Simulating the Quantum Fourier Transform

Francisco R. Pereira[1], **Elloá B. Guedes**[1,2], **Francisco M. de Assis**[1]

[1]Institute for Studies in Quantum Computation and Information
Federal University of Campina Grande
Av. Aprígio Veloso, 882 – 58429-900 – Campina Grande – PB – Brazil

[2]Superior School of Technology
University of the Amazonian State
Av. Darcy Vargas, 1200 – 69050-020 – Manaus – AM – Brazil

{revson.ee, elloaguedes, fmarassis}@gmail.com

Abstract. *Quantum Computing is a computational paradigm that takes into account the laws of Quantum Physics in the steps of the computation which advantages were verified both in Computation and Communications. No scalable quantum computer was developed so far and to execute, to test, and to create new quantum algorithms the simulation of quantum computers on classical computers plays an important role. In this work, we show the design, tools and results obtained for the simulation of the Quantum Fourier Transform algorithm. As a result, we developed an open-source tool, called* FTSimulator*; and we could simulate up to 12 qubits according to the procedures specified by an experimental test.*

1. Introduction

In 1982, Feynman observed that it did not appear possible for a Turing machine to simulate certain quantum physical processes without incurring an exponential slowdown [Feynman 1982]. Considering this difficulty, years latter Deutsch proposed a computing model – the quantum Turing machine – which considered the processes of Quantum Mechanics in the steps of computation [Deutsch 1985, Deutsch 1989]. With such contribution, Deutsch inaugurated the *Quantum Computing paradigm*.

The first algorithms proposed according to Quantum Computing already showed some advantages over their classical counterpart [Nielsen and Chuang 2010]. However, the first outstanding algorithm proposed according to this paradigm was the *quantum factorizing algorithm* [Shor 1997]. This algorithm has a polynomial-time complexity on the quantum Turing machine while no similar time complexity algorithm is known for classical computers. The *quantum search algorithm* is also a remarkable result [Grover 1997]. This algorithm has a quadratic speedup over classical algorithms on searching over an unsorted database.

The intrinsic properties of Quantum Mechanics such as superposition and entanglement motivated the research and proposition of quantum algorithms not only for computing, but also for communication systems [Imre and Gyongyosi 2012]. Developments made by literature updated some computing models such as automatons and grammars to take into account quantum properties in their computation [Bacon and van Dam 2010]. However, a formal proof that the Quantum Computing paradigm is better or not than the Classical/Probabilistic Computing paradigm is not known yet.

Despite the already consolidated results on quantum algorithms and the advantages verified, the practical implementation of a quantum computer is still far from being scalable. Different technologies are under research and some of them are very restrictive, demanding a very low temperature, a high degree of precision, among other hard to meet requirements [Ladd et al. 2010].

Considering the practical difficulty of building a quantum computer, despite the limitations of simulating quantum systems on classical computers, sometimes it is the only way to "execute" a quantum algorithm over some input. Given the importance of simulation in the current context of Quantum Computing, this article aims at presenting a simulation of the *Quantum Fourier Transform* algorithm. This algorithm is in the heart of the quantum factorization algorithm [Shor 1997] and was applied in the development of quantum codes [Santos et al. 2013], in the design of quantum attacks to pseudorandom generators [Guedes et al. 2013], and even in other quantum algorithms [Williams 2011].

To present such simulation, this article is organized as follows. The description of the Quantum Fourier Transform algorithm, its quantum circuit and applications are described in Sec. 2. The simulation of the Quantum Fourier Transform is described in Sec. 3 and its analysis in Sec. 4. Lastly, final remarks and future work are presented in Sec. 5.

Notations, Conventions and Suggestions Along this paper, the Dirac notation will be used to denote quantum states and operations. Moreover, $\mathbb{1}$ is the identity matrix and ι is the imaginary unity. If the reader is not familiar with Quantum Computing, the books of Nielsen and Chuang [Nielsen and Chuang 2010, Chapter 2] and of Williams [Williams 2011, Part I] are recommended.

2. Quantum Fourier Transform

The *Quantum Fourier transform* (QFT) is the quantum counterpart of the well known discrete Fourier transform. The QFT takes as input a quantum state and produces a superposition of quantum states as formalized in Definition 1.

Definition 1 (**Quantum Fourier Transform**). *Let $|j\rangle$ be an orthonormal vector in a Hilbert space $\mathcal{H}$ with dimension 2^m ($m > 0$), i.e., belonging to the basis $\{|0\rangle, |1\rangle, \dots, |2^m - 1\rangle\}$. The quantum Fourier transform of $|j\rangle$ is given by*

$$\mathrm{QFT}\,|j\rangle = \frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{\frac{2\cdot\pi\cdot k\cdot j\cdot\iota}{2^m}} |k\rangle\,. \tag{1}$$

The QFT is unitary, invertible and has quadratic polynomial time over the input size. The quantum circuit which implements such transform is shown on Figure 1. The matricial expression for the gates H and R_k are given by

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2\cdot\pi\cdot\iota}{2^k}} \end{bmatrix} \tag{2}$$

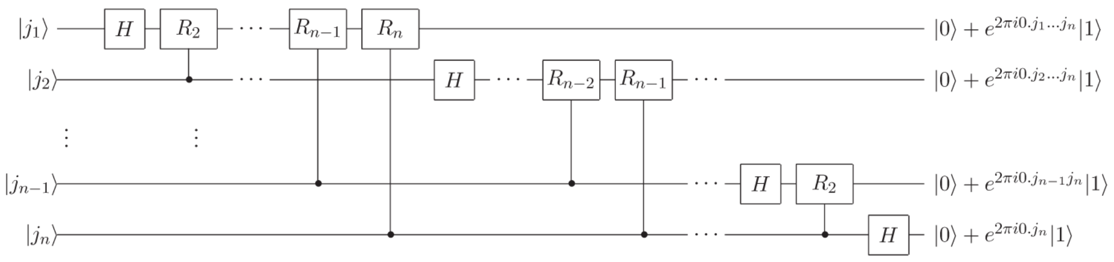

$|j_1\rangle$
$|j_2\rangle$
$|j_{n-1}\rangle$
$|j_n\rangle$
H
R_2
R_{n-1}
R_n
R_{n-2}
$|0\rangle + e^{2\pi i 0.j_1 \dots j_n}|1\rangle$
$|0\rangle + e^{2\pi i 0.j_2 \dots j_n}|1\rangle$
$|0\rangle + e^{2\pi i 0.j_{n-1} j_n}|1\rangle$
$|0\rangle + e^{2\pi i 0.j_n}|1\rangle$

Figure 1. Quantum circuit which implements the QFT [Nielsen and Chuang 2010].

The QFT is very important for Quantum Computation and Information. Its conception enabled the development of the quantum factorizing algorithm, the first one that was identified as having a superpolynomial speedup over its best classical counterpart [Shor 1997]. The practical implementation of these algorithms may harm the security of RSA-based systems widely used nowadays [Paar and Pelzl 2010]. Besides these applications, the QFT is used to compromise the unpredictability of cryptographically secure pseudorandom generators [Guedes et al. 2013], to calculate the decoding syndrome for quantum graph codes [Santos et al. 2013], etc.

3. Simulation of the Quantum Fourier Transform

In order to simulate the QFT in a classical computer, we developed a C++ software called *FTSimulator*. The C++ programming language was chosen because its advantages due to be compiled which results in a better efficiency when compared to interpreted programming languages [Grune et al. 2012]. The input for the simulation is a text file which describes the density matrix of the initial quantum state. The output file is also a density matrix containing the result of the input state transformed by the QFT.

The *FTSimulator* architecture follows the Model-View-Controller design pattern [Gamma et al. 1994]. The advantages of using this pattern is to enable a clear modularization and to favor the encapsulation of the classes implemented. An overview of the architecture of *FTSimulator* is shown on Figure 2. Some elements presented in the architecture will be depicted later.

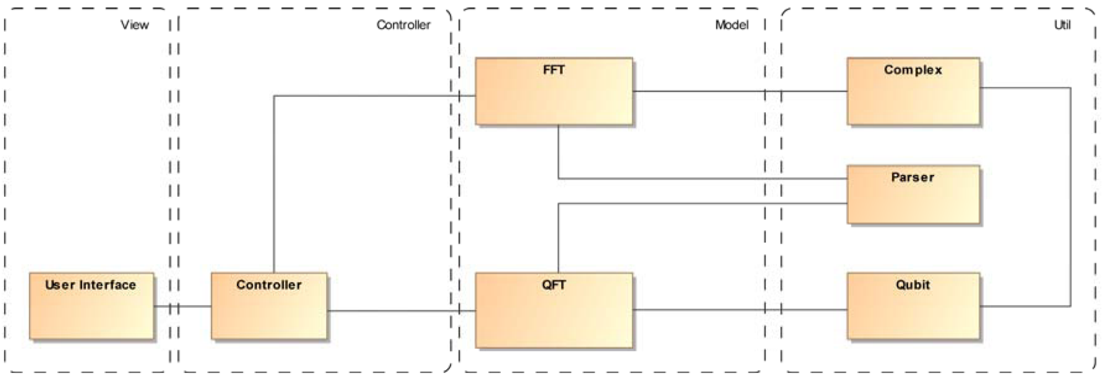


Figure 2. Architectural view of the *FTSimulator*.

The *View* part includes the textual interface with which the user interacts. The *Controller* part contains the layer that converts the user input into commands and that presents the results to the user. The *Model* part includes the QFT module and also the FFT module, that will be explained latter. Utilities classes can also be found, such as for complex numbers and qubits representation as well as a parser that reads the input files according to their formatting.

The QFT module considers the input as being a density matrix. Such representation enables entangled and superpositioned multi-qubits input. Given a density operator ρ, the *FTSimulator* will perform the operation $\mathrm{QFT}\,\rho\,\mathrm{QFT}^\dagger$, where † denotes the transpose conjugate. The difficulty is to build and apply the QFT operator for different input dimensions. To do so, the algorithm that builds the QFT operator of Eq. (1) is shown on Figure 3.

```
function CQFT::QFTOPERATOR(complex **qftOp, int N)
    for int $i = 0; i < N; i++$ do
        for int $j = 0; j < N; j++$ do
            qftOp[i][j] = complex((double) (1/sqrt((float) N))*cos(2*PI*i*((float) j/((int) N))),
            (double) (1/sqrt((float) N))*sin(2*PI*i*((float) j/((int) N))))
        end for
    end for
end function
```

Figure 3. Algorithm in C++ syntax-like style showing how the QFT operator is defined.

Due to the inefficiency issues when simulating a quantum system on a classical computer, the *FTSimulator* tries to minimize the possible problems by disposing data structures out of the memory as long as they are not needed. The positive effects of such practice are specially important when the input considered has a considerable number of qubits.

In order to exemplify the execution of the *FTSimulator*, consider that the Bell state $|\beta_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ whose density matrix is shown on Eq. (3) is in the input file.

$$\rho_{\text{input}} = \begin{bmatrix} 0,5 & 0 & 0 & 0,5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0,5 & 0 & 0 & 0,5 \end{bmatrix}. \tag{3}$$

It must be emphasized that this 2-qubit quantum state is in superposition and is also highly entangled (mixed state), showing two unique quantum characteristics. The *FTSimulator* performs the following processing: $\rho_{\text{output}} = \mathrm{QFT}\,\rho_{\text{input}}\,\mathrm{QFT}^\dagger$, in which the QFT operator is described in Sec. 2. As a result, the density matrix ρ_{output}, shown in Eq. (4) is stored in the output file.

$$\rho_{\text{output}} = \begin{bmatrix} 0,5 & 0,25 + 0,25\iota & 0 & 0,25 - 0,25\iota \\ 0,25 - 0,25\iota & 0,25 & 0 & -0,25\iota \\ 0 & 0 & 0 & 0 \\ 0,25 + 0,25\iota & 0,25\iota & 0 & 0,25 \end{bmatrix}. \tag{4}$$

Besides performing the QFT, the *FTSimulator* is also able to perform the *Fast Fourier transform* (FFT) [Cooley and Tukey 1965] on any matricial input. Different from the QFT, the FFT can be applied to any matrix and it specifies a transformation which is applied to numbers instead of quantum states. The FFT is very important to communications, audio and image processing, differential equations, among other applications [Rao et al. 2010]. The FFT output on *FTSimulator* can be automatically converted to LaTeX graphics that can be used in reports, articles, etc. The FFT module and its relation with the other elements of the *FTSimulator* can be view in Figure 2.

4. Analysis of the Simulation of the Quantum Fourier Transform

To analyze the simulation of the QFT performed by the *FTSimulator*, we considered an experimental approach. This approach would be helpful in the determination of the maximum number of qubits of the input able to be simulated and also the time spent by this simulation. To do so, we considered a machine with a processor with 3.2 GHz and with 15 GB of main memory.

To evaluate the *FTSimulator* regarding the QFT, it was necessary to develop two additional modules: (i) a module for generation of density matrices, respecting the properties that they are hermitian, positive and have trace equal to 1; and (ii) a test control module which receives n and r where n is the number of qubits to be tested and r is the number of repetitions necessary to reach certain significance level.

The test control module calls the module for generation of density matrices, uses them as input for the QFT and stores the time that each matrix took to be transformed with the *FTSimulator*. The organization of these modules and the data flow are illustrated in Figure 4.

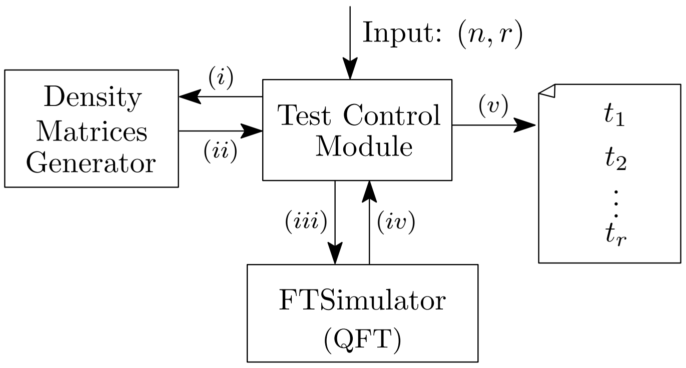
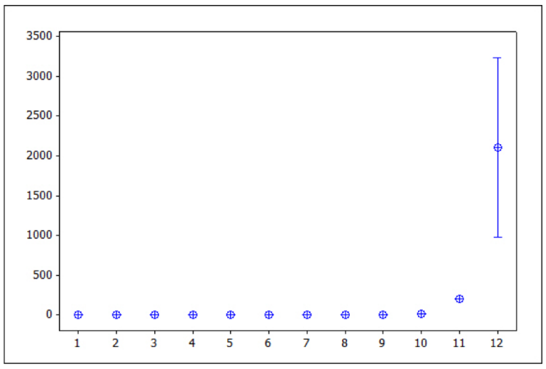


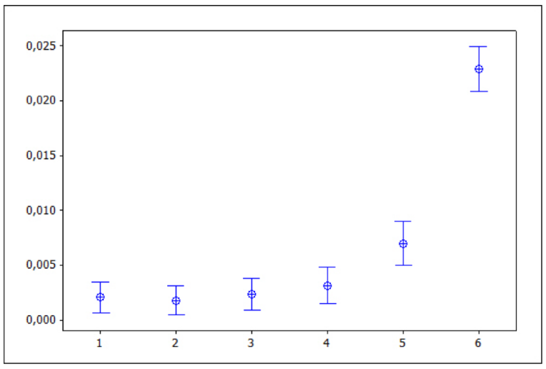
Figure 4. General idea of the evaluation of the QFT in the *FTSimulator*.

The confidence level considered was 95% for the mean and the data obtained for each n did not revealed a normal distribution tendency. The confidence interval plot of the results obtained is shown on Figure 5 and other statistics are shown on Table 1.[1]

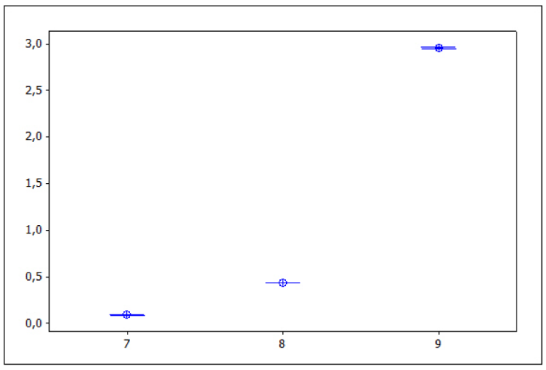
[1]For more information regarding experimental tests and their results meaning, the books of Jain [Jain 1991] and Lilja [Lilja 2004] are recommended.



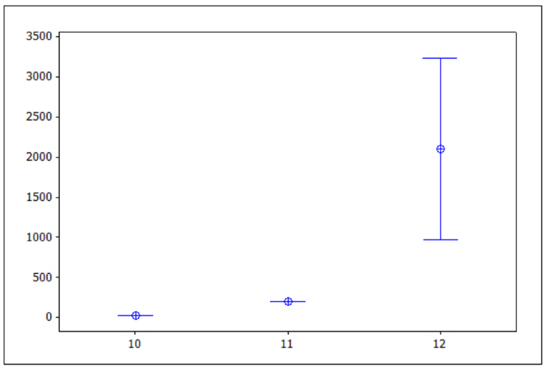
(a)



(b)



(c)



(d)

Figure 5. Interval plots for the mean with confidence level of 95%. The x axis represents the number of qubits and the y axis represents the time in seconds.

Table 1. Mean and standard deviation obtained for the time of simulation per number of qubits.

Qubits	Mean	Standard Deviation
1	0,002083	0,005359
2	0,001817	0,005044
3	0,002333	0,005605
4	0,003167	0,006389
5	0,00700	0,00781
6	0,02290	0,00792
7	0,09078	0,01274
8	0,43967	0,00884
9	2,9572	0,0358
10	22,540	0,178
11	200,52	0,774
12	2107	709

The most of confidence intervals are small as can be confirmed by the respective standard deviation. A narrow confidence interval shows an accurate result for the simulation time for the respective number of qubits. As the number of qubits increase, a exponential-like growth can also be observed. This is an expected consequence since the simulation of quantum systems by classical computers is not efficient.

The results for 12 qubits, however, were expected to be a little better, i.e., the simulation of the QFT for 12-qubit input would be desired to require less time. We strongly believe that it happened due to operating system issues where paging was required and data from main memory was moved to a lower access secondary memory.

5. Conclusion

In this paper we presented a simulation of the quantum Fourier transform algorithm. This simulation was performed in a classical computer due to the non-existence of a scalable implementation of a quantum computer. However, as shown previously, this kind of simulation is not efficient and issues like input size, memory management, data structure management, among others had to be considered.

Aiming at performing the proposed simulation, a software called *FTSimulator* was build. This software is able to performed the QFT on up to 12 qubits. The time required by these simulations was analyzed and narrow confidence intervals were verified, showing that the *FTSimulator* has a very similar performance on input of same size. In this analysis, the 12 qubit input revealed a high standard deviation when compared to the other number of qubits. Although we strongly believe that it is due to an operating system behavior, we are interested in performing more tests on such input size considering operating system variables in order to verify this hypothesis.

The *FTSimulator* is a multi-platform open-source application under the GNU Public License 3 which code is available on `http://ftsimulator.googlecode.com`. Besides the QFT, the *FTSimulator* is also able to perform the FFT, generating LaTeX graphics for the output.

In future work we aim at improving the number of qubits able to be simulated with the *FTSimulator* by using more complex data structures (considering the existence of sparse matrices, for instance) and other optimization techniques for quantum circuits simulation [Viamontes et al. 2009]. These improvements would be helpful to use the *FTSimulator* as part of a simulation of the quantum factorizing algorithm. Moreover, we would like to develop didactic resources to help Computer Science and Engineering students to use *FTSimulator* as a tool to support the learning of the Quantum Computing paradigm.

Acknowledgements

The authors acknowledge the financial support rendered by the CAPES, CNPQ, UFCG and by the project QUANTA/RENASIC/FINEP. The authors are thankful for the help and ideas provided by the Professors Aércio de Lima and Bruno Albert.

References

Bacon, D. and van Dam, W. (2010). Recent Progress in Quantum Algorithms. *Commun. ACM*, 53(2):84–93.

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301.

Deutsch, D. (1985). Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer. *Proceedings of the Royal Society of London A*, 400:97–117.

Deutsch, D. (1989). Quantum computational networks. *Proc. R. Soc. London A*, 425:73–90.

Feynman, R. (1982). Simulating Physics with Computers. *International Journal of Theoretical Physics*, 21:467–488.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

Grover, L. K. (1997). Quantum Mechanics Helps in Searching for a Needle in a Haystack. *Physical Review Letters*, 79:325–328.

Grune, D., van Reeuwijk, K., Bal, H. E., Jacobs, C. J., and Langendoen, K. (2012). *Modern Compiler Design*. Springer.

Guedes, E. B., de Assis, F. M., and Lula Jr., B. (2013). Quantum attacks on pseudorandom generators. *Mathematical Structures in Computer Science*, 23:1–27.

Imre, S. and Gyongyosi, L. (2012). *Advanced Quantum Communications: An Engineering Approach*. Wiley-IEEE Press.

Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. John Wiley.

Ladd, T. D., Jelezko, F., Laflamme, R., Nakamura, Y., Monroe, C., and O'Brien, J. L. (2010). Quantum computers. *Nature*, 464:45–53.

Lilja, D. (2004). *Measuring Computer Performance*. Cambridge University Press.

Nielsen, M. A. and Chuang, I. L. (2010). *Quantum Computation and Quantum Information*. Bookman.

Paar, C. and Pelzl, J. (2010). *Understanding Cryptography*. Springer.

Rao, K., Kim, D., and Hwang, J. (2010). *Fast Fourier Transform: Algorithms and Applications*. Springer.

Santos, G. O., de Assis, F. M., and de Lima, A. F. (2013). Explicit error syndrome calculation for quantum graph codes. *Quantum Information Processing*, 12(2):1269–1285.

Shor, P. (1997). Polynomial-time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26:1484–1509.

Viamontes, G. F., Markov, I. L., and Hayes, J. P. (2009). *Quantum Circuit Simulation*. Springer.

Williams, C. P. (2011). *Explorations in Quantum Computing*. Springer.