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

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 practical instances of this computational paradigm in the procedures for computing the quantum Fourier transform and its inverse. 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 [11] 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 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. The semiclassical solution advanced by Griffiths and Niu [7] to reduce the complexity of a circuit for computing the quantum Fourier transform is reviewed in section 4. A quantum pipeline array architecture is devised in section 5 in order to further speed up the computation of the quantum Fourier transform or its inverse through parallel processing. Section 6 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. The last section offers a short discussion on the key issues affecting the parallelization of a computation with steps of varying complexity, followed by a summary of the paper's 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 [12, 9, 13, 4, 8].

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 [14] 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:

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

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 [12]).

Note that each Fourier transformed qubit is in a balanced superposition of $|0\rangle$ and $|1\rangle$. These qubits differ from one another only in the relative phase between the $|0\rangle$ and the $|1\rangle$ components. 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

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

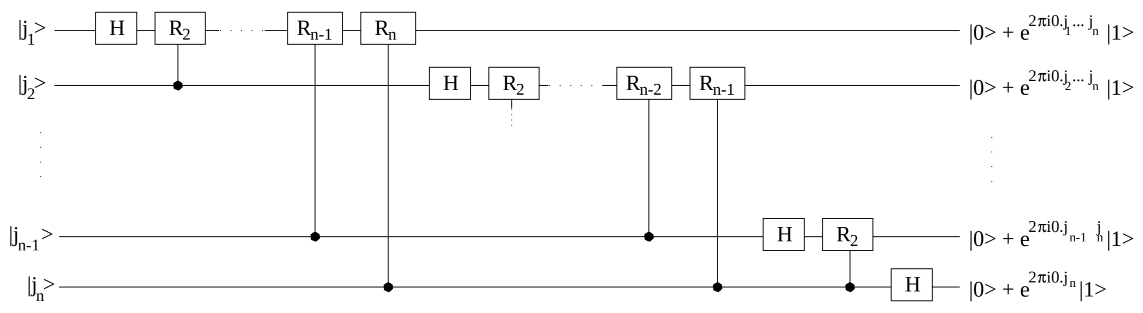
|j1>
|j2>
|jn-1>
|jn>
H
R2
Rn-1
Rn
Rn-2
|0> + e^{2πi0.j1...jn} |1>
|0> + e^{2πi0.j2...jn} |1>
|0> + e^{2πi0.jn-1 jn} |1>
|0> + e^{2πi0.jn} |1>

Figure 1: Quantum circuit performing the discrete Fourier transform.

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

$$R_k \equiv \left[\begin{array}{cc} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{array} \right].$$

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 essentially similar steps 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

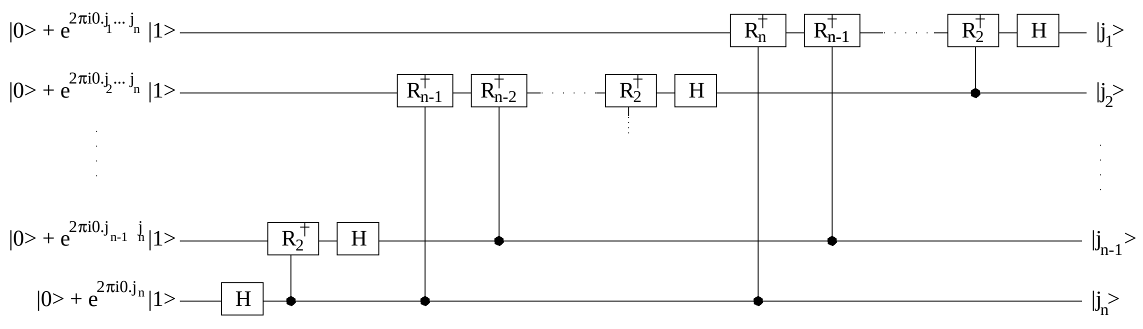
|0> + e^{2πi0.j_1...j_n} |1>
|0> + e^{2πi0.j_2...j_n} |1>
|0> + e^{2πi0.j_{n-1} j_n} |1>
|0> + e^{2πi0.j_n} |1>
H
R_2^†
R_{n-1}^†
R_{n-2}^†
R_n^†
|j_1>
|j_2>
|j_{n-1}>
|j_n>

Figure 2: Quantum circuit performing the inverse Fourier transform.

is not constant, though it's neither strictly increasing nor strictly decreasing with the rank.

Computing the quantum Fourier transform and its inverse are also examples of algorithms with rank-varying complexity. 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. Starting with $|j_1\rangle$, the time needed to complete each step decreases 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 however. Because of the interdepen-

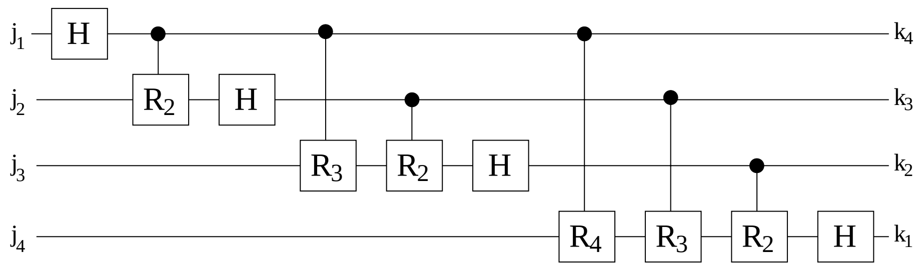

Figure 3: Alternative arrangement of gates in the circuit performing the quantum Fourier transform.

dencies 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 illustrated in Figure 2 is a procedure the complexity of whose steps increases with their rank.

4 Semiclassical solution

Although the circuits for computing the quantum Fourier transform and its inverse are efficient in terms of the total number of gates employed, the majority of these gates operate on two qubits. This makes a practical implementation difficult, since arranging for one qubit to influence another in a desired way is far greater a challenge than evolving a single-qubit closed quantum system in accordance with any unitary transformation.

A method to replace all the two-qubit gates in the circuit performing the quantum Fourier transform by a smaller number of one-qubit gates controlled by classical signals has been developed by Griffiths and Niu [7]. Their approach takes advantage of the fact that the roles of the control and target qubits in any of the two-qubit gates required to carry on the computation of the quantum Fourier transform are interchangeable. Consequently, the quantum circuit in Figure 1 is equivalent to the one depicted in Figure 3 (for inputs restricted to four qubits).

Note that, from this new perspective, the computation of the quantum Fourier transform appears to be a procedure whose steps are of increasing

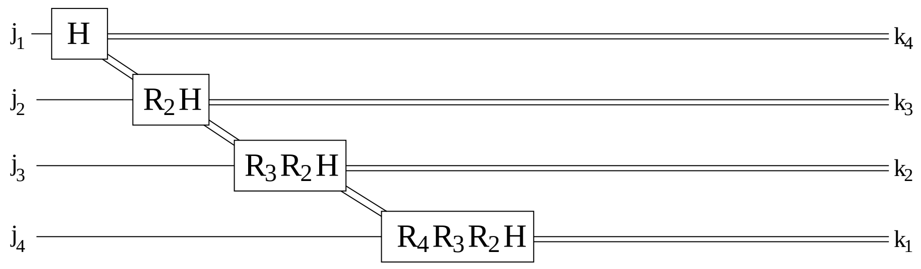

Figure 4: Semiclassical circuit for computing the quantum Fourier transform.

complexity. However, under the assumption that the Fourier transform is immediately followed by a quantum measurement, the complexity of each step in the computation can be made constant. Since a control qubit enters and leaves a two-qubit gate unchanged, it follows that the top qubit in Figure 3 yields the same result regardless of whether it is measured as it exits the circuit or immediately after undergoing the Hadamard transform. In the latter case, the result of the measurement can be used to determine the phase shift that needs to be applied on the second qubit, before it too is subjected to a Hadamard transform and then measured. The phase computed for the second qubit together with the result of the second measurement are passed down as classical inputs for the rotation applied to the third qubit.

The computation proceeds in this manner all the way down to the last qubit, with a phase rotation, a Hadamard gate and a measurement being performed at each step. The process is illustrated in Figure 4, where double lines have been used to denote a classical signal, according to the usual convention. Although the phase shift applied to each qubit is considered a single operation, conceptually, it is a combination of the gates depicted in the corresponding box, with each component being applied only if the controlling qubit was measured as 1.

This semiclassical approach to computing the quantum Fourier transform achieves optimality in terms of the number of elementary unitary transformations that have to be applied. It also has the important advantage of employing only quantum transformations acting on a single qubit at a time. However, there is still room for improvement, as the total time needed to complete the computation can be further squeezed down if parallelism is brought into play. In the next section we show how a quantum pipeline architecture is able to speed up the computation of the Fourier transform.

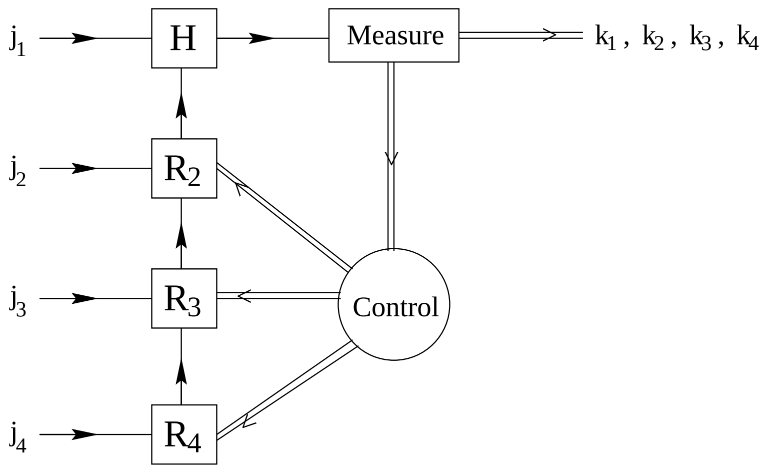

Figure 5: Quantum pipeline array for computing the Fourier transform.

5 Parallel approach

The solution developed in [7] to reduce the complexity of the quantum Fourier transform envisages a purely sequential approach, which is motivated by the same data dependency that causes the complexity of a step to vary with its rank. Nevertheless, there is a certain degree of parallelism that is worth exploiting in the computation of the quantum Fourier transform (or its inverse) in order to minimize the overall running time.

Our parallel approach is based on the observation that once a qubit has been measured, all phase shift gates classically controlled by the outcome of that measurement can be applied in parallel. The arrangement, again for just four qubits, is shown in Figure 5. The one-qubit gates are ordered into a linear array having a Hadamard transform at the top and followed by a $\pi/2$ phase shift gate. The phase shift induced by any other gate down the array is just half the rotation performed by the immediately preceding gate.

This architecture allows R_2, R_3 and R_4 to be performed in parallel during the first cycle. Since each phase shift gate acts on a different qubit, they can all be applied simultaneously, if the top qubit yielded a 1 upon measurement. In the second cycle, each qubit in the array travels up one position, except of course for the top one, which has already been measured. Now, depending on the outcome of the second measurement, R_2 and R_3 can be simultaneously effected on the corresponding qubits. In the third cycle, only R_2 is needed and only if the control is 1. The computation ends with the last qubit reaching the Hadamard gate and being measured afterwards. A formal description of the procedure, in the general case, is given below.

```
Procedure $Parallel\_Quantum\_Fourier\_Transform$
Input: $|j_1 j_2 \cdots j_n\rangle$
Output: $k_1 k_2 \cdots k_n$

for $i = 1$ to $n$ do
    $|j_i\rangle \longleftarrow H|j_i\rangle$;
    Measure $|j_i\rangle$ as $k_{n-i+1}$;
    if $k_{n-i+1} = 1$ then
        for $l = 2$ to $n - i + 1$ do in parallel
            $|j_{i+l-1}\rangle \longleftarrow R_l|j_{i+l-1}\rangle$;
            $|j_{i+l-1}\rangle$ moves one position up in the array
        endfor
    endif
endfor
```

In the worst case, when all qubits are measured as 1, there is no difference between the parallel algorithm outlined above and the sequential solution envisaged by Griffiths and Niu [7] with respect to the overall running time. Assuming, for analysis purposes, that measuring a qubit, applying a phase shift, and performing a Hadamard transformation, each takes one time unit, then the total time necessary to complete the Fourier transform on a quantum register with n qubits is $3n-1$, as the top qubit in both the sequential circuit of Figure 4 and the parallel circuit of Figure 5 does not require a phase shift.

However, in the average case, some of the classical signals controlling the array of phase shift gates in Figure 5 will have been observed as 0, meaning that no phase shifts have to be performed during those respective cycles. In contrast, the sequential solution depicted in Figure 4 requires the application of a phase shift at every step following the first measurement with outcome 1. If the expected probability of a measurement yielding 0 equals the expected probability to observe a 1 following a measurement, then the running time of the parallel solution is shorter than the sequential running time by a difference proportional to the time it takes to effect a number of $O(n)$ phase shift gates, where n is the size of the input register.

The difference between the sequential running time and the parallel running time is maximum when $|j_1\rangle$ is measured as 1 and all the other qubits are observed in the state 0. In this case, the circuit in Figure 4 still performs $n-1$ phase shifts, for a total running time of $3n-1$ time units, while the circuit in Figure 5 executes all $n-1$ phase shifts in parallel during the first cycle, thus completing the computation in $2n+1$ time units.

The second advantage of the parallel approach is that the phase shift gates that need to be applied during the computation are known at the outset, making it easy to set them up beforehand in order to form the required linear array architecture. The systolic mode of operation of the quantum array compensates for the fixed characteristics of each gate, the qubits traversing the array to undergo a specific quantum evolution at each node. In the sequential approach, the phase shift applied to each qubit is not known at the outset, as it is computed on the fly based on the information about the measurements performed so far and transmitted as classical signals. This means that the gates effecting the necessary phase shifts in the semiclassical approach of Griffiths and Niu [7] have to be "programmed" or adjusted during the computation, in order to accommodate a discrete set of possible values for the phase shift.

The semiclassical Fourier transform and its parallelization are applicable to those quantum computations in which the Fourier transform immediately precedes a measurement of the qubits involved in the computation, like in Shor's algorithms for factoring integers and computing discrete logarithms [14]. Furthermore, the quantum systolic array architecture works equally fine if the input is already classical, in which case the restriction to measure the qubits after applying the Fourier transform can be lifted altogether.

When $j_1, j_2, \cdots, j_n$ are classical bits, the topology of the circuit in Figure 5 remains unchanged, except that no measurements are performed and the flow of data through the linear array is reversed, as shown in Figure 6. As more data are fed into the linear array through the Hadamard gate, after having "controlled" the parallel execution of a set of phase shifts, the computational complexity of each step increases with its rank. When j_1 enters the array, only the Hadamard gate is active, but with each consecutive step, a new gate down the array joins the ones above it to operate on the qubits traversing

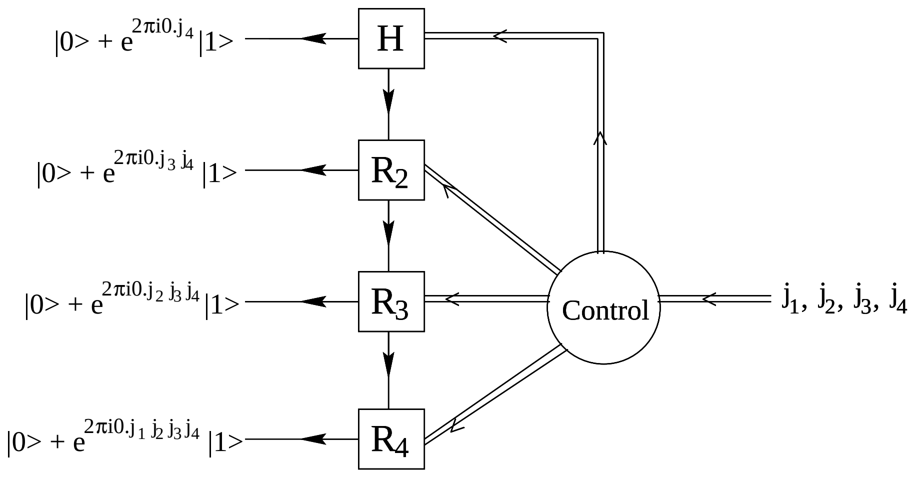

Figure 6: Quantum pipeline array for computing the Fourier transform on classical inputs.

the array. Because these gates operate in parallel, the execution time of each step is maintained constant. Also note that, in this case, all outputs are simultaneously obtained during the last step of the computation.

The overall parallel running time, in the worst case, is therefore $2n - 1$ time units, as there are no measurements to perform. The complexity of the sequential approach is given by the number of gates composing the circuit in Figure 1, that is $n(n + 1)/2$ time units, since the non-classical, genuine quantum mechanical ouput makes the semiclassical method not applicable.

Although applying the quantum Fourier transform on a classical input is of little value for quantum computing, the situation is different for quantum cryptography. Distributing classical keys through quantum means is a procedure that may use the quantum Fourier transform and its inverse as encoding and decoding algorithms to protect vital information while in transit [10].

Naturally, the parallel approach detailed in this section for the computation of the direct Fourier transform is also applicable, with the same results, to the circuit in Figure 2, performing the inverse 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.

6 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 quantum Fourier transform and its inverse.

In the context of a quantum key distribution protocol [10], consider the task of recovering the original (classical) 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. 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 if δ lies between the parallel completion time and the sequential completion time, that is

$$2n - 1 < \delta < \frac{n(n+1)}{2}, \tag{6}$$

then the quantum pipeline array may be the only architecture capable to precisely recover all digits in the binary expansion of j. From a different perspective, the parallel solution allows for longer bit strings to be transmitted between the communicating parties, thus achieving better scalability over the purely sequential approach.

In the context of quantum computing, when the Fourier transform is followed by a quantum measurement, Griffiths and Niu [7] also point to decoherence as a possible problem and suggest to counter it by arranging the computation in such a way that the more significant bits of the input register are produced earlier than the less significant ones. If this is not possible, then the parallel approach described in the previous section may make the difference between success and failure when computing the quantum Fourier transform in a practical setting. Alternatively, since the overall running time scales up with the number of input qubits, parallelism may be a way to improve scalability and still complete the computation before decoherence effects take hold. In this context, we emphasize that scalability and decoherence are the two most important issues in designing a practical quantum computer.

7 Conclusion

The difficulty of devising a parallel algorithm for computing the quantum 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 quantum 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 either the input or the output to be classical. No parallel algorithm exists in the general case, when an arbitrary superposition of the basis vectors is Fourier transformed and we are not allowed to measure the output. The reason for this impossibility is the quantum mechanical nature of the qubits controlling the phase shifts in Figures 1 and 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.

On the other hand, perhaps there exist computations made up of steps of various rank-dependent complexities, for which the order of execution is of no consequence to the correctness of the computation. Imagine, for instance, a task made up of n steps: $S_1, S_2, \ldots, S_n$, where the steps can be executed in any order, but the more steps we execute before a certain step S_j $(1 \leq j \leq n)$, the more time it will take to complete S_j. For example, S_j may require i elementary operations (time units) if executed i^{th}, for $i = 1, 2, \ldots, n$. This rank-driven increase in complexity may be due to how many pieces of data have to be taken into consideration at each consecutive step, how the data were affected by executing the previous steps, or it may be justified by the size of the partial solution that has to be constructed at each step.

In any case, the problem of coping with steps of ever increasing complexity is avoided altogether by a parallel machine endowed with n processing units. All steps would then be executed simultaneously, and since each step has the rank 1 in such a parallel approach, the computational complexity is kept constant (one time unit, in our example) for all steps. The difference between a sequential and a parallel approach is even more dramatic if the complexity of a step grows faster with its rank. In the case where step S_j $(1 \leq j \leq n)$ needs 2^i elementary operations (time units) to be completed, if executed i^{th}, $i = 1, 2, \ldots, n$, the benefits of using a parallel approach are much higher.

The practical examples presented in this paper underline 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 computing the quantum Fourier transform or its inverse, 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. *International Journal of High Performance Computing and Networking*, 4(1/2):85–90, 2006.

[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] Robert Griffiths and Chi-Sheng Niu. Semiclassical Fourier transform for quantum computation. *Physical Review Letters*, 76:3228–3231, 1996.

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

[9] 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.

[10] Marius Nagy and Selim G. Akl. Quantum key distribution revisited. Technical Report 2006-516, School of Computing, Queen's University, Kingston, Ontario, June 2006.

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

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

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

[14] 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.