Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. V (2010), No. 5, pp. 634-641

Parallel Simulation of Quantum Search

S. Caraiman, V. Manta

Simona Caraiman, Vasile Manta
"Gheorghe Asachi" Technical University of Iasi
Romania, 700050 Iasi, 27 Dimitrie Mangeron
E-mail: {sarustei,vmanta}@cs.tuiasi.ro

Abstract: Simulation of quantum computers using classical computers is a computationally hard problem, requiring a huge amount of operations and storage. Parallelization can alleviate this problem, allowing the simulation of more qubits at the same time or the same number of qubits to be simulated in less time. A promising approach is represented by executing these simulators in Grid systems that can provide access to high performance resources. In this paper we present a parallel implementation of the QC-lib quantum computer simulator deployed as a Grid service. Using a specific scheme for partitioning the terms describing quantum states and efficient parallelization of the general singe qubit operator and of the controlled operators, very good speed-ups were obtained for the simulation of the quantum search problem.
Keywords: quantum computer simulation, parallel computing, quantum search.

1 Introduction

The research in quantum informatics has gained an immense interest due to the remarkable results obtained for the factorization [11] and search [6] problems. These results prove the huge computational power of a quantum machine with respect to the classical computers. However, building quantum computers represents an immense technological challenge and, at present, the quantum hardware is only available in research labs. Under these circumstances quantum simulators have become valuable instruments in developing and testing quantum algorithms and in the simulation of physical models used in the implementation of a quantum processor.

According to Feynman's paper [3], classical computers will never be able to simulate quantum systems in polynomial time. The simulation of 29 qubits (quantum bits) uses 32 GB of memory [1] and any additional qubit doubles the resources needed: time, memory, computational power and space.

In this paper we present a solution based on Grid computing for the quantum simulation problem. Our simulator relies on paralel processing for storing quantum states and applying quantum operators. The deployment of this solution in Grid systems provides access to high performance computing devices for simulation and availability in the context of collaboration through the means of Virtual Organizations.

Our quantum simulator, GQCL, partitions the terms coresponding to a quantum state between several procesing nodes using a scheme that minimizes communication between nodes during the application of quantum operators. In a previous paper [1] we describe the development of a grid service that provides this functionality to client applications by enabling the Quantum Computation Language [9] through a parallel implementation of the QC-lib simulator [8]. The results recorded for the application of the Hadamard transform illustrate the performances of this approach [1]. In the following we present the parallelization of the general single qubit operator, the conditional operators and of the measurement process. This allows us to study the performance of our simulator regarding the quantum search problem.

2 Basic Concepts in Quantum Computing

The quantum analogous of the classical bit is the qubit. A qubit is a quantum system whose states can be completely described by the superposition of two orthonormal basis states, labeled $|0\rangle$ and $|1\rangle$ (in a Hilbert space $\mathcal{H} = C^2$, $|0\rangle = (1 \quad 0)^T$, $|1\rangle = (0 \quad 1)^T$). Any state $|\Psi\rangle$ can be described by:

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad |\alpha|^2 + |\beta|^2 = 1, \tag{1}$$

where α and β are complex numbers. Thus, unlike the classical bit, the qubit can also be in a state different from $|0\rangle$ and $|1\rangle$: linear combinations of states can be formed, called superpositions (eq. 1). When measuring a qubit either the result 0 is obtained, with probability $|\alpha|^2$, or 1 with probability $|\beta|^2$. The sum of the probabilities must be 1, so the state of a qubit represents a unit vector in a complex bi-dimensional vector space.

A collection of n qubits is called a quantum register with dimension n. The general state of a n-qubit register is

$$|\Psi\rangle = \sum_{i=0}^{2^n-1} a_i|i\rangle, \tag{2}$$

where $a_i \in \mathcal{C}$, $\sum_{i=0}^{2^n-1} |a_i|^2 = 1$. This means that the state of a n-qubit register is represented by a complex unit vector in Hilbert space $\mathcal{H}_{2^n}$.

The quantum analogous of the classical NOT gate is labeled X and can be defined such that $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$. The quantum NOT gate acts similarly with its classical counterpart, although, unlike in the classical case, its action is linear: state $\alpha|0\rangle + \beta|1\rangle$ is transformed in a coresponding state $\beta|0\rangle + \alpha|1\rangle$. A convenient way of representing the action of the quantum NOT gate is in matrix form:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{3}$$

Controlled gates are quantum logical gates acting on more than one qubit. The notion of controlled gate allows the implementation of the $\mathtt{if-else}$ constructs. Quantum controlled gates use a control qubit to determine whether a specific unitary action is applied to a target qubit.

The controlled-NOT operator (CNOT) is the prototypical multi-qubit gate. The first parameter of a CNOT gate is the control qubit. If this qubit is in state $|0\rangle$, the target qubit is left unchanged and if the control qubit is in state $|1\rangle$, the target qubit is flipped:

$$|00\rangle \to |00\rangle; \quad |01\rangle \to |01\rangle; \quad |10\rangle \to |11\rangle; \quad |11\rangle \to |10\rangle.$$

The CNOT operator is a generalization of the classical XOR, since its action can be summarized as $|x, y\rangle \to |x, x \oplus y\rangle$, where $\oplus$ is addition modulo two. The matrix representation of CNOT is:

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{4}$$

There are several other multi-qubit gates, Nevertheless, the controlled-NOT gate and the single qubit gates represent the prototypes for any other quantum gate because of the folowing remarkable universality result: any multi-qubit gate can be built out of CNOT gates and single qubit gates. The proof of this statement represents the quantum analogous to the universality of the classical NAND gate.

3 Parallel Simulation of Quantum Computation

The state of a n-qubit register is represented by a complex unit vector in Hilbert space $\mathcal{H}_{2^n}$. Storing a complex number $a = x + iy$ on a classical computer requires storing the pair of real numbers (x, y) for which the 8 byte representation is prefered. Thus, in order to store an n-qubit quantum register using a conventional (classical) computer, at least 2^{n+4} bytes are required. The memory needed for simulating a n-qubit quantm computer grows exponentially with respect to the number n. For example, when $n = 24$ ($n = 36$) at least 256 MB (1 TB) of memory is required to store a single arbitrary state $|\Psi\rangle$. The time evolution of a n-qubit quantum register is determined by a unitary operator defined in the $\mathcal{H}_{2^n}$ space. The matrix dimension is 2^n x 2^n. In general, 2^n x 2^n space and $2^n(2^{n+1} - 1)$ arithmetic operations are needed to execute such an evolution step.

Thus, the simulation of a quantum computer using a classical device represents a computationally hard problem and the memory and processor generate drastic limitations on the size of the quantum computer that can be simulated. Because of the exponential behavior of quantum systems, simulation using classical computers enforces the use of exponential memory space and the execution of an exponential number of operations. It is obvious that the simulation of quantum problems of interesting sizes enforces the use of high performance computing devices. Parallel computing can represent a solution to this problem [5, 7, 10, 13].

Nevertheless, the development of a quantum simulator must consider another important aspect: it has to be easily accessible. But this contradicts the first requirement, that it is parallel, which deeply restricts the group of potential users. A solution based on the concept of Grid systems ca be used to solve this contradiction and to provide the scientific community with an useful and easily accessible instrument. The Grid concept adresses the problem of coordinated resource sharing and problem solving in dinamic, multi-institutional virtual organisations [4].

A Grid enabled quantum computer simulator is GQCL [1]. This simulator allows the use of the QCL [9] quantum programming language to implement quantum algorithms and the quantum programs are executed using a parallel version of the QC-lib simulation library [8]. Using a specific data partitioning scheme and efficient storing of quantum states allowed very good speedups and efficiency of this parallel implementation which will be discussed in the following.

3.1 Overview of GQCL Grid Service for Quantum Simulation

Our quantum computer simulator is based on the QC-lib simulation library which provides a framework to execute programs written in a quantum programming language, QCL, in the absence of quantum hardware. The reasons that lead to this choice for a quantum programming language are detailed in [1] and mainly consider the representation of the quantum state using complex numbers, the possibility to write complex quantum operators, the classical extension and its universality. QCL was conceived by Ömer [9] and the first version appeared in 1998 and the last one in 2004. It is open-source running under Linux operating system and it is a procedural high level language with a C like syntax. QC-lib is a C++ library for the simulation of quantum computers at an abstract functional level [8], and it is used as the back end interpreter for QCL.

For the execution of quantum programs written in QCL we developed a parallel version of the QC-lib simulator in which the terms representing quantum states are distributed across multiple processing nodes. We have chosen to expose the parallel implementation of QC-lib through a Globus Toolkit 4 (GT4 for short) Grid service. Parallelization has been achieved through the use of LAM 7.1.2/7.1.4 implementation of the MPI-2 standard. In GQCL, the execution of

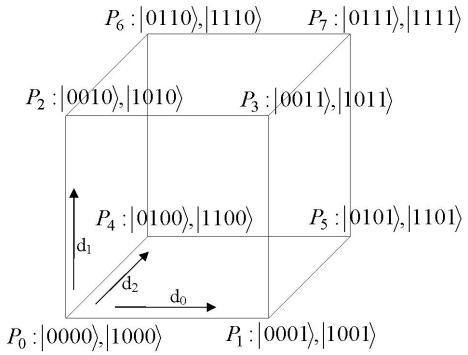


Figure 1: Distribution of the 16 basis states of a 4-qubit register using $2^3 = 8$ proccesing nodes. The processing nodes represent the corners of a 3-dimensional hypercube.

the MPI implementation of QC-lib is enabled through the means of a wrapper service based on the Factory/Instance architecture [12]. In GQCL, the instance service, is responsible for actually managing the quantum simulation as a Grid job. Through the use of a *WS-Resource*, the instance service starts and monitors the job state, notifying the client application on any relevant changes. Also, file staging is automated and when the job finishes, the client is given access to the results of the simulation.

The architectural details of GQCL and its advantages are presented in [1]. In the following we discuss the complete parallel implementation of the simulation library, addressing the issues regarding the representation of quantum states using multiple processing nodes, the application of single- and multi-qubit quantum operators and measurements.

3.2 Parallel Implementation of the QC Library

A quantum register in QCL contains a number of basis vectors, each with a corresponding amplitude. When the qubits forming the quantum register are in a superposition of states, the number of vectors grows exponentially. In QC-lib, the superposition state of a 1-qubit register is represented by two basis vectors (terms) for which the correponding complex amplitudes must be stored: $2 \text{ x } \texttt{complex} < \texttt{double} >= 32$ bytes. When appplying a quantum operator, two term lists are created: one for storing the terms in the current state and one to accumulate the result of an operation on the state. This gives a total of 64 bytes/term which for a n-qubit register requires the use of 2^{n+6} bytes. Thus for $n = 25$ ($n = 29$) qubits 2 GB (32 GB) of memory is necessary.

In order to provide an efficient parallel execution of QC-lib we take advantage of the specific form of the quantum computation process and distribute the 2^n basis states of a quantum register to 2^p processors based on the p least significant bits. In this representation, the processing nodes and the basis vectors are actually considered the coordinates of a n-dimensional hypercube (Figure 1). Each processing node applies quantum operators only to local terms and communicates the generated terms to corresponding processing nodes if necessary. Another feature of our implementation is that for a quantum state only non-zero amplitude terms are stored thus diminishing the communication costs in early stages of some operator execution and the space required to store a quantum state.

The General Single Qubit Operator. Communication between processing nodes is only necessary when applying the operator to a qubit determining the data distribution. Applying a

general single qubit operator $U = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix}$ on a single qubit with state $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ yelds

$$U|\Psi\rangle = (\alpha u_{11} + \beta u_{12})|0\rangle + (\alpha u_{21} + \beta u_{22})|1\rangle \tag{5}$$

If 2 processors are used, then each processor holds the amplitude of one basis state. Applying operator U locally on each processor, terms are created that are not owned by the processor, and so communication of these terms is needed:

$$\begin{array}{l} P_0 : \alpha|0\rangle \xrightarrow{U} u_{11}\alpha|0\rangle + u_{21}\alpha|1\rangle \\ P_1 : \beta|1\rangle \xrightarrow{U} u_{12}\beta|0\rangle + u_{22}\beta|1\rangle \end{array} \xrightarrow{\text{Comm}} \begin{array}{l} u_{11}\alpha|0\rangle + u_{12}\beta|0\rangle \\ u_{21}\alpha|1\rangle + u_{22}\beta|1\rangle \end{array}$$

For each term in the initial state at most two terms are created, out of which at most one needs to be communicated. If working, for example, with an n-qubit register and 2^p processors, communication is necessary only when applying a single qubit operator on any of the qubits that form the distribution key. For the rest of the qubits, all the terms needed for computing the amplitude of the resulting state are locally owned by each processor. Moreover, in the first case, for each processor, all the remotely owned terms are owned by the same other processor as a single bit is flipped in the distribution key.

The parallel implementation of the general single qubit operator allows the parallel execution of NOT, Hadamard, phase shift of the amplitude and exponentiation gates.

Controlled Gates. CNOT operator (controlled-NOT) In the parallel implementation of QC-lib, when the control qubit is in state $|0\rangle$, the state of the target qubit doesn't change, so no new terms are generated and the amplitudes of existing terms are left unchanged. When the control qubit is in state $|1\rangle$, the state of the target qubit is flipped. In this case new terms are generated that need to be communicated to another processing node if the target qubit is part of the distribution key. For example, working with 4 processing nodes and applying CNOT to the least significant qubit in a 3-qubit register initially in the state $\alpha|100\rangle + \beta|011\rangle$, and the control qubit is qubit 2, the following evolution is obtained:

$$\begin{array}{l} P_0 : \alpha|100\rangle \xrightarrow{\text{CNOT}} \alpha|101\rangle \\ P_1 : - \\ P_2 : - \\ P_3 : \beta|011\rangle \xrightarrow{\text{CNOT}} \beta|011\rangle \end{array} \xrightarrow{\text{Comm}} \begin{array}{c} - \\ \alpha|101\rangle \\ - \\ \beta|011\rangle \end{array}$$

In QC-lib the CNOT operator can act on two registers: the control register and the target register. These registers can represent substates of the quantum basis state (sub-registers). In this case, the CNOT gate inverts the state of the target (sub-)register if the control (sub-)register is in the state $|1\rangle_c$, where c is the number of qubits in the control register. For example, let a be a 1-qubit register in state $|0\rangle$, b a 2-qubit register in state $\alpha_0|01\rangle + \beta_0|10\rangle$ and c a 2-qubit register in state $\alpha_1|10\rangle + \beta_1|11\rangle$. Applying CNOT to target register b with control register c, the state of the entire quantum memory distributed to 8 processing nodes will be:

$$\begin{array}{l} P_{\mathbf{2}} : |10010\rangle, |11010\rangle \xrightarrow{\text{CNOT}} |10010\rangle, |11\mathbf{1}00\rangle \\ P_{\mathbf{4}} : |10100\rangle, |11100\rangle \xrightarrow{\text{CNOT}} |10100\rangle, |11\mathbf{0}10\rangle \end{array} \xrightarrow{\text{Comm}} \begin{array}{l} |10010\rangle, |11010\rangle \\ |10100\rangle, |11100\rangle \end{array}$$

Similar to the case of the general single qubit operator, each processing node communicates with at most one other process node. The index of the process involved in communication is determined by the qubits of the target (sub-)register that make part of the distribution key of the whole quantum memory.

The CPhase operator is another example of 2-qubit quantum gate implemented in QC-lib and allows for a controlled phase shift of the amplitudes. Its inputs are a rotation angle, θ, and a control qubit that acts in the same manner as in the CNOT case. The amplitudes of the basis states where the control qubit is $|0\rangle$ are left unchanged, an if the control qubit is in state $|1\rangle$, the phase of the amplitudes of the basis states are multiplied by $e^{i\theta}$. The matrix form of the CPhase operator is:

$$CPhase = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{pmatrix}. \tag{6}$$

Operator CPhase can also act on (sub-)registers of the quantum memory and its parallel implementation is analogous to that of the CNOT operator. One important difference between the two implementations is that when applying the CPhase operator communication between processing nodes is not necessary as the action of this operator doesn't generate new terms, and only the amplitudes of the local terms are modified.

Measurement of Quantum States. In QC-lib, the measurement of a n-qubit quantum register is simulated in $O(2^n)$ time. Let $|\Psi\rangle = \sum_{j=0}^{2^n-1} \alpha_j |j\rangle$ be the state of a n-qubit quantum register. The measurement step is simulated in the following manner:

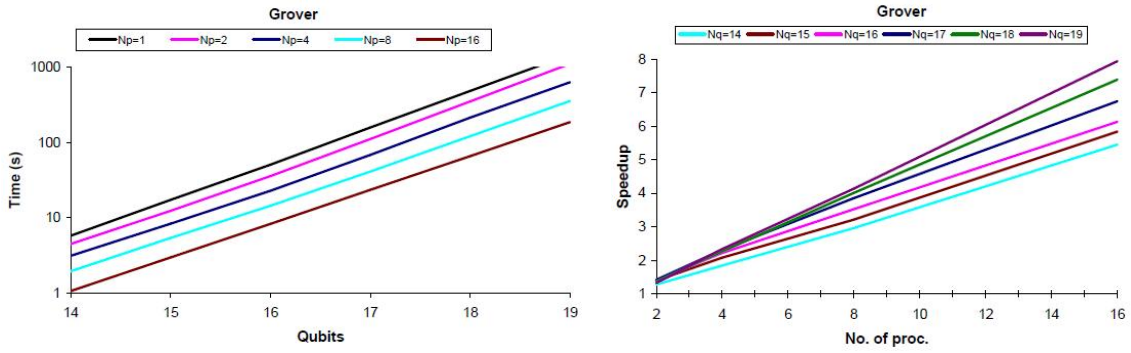
1. Rrandomly generate a number p, $0 \leq p < 1$,
2. Randomly generate a positive integer x, smaller than the number of terms with non-zero amplitude,
3. Determine an integer i, $0 \leq i < 2^n - 1$, such that $\sum_{j=x}^{i-1} |\alpha_j|^2 \leq p < \sum_{j=x}^{i} |\alpha_j|^2$.

Integer i is the representation of the measured state. After measurement, the state of the register becomes $|i\rangle$. Because the terms of the quantum register are distributed across processing nodes, the measurement operation requires communication between these nodes in order to correctly select the term i, but also to colapse the register in state $|i\rangle$. Thus, a master process is responsible with the random generation of numbers p and i and with computing the sum. Sincronization between processing nodes is achieved using MPI_Bcast operations such that the master process could receive the norm of the amplitude of a term j from the owning processing node. After selecting number i, all processing nodes know this number and can pass the state of the quantum register in $|i\rangle$.

4 Simulation of Quantum Search in GQCL

Many problems in classical computing can be reformulated to express the search of a unique element that satisfies a certain predefined condition [2]. If there is no additional information about the search condition, the best classical algorithm is a brute-force search, meaning that the elements are sequentially tested against the search condition. For a list of N elements, this algorithm executes an average of $N/2$ comparisons. By exploiting the advantages of quantum parallelism and interference of quantum states, Grover formulated a quantum algorithm that can find the searched element in an unstructured database in only $O(\sqrt{N})$ steps [6].

Grover's algorithm is based on the concept of amplitude amplification and its principle is to encode the elements in the data set as quantum states of a quantum register and to apply an operator, G, whose effect is to raise the probability that the system finds itself in the marked state (the state encoding the solution of the search problem). Because only unitary transformations are used to act upon the system, the probability conservation takes place. This allows that as the probability that the system finds itself in the desired state grows, the probability of all



(a) (b)

Figure 2: *(a)* Execution time for Grover's algorithm; (b) Speed-up for Grover's algorithm.

other (unmarked) states are correspondingly diminished. Applying Grover's operator G a certain number of times will determine the probability of the marked state to be very close to $\mathsf{1}$. In order to acheive this behavior of the quantum system, a Grover iteration first inverts the phase of the amplitude of the marked state and then inverts the phase of the amplitude of all states around the mean. The inversion of the solution state can be obtained using a so-called "black box" function (known as a quantum oracle) which must be able only to identify wheather a certain record is member of the solution set and thus the mechanism is very general.

After one application of Grover operator, the amplitude of the marked state grows with a factor of $\mathsf{O}(\frac{1}{\sqrt{\mathsf{N}}})$, while the amplitudes of the unmarked states lower correspondingly. To obtain $\mathsf{O}(1)$ probability for the solution state, Grover iteration must be applied $\mathsf{O}(\sqrt{\mathsf{N}})$ times. There is a finite probability that the search operation doesn't end in success, in which case, Grover's algorithm must be repeated.

The performance of our GQCL quantum simulator with respect to the quantum search problem is evaluated. In order to eloquently compare the running times for different problem sizes, we only measure the execution time for one application of Grover's algorithm. In figure 2 we present the results obtained for different problem sizes on various numbers of processing nodes. It can be observed from figure 2(a) that the run time grows with an average factor of about 2.8 for each additional qubit. This is due to the fact that each extra qubit represents a doubling of the problem size and that the number of applied Grover iterations grows with a factor of $\sqrt{2}$ for an additional qubit. Variation of the speedup with the number of processing nodes is presented in figure 2(b). For only 19 qubits we obtain a speed-up of 7.9 and the measurements reveal a growing trend of the speed-up with the increase of the problem size.

5 Conclusions

Classical sequential computers enforce drastic limitations over the quantum computation simulation process. Quantum computer simulators have become an attractive alternative for experimentation with quantum algorithms, but their purpose cannot be achieved without significant computing resources. A promising approach is represented by executing these simulators in Grid systems that can provide high performance resources. The quantum computer simulator described in this paper relies on parallel processing implemented in QC-lib. Besides the parallelization of the general single qubit operator, we also described the parallelization of the control gates (CNOT, CPhase) and of the measurement process. The efficient representation and partitioning of the quantum states using the distributed memory of a computer cluster allowed

very good speed-ups to be recorded at the execution of Grover's search algorithm.

Bibliography

[1] S. Caraiman, A. Archip, and V. Manta. A grid enabled quantum computer simulator. In *Proc. of SYNASC'09*. IEEE Computer Society, 2009.

[2] S. Caraiman and V. Manta. New applications of quantum algorithms to computer graphics: the Quantum Random Sample Consensus algorithm. In *Proc. of ACM CF '09*, pages 81–88, New York, NY, USA, 2009. ACM.

[3] R. Feynman. Simulating physics with computers. *Int. J. Theor. Phys.*, 21(6):467–488, 1982.

[4] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, 2001.

[5] I. Glendinning and B. Omer. Parallelization of the QC-lib quantum computer simulator library. In *Proc. of PPAM 2003*, volume 3019 of *LNCS*, pages 461–468. Springer, 2004.

[6] L. Grover. A fast quantum mechanical algorithm for database search. In *Proc. of 28th ACM Annual STOC*, pages 212–219, 1996.

[7] J. Niwa, K. Matsumoto, and H. Imai. General-purpose parallel simulator for quantum computing. In *Proc. of UMC '02*, pages 230–251, London, UK, 2002. Springer-Verlag.

[8] B. Ömer. Simulation of quantum computers, 1996. http://tph.tuwien.ac.at/ oemer/doc/qcsim.ps.

[9] B. Ömer. *Structured Quantum Programming*. PhD thesis, TU Vienna, 2003.

[10] K. D. Raedt, K. Michielsen, H. D. Raedt, B. Trieuc, G. Arnoldc, M. Richterc, T. Lippertc, H. Watanabed, and N. Itoe. Massively parallel quantum computer simulator. *Computer Physics Communications*, 176(2):121–136, 2007.

[11] P. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proc. of SFCS '94*, pages 124–134. IEEE Computer Society, 1994.

[12] B. Sotomayor. The Globus Toolkit 4 programmer's tutorial, 2005. http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html.

[13] F. Tabakin and B. Juliá-Díaz. QcMpi: A parallel environment for quantum computing. *Computer Physics Communications*, 180(6):948–964, 2009.