Simulation of Quantum Gates on a Novel GPU Architecture

ELADIO GUTIERREZ, SERGIO ROMERO, MARIA A. TRENAS, EMILIO L. ZAPATA
University of Malaga
Department of Computer Architecture
29071 Malaga, SPAIN

Abstract: Quantum computers aim to achieve a huge reduction of the time required for solving problems with an exponential complexity, but their simulation in conventional computers results itself on a problem with a similar complexity. As this limits considerably the dimensions of the quantum computer we can simulate, multiprocessor architectures are an almost obliged tool when tackling with such simulations. In this paper we explore the application of the new graphical processor architectures in the simulation of the elementary operators that constitute the basic building blocks of quantum computers. Recently, these GPUs are being used as general purpose multiprocessors, as they demonstrate to possess a floating point computing power larger than classic CPUs. In particular our implementation makes use of the new CUDA software/hardware architecture developed recently by NVIDIA.

Key–Words: Quantum computing simulation, Graphics Processing Unit (GPU), parallel processing

1 Introduction

Unlike the conventional (classical) computation, the so denominated quantum computers are devices that process information on the basis of the laws of the quantum physics and that would be able to solve some problems of non-polynomial complexity in a much smaller time [10]. Most of the power of quantum computers is due to the quantum parallelism that allows to perform simultaneous operations on an exponential set of superimposed data. This is why the simulation of this kind of computers requires an exponential effort. Parallelism is a suitable tool in order to mitigate such computational requirements [7, 11], and will allow the emulation of quantum computers of a greater dimension in a reasonable time.

Although the number of known algorithms that are really effective [8, 13, 5] is actually reduced, and there are no physical implementations of operative dimensions, the analysis of this model of computation constitutes, at the moment, a topic of great interest for physicists, computer scientists and engineers. In this context, different simulators has been developed, both in software [3, 7, 11, 12] as in hardware [6, 9, 14].

This work follows the model shown in Fig. 1 [7] where the quantum computer acts like an accelerating hardware of the classical processor, which sends the orders required to solve a concrete problem. According to the laws that govern it, it is not possible to know the state of this quantum computer. Therefore, the output of the quantum algorithm will be obtained by a measurement process with certain probability.

This paper presents a parallel simulation of the basic operators on whom an ideal quantum computer is constructed. New emergent architectures, such as general purpose graphical processors (GP-GPU) [2] are put to use. The interface adopted between the host and the platform on which the quantum computer is simulated is the one defined by libquantum [3]. It is one of the more popular simulation softwares, as well as a part of the well-known benchmark SPEC2006.

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. A measure of the final state provides the output towards the classical world.

A 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 $|1\rangle$, respectively. In vector notation, we can write $\Psi = \binom{\alpha_1}{\alpha_0}$, $|0\rangle = \binom{0}{1}$ and $|1\rangle = \binom{1}{0}$.

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

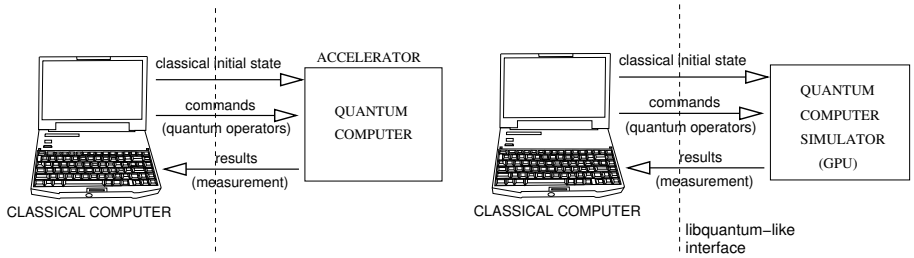

Figure 1: Quantum computer model as a hardware accelerator and its simulation.

classical states provided by n bits. After this, the state of a quantum register can be written as

$$\Psi = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \text{ with } \alpha_i \in \mathbb{C}, \quad \sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1,$$

since $|\alpha_i|^2$ is interpreted as the probability of obtaining $|i\rangle$ when the register is measured in such state.

Let Ψ be an element of a 2^n-dimensional complex vector space, where $|i\rangle$ constitutes a basis, with $0 \leq i \leq 2^n - 1$. For example, for the value $n = 3$, we will write $|5\rangle = |101\rangle = (0\,0\,0\,0\,0\,1\,0\,0)^T$. After applying the Kronecker's tensor product, it is possible to represent the elements of the state space basis for the registry as a function of the individual states of the qubits. For example $|3\rangle = |011\rangle = |0\rangle \otimes |1\rangle \otimes |1\rangle$. Generally, this factorization is not always possible for any state Ψ of the quantum register.

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 $UU^* = I$, where U^* is matrix U both conjugated and transposed, and I is the unitary matrix. As a consequence, every valid transformation must be a reversible one. Usually, this kind of transformations are represented in the manner of Fig. 2(a).

As a particular instance, let us consider the application of a transformation over one particular bit, as shown in Fig. 2(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 $i-$th qubit then:

$$U_g = I \otimes I \overset{n-i-1 \text{ times)}}{\cdots} \otimes U \otimes I \overset{i \text{ times)}}{\cdots} \otimes I = \\ = I^{\otimes n-i-1} \otimes U \otimes I^{\otimes i} \quad (1)$$

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 transformation. 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.

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	$R_z(\alpha) = e^{j\alpha/2}\|0\rangle\langle 0\| + e^{-j\alpha/2}\|1\rangle\langle 1\|$
Phase shift	$\Phi(\delta) = e^{j\delta}\|0\rangle\langle 0\| + e^{j\delta}\|1\rangle\langle 1\|$
2-qubit transformation	
Controled NOT	CNOT=$\|0\rangle\langle 0\|+\|1\rangle\langle 1\|+\|2\rangle\langle 3\|+\|3\rangle\langle 2\|$

Table 1: Some well-known quantum gates.

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. Notice that the number of qubits at the gate's output must be equal to the one at its input, as it is a reversible transformation. This does not occur with conventional logic gates.

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]. Thus, transformations successively applied to subsets of bits can be interpreted as a factorization of the global transformation that is tensorial as refers to the bits, and conventional as refers to its successive applications.

Different minimal sets of gates have been proposed looking for their universality, that is, any n-qubit transformation should be able of being expressed as a tensor product of the chosen gates. It is not possible to find an universal minimal set consisting only on 1-qubit gates. [1] states that a complete set is built from gates $\Phi(\delta)$, $R_z(\alpha)$, $R_y(\theta)$ (1 qubit) and CNOT (2 qubits).

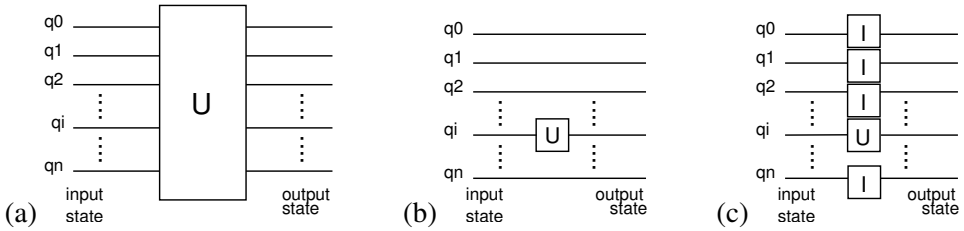

Figure 2: Operating on a n-qubit register.

3 The GPU programming model

A GPU (Graphic Processor Unit) is a device specialized in algorithms such as graphics rendering involving very intensive and highly parallel computations. These devices are nowadays implemented as a set of multiprocessors with a Single Instruction Multiple Data (SIMD) architecture. Due to their high computational power, these GPUs are used both for graphics and general purpose processing. In this scope, they operate as a coprocessor, or hardware accelerator, to the main CPU, or host.

NVIDIA® has recently presented its Compute Unified Device Architecture (CUDA™), as a both hardware and software architecture for issuing and managing computations on the GPU as a truly generic data-parallel computing device with a very high level of parallelism. An extension to the C programming language is provided in order to develop source codes.

CUDA programming model is based on a hierarchy of abstraction layers: grids, blocks, warps and threads. All threads in a block behave as an SIMD, whereas different blocks of a grid are scheduled among the set of multiprocessors by the API runtime in a transparent way. Programmers specify the number and shape (1D, 2D or 3D) of some of such levels, without the additional charge of coding strategies to balance the workload among the actual hardware configuration. However, some limitations exist, among others: the maximum number of threads in a block is 512, a block of threads is executed in one multiprocessor (manufacturer recommends the use of a large number of blocks), memory accesses for all threads in a warp must be coalesced, and only threads in a block can be synchronized at the device side, while the synchronization of different blocks of threads must be explicitly done by the host.

According to this model, an application running on the host invokes a unique kernel code that will be executed for each thread at the device side, but operating over different data sets.

4 Quantum computing 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.

To decompose this transformation in a set of successive transformations with a lower number of qubits (stages), translates into a reduction of the number of operations per stage. This idea can be illustrated by means of the application of a 1-qubit quantum gate, which performs the operation $|\Psi^{\text{out}}\rangle = U_g|\Psi^{\text{in}}\rangle$. U_g comes from the 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.

Let us 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 $\Psi^{\text{in}} = |i\rangle = |b_{n-1}b_{n-2}...b_1b_0\rangle$, that is, a element of the space of states base, where b_k represent the bits on the binary expression of natural i. Transformation U over the q-th bit b_q will result on:

$$\Psi^{\text{out}} = |b_{n-1}\rangle \otimes |b_{n-2}\rangle \otimes ...U|b_q\rangle \otimes ...|b_1\rangle \otimes |b_0\rangle =$$
$$= \begin{cases} u_{00}|b_{n-1}...0...b_1b_0\rangle + u_{10}|b_{n-1}...1...b_1b_0\rangle \text{ if } b_q = 0 \\ u_{01}|b_{n-1}...0...b_1b_0\rangle + u_{11}|b_{n-1}...1...b_1b_0\rangle \text{ if } b_q = 1 \end{cases} \quad (2)$$

This leads to:

$$\alpha_i^{\text{out}} = \alpha^{\text{out}}_{b_{n-1}b_{n-2}...b_q...b_1b_0} =$$
$$= \begin{cases} u_{00}\alpha^{\text{in}}_{b_{n-1}...0...b_1b_0} + u_{01}\alpha^{\text{in}}_{b_{n-1}...1...b_1b_0} \text{ if } b_q = 0 \\ u_{10}\alpha^{\text{in}}_{b_{n-1}...0...b_1b_0} + u_{11}\alpha^{\text{in}}_{b_{n-1}...1...b_1b_0} \text{ if } b_q = 1 \end{cases}$$
$$= \begin{cases} u_{00}\alpha_i^{\text{in}} + u_{01}\alpha^{\text{in}}_{i\oplus 2^q} & \text{if } b_q = 0 \\ u_{10}\alpha^{\text{in}}_{i\oplus 2^q} + u_{11}\alpha_i^{\text{in}} & \text{if } b_q = 1 \end{cases} \quad (3)$$

where $\oplus$ stands for the bitwise logical exclusive-or.

This means we can compute the output coefficients from the input ones. But this requires to traverse 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).

A generalization of expression 3 to a p-qubit gate, its simulation will imply a loop of $\mathcal{O}(2^{n-p})$ iterations with an iteration load equivalent to the matrix-vector product of order $2^p \times 2^p$. Therefore, whenever it is possible a tensorial factorization of a generic n-qubits transformation in K stages of 1-qubit gates, the complexity of the simulation would be reducing to $\mathcal{O}(K2^{n-1})$ instead of $O(2^{2n})$.

5 Parallel implementation

This work focuses on the parallel simulation of a $U^{\otimes n}$ operator applied to a n qubits register, based on an 1-qubit elementary transformation U. 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).

When designing the kernel code some limitations may be encountered. The first one arises from the device's memory system organization. On the one hand, the vector of coefficients describing the quantum state of the registry is very big, and only the global memory is able to store the whole of it. Although shared memory is much more fast (it is local to the blocks of threads), it has a reduced size and it will not be able to store but a reduced portion of the coefficients. Subsets of coefficients are 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. If not so, memory accesses will be serialized, which affects negatively on the efficiency of the parallel code.

Another important limitation comes from the synchronization mechanism inherent to CUDA. It only allows to synchronize threads belonging to the same block, not providing synchronization between blocks. Notice that, as the number of threads may be several orders of magnitude bigger than the number of threads per block, this is just a short range synchronism. Synchronization of threads pertaining to different blocks will be solved after returning the control to de host, which gives rise to a significant overhead.

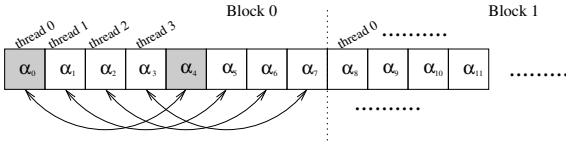

Figure 3: Each thread computes the 1-qubit transformation for a pair of coefficients; in this example the qubit no. 2 is transformed with 8 threads per block.

Simulation of 1-qubit gate U The simulation of a 1-qubit gate U is derived from the parallel SIMD execution of Eq. 3, where a quantum transformation U is applied to the q-th qubit of a n-qubit register. According to this expression observe that the computation of a coefficient α_i^{out} requires accessing to the coefficient α_i^{in} itself and the coefficient $\alpha_{i\oplus 2^q}^{\text{in}}$. Even more, once these two coefficients are accessed, also $\alpha_{i\oplus 2^q}^{\text{out}}$ can be calculated.

Consequently, the CUDA kernel code must determine in parallel every pair of the form $\{\alpha_i, \alpha_{i\oplus 2^q}\}$. 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 (Fig. 3).

As a given coefficient belongs to one and only one pair, it is necessary only one read and one write operations 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 particular case of simulating a multiple-qubit gate factorizable in terms of the Kronecker's product of 1-qubit gates. Without loss of generality the gate is applied to every qubits of the register, and the same 1-qubit gate U is used. 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 in 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 are 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 inefficacy of not being able to use the fast shared memory.

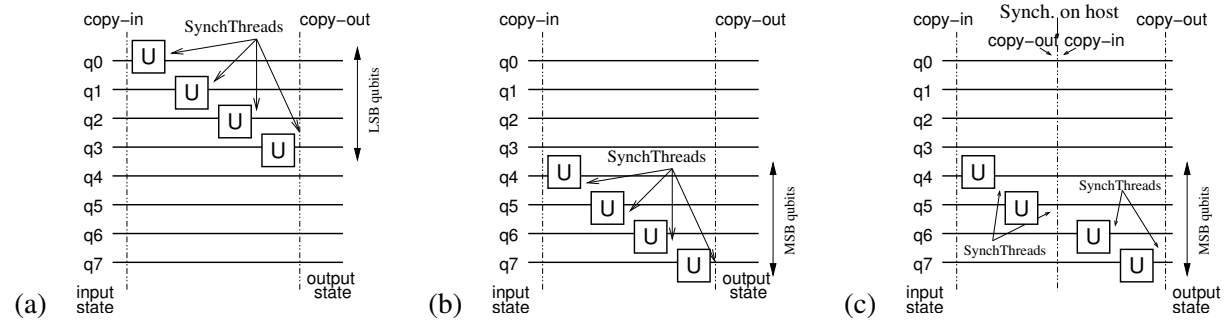

Figure 4: Shared memory allows to save synchronizations on host (a), but when applied to the MSB qubits (b) lack of coalescing may cause serialization that can be mitigated reusing contiguous memory locations (c).

In contrast to this solution, a more efficient proposal is introduced hereafter. The key idea consist of copying-in a subset of coefficient from global to shared memory, perform 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. This is, more accesses to fast shared memory and less accesses to global memory.

This proposal proceeds as depicted in Fig. 4(a). After copying-in all P pairs of coefficients that fit in the share memory of a block, all the threads in such a block performs $log_2(P) + 1$ stages (1-qubit gates). A synchronization barrier is required between two consecutive stages. Note that this synchronization can be achieved at thread level for threads in the same block, with an overload of only two clock cycles. Finally coefficients are copied-out to the global memory. In general the size of the shared memory is not large enough to allocate all coefficients of the state vector. So, for a high number of qubits, the previous procedure must be repeatedly invoked, processing at most $log_2(P)+1$ qubits each time (Fig. 4(b)).

Nevertheless, excluding the $log_2(P) + 1$ lowest significant qubits, the application of such a procedure for subsequent qubits results in a loss of performance. This fact is derived from the memory access pattern generated when coefficients are accessed. The more significant qubits, the higher memory address strides are generated. As the device memory system is organized into interleaved banks, threads in a warp must coalesce with memory accesses, i.e., they must access to contiguous aligned memory addresses in order to be efficient. Otherwise, the memory operations may be actually serialized. Therefore, being M the number of consecutive coefficients that provide the maximum coalescing, the shared memory for a block may allocate $2P/M$ groups of M coalesced coefficients. The key feature is selecting the base address of every group in such a way that its stride allows to compute the $log_2(P/M) + 1$ transformations for next qubits. Getting coalescing involves a lower level of reuse of coefficients stored in the shared memory because they can not be chosen arbitrarily without a high memory bandwidth penalty. As shown in the example of Fig. 4(c), where $P = 4$ and $M = 2$, memory access coalescing means introducing extra synchronization on host, which in turn involves copy-out and copy-in operations.

6 Results

Following the strategies above discussed, a multiple-qubit gate $U^{\otimes n}$ has been simulated. In particular $U^{\otimes n}$ has been built from the 1-qubit Hadamard transformation (Table 1), giving rise to the so-called Walsh gate [10].

Experiments have been conducted on a NVIDIA GeForce®8800GTX GPU, which includes 16 multiprocessors of eight processors working at 1.35GHz with a device memory of 768MB. In addition, each multiprocessor has a 8KB parallel data cache (shared memory). The latency for global memory is about 200 clock cycles, whereas the latency for the shared memory is only one cycle (conflict free accesses). Parallel execution is limited to 512 threads per block, scheduled in warps of 32 threads.

Experimental results are summarized in Table 2, where the execution time of three implementations are shown for different number of qubits. Upper limit for this parameter is 26 qubits, imposed by the memory size of the device. Walsh gate simulated qubit by qubit (that is, applying the 1-qubit Hadamard gate to every qubit) is labelled as *GPU Version I*. Remember that this approach requires synchronizing on host for every 1-qubit gate invocation. With *GPU Version II* we refer to the second approach of the section 5 (Fig. 4(c)) with parameters $P = 512$, $M = 32$. This approach benefits from the use the shared memory, reducing the synchronization at the host side and keeping memory

Number of qubits	18	19	20	21	22	23	24	25	26
CPU (sequential)	31	78	156	328	688	1453	3031	6281	13062
GPU Version I	1.8	3.2	6.1	12.0	24.3	49.9	102	212	439
GPU Version II	1.1	2.0	4.1	8.8	18.1	37.1	76.2	158	342

Table 2: Simulation of the Walsh gate: execution time (msec) on the GPU and the CPU.

coalescing for warps of threads. For both versions, the total number of threads was a half of the number of coefficients of the state vector. With the purpose of setting a time reference, the simulations have been also sequentially executed on a Intel Core 2 based platform at 2.13GHz.

Three facts about these results can be highlighted. First we can observe that a good scalability is obtained for both parallel versions, having into account that the complexity of simulation is $\mathcal{O}(n2^n)$. Secondly, note that the *GPU Version II* exhibits a better performance for all the range. So, whenever possible (tensor-factorizable gates) this second version should be chosen. Finally, comparing the GPU times with those of CPU, a relatively high speed-up is achieved, near 40 for the fastest execution.

7 Conclusions

This work presents simulation results for several basic algorithms related to quantum computation. Particularly, a tensor-factorizable multi-qubit gate has been analyzed. As these algorithms involve a high computational complexity, a parallel implementation is selected, and moreover, a brand new GPU platform is chosen.

In order to take advantage of the architectural characteristics of the target GPU platform, two alternatives are proposed. The main concerns are to diminish the number of required synchronization points between target and host, and to exploit the parallel data cache of the target device.

Experimental results exhibit both a high scalability with the size of the quantum register, as well as a good speedup when compared with a conventional monoprocessor platform.

References:

[1] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVicenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, and H. Weinfurter. Elementary gates for quantum computation *Phys. Rev. A*, 52(5): 3457-3467, Nov. 1995.

[2] NVIDIA CUDA Homepage. Available at: http://developer.nvidia.com/object/cuda.html

[3] B. Butscher, H. Weimer. The libquantum library. Available at: http://www.enyo.de/libquantum/

[4] D. Deutsch. Quantum Computational Networks. *Proceedings of Royal Society of London*, A425:73-90, 1989.

[5] D. Deutsch and R. Jozsa. Rapid Solution of Problems by Quantum Computation. *Proceedings of Royal Society of London*, A: 439-553, 1992.

[6] M. Fujishima. FPGA-Based High-Speed Emulator of Quantum Computing *IEEE Int'l Conference on Computer Design*, 2004.

[7] I. Glendinning and B. Ömer. Parallelization of the QC-lib Quantum Computer Simulator Library. *Lectures Notes in Computer Science*, 3019: 461-468, 2004.

[8] L.K. Grover. A Fast Quantum Mechanical Algorithm For Database Search. *Annual ACM Symposium on the Theory of Computation*, 212-219, 1996.

[9] A.U. Khalid, Z. Zilic, K. Radecka. FPGA Emulation of Quantum Circuits. *IEEE Int'l Conference on Field-Programming Technology*, 2003.

[10] M.A. Nielsen and I.L Chuang. Quantum Computation and Quantum Information. Cambridge University Press, 2004.

[11] J. Niwa, K. Matsumoto and H. Imai. General-Purpose Paralel Simulator for Quantum Computing *Phys. Rev. A*, 66(6): 623171–6231711, 2002.

[12] K. De Raedt, K. Michielsen, H. De Raedt, B. Trieu, G. Arnold, M. Richter, T. Lippert, H. Watanabe and N. Ito. Massively Parallel Quantum Computer Simulator. *Computer Physics Communications*, 176:121–136, 2007.

[13] P.W. Shor. Algorithms for Quantum Computation: Discrete Logarithm adn Factoring. *Proc. 35th Symposium on Foundations of Computer Science*, 124–134, 1995.

[14] M. Udrescu, L. Prodan and M. Vladutiu. Using HDLs for Describing Quantum Circuits: A Framework for Efficient Quantum Algorithm Simulation. *Computing Frontiers Conference*, 2004.