Technical Report No. 2006–507 Coping with Decoherence: Parallelizing the Quantum Fourier Transform

Marius Nagy and Selim G. Akl
School of Computing
Queen's University
Kingston, Ontario K7L 3N6
Canada
Email: {marius,akl}@cs.queensu.ca

March 2006

Abstract

Rank-varying computational complexity describes those computations in which the complexity of executing each step is not a constant, but evolves throughout the computation as a function of the order of execution of each step [2]. This paper identifies a practical instance of this computational paradigm in the procedure for inverting the quantum Fourier transform. It is shown herein that under the constraints imposed by quantum decoherence, only a parallel approach can guarantee a reliable solution or, alternatively, improve scalability.

1 Introduction

The computations carried out today are qualitatively different from those performed more than half a century ago, when the age of computers was only just beginning. The traditional concept of computation is best captured by the functioning of the Turing machine. A sequence of operations (or transformations) forming the *algorithm* is applied to a set of *input* data to produce an *output* (or result).

In time, this rather simplistic view on computation has been challenged by increasingly demanding applications and real-world problems. We need better solutions, faster, to problems whose input specifications may vary with time. Often, our results need to be obtained before certain deadlines, or else various penalties can be applied. If a deadline is set by human, there may still be some value in a result that misses it, but sometimes Nature itself imposes strict deadlines on our computations and any algorithm running past the specified time can no longer produce a valid or correct solution. Parallel processing may be the only viable alternative in such situations.

Computational environments have been identified in which a task can be efficiently executed in parallel, but impossible to carry out sequentially [1]. As a concrete example, a parallel approach makes the difference between success and failure when trying to measure (or set) the parameters of a dynamical system [3].

The quest to make computation more efficient and meet today's requirements has led to the development of entirely new computational paradigms. These are based on revolutionary principles like the laws of quantum mechanics, the Watson-Crick complementarity of the building blocks forming DNA strands, the dynamics of complex chemical reactions or the structure of a living cell. And it seems that parallelism plays an important role also in these novel ways of performing a computation. We demonstrated in [10] the importance of parallelism for quantum measurements. In this paper we focus on quantum computation and present an instance of a rank-varying complexity algorithm which can only be reliably implemented in a parallel setting. The example we describe involves computing the inverse quantum Fourier transform, under the constraints imposed by avoiding the undesired effects caused by coupling with the environment.

The remainder of the paper is structured as follows. The next section is intended to familiarize the reader with the quantum circuit performing the discrete Fourier transform. In section 3, computing the quantum Fourier transform and its inverse are shown to belong to the class of computations with rank-varying complexity. A way to speed up the application of the inverse quantum Fourier transform through parallel processing is also described in the same section. Section 4 presents the same problem from the practical perspective of avoiding the undesired effects of quantum decoherence and demonstrates the importance of the parallel approach in the given context.

This is followed in section 5 by a discussion on the key issues affecting the parallelization of a computation with steps of varying complexity. The paper concludes with a short section summarizing its main ideas and contributions.

2 Quantum Fourier Transform

The theory of quantum computation is already well-developed and a great deal of effort is put nowadays into filling the gap between theory and practical implementations of quantum computing devices. Quantum computation harnesses the quantum mechanical principles of superposition and interference in order to achieve a potential exponential speed-up over classical algorithms. For a grasp of the basic concepts in quantum computation we refer the unfamiliar reader to what we consider a few good introductions to the field [11, 8, 12, 4, 7].

The Fourier transform is a very useful tool in computer science and it proved of crucial importance for quantum computation as well. Since it can be computed much faster on a quantum computer than on a classical one, the discrete Fourier transform allows for the construction of a whole class of fast quantum algorithms. Shor's quantum algorithms for factoring integers and computing discrete logarithms [13] are the most famous examples in this category.

The quantum Fourier transform is a linear operator whose action on any of the computational basis vectors $|0\rangle, |1\rangle, \cdots, |2^n-1\rangle$ associated with an n-qubit register is described by the following transformation:

$$|j\rangle \longrightarrow \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi i jk/2^n} |k\rangle,\ 0 \leq j \leq 2^n - 1. \qquad (1)$$

However, the essential advantage of quantum computation over classical computation is that the quantum mechanical principle of superposition of states allows all possible inputs to be processed at the same time. Consequently, if the quantum register is in an arbitrary superposition of the basis vectors

$$\sum_{j=0}^{2^n-1} x_j |j\rangle,$$

then the quantum Fourier transform will rotate this state into another superposition of the basis vectors

$$\sum_{k=0}^{2^n-1} y_k|k\rangle,$$

in which the output amplitudes y_k are the classical discrete Fourier transform of the input amplitudes x_j. Classically, we can compute the numbers y_k from x_j using $\Theta(2^{2n})$ elementary arithmetic operations in a straightforward manner and in $\Theta(n2^n)$ operations by using the Fast Fourier Transform algorithm.

In contrast, a circuit implementing the quantum Fourier transform requires only $\Theta(n^2)$ elementary quantum gates. Such a circuit can be easily derived if equation 1 is rewritten as a tensor product of the n qubits involved:

$$|j_1 j_2 \cdots j_n\rangle \longrightarrow \frac{(|0\rangle + e^{2\pi i 0.j_n}|1\rangle) \otimes (|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2\pi i 0.j_1 j_2 \cdots j_n}|1\rangle)}{2^{n/2}}. \tag{2}$$

using the binary representation $j_1 j_2 \cdots j_n$ of j and binary fractions in the exponents (for full details see [11]).

Note that each Fourier transformed qubit is in a balanced superposition of $|0\rangle$ and $|1\rangle$. They differ from one another only in the relative phase between the $|0\rangle$ and the $|1\rangle$ component. For the first qubit in the tensor product, j_n will introduce a phase shift of 0 or π, depending on whether its value is 0 or 1, respectively. The phase of the second qubit is determined (controlled) by both j_n and j_{n-1}. It can amount to $\pi + \pi/2$, provided j_{n-1} and j_n are both 1. This dependency on the values of all the previous qubits continues up to (and including) the last term in the tensor product. When $|j_1\rangle$ gets Fourier transformed, the coefficient of $|1\rangle$ in the superposition involves all the digits in the binary expansion of j.

In the case of each qubit, the 0 or π phase induced by its own binary value is implemented through a Hadamard gate. The dependency on the previous qubits is reflected in the use of controlled phase shifts, as depicted in Figure 1. In the figure, H denotes the Hadamard transformation

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

while the gate R_k implements a $\pi/2^{k-1}$ phase shift of the $|1\rangle$ component, according to the unitary transformation

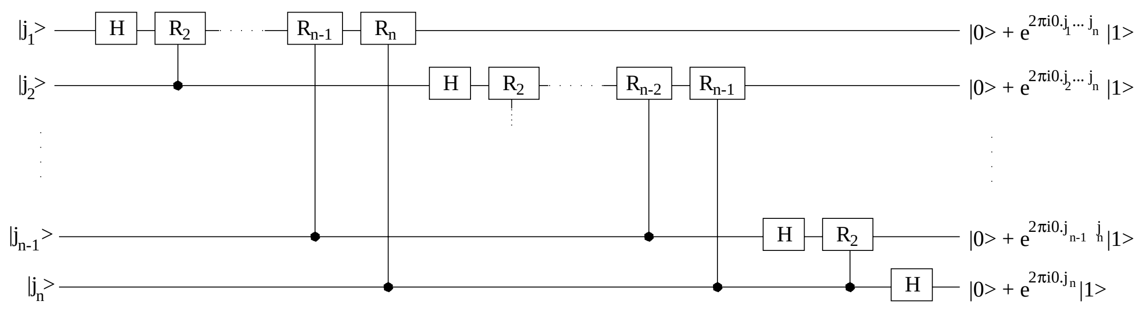
|j1>
|j2>
|jn-1>
|jn>
H
R2
Rn-1
Rn
Rn-2
|0> + e^{2πi0.j1... jn} |1>
|0> + e^{2πi0.j2... jn} |1>
|0> + e^{2πi0.jn-1 jn} |1>
|0> + e^{2πi0.jn} |1>

Figure 1: Quantum circuit performing the discrete Fourier transform.

$$R_k \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}.$$

3 Rank-varying complexity

When analyzing the computational complexity of a given algorithm, we usually focus on how this quantity varies as a function of the problem size, without paying too much attention to how the complexity of each step in the algorithm varies throughout the computation. Though in many cases the complexity of each step is a constant, there are computations for which the cost of executing each step is different from one step to another.

One factor determining such a variation could be *time*. Data may be affected by the passage of time, making the same computation increasingly harder as time goes by. A variety of instances demonstrating time-varying complexity are described in [2]. In other computational environments, it is rather the *rank* of a step, defined as the order of execution of that step, which dictates its complexity [2]. Examples of this kind are hardly new. Euclid's algorithm for computing the greatest common divisor of two numbers executes the same basic operation (a division) at each step, but the size of the operands (and implicitly the complexity of the operation) decreases continually. Algorithms for which an amortized analysis can be applied also make good examples of rank-varying computational complexity. Incrementing a binary counter [6] is a procedure in which the number of bit flips at each step is not constant, though it's neither strictly increasing nor strictly decreasing with the rank.

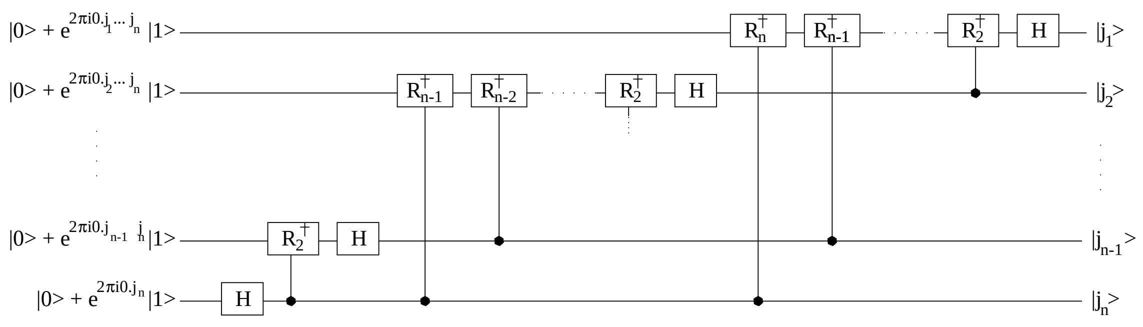
|0> + e^{2πi0.j_1...j_n} |1>
|0> + e^{2πi0.j_2...j_n} |1>
|0> + e^{2πi0.j_{n-1} j_n} |1>
|0> + e^{2πi0.j_n} |1>
H
R_2^†
R_{n-1}^†
R_{n-2}^†
R_n^†
|j_1>
|j_2>
|j_{n-1}>
|j_n>

Figure 2: Quantum circuit performing the inverse Fourier transform.

Of particular interest to us are those instances where the computational requirements grow with the rank. Computing the quantum Fourier transform and especially its inverse are such examples. According to the quantum circuit above, we need n Hadamard gates and $n-1+n-2+\cdots+1$ conditional rotations for a total of $n(n+1)/2$ gates required to compute the Fourier transform on n qubits. But this total amount of work is not evenly distributed over the n qubits. The number of gates a qubit needs to be passed through is in inverse relation with its *rank*. $|j_1\rangle$ is subjected to n elementary quantum gates, $n-1$ elementary unitary transformations are applied to $|j_2\rangle$, and so on, until $|j_n\rangle$, which needs only one basic operation.

If we break down the quantum Fourier transform algorithm into n steps (one for each qubit involved), then its complexity varies with each step. Depending on whether we start with $|j_1\rangle$ or $|j_n\rangle$, the time needed to complete each step decreases or increases, respectively, over time. Since the rank of each step dictates its complexity, the circuit implementing the quantum Fourier transform is an example of a rank-varying complexity algorithm.

Naturally, the computation of the inverse quantum Fourier transform can also be decomposed into steps of varying complexity. Reversing each gate in Figure 1 gives us an efficient quantum circuit (depicted in Figure 2) for performing the inverse Fourier transform. Note that the Hadamard gate is its own inverse and $R_k^\dagger$ denotes the conjugate transpose of R_k:

$$R_k^\dagger \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{-2\pi i/2^k} \end{bmatrix}.$$

Getting back to the original $|j_1 j_2 \cdots j_n\rangle$ from its Fourier transformed expression has a certain particularity though. Because of the interdependencies introduced by the controlled rotations, the procedure must start by computing $|j_n\rangle$ and then work its way up to $|j_1\rangle$. The value of $|j_n\rangle$ is needed in the computation of $|j_{n-1}\rangle$. Both $|j_n\rangle$ and $|j_{n-1}\rangle$ are required in order to obtain $|j_{n-2}\rangle$. Finally, the value of all the higher rank bits are used to determine $|j_1\rangle$ precisely. Thus, computing the inverse Fourier transform by the quantum circuit above is an increasing complexity procedure.

As in the case of the circuit in Figure 1, a purely sequential approach requires $n(n+1)/2$ time units (in the worst case) to complete the work, assuming each quantum gate completes its unitary evolution in one time unit. However, there is a certain degree of parallelism in these two algorithms that can be exploited. Referring to the circuit for performing the inverse Fourier transform, we observe that as soon as we know j_n (at the end of the first step), we can apply all the rotations controlled by the value of j_n, in parallel, in the second step. With the value of j_{n-1} obtained at the end of the second step, we can perform all $n-2$ j_{n-1}-controlled rotations in step 3. The number of operations performed in parallel, in each step, decreases steadily until there is only one controlled rotation left for the last step.

Therefore, in terms of the time elapsed, a parallel approach only needs $1+2+2+\cdots+2 = 2n-1$ time units to complete the procedure. A similar analysis (yielding the same results) can also be carried out for the quantum circuit computing the direct Fourier transform. The difference in time complexity between the sequential approach and the parallel one, although seemingly insignificant from a theoretical perspective, may prove essential under practical considerations, as we show next.

4 Quantum decoherence

Qubits are fragile entities and one of the major challenges in building a practical quantum computer is to find a physical realization that would allow us to complete a computation before the quantum states we are working with become seriously affected by quantum errors. In an ideal setting, we evolve our qubits in perfect isolation from the outside world. But any practical implementation of a quantum computation will be affected by the interactions taking place between our system and the environment. These interactions cause quantum information to leak out into the environment, leading to errors in our qubits. Different types of errors may affect an ongoing computation in different ways, but *quantum decoherence*, as defined below, usually occurs extremely rapidly and can seriously interfere with computing the inverse Fourier transform.

Consider the task of recovering the original bit string $j = j_1 j_2 \cdots j_n$ from its quantum Fourier transformed form. The circuit performing this computation (see Figure 2) takes as input n qubits. The state of each qubit can be described by the following general equation:

$$|\psi_k\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{e^{i\theta_k}}{\sqrt{2}}|1\rangle,\ 1 \leq k \leq n \tag{3}$$

where the relative phase θ_k, characterizing the qubit of rank k, depends on the values of bits $j_k, j_{k+1}, \cdots, j_n$. The corresponding density operator is given by

$$\rho_k = |\psi_k\rangle\langle\psi_k| = \frac{1}{2}|0\rangle\langle 0| + \frac{e^{-i\theta_k}}{2}|0\rangle\langle 1| + \frac{e^{i\theta_k}}{2}|1\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1|, \tag{4}$$

or in matrix form

$$\rho_k = \frac{1}{2}\begin{bmatrix} 1 & e^{-i\theta_k} \\ e^{i\theta_k} & 1 \end{bmatrix}. \tag{5}$$

The diagonal elements (or the *populations*) measure the probabilities that the qubit is in state $|0\rangle$ or $|1\rangle$, while the off-diagonal components (the *coherences*) measure the amount of interference between $|0\rangle$ and $|1\rangle$ [5]. Decoherence then, resulting from interactions with the environment, causes the off-diagonal elements to disappear. Since that is where the whole information carried by a qubit is stored, the input qubits for computing the inverse Fourier transform are very sensitive to decoherence. When they become entangled with the environment, the interference brought about by the Hadamard gate is no longer possible, as the system becomes effectively a statistical mixture. In other words, decoherence makes a quantum system behave like a classical one.

Naturally, this process is not instantaneous, but it usually occurs extremely rapidly, subject to how well a qubit can be isolated from its environment in a particular physical realization. Recall that applying the quantum gates depicted in Figure 2 in a strict sequential order takes $(n^2 + n)/2$ steps, while taking advantage of parallelism ensures the completion of the computation in only $2n - 1$ time units. Because of decoherence, we must obtain the values of $j_1, j_2, \cdots, j_n$ before time limit δ, after which the errors introduced by the coupling with the environment are too serious to still allow the recovery of the binary digits of j.

The precise value of δ will certainly depend on the particular way chosen to embody quantum information, but, in general, we can assert the existence of integers n_s and n_p such that

$$2n - 1 < \delta < \frac{n^2 + n}{2}, \;\; \forall n \; n_s < n < n_p. \tag{6}$$

Here, n_s represents the maximum number of qubits for which a sequential solution to the problem of inverting a quantum Fourier transform may still produce accurate results. Similarly, we denote by n_p the size of the problem for which even a parallel approach is no longer able to cope with decoherence effects.

We conclude that within the interval delimited by n_s and n_p, the parallel algorithm is the only one that succeeds, while a sequential algorithm will fail to precisely recover all digits in the binary expansion of j. Therefore, in this case, parallelism may be regarded as a means of improving the scalability of computing the inverse quantum Fourier transform.

5 Discussion

In this paper, we have investigated the benefits of a parallel approach when addressing a computational problem with steps of increasing complexity, in the presence of deadlines. Our focus was on the quantum algorithm used to compute the inverse Fourier transform, when the qubits $|j_1\rangle$, $|j_2\rangle$, $\cdots$, $|j_n\rangle$ have classical values (they are classical bits). This restriction is crucial in allowing the parallelization we have described. No parallel algorithm exists in the general case, when an arbitrary superposition of the basis vectors is Fourier transformed and we seek to reverse that transformation.

The reason for this impossibility is the quantum mechanical nature of the qubits controlling the inverse rotations in Figure 2. Such a controlled rotation corresponds to a two-qubit gate and we need to apply, in parallel, a number of two-qubit gates, where the control qubit is the *same* in all gates. Since we cannot gain knowledge of the control qubit's state through measurement and cloning an unknown quantum bit is forbidden by the laws of quantum mechanics, any attempt to parallelize the procedure in the general case is doomed to failure. However, in the particular case when the value of the control qubit is 0 or 1, we know that either there are no rotations to be reversed, or that all the lower-rank qubits are simultaneously subjected to single-qubit rotations, respectively.

It follows that quantum algorithms will not benefit from parallelizing the quantum Fourier transform and its inverse since they rely heavily on processing inputs in superposition to achieve speed-up over classical algorithms. Nevertheless, there are other areas of quantum information processing, like quantum cryptography, that may still make good use of the quantum Fourier transform and its inverse, even in the simple form we analyzed herein. Thus, we show in [9] how existing quantum key distribution protocols can be enhanced under the assumption that the receiver is able to store the qubits transmitted until the communication through the public channel takes place.

The difficulty of devising a parallel algorithm for computing the inverse Fourier transform comes from the data dependency between the different steps of the procedure. In most cases, it is exactly this precedence among the steps composing an algorithm that determines the variation in complexity. As a consequence, it is not easy, in general, to design a parallel solution to a problem whose steps are characterized by rank-varying complexity. The data dependency may impose a strict order of execution, making the resulting algorithm inherently sequential (think about Euclid's algorithm again).

In the case of the inverse Fourier transform, it is interesting to note that strict sequentiality is enforced by the laws of quantum mechanics, but we still have a chance to speed up the computation, provided we restrict the qubits onto which the quantum Fourier transform is applied to classical bits. At the other end, perhaps there exist computations made up of steps of various complexities, for which the order of execution is of no consequence to the correctness of the computation. A problem belonging to this paradigm would benefit most from a parallel approach.

6 Conclusion

The example presented in this paper underlines the importance of parallel processing for those computational environments in which the complexity of each step evolves with its rank. This also extends to the cases where the variation in complexity is time-driven [2]. The use of a parallel approach becomes critical when the solution to such a problem must accommodate a deadline. In our case, quantum decoherence places an upper bound on the scalability of the quantum circuit in Figure 2 and the only chance to reach beyond that limit is through a parallel solution.

References

[1] Selim G. Akl. Coping with uncertainty and stress: A parallel computation approach. to appear in International Journal of High Performance Computing and Networking.

[2] Selim G. Akl. Evolving computational systems. In Sanguthevar Rajasekaran and John H. Reif, editors, *Parallel Computing: Models, Algorithms, and Applications.* CRC Press, 2006.

[3] Selim G. Akl, Brendan Cordy, and W. Yao. An analysis of the effect of parallelism in the control of dynamical systems. *International Journal of Parallel, Emergent and Distributed Systems*, 20(2):147–168, June 2005.

[4] André Berthiaume. Quantum computation. In Lane A. Hemaspaandra and Alan L. Selman, editors, *Complexity Theory Retrospective II*, pages 23–51. Springer-Verlag, New York, 1997.

[5] C. Cohen-Tannoudji, B. Diu, and F. Laloe. *Quantum Mechanics*, volume 1 and 2. Wiley, New York, 1977.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts, 2001.

[7] Mika Hirvensalo. *Quantum Computing.* Springer-Verlag, 2001.

[8] Samuel J. Lomonaco Jr., editor. *Quantum Computation: A Grand Mathematical Challenge for the Twenty-First Century and the Millennium*, volume 58 of *Proceedings of Symposia in Applied Mathematics.* American Mathematical Society, Short Course, Washington, DC, January 17-18 2000.

[9] Marius Nagy and Selim G. Akl. manuscript. 2006.

[10] Marius Nagy and Selim G. Akl. Quantum measurements and universal computation. *International Journal of Unconventional Computing*, 2(1):73–88, 2006.

[11] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[12] Eleanor Rieffel and Wolfgang Polak. An introduction to quantum computing for non-physicists. *ACM Computing Surveys*, 32(3):300–335, September 2000.

[13] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *Special issue on Quantum Computation of the SIAM Journal on Computing*, 26(5):1484–1509, October 1997.