

The Deutsch–Jozsa algorithm shows once again that quantum computers could be capable of solving some computational problems much more efficiently than classical computers. However, the problem it solves has no practical interest.

Appendix C

TMFQS User Guide

C.1 Requirements

To obtain better performance using TMFQS Intel Oneapi toolkit is required. For Linux distributions derived from RedHat you can follow the following steps.

1. Edit the file `/etc/yum.repos.d/oneAPI.repo` and add the following lines:

```
[oneAPI]
name=Intel oneAPI repository
baseurl=https://yum.repos.intel.com/oneapi
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://yum.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-PRODUCTS.PUB
```

2. Then, install the toolkit excuting the following commands:

```
rpm --import \
https://yum.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-PRODUCTS.PUB
dnf -y install intel-hpckit
```

C.2 Cloning the Source Code

You can download the simulator's source code from the github repository

```
git clone https://github.com/diaztoro/TMFQS
```

C.3 Compiling TMFQS

In order to compile the simulator please execute the following commands:

```
cd TMFQS
source ./environment
make
```

This creates the library located in the `lib64` directory. Several examples are in the `examples` directory, and the corresponding executables are placed in the `bin` directory.

The following example shows the QFT implementation using TMFQS.

```
#include "tmfqs.h"
#include <stdlib.h>
#include <iostream>
#include "utils.h"


using namespace std;

//TMFQS
int main(int argc, char *argv[]) {

	if(argc != 3) {
		cout << "./qft <Number of Qubits> <initialState>" << endl;
		return 1;
	}
	else {
		int i, j;

		unsigned int numberOfQubits = 0, initState = 0;
		numberOfQubits = atoi(argv[1]);
		initState = atoi(argv[2]);
		QuantumRegister qureg(numberOfQubits, initState);

		quatumFourierTransform(&qureg);
		qureg.printStatesVector();

		return 0;
	}
}
```