Simulation of Quantum Fourier Transform through tensor networks approach

Edara Yaswanth Balaji
EP22BTECH11007

Project Supervisor: Dr. Archak Purkayastha

Department of Physics
Indian Institute of Technology, Hyderabad

December 1, 2024

Abstract

This report investigates the use of Tensor Networks for the classical simulation of Quantum Fourier Transform (QFT), a fundamental operation in quantum computing. The QFT, essential to various quantum algorithms, is typically computationally expensive due to its large unitary matrix representation. However, leveraging the inherent low entanglement in QFT allows for efficient simulation using Tensor Networks, specifically using Matrix Product States (MPS) and Matrix Product Operators (MPO). This work implements the QFT simulation on classical computers, comparing its performance with the classical Fast Fourier Transform (FFT) algorithm. Through empirical analysis, we demonstrate that QFT with Tensor Networks provides significant runtime advantages for structured data, particularly as the number of qubits increases. However, for randomly structured data, FFT remains the more efficient approach. This study highlights the potential of quantum-inspired classical algorithms, opening avenues for faster simulations in areas such as signal processing, data analysis, fluid dynamics and quantum chemistry simulations.

1 Introduction

1.1 Fourier Transform

The Fourier Transform is a mathematical technique that transforms a signal from its original domain, typically time or space, into the frequency domain. It plays a fundamental role in various fields, including signal processing, image processing, communications, and data analysis. The Discrete Fourier Transform (DFT) computes the Fourier Transform for discrete signals, and its efficient computation is facilitated by the Fast Fourier Transform (FFT) algorithm. The Discrete Fourier Transform (DFT) of a signal $x[n]$ of length N is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i\frac{2\pi}{N}kn}, \quad k = 0, 1, 2, \ldots, N-1$$

The Discrete Fourier Transform (DFT) of a signal $\mathbf{x} = [x_0, x_1, \ldots, x_{N-1}]^T$ can be represented as a matrix multiplication, where the DFT matrix $\mathbf{F}_N$ is given by:

$$\mathbf{F}_N = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & e^{-\frac{2\pi i}{N}} & e^{-\frac{4\pi i}{N}} & \ldots & e^{-\frac{2(N-1)\pi i}{N}} \\ 1 & e^{-\frac{4\pi i}{N}} & e^{-\frac{8\pi i}{N}} & \ldots & e^{-\frac{4(N-1)\pi i}{N}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & e^{-\frac{2(N-1)\pi i}{N}} & e^{-\frac{4(N-1)\pi i}{N}} & \ldots & e^{-\frac{2(N-1)(N-1)\pi i}{N}} \end{bmatrix}$$

$$\mathbf{X} = \mathbf{F}_N \mathbf{x}$$

The **Fast Fourier Transform (FFT)** is an optimized algorithm for computing the Discrete Fourier Transform (DFT), reducing the computational complexity from $O(N^2)$ to approximately $O(N \log N)$, making it practical for real-time applications and large datasets. However, as the size of the data grows, the time required for FFT also increases, especially for large datasets with a size greater than 2^{20}, prompting the search for more efficient algorithms for transforming large signals.

1.2 Quantum Computing

Quantum computing is based on the principles of quantum mechanics, specifically superposition, entanglement, and quantum interference. Unlike classical bits, quantum bits (or qubits) can exist in a superposition of states, meaning they can represent both 0 and 1 simultaneously. This property allows quantum computers to perform many calculations in parallel. Additionally, qubits can become entangled, which means the state of one qubit is inherently linked to the state of another, regardless of the distance between them. Quantum circuits are built using quantum gates, represented by unitary matrices that manipulate qubit states. Quantum operations are probabilistic in nature, meaning that upon measurement, the outcome is not deterministic but rather occurs with a certain probability distribution.

The **Quantum Fourier Transform (QFT)** is a quantum algorithm that transforms a quantum state $|x\rangle$ into its Fourier components. It is a quantum analog of the classical **Discrete Fourier Transform (DFT)**, but it operates on quantum bits. The main difference between QFT and the classical FFT is that QFT uses quantum superposition to transform the state into its Fourier representation with exponential speedup in some cases.

The QFT of a quantum state $|x\rangle$, where x is an integer, is defined as:

$$QFT|x\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i \frac{xk}{N}} |k\rangle$$

Where $N = 2^n$ (for n qubits), x is the computational basis state, and k is the Fourier component. The QFT matrix is a unitary matrix, and its application on the quantum state $|x\rangle$ yields the Fourier-transformed state $|k\rangle$.

The QFT can be implemented efficiently using a quantum circuit that involves **Hadamard gates** and **controlled-phase gates**. These gates apply phase shifts to the qubits in a way that corresponds to the Fourier coefficients. The quantum circuit for QFT on n qubits can be visualized as a series of Hadamard gates followed by controlled phase gates. The matrix representation of the QFT on n-qubits is:

$$QFT_n = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \sum_{k=0}^{N-1} e^{2\pi i \frac{xk}{N}} |x\rangle\langle k|$$

This represents the QFT as a unitary matrix applied to a quantum state. This is exactly the same as the matrix $\mathbf{F}_N$ that is used to perform DFT. This tells us that the same operation is applied here using unitary matrices.

2 Quantum Fourier Transform

The quantum circuit for implementing the QFT can be broken down into three main components:

- **Hadamard Gate (H)**: This gate is applied to the first qubit and creates a superposition of all states.
- **Controlled-Phase Gates ($\mathbf{C}_\theta$)**: These gates are used to introduce the necessary phase shifts between qubits. The phase angle θ is related to the position of the qubits in the quantum state.
- **Swap Gates**: These are used to reverse the order of the qubits, as the QFT operation works from the highest to the lowest qubit.

The QFT can be implemented efficiently using these gates in a logarithmic number of operations, making it exponentially faster than classical methods for large datasets. There are mainly two different ways of apply QFT using quantum gates.

- **QFT circuit with swap gates**: In this circuit, each controlled-phase gate (R_k) is followed by a swap gate. The algorithm is recursive in nature with similar structure for any number of qubits. But at the end of the algorithm, we get the results in the reverse order to the input if we implement this circuit. (figure 1)
- **QFT circuit without swap gates**: In the QFT circuit without swapping, the circuit is designed similarly but changing the structure slightly to avoid the use of swap gates. Both the circuits implement the same algorithm but with a different set of gates. (figure 2)

While the Quantum Fourier Transform (QFT) provides a powerful theoretical advantage in certain quantum algorithms, its direct application on quantum hardware faces several significant limitations. These limitations arise from both the inherent complexity of quantum circuits and the resource constraints of current quantum computers. The primary challenges include the following:

1. **Exponential Growth of Matrix Size**
   One of the key limitations in applying QFT directly to quantum systems is the exponential growth of the matrix size with respect to the number of qubits. For a quantum system with n qubits, the corresponding QFT matrix is of size $2^n \times 2^n$, which grows exponentially as n increases. This results in a matrix that is not only computationally infeasible to compute directly but also difficult to store.

2. **Input Size Constraints**
   Another challenge when applying QFT on a quantum computer is the requirement for the input size to be a power of two, i.e., 2^n. This restriction is similar to the Fast Fourier Transform (FFT) in classical computing, where even the FFT is most efficient when the number of data points is a power of two.

So the simulation of QFT on a classical computer requires efficient optimization techniques for runtime faster than FFT while maintaining good accuracy.

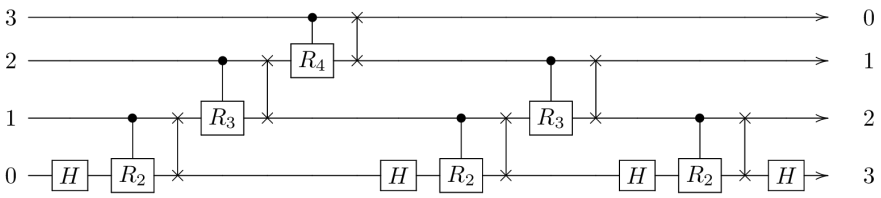


Figure 1: Quantum circuit for QFT algorithm using swap gates

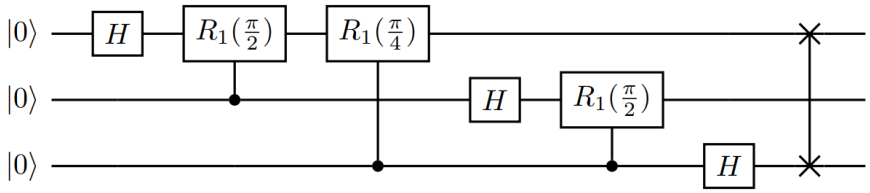


Figure 2: Quantum circuit for QFT algorithm without using swap gates

3 Tensor Networks

Quantum computing has the potential to revolutionize many fields, but directly simulating quantum algorithms on classical computers is challenging due to the exponential growth of the unitary matrices required to represent quantum gates. As the number of qubits n increases, the size of these matrices grows exponentially, making direct application on a classical computer infeasible. The unitary matrices that apply quantum gates become prohibitively large, and so applying them directly would require an enormous amount of computational power and memory.

To address this issue, quantum-inspired classical algorithms leverage **tensor networks** to represent quantum states and operators more efficiently. Tensor networks provide a compact representation by decomposing large tensors into smaller tensors, significantly reducing the computational cost of simulating quantum circuits. Tensor networks are the most powerful simulating architecture and they help us in converting quantum algorithm to quantum-inspired classical algorithm

3.1 Matrix Product State (MPS)

Matrix Product States (MPS) are an efficient representation of quantum states in a one-dimensional lattice of qubits. MPS can represent quantum states in a way that reduces the exponential scaling of the Hilbert space, particularly when the quantum state exhibits low entanglement. This efficiency is achieved by decomposing the state vector into a series of matrices connected by shared indices, where each matrix corresponds to a qubit in the system. This is done by finding similarities between the individual small tensors and exploiting their structure. The tensor is then iteratively decomposed using **Singular Value Decomposition (SVD)**, a technique that decomposes a tensor into simpler components. By applying SVD repeatedly, till we are able to approximate the original tensor with a much smaller rank.

For a matrix $A \in R^{m \times n}$, the SVD is given by:

$$A = U\Sigma V^T$$

where:

- U is an orthogonal matrix of size $m \times m$,
- Σ is a diagonal matrix of size $m \times n$ with elements in decreasing order,
- V^T is an orthogonal matrix of size $n \times n$.

Now in every iteration, we keep the most significant components while discarding the ones with lower values by setting a cutoff limit to reduce the order. By doing this the order of Σ changes to $p \times p$, U changes to $m \times p$, and the order of V^T changes to $p \times n$. We then multiply V^T and Σ to get another Tensor of lesser order. Then we proceed to apply SVD to this tensor and repeat the above process. Here, p is defined as the **bond order** which depends on the cutoff limit that we set.

$$\mathcal{A} = \mathcal{U}_i^{[1]} \cdot \mathcal{U}_i^{[2]} \cdot \cdots \cdot \mathcal{U}_i^{[r]} \cdot \Sigma V^T$$

By setting a good cutoff limit and making sure the tensor created in this way is close to the original Tensor, we can reduce its order by a lot. We can verify the error by calculating the modulus of the tensors which is just the sum of squares of elements in Σ. This way of representing the state is called **Matrix Product State (MPS)**

This approach makes it feasible to simulate large quantum systems on classical computers, enabling efficient quantum simulations without having to explicitly compute or store the full unitary matrices. Tensor networks, particularly Matrix Product States have become essential tools in the simulation of quantum circuits, as each tensor in the MPS can be considered as a qubit when we are simulating quantum circuits.

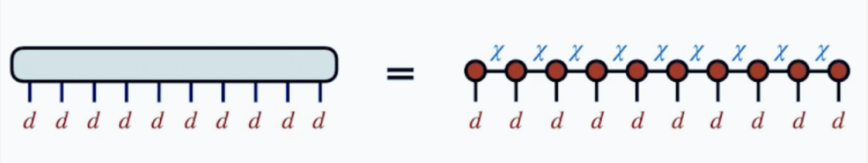


Figure 3: Conversion of a large tensor into MPS using SVD with a bond order of x

3.2 Matrix Product Operator (MPO)

Matrix Product Operators (MPO) offer an efficient way to represent quantum operators, particularly in the simulation of quantum circuits. They are an extension of Matrix Product States (MPS), where instead of representing quantum states, MPOs represent operators acting on quantum states. This representation is particularly powerful for simulating quantum circuits on classical computers, as it allows the operator to be precomputed and stored in a compressed form. By decomposing quantum operations into smaller tensors, MPOs significantly reduce the computational cost and memory required for quantum simulations.

The typical process for using MPOs involves multiplying all the quantum gates involved in a circuit with the identity MPO. The resulting operator is then stored as an MPO in a file, which can later be read and multiplied with any input MPS. This process eliminates the need to apply gates to the quantum state every time, drastically improving efficiency. Instead, the MPO operator is already precomputed, reducing both time and computational resources.

The MPO format enables storing the full quantum circuit as a single operator that can be directly multiplied with an MPS. This method is particularly beneficial when simulating quantum operations over large qubit systems, as it avoids the exponential growth of matrix sizes. The efficiency comes from the fact that an MPO can be created and stored once, and reused as needed for different quantum states.

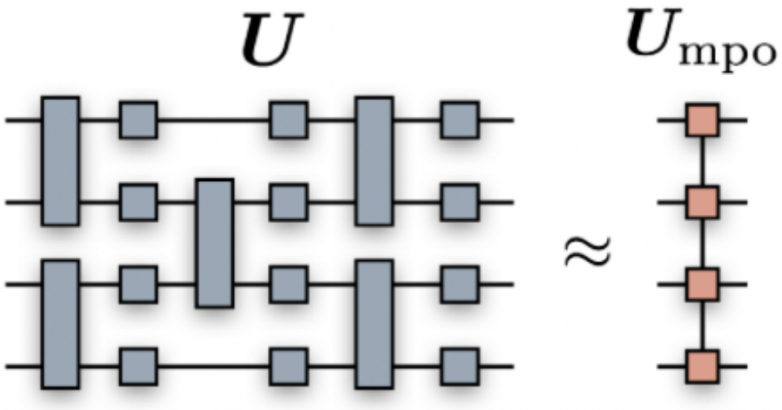


Figure 4: Converting a combination of quantum gates into MPO

3.3 Efficiency of Tensor Network Simulations

Tensor network simulations, such as those using Matrix Product States (MPS) and Matrix Product Operators (MPO), are most efficient when applied to quantum systems with **low entanglement**. The Quantum Fourier Transform is well-suited to tensor network simulation because it involves low entanglement. This property allows us to construct the MPO

for QFT efficiently, using a **low bond order**, which significantly reduces the computational complexity and memory requirements for large quantum systems.

It also depends on other factors such as the the cutoff-limit that we set when we are creating the input MPS. We need to choose an appropriate cutoff to make sure we get accurate results while also gaining a clear time advantage over the classical scenario.

4 Implementation

For the simulation of the Quantum Fourier Transform (QFT) using tensor networks, we utilize the **Julia programming language** and **ITensors library** . In this context, each qubit in the quantum circuit is represented as a site in the MPS, a format used to represent quantum states in MPS. These sites are indexed and connected similar to an a regular MPS structure, allowing for efficient manipulation of the quantum state.

One particular feature of the ITensors library in Julia is the handling of qubit data. The MPS function in ITensors that converts an array directly into MPS, reverses the data order during processing. However, when multiplying this MPS with the MPO that we create, the data order is reversed back to its original form when we use QFT algorithm. This ensures that the final result maintains the correct ordering of the qubits, which is crucial for accurate results in the same order as produced by FFTW.

The process of simulating QFT using tensor networks can be broken down into the following steps:

- **Input Handling**: The program takes the input as an array or tensor, which represents the quantum state of the system.
- **MPS Conversion**: The input tensor is then converted into a Matrix Product State (MPS) using the function available, breaking the quantum state into smaller tensors that are easier to manipulate.
- **MPO creation**: Quanntum gates are defined and then used to compute MPO by ordering them in the same way as the QFT algorithm. The MPO, representing the quantum gates for the QFT operation, is precomputed and stored.
- **MPS and MPO multiplication**: The MPO is constructed in advance to avoid recalculating the gates for every simulation. The MPO is then multiplied with the MPS, applying the QFT operation to the quantum state.
- **Output**: The resulting quantum state after applying the QFT is the output of the simulation. We can verify it by a dot product with the basis states and get the required values in an array. This can be compared to the results of FFTW.

In the implementation, it is essential that the site indices of the Matrix Product States (MPS) and Matrix Product Operators (MPO) match for the multiplication to work correctly. During the initialization of the sites for the MPS, the site indices are read from the precomputed MPO files. This ensures that the indices of the MPS align properly with those in the MPO, enabling valid tensor multiplication. Specifically, the indices in the MPS must correspond to the qubits represented in the MPO. By ensuring the correct alignment of these indices, we guarantee that the quantum state is transformed accurately by the MPO, avoiding any misalignment or errors during the simulation.

This approach ensures that the QFT operation is computed efficiently by leveraging tensor network representations and reusing precomputed MPOs.

4.1 Bond Dimension Comparison for qft algorithms

When applying the quantum algorithm with swap gates, the bond dimension of the Matrix Product States (MPS) increases progressively with the application of each gate. This increase in bond dimension adds computational overhead, making the simulation more resource-intensive as the number of gates and qubits grows. On the other hand, when using the algorithm without swap gates, the bond dimension remains relatively stable, resulting in a more efficient computation.

For example, as shown in the graph below for 10 qubits, the bond dimension for the algorithm that includes swap gates is significantly higher compared to the algorithm without swap gates. The gates are applied layer by layer in a total of 10 layers and the bond order of the input MPS after every layer is checked and plotted below. This makes us choose the algorithm without including the swap gates for most efficiency.

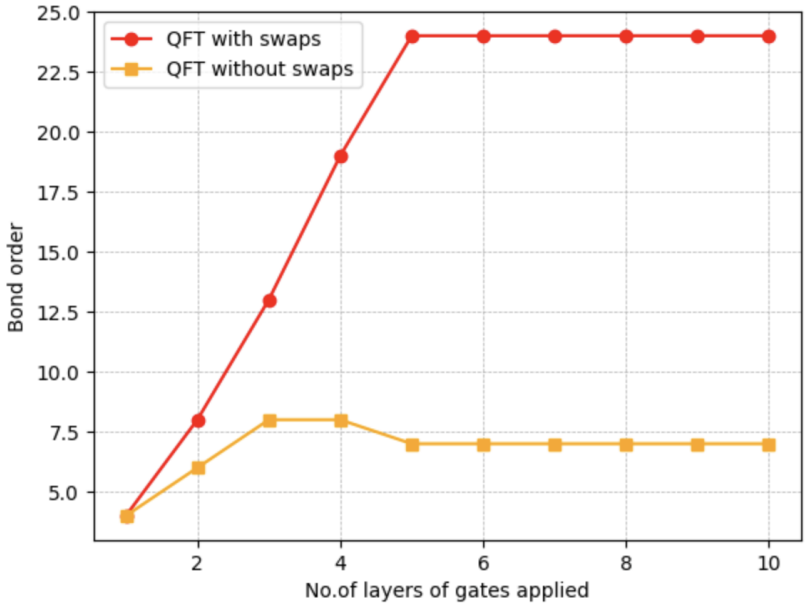


Figure 5: Comparing the bond order of the MPS when the quantum gates for qft are being applied gradually layer by layer for a n=10 qubit input

4.2 Verifying the Output

To verify the correctness of the simulation, we compare the results obtained from our simulation with the outputs of the classical Fast Fourier Transform (FFT) algorithm. The FFT implementation is performed using the widely-used **FFTW** library in Julia, which is known for its efficient and optimized computation of the discrete Fourier transform. By comparing the quantum simulation results with those obtained from the FFT algorithm, we ensure that the tensor network approach correctly approximates the QFT operation. The results comparing the FFT output values and QFT output values are shown in the image below. These values corrospond to the first 8 values in the fourier transform of a **cosine function with 2^{18} values in the input array**.

```
0.9999999999897371 + 0.0im
131072.24998792697 + 1.5707993227664905im
-0.3333367240723126 - 7.989564555325709e-6im
-0.12500107278255507 - 4.494122164867662e-6im
-0.06666720910194392 - 3.195819330475668e-6im
-0.04166699770753095 - 2.4967312395262473e-6im
-0.028571652684777963 - 2.054454686342523e-6im
-0.020833495493452498 - 1.7477150062777313e-6im
```

(a) fft output values

```
1.0000000015010373 + 1.0787740962585952e-11im
131072.24998794138 + 1.5707992797225074im
-0.3333367223195199 - 7.989483647659708e-6im
-0.12500107162919707 - 4.494088396464478e-6im
-0.06262365481479092 - 3.0019686202342367e-6im
-0.04166704359218955 - 2.496565990403811e-6im
-0.03567050567295504 - 2.5648787001972964e-6im
-0.02083350938717971 - 1.7442987619644713e-6im
```

(b) qft output values

Figure 6: Verifying the accuracy of qft algorithm by comparing the values

4.3 Runtime Comparison: FFT vs QFT

In this section, we present a series of plots comparing the runtime of the Fast Fourier Transform (FFT) and the Quantum Fourier Transform (QFT), as simulated using tensor networks. The plots depict the computational time required for different numbers of qubits, input data types, and cutoff limits. Each plot shows the performance of the FFT algorithm, implemented using the FFTW library in Julia, alongside the performance of the QFT simulation using the tensor network approach.

It is important to note that the time plotted in the graph below reflects only the computation of the QFT, **excluding the initial time required to convert the input data into Matrix Product States (MPS)**

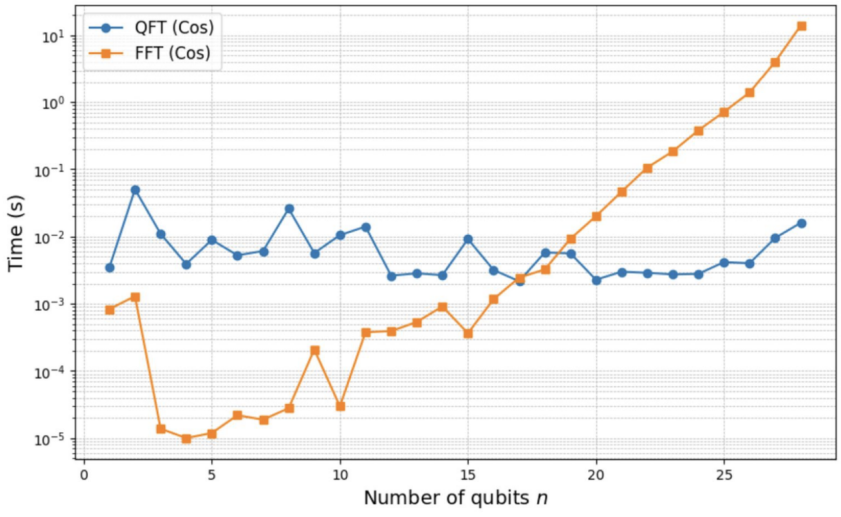
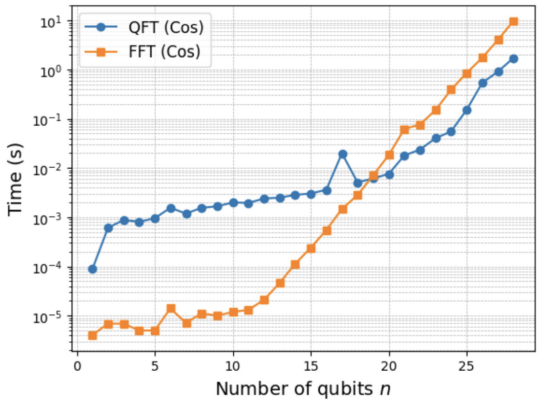


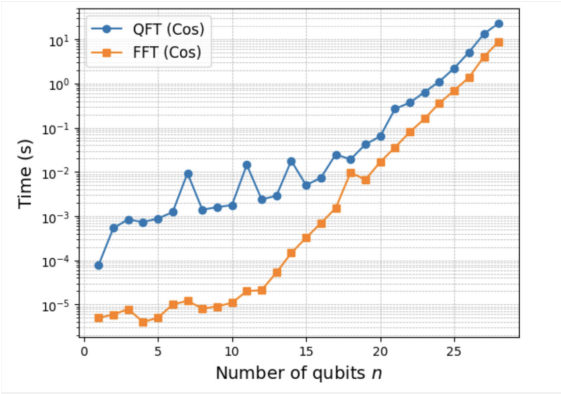
Figure 7: QFT vs FFT for a cosine function input and a cutoff value of 10^{-18}

When considering the time required to convert the input data into Matrix Product States (MPS), the advantage of using the Quantum Fourier Transform (QFT) over the classical **Fast Fourier Transform (FFT) diminishes, especially for larger cutoff values. As the cutoff value required to achieve higher accuracy moves closer to zero, the time required for MPS conversion becomes significant, reducing the overall efficiency of the QFT algorithm in comparison to FFT.

In cases where high accuracy is needed, a larger bond order may be required, as it captures more detailed features of the quantum state. However, this comes at the cost of increased computational resources and time. The graphs presented below illustrate these results.



(a) Cutoff value - 10^{-12}



(b) Cutoff value - 10^{-18}

Figure 8: These are time comparision plots where QFT time also includes the time required to convert the data to MPS with different cutoff values

4.4 Runtime Dependencies

The runtime of the Quantum Fourier Transform (QFT) simulation still depends on several factors, including the type of data being processed and the cutoff value used when converting the data to Matrix Product States (MPS).

- **Type of data**: It plays a significant role in determining the efficiency of QFT compared to the classical Fast Fourier Transform (FFT). QFT performs better than FFT only when there is some inherent structure in the data, such as periodicity or patterns that can be exploited by the tensor network representation. However, when the data is random, FFT consistently outperforms QFT due to its simplicity and directness in handling such inputs.

- **Cutoff Value**:This is another important factor used comes into play during the conversion of data into MPS. If the cutoff is set too low, more computational time is required to calculate the MPS, as more terms need to be retained to maintain accuracy. On the other hand, setting the cutoff too high may result in reduced accuracy. Therefore, selecting an optimal cutoff value is essential, as it strikes a balance between accuracy and computational time, ensuring that the simulation is both efficient and precise.

5 Future Work

There are several avenues for future work to enhance the current approach and extend its applications:

- **MPS conversion**: Develop a faster method to convert input data into Matrix Product States (MPS). The current conversion process represents a significant portion of the computational time, and improving its efficiency would make quantum simulations more scalable, especially for larger systems.

- **Use QFT instead of FFT**: Using this algorithm as an alternative to the classical Fast Fourier Transform (FFT). This approach aims to achieve significant time advantages in various applications, such as simulations and signal processing.

- **MPS based algorithm**: This approach aims to achieve significant time advantages by completely working with the data in MPS state from beginning till the end without the need of converting it from array to MPS and back. This will give us a time advantage over classical methods but requires new algorithms to be discovered as the classical algorithms do not work the same with MPS data structure.

- **Other Algorithms**: Try to simulate other quantum algorithms that are difficult or impossible to simulate classically. An example is Grover's algorithm, which provides a quantum speedup for unsorted database search problems. Adapting this quantum-inspired classical method would allow us to simulate Grover's algorithm in a classical setting, enabling more efficient solutions to problems that are typically restricted to quantum computing.

6 References

- Jielun Chen, E.M. Stoudenmire, and Steven R. White. **Quantum Fourier Transform Has Small Entanglement** *PRX Quantum*, 4, 040318 (2023). Chen, J., Stoudenmire, E. M., White, S. R. (2023). Quantum Fourier Transform Has Small Entanglement. *PRX Quantum*, 4(4), 040318. Department of Physics and Astronomy, University of California, Irvine, California 92697-4575, USA; Department of Physics, California Institute of Technology, Pasadena, California 91125, USA; Center for Computational Quantum Physics, Flatiron Institute, 162 Fifth Avenue, New York, New York 10010, USA.

- YouTube Video: *Tensor Networks Explanation Images taken from here.* Available at: `https://www.youtube.com/watch?v=fq3_7vBcj3g`.

- MPO and MPS ITensors code documentation to create the simulation code. Available at: `https://itensor.github.io/ITensors.jl/dev/MPSandMPO.html`.