Proceeding Series of the Brazilian Society of Computational and Applied Mathematics

Reduction and Decomposition Optimizations in Quantum Computing Simulation Applied to the Shor's Algorithm

Anderson Avila[1]
Renata Reiser[2]
Maurício Pilla[3]
PPGC, CDTEC, UFPEL, Pelotas, Brazil

Abstract. Due to the expansion of transformations and read/write memory states by tensor products in multidimensional quantum applications, the exponential increase in temporal and spatial complexities constitutes one of the main challenges for quantum computing simulations. Simulation of these systems is important in order to develop and test new quantum algorithms. This work presents reduction and decomposition optimizations for the Distributed Geometric Machine environment. By exploring properties as the sparsity of the Identity operator and partiality of dense unitary transformations, better storage and distribution of quantum information are achieved. The main improvements are implemented by decreasing replication and void elements inherited from quantum operators. In the evaluation of this proposal, Shor's algorithm considering $2n+3$ qubits in the order-finding quantum algorithm was simulated up to 25 qubits over CPU, sequentially and in parallel, and over GPU. Results confirm that temporal complexity is reduced. When comparing our implementations running on the same hardware with LIQUi$|\rangle$, academic release version, our new simulator was faster and allowed for the simulation of more qubits.

Key-words. Quantum Computing Simulation, Parallel Processing, GPU, Shor's algorithm

1 Introduction

Quantum Computing (QC) is a new paradigm exploring quantum mechanics to provide unsurpassed parallelism. However, quantum computers are still in their early days and current implementations cannot provide more than a few quantum bits, or qubits, in a reliable way. Until quantum computers become widely available, development and test of new algorithms for these systems may be done by analytical processes or by simulation. Although iterative simulation provides many advantages over analytical processes, simulation of QC over classical computers is a demanding task both in terms of temporal and spatial complexity. As qubits may be represented as matrices and each quantum transformation (QT) as an operation between two matrices, resulting matrices have their sizes greatly increased with the number of qubits.

[1]abdvila@inf.ufpel.edu.br
[2]reiser@inf.ufpel.edu.br
[3]pilla@inf.ufpel.edu.br

In this work, the representation of quantum transformations (QTs) are improved by the clever use of the Identity Operator (Id-operator) and by splitting independent operations. Instead of executing QTs in a single step, they are divided in sub-quantum transformations and only the different values from Id-operators are stored.

Although the number of steps required for simulation is increased, simulation time is greatly reduced even when steps are sequentially executed. Relative speedups of more than 10,000× were achieved for the most demanding transformations when compared to previous works [1], thus making quantum computing simulation more affordable and interesting for a range of algorithms.

In the evaluation of this proposal, simulations of Shor's algorithm based on the quantum circuit described in [2] are performed up to 25 qubits over CPU and GPU using the D-GM Environment [3]. When compared to our implementations running on the same hardware with LIQUi|⟩: Simulation and Compilation of Quantum Algorithms [4], our simulator is faster and allows for the simulation of more qubits than their academic release version.

2 Reducing Simulation Complexity

In previous works [3, 5, 6], we defined processes related to n-dimensional QTs through lower order basic matrices to reduce the memory volume used on simulations due to the exponential growth of their representation using a unique matrix ($2^n \times 2^n$). QT elements required for the computation of a QA are generated during execution time through iterations on those elementary processes, simulating the tensor product behavior on quantum processes and states. However, the computational time spent on those iterations is high, becoming a bottleneck for QA execution. The optimization proposed on this paper is mainly related to the reduction of spatial and temporal complexities associated to QTs by the intelligent use of the Identity operator (Id-operator).

Two distinct approaches were used, which are described in the following subsections.

2.1 Avoiding replication and sparsity inhered from Id-operators

The first optimization explores the behavior associated with the Id-operator and other QTs by tensor products. In such cases, the Id-operator not only replicates the values of other operators but also introduces sparsity in QTs. Thus, it is possible to store only the tensor product expansion among QTs different from the Id-operator, decreasing the spatial complexity by generating a reduced matrix (RM).

Since the RM order is lower than the state dimension, it is not possible to perform the multiplication between matrix/vector as it is usually done to calculate other new amplitudes. This optimization adopts a different approach where information about the calculation of each new amplitude is described as follows:

(i) Each bit of a new amplitude position is related to an operator; the most significant bit to the 1st-qubit operator, the 2nd most significant bit to the 2nd-qubit operator, and so on;

(ii) Bits related to operators diverse from I are considered on-bits;

(iii) The RM line used for the calculation is determined by the concatenation of the on-bits;

(iv) Each element of this line is multiplied by an amplitude of the read state, determined replacing the bits that represent the element column by the on-bits of the new amplitude; and

(v) The new amplitude value is the sum of these products.

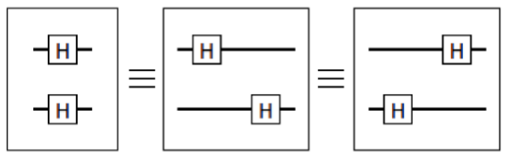
As an illustration, we shall consider a generic operator applied to the first qubit of a 2-dimensional state in Eq.(1). RM elements are described in the form $\mathbf{m_{ij}}$, where i and j are its line and column, respectively. State amplitudes are described in the form a_b, where b is the amplitude position on the state, with its on-bits in red for better visualization.

$$\begin{pmatrix} \mathbf{m_{00}} & \mathbf{m_{01}} \\ \mathbf{m_{10}} & \mathbf{m_{11}} \end{pmatrix} \times \begin{pmatrix} \mathbf{a_{00}} \\ \mathbf{a_{01}} \\ \mathbf{a_{10}} \\ \mathbf{a_{11}} \end{pmatrix} = \begin{pmatrix} \mathbf{a_{00}} \times \mathbf{m_{00}} + \mathbf{a_{10}} \times \mathbf{m_{01}} \\ \mathbf{a_{01}} \times \mathbf{m_{00}} + \mathbf{a_{11}} \times \mathbf{m_{01}} \\ \mathbf{a_{00}} \times \mathbf{m_{10}} + \mathbf{a_{10}} \times \mathbf{m_{11}} \\ \mathbf{a_{01}} \times \mathbf{m_{10}} + \mathbf{a_{11}} \times \mathbf{m_{11}} \end{pmatrix} \quad (1)$$

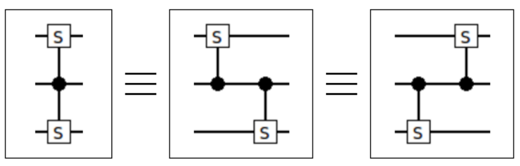
Although this concept optimizes the representation of QTs involving Id-operators, not all QTs have (enough) Id-operators to make possible their representation through a single matrix in memory. Overcoming this limitation, the next optimization considers the decomposition of QTs.

2.2 Decomposing QTs based on *Id*-operators

An n-dimensional QT can be decomposed increasing the number of steps for its computation, allowing to distribute their operators in these steps and control the amount of Id-operators in each one, preserving the behaviour and properties of the QT. Figure 1(a) shows the QT $H \otimes H$ and its decomposition in two steps, $H \otimes I$ and $I \otimes H$, that can be computed in any order. Controlled QTs can also be decomposed conserving the controls associated to the operators, as shown in Figure 1(b). Using this approach, the QT spatial complexity can be reduced limiting the number of Id-operators in decomposed steps, providing representation of each step by a single matrix.



(a) Decomposition of non-controlled QTs



(b) Decomposition of controlled QTs

Figure 1: Decomposition of QTs

3 Improving scalability of QTs

Despite the possibility of modeling QTs with lower spatial complexity using the approach presented on the previous section, the size of read/write memory-states becomes a limit for n-dimensional QTs, since it also increases exponentially (2^n). Once the GPU memory is typically smaller than the main RAM, it is necessary to adopt an approach which provides scalability to multi-qubit QTs.

The Mixed Partial Process (MPP) concept, presented in [1], provides control over the increase in the size of read/write memory-states in the calculation of a QT. Based on that, n-dimensional QTs with more qubits than the limit set by the GPU memory may have their read/write memory-states partitioned into 2^p sub-states, where p indicates the number of qubits beyond the GPU memory limit, making its calculation possible. The number of read sub-states that each write sub-state needs access to perform the calculation of their amplitudes is 2^r, where r is the number of operators affected by the partition. Affected operators refer to the number of operators different from Id-operator present in the first p qubits of the step being computed. Therefore, steps without affected operators only need the correspondent read sub-state, which makes them independent.

4 Implementation

The integrated approach considering the concepts described on the previous sections aims to reduce spatial and temporal complexity in simulation of multi-qubits quantum applications. The QT decomposition for executions on GPUs is divided in two parts: (i) **classification of QTs** in groups, dividing operators non-controlled and with distinct controls; (ii) **QT steps** are formed by operators which belong to the same group and act on consecutive qubits respecting the established limit of operators by step. Affected and non-affected operators can not be part of the same step if the memory was partitioned.

Figure 2 shows a 9-qubit QT considering limits of 3 operators by step and 8 qubits for execution. The QT is firstly divided into 3 groups and from these, in 5 steps. The group 1 is divided into 2 steps, despite having 3 operators in consecutive qubits, since the memory partition affects the first qubit.

After the decomposition, QT steps are recalculated. As seen in Section 3, affected steps are calculated one by one, a kernel call is performed for each combination of write and read sub-states in its computation. Non-affected steps are iteratively executed, partition by partition, reducing the communication between host and GPU, since the GPU memory space with the SUB-QT calculation related to that partition serves as input to the execution of the next one, for the same partition.

The best results in CPUs using the decomposition approach are reached when limits in the number of operators per step of 1 and 2 are considered. Hence, the option of calculating operator by operator, or a limit of 1, was chosen for all simulations by two classes of operators: (i) **Dense** - operators defined by matrices without void elements. These operators do not allow the application of aggressive optimizations; and (ii) **Sparse** - operators with void elements in most positions but in the main diagonal. In these cases, optimizations discarding calculations with void elements may be applied.

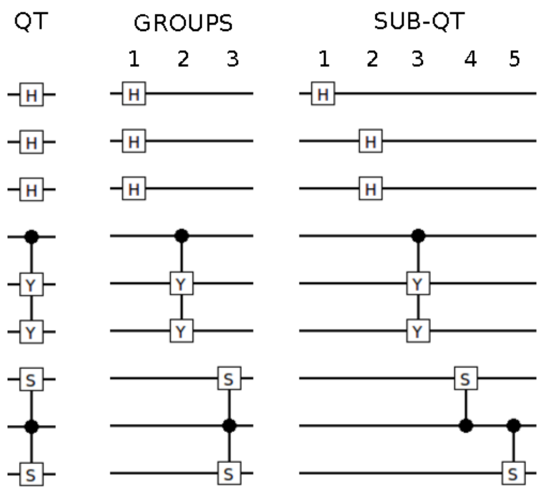


Figure 2: SUB QT

Computing dense operators is as described in Eq. (1). For sparse operators, Eqs. (2) and (3) define how each amplitude can be calculated using a single value from the matrix and state to be calculated, while dense operators require two values of each structure.

$$\begin{pmatrix} \mathbf{m}_{00} & \mathbf{0} \\ \mathbf{0} & \mathbf{m}_{11} \end{pmatrix} \times \begin{pmatrix} \mathbf{a}_{00} \\ \mathbf{a}_{01} \\ \mathbf{a}_{10} \\ \mathbf{a}_{11} \end{pmatrix} = \begin{pmatrix} \mathbf{a}_{00} \times \mathbf{m}_{00} \\ \mathbf{a}_{01} \times \mathbf{m}_{00} \\ \mathbf{a}_{10} \times \mathbf{m}_{11} \\ \mathbf{a}_{11} \times \mathbf{m}_{11} \end{pmatrix} \tag{2}$$

$$\begin{pmatrix} \mathbf{0} & \mathbf{m}_{01} \\ \mathbf{m}_{10} & \mathbf{0} \end{pmatrix} \times \begin{pmatrix} \mathbf{a}_{00} \\ \mathbf{a}_{01} \\ \mathbf{a}_{10} \\ \mathbf{a}_{11} \end{pmatrix} = \begin{pmatrix} \mathbf{a}_{10} \times \mathbf{m}_{01} \\ \mathbf{a}_{11} \times \mathbf{m}_{01} \\ \mathbf{a}_{00} \times \mathbf{m}_{10} \\ \mathbf{a}_{01} \times \mathbf{m}_{10} \end{pmatrix} \tag{3}$$

The QT execution is realized operator by operator. For each one, its type is identified and then the corresponding loop is executed in order to produce the new amplitudes. Parallel execution in CPUs was implemented in OpenMP [7], adding the "parallel for" pragma in loops where the new amplitudes are computed.

5 Results

The main contributions of this work can be evaluated through simulation of Shor's algorithm considering numbers of 6 up to 12 bits, 15 up to 25 qubits.

CPU simulations were sequential and parallel up to 4 Threads, GPU simulations parameters considered were operators limited by step of 1 to 6 and limit of GPU qubits of 26 up to 28 in order to analyze the behavior of the new D-GM algorithm. Tests were performed in a desktop with an Intel Core i7-3770 processor, 8 GB RAM, and an NVidia GTX Titan X GPU. The experiments were executed over Ubuntu Linux version 14.04, 64 bits, and CUDA Toolkit 7.0.

During the simulation, the average simulation time for Shor's algorithm was calculated from 10 executions.

5.1 Shor's Algorithm

Simulation time obtained for Shor's algorithm using LIQ$Ui|\rangle$ and D-GM simulator can bee seen in Table 1. LIQ$Ui|\rangle$ execution time was obtained from the "minutes for running" on the log file generated after executing their built-in Shor with the optimization option in "true". Note that the D-GM simulator presented better results on every number factored for all types of execution. CPU execution shows gain of performance for the parallel execution as the number of threads increase. As expected, GPU executions have the best times of all cases when the quantum part of Shor's algorithm becomes the simulation bottleneck, that would be for numbers with 7 or more bits.

Table 1: Average Simulation Times for Shor's Algorithm, measured in seconds.

			LIQUi|⟩	*D-GM*				
Number	*Bits*	*Qubits*		Seq.	1 Thread	2 Threads	4 Threads	GPU
57	6	15.00	11.01	1.49	1.53	0.92	0.87	0.99
119	7	17.00	47.00	9.26	9.36	5.01	3.41	2.64
253	8	19.00	201.39	57.10	58.02	30.27	17.70	10.17
485	9	21.00	1,166.36	358.89	348.49	239.69	211.64	50.84
1,017	10	23.00	5,905.24	2076.11	2,055.28	1,389.96	1,249.54	280.67
2,045.00	11	25.00	*SL*	*NE*	*NE*	*NE*	*NE*	1,623.87

Seq. - Sequencial. SL - Simulator Limit. NE - Not executed.

6 Conclusion

In this paper, a new approach to reduce quantum computing simulation's temporal and spatial complexity was presented. By using mixed partial processes in conjunction with the Id-operator and sub-quantum transformations, it was possible to simulate a larger number of qubits in a single GPU.

The two approaches proposed in the introduction to improve performance were implemented in the D-GM framework, but are not restricted to this environment.

Shor's simulations showed best results on the D-GM framework when compared to LIQUi$|\rangle$ simulator. For the largest factored number (10 bits), CPU sequential execution was 2.84× faster, CPU parallel execution 4.72× faster with 4 threads, and the GPU execution 21.03× faster.

Ongoing work considers modeling the behavior of agents using fuzzy quantum computations [8], and ranges from applications to develop QAs for quantum information.

In future works, we intend to use the quantum Id-operator optimizations in distributed heterogeneous systems, comprised of multiple computers with GPUs.

Acknowledgments

Project partially founded by grants 309533/2013-9 (PQ-2 CNPq), 309533/2013-9 (PqG FAPERGS), and 448766/2014-0 (MCTI/CNPQ/Universal 14/2014 - B). Student supported by masters grant from CAPES.

References

[1] A. B. de Avila, M. F. Schumalfuss, R. H. S. Reiser, M. L. Pilla, and A. K. Maron. Optimizing Quantum Simulation for Heterogeneous Computing: a Hadamard Transformation Study. *Journal of Physics: Conference Series*, 649(1):012004, 2015.

[2] Stephane Beauregard. Circuit for shor's algorithm using 2n+3 qubits. *Quantum Info. Comput.*, 3(2):175–185, March 2003. ISSN: 1533-7146.

[3] A. B. de Avila, A. K. Maron, R. H. S. Reiser, M. L. Pilla, and A. Yamin. GPU-Aware Distributed Quantum Simulation. In *Symposium on Applied Computing*, pages 860–865, Gyeongju, March 2014. Proc. of the 29th ACM Symposium on Applied Computing (SAC). DOI: 10.1145/2554850.2554892.

[4] D. Wecker and K. M. Svore. LIQ$Ui|\rangle$: A Software Design Architecture and Domain-Specific Language for Quantum Computing, 2014. Available at http://arxiv.org/abs/1402.4467.

[5] A. K. Maron, R. H. S. Reiser, and M. L. Pilla. High-performance quantum computing simulation for the quantum geometric machine model. In *CCGRID 2013 IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 1–8, NY, May 2013. IEEE. DOI: 10.1109/CCGrid.2013.50.

[6] A. K. Maron, R. H. S. Reiser, M. L. Pilla, and A. Yamin. Quantum processes: A new interpretation for quantum transformations in the VPE-qGM environment. In *CLEI 2012*, pages 1–10. IEEE Computer Society - Conference Publishing Services, 2012. DOI: 10.1109/CLEI.2012.6426919.

[7] OpenMP Architecture Review Board. The OpenMP API specification for parallel programming, 2015. Available at http://openmp.org/wp/openmp-specifications.

[8] A. Raghuvanshi and M. Perkowski. Fuzzy quantum circuits to model emotional behaviors of humanoid robots. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010. DOI: 10.1109/CEC.2010.5586038.