Parallel Quantum Computer Simulation on the CUDA Architecture

Eladio Gutierrez, Sergio Romero, Maria A. Trenas, and Emilio L. Zapata

Department of Computer Architecture, University of Malaga,
29071 Malaga, Spain

{eladio,sromero,maria,ezapata}@ac.uma.es

Abstract. Due to their increasing computational power, modern graphics processing architectures are becoming more and more popular for general purpose applications with high performance demands. This is the case of quantum computer simulation, a problem with high computational requirements both in memory and processing power. When dealing with such simulations, multiprocessor architectures are an almost obliged tool. In this paper we explore the use of the new graphics processor architecture NVIDIA CUDA in the simulation of some basic quantum computing operations. This new architecture is oriented towards a more general exploitation of the graphics platform, allowing to use it as a parallel SIMD multiprocessor. In this direction, some implementation strategies are proposed, showing that the effectiveness of the codes is subject to a right exploitation of the underlying memory hierarchy.

1 Introduction

Contrary to classical computers, quantum computers are devices that process information on the basis of the laws of quantum physics. Due to this fact, they could provide an efficient implementation of different algorithms as refers both to computing time and storage requirements. This way, they would be able to solve some problems of non-polynomial complexity in a much smaller time [10].

Although this approach is quite promising, at the present time it is still necessary to face certain limitations. On the one hand, existing technology allows only to construct quantum computers of very reduced dimensions [7], and on the other hand, only a small number of effective algorithms [5,8,13] are known. Nevertheless, the analysis of this computational model constitutes a topic of great interest for physicists, computer scientists and engineers. Actually, a quantum computer can be considered as a hardware accelerator of the classical processor, from which it receives the orders for the resolution of a concrete problem [7], as shown in Fig. 1. As it is not possible to know the inner state of a quantum computer, according to the laws that govern it, outputs must be obtained by means of a measurement, issuing a result with a certain probability.

The quantum parallelism is one of the sources of the power of quantum computers as it allows to perform simultaneous operations on an exponential set of

superposed states. This causes quantum computer simulation to demand high computational power. This way, parallelism is a suitable tool for mitigating such requirements [7,11]. The simulation of quantum computers not only requires a high computational effort but they present data access patterns with low locality characteristics. Several interests arise for this kind of simulation giving raise to the development of different simulators, both in software [2,3,7,11] as in hardware [6,9,14].

In this paper we prove that modern architectures based on Graphic Processor Units (GPU) are suitable to accomplish an efficient simulation of quantum computers. GPUs are devices specialized in graphics algorithms involving very intensive and highly parallel computations that due to their high computational power are nowadays being used also for general purpose applications. With this purpose, we have implemented several parallel approaches to the simulation of the basic operators of an ideal quantum computer, using the new compute unified device architecture (CUDA) [12], lately released by the GPU manufacturer NVIDIA. Different strategies are explored, looking for the exploitation of the data reference locality in this sort of architecture.

2 Quantum Computing

The ideal quantum computer to be simulated follows the model presented in [4], consisting on the successive application of a network of quantum gates to a quantum register with a classical initial state. The quantum bit (qubit) can be imagined as the linear superposition of two homologous classical states we will note as $|0\rangle$, $|1\rangle$, in Dirac notation. The state of a qubit can be represented using a complex two-dimensional vector, where the basis for these two states are $|0\rangle$ and $|1\rangle$. Thus, the state of a qubit can be written as $\Psi = \alpha_0|0\rangle + \alpha_1|1\rangle$, where the coefficients, or amplitudes, verify $|\alpha_0|^2 + |\alpha_1|^2 = 1$. $|\alpha_0|^2$ and $|\alpha_1|^2$ are interpreted as the probability of measuring $|0\rangle$ or $|1\rangle$. In vector notation, we write $\Psi = \begin{pmatrix} \alpha_1 \\ \alpha_0 \end{pmatrix}$, $|0\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$.

A quantum register generalizes the qubit definition. The state of a n -qubit quantum register is determined by the linear superposition of the 2^n possible classical states provided by n bits. After this, the state of a quantum register can be written as $\Psi = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle$ with $\alpha_i \in \mathbb{C}$, $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$, since $|\alpha_i|^2$ is interpreted as the probability of obtaining $|i\rangle$ when the register is measured. Thus, Ψ belongs to a 2^n -dimensional complex vector space, where $|i\rangle$ constitutes a basis, with $0 \leq i \leq 2^n - 1$. For example, for $n = 3$, we will write $|6\rangle = |110\rangle = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0)^T$. By applying the Kronecker's tensor product, it is possible to represent the elements of the state space basis for the register as a function of the individual states of the qubits. For example $|6\rangle = |110\rangle = |1\rangle \otimes |1\rangle \otimes |0\rangle$.

The state of a quantum register will evolve according to a transformation, which can be interpreted as an operator U applied to the register state. Quantum physics laws settle that operator U must be a linear and unitary one. It follows that for a n -qubit register, an order $2^n \times 2^n$ matrix can be found verifying

Fig. 1. A quantum computer model

Fig. 2. A QFT implementation

Fig. 3. Representation of quantum transformations as quantum gate networks

$UU^* = I$, where U^* is matrix U both conjugated and transposed, and I is the unitary matrix. Usually, this kind of transformations are represented in the manner of Fig. 3(a). As a particular instance, let us consider the application of a transformation over one particular bit, as shown in Fig. 3(b). In this case, the global transformation will be the tensor product of all the 1-qubit transformations simultaneously applied to each individual qubit. This means that the global resulting transformation U_g will be equivalent to the application of the identity transformation to the residuary bits. If the 1-qubit operator U is applied to the k -th qubit then:

$$U_g = I \otimes (\dots \otimes I \otimes U \otimes I \otimes \dots \otimes I) = I^{\otimes n-k-1} \otimes U \otimes I^{\otimes k} \quad (1)$$

The transformation applied to one single qubit can be interpreted like a unitary quantum gate of order 2×2 . Table 1 presents several well-known transformations. As an example, Pauli transformation $X = |0\rangle\langle 1| + |1\rangle\langle 0|$, does project component $|0\rangle$ over the $|1\rangle$ one, and vice versa, following that its quantum application to a classic state 0 or 1 is equivalent to the logic operator NOT. The generalization to gates with more than one qubit is straightforward, resulting in an associated matrix of order $2^n \times 2^n$, for n qubits.

A quantum computer can be thought to be a quantum device on which a sequence (or network) of transformations can be applied successively to the state of a quantum register [4]. Different minimal universal sets of gates have been proposed, in such a way that any n -qubit transformation can be expressed as a network of these gates. It is proven that a complete set can be built from gates $\Phi(\delta)$, $R_z(\alpha)$, $R_y(\theta)$ and CNOT (2 qubits), as stated in [1].

A remarkable transformation is the Quantum Fourier Transform (QFT). The QFT [10] is a key element of some quantum algorithms like the integer

Table 1. Inventory of quantum computer gates

1-qubit transformations		
Identity	$I = 0\rangle\langle 0 + 1\rangle\langle 1 $
Pauli X	$X = 0\rangle\langle 1 + 1\rangle\langle 0 $
Pauli Y	$Y = j 1\rangle\langle 0 - j 0\rangle\langle 1 $
Pauli Z	$Z = 0\rangle\langle 0 - 1\rangle\langle 1 $
Hadamard	$H = \frac{1}{\sqrt{2}}(X + Z)$
y-axis rotation	$R_y(\theta) = \cos(\frac{\theta}{2})I + \sin(\frac{\theta}{2})Y$
z-axis rotation	$R_z(\alpha) = e^{j\alpha/2} 0\rangle\langle 0 + e^{-j\alpha/2} 1\rangle\langle 1 $
2-qubit transformation		
Controlled NOT	$\text{CNOT} = 0\rangle\langle 0 + 1\rangle\langle 1 + 2\rangle\langle 3 + 3\rangle\langle 2 $
Controlled Phase Shift	$\text{CPh}(K) = 0\rangle\langle 0 + 1\rangle\langle 1 + 2\rangle\langle 2 + e^{j\frac{2\pi}{K}} 3\rangle\langle 3 $

factorization proposed by Shor [13], of great interest in cryptography. An implementation of the QFT is depicted in Fig. 2 in terms of 1-qubit Hadamard gates and 2-qubit controlled-phase gates. QFT is defined analogously to the classical transform but a normalization coefficient $\frac{1}{\sqrt{2^n}}$ makes it unitary:

$$|\psi^{\text{out}}\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \alpha_k^{\text{in}} \sum_{c=0}^{2^n-1} e^{\frac{2\pi ck}{2^n}} |c\rangle \tag{2}$$

3 Elementary Quantum Gate Simulation

Simulation of a quantum computer will consist on determining the state of a n -qubit register, after the application of a unitary linear transformation. This means that we have to compute the register's state vector $|\psi^{\text{out}}\rangle = \sum_{i=0}^{2^n-1} \alpha_i^{\text{out}} |i\rangle$, from initial state $|\psi^{\text{in}}\rangle = \sum_{i=0}^{2^n-1} \alpha_i^{\text{in}} |i\rangle$, that is, to determine coefficients α_i^{out} for the final state as a function of coefficients α_i^{in} for the initial state and the unitary matrix U defining the transformation. In general, the application of this unitary transformation will require computations with a complexity order $\mathcal{O}(2^{2n})$ as matrix U is of order $2^n \times 2^n$, since 2^n is the dimension of the associated vector space. By decomposing this transformation into a set of successive transformations with a lower number of qubits (quantum gates or stages), the effective complexity of the simulation can be reduced. As case of study, the simulation of three basic quantum computing operations is analyzed: a generic gate U over one qubit, the operator $U^{\otimes n}$ applied to an n -qubit register, and the QFT.

Simulation of 1-qubit Gate U. The application of a 1-qubit quantum gate performs the operation $|\psi^{\text{out}}\rangle = U_g |\psi^{\text{in}}\rangle$, where U_g comes from expression (1) as a function of the 1-qubit transformation U . If we consider the initial state is a superposition of states $\psi^{\text{in}} = \sum_{i=0}^{2^n-1} \alpha_i^{\text{in}} |i\rangle$, the effect of the transformation over the coefficients α_i^{in} can be determined.

Consider that the 1-qubit transformation $U = u_{00}|0\rangle\langle 0| + u_{01}|0\rangle\langle 1| + u_{10}|1\rangle\langle 0| + u_{11}|1\rangle\langle 1|$ is applied to the q -th qubit of a n -qubit register. If the initial state is a classical one $\Psi^{\text{in}} = |i\rangle = |b_{n-1}b_{n-2}\dots b_1b_0\rangle$, that is, an element in the basis of the state space, where b_k represent the bits on the binary expression of natural i . Transformation U over the q -th bit b_q will result in:

$$\Psi^{\text{out}} = |b_{n-1}\rangle \otimes \dots \otimes U|b_q\rangle \otimes \dots \otimes |b_0\rangle = \begin{cases} u_{00}|b_{n-1}\dots 0\dots b_1b_0\rangle + u_{10}|b_{n-1}\dots 1\dots b_1b_0\rangle & \text{if } b_q = 0 \\ u_{01}|b_{n-1}\dots 0\dots b_1b_0\rangle + u_{11}|b_{n-1}\dots 1\dots b_1b_0\rangle & \text{if } b_q = 1 \end{cases} \quad (3)$$

By means of the bitwise exclusive or ($\oplus$), this can be expressed as:

$$\begin{aligned} \alpha_i^{\text{out}} = \alpha_{b_{n-1}b_{n-2}\dots b_q\dots b_1b_0}^{\text{out}} &= \begin{cases} u_{00}\alpha_{b_{n-1}\dots 0\dots b_1b_0}^{\text{in}} + u_{01}\alpha_{b_{n-1}\dots 1\dots b_1b_0}^{\text{in}} & \text{if } b_q = 0 \\ u_{10}\alpha_{b_{n-1}\dots 0\dots b_1b_0}^{\text{in}} + u_{11}\alpha_{b_{n-1}\dots 1\dots b_1b_0}^{\text{in}} & \text{if } b_q = 1 \end{cases} \\ &= \begin{cases} u_{00}\alpha_i^{\text{in}} + u_{01}\alpha_{i\oplus 2^q}^{\text{in}} & \text{if } b_q = 0 \\ u_{10}\alpha_i^{\text{in}} + u_{11}\alpha_{i\oplus 2^q}^{\text{in}} & \text{if } b_q = 1 \end{cases} \end{aligned} \quad (4)$$

This means we can compute the output coefficients from the input ones. But this requires traversing each one of the coefficients, with a $\mathcal{O}(2^n)$ complexity. Actually, coefficients associated to both $b_q = 0$ and $b_q = 1$ can be computed simultaneously. This reduces the complexity of the simulation loop to $\mathcal{O}(2^{n-1})$, making in each iteration an effort equivalent to the matrix-vector product of order 2×2 (computation in place).

Simulation of a Factorizable r -qubit Gate. Let us consider the case of simulating a r -qubit gate to an n -qubit register, being this gate factorizable in terms of the Kronecker's product of 1-qubit gates. Its simulation follows from the simulation of the single qubit gate. Basically it consists of applying the gate U to one qubit after another resulting in r consecutive steps.

With the purpose of analyzing the effect of the transformation over the coefficient space, let us study the application of an 1-qubit transformation U over r consecutive qubits, from the q -th to the $(q+r-1)$ -th qubit. Following the argument of equations (3) and (4) we can infer that, for this case, the number of input coefficients that contribute to the calculation of a given output coefficient α_k^{out} is 2^r . Furthermore, this set of coefficients is given by

$$G_k^{q,r} = \bigcup_{m=0}^{2^r-1} \{\alpha_{k\oplus 2^q \cdot m}\} \quad (5)$$

Observe that such groups, just defined, act like closed groups because it is possible to determine, for all $\alpha^{\text{in}} \in G_k$, its corresponding output coefficient α^{out} without need of information outside of G_k . For example, when transforming the 4^{th} and 5^{th} qubits, we will have $q=4$, $r=2$ and $G_k^{4,2} = \{\alpha_k^{\text{in}}, \alpha_{k\oplus 2^4}^{\text{in}}, \alpha_{k\oplus 2 \cdot 2^4}^{\text{in}}, \alpha_{k\oplus 3 \cdot 2^4}^{\text{in}}\}$. Thus, the coefficients α_k^{out} , $\alpha_{k\oplus 2^4}^{\text{out}}$, $\alpha_{k\oplus 2 \cdot 2^4}^{\text{out}}$ and $\alpha_{k\oplus 3 \cdot 2^4}^{\text{out}}$ can be computed from those only included in $G_k^{4,2}$.

The number of disjoint groups existing in the coefficient space is 2^{n-r} . Therefore the computation complexity will be equivalent to apply 2^{n-r} times the

1-qubit gate U to an r -qubit register. In order to simulate a given gate $U^{\otimes n}$ applied to an n -qubit register, computations can be organized by splitting the register into subregister of r qubits. This way we can work over closed groups of a desired size. Note that the partition of the coefficient space into groups is different for the different subregisters.

Simulation of the QFT. The n -qubit Quantum Fourier Transform [10,3] can be implemented as shown in Fig. 2. A straightforward simulation, gate by gate, will involve $\frac{n(n-1)}{2}$ steps which is very inefficient. Nevertheless, a more efficient implementation can be performed by grouping the gates into n stages, one for each k -th qubit, denoted as U_k in Fig. 2: $QFT = U_{n-1} \cdots U_1 \cdot U_0$.

It is remarkable that each one of these stages U_k operates only on one of the qubits through controlled phase transformations and Hadamard gates. Therefore, the computational workload of simulating each stage U_k is the same to the simulation a 1-qubit gate. This way, the resulting sequence of stages follows a similar scheme than the previously described simulation of $U^{\otimes n}$.

4 Implementation

In this section we proceed to map our simulation schemes onto the CUDA architecture for the three cases under study: 1-qubit gate, $U^{\otimes n}$ transformation and QFT. As forementioned, our simulation model (Fig. 1) involves a classical computation (code running on the host), and a quantum computation which is just simulated on the GPU (kernel code running in parallel on the device).

The main challenge when designing the kernel code is to handle the limitations coming from the CUDA features. The first limitation arises from the device's memory system organization. On the one hand, the vector of coefficients describing the state of the quantum register uses to be very large, and only the device's global memory is able to store the whole of it. Although shared memory is two orders of magnitude faster (as it acts like a parallel data cache for each multiprocessor), it has a reduced size and it will not be able to store but a reduced portion of the coefficients. Subsets of coefficients can be transferred from global to shared memory (copy in) when they are frequently reused and therefore a substantial increment of performance is granted. Notice that results will have to be transferred back to the global memory (copy out). On the other hand, an efficient transference among global and shared memory is restricted to contiguous words (coalescing).

Another important limitation comes from the synchronization mechanism inherent to CUDA. This is just a short range synchronism, as it only allows synchronizing threads belonging to the same block. Synchronization of threads pertaining to different blocks must be solved at the host side.

Simulation of 1-qubit Gate U . The simulation of a 1-qubit gate U is derived from the parallel SIMD execution of (4), where U is applied to the q -th qubit of a n -qubit register. According to this expression, the computation of a coefficient

α_k^{out} requires accessing to the coefficient α_k^{in} itself and the coefficient $\alpha_{k \oplus 2^q}^{\text{in}}$. Even more, once these two coefficients are accessed, also $\alpha_{k \oplus 2^q}^{\text{out}}$ can be calculated.

Consequently, the kernel code must determine in parallel every pair of the form $\{\alpha_k, \alpha_{k \oplus 2^q}\}$. In total, as there exists 2^{n-1} pairs, the same number of threads will be required. Each thread is in charge of computing the transformation U for each pair. As a given coefficient belongs to one and only one pair, it is necessary only one read and one write operations in global memory, assuming that each coefficient α_k is located in the k -th position of the state vector in global memory. Thus, when simulating a gate separately, the use of shared memory will not improve the performance because there is no reusing of data transferred from global to shared memory.

Due to the disjointness of different pairs, the coefficients computed after a transformation can be directly overwritten (in-place computation). This way, a higher number of qubits can be simulated. Note that synchronization points become mandatory when consecutive 1-qubit gates are going to be simulated in order to guarantee the correctness of the computation.

Simulation of a Factorizable n-qubit Gate. Let us consider the simulation of a multiple-qubit gate factorizable in terms of the Kronecker's product of 1-qubit gates and/or controlled gates. Without loss of generality, the same 1-qubit gate U is applied to every qubit of the register, that is, the transformation to be analyzed is $U^{\otimes n}$.

A first approach follows from the simulation of the single qubit gate. It consists of applying the gate U to one qubit after another, resulting in n consecutive stages. Due to the lack of inter-block synchronization in the GPU side, synchronization in the host side is necessary. This fact involves a different kernel invocation for each qubit, i.e., for each stage. This solution entails two main disadvantages: the overhead in time due to host synchronization and the inefficiency of not being able to use the fast shared memory.

In contrast to this solution, a more efficient proposal is introduced hereafter. The key idea consists of copying-in a subset of coefficients from global to shared memory, performing all possible computations and then copying-out the results from shared to global memory. These coefficients allocated in shared memory can be reused several times, resulting in more accesses to fast shared memory and less accesses to global memory.

Basically, we consider quantum subregisters of consecutive qubits in such a way that the corresponding groups G_k of expression (5) fit the multiprocessor shared memory. That is, resulting groups must fulfil the condition $\text{Card}(G_k) = S$, being S the number of coefficients fitting the shared memory. Thus, the number of qubits in the subregister to be transformed is $\log_2(S)$. This approach is expressed in Fig. 5 which corresponds with the scheme of Fig. 4(a) for $q = 0$, $r = 4$ and $S = 2^r$, where the gates are applied to the $\log_2(S)$ less significant qubits. Note that in this case, $q = 0$, coefficients in a group G_k are allocated in consecutive memory positions. This fact involves that copy-in/out operations benefit from the coalescing condition.

Fig. 4. Using the shared memory allows to reduce the number of GPU/CPU synchronizations (a), but for the most significant qubits (b) the lack of coalescing may degrade the performance. However contiguous memory locations can be used by defining coalesced groups of coefficients (c).

$$\text{Let } G = G_k^{r,q=0} = \bigcup_{m=0}^{2^r-1} \{\alpha_{k \oplus m}\} \\ / \text{Card}(G) = 2^r = S$$

$$\text{Let } H = G_k^{r-m,q} \cup G_{k+1}^{r-m,q} \dots \cup G_{k+M}^{r-m,q} \\ / m = \log_2(M), q > \log_2(S) \\ / \text{Card}(G_k^{r-m,q}) = 2^{r-m} = \frac{S}{M}, \\ \text{Card}(H) = S$$

Copy_in(G)
Apply $U^{\otimes r}$ to $\alpha \in G$
Copy_out(G)

Copy_in(H)
Apply $U^{\otimes r-m} \otimes I^{\otimes m}$ to $\alpha \in H$
Copy_out(H)

Fig. 5. Kernel for LSB qubits of $U^{\otimes n}$

Fig. 6. Kernel for MSB qubits of $U^{\otimes n}$

When proceeding with the remaining $n - \log_2(S)$ qubits ($q > \log_2(S)$), a first attempt may consist of building new groups $G_k^{q,r}$ with cardinality S , as described above. This situation is shown in Fig. 4(b). Observe that, according to equation (5), the coefficients of these groups are located far apart in global memory when $q > \log_2(S)$. This means that global memory accesses are not coalesced, influencing adversely the performance.

With the purpose of overcoming this lack of coalescing, a superset of coefficient groups can be copied-in/out by selecting them appropriately. Being M the number of these new groups, $\log_2(\frac{S}{M})$ qubits can be now transformed because the maximum number of coefficients in shared memory is S . These M groups are selected in such a way that the coefficients altogether constitute S/M series of M consecutive coefficients. This idea is illustrated in Fig. 6, where H is referred to the superset of groups. In contrast to the approach followed for the LSB qubits, where $\log_2(S)$ gates are simulated, this one reduces the number of qubits to be processed to $\log_2(\frac{S}{M})$, but locality is severely improved. This results in more kernel invocations as depicted in Fig. 4(c), where $q = 4$, $S = 16$ and $M = 4$. Nevertheless, extra copy-in/out operations, including host side synchronizations, are worthwhile because memory accesses are coalesced.

Simulation of the QFT. As discussed in section 3 the QFT can be carried out in n stages, denoted U_k , one for each k -th qubit, as shown in Fig. 2. This way, the resulting chain of stages is analogous to the simulation of $U^{\otimes n}$ previously

described, but the transformation to be considered for each stage (qubit) depends on each qubit. We have considered two implementations. The first one proceeds stage by stage, working on the state vector coefficients stored in the device global memory. This requires n different kernel code invocations, one for each stage U_k .

A better performance can be achieved by using the shared memory to allocate a portion of the coefficient space and transferring data between global and shared memory according to the coalescing criterion. But in this case the computational scheme becomes more complex for two reasons. On the one hand each transformation U_k implies controlled (conditional) updates of the state vector coefficients. This makes each coefficient to be transformed in a different way, as contrasted with $U^{\otimes n}$, where the same matrix is applied to all qubits. On the other hand, transformations U_k must be translated properly when working on the local space of coefficients copied to the share memory, since coalesced consecutive coefficients in the shared memory do not necessarily correspond with consecutive coefficients in the global space.

5 Experimental Results

Simulations of a multiple-qubit gate $U^{\otimes n}$ and the QFT for n qubits have been implemented and tested. The target GPU platform has been an NVIDIA GeForce®8800GTX with the following features: 16 multiprocessors of 8 processors at 1.35GHz, a device global memory of 768MB with a latency up to 600 clock cycles, and 8KB parallel data cache (shared memory) per multiprocessor with a latency of 4 cycles. CUDA 1.1 SDK and toolkit was used. The goal was to evaluate the two main strategies previously introduced: *gate-by-gate* (*stage-by-stage* for the QFT) and *coalescing-improved* implementations. In the case of $U^{\otimes n}$, we select $U = H$ (Hadamard gate) giving rise to the so-called Walsh gate [10].

Table 2 shows the execution time in milliseconds for the *stage-by-stage* and *coalescing-improved* strategies on the GPU platform for different sizes of the quantum register. It is also shown, with the purpose of setting a time reference, a sequential simulation on a Intel Core2 6300 based platform at 1.86GHz with linux OS using libquantum [2], one of most popular quantum simulation software.

The upper register size limit is 26 qubits, which is imposed by the memory size of the device. Notice that each state vector coefficient is a complex number made of two simple-precision floating point numbers (32 bits). Other two significant parameters, for the *coalescing-improved* version, are the number of coefficients fitting the shared memory (S) and the optimal number of consecutive coefficients to be transferred (M). These values are platform-dependent, being $S = 1024$ and $M = 32$ for our device. Also, the number of threads is set to a half of the number of coefficients.

Some facts can be highlighted concerning these results. Firstly, a good scalability with respect to the number of coefficients can be observed for both parallel versions. Secondly, note that the *coalescing-improved* version exhibits a better performance for all the range. In other words, a high performance requires a good exploitation of the device memory hierarchy. On the one hand, data to be reused must be allocated in the shared memory, whenever possible. On the

Table 2. Execution time (msec) for the Walsh gate and the QFT simulations

Implementation		Number of qubits of the quantum register											
		15	16	17	18	19	20	21	22	23	24	25	26
$H^{\otimes n}$	CPU sequential: libquantum	-	-	-	31	78	156	328	688	1453	3031	6281	13062
	GPU stage-by-stage	-	-	-	1.8	3.2	6.1	12.0	24.3	49.9	102	212	439
	GPU coalescing-improved	-	-	-	1.1	2.0	4.1	8.8	18.1	37.1	76.2	158	342
QFT	CPU sequential: libquantum	10	30	80	150	350	730	1560	3260	6710	13910	28160	56890
	GPU stage-by-stage	2.38	4.58	9.09	18.6	39.5	85.2	185	402	874	1897	4100	8847
	GPU coalescing-improved	0.20	0.43	0.81	1.83	3.78	7.84	16.9	35.1	72.9	150	313	667

other hand, it is crucial to coalesce the accesses to the global memory. Finally, the coalescing-improved GPU version can reach a speedup up to 85 relative to the CPU implementation, for the fastest execution.

References

1. Barenco, A., Bennett, C.H., Cleve, R., DiVicenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H.: Elementary Gates for Quantum Computation. *Phys. Rev. A* 52, 3457–3467 (1995)
2. Butscher, B., Weimer, H.: The libquantum Library, <http://www.enyo.de/libquantum/>
3. De Raedt, K., Michielsen, K., De Raedt, H., Trieu, B., Arnold, G., Richter, M., Lipert, T., Watanabe, H., Ito, N.: Massively Parallel Quantum Computer Simulator. *Computer Physics Communications* 176, 121–136 (2007)
4. Deutsch, D.: Quantum Computational Networks. *Proceedings of Royal Society of London, Series A* 425, 73–90 (1989)
5. Deutsch, D., Jozsa, R.: Rapid Solution of Problems by Quantum Computation. *Proceedings of Royal Society of London, Series A* 439, 553–558 (1992)
6. Fujishima, M.: FPGA-Based High-Speed Emulator of Quantum Computing. In: *IEEE Int’l Conference on Computer Design* (2004)
7. Glendinning, I., Ömer, B.: Parallelization of the QC-Lib Quantum Computer Simulator Library. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) *PPAM 2004. LNCS*, vol. 3019, pp. 461–468. Springer, Heidelberg (2004)
8. Grover, L.K.: A Fast Quantum Mechanical Algorithm For Database Search. In: *Annual ACM Symposium on the Theory of Computation*, pp. 212–219 (1996)
9. Khalid, A.U., Zilic, Z., Radecka, K.: FPGA Emulation of Quantum Circuits. In: *IEEE Int’l Conference on Field-Programming Technology* (2003)
10. Nielsen, M., Chuang, I.: *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge (2004)
11. Niwa, J., Matsumoto, K., Imai, H.: General-Purpose Parallel Simulator for Quantum Computing. In: Calude, C.S., Dinneen, M.J., Peper, F. (eds.) *UMC 2002. LNCS*, vol. 2509, pp. 230–251. Springer, Heidelberg (2002)
12. NVIDIA CUDA Programming Guide, SDK and Toolkit, <http://developer.nvidia.com/object/cuda.html>
13. Shor, P.W.: Algorithms for Quantum Computation: Discrete Logarithm and Factoring. In: *35th Symposium on Foundations of Computer Science*, pp. 124–134 (1995)
14. Udrescu, M., Prodan, L., Vladutiu, M.: Using HDLs for Describing Quantum Circuits: A Framework for Efficient Quantum Algorithm Simulation. In: *Computing Frontiers Conference* (2004)