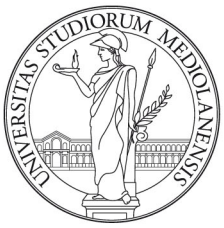


UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Laurea Triennale in Fisica

Benchmarking the performance of quantum simulations with tensor network and state vector strategies

Relatore interno:
Prof. Stefano Carrazza

Correlatore interno:
Alessandro Candido

Tesi di Laurea di:
Tommaso Galletti

Matr: **973106**

Anno accademico 2021/2022

Abstract

Quantum computing is a technology that aims to solve problems that are inaccessible to classical computers by exploiting the laws of quantum physics. To date, quantum computing is severely limited by the technical difficulties in building a scalable quantum computer that can operate without noise, errors or loss of quantum coherence.

To overcome these challenges and harness the power of quantum computing, it is essential to develop new algorithms and simulation procedures that can efficiently and accurately model quantum systems.

The most popular quantum simulations work by applying instructions and logic gates to qubits (the unit of quantum information), thus creating a quantum circuit and calculating the state vector (wave function) of the qubits at each step. However, the state space in which quantum states evolve is exponentially greater in size than the number of parameters needed to describe a specific state. In other words, the number of state amplitudes required for an exact simulation increases exponentially with the number of qubits, thus making it impossible to simulate systems with a large number of qubits.

My thesis aimed to use tensor networks to simulate quantum circuits with many qubits, evaluating the simulations' accuracy and computation time as a function of qubit number. More precisely, I explored the applicability of tensor networks to specific quantum circuits, namely the Quantum Fourier Transform (QFT) and the Hidden Shift circuit (HS).

Based on the comparison of the results obtained by sampling the two outcomes of the tensor network and the "exact" state vector simulations, I observed that the accuracy of the tensor network simulations remained consistent even with an increasing number of qubits.

This is significant because the approximations made by the tensor network method do not compromise the quality of the results while significantly reducing program execution time and computational burden.

In conclusion, for circuits with a reduced number of qubits and layers of gates, no remarkable difference has been observed. However, for simulations that are computationally intensive, such as those involving a high number of qubits or layers of gates, tensor network simulations are a more efficient option.

Contents

1 Mathematical foundations of quantum computing **7**

1.1 The postulates of quantum mechanics 7

1.1.1 Hilbert space $\hat{H}$ 7

1.1.2 Evolution 7

1.1.3 Unitary operators 8

1.1.4 Quantum measurement 9

1.1.5 Measurement on the computational basis 9

1.1.6 Composite systems 9

1.1.7 The no-cloning theorem 10

2 Introduction to quantum computing and simulations **11**

2.1 Quantum computing 11

2.1.1 Superposition 11

2.1.2 Interference 12

2.2 Quantum computing simulations 12

2.3 Quantum algorithms 13

2.4 Quantum circuits 13

2.4.1 Quantum gates 14

2.5 State vector and tensor network approaches 14

2.5.1 State vector (SV) 14

2.5.2 Tensor network (TN) 15

3 Simulation and Benchmarking results: Qibo vs Quimb. **17**

3.1 Qibo 17

3.1.1 Simulation steps 17

3.2 Quimb 18

3.2.1 Tensor module 19

3.2.2 Tensors 19

3.2.3 Tensor networks 20

3.2.4 Contraction 21

3.2.5 Gates 22

3.2.6 Simulation Steps 22

3.3 Simulation 23

3.3.1 QFT 23
3.3.2 Hidden Shift 29

4 Conclusions **33**

Bibliography **35**

Chapter 1

Mathematical foundations of quantum computing

You can find a more in-depth version of this section in the books: "Quantum Computation and Quantum Information" [16], "Quantum Information and Computation" [23], "Quantum Computer Science" [13], "Quantum information and computation" [2].

1.1 The postulates of quantum mechanics

Postulate 1 Associated to any isolated physical system is a complex vector space with inner product (that is, a Hilbert space) known as the state space of the system. The system is completely described by its state vector, which is a unit vector in the system's state space.

1.1.1 Hilbert space $\hat{H}$

A single bit of classical information can take on one of two possible values: 0 and 1.

n classical bits of information span the 2^n-dimensional space $\{0,1\}^n$.

The Hilbert space $H_{QB(n)} = \ell^2(\{0,1\}^{\otimes n})$ is the quantized version of the classical n-bit space $\{0,1\}^n$. It is a complex vector space consisting of square-integrable functions defined on $\{0,1\}^{\otimes n}$, and possesses a natural inner product structure. This space can be viewed as the linear combination or superposition of classical bit strings. It should be noted that $H_{QB(n)}$ is a vector space over the complex numbers with a dimension of 2^n. The state-vectors of n-qubit quantum registers are represented by the elements of this vector space.

1.1.2 Evolution

How does the state, $|\psi\rangle$, of a quantum mechanical system evolve in time?

Postulate 2 The evolution of a *closed* quantum system is described by a unitary transformation. 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 $\hat{U}$ which depends only on the times t_1 and t_2

$$|\psi'\rangle = \hat{U}|\psi\rangle, \tag{1.1}$$

Postulate 2' The time evolution of the state of a *closed* quantum system is described by the Schrödinger equation:

$$i\hbar\frac{d|\psi\rangle}{dt} = \hat{H}|\psi\rangle, \tag{1.2}$$

where $\hbar$ is the reduced Planck's constant and $\hat{H}$ is the Hamiltonian of the closed system.

Any unitary operator $\hat{U}$ can be realized in the form $\hat{U} = exp(i\hat{K})$ for some Hermitian operator K. There is therefore a one-to-one correspondence between the discrete-time description of dynamics using unitary operators, and the continuous time description using Hamiltonians. [31]

1.1.3 Unitary operators

In mathematics, a unitary operator is a linear transformation that preserves the inner product of a given vector space. More specifically, if $\hat{U}$ is a unitary operator on a vector space H, then for any two vectors u and v in H, the inner product of $\hat{U}$u and $\hat{U}$v is equal to the inner product of u and v:

$$\langle \hat{U}u, \hat{U}v\rangle = \langle u, v\rangle, \tag{1.3}$$

where $\langle\ ,\ \rangle$ denotes the inner product. In simpler terms, an operator $\hat{U}$ is unitary if and only if each of its matrix representations is unitary:

$$\hat{U}^\dagger\hat{U} = \mathbb{1}, \tag{1.4}$$

The fact that these unitary operators preserve the inner product suggests the following outer product representation of any unitary $\hat{U}$.

Let $|v_i\rangle$ be any orthonormal basis set. Define $|w_i\rangle \equiv \hat{U}|v_i\rangle$, so $|w_i\rangle$ is also an orthonormal basis set, since unitary operators preserve inner products. Note that $\hat{U} = \sum_i |w_i\rangle\langle v_i|$.

Conversely, if $|v_i\rangle$ and $|w_i\rangle$ are any two orthonormal bases, then it is easily checked that the operator $\hat{U}$ defined by $\hat{U} \equiv \sum_i |w_i\rangle\langle v_i|$ is a unitary operator.

When discussing quantum computation and information, it's common to refer to the application of a unitary operator to a specific quantum system. However, this seems to contradict the idea that unitary operators describe

the evolution of a closed quantum system, as applying an operator suggests an external entity interacting with the system.

Nonetheless, it's possible to have a time-varying Hamiltonian for a quantum system that's under an experimentalist's control and may change during an experiment, meaning that the system isn't completely closed but still follows Schrodinger's equation with a time-varying Hamiltonian. Despite this, we usually describe the evolution of quantum systems, even those that aren't closed, using unitary operators, with the exception of quantum measurement.

1.1.4 Quantum measurement

Postulate 3 Quantum measurements are described by a collection $\{M_m\}$ of *measurement operators*. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement then the probability that result m occurs is given by

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle, \tag{1.5}$$

and the state of the system after the measurement is $\frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}$

The measurement operators satisfy the *completeness equation* ([21], chapter 4) (this expresses the fact that the sum of probabilities over all possible outcomes sums to 1)

$$\sum_m M_m^\dagger M_m = \mathbb{1}. \tag{1.6}$$

1.1.5 Measurement on the computational basis

Measurements are operations strictly performed on a single qubit, therefore it's easy to define $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$. It is immediate to see that these measurement operators are Hermitian and idempotent, and that the completeness relation is obeyed ($\mathbb{1} = M_0^\dagger M_0 + M_1^\dagger M_1 = M_0 + M_1$).

If the state being measured is $|\psi\rangle = a|0\rangle + b|1\rangle$ Then the probability of obtaining outcome $|0\rangle$ is

$$p(0) = \langle\psi|M_0^\dagger M_0|\psi\rangle = \langle\psi|M_0|\psi\rangle = |a^2|, \tag{1.7}$$

The two possible resulting states are $|0\rangle$, $|1\rangle$. [17]

1.1.6 Composite systems

Suppose we are considering a composite quantum system that consists of two (or more) separate physical systems. How can we represent the states of the composite system mathematically? The following postulate outlines how the state space of a composite system is constructed from the state spaces of the individual component systems

Postulate 4 The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered through n, and system number i is prepared in the state $|\psi_i\rangle$, then the joint state of the total system is

$$|\psi_1\rangle \otimes ... \otimes |\psi_n\rangle, \tag{1.8}$$

According to the postulate, the state space of a composite physical system can be represented as the tensor product of the state spaces of the component physical systems. This means that the overall state of the composite system can be expressed as a product of the states of the individual systems.

It's important to note that this postulate applies even if the component systems are not initially in a separable state, meaning their states cannot be expressed as a product of individual states. In this case, the state of the composite system cannot be expressed as a simple tensor product, and the individual states are said to be entangled. [8]

1.1.7 The no-cloning theorem

The no cloning theorem is a fundamental result in quantum mechanics that states that it is impossible to create an exact copy of an arbitrary unknown quantum state. More precisely, the no cloning theorem can be stated as follows:

If we have an arbitrary quantum state $|\psi\rangle$ and another state $|\phi\rangle$ that we want to copy $|\psi\rangle$ onto, the no-cloning theorem states that there is no unitary operator U that can transform the initial state $|\psi\rangle \otimes |\phi\rangle$ into the final state $|\psi\rangle \otimes |\psi\rangle$.

In other words, the no cloning theorem prohibits the existence of a quantum process that can make an exact copy of a given quantum state.

This has important implications for quantum information processing, because it means that quantum information cannot be perfectly replicated or backed up like classical information can be. [33], [26]

Chapter 2

Introduction to quantum computing and simulations

2.1 Quantum computing

Quantum computing seeks to harness the properties of quantum mechanics to solve computational problems that are intractable for classical computers.

The main advantages of quantum computing come from the ability of quantum systems to exist in superposition states and to exhibit interference between different quantum states. [15]

2.1.1 Superposition

The computational basis for a quantum computer is $\{|0\rangle, |1\rangle\}$.

A classical bit of information is either exactly 0, or exactly 1; instead, a qubit is in a superposition of the basis states: the wave-function describing a single qubit is thus $|\psi\rangle = c_0|0\rangle + c_1|1\rangle$ One important consequence of superposition is that quantum systems can perform multiple computations in parallel. This is because each state in the superposition represents a different computation, and the computation performed by the system as a whole is a weighted sum of all these computations. In quantum computing, superposition is harnessed to perform computations more efficiently than classical computers. Quantum algorithms are designed to exploit the power of superposition by encoding a problem as a superposition of states and manipulating the state through quantum gates to obtain the answer.

Superposition is a fundamental concept in quantum mechanics that enables quantum systems to perform multiple computations in parallel and is a key ingredient in quantum computing and quantum simulations.

2.1.2 Interference

Interference arises due to the wave-like nature of particles in quantum mechanics: particles can also exhibit wave-like behavior and their wave functions can interfere. This means that the probability of finding a particle in a particular state can be either enhanced or reduced depending on the interference of its wave function with other particles. In quantum computing, quantum interference is a key ingredient that allows quantum algorithms to be exponentially more efficient than classical algorithms for certain types of problems. Quantum algorithms take advantage of quantum interference to ensure that the incorrect answers cancel out while the correct answer is reinforced, resulting in a higher probability of obtaining the correct answer. For example, in Shor's algorithm, the interference between the wave functions of qubits is used to efficiently factor large numbers, which is a problem that is believed to be intractable for classical computers. [27]

Another example is the Grover's algorithm, which uses interference to search an unsorted database with a quadratic speedup compared to classical algorithms. [7]

2.2 Quantum computing simulations

A quantum simulation aims to simulate the behavior of a quantum system using a classical computer. To do this, we first need to represent the quantum state of the system as a state vector, which is a complex-valued vector in a Hilbert space.

The state vector of the quantum system is then evolved over time using the laws of quantum mechanics. This evolution is represented using a unitary operator, which is a matrix that describes the transformation of the state vector over time. The unitary operator is applied to the state vector using matrix multiplication, and the resulting state vector represents the state of the system at a later time.

In a quantum simulation, the unitary operator is typically represented using a quantum circuit. A quantum circuit is a graphical representation of a quantum algorithm, where the nodes in the circuit represent quantum bits (qubits) and the edges represent the interactions between qubits.

The quantum circuit is made up of logic gates, which are operations that manipulate the state of the qubits. The logic gates used in a quantum circuit are typically unitary operators, which means they can be represented by matrices that preserve the normalization of the state vector.

By applying a sequence of logic gates to the qubits in a quantum circuit, we can construct a unitary operator that represents the evolution of the quantum state over time. The resulting unitary operator can then be applied to the initial state vector of the system to obtain the state of the system at a later time.

One approach to quantum simulations is to use matrix representations of quantum operators to simulate the evolution of the quantum system. This approach is called matrix product state (MPS) simulation and is particularly useful for simulating 1D quantum systems. MPS simulation involves decomposing the state of the system into a series of tensors, with each tensor representing the state of a particular segment of the system. The evolution of the system is then simulated by applying matrix operations to these tensors. [22], [18]

In summary, a quantum simulation involves representing the quantum state of a system and evolving it over time using a unitary operator represented by a quantum circuit, and using the resulting state vector to extract information about the behavior of the quantum system.

2.3 Quantum algorithms

A quantum algorithm is an algorithm that runs on a model of quantum computation, i.e. a quantum computer or a realistic simulation of one.

A classical algorithm is a series of instructions, where each step can be performed on a classical computer, likewise a quantum algorithm is a step-by-step procedure that can be performed on a quantum computer. While all classical algorithms can be performed on a quantum computer [10], the term quantum algorithm exclusively refers to procedures which make use of inherently quantum effects, such as superposition, entanglement and interference.

Quantum algorithms are designed to run on ideal, error-free quantum computers; however, in reality, physical quantum computers are subject to errors due to various factors, such as the imperfect control of the qubits and the interactions between the qubits and their environment. Therefore, a realistic model of quantum computation is necessary to study the behavior of quantum algorithms in the presence of these errors and to develop methods to mitigate them. One example of a realistic model of quantum computation is the quantum circuit model, which represents quantum computation as a sequence of quantum gates acting on qubits. This model takes into account the fact that quantum gates have finite precision and may introduce errors, and allows for the simulation of noisy quantum circuits.

2.4 Quantum circuits

Quantum circuits offer a practical framework for quantum computation, where algorithms consist of a sequence of qubit initializations, quantum logic gates, and measurements.

2.4.1 Quantum gates

The quantum logic gates are reversible unitary transformations on at least one qubit, i.e. a small circuit acting on at most a few qubits and performing reversible operations. [1]

The majority of basic logic gates in classical computing are not reversible, meaning that it is not always possible to retrieve the initial input bits from the resulting output bit. For instance, in the case of an AND gate, the two input bits cannot always be retrieved from the output bit.

An n-bit (classical) reversible gate is a bijective mapping f from the set $\{0,1\}^n$ of n-bit data onto itself.

In the context of reversible logic gates, the Pigeonhole Principle states that any gate which consumes its inputs and allows for all input computations must have no fewer input bits than output bits. This implies that the number of possible input combinations must be less than or equal to the number of output combinations. Specifically, for one input bit, there are only two possible reversible gates, namely the NOT gate and the identity gate. Similarly, for two input bits, the only non-trivial reversible gate is the controlled NOT gate, which performs an XOR operation on the second bit controlled by the first bit, while leaving the first bit unchanged. However, this limited set of gates is not sufficient to compute all arbitrary reversible functions, and additional gates are required. One such gate is the Toffoli gate, which was introduced by Toffoli in 1980.

The Toffoli gate The Toffoli gate is, as mentioned before, a universal reversible logic gate: it has the capacity to perform any logical operation that can be carried out by a reversible circuit. The Toffoli gate, or CCNOT (controlled-controlled-not) has 3-bit inputs and outputs; if the first two bits are both set to 1, it inverts the third bit, otherwise all bits stay the same. In other words, the Toffoli gate can be viewed as a function that operates on three input bits a,b and c and maps them like: $\{a, b, c\} \rightarrow \{a, b, c \oplus ab\}$. The Toffoli gate having a direct quantum equivalent and it being universal are the reasons why every classical algorithm can be represented as a quantum circuit. [32]

2.5 State vector and tensor network approaches

2.5.1 State vector (SV)

State vector quantum computing simulations are a type of quantum computing simulation that involves manipulating and analyzing the state vectors of quantum systems. In quantum computing, a state vector represents the state of a quantum system and contains all the information about the system that can be extracted through measurement.

In state vector quantum computing simulations, the state vector is manipulated using mathematical operations to simulate the behavior of quantum computing algorithms on quantum hardware.

However, as the number of particles in the system increases, the number of components in the state vector grows exponentially, making it impractical to store and manipulate the entire state vector. [11]

This is where tensor network methods come in, which are a class of algorithms for efficiently representing and manipulating high-dimensional tensors, such as the state vector of a large quantum system.

2.5.2 Tensor network (TN)

Every quantum circuit is in precise one-to-one correspondence with a tensor network. [19], [30], [12]

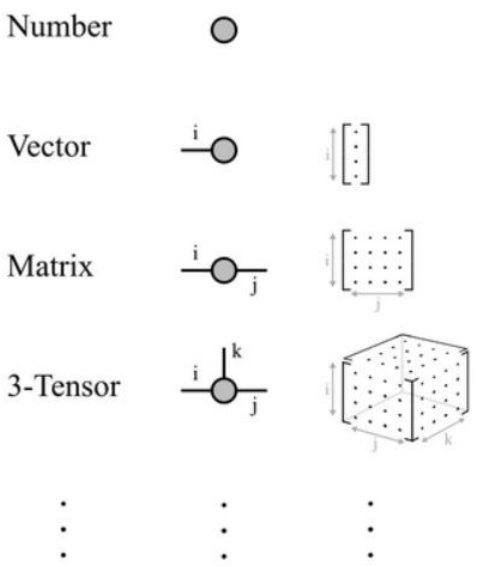
Tensor networks are a mathematical framework used in quantum physics and machine learning to efficiently represent and manipulate high-dimensional data structures. In quantum physics, tensor networks are used to represent the wave function of a quantum state in a compact form. They are also used to simulate quantum systems and perform quantum calculations efficiently. [25]

A tensor network is a network of tensors connected by tensor contractions, where each tensor represents a multi-dimensional array of numbers. The network structure specifies how the tensors are connected, and the connections represent the contractions performed between tensors.

The Quimb python library is a library for working with tensor networks, including the ability to construct and manipulate tensor networks, perform tensor contractions, and simulate quantum circuits using tensor networks. With Quimb, you can use tensor networks to represent and manipulate complex data structures efficiently.

Penrose graphical notation The Penrose graphical notation (or tensor diagram notation) is a visual depiction of tensors. [20] It consists of differently shaped nodes linked together by lines. In tensor algebra, particular tensors are associated with specific shapes, lines linking different tensors represent contraction of indices. This notation is inherently basis-independent

Figure 2.1: Penrose graphical notation - some examples



Chapter 3

Simulation and Benchmarking results: Qibo vs Quimb.

3.1 Qibo

Qibo is an open-source Python library for quantum computing that provides a set of tools for simulating and running quantum circuits on both classical and quantum computers. It allows users to define quantum circuits, perform quantum operations, and simulate the results of these operations.

Qibo provides a user-friendly interface for defining these quantum gates and circuits, and for simulating their behavior using state vector simulations.

Concretely, in the Qibo framework, quantum circuits are represented as a sequence of operations that can be executed on a quantum register. The register contains a set of qubits that can be initialized to a specified state. These operations can include single-qubit gates, such as the Pauli X gate or the Hadamard gate, as well as multi-qubit gates, such as the CNOT gate.

Qibo allows users to simulate the behavior of these circuits using state vector simulations, which simulate the evolution of the quantum state of the system over time. This simulation can be used to calculate the probabilities of measuring different outcomes, as well as to visualize the behavior of the quantum system as it evolves. [3], [4], [5]

3.1.1 Simulation steps

The general procedure for simulating a quantum circuit using Qibo is as follows:

1. Define the quantum circuit: Use the provided interface to define the quantum circuit as a sequence of quantum gates acting on a set of qubits.

2. Define the initial state: Specify the initial state of the qubits in the

circuit. This can be done either by setting the qubits to a known state (e.g. $|0\rangle$), or by defining a custom initial state vector.

3. Simulate the quantum circuit: Use the simulate function provided by Qibo to simulate the behavior of the circuit. This function takes as input the circuit and the initial state, and returns the final state of the circuit.

4. Compute observables: Once the final state of the circuit has been obtained, use Qibo's tools to compute any desired observables or quantities, such as probabilities of measuring certain outcomes or entanglement measures. This can be done using the measure function, which performs measurements on the final state, or by using other functions provided by Qibo for calculating specific observables.

3.2 Quimb

Quimb is a Python library for quantum information and many-body simulations. It provides a wide range of tools and functionality for simulating quantum systems, including states, operators, and measurements. [28]

Quimb is built on top of NumPy and SciPy, making it fast and efficient for large-scale simulations. It also supports distributed computing for even greater performance.

- Efficient handling of sparse matrices, which is important for simulating large quantum systems.
- Support for a variety of quantum states, including pure states, density matrices, and mixed states.
- A range of built-in operators, such as the Hamiltonian and spin operators, as well as the ability to define custom operators.
- Tools for measuring observables and computing entanglement entropy.
- Support for both exact diagonalization and tensor network methods.

Quimb is a powerful library for simulating quantum systems and is used in a wide range of applications, including quantum computing, condensed matter physics, and quantum chemistry.

In my thesis work, I exclusively used the Quimb module related to tensor networks.

3.2.1 Tensor module

The quimb.tensor module is a powerful tool for working with tensors and tensor networks. Its key feature is its ability to automatically handle arbitrary geometry, making it useful for simulations beyond 1D and 2D lattices. With this module, you can construct and manipulate arbitrary (hyper) graphs of tensor networks, and automatically contract, optimize, and draw networks.

In addition, the tensor module supports various backend array libraries, including Jax and Torch via autoray, which allows you to take advantage of their performance and optimization features.

The tensor module also includes specific algorithms for simulating quantum systems, such as MPS (Matrix Product State), PEPS (Projected Entangled Pair States), MERA (Multi-scale Entanglement Renormalization Ansatz), and quantum circuit algorithms, including DMRG (Density Matrix Renormalization Group) and TEBD (Time-Evolving Block Decimation).

Overall, the quimb.tensor module provides a comprehensive set of tools for working with tensors and tensor networks, in particular it's its ability to represent and contract arbitrary geometry tensor networks that makes it so powerful in simulating quantum circuits.

3.2.2 Tensors

A tensor object has 3 main attributes: tensor.data, tensor.inds, tensor.tags

tensor.data n-dimensional array looking object, i.e. something with a .shape attribute.

tensor.inds An ordered tuple of index names - one for each dimension of .data. These are propagated through all operations. For example, operations such as transposing the underyling array can just be done via reordering the inds of the tensor. The names of indices explicitly define the geometry of bonds (edges) in tensor networks, more precisely, if 2 or more tensors share the same index they will be linked together by an edge in the graph of the tensor network, edges represent contraction.

tensor.tags Ordered set of additional identifiers for tensors in a network. The idea behind tags is that within a tensor network you can use any number of simultaneous labelling schemes to identify tensors.

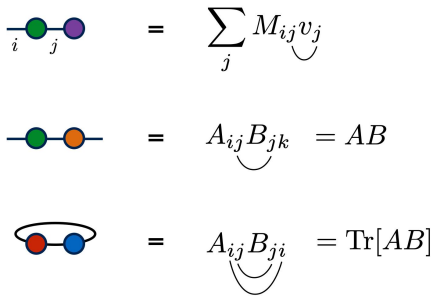
Creating a tensor To create a tensor we need arrays of data and indices to label their dimension; tensors sharing the same index name automatically form a 'bond' or implicit contraction when put together.

3.2.3 Tensor networks

Creating tensor networks To create a network out of a collection of tensors we simply join them together using the *&* operator.

The geometry of the tensor network is completely defined by the repeated indices which form bonds between different tensor objects, see the figure below for some examples.

Figure 3.1: Penrose graphical notation - some examples of operations between tensors



Contraction Tensor contraction is the process of summing over repeated indices in a tensor product. The contraction between two tensors with shared indices is achieved by multiplying the two tensors element-wise and then summing over the shared indices.

For example, in tensor index notation, the contraction of two rank-2 tensors A_{ij} and B_{jk} over the index j can be written as $C_{ik} = \sum_j A_{ij} B_{jk}$; between two rank-3 tensors as: $C_{ijk} = \sum_{m=1}^{M} \sum_{n=1}^{N} A_{imn} B_{jmn}$ and so on.

A more in-depth discussion of TN contraction can be found in subsection 3.2.4

Decomposition Section 1.5 of the quimb documentation introduces the concept of tensor decomposition, which involves expressing a tensor as a sum of simpler, more structured tensors.[9] The most common types of tensor decomposition are the Singular Value Decomposition (SVD) and the Tucker decomposition. The SVD expresses a tensor as a sum of outer products of singular values and orthogonal matrices, while the Tucker decomposition expresses a tensor as a core tensor multiplied by a matrix along each mode.

Quimb provides functions for computing both the SVD and the Tucker decomposition of a tensor.

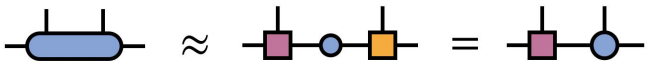
Tensor decomposition is useful for compressing and analyzing large tensors, as well as for identifying patterns and structure within the data.

Shown in the figure below is the Penrose representation of a truncated singular value decomposition, which is a technique used to approximate a ten-

sor after it has been decomposed using singular value decomposition (SVD). SVD is an exact method that decomposes a tensor into a set of singular values and corresponding singular vectors.

The truncated version of SVD involves keeping only a subset of the largest singular values and their corresponding singular vectors. This allows for an approximation of the original tensor while reducing the computational burden and storage requirements. TSVD is a widely used technique in tensor network methods, particularly in the context of quantum information and quantum simulations, where it is used to reduce the complexity of large-scale tensor networks.

Figure 3.2: Penrose graphical representation of a truncated SVD of a 4-tensor



Gauging Gauging is a procedure for transforming a tensor network while preserving its overall properties. Gauging involves inserting or removing additional tensors, known as gauge tensors, at certain points in the network. These gauge tensors are chosen such that they have no physical effect on the tensor network but can be used to simplify or modify its structure.

Gauging is useful for simplifying and optimizing tensor networks, as well as for removing certain types of singularities or degeneracies in the network.

Compressing Tensor compression involves approximating a tensor with a smaller tensor that captures the most important features of the original tensor.

Tensor compression is useful for reducing the storage and computational requirements of large tensors while preserving their essential properties.

Quimb provides functions for compressing tensors using several different methods, including the truncated SVD, the Higher-Order Orthogonal Iteration (HOOI) method, the Canonical Polyadic decomposition (CP) and the Alternating Least Squares (ALS) method. [14] These methods all involve finding a smaller tensor that approximates the original tensor while minimizing some measure of error or loss.

3.2.4 Contraction

Tensor contraction involves contracting a tensor network into a single tensor, and doing so in an optimal way can greatly affect the computational cost of the contraction.

Instead of contracting the entire network at once, it is often more efficient to contract pairs of tensors in a pairwise path of intermediates. Contracting a pair of tensors can remove their inner indices entirely from the rest of the contraction, simplifying the calculation.

The optimal sequence of pairwise contractions is specified by a "contraction tree". However, the cost of the contraction is incredibly sensitive to the choice of the contraction tree, and the space of these trees is very large, making it a tricky problem. Nevertheless, this problem can be automated using algorithms like those implemented in the quimb library.

There is a trade-off between the time spent finding the optimal contraction tree and the time spent actually doing the contraction. Finding the optimal contraction tree can be time-consuming, but it can significantly reduce the computational cost of the contraction itself. Therefore, it is important to strike a balance between these two stages in order to achieve efficient tensor contraction.

3.2.5 Gates

In the context of tensor networks, gates are unitary tensors that represent operations that can be applied to the network. These operations can represent anything from quantum gates to classical updates in a Markov Chain Monte Carlo simulation. For example, in the simulation of a quantum circuit, gates can be used to simulate the time evolution of the system.

3.2.6 Simulation Steps

The general procedure for simulating a tensor circuit using Quimb is as follows:

1. Initialize a tensor network representing the initial state of the circuit. (the default initial state is: $|000\ldots00\rangle$)
2. Add tensors representing logical gates to the circuit.
3. After all gates have been applied, contract the entire tensor network to obtain the final state of the circuit.
4. Compute any desired observables or quantities from the final state. This includes the possibility to sample a final bit string.

Some examples of observables that can be directly calculated with Quimb are:

- Expectation values: Quimb can calculate the expectation value of any operator on a quantum state, including the Hamiltonian, spin operators, and custom operators defined by the user.

- Correlation functions: Quimb can calculate correlation functions of various types, including spin-spin correlations, density-density correlations, and more.
- Reduced density matrices: Quimb can compute the reduced density matrix of a subsystem of a larger quantum state, which can be used to study entanglement and other properties.
- Quantum information measures: Quimb provides tools for computing a range of quantum information measures, such as von Neumann entropy, mutual information, and quantum fidelity.

The process of computing the implicit sum-of-products represented by a tensor network requires a series of pairwise contractions to convert the network into a single tensor with the same outer shape and indices. However, the cost of these intermediate contractions can be highly dependent on the chosen path or sequence. In the context of the quimb python library, both the path-finding stage and the contraction stage are automated. Nevertheless, there exists a trade-off between the time spent finding the optimal path and the time taken to perform the contraction itself. [6]

3.3 Simulation

To compare the performance of the two types of quantum computing simulations, I implemented and performed measurements on a quantum circuit performing the quantum Fourier transform (QFT) and on one implementing a specific Hidden Shift problem (HS).

By implementing these circuits and measuring their output and execution times I was able to compare the accuracy and efficiency of the two methods.

All programs were executed with the same parameters in order to obtain commensurable time measurements: on 4 CPUs, each with a minimum memory of 16GB, specifically on Intel(R) Xeon(R) CPU E5-2650, @2.00 GHz.

3.3.1 QFT

The QFT is a crucial algorithm in quantum information theory.

It is a unitary transformation used in quantum computing that is analogous to the classical discrete Fourier transform. It plays a crucial role in many quantum algorithms, including Simon's algorithm, phase estimation, quantum error correction, and amplitude estimation, in addition to the aforementioned Shor's factoring algorithm.

Its importance in quantum computing stems from the fact that it allows for the efficient manipulation of the phase of a quantum state, which is a key resource in many quantum algorithms.

The QFT acts on a quantum state represented by a vector $|\psi\rangle$ of length $N = 2^n$ for n qubits, mapping it to another vector $|\tilde{\psi}\rangle$ by applying a linear transformation to each basis state $|j\rangle$:

$$|\psi\rangle = \sum_{j=0}^{N-1} x_j |j\rangle, \tag{3.1}$$

$$|\hat{\psi}\rangle = \sum_{k=0}^{N-1} y_k |k\rangle, \tag{3.2}$$

where

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_N^{j,k}, \tag{3.3}$$

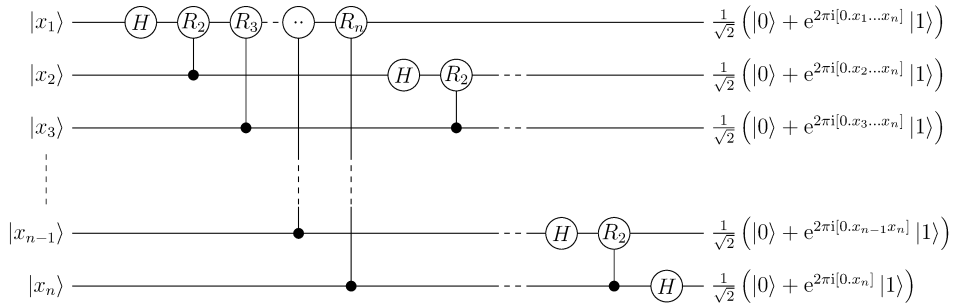
and

$$\omega_N^{j,k} = \exp^{2\pi i \frac{jk}{N}}, \tag{3.4}$$

where $|j\rangle$ is the j-th basis state of the input vector $|\psi\rangle$, and $|k\rangle$ is the k-th basis state of the output vector $|\tilde{\psi}\rangle$.

The circuit for the Quantum Fourier Transform (QFT) is composed of single-qubit Hadamard gates and two-qubit rotation gates.

Figure 3.3: QFT - circuit diagram



Results

In our investigation of the quantum Fourier transform (QFT), we observed that both state vector and tensor network simulations exhibit exponential scaling in time with respect to the number of qubits. However, we found that the tensor network simulation significantly outperformed the state vector simulation in terms of computational speed and efficiency, particularly in the execution and sampling of results. The final time obtained is the result of 1000 circuit creations and sampling of 10,000 outputs in each creation.

The sampling method in the Quimb library is responsible for finding the contraction path of the circuit represented by the tensor network autonomously during the first sampling. Once the best path is found, the subsequent samples are computationally lightweight, so the difference in time between taking a single sample and taking a million samples is minimal; as you can see in the table below, where are shown the sampling times of an 8 qubit Tensor Network QFT:

TN QFT(8_{qb})	t [s]	σ_t [s]
1 time	2.95×10^{-6}	0.26×10^{-6}
10^6 times	3.24×10^{-6}	0.34×10^{-6}

Table 3.1: 8 qubit Tensor Network QFT - sampling times

The same measurements were also taken for an 8-qubit quantum Fourier transform (QFT) state vector circuit:

SV QFT(8_{qb})	t [s]	σ_t [s]
1 time	0.195	0.086
10^6 times	0.873	0.214

Table 3.2: 8 qubit State Vector QFT - sampling times

After fitting the total execution time as a function of the number of qubits for both state vector and tensor network simulations, I obtained the following two functions:

$$f_{SV}(N_{qb}) = 6.48 \times 10^{-6} e^{7.39 \times 10^{-1} N_{qb}} + 7.20 \times 10^{-1}, \tag{3.5}$$

$$f_{TN}(N_{qb}) = 5.77 \times 10^{-2} e^{7.19 \times 10^{-2} N_{qb}} + 7.26 \times 10^{-2}, \tag{3.6}$$

The fit function for the state vector simulation exhibited an exponential scaling with a larger coefficient (7.39×10^{-1}) compared to that of the tensor network simulation (7.19×10^{-2}), indicating superior computational efficiency.

Figure 3.4: State vector QFT - execution time as a function of the number of qubits

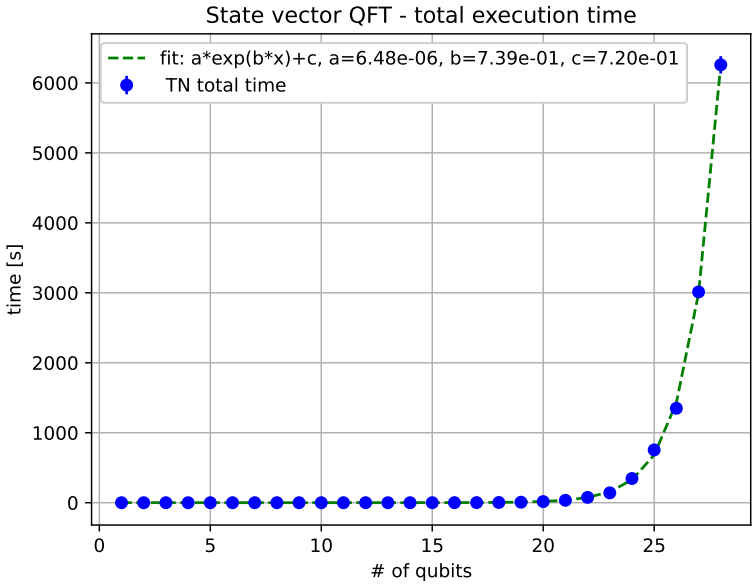
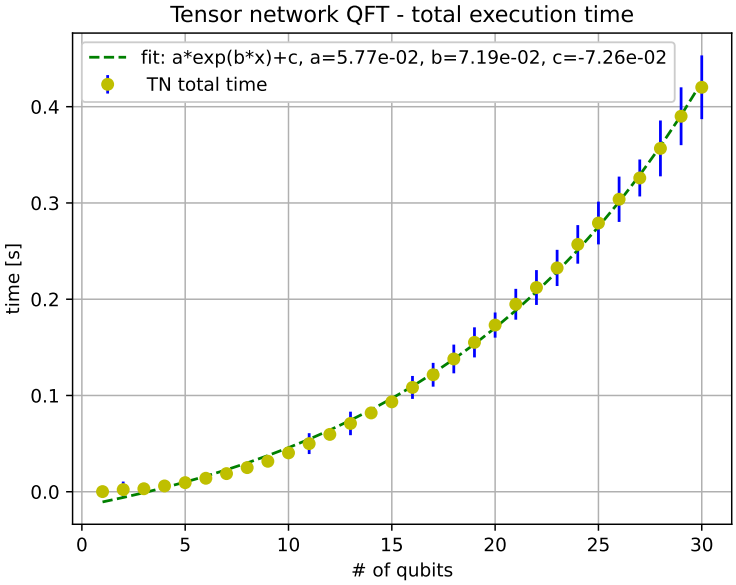


Figure 3.5: Tensor network QFT - execution time as a function of the number of qubits



We also compared the accuracy of the tensor network simulation to that of the exact state vector simulation: I constructed tensor network and state vector circuits for the quantum Fourier transform ranging from 1 to 28 qubits. I then measured all output samples and constructed frequency vectors for both circuit types. In the figure below are shown the frequency arrays for a 28 qubit QFT (Bra: $|Bra\rangle = |00...0\rangle$).

Figure 3.6: 28 qubit tensor network QFT frequency array

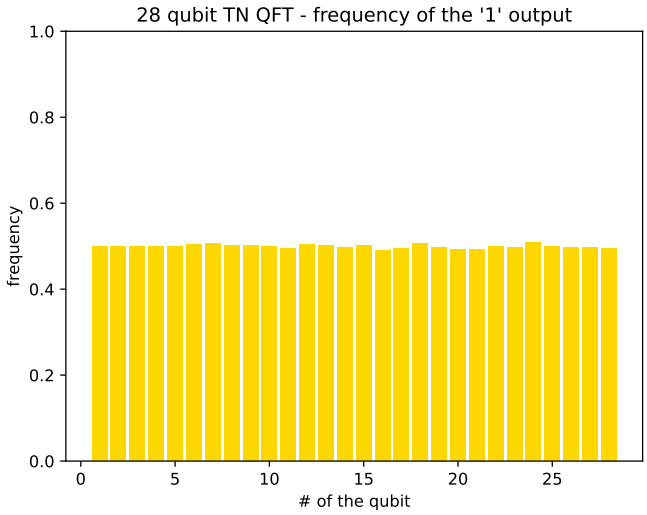
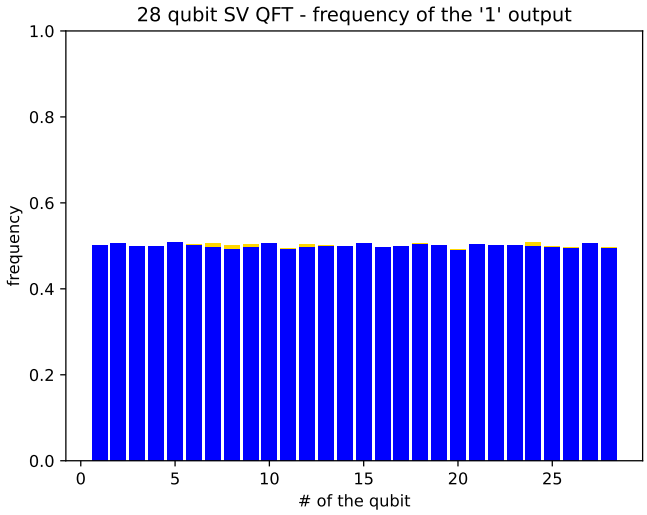


Figure 3.7: 28 qubit state vector QFT frequency array



By taking the difference between the frequency vectors of the tensor network and state vector circuits, I calculated the l^2 norm of the difference vector to understand the error behavior of the tensor network simulation compared to the state vector simulation.

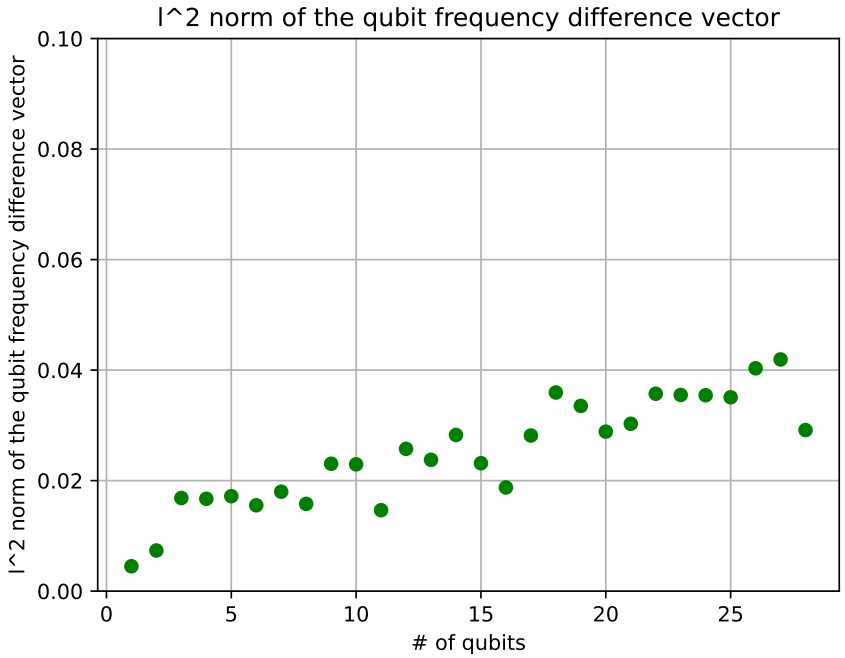
The l^2 norm of a vector $\mathbf{v}$ with n components is defined as the square root of the sum of the squares of its components v_i:

$$|\mathbf{v}|^2 = \sqrt{\sum_{i=1}^{n} v_i^2} \tag{3.7}$$

I found that the total error of the tensor network simulation slightly worsen as the number of qubits increases: the absolute error increases due to the presence of more qubits and hence, more contributions to the l^2 norm of the difference vector.

As such, to mitigate this error contribution, it is recommended to increase the number of samples taken by the tensor network simulation for high qubit counts.

Figure 3.8: l^2-norm of the difference between sampled frequencies in the TN vs SV implementations of the QFT, as a function of the number of qubits



3.3.2 Hidden Shift

The Hidden Shift Problem is an important problem in quantum computing with practical implications in cryptography and computer science. It is one of the known problems whose quantum algorithm solution shows an exponential speedup over classical computing. Part of the advantage lies in the ability to perform Fourier transforms efficiently, which can be used to extract correlations between certain functions. [34], [24]

The Hidden Shift problem can be defined as follows: Given an oracle O that encodes two functions f and g, there is an n-bit string s for which $g(x) = f(x \oplus s)$ for all x. The goal is to find s. Many functions, such as the Legendre symbol and Bent functions, satisfy these constraints.

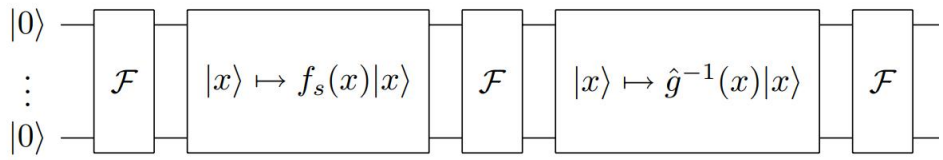
A quantum algorithm can be used to solve the Hidden Shift problem efficiently. The algorithm involves preparing a superposition of all possible values of the function using a series of quantum gates, including Hadamard gates and controlled phase gates, and then applying the quantum Fourier transform to extract the period of the function.[29]

The circuit for the Hidden Shift problem can be expressed as:

$$|s\rangle = H^{\otimes n} O_f H^{\otimes n} O_{\hat{g}} H^{\otimes n} |0^n\rangle, \tag{3.8}$$

where H is the Hadamard gate, O_f and $O_{\hat{g}}$ are the oracle gates for f and the Fourier transform of g respectively, and $|0^n\rangle$ is the n-qubit all-zero state (ground state).

Figure 3.9: Circuit diagram for a known function g and an unknown shift s of the black-box $f_s(x) = g(x + s)$



Simulation

The specific Hidden Shift problem I implemented is as follows:

Consider two functions f and g defined as $f, g : 0, 1^N \to 0, 1$, such that $g(x) = f(x \oplus s)$ for all $x \in 0, 1^N$, where s is a hidden bit string (the $\otimes$ is the XOR operation).

In this example, the functions considered are the so-called "bent" functions defined as $f(x) = \sum_{i=0}^{N/2-1} x_{2i} \cdot x_{2i+1}$, where x_i is the i-th bit of x.

Classically, the Hidden Shift Algorithm requires $2^{N/2}$ queries, while quantumly, it can be solved in $O(N)$ operations. The steps of the algorithm are as follows:

1. Prepare the quantum state in the initial state $|0\rangle^N$
2. Make a superposition of all inputs $|x\rangle$ with a set of Hadamard gates, which act as a (Quantum) Fourier Transform.
3. Compute the shifted function $g(x) = f(x \oplus s)$ into the phase with a proper set of gates. This is done first by shifting the state $|x\rangle$ with X gates, then implementing the bent function as a series of Controlled-Z gates, and finally recovering the $|x\rangle$ states with another set of X gates.
4. Apply a Fourier Transform to generate another superposition of states with an extra phase that is added to $f(x \oplus s)$.
5. Query the oracle f into the phase with a proper set of controlled gates. One can then prove that the phases simplify giving just a superposition with a phase depending directly on the shift.
6. Apply another set of Hadamard gates which act now as an Inverse Fourier Transform to get the state $|s\rangle$
7. Measure the resulting state to get s.

Note that we only query g and f once to solve the problem.

Results

Both the state vector and tensor network implementations of this specific hidden shift problem successfully achieve their intended purpose of discovering the secret key (the shift) bit-string, with each sampled output producing the correct result.

However, when considering the computational efficiency of the state vector circuits and the tensor network ones, we observe a significant disparity. Specifically, we find that the state vector simulation exhibits a much steeper curve of execution time as a function of the number of qubits in the circuit, in contrast to the tensor network implementation: although the state vector simulation outperforms the tensor network implementation for circuits of up to approximately 20 qubits, its computational cost increases drastically for larger circuits, ultimately making the tensor network approach more efficient.

I attribute this behavior to the unique architecture of the specific hidden shift problem/circuit in question, where qubits are entangled and operate in pairs rather than with all other qubits, as is the case with the quantum Fourier transform.

Figure 3.10: Tensor network HS - execution time as a function of the number of qubits

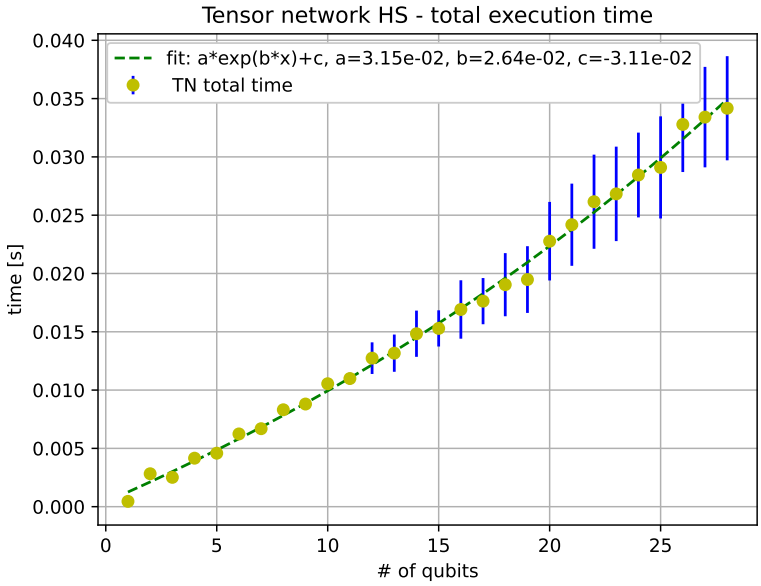
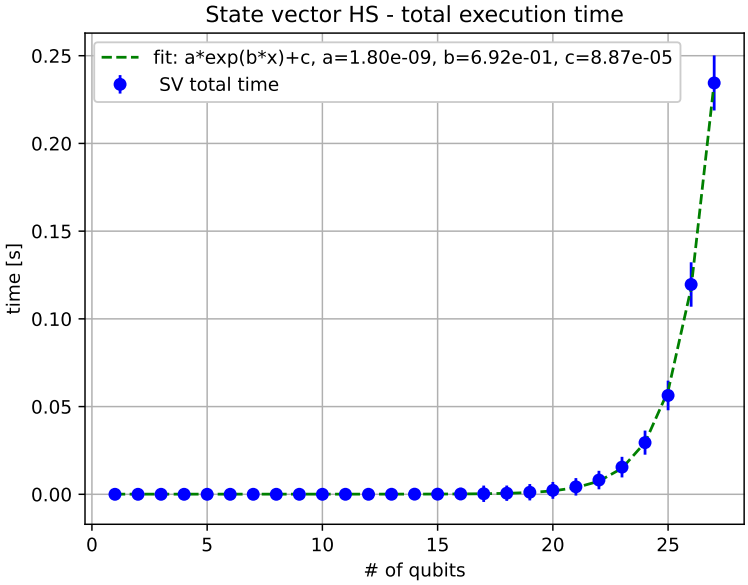


Figure 3.11: State vector HS - execution time as a function of the number of qubits



Chapter 4

Conclusions

Based on the comparison of the results obtained by sampling the two outcomes of the tensor network and the "exact" state vector simulations, I observed that the tensor network simulation did not deteriorate in accuracy with an increasing number of qubits.

The approximations made during tensor network construction (e.g. gauging, singular value decomposition + truncation, canonical polyadic decomposition of tensors) do not compromise the quality of the results of the simulations while significantly reducing program execution time and computational burden. In fact, the tensor network method executes much faster and its approximations do not lead to significant accuracy loss.

In conclusion, for circuits with a reduced number of qubits and layers of gates, no remarkable difference has been observed. However, for simulations that are computationally intensive, such as those involving a high number of qubits or layers of gates, tensor network simulations are a more efficient option.

Nonetheless, it is important to consider the circuit dependency of the speedup achieved by tensor networks, as this efficiency may vary depending on the specific characteristics of the quantum circuit being simulated. Moreover, while tensor network simulations have been shown to provide accurate results for many quantum circuits, there may be cases where the approximations made during tensor network construction lead to significant accuracy loss.

[1]

[1] You can find all the code used in this thesis at the following GitHub repository: https://github.com/TommasoGalletti/TN_quantum_simulation

Bibliography

[1] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, 1995.

[2] C. H. Bennett and D. P. DiVincenzo. Quantum information and computation. *Nature*, 404(6775):247–255, 2000.

[3] S. Carrazza, S. Efthymiou, M. Lazzarin, and A. Pasquale. An open-source modular framework for quantum computing. *Journal of Physics: Conference Series*, 2438(1):012148, feb 2023.

[4] S. Efthymiou, M. Lazzarin, A. Pasquale, and S. Carrazza. Quantum simulation with just-in-time compilation. *Quantum*, 6:814, Sept. 2022.

[5] S. Efthymiou, S. Ramos-Calderer, C. Bravo-Prieto, A. Pérez-Salinas, D. García-Martín, A. Garcia-Saez, J. I. Latorre, and S. Carrazza. Qibo: a framework for quantum simulation with hardware acceleration. *Quantum Science and Technology*, 7(1):015018, dec 2021.

[6] J. Gray and S. Kourtis. Hyper-optimized tensor network contraction. *Quantum*, 5:410, mar 2021.

[7] L. K. Grover. A fast quantum mechanical algorithm for database search. *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[8] M. Horodecki, P. Horodecki, R. Horodecki, and K. Horodecki. Entanglement in quantum information theory. *Rev. Mod. Phys.*, 81(2):865–942, 2009.

[9] T. Kim and B.-S. Choi. Efficient decomposition methods for controlled-r n using a single ancillary qubit. *Scientific reports*, 8(1):1–7, 2018.

[10] M. Lanzagorta and J. Uhlmann. *Quantum Computer Science*. Synthesis lectures on quantum computing. Morgan & Claypool Publishers, 2009.

[11] S. Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–1078, 1996.

[12] I. L. Markov and Y. Shi. Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing*, 38(3):963–981, 2008.

[13] N. D. Mermin. *Quantum Computer Science*. Cambridge University Press, 2007.

[14] R. Minster, I. Viviano, X. Liu, and G. Ballard. Cp decomposition for tensors via alternating least squares with qr decomposition, 2021.

[15] S. Montanaro. Quantum algorithms: an overview. *npj Quantum Information*, 2(1):15023, 2016.

[16] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[17] M. A. Nielsen. Quantum information theory. *Reports on Progress in Physics*, 70(4):1–51, 2007.

[18] R. Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014.

[19] F. Pan and P. Zhang. Simulation of quantum circuits using the big-batch tensor network method. *Phys. Rev. Lett.*, 128:030501, Jan 2022.

[20] R. Penrose. *Techniques of Differential Topology in Relativity*. SIAM, 1971.

[21] A. Peres. *Quantum Theory: Concepts and Methods*. Kluwer Academic Publishers, 1993.

[22] D. Perez-Garcia, F. Verstraete, M. M. Wolf, and J. I. Cirac. Matrix product state representations. *Quantum Info. Comput.*, 7(5):401–430, jul 2007.

[23] J. Preskill. Quantum information and computation. *arXiv preprint quant-ph/9712048*, 1998.

[24] M. Roetteler. Quantum algorithms for highly non-linear boolean functions, 2009.

[25] S. Sánchez Ramírez. Large-scale cuantum computing simulation using tensor networks. Master's thesis, Universitat Politècnica de Catalunya, 2021.

[26] V. Scarani, S. Iblisdir, N. Gisin, and A. Acin. Quantum cloning. *Reviews of Modern Physics*, 77(4):1225–1256, 2005.

[27] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE, 1994.

[28] E. M. Stoudenmire and D. J. Schwab. Quimb: A python library for quantum information and many-body theory. *Journal of Open Source Software*, 1(3), 2016.

[29] W. van Dam, S. Hallgren, and L. Ip. Quantum algorithms for some hidden shift problems. *SIAM Journal on Computing*, 31(2):417–426, 2002.

[30] G. Vidal. Efficient classical simulation of slightly entangled quantum computations. *Phys. Rev. Lett.*, 91:147902, Oct 2003.

[31] J. D. Whitfield, J. Biamonte, and A. Aspuru-Guzik. Simulation of electronic structure hamiltonians using quantum computers. *Molecular Physics*, 109(5):735–750, 2011.

[32] C. Williams. *Explorations in Quantum Computing*. Texts in Computer Science. Springer London, 2010.

[33] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.

[34] K. Wright, K. M. Beck, S. Debnath, J. M. Amini, Y. Nam, N. Grzesiak, J.-S. Chen, N. C. Pisenti, M. Chmielewski, C. Collins, K. M. Hudek, J. Mizrahi, J. D. Wong-Campos, S. Allen, J. Apisdorf, P. Solomon, M. Williams, A. M. Ducore, A. Blinov, S. M. Kreikemeier, V. Chaplin, M. Keesan, C. Monroe, and J. Kim. Benchmarking an 11-qubit quantum computer. *Nature Communications*, 10(1), nov 2019.