UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Thesis

HybriD-GM: Parallel Model for Quantum Computing Targeted to Hybrid Architectures

Anderson Braga de Avila

Pelotas, 2020

Anderson Braga de Avila

HybriD-GM: Parallel Model for Quantum Computing Targeted to Hybrid Architectures

Thesis presented to the Programa de Pós-Graduação em Computação at the Centro de Desenvolvimento Tecnológico of the Universidade Federal de Pelotas, as a partial requirement to obtain the title of Doctor in Computation.

Advisor: Prof. Dra. Renata Hax Sander Reiser
Coadvisor: Prof. Dr. Maurício Lima Pilla
Collaborator: Prof. Dr. Adenauer Yamin

Pelotas, 2020

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

A111h Avila, Anderson Braga de

HybriD-GM : parallel model for quantum computing targeted to hybrid architectures / Anderson Braga de Avila ; Renata Hax Sander Reiser, orientadora ; Mauricio Lima Pilla, Adenauer Correa Yamin, coorientadores. — Pelotas, 2020.

96 f.

Tese (Doutorado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2020.

1. Quantum computing. 2. Hybrid simulation. 3. GPU. I. Reiser, Renata Hax Sander, orient. II. Pilla, Mauricio Lima, coorient. III. Yamin, Adenauer Correa, coorient. IV. Título.

CDD : 005

Elaborada por Maria Inez Figueiredo Figas Machado CRB: 10/1612

Anderson Braga de Avila

HybriD-GM: Parallel Model for Quantum Computing Targeted to Hybrid Architectures

Thesis approved, as a partial requirement, to obtain the degree of Doctor in Computation, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Defense Date: 8 July 2020

Examining Board:

Prof. Dra. Renata Hax Sander Reiser (advisor)
Doctor in Computer Science by the Universidade Federal do Rio Grande do Sul.

Prof. Dr. André Rauber Du Bois.
Doctor in Computation by the Heriot-Watt University.

Prof. Dr. Carlos Amaral Hölbig.
Doctor in Computation by the Universidade Federal do Rio Grande do Sul.

Prof. Dr. Luiz Gustavo Leão Fernandes.
Doctor in Computing by the Institut National Polytechnique de Grenoble.

RESUMO

AVILA, Anderson Braga de. **HybriD-GM: Parallel Model for Quantum Computing Targeted to Hybrid Architectures**. Orientador: Renata Hax Sander Reiser. 2020. 96 f. Tese (Doutorado em Computation) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2020.

Uma grande quantidade de pesquisas científicas e desenvolvimentos tecnológicos ainda depende de simulações de computação quântica, uma vez que os computadores quânticos ainda são limitados por implementações físicas. Atualmente, o desenvolvimento de algoritmos de computação quântica tem sido realizado por procedimentos analíticos ou de simulação enquanto os computadores quânticos não estão amplamente disponíveis. Embora a simulação de computação quântica seja paralela por natureza, a complexidade espacial e temporal são os maiores riscos de desempenho, pois os estados quânticos e as transformações quânticas aumentam exponencialmente com o número de qubits simulados. Esta proposta contribui desde a concepção até a consolidação do modelo HybriD-GM, bem como introduz a extensão do ambiente D-GM, proporcionando execuções paralelas eficientes para computação quântica, que neste trabalho é voltada para arquiteturas híbridas, considerando tanto CPU quanto GPU. O modelo HybriD-GM explora as potencialidades da Computação de Alto Desempenho, fornecendo funcionalidades e explorando operadores de projeção que atuam em estruturas quânticas, estados e transformações, para manipular a granularidade e distribuição de cálculos. Neste contexto, a distribuição das computações é baseada em estruturas de dados em árvore, onde os nós intermediários e finais correspondentes às camadas de projeção e execução são configurados para otimizar os recursos de hardware. Simulações dos algoritmos de Shor e Grover foram realizadas a fim de avaliar o modelo HybriD-GM, e os resultados alcançaram melhorias de desempenho significativas para execuções em CPU e GPU. Quando comparados com a versão anterior do D-GM, eles apresentaram speedups de $21\times$ e $9,5\times$ para simulações paralelas em CPU e de $61,9\times$ e $38,32\times$ para simulações em GPU. Além disso, em relação aos simuladores LIQUi$|\rangle$ e ProjectQ, a simulação paralela com 23 qubits do Shor foi $4,64\times$ mais rápida e do Grover foi $32\times$ mais rápida. Os resultados das simulações híbridas mostraram que é possível aumentar o desempenho para algumas classes de algoritmos, melhorando o desempenho do algoritmo de Grover em até $3,18\times$ em comparação ao uso apenas da abordagem GPU.

Palavras-chave: Keyword-one. Keyword-two. Keyword-three. Keyword-four.

ABSTRACT

AVILA, Anderson Braga de. **HybriD-GM: Parallel Model for Quantum Computing Targeted to Hybrid Architectures**. Advisor: Renata Hax Sander Reiser. 2020. 96 f. Thesis (Doctorate in) – , Universidade Federal de Pelotas, Pelotas, 2020.

HybriD-GM: Parallel Model for Quantum Computing Targeted to Hybrid Architectures

A huge amount of scientific research and technological developments still depends on quantum computing simulations, since quantum computers still limited by physical implementations. Currently, the development of quantum computing algorithms has been carried out by analytic or simulation procedures while quantum computers are not widely available. Although quantum computing simulation is parallel by nature, spatial and temporal complexity are major performance hazards, as quantum states and quantum transformations increase exponentially with the number of qubits simulated. This proposal contributes from the conception to the consolidation of the HybriD-GM model, as well as introduces the extension of D-GM environment, providing efficient parallel executions for quantum computing, which in this work is targeted to hybrid architectures, considering both CPU and GPU architectures. The HybriD-GM model explores the potentialities of High Performance Computing, providing functionalities and exploring projection operators which are acting on quantum structures, state and transformations, to manipulate the granularity and distribution of computations. In this context, the distribution of computations are based on tree data-structures, where intermediate and final nodes corresponding to projection and execution layers are configured to optimize the hardware resources. Simulations of Shor's and Grover's algorithms were performed in order to evaluate the HybriD-GM model, and the results achieved significant performance improvements for executions in both CPU and GPU. When compared to the D-GM previous version, they presented speedups of $21\times$ and $9.5\times$ for parallel CPU simulations and of $61.9\times$ and $38.32\times$ for GPU simulations. In addition, related to LIQUi$|\rangle$ and ProjectQ simulators, the parallel simulation with 23 qubits was $4.64\times$ faster for Shor's and $32\times$ faster for Grover's. Results for hybrid simulations showed that is possible to increase performance for some classes of algorithms, improving the performance of Grover's algorithm up to $3.18\times$ in comparison to only using GPU approach.

Keywords: Quantum Computing. Hybrid Simulation. GPU.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

CPU	*Central Processing Unit*
D-GM	*Distributed Geometric Machine*
DMA	*Direct Access Memory*
GPU	*Graphical Process Unit*
HPC	*High-Performance Computing*
MPP	*Mixed Partial Process*
QA	*Quantum Algorithm*
QC	*Quantum Computing*
QFT	*Quantum Fourier Transform*
qGM	*Quantum Geometric Machine*
QM	*Quantum Mechanics*
QS	*Quantum State*
QT	*Quantum Transformation*
RB	*Read Basis*
RM	*Reduced Matrix*
VPE-qGM	*Visual Programming Environment for the qGM Model*
VIRD-GM	*Virtual Distributed Geometric Machine*
WB	*Write Basis*

CONTENTS

1 INTRODUCTION

Quantum computing (QC) promises to solve some problems that in classical computing would be impractical. Thus, the imminence of quantum supremacy is a reality, meaning that a quantum computer can perform a calculation task which would be intractable on a classical supercomputer (AARONSON; CHEN, 2017).

Several quantum algorithms (ordering, prime factorization, modular exponentiation) have already been developed leading to significant improvements when compared to the best known classical algorithms (AHARONOV; LANDAU; MAKOWSKY, 2007).

Consequently, quantum computing has been considered potentially significant in many areas such as reversible logic, quantum cryptography, information processing, communication, data coding methods and so many other (BISWAS et al., 2017).

However, quantum computers are still in their early days. Although the construction of quantum hardware is still a technological challenge, restricted to laboratories and huge companies, it might become a feasible technology in the near future.

Until quantum computers become widely available, the development and testing of quantum algorithms may be done simulating quantum computing on classical computers. Although specific classes of quantum algorithms can be efficiently simulated (VIDAL, 2003), this is not the case for most quantum algorithms.

This research work contributes to the development of a computational model supporting efficient quantum computing simulation targeted to classical hybrid architectures.

1.1 Main Motivations

In the next subsections, the main motivations in the development of this research are characterized, organized under the perspective of the relevance of quantum computing, quantum algorithms and quantum computing simulation.

1.1.1 Relevance of Quantum Computing

There is an increasing interest in devices taking advantages from resources related to atomic/nano scale of quantum mechanical effects, and as consequence, in quantum computing. The main reasons compelling evidences for the relevance of quantum computation are briefly reported in the following.

(i) In the sense of complexity theory, the quantum model of Turing machine is more powerful than the classical one, meaning that a simulation of a finite-size quantum system is a more complex computational task when performed by classical means. In addition, it can be performed without an exponential slowdown (NIELSEN; CHUANG, 2000).

(ii) Highlighting the universality property, described in the complexity theory as the ability of a computer to simulate any other computer in polynomial time. As a mathematical quantum model, the universal quantum Turing machine enables to overall the Hamiltonian systems with local and time of computations. Thus, in such context, several experiments have been shown that quantum computation can be universal, not only by taking single qubits but also by considering quantum registers systems. Such approach seems more useful from a computational viewpoint, as effectively performed by the Fredkin-Toffoli gate (FEYMANN, 1982; BARENCO et al., 1995).

(iii) In the current computer technology, the heat production problem is one of the factors limiting speed and size reduction of digital computers. Overcoming such problem and contributing with this relevant research question, new technological developments provided by quantum devices are based on an exceptional characterization, meaning that the energy consumption in the performance of quantum computational tasks can be minimized (BENNETT, 1989). Thus, during a computation, an ideally closed quantum system preserves reversibility, consequently no energy is consumed, and as a consequence there is no theoretical limit to the computational speed, due to the thermodynamic foundations.

(iv) Another important result illustrating the power of quantum computation is related to potential of quantum algorithms. See, e.g., in (GROVER, 1996a), the Grover's algorithm performing a search for an element in a database and solving the problem in expected time which grows proportional to the square root of the number of entries (CAFARO; MANCINI, 2015). Moreover, the Shor's algorithm, factorizing numbers in polynomial time with respect to a quantum probabilistic Turing machine (BENNETT; SHOR, 1998).

In addition, quantum technological advancement impacts are not restricted to accelerate the connected world on the globalization, demanding by easy and fast commu-

nication as well as providing unprecedented changes on our digital era. It also rapidly has been promoting innovative technologies, leading to steady advances, directly reflecting in scientific research and emerging technologies. See, for instance, in quantum nano science and biomedical sciences, including food and pharmako biotechnologies, nanoproducts and bioprocessing industries. And, Natural Computing and Neurocomputing supporting physical systems can also reach unsurpassed parallelism (VERDON et al., 2019).

1.1.2 Relevance of Quantum Algorithms

Quantum computing impacts on the weakening of traditional hardware scaling laws, improving and stimulating industrial, supporting commercial and security new interests, promoting abroad approaches for scientific and technological developments.

By explore the quantum-mechanical effects as superposition, entanglement, and quantum tunneling, quantum computers perform many executions and much more efficiently tasks when compared to traditional digital computing.

(i) Quantum superposition, referring not only to the fact that each qubit is indeed in both basis states simultaneously, but also to the distribution probability, proportional to the quantum state amplitude. And thus, in a quantum register each qubit can be in two places at once. Such implicitly parallel nature of quantum mechanic's phenomena provide solutions to problems whose computational complexity makes unsuitable for classical computers (AARONSON; CHEN, 2017).

(ii) Quantum entanglement is presented as inherent property correlating two quantum particles not only in one direction but also in possible all directions. Entanglement provides an effect that is not easily simulated classically, since it has no classical contra part and hence, determining a potential quantum advantage from classical computers (JEFFERY; MAGNIEZ; WOLF, 2014). Thus, two particles created together are capable of interacting immediately even when spatially separated by any distance. And then, by subjecting one particle to one particular effect, the other particle entangled will react instantly (NIELSEN; CHUANG, 2000).

(iii) Quantum tunneling, often explaining in terms of the Heisenberg uncertainty principle, is described as a property allowing that, in an exact location, atomic particles break rules of classical mechanics and move in the space without passing over the potential energy barrier. New physical limits to the size of transistors can be achieved, meaning that microprocessors can be tunneling-projected, due to atomic particles enabling to tunnel past them even when transistors are too small (DENCHEV et al., 2016) (TRIEU, 2009).

The development of new computational applications, which can be supported by the

quantum computing paradigm, passes trough the understanding their main properties, The design of new quantum algorithms can be improved, by extracting the potential of such inherent properties of quantum mechanics. This strategy provides support to advances in research areas. In particular, this work is focused on simulation of inner quantum parallel via high performance computing (HPC), simulating a quantum parallelism related to superposition of multiple-qubit applications.

1.1.3 Relevance of Quantum Computing Simulation

This research promote a simulation of quantum algorithms by classical technologies mainly motivated by the arguments described below.

(i) Although the construction of quantum hardware is still a technological challenge, several quantum algorithms (ordering, prime factorization, modular exponentiation) have already been developed showing significant improvements when compared to the best known classical algorithms (BISWAS et al., 2017).

(ii) Improvements toward the better understanding of many classes of quantum algorithm and their behaviors might be the first step taking an approach to solving huge computational problems, in emerging computing areas such as artificial intelligence and robotics, huge big data analysis, bioinformatics, and also cyber security (SILVA; OLIVEIRA; LUDERMIR, 2016).

(iii) The new computation techniques to simulate quantum algorithms has made interesting progress, integrating relevant areas as quantum neural computing and appearing as a new paradigm built upon the combination of artificial intelligence and neural computation. Simulations modeling brain functionality and interpreting the behavior of a quantum algorithm (QA) have contributed to creating new systems for information processing, including new solutions for classically intractable problems(LU; JUANG, 2011).

(iv) Simulations of a generic quantum algorithm on classical computers is a demanding task both regarding temporal and spatial complexity. As quantum states (QS) may be represented as vectors and quantum transformations (QT) as matrices, register sizes increase exponentially with the number of qubits of the application (LI; YANG; LI, 2015; YANG et al., 2015).

(v) New methodologies dealing with quantum computational models are grounded on the understanding of power and complexity of QC by resorting to the known capacity of learning about the main characteristic of good results of quantum algorithms.

(vi) Until quantum computers become broadly available, the development and testing of quantum algorithms may be done by simulation, providing many advantages over analytical processes such as detailed study of their behaviour without supporting of a quantum physical environment;

1.2 Main research questions

As main research questions, this work considers the following:

- How to explore the potentialities of High Performance Computing (HPC) in order to provide support and optimization of quantum computing simulation?

Computer simulation has long been accepted as consolidate methodology in many branches of science and engineering. In addition, classical computers can simulate the abstract model of an ideal quantum computer providing fundamental and theoretical results and, most relevantly, they can also simulate the physical behaviour of a quantum algorithm exploiting its potential to new technological applications.

Classical computers (Desktops) have been used to simulate quantum algorithms which are relatively small (up to 30 qubits with 16 GB RAM) but are significantly larger than the experimental models available and restricted to research laboratories.

By making use of simulations carried out on present-day (but not necessarily super) computers and massive parallel architectures, quantum computing simulation is established as a research area for HPC, trying to explore optimization techniques to improve performance.

- How to explore computational strategies in order to control and manage the huge data volume related to memory and processes on simulations, due to the exponential growth of their representation of quantum states and quantum transformations?

In the context of quantum computing simulation, quantum transformations (QT) are modeled as $2^n \times 2^n$ matrices quantum and states (QS) as 2^n vectors, with sizes increasing exponentially with the number of qubits (n) in a quantum application. Therefore, it involves a lot of computational resources, not only to store the quantum structures, but also to perform the quantum state evolution by matrix-vector linear algebraic operations.

Considering the use of mathematical constructors as composition and projection operators can help to control the computation granularity and explore optimizations properties such as the sparsity of control operators and the partiality of unitary transformations. Consequently, programming in multidimensional database can deal with queries in parallel which can improve storage and distribution of quantum processes and states.

1.3 Proposal and main objectives

This proposal aims the development of improvements on performance of quantum computing simulation research area, in order to better assist the study of quantum algorithms. The related study and research efforts were organized in two moments, each with its specific objectives.

(i) the HybriD-GM model, which is conceived as a computational methodology:

- optimizing resources and enabling scalability even with upcoming CPU and/or GPU hardwares;
- considering composition and projection operators acting on quantum structures used to structure a model for CPU and/or GPU architectures as hardware independent;
- customizing a uniform programming based on the HybriD-GM Model, by explore not only the potential of different processing units, but also integrate them face the differences in their architectures, programming models and performance.

(ii) the HybriD-GM framework supporting an extension of the D-GM framework:

- assisting simulation of quantum applications which use memory superposition/entanglement related to quantum overlap/parallel processes.

The research project are developed integrating three strategical objectives:

(i) Study and analysis of properties and main characteristics demanding by projection operators acting on quantum structures:

- Projecting n-dimensional basis of quantum states and n^2-dimensional basis of quantum processes in a hybrid model structured;
- studying the D-GM environment (AVILA et al., 2015, 2016a), functionalities and thier code structures;
- Revising literature based on frameworks for general purpose simulation of quantum computing algorithms.

(ii) Structuring and development of the HybriD-GM model congregating CPU and/or GPU architectures considering a hybrid approach;

- Extending the D-GM functionalities and code structures in order to support the approach of the proposed model.

(iii) Validation and evaluation of the HybriD-GM model in the extended D-GM environment;

- Developing case studies based on the simulations of quantum algorithms.

The proposal of a hybrid software architecture for quantum computing is conceived as independent of hardware, where the computations can be performed from regular desktops for sequential, multiple GPU or hybrid simulations of multi-qubits quantum applications.

1.4 Thesis Outline

The text is organized as follows. Firstly, quantum computing foundations are described in Chapter 2. In Chapter 3, the previous structure of the D-GM environment, mainly reported functionalities incorporated at the beginning of the thesis proposal is presented. Related works are reported in Chapter 4, describing their approaches and implementations. The conceptual model of the HybriD-GM proposal is described in Chapter 5 followed by the description of its architecture model in Chapter 6. Chapter 7 reports the proposal evaluation considering as case study the factorization and sorted quantum algorithms, named Shor's algorithm and Grover's algorithms. Finally, the Conclusion presents main results and contributions achieved with this work, also including further work.

2 QUANTUM COMPUTING FOUNDATIONS

The current chapter considers the main concepts of quantum computing and quantum mechanics (QM), underlying the results discussed in this work.

2.1 Basic Concepts

The conceptual presentation of quantum computing has grounded the quantum systems in very actual applied research areas connected to computer science. See, e.g., in (DO et al., 2020), proposing a model for compilation of a quantum algorithm for graph coloring, and in (STOLLENWERK et al., 2020), promoting a model for air traffic management in computer engineering, both structured based on foundations from quantum computing.

Particle manipulation at the atomic/subatomic scale comprises a task of high complexity, since in these situations, particles (photons, electrons and other individual structures of the same scale) exhibit unusual behaviors which are not completely defined by the laws of classical physics. And, quantum mechanics is the area of physics that studies such behavior, presenting theories which precisely define the phenomena occurring on a atomic/subatomic scale.

Besides the quantum-mechanical effects as superposition, entanglement, and quantum tunneling as briefly described in introduction, the QM also contemplates the interpretation of the inference phenomenon of wave-particle duality, exemplified by the Double-Slot Experiment (YOUNG, 2017), in which an atomic particle is capable of presenting two distinct behaviors: (i) the wave, where its trajectory is described by a superposition of waves; (ii) the corpuscular, with a well defined trajectory. A quantum particle is defined by a wave function, consisting of a wide variety of possible states. However, by using any device to measure the state of that particle, its wave function collapses into one of their possibilities, behaving from that point on as a particle with a well-defined state (PESSOA, 2003).

From this behavior emerges Heisenberg's uncertainty principle, which, in short, establishes that by measuring the state of a quantum object, its state will be instanta-

neously altered, leaving no quantum properties present. Note, therefore, the impossibility of determining the trajectory of a quantum particle, since, when measuring the corresponding position, its state collapses, preventing the measurement of its velocity (PORTUGAL; LAVOR; MACULAN, 2004).

2.1.1 Postulates of Quantum Mechanics

These initial considerations ground the description of the behavior of quantum systems, which are mathematically specified through four postulates defined by QM, allowing the analogy with physical systems:

- **State Space**
  In QC, the qubit is the basic unit of information, mathematically represented by a unit vector in a Hilbert space $\mathbb{C}^2$ with two basis-elements in superposition. Accordingly, the vectors of $\mathcal{H}$ are indicated by $|\psi\rangle$, $|\varphi\rangle$,... ; while the basis-elements are denoted by $|0\rangle$, $|1\rangle$. Thus a qubit $|\psi\rangle$ will have the form:

  $$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

  where the complex coefficients α, β are called amplitudes such that:

  $$|\alpha\rangle^2 + |\beta\rangle^2 = 1.$$

  The information related to superposition-states representing involves a certain uncertainty degree. In particular, the number $|\alpha\rangle^2$ correspond to the probability-value of the information described by the basic state $|\alpha\rangle$; while $|\beta\rangle^2$ correspond to the probability-value of the information described by the basic state $|\beta\rangle$.

- **Composite Systems**
  A vector space of a n-qubit system is represented by a unit vector in the n-fold tensor product Hilbert space $\otimes^n\mathbb{C}^2 := \underbrace{\mathbb{C}^2 \otimes ... \otimes \mathbb{C}^2}_{n \text{ times}}$ (where $\otimes^1\mathbb{C}^2 := \mathbb{C}^2$). In a quantum system with two qubits, $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ and $|\varphi\rangle = \gamma|0\rangle + \delta|1\rangle$, the corresponding state space is composed by tensor product

  $$|\psi\rangle \otimes |\varphi\rangle = \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle. \quad (1)$$

- **System Evolution**
  The state transition in a quantum system is done by unitary quantum transformations, associated with orthonormal matrices of order $2^n \times 2^n$, where n is the number of the qubits of the transformation.

- **Quantum Measurements** The process of extracting information from a quantum

system, identifying the most probable current state, is studied from the measurement operation. The projective quantum measure considers:both operators: (i) projection operators, which apply different filtering processes over the space of states, and (ii) normalization operator. The final state of the system depends on the projection operator executed.

Due a reflection on the four postulates, any isolated physical system can be mathematically interpreted as a Hilbert space, a complex vector space with an inner product known as the state space. Such system is completely described by its state vector, a unit vector in the state space.

The evolution of quantum systems is deterministic, modeled by unitary quantum gates corresponding to unitary linear operators on the Hilbert space, which is restricted to normalized vectors of the state space for quantum systems. So it can be said that quantum algorithms are represented by the expression of a unit operator, projections and/or combinations of unit operators (NIELSEN; CHUANG, 2000).

Now, ome remark commenting limits and restrictions of quantum computers are presented, for details see (KAYE; LAFLAMME; MOSCA, 2007).

There is no computer with a local Hamiltonian that has a similarly striking advantage in the sense of complexity theory (DEUTSCH, 1985). However, in contrast to computation by classical means, by the quantum state superpositions, there is no limit for the parallelism of quantum computations (DEUTSCH, 1985). And, if there is no restriction on the number of quantum processors, every Turing computable function can be evaluated in an arbitrarily short time (KNILL; LAFLAMME, 1998).

In fact, the quantum systems are ideally closed, meaning that there are no interactions with the outside world except when the initial state is prepared and at the time of the final measurement, in order to read out the result. So, systems characterized by only spatially local energy interactions are probably simpler to realize since the communication pathways are short.

The main advantage of such systems is that there is no energy dissipation, which is one of the most severe factors limiting the speed of computing. And also, it is possible to describe the entire dynamics of a closed system by the Schrödinger equation.

2.1.2 Quantum Circuit Model

The most recurrent model for describing quantum applications is the quantum circuit model (NIELSEN; CHUANG, 2000). This representation is one of the most fundamental of the QC, being characterized by an intuitive graphical notation that refers to the model of digital circuit used in classical computing.

Quantum circuits comprise synchronizations and compositions of unitary quantum gates and measurement operations, modeling any type of quantum algorithm, as

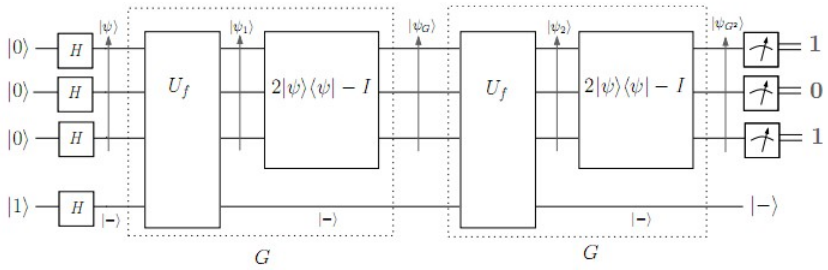


Figure 1 – Typical quantum circuit.

shown in Figure 1. Some conventions are adopted aiming at a homogeneous description of quantum algorithms, being described as follows:

- Horizontal Lines: each line represents one qubit of the system, and the corresponding time evolution occurs from left to right;
- Vertical Lines: indicate that a given quantum transformation acts on the qubits connected through this line;
- Control: represented by a circle on the line of a qubit. If the circle is closed, it indicates that the state $|1\rangle$ of the qubit is considered; if it is opened, the state $|0\rangle$ is considered;
- Quantum gates: unitary transformations that manipulate the qubit on which they are applied;
- Measurement: at the end of each line of the circuit can appear a measurement operation, determining the classic state of the corresponding qubit.

2.1.3 Quantum Transformations

Unitary quantum transformations are the operations responsible for manipulating the amplitudes associated with the states of a quantum system. These transformations are defined by unitary square matrices of order 2^n, where n represents the number of qubits on which the transformation will act. The main basic quantum transformations are described in the sequence.

- ***Hadamard***, which is a transformation generating the superposition of a multidimensional states. Its matrix definition is:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2}$$

The application of H on the state vector of the generic qubit $|\psi\rangle$, defined in the first postulate of QM, results in:

$$H|\psi\rangle = \frac{1}{\sqrt{2}}(\alpha + \beta, \alpha - \beta)^t \tag{3}$$

- ***Pauly X***, which is equivalent to the classical gate *NOT*, inverting the amplitudes of a qubit state. The corresponding matrix definition and application on the state vector of $|\psi\rangle$ is represented by:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} \tag{4}$$

- ***Pauly Y***, a transformation applied to $|\psi\rangle$ resulting on $Y|\psi\rangle = -i\alpha|0\rangle + i\beta|1\rangle$. The correspondent matrix definition is:

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{5}$$

- ***Pauly Z***, the operation matrix of this transformation is given by:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{6}$$

  Its function is to performing the inversion of the qubit phase, transforming the state $(\alpha, \beta)^t$ vector into $(\alpha, -\beta)^t$.

- ***Phase (*S*)***, introducing a relative phase, which means, taking the qubit $|\psi\rangle$ to the state $S|\psi\rangle = \alpha|0\rangle + i\beta|1\rangle$, where the amplitude $|0\rangle$ remains unchanged, while the amplitude $|1\rangle$ differs by a phase factor equal to *i*. The matrix corresponding to the Phase gate is described by:

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \tag{7}$$

- $\pi/8$, which is the quantum transformation associated with the following unitary matrix:

$$T = \begin{pmatrix} 1 & 0 \\ 0 & exp(i\pi/4) \end{pmatrix} \tag{8}$$

  By the application of T to the state vector $|\psi\rangle$, it results on $(\alpha, exp(i\pi/4)\beta)^t$.

In order to exemplify the exponential increase of quantum transformations applied to multiple qubits systems, we first consider the *Hadamard* (H) transformation applied

to a 1-*qubit* system. The following matrix representation describes such scenario:

$$H|\psi\rangle \equiv \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \times \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} \alpha + \beta \\ \alpha - \beta \end{pmatrix} \tag{9}$$

Considering now the simultaneous application of H to a 2-qubits system, we have the following matrix structure

$$H^{\otimes 2} \equiv \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}, \tag{10}$$

which is obtained from the tensor product operation ($\otimes$) between the correspondent basic matrices compounding the quantum system. The $\otimes$ operator generates an exponential increase in the amount of elements in the resulting matrix.

The application of $H^{\otimes 2}$ given in the quantum system, Eq. 10, to the state space defined in Eq. 1, results on the following vector structure

$$\frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \times \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} = \frac{1}{2} \begin{pmatrix} \alpha + \beta + \gamma + \delta \\ \alpha - \beta + \gamma - \delta \\ \alpha + \beta - \gamma - \delta \\ \alpha - \beta - \gamma + \delta \end{pmatrix} \tag{11}$$

In addition, in order to manipulate qubits from a multidimensional quantum state/-transformation, we can make use of controlled transformations.

By applying such operations, we can modify the state of one or more qubits considering the current state. Among the controlled transformations, the following stands out two operators considered in the algorithms simulated in this work.

Controled NOT (CNOT) Transformation

The bidimensional quantum transformation *CNOT* receives 2 *qubits*, $|\psi\rangle$ and $|\varphi\rangle$, as input and applies the *NOT* (*Pauly X*) to one of them (target qubit), by considering the current state of the other (control qubit). As an example, lets consider first qubit with control in $|1\rangle$ and the second qubit as the target, then we have the following system:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \\ \delta \\ \gamma \end{pmatrix} \tag{12}$$

Toffoli Transformation

In the three dimensional controlled transformation *Toffoli*, the transformation *Pauly*

X is applied to a *qubit* when the state of the other two qubits are $|1\rangle$. Its graphical representation in the quantum circuit model is exemplified by Figure 2.

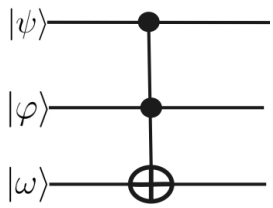


Figure 2 – Toffoli gate in the quantum circuit model.

Controlled-U Transformation

Generic controlled transformations (*Controlled-U*) (NIELSEN; CHUANG, 2000) can be defined in order to use various configurations of control qubits and apply any unitary transformation *U* to the target qubit(s). See, an illustration in Figure 3.

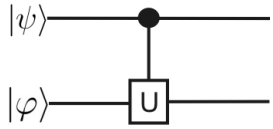


Figure 3 – Controlled-U gate in the quantum circuit model.

Quantum algorithms are constructed from the sequential and synchronous composition of quantum transformations.

2.1.4 Quantum Measurements

In order to obtain information from a quantum system, it is necessary to apply measurement operators, defined by a set of linear operators M_m called projections, with index m refers to the possible measurement results, related to an m-dimensional of classical basis and corresponding to probability measure.

Moreover, the $M_m^\dagger$ operator is called the adjoint operator or Hermitian adjoint operator of M_m (NIELSEN; CHUANG, 2000; KNILL; NIELSEN, 2000).

Let $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, with $\alpha, \beta \neq 0$, be a state of a one-dimensional quantum system. Thus, immediately before the measurement, the probability of an outcome occurrence is given by

$$p_m(|\psi\rangle) = \frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}} \tag{13}$$

verifying the complementary relation

$$\sum_m M_m^\dagger M_m = I$$

and, considering the Hermitian operators:

$$M_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = M_0^\dagger$$

$$M_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = M_1^\dagger$$

which are not reverse operators, as can one can easily observe in the following:

$$M_0^2 = M_0^2,\ M_1^2 = M_1^2 \text{ and } M_0^\dagger M_0 + M_1^\dagger M_1 = I_2 = M_0 + M_1.$$

Thus, when a measure is performed on *qubit* $|\psi\rangle$, the probability of observing $|0\rangle$ and $|1\rangle$ are, respectively, given by the following expressions:

$$\begin{aligned} p_0(|\psi\rangle) = &\ \langle\phi|M_0^\dagger M_0|\phi\rangle = \langle\phi|M_0|\phi\rangle = |\alpha|^2; \\ p_1(|\psi\rangle) = &\ \langle\phi|M_1^\dagger M_1|\phi\rangle = \langle\phi|M_1|\phi\rangle = |\beta|^2. \end{aligned}$$

And, after the measuring process, the quantum state $|\psi\rangle$ has $|\alpha|^2$ as the probability to be in the state $|0\rangle$ and $|\beta|^2$ as the probability to be in the state $|1\rangle$.

In multidimensional quantum systems, for a measure performed on an n-dimensional quantum state $|\psi\rangle$, the M_m^n-projection operators in Eq.(13) are considered.

2.2 Shor's Algorithm for Quantum Factoring

Consider the Prime Factorization Problem: "*given a composite odd positive integer N (typically several hundred digits long), how can one find its prime factors*?"

It is well known that factoring N can be reduced to the task of choosing at random an integer a relatively prime to N, and then determining its module N multiplicative order r. This approach to factorization enabled Shor to construct his factoring algorithm for quantum computing (SHOR, 1994, 1995). It consists of a classical pre-processing related to a quantum algorithm for order-finding, and a classical post-processing (WECKER; SVORE, 2014).

The only usage of quantum phenomena in Shor's algorithm is to find the order of a modulo N, where N is an n-bit integer that we want to factor. Additionally, the order r of a modulo N is the least positive integer such that $a^r \equiv 1(mod\ N)$.

Given a number N to factor, see the algorithm steps (BEAUREGARD, 2003):

1. If N is even, return the factor 2.

2. Classically determine if $N = pq$ for $p \geq 1$ and $q \geq 2$ and if so return the factor p (this can be done in polynomial time).

3. Choose a random number a such that $1 < a \leq N - 1$. Using Euclid's algorithm, determine if $gcd(a, N) > 1$ and if so, return the factor $gcd(a, N)$.

4. Use the order-finding quantum algorithm to find the order r of a module N .

5. If r is odd or r is even but $a^{r/2} = -1(mod\ N)$, then go to step (3). Otherwise, compute $gcd(a^{r/2} - 1, N)$ and $gcd(a^{r/2} + 1, N)$. Test to see if one of these is a non-trivial factor of N, and return the factor if so.

Due to its potential exponential advantage over conventional algorithms and its application to breaking public key cryptography it is the most celebrated of quantum algorithms (THAKER et al., 2006). It is primarily composed of two parts, the modular exponentiation, whose execution is dominated by Toffoli quantum gates and the quantum fourier transform but requiring huge communication between data qubits (SHOR, 1999).

The order-finding quantum algorithm used in this paper is the one described in (BEAUREGARD, 2003), using $2n + 3$ qubits in a n-bit integer factorization.

2.3 Grover's Algorithm for Quantum Search

A search algorithm considers the problem to find an element, satisfying a known condition, in an unsorted or unstructured database with N elements. Classically, when each element is tested at a time, until hit the one searched for, it takes average $\frac{N}{2}$ attempts or N in the worst case, therefore the complexity is $O(N)$. Lov Kumar Grover devised in 1996 a quantum algorithm which would do this search in time complexity $O(\sqrt{N})$, having a quadratic speedup over the classical one (GROVER, 1996b). Instead of checking possibilities one by one, a uniform superposition over all possibilities are created, which repeatedly and destructively interferes with states that are not solutions.

In contrast with classical search algorithms, Grover's algorithm does not search through lists but through function inputs. By taking a function f, in a high probability, it searches through the implicit list ($L = \{0, 1, ..., N - 1\}$) of possible inputs of such function, returning the single input (i_0) which causes the function to return true (1), or false (0) otherwise.

Quantum searching is a relevant theoretical research area, with many applications in scientific computing. Since (BRASSARD; HØYER; TAPP, 1998), extensions of Grover's quantum searching algorithm are considered, extending the Grover iteration in the light of a concept called amplitude amplification. In (LU; JUANG, 2011), the evolutionary quantum-inspired space search algorithm (QSSA) is introduced for solving numerical optimization problems, which may be applied to a series of numerical optimization problems. More recently, in (CAFARO; MANCINI, 2015), information

related to a geometric characterization of Grover's quantum search algorithm is discussed, showing that the quantum searching problem can be recast in an information geometric framework.

2.4 Summarizing

This chapter presents basic concepts of QC, reporting the postulates of QM, including the quantum computing circuit model and main quantum operators as the Hadamard quantum transformations and controlled transformations, such as Controlled Not, Controlled-U and Toffoli transformations.

These concepts are theoretical foundations to understanding the main characteristics of the quantum algorithms as Grover's quantum search and Shor's algorithm for quantum factoring, and also, the simulation environment, structured by the D-GM framework which is presented in the next chapter.

3 D-GM FRAMEWORK

The quantum approach of Distributed Geometric Machine Model (D-GM) is based on the subcategory of coherence spaces and linear functions (GIRARD; LAFONT; TAYLOR, 1989), conceived as a domain-theoretical model (DTM) for interpretation of quantum computing (SCOTT, 1967). Over such structure it is possible to guide the ordered construction of quantum processes and quantum memory through the Scott-style inverse colimit construction (AMARAL; REISER; COSTA, 2009).

A complete structure for supporting of quantum algorithms is consolidated in the D-GM framework, and graphical interfaces for modeling/simulating applications were incorporated to its architectural structure. The D-GM environment has been in constant evolution, which development leaded to support sequential and parallel simulations approaches, using multicore CPU or GPU, by considering strategies based on partiality of quantum states and processes (MARON et al., 2013a; AVILA et al., 2015).

Observe in Figure 4 how the D-GM Simulation Framework is organized. It is divided into the following levels, in a top-down sequence:

(i) *Quantum Circuit Level:* describing the application in the circuit model and then automatically exporting it to a representation for the qGM (Quantum Geometric Machine) model (MARON et al., 2013b).

(ii) *qGM Level:* containing the Visual Programming Environment for the qGM Model (VPE-qGM) which allows the user to describe/simulate computations under the *qGM* (MARON; REISER; PILLA, 2013).

(iii) *D-GM Level:* implementing the distributed simulation manager, Virtual Distributed Geometric Machine (VirD-GM), which handles tasks such as scheduling, communication, and synchronization required in distributed simulations (AVILA et al., 2014).

(iv) *Hardware Level:* enlisting all the devices that can be used by the framework, from regular desktops for sequential simulations to clusters with multiple GPU.

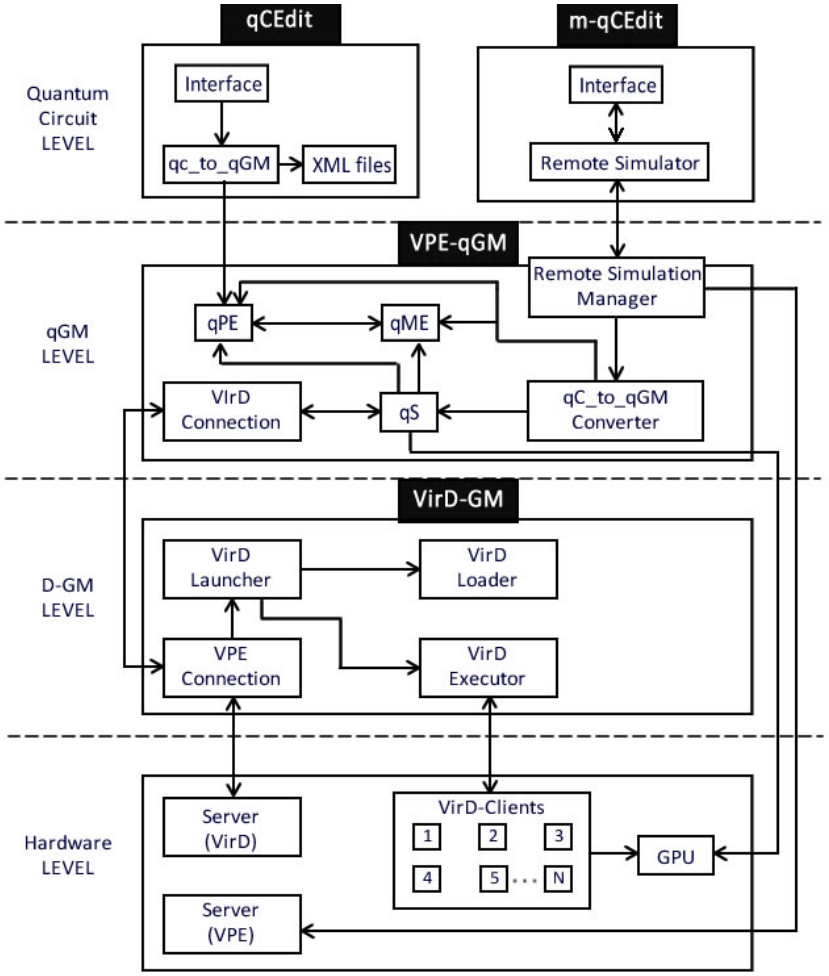


Figure 4 – The D-GM simulation framework. (AVILA et al., 2014).

In more recent years, the D-GM framework has been worked on in the LUPS/UF-PEL projects, and a new paradigm for optimization of quantum computation simulations was incorporated, which are in this case, mainly based on decomposition and reduction operators in order to control the granularity and distribution of the computations, showing significant improvements compared to its previous version (AVILA et al., 2015, 2016b; AVILA; REISER; PILLA, 2016).

This consolidated structure already under development by both research groups, the LUPS and MFFMCC, integrating HPC and DTM researches, justifies the support in this work by the D-GM framework, for assistance in the design of the HybriD-GM model as well as for developing its concepts through the D-GM extension.

In the next sections, the D-GM framework stage at the beginning of this work will be detailed.

3.1 Reducing Simulation Complexity

The optimizations of the D-GM environment (AVILA et al., 2016a,b; AVILA; REISER; PILLA, 2016), mainly related to reduce the spatial and temporal complexity associated to QT by the smart use of the Identity operator (Id-operator), are described in the following two subsections.

3.1.1 Avoiding replication and sparsity inhered from Id-operators

The first optimization explores the behavior associated with the Id-operator and other QT by tensor products. In such cases, the Id-operator not only replicates the values of other operators but also introduces sparsity in the QT. Thus, it is possible to store only the tensor product expansion among QT different from the Id-operator, decreasing the spatial complexity by generating a reduced matrix (RM). See this behaviour depicted in Eq. (14) related to the QT $Id \otimes H$.

$$Id \otimes H = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 & 0 \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ 0 & 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{pmatrix} \quad (14)$$

Since the RM order is lower than the state dimension, it is not possible to perform the multiplication between matrix/vector as it usually is 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 multiplications.

As an illustration, it considers a generic operator applied to the first qubit, Eq.(15), and to the second qubit, Eq.(16), of an 2-dimensional state. 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.

$$\begin{pmatrix} \mathbf{m_{00}} & \mathbf{m_{01}} \\ \mathbf{m_{10}} & \mathbf{m_{11}} \end{pmatrix} \times \begin{pmatrix} \mathbf{a_{00}} \\ \mathbf{a_{01}} \\ \mathbf{a_{10}} \\ \mathbf{a_{11}} \end{pmatrix} = \begin{pmatrix} \mathbf{a_{00}} \times \mathbf{m_{00}} + \mathbf{a_{10}} \times \mathbf{m_{01}} \\ \mathbf{a_{01}} \times \mathbf{m_{00}} + \mathbf{a_{11}} \times \mathbf{m_{01}} \\ \mathbf{a_{00}} \times \mathbf{m_{10}} + \mathbf{a_{10}} \times \mathbf{m_{11}} \\ \mathbf{a_{01}} \times \mathbf{m_{10}} + \mathbf{a_{11}} \times \mathbf{m_{11}} \end{pmatrix} \tag{15}$$

$$\begin{pmatrix} \mathbf{m_{00}} & \mathbf{m_{01}} \\ \mathbf{m_{10}} & \mathbf{m_{11}} \end{pmatrix} \times \begin{pmatrix} \mathbf{a_{00}} \\ \mathbf{a_{01}} \\ \mathbf{a_{10}} \\ \mathbf{a_{11}} \end{pmatrix} = \begin{pmatrix} \mathbf{a_{00}} \times \mathbf{m_{00}} + \mathbf{a_{01}} \times \mathbf{m_{01}} \\ \mathbf{a_{00}} \times \mathbf{m_{10}} + \mathbf{a_{01}} \times \mathbf{m_{11}} \\ \mathbf{a_{10}} \times \mathbf{m_{00}} + \mathbf{a_{11}} \times \mathbf{m_{01}} \\ \mathbf{a_{10}} \times \mathbf{m_{10}} + \mathbf{a_{11}} \times \mathbf{m_{11}} \end{pmatrix} \tag{16}$$

Although this concept optimizes the representation of QT involving Id-operators, not all QT 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 QT.

3.1.2 Decomposing QT based on *Id*-operators

An n-dimensional QT can be decomposed increasing the number of steps for its computation, allowing to control the amount of Id-operators in each step, preserving the behaviour and properties of the QT. Figure 5 shows the QT $H \otimes H$ and its decomposition in two steps, $H \otimes I$ and $I \otimes H$, keeping the same behaviour regardless the composition order of these steps.

Controlled QT can also be decomposed conserving the controls associated to the operators, as show in Figure 6.

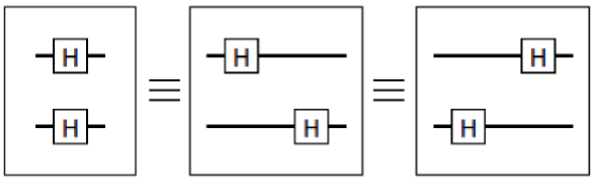


Figure 5 – Decomposition of non-controlled QT

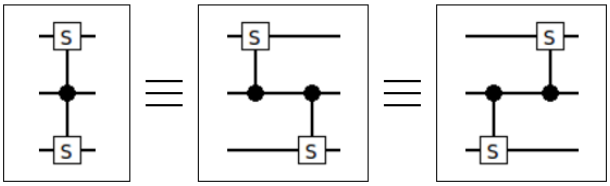


Figure 6 – Decomposition of controlled QT

Using these optimized QT, the spatial complexity can be reduced, limiting the number of Id-operators in steps of decomposed QT, providing representation of each step by a single matrix.

3.2 Improving scalability of QT

Despite the possibility of modeling QT 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 QT, since it also increases exponentially (2^n). For example, a 28-dimensional QT needs 4 GB memory space to store both states. Once the GPU memory is typically smaller than the main RAM, it is necessary to adopt an approach providing scalability to multi-qubit QT.

The Mixed Partial Process (MPP) concept, presented in (AVILA et al., 2015), is a control strategy over the increase in the size of read/write memory-states, improving the scalability related to QT computations. Additionally, based on the above, n-dimensional QT 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 computation possible.

Using the MPP optimization, we are able to deal with the number of read sub-states accessed by each write sub-state in order to perform the 2^r calculations related to their amplitudes, when r is the number of operators affected by the partition. Here, affected operators refering to the number of operators different from Id-operator, which is presented in the first p qubits of the current step of computations. Therefore, steps with none affected operators need only the correspondent read sub-state, which makes them totally independent.

Due to the access dependencies and the consistence preservation, affected steps should have all sub-states previously calculated, before passing it to the next step. Steps not affected may be calculated in an analogous way or by an iterative method, sub-state by sub-state, since there are no dependencies. For controlled steps, only sub-states satisfying the controls must be calculated or read.

3.3 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 GPU is divided in two parts:

- (i) **classification of QT in groups**, dividing operators non-controlled and with distinct controls;
- (ii) **definition of QT steps**, each one is formed by operators that belongs 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.

See Figure 7 for a 9 qubits QT considering limits of 3 operators by step and 8 qubits for execution. QT is firstly divided into 3 groups and from these, in 5 steps. The group 1 is divided into 2 step, despite having 3 operators in consecutive qubits, since the memory partition affects the first qubit.

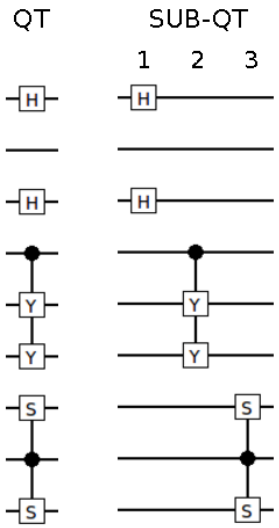


Figure 7 – Decomposition of a 9-qubit quantum transformation.

3.3.1 CPU Execution

The best results in CPU using the decomposition approach are reached when limits in the number of operators in steps 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, such as the Hadamard operator. 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, as the Pauly Y operator, or in the secondary diagonal, such as the Pauly X operator. In these cases, optimizations discarding calculations with void elements may be applied without modifications in the results.

The calculation of dense operators is as described in Eq. (15). For sparse operators, Eq. (17) and (18) 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.

$$\begin{pmatrix} \mathbf{m_{00}} & \mathbf{0} \\ \mathbf{0} & \mathbf{m_{11}} \end{pmatrix} \times \begin{pmatrix} \mathbf{a_{00}} \\ \mathbf{a_{01}} \\ \mathbf{a_{10}} \\ \mathbf{a_{11}} \end{pmatrix} = \begin{pmatrix} \mathbf{a_{00}} \times \mathbf{m_{00}} \\ \mathbf{a_{01}} \times \mathbf{m_{00}} \\ \mathbf{a_{10}} \times \mathbf{m_{11}} \\ \mathbf{a_{11}} \times \mathbf{m_{11}} \end{pmatrix} \tag{17}$$

$$\begin{pmatrix} \mathbf{0} & \mathbf{m_{01}} \\ \mathbf{m_{10}} & \mathbf{0} \end{pmatrix} \times \begin{pmatrix} \mathbf{a_{00}} \\ \mathbf{a_{01}} \\ \mathbf{a_{10}} \\ \mathbf{a_{11}} \end{pmatrix} = \begin{pmatrix} \mathbf{a_{10}} \times \mathbf{m_{01}} \\ \mathbf{a_{11}} \times \mathbf{m_{01}} \\ \mathbf{a_{00}} \times \mathbf{m_{10}} \\ \mathbf{a_{01}} \times \mathbf{m_{10}} \end{pmatrix} \tag{18}$$

The QT execution is realized operator by operator. For each one, its type is identified and then the corresponding loop is executed, where each iteration produces one new amplitude. Parallel execution in CPU was implemented using OpenMP (OpenMP Architecture Review Board, 2015), adding the “parallel for” pragma in these loops, so each thread calculate $\frac{2^n}{th}$ new amplitudes, with n being the number of qubits of the state and th the number of threads.

3.3.2 GPU Execution

After the decomposition, QT steps are calculated. As seen in Section 3.2, 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 iterative 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.

All data that is used just for reading is stored on the GPU constant memory. Each call of the CUDA kernel receive the following parameters:

- Read/Write memory states (sub-states);
- Reduced matrix from the current step;
- Controls value and positions (if there);
- Access information for these structured described above.

The CUDA kernel computation may be divided into 5 steps, as described below.

Step 1: Identifies each thread’s $lineId$ from information about the current thread and control information. The $lineId$ defines the write position for each thread.

```
long read_shift = arg[SHIFT_READ];
long shift_write = arg[SHIFT_WRITE];
long lineId = (blockIdx.y*gridDim.x+blockIdx.x)*blockDim.x+threadIdx.x;
if (arg[CTRL_COUNT]){
   for(i=arg[CTRL_COUNT]-1;i>=0;i--)
      lineId = (lineId*2) - (lineId & (1<<(ctrl_pos[i])-1));
   lineId = lineId | arg[CTRL_VALUE];
}
lineId = lineId | shift_write;
```

Step 2: Initializes local variables using the $lineId$:

```
long p = arg[MAT_START];
long size = arg[MAT_SIZE];
long shift = arg[SHIFT];
long r_mask = (size-1) << shift;
long inc = 1 << shift;
long read_pos = (lineId & ~r_mask) + (p<<shift);
long base = ((lineId&r_mask)>>shift) * size;
long end = arg[MAT_END];
long r_shift = arg[SHIFT_READ];
```

Step 3: Computes the new amplitude partial update from variables calculated in the previous step to control the reduced matrix and state access:

```
cuFloatComplex accum=make_cuFloatComplex(0.0,0.0);
for (;p<end;p++){
   accum=cuCaddf(accum,
         cuCmulf(rMem[read_pos-r_shift],
         matrix[base+p]));
   read_pos += inc;}
```

Step 4: Stores and accumulates new amplitudes to the write memory state vector in the GPU's global memory. The first kernel provides a write memory partition, whose values are stored in the next calls in the same write memory partition, and accumulated with the previous values:

```
lineId-=shift_write;
if (arg[ACUMM])
   wMem[lineId] = cuCaddf(wMem[lineId],accum);
else
   wMem[lineId] = accum;
```

Step 5: Copies the complement positions associated to the QT control from the read to the write memory state vector:

```
if (arg[CTRL_CMPL]){
   lineId = lineId & (~arg[CTRL_MASK]);
   for (i = 0; i < arg[CTRL_CMPL]; i++){
      p = lineId | ctrl_cmpl[i];
      wMem[p] = rMem[p];}}
```

3.4 Summarizing

The D-GM environment supports the strategy named ReDId, which provides optimizations based on minimizing replication and exploring Identity operators for significant reduction in spatial complexity and, when combined with a decomposition of quantum transformations, they can also be reduced from the temporal complexity of the simulations.

In addition, quantum applications can explore the potential of the components VPE-qGM and VirD-GM which integrate the D-GM environment.

For evaluation, modeling and implementation of the execution of quantum algorithms, these strategies can be simulated in C/C ++ for executions over CPU, sequentially and in parallel, and on *CUDA* for executions on in GPU, showing reduced temporal complexity and, consequently, shorter simulation time and obtaining the highest possible performance in each one of these architectural structures.

4 QUANTUM COMPUTING SIMULATION: RELATED-WORK

One of the main obstacles for the adoption of quantum algorithm simulation is the exponential increase in temporal and spatial complexities, due to the expansion of transformations and read/write states by using tensor product in multi-dimensional applications. Simulation of these systems is very relevant to develop and test new quantum algorithms (DE RAEDT et al., 2019).

Recently, relevant results have also been published in quantum computing simulation, see, e.g. (HILLMICH; ZULEHNER; WILLE, 2020) promoting concurrency in DD-based Quantum Circuit Simulation. In addition, quantum computing simulation has the potential to provide solutions to many problems which are challenging or out of reach of classical computers. See, e.e.g, several problems in rendering which are amenable to being solved in quantum computers, as proposed in (ALVES; SANTOS; BASHFORD-ROGERS, 2019) an implementation of Grover's Algorithm (a quantum search algorithm) for ray casting.

This Chapter presents the selected quantum simulators, their main characterization, execution mode and simulation results, in order to provide the comparison and analysis in Section 4.7 and summarizing related results at the end.

In the selection of six related work, the following characteristics were considered:

(i) Consolidate projects dealing with general purpose quantum computing simulation;

(ii) HPC approach having multi-core CPU, GPU and/or distributed executions; and,

(iii) Reporting optimizations that were considered relevant for this proposal.

In the sequence, a description for selected related work are presented.

4.1 LIQUi$|\rangle$

LIQUi$|\rangle$ which stands for Language Integrated Quantum Operations (WECKER; SVORE, 2014), conceives as a software architecture and tool suited for QC being developed by the Quantum Architectures and Computation Group (QuArC) at Microsoft Research, providing users with an end-to-end exploration and control environment from

algorithm writing, increasing visualization, simulation, emulation, and deployment on target hardware, with an ultimate goal of controlling quantum hardware.

LIQUi|⟩'s architecture is summarized in Figure 8. High level programming uses the language F# and its compiler, interpreter, or any other high-level language (e.g., C#) that can be linked with the LIQUi|⟩ library.

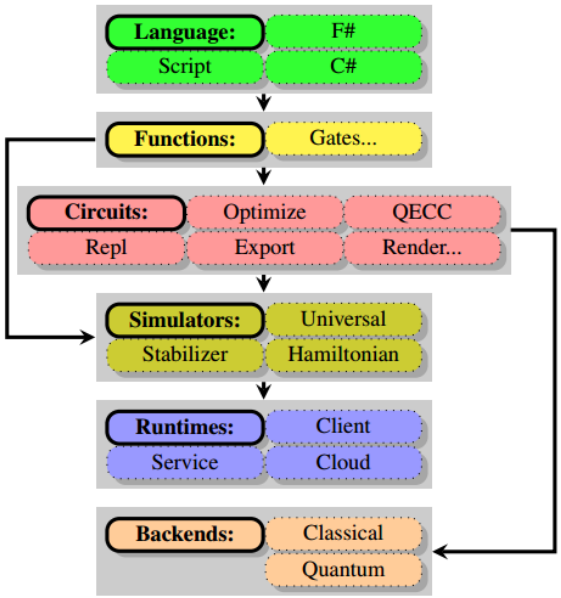


Figure 8 – LIQUi|⟩ architecture.

An executable *Gate* used in a QA is referred to in LIQUi|⟩ as *operation*, meaning an F# function whose signature is required to have the last argument as a list of qubits (state vector).

4.1.1 Execution Modes and Simulation Results

Execution modes consider the following functions:

(i) *Test mode*, invoked from the command line, such as Shor's algorithm.

(ii) *Script mode*, running directly from an F# text script (.fsx file) and allowing the simulator to be operated by simply running the executable with no need to install a complete development environment and also used for submission to Cloud services.

(iii) *Function mode*, requiring a compilation environment (e.g., Visual Studio) and the use of a .Net language (typically F#) and providing the full range of APIs to extend the environment in many ways as well as allowing the users to build their own complete applications.

(iv) *Circuit mode*, compiling a function mode into a circuit data structure, running through built-in optimizers, having quantum error correction added, rendered as

drawings and exported for using in other environments, and may be running directly over all the simulation engines.

The largest number factored on LIQUi|⟩ with Shor's algorithm is a 14-bit number (8193) which required 31 qubits in 50GB of memory, 28 rounds with half a million gates per round (reduced to 18,000 using gate growing), and ran for 43,384 minutes (30.1 days).

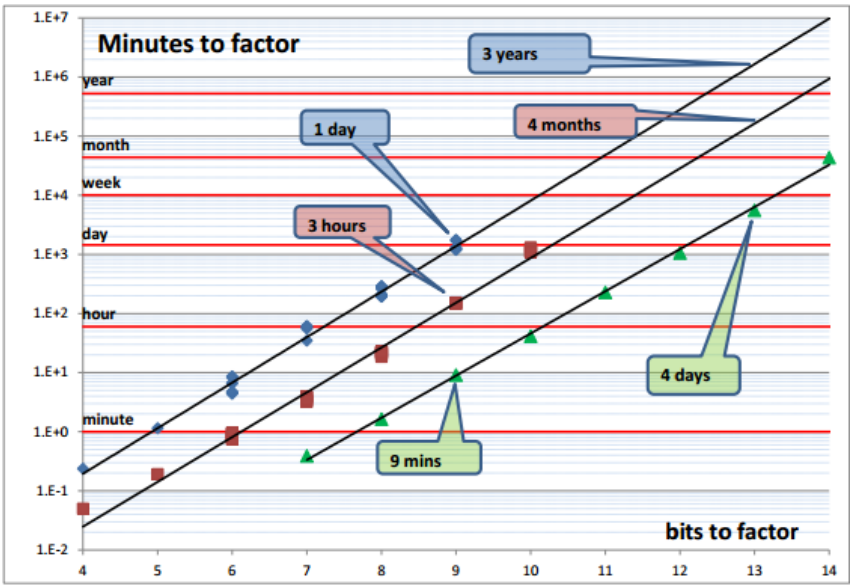


Figure 9 – Shor results for LIQUi|⟩. Source: (WECKER; SVORE, 2014)

4.2 qHiPSTER

The qHiPSTER (Quantum High Performance Software Testing Environment) is a distributed high-performance implementation of a quantum simulator on a classical computer developed by Intel's Parallel Computing Lab (SMELYANSKIY; SAWAYA; ASPURU-GUZIK, 2016), focusing on general single-qubit gates as well as two-qubit controlled gates (including, controlled-NOT gate), which are known to be universal. However, similar to the D-GM framework, they do not build the entire transformation, storing only the single-qubit gate matrix.

The *Single node implementation* follows the explanation above, including optimizations as vectorization, threading and cache blocking through gate fusion integrated to pseudo-code of computations.

In their distributed implementation, a state vector of 2^n amplitudes (2^{n+4} bytes) is distributed among 2^p nodes, such that each node stores a local state of 2^m amplitudes, where $m = n - p$. For a gate operating on a qubit k, if $k < m$ the operation is contained within a node, otherwise communication is required. In the communication scheme,

each node stores its 2^m local state vectors in two halves and has an extra 2^{m-1} temporary vector that is used to perform the pairwise exchange of such halves (pair of nodes varies according to k). Then, the gate is computed on the temporary vector and the halves are exchanged back. For controlled gates the approach is similar, using only the target qubit to determine if communication will be needed. Enabling the simulation with 31 qubits and increasing the number of steps, it makes use of 8GB for the temporary vector. As long as the amount of data exchanged within each step is large enough to saturate the network bandwidth, the overall run-time remains the same.

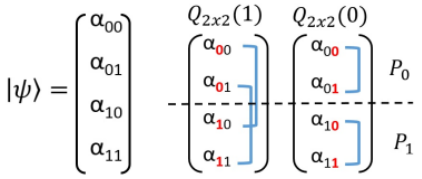


Figure 10 – Example of two qubit state and single-qubit gate operations. Source: (SMELYANSKIY; SAWAYA; ASPURU-GUZIK, 2016)

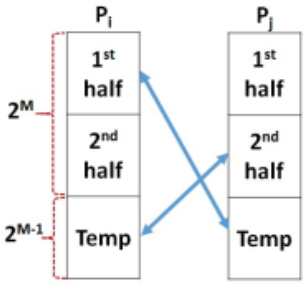


Figure 11 – Distributed implementation of a single-qubit gate operation. Source: (SMELYANSKIY; SAWAYA; ASPURU-GUZIK, 2016)

4.2.1 Simulation Results

The performance and scalability of qHiPSTER was evaluated on the Stampede supercomputer (Texas Advanced Computing Center (TACC), 2017) consisted of of 6,400 compute nodes, each one with two Xeon E5-2680 sockets connected via QPI and 32GB of DDR4 memory per node (16GB per socket).

Several tests were made, showing single node and multi node performances for single-qubit gates (and controlled gates) varying the qubit it is being applied (as well for the controls of controlled gates). The performance for a QFT was reported varying the number of qubits from 29 to 40, using local state vectors of size 2^{29}.

4.3 ProjectQ

ProjectQ is an Open Source Software Framework for QC, its interface is implemented in Python because of its simple learning curve and its kernels in C++ for more performance on simulation(STEIGER; HäNER; TROYE, 2018). The ProjectQ framework can be seen in Figure 12, its main components (quantum program, compiler and back-ends) are modular allowing for easier extensions.

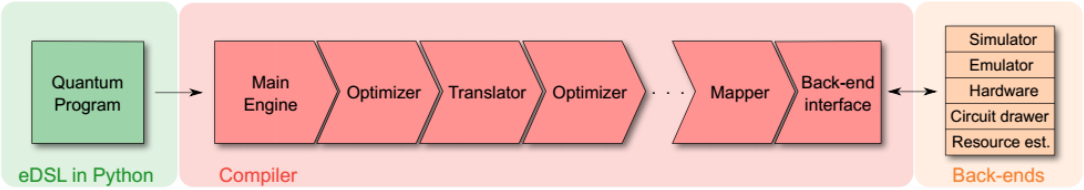


Figure 12 – ProjectQ's full stack software framework. Source: (STEIGER; HäNER; TROYE, 2018)

QAs written in a high-level domain-specific language embedded in Python can be compiled into low-level instruction sets supported by the various back-ends, including interfaces to quantum hardware, a high-performance quantum simulator and an emulator providing a circuit drawer and a resource counter.

4.3.1 Simulation Results

Simulation is restricted to Hadamard-transformations including a chain of controlled Z-rotations, running on an Intel Core i7-5600U CPU. By simulation times shown in Figure 13, ProjectQ was between $3\times$ and $5\times$ faster than its predecessor (HäNER et al., 2016), which had already presented better results when compared to LIQUi$|\rangle$ and qHiPSTER.

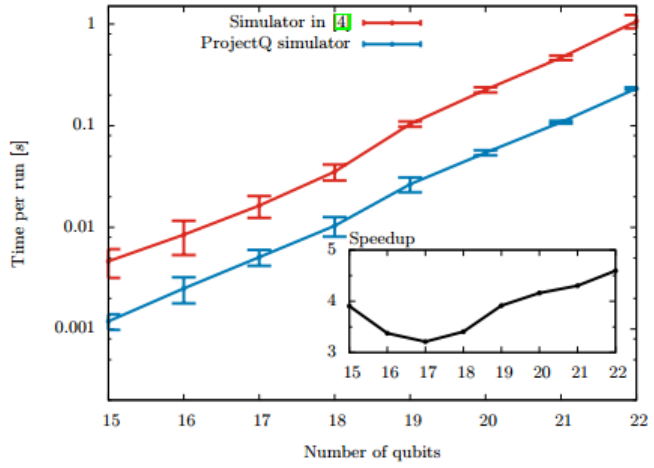


Figure 13 – Runtime comparison the simulator from (HäNER et al., 2016) to the ProjectQ. Source: (STEIGER; HäNER; TROYE, 2018)

4.4 Haner Distributed Simulator

Haner and Steiger (2017) introduced a distributed simulator that most likely will be included in the ProjectQ framework in a near future, since the authors are the same. In (HäNER et al., 2016) they mentioned works on including a distributed massively parallel quantum simulator to the framework.

Following most significant simulators, the full transformation matrix is not stored. So, k-dimensional gates are directly applied to the state vector when k is the lower order, allowing to store gate transformations in memory. To reduce the memory requirements by a $2\times$-factor, the state vector calculation is performed in-place instead of using other state vector for the output.

In simulations on single-core, the number of operations grows exponentially, meaning that to apply a k-dimensional transformation ($2^k \times 2^k$ matrix position) to a state vector of size 2^n, all sets of amplitudes with equal neutral-bits (2^{n-k} sets of size 2^k) are calculated at one time: loading their values into a temporary vector, performing the matrix-vector multiplication and then writing the result (stored in another temporary vector) back to the state vector.

The k-dimensional matrix is used at 2^{n-k} times, improving performance by permuting matrix before-hand sorted qubit indices and therefore a more local fashion memory access. Related to the matrix size dependency, they make use of blocking computation related to an automatic code-generation/benchmarking feedback. The two main optimizations at instruction level consist of:

(i) *Vectorization*, parallelizing updates of consecutive values of the output vector within a block using AVX/AVX512 when it is supported;

(ii) *Instruction Reordering*, avoiding artificial dependencies and additional permutations related to a multiplication between one complex entry of the temporary vector with one complex entry of the gate matrix and also summing the result into the temporary output vector.

In simulations on single-node, as long as the application remains memory bound, larger gates can be required (almost) at the same amount of time on this simulator. Thus, by combining multiple gates acting on k different qubits into one large k-dimensional gate, it increases related performance. The choice of k-parameter takes into account the peak performance, the memory-bandwidth, the cache-size & associativity of the system and the quantum application.

The simulation considers one kernel for each value of k-parameter, using OpenMP with NUMA-aware initialization of the state vector, scaling beyond 1 NUMA node.

4.4.1 Circuit Optimizations and Simulation Results

The circuit was optimized in order to reduce the number of communication steps and better use their kernels by considering the following two techniques, gate scheduling and qubit mapping.

(i) *Gate scheduling*: The optimizations related to gate scheduling are divided into two main steps described in the following:

(ii) Minimizing the number of communication steps by reordering (if possible) the gates into stages, where each stage consists of a large sequence of possible quantum gates, which only acts on local qubits.

(iii) By swapping global qubits with the lowest-order local qubits, the application is able to achieve at an upper bound for the number of communication steps required. Additionally, a search algorithm is considered to find better local qubits.

(iv) Minimizing the number of k-qubit gates, considering sequences of consecutive $1-$ or $2-$dimensional gates into k-dimensional gates, greedily trying to increase k till k_{max}, where k_{max} is the largest k for which the k-dimensional gate kernel still shows good performance on the target system.

Qubit mapping: The bit-location of each qubit is remapped to reduce the number of k-qubit gates with performance decreased resultant from the set-associativity of the last-level cache. Initially, it is assigned to bit-location 0 the qubit that maximizes the number of k-dimensional gates accessing this bit-location.

Simulations were performed on the Cori II system at the Lawrence Berkeley National Laboratory (LBNL), with results showing the performance evaluation on single-node varying the k-qubit gates kernel along with the qubits being applied (low and high order); as well as the scaling behaviour for these kernels on multi-node. The only algorithm simulation performed was of quantum supremacy circuits (BOIXO et al., 2018) featuring 30, 36, 42, and 45 qubits and the speedup obtained with the increases in number of nodes are shown, as well as the percentage in time spent in communication on each configuration.

In order to compare with the results presented in (BOIXO et al., 2018), $30-$ and $36-$dimensional quantum supremacy circuits were simulated on the Edison system, also at LBNL. Using up to 64 sockets, each featuring a 12-core Intel® Xeon® Processor E5-2695 v2 at 2.4GHz. Showing a speedup of $3\times$ and $4\times$ for the single-node and multi-node executions, respectively.

4.5 Gutierrez simulator in CUDA

Gutierrez et al. (2010) introduced a quantum computing simulator using the CUDA programming model, with an implementation for a single GPU. QT are decomposed into a sequence of stages, each stage has lower order than the transformation and is defined by a group of gates in the original order, allowing a partition of the quantum state into sets satisfying these three features:

(i) sets are closed for every gate in the stage;
(ii) the cardinality of all the sets is at most 2^r, for a certain value r;
(iii) all the sets are at least c-coalesced for a certain c, which means that each set can be divided into sequences with at least 2^c amplitudes.

Each kernel call determines the calculation related to one stage, requiring each CUDA block will be in charge of processing one (or several) closed groups and the CUDA threads within a block will be responsible for computing a couple of amplitudes up to a whole closed group depending on the defined granularity.

Related to the kernel call for a c-coalesced stage, firstly, each CUDA block transfers its closed set(s) to the shared memory in a c-coalesced way. In the sequence, the gates belonging to the stage are applied in-place over the subset in shared memory with a thread-level synchronization between gates. Finally, the amplitudes are transferred back to their positions on global memory in a c-coalesced way.

Figure 14 illustrates a CUDA block processing performed in the closed-set denoted by P_0. In this first stage, the amplitudes belonging to P_0 are copied-in from global memory to shared memory, keeping a coalescing degree of 2^c. After that, all gates of the stage are computed one by one, considering that each thread calculates a minimum closed set of two amplitudes in-place on the shared memory. Observe that, a thread level synchronization is necessary, integrating computations of each gate. Each thread is in charge of computing a pair of coefficients operating in-place. Finally, the amplitudes are copied back to global memory.

4.5.1 Simulation Results

Experiments were performed over a GeForce 8800GTX GPU NVIDIA, considering a Hadamard transformation applied to all qubits and a QFT up to 26 qubits (memory limit) showing their results for combinations of value c and r, explaining corresponding differences in performance evaluation. Execution times were compared to the sequential simulator libquantum (BUTSCHER; H., 2017) achieving speedups of $95\times$ for the 26 qubit QFT.

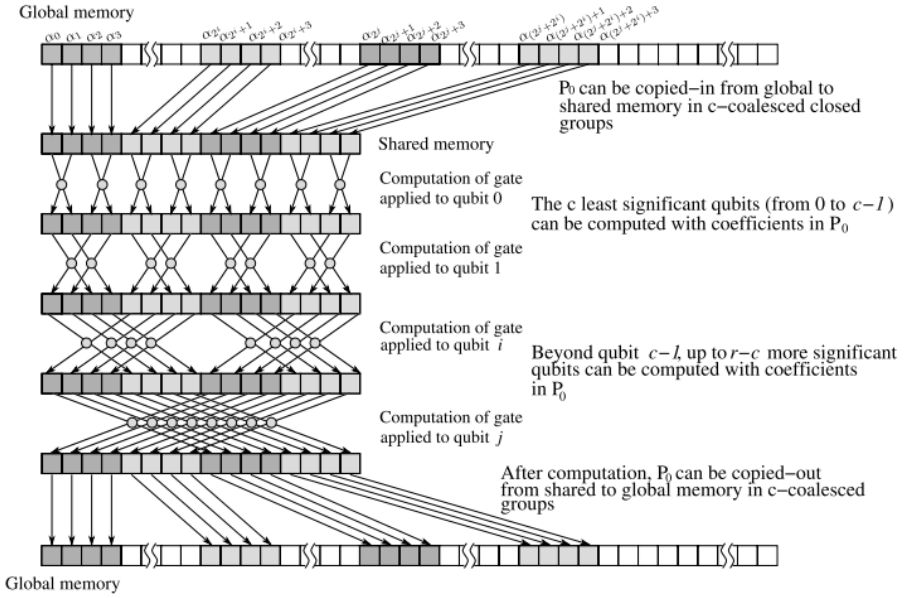


Figure 14 – Coalescing-aware strategy for a block. Source: (GUTIERREZ et al., 2010)

4.6 Zhang simulator

Zang et al. (2015) introduced a quantum computer simulator, which is able to consider simulations over multiple GPU within a single node.

For GPU with capacity of allocating state vectors of size 2^l, qubits $< l$ are considered local and $\geq l$ global. Gates applied to local qubits do not require communication between devices, in opposite situation of those applied to global qubits. This analysis is analogous to discussion on distributed simulators, previously presented. In order to avoid performing data transfer between each device for gates on global qubits, they make use of the same idea of swapping global qubits with local qubits.

For local-qubit gates, only one transfer of 2^l continuously amplitudes is made to each device. And, for global-qubit gates, global qubits are swapped with the highest local qubits. This way the size of each continuous batch is maximized, 2^r were r is the number of local qubits not swapped, while the number of batches transferred to each device is minimized, 2^s were s is the number of qubits swapped. Batches with the same s bits on the highest local qubits are mapped to the same device.

Figure 15 shows an example for for $n = 5$ and $l = 3$. The size of each batch is $2^1 = 2$, and $2^2 = 4$ batches are transferred to each device. The workflow of a simulation in n GPU within a node is shown in Figure 16, and its steps are detailed below:

1 - Initialization of the quantum state performed in CPU.

2 - The state amplitudes are distributed along the GPU, according to the explanation above that minimizes the number of transfers.

3 - The GPU kernel computes the quantum gate. One closed set of amplitudes is assigned to each CUDA block, which copies it to shared memory in a coalesced

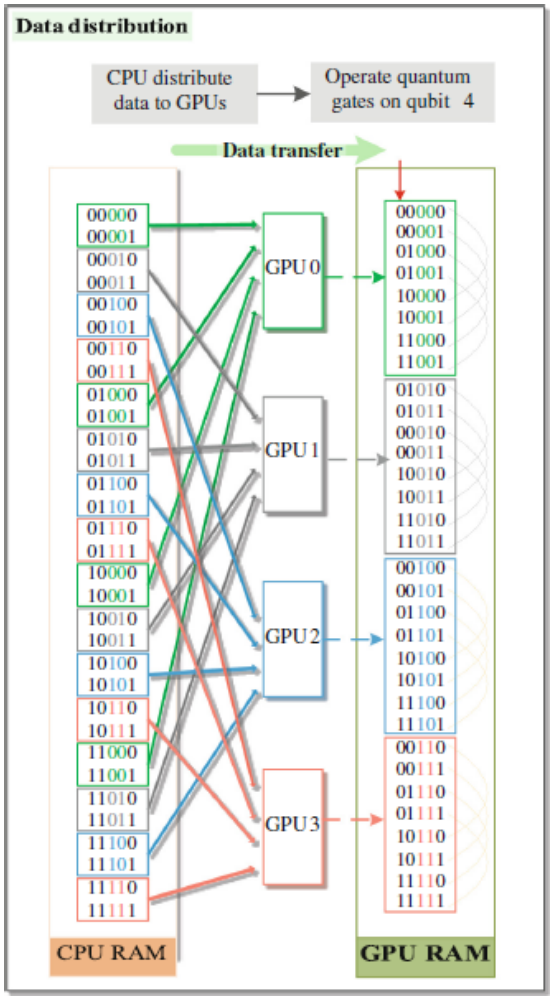


Figure 15 – Data distribution for $n = 5$ and $l = 3$. Source: (QUANTUM COMPUTER SIMULATION ON MULTI-GPU INCORPORATING DATA LOCALITY, 2015)

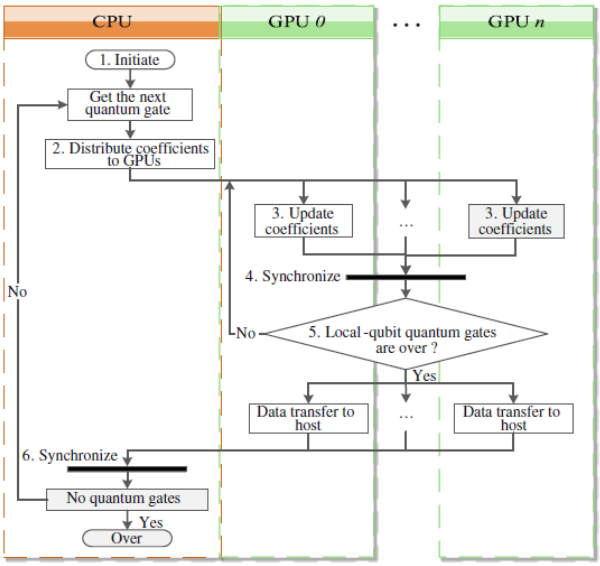


Figure 16 – The workflow of this quantum computer simulation. Source: (QUANTUM COMPUTER SIMULATION ON MULTI-GPU INCORPORATING DATA LOCALITY, 2015)

way following the same approach presented in (GUTIERREZ et al., 2010). Then each CUDA thread in the block computes one amplitude.

4 - After the gate computation, a synchronization of multi-GPU is required.

5 - Consecutive quantum gates operating on local qubit are processed continuously on the device memory. And then the amplitudes on shared memory are copied back to global memory.

6 - After the kernel execution, the quantum state is transferred back to host and a synchronization of multi-GPU is performed on the CPU side to guarantee data consistency.

The simulation of a 4-qubit QFT in a node of 4 GPU is presented in Figure 17.

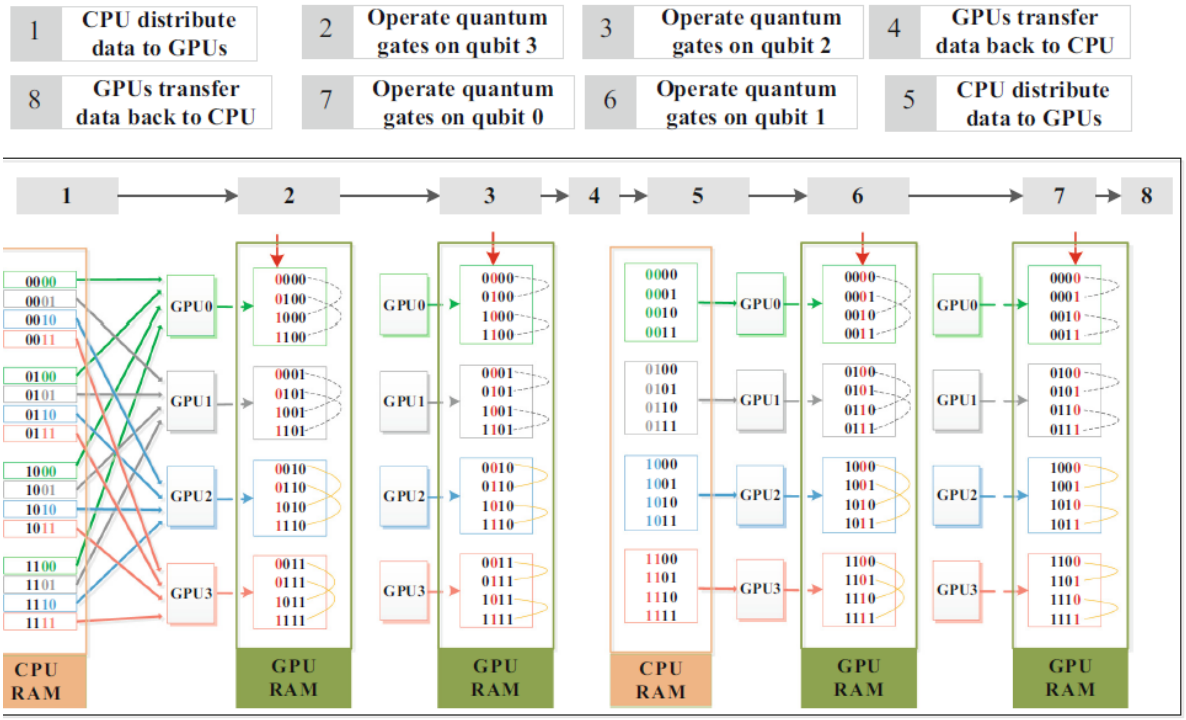


Figure 17 – Simulation of 4-qubits QFT in a node with 4 GPU. Source: (QUANTUM COMPUTER SIMULATION ON MULTI-GPU INCORPORATING DATA LOCALITY, 2015)

4.6.1 Results

These tests were performed on a platform with four NVIDIA K20c @706 MHz GPU and an Intel Xeon E5-2609 v2 @ 2.5 GHz CPU. QFTs were executed and the performance of the simulator was evaluated by comparing the results of simulations with 1 GPU, 2 GPU and 4 GPU, to intermediate versions of their simulator and to the sequential simulation on CPU using the libquantum achieving speedups of $378\times$ on 4 GPU and $189\times$ on 2 GPU for the 30 qubits QFT.

4.7 Analysis of selected quantum computing simulators

Now, the analysis of selected quantum simulators is presented, considering the use and multicore structure, single or multi-GPU characterization, distribution of computations and circuit optimizations.

LIQUi$|\rangle$ simulator, reported in Section 4.1, which has it mains computation running in a functional language ($F\#$), presented as one of the best option for high performance applications. Two interesting optimizations should to be emphasized: (i) gate growing, allowing to greatly reduce the number of gates on a QA; and (ii) full rewrite of the complex math package, reducing drastically simulation time.

qHiPSTER, see details in Section 4.2, presenting results to understand its performance behaviour on a supercomputer but restrict to small cases, as complex

algorithms were not simulated. A multi-qubit simulation, over 29 qubits of a Quantum Fourier Transformation (QTF) in a single node took 116.6 s while the simulation on the D-GM took less then 5 s. Simulation time remains very large, independently of the selected (double-/single-) precision storing the state vector.

ProjectQ, referred in Section 4.3, seems interesting as an open source framework but the work on (STEIGER; HäNER; TROYE, 2018) does not present any detailed information about its simulator for general QA. Despite presenting better results than LIQUi$|\rangle$ and qHiPSTER, a simple hardware (notebook with only two cores) was used on the simulations. Further investigations are needed to evaluate its performance over robust parallel processing power architectures.

Haner's simulator, reported from (HANER; STEIGER, 2017) in Section 4.4, applies all the optimizations on the table for multi-core and distributed simulations. Since it was built to primarily simulate quantum supremacy algorithms, main optimizations and presented results are target towards this algorithm types, making difficult an analysis performed over more complex algorithms.

Gutierrez's simulator, Section 4.5, even though it was the oldest selected simulator here (2010) impacting that GPU having further increased both performance and memory storage since then, its strategy shows an approach similar to the D-GM, by considering coalesced access to GPU global memory. However, it is not possible to directly compare results with newer hardware.

Zhang's simulator, presented in Section 4.6, considers a single node multi-GPU implementation having a kernel similar to (GUTIERREZ et al., 2010) and using the same approach presented in (HANER; STEIGER, 2017) to avoid communication between devices and reducing memory transfers. The results presented considers only QFT and comparison with intermediate versions of itself. And, an analysis of the performance scalability is not discussed.

4.8 Summarizing

Table 1 summarizes an overview of the simulators showing main optimizations for quantum computing simulation over multi-cores CPU, multi-nodes CPU and single-node GPU (with single or multiple GPU). None of the simulators has a hybrid approach combining CPU and GPU, and although they have several interesting optimization strategies, many of those are targeted to a specific architecture. To overcome that, the proposal of this work was conceived, considering a computational model for quantum computing simulation that can be applied to any architecture while optimizing its resources.

Table 1 – Overview and summary analysis of literature review related to selected quantum computing simulators

		D-GM	LIQUi|⟩	qHiPSTER	ProjectQ	Dist 45	Gutierrez	Zhang
Multi-core	Model for Parallel Programming	OpenMP	$F\#$	OpenMP	OpenMP	OpenMP	–	–
	AVX support	X	X	X	•	•	–	–
	Cache optimizations	X	•	•	X	•	–	–
	Instruction Reordering	X	X	X	X	•	–	–
Single- or Multi-GPU	Global Memory Coalesced Access	X	–	–	–	–	•	•
	Uses Shared Memory	•	–	–	–	–	•	•
	Uses Constant Memory	•	–	–	–	–	X	X
	Many gates per kernel call	•	–	–	–	–	•	•
Distributed	With	X	Azure	No API	X	MPI	X	X
	Global Gates opt. (controlled and sparses)	–	X	X	–	•	–	–
Circuit Opt.	Scheduling	X	X	X	X	•	X	X
	Gate Growing	X	•	X	X	•	X	X

• implemented X not implemented – cannot be

5 HYBRID-GM PROPOSAL: CONCEPTUAL MODEL

The state space and operators in QC are mathematically described by the Hilbert space, where a quantum register comprising a number of qubits is given as a vector in a multidimensional Hilbert space while quantum gates are Hilbert space operators that rotate the quantum register vectors (HIRVENSALO, 2001). Since quantum computing can be conceived as a process that incorporates interacting physical systems represented by quantum bits and quantum gates, the corresponding quantum circuit model is also considered to graphically modeling computations(KNILL et al., 2002)(NIELSEN; CHUANG, 2000).

Underlying the main quantum circuit model concepts, composition, normalization, and projection operators are explored in the HybriD-GM model considering the following targets: (i) modeling the computation of quantum states and transformations represented by matrix-structures in a quantum application; (ii) preserving different levels in the interpretation for quantum transformations, based on the partiality of processes to control the granularity of compelling computations; and (iii) minimizing redundancies by promoting decomposition, reduction operations and making use of projections on multidimensional systems.

In the following sections, the main characteristics of the matrix-structure of projection operators are presented, including a discussion about the dynamic of computations based on projected structures.

5.1 Matrix-structure of quantum state projections

Projection operators applied to a multidimensional QS result on the partition of the classical basis components. Thus, each new subset of basis components preserves the corresponding amplitudes, meaning that it preserves the dimension of QS related to the projected qubits.

By taking n as a non-zero natural number, $n \in \mathbb{N}^*$, and $m \in \{1, \ldots, n\}$, an m-qubit projection of a n-dimensional QS (defined by n-qubits) will return a basis partition with 2^m subsets, each one with 2^{m-n} amplitudes, for $n \geq m$, $n, m \in \mathbb{N}$.

The amplitudes of n-dimensional QS are defined by memory values (complex numbers), which are placed by classical basis (memory positions) of 2^n numbers in $\{0, 1, \ldots, 2^n - 1\}$, described binary digits.

Example 5.1.1 *An example of the action of projection operator is considered for a* 3*-dimensional QS, graphically described in Figure 18.*

The 2^3 *amplitudes of the QS were defined by values correspondent to their classical memory positions, represented using* (3) *binary digits, contained in the set* $B = \{000, 001, \ldots, 111\}$*.*

Since each digit can be associated to a classic basis, they were didactically presented by using three colors: red to the first basis, blue to the middle basis and green to the last one.

In this example, the projections are sequential performed over the second (2) and third (3) basis, respectively. As consequence of the action of the projection over B_3 *give us the partition described as the following power-set* B_2*:*

$$\left\{B_{2(0)} = \{000, 001, 100, 101\}, B_{2(1)} = \{010, 011, 110, 111\}\right\}. \tag{19}$$

In sequence, the action of projection M_2 *in Eq.(19) results on the new power-set* B_{23}

$$\left\{B_{2(0),3(0)}=\{000, 100\}, B_{2(0),3(1)} = \{001, 101\}, B_{2(1),3(0)}=\{010, 110\}, B_{2(1),3(1)}=\{011, 111\}\right\}.$$

This partition on classical basis will be reported as code memory-positions.

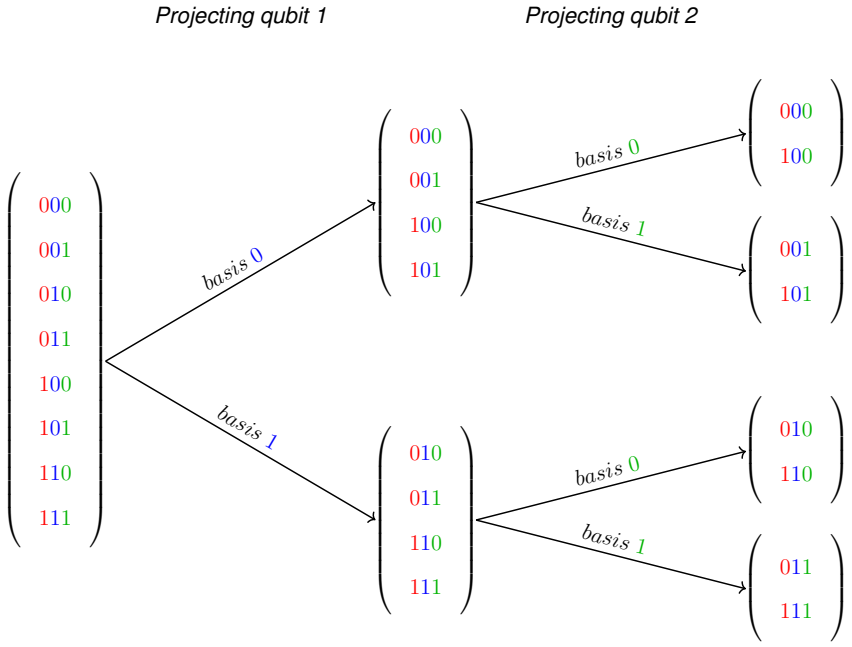


Figure 18 – Projection of qubits 1 and 2 of a 3-dimensional quantum state.

5.2 Matrix-structure of quantum transformation projections

In this proposal we propose the projection of a QT regarding its basis, which are obtained by two approaches:

Write Basis (WB), regarding the matrix lines whenever the WB value is associated to the QS basis it computes; and

Read Basis (RB), regarding the matrix columns, where the RB value is related to the QS basis it uses for the computation.

Single-qubit quantum operators can be classified into 3 types according to their non-zero values, their projections are shown in Figure 19. As can be observed, a operator generates 4 projections with distinct combinations of WB and RB values. Projections that have only zeros are called null-projections since they do not imply in any computation, and therefore can be discarded. For dense operators, each WB is associated to two RB. Meanwhile, for sparse operators, each WB is associated to only one RB.

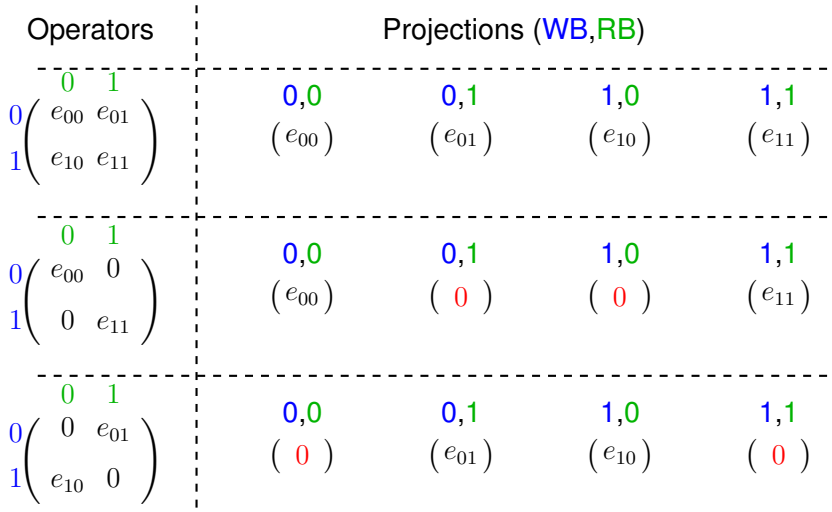


Figure 19 – Quantum operators projection.

Figure 20 shows a example for a generic 3-qubit QT having one basis (second) being projected. The number of projections generated grows exponentially with the number of basis being projected.

For QT derived from single-qubit quantum operators, instead of performing the tensor product to generate the QT matrix and then perform the projections over it, a QT projection for a given basis can be obtained by individually projecting the operator related to that basis and only then performing the multiplication of the projection values with the tensor product between the other operators.

For example, lets consider the 2-dimensional QT given by $Id \otimes H$ and the projection over the first basis which is related to the Id-operator. The Id-operator projection will results in the following matrices

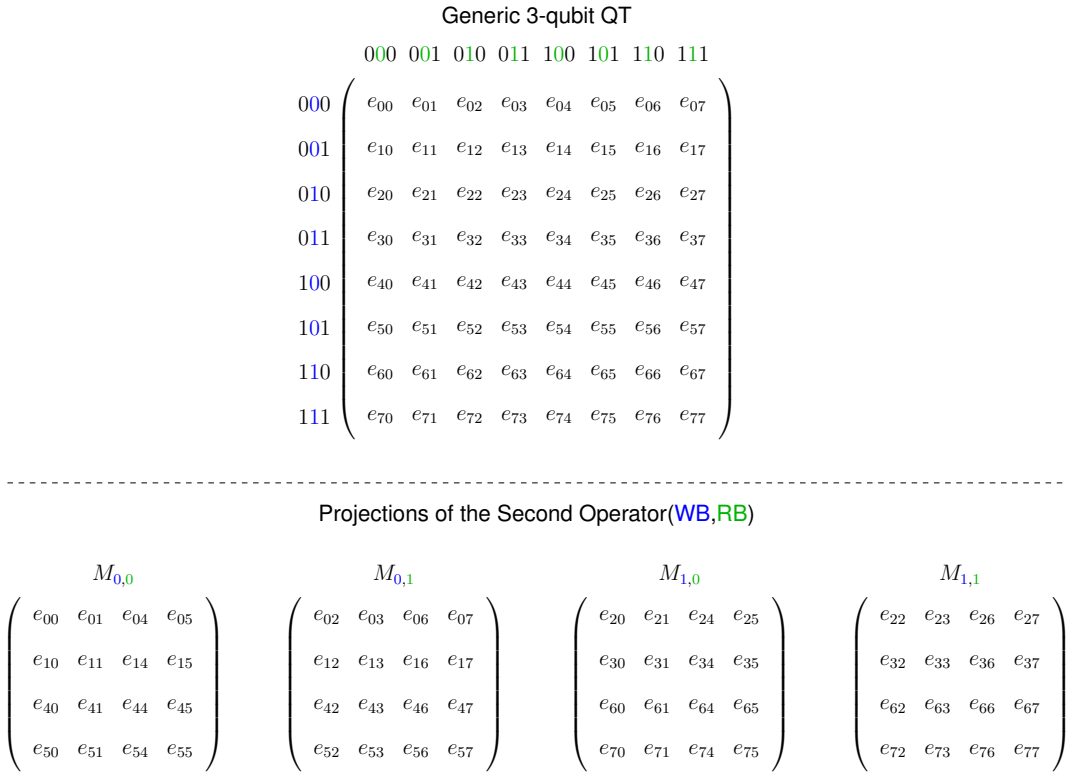


Figure 20 – Generic 3-qubit QT projection.

$$\underset{(\ 1\ )}{0,0} \qquad \underset{(\ 0\ )}{0,1} \qquad \underset{(\ 0\ )}{1,0} \qquad \underset{(\ 1\ )}{1,1}$$

and then each value can be multiplied to the H-operator to obtain the QT projection, resulting in the matrices presented in Figure 21.

$$\underset{\begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{pmatrix}}{0,0} \qquad \underset{\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}}{0,1} \qquad \underset{\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}}{1,0} \qquad \underset{\begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{pmatrix}}{1,1}$$

Figure 21 – Projections of the first basis of the QT $Id \otimes H$.

5.3 Computations with projections over matrix-structures

Given a QT and a QS projected on related qubits/operators (same basis), to apply the QT on this QS you have to compute the multiplication between each QT projection and the correspondent QS projection (defined by the RB value) and them sum of the results associated to the same WB to have the QS resultant for that basis.

Figures 22 and 23 shows an example for a generic 2-qubit system having the first qubit/operator projected. First the structures are projected, Figure 22, and then the computations followed by the reconstruction of the QS are performed, Figure 23.

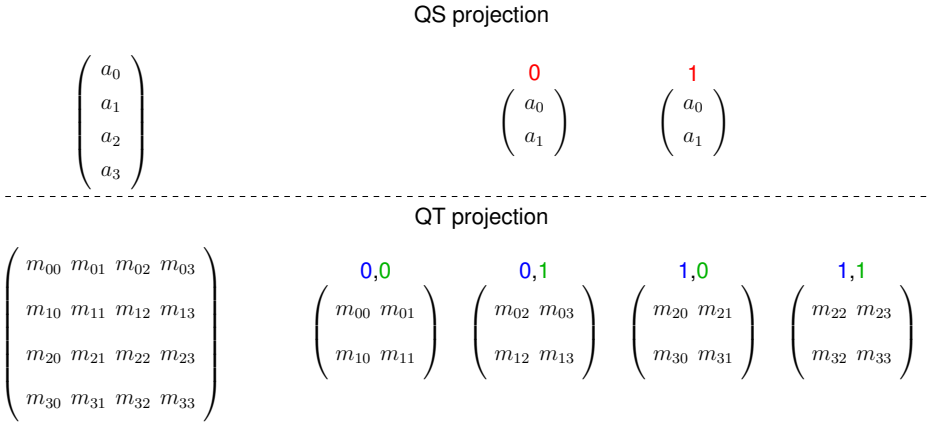


Figure 22 – Structures projection for a generic 2-qubit system.

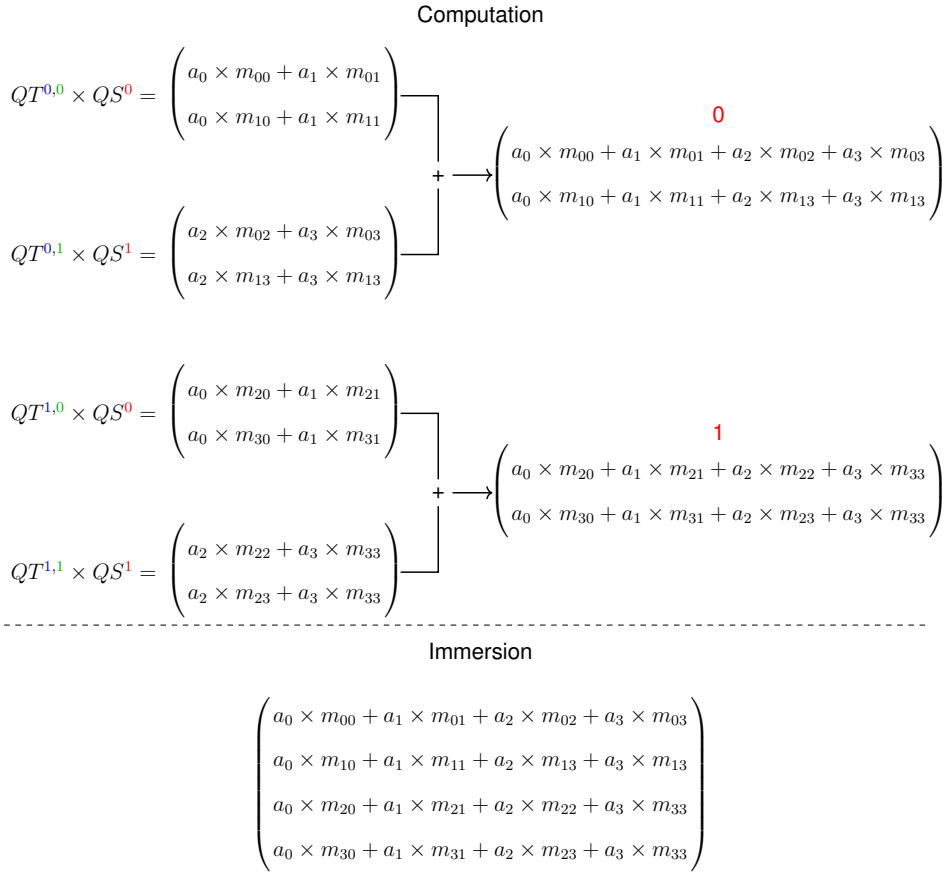


Figure 23 – Computation of a generic 2-qubit system.

Example 5.3.1 *Let* $|001\rangle$ *be an initial QS. The Hadamard operator* H^3 *performed on*

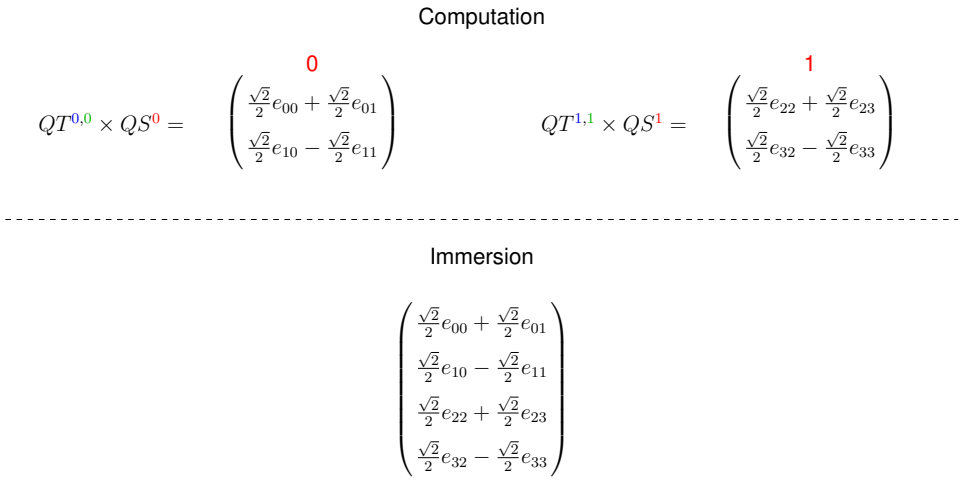


Figure 24 – Computing a 2-qubit system.

$|001\rangle$ *results on the superposition state:*

$$|\phi\rangle = \frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \frac{1}{2\sqrt{2}} \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} \tag{20}$$

In this case, if the following matrix-structure is considered:

$$M_{0,0} = M_{0,1} = M_{1,0} = M_{1,1} = \frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \tag{21}$$

By applying a partition as presented in Example 5.1.1, Eq.(19), over the quantum state $|001\rangle$, we can obtain the related partition on the resulting QS $|\phi\rangle$, as follows:

$$\begin{aligned} M_{0,0}|001\rangle_{1(0)} + M_{1,0}|001\rangle_{1(0)} &= |\phi\rangle_{1(0)} \\ M_{0,1}|001\rangle_{1(1)} + M_{1,1}|001\rangle_{1(1)} &= |\phi\rangle_{1(1)} \end{aligned}$$

5.4 Summarising

In this chapter, first subject addressed was the matrix structure of the projection operators, briefly describing their dynamic construction and step structure to consolidate the HybriD-GM conception. We also consider the action of projection operators on quantum computing structures, reflecting on computations performed by projected structures. Including illustrated exemplifications for such constructions.

6 HYBRID-GM PROPOSAL: ARCHITECTURAL MODEL

In this chapter, the main concepts structuring the architectural model of the HybriD-GM model are exposed, displaying the architectural level structures of the model, which includes detailed insights mainly related to the preprocessing structures, projection layers, execution layers and also, its related extended execution approaches.

6.1 Structuring the HybriD-GM model

The HybriD-GM model explores the projection of quantum states and transformations to control the distribution and granularity of computations while optimizing hardware resources. This work focuses on hybrid architectures dealing with CPU and GPU on a single machine, but the model has a flexible structure enabling extensions to other architectures.

In Figure 25, the architectural structure of the HybriD-GM model is graphically presented, and its main components levels are introduced.

6.1.1 Functionalities of component levels on the HybriD-GM model

In sequence, general functionalities of each level are summarized.

Preprocessing: The preprocessing goal is to receive the application's input data, simulation type, quantum algorithm and quantum state, converting them into structures that can be manipulated by the subsequent level of the model.

Projection Control: This level manages and performs projection operators considering the configurations in the two next levels to define how they need to be designed and executed. Both SEQ and PAR constructors can perform this task; the former is responsible for support of sequential projections and the later, of parallel projections.

Projections Layer: The configurations of projections are defined in this level, identifying specifications of granularity and coalescence for various projection situations,

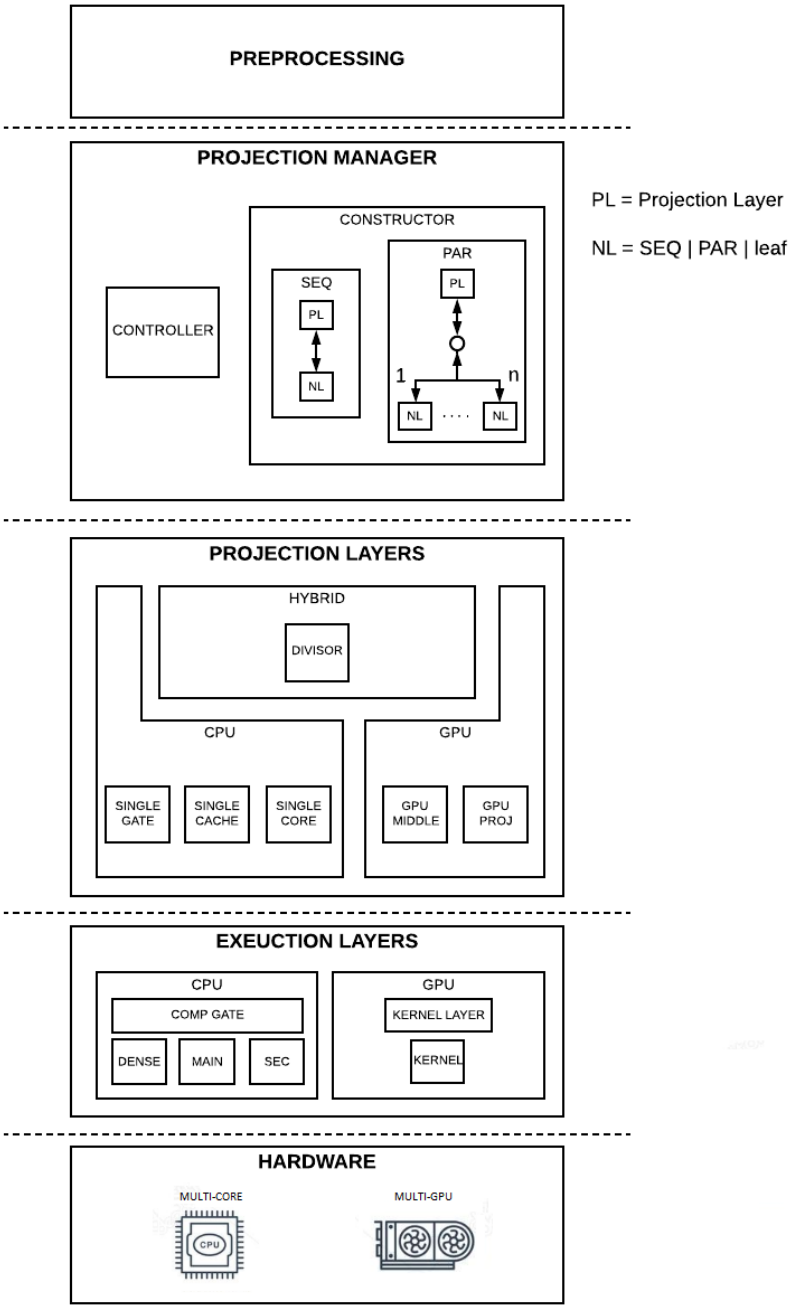


Figure 25 – Projection model overview.

considering each hardware structure available. It can model hybrid or non-hybrid computations.

Execution Layers: This level contains all the operators and functions for computing projections over each hardware structure available. For that, it considers all information related to the hardware to optimize its resources.

Hardware: The infrastructure level contains the target hardware of executions integrate both multi-core and/or multiprocessor as CPU and/or GPU architectures, according to each “type” hardware and application specifications.

6.1.2 Data-structures of component levels on the HybriD-GM model

And now, the main data-structures concerned to the model architecture of HybriD-GM is described, containing data regarding quantum state, quantum transformations and projections as listed below:

Projection Structure, module of data structure to support the following action: (i) containing quantum state amplitudes which can be stored directly or indirectly; (i) defining the quantum state dimension; (iii)including the number of additional qubits related to the projections in case of indirect representation; (iv) identifying the qubits related to a selected projection.

Projection Instance, as the data structure module: (i) providing reference to the generated projection structure, (ii) enumerating the projected basis values.

Gate Structure, as the data structure module: (i) containing the Matrix construction; (ii) informing the target qubit; and (iii) enumerating the control qubits and corresponding values, if any.

In the following subsection 6.2, relevant characteristics of five levels of HybriD-GM architecture are described, focusing in the dynamic of computations. The projection-trees settings for distinct scenarios of simulation are presented in Section 6.3.

6.2 Main level components of the HybriD-GM model

See below, the five level in the architectural structures of the HybriD-GM.

6.2.1 Preprocessing in HybriD-GM model

This level of the of the HybriD-GM model receives the input data for simulation which are the quantum application (quantum circuit), the quantum state and the type of simulation. These data are converted into structures that can be used by the next levels, carried out in two steps described below:

Decomposition of the quantum application: In this level, a quantum application is represented as quantum circuit and defined by composition of quantum transformations can be decomposed into unitary gates, converted to gate structure data and stored in a list preserving their occurrence order.

Conversion of the Quantum State: In this level, the vector representing a quantum state of a quantum application can be converted to a projection structure containing a single instance in order to suit the input patterns, which will be performed by the next levels of the model.

Even other optimization steps can be easily added to this model. See, for instance, the concept of “fusion gates" as applied in (WECKER; SVORE, 2014) decreasing the quantity of quantum transformations, or even the notion of “list restructure” similar to (HANER; STEIGER, 2017), optimizing the projection steps, maintaining the execution consistence and resulting on the reduction in number of distributions.

After performed all preprocessing steps, the obtained results containing the list of “gate structure” and “projection structure” data are passed to the next HybriD-GM model level named as Projection Manager.

6.2.2 Projection Manager Structure

This level of the HybriD-GM model manages and performs the projections according to the quantum application. The control module is responsible for carrying out the projections in multiple layers, controlling the granularity of the computations as well as optimizing the hardware resources.

6.2.2.1 Project manager methodology

The main builders in the project manager are used to define the graph data structure as a tree describing projections/computations, and are described in the following:

- SEQ - to a single node, called a sequential projection;
- PAR - to multiple nodes called as parallel projections and having an intermediate step to manage/synchronize the immediate descendent nodes.

The nodes presented in those builders have the following definition:

- PL - defining the projection layer, which are conceived as projection nodes;
- NL - indicating the next projection layer, which can be related to one of the two builders type, SEQ and PAR, adding another projection layer; as well as an execution node (leaf) ending the branch of a projection tree.

In the manage constructor structure, projection-nodes included in the projection layer level provide useful information to perform projections and execution layer level nodes to perform the computations of a projection. So, intermediate-nodes are responsible for projections and leaf nodes are responsible for executions.

In addition, see the following comments listing other main nodes characteristics in the four graph structures:

Node-structure receive a projection structure and a list of gate structures to be projected or computed;

Root node receive the structures created in the preprocessing level;

Children nodes always have a smaller granularity than their parent node, meaning that the lower the level in a branch-tree, the smaller the projection size they will work on;

Sibling nodes, from the same parent node, will have the same granularity, but adjacent nodes belonging to another parent may have another.

Figure 26 illustrated the flow diagram for a projection node, which receives a projection structure and a list of gate structures.

The control module performs a loop requesting projection instances for the projection structure. Thus, for each instance perform another loop that, according to the configurations defined to the node, on each iteration defines the next set of qubits (basis) to be projected and generate two structures using it:
(i) **projection structure**, from the projection instance, and
(ii) **sub-list of gate structure**, from list of gate structures.
And then, these structures are passed to NL (Next Layer) awaiting the return to move on to the next iteration. After all instances have been computed the parent-node is notified, concluding the control module execution.

6.2.2.2 Projection qubits selection methodology

Let m be the number of qubits to be projected and n be the number of qubits of the quantum state. Then, the selection of the qubits to be project can be performed in distinct approaches described as follows:

Interval approach, where the qubits selected for projection can be defined as an m-dimensional interval considering two options:

1. **Static structured**, selected from a previously defined set of intervals, that together cover all n qubits and, therefore, ensure that all quantum gates will be associated to at least one interval. For example: taking intervals as

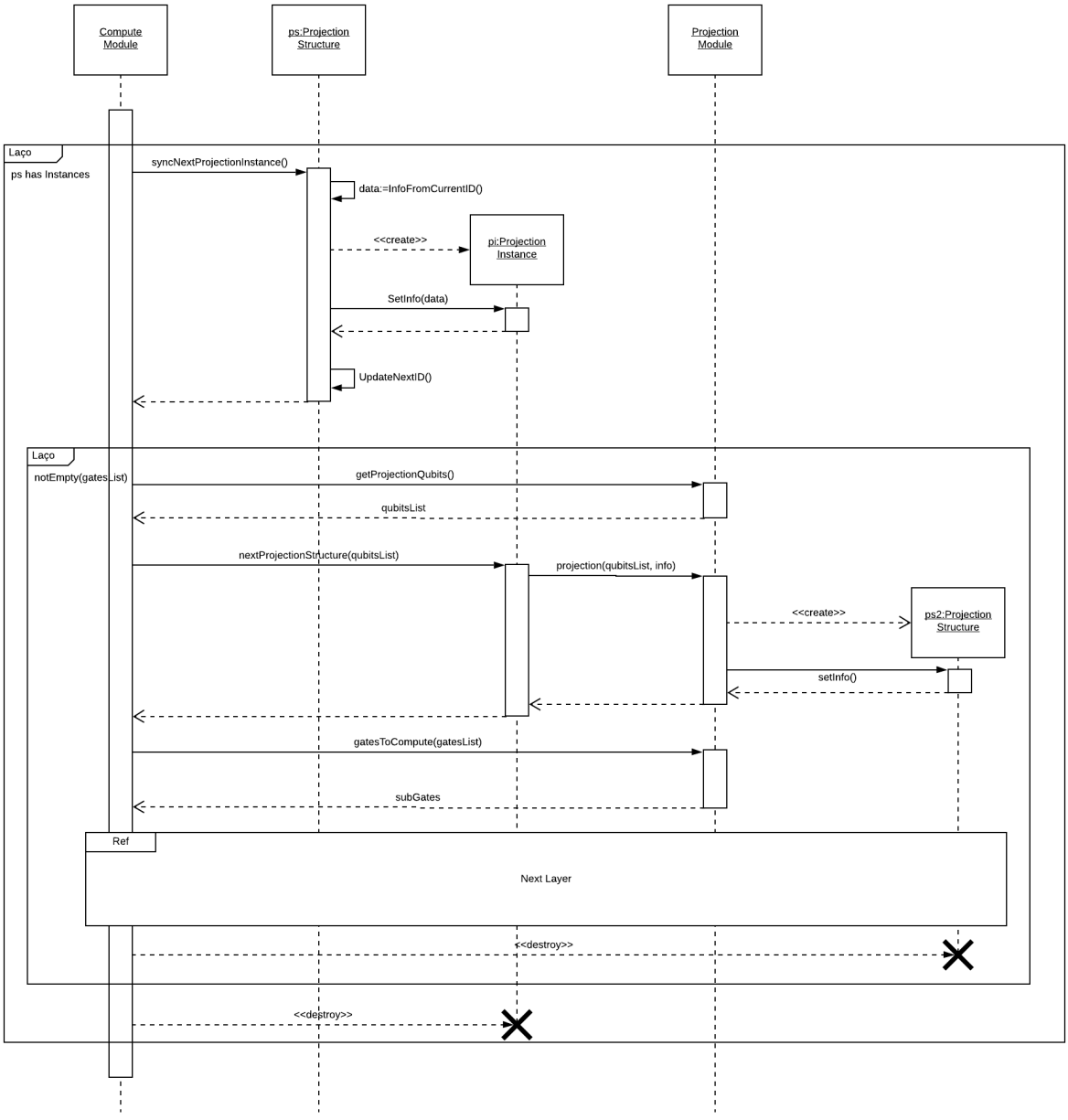


Figure 26 – Generic projection layer

follows: $[0; m-1], [m; 2m-1], \ldots, [n-m : n-1]$, which can be selected alternated and repeated until all operators have been computed;

2. **Adaptable structured**, the information of the next operators in the list of gate structures are used to define the interval, by adding their target qubits to a set until the interval between the smallest and the largest number not exceeds the limit of m qubits.

Dynamic approach, where set of qubits can be defined dynamically, which allows only qubits with operators to be chosen for the next projection step. The non-executed operators can be iterated and non-duplicated target qubits are added to a set until its size reaches m.

Fixed approach, where projections are always performed over predefined set of c qubits, with $c < m$, implying the need for a combination with one of previous approaches to cover select m qubits in total. After disregard those c qubits as selection options, the other approaches can be performed to select the remaining $m-c$ qubits. When fixing the first c-qubits of a quantum state, implies that all projection instances will have contiguous segments with a minimal of 2^c amplitudes.

6.2.2.3 Project quantum state methodology

Now, two types of projections are considered in the HybriD-GM model regarding quantum states on projection structures, and are described in the following:

Direct approach, where amplitudes corresponding to a given projection instance are copied/transferred to a new memory space. Therefore, information from previous projections does not have to be carried out in order to further projections/computations be performed, and consequently can be interpreted as a root-projection. Mainly related to projections from one memory architecture to another but nothing prevents it to be applied in the same architecture. It is used in the context of this work to project from CPU RAM to GPU RAM and GPU RAM to GPU shared memory, but it is not limited to that, can also be used to perform projections of HARD DISK to CPU RAM, or between nodes in a cluster for example.

Indirect approach, where the projection state containing the projection instance is passed as a reference, implying the need to use information regarding the projection instance to determine the access of its amplitudes. When performing successive indirect projections, is necessary to have the combined information of previous projections in the branch-tree up to the closest that can be interpreted as a root-projection. Its is mainly related to projections in the same memory architecture, being able to prevent intermediate copies of the projections from being performed when it is not necessary.

6.2.2.4 Project quantum transformation methodology

In sequence, two main characteristic addressing the project quantum transformation methodology are briefly described:

- After defining the projection qubits, operators acting on non-projected qubits are transferred to another list of gate structures as long as they can be executed preserving the computation consistency.
- Passing by reference can be applied if the state projection performed was indirect, since the acting and control qubits remain the same. Otherwise, it is necessary to copy/transfer the operator and map their target and control quits to match their correspondent in the new quantum state, preserving the node-independence related to previous projection information in tree-branch modeling the application.

Example 6.2.1 *As an example, consider the 6-qubit QA in Figure 27 containing 5 QT, lets define two sets of qubits for projections* $[0-2], [3-5]$*.*

The only dependency present in the QA is the controlled transformation in the third QT, which implies that all quantum gates on QT 1-2 have to be executed before the operators in QT 4-5 to maintain the result consistency. So, the best projection sequence would be to first project $[0-2]$ *and execute the gates in QT 1-2 for that interval, allowing to then project* $[3-5]$ *and execute all gates in that interval to finally project* $[0-2]$ *and execute the remaining gates in QT 4-5.*

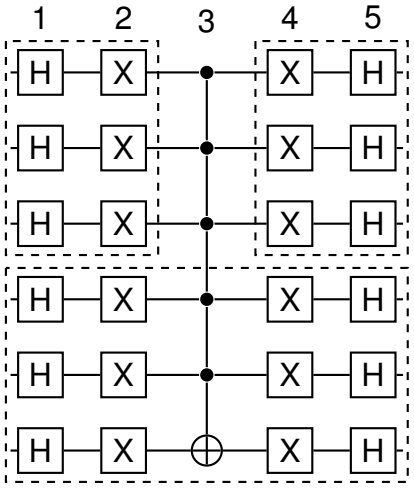


Figure 27 – Diffusion operator with 6-qubit.

Optimizations of quantum transformation projections were not included at the current state of the model. So instead of having 3 steps of execution in the example above, would be necessary 8 steps to perform the same QA.

6.2.3 Projections Layers

The projections layers level of the HybriD-GM model contemplate three categories of projections in the contexts of hybrid architectures, related to the type of simulation: CPU, GPU and Hybrid approaches. Wevare focusing to optimize hardware resources for various projection scenarios by defining setup regarding:

1. Granularity;
2. Coalescence;
3. Selection form of projection qubits; and
4. Type of quantum state projections.

Such information for each scenario are detailed in the following:

1. **CPU projections** required for CPU simulations:

   Single gate, this step has 1 of granularity and no coalescence, the target qubit of the first operator in the list of gate structures is used to determine the projection qubit and this operator is also project individually.

   Single cache, where granularity and coalescence are defined w.r.t. the memory space associated to the projected quantum state and its chunks of subsequent amplitudes in a way to not exceed the sizes of the last cache level individual to each core and the first cache level, respectively. Uses the fixed and the dynamic approach to select the projection qubits;

   Single core, where the granularity and coalescence are always greater values then the ones for Single Cache, with values depending on the type of simulation.

2. **GPU projections** required for GPU simulations

   GPU MIDDLE, granularity limited to not exceed the GPU memory size, and coalescence to have the memory space of subsequent amplitudes equal or higher then the minimum memory transaction size between CPU and GPU;

   GPU PROJ, granularity defined to not exceed the GPU shared memory per thread block, and coalescence to have chunks of subsequent amplitudes matching the transaction size between the GPU global memory and shared memory.

3. **Hybrid projections** allowing hybrid simulations in CPU and GPU, encapsulating the projection configurations defined in the above categories as well as the follow configuration:

DIVISOR, were granularity will be a equal or higher value, and coalescence a equal value to its child nodes maximum values.

The quantum state in the projection-structure is indirectly projected for the above scenarios, except to the GPU MIDDLE category, which performs a direct projection transferring it to the GPU memory.

6.2.4 Execution Layers

The execution layers level of the HybriD-GM model contemplate two execution categories, CPU and GPU, which are described in the next subsections.

6.2.4.1 CPU Layers

Preliminary results performing computations using a generic projection size showed the best results when resulting on a quantum state with granularity of 1 and 2. Hence, was opted to use at this stage of the work granularity always equal to 1 for layers of execution, implying on a gate by gate computation approach which allows further optimizations according to the types of quantum gates, which can be classified into two classes:

(i) *Dense operators* - defined by matrices without void elements, such as the Hadamard operator; and

(ii) *Sparse operators* - with void elements in the main diagonal, as the Pauly X gate, or in the secondary diagonal, such as the Pauly Y gate.

Thus, see Figure 28 showing the flow diagram describing the **COMP GATE** module, which is responsible for managing the computation of a single gate for a given projection structure.

The control module requires the matrix operator and the type of the gate, following a loop that iterates through all instances of the projection-structure, requesting the pair of amplitudes related to the current instance and invoke the computation module correspondent to the gate type, passing the pair of amplitudes (a_0, a_1) and the matrix (M).

The computation modules for the tree types of gates are: (i) **DENSE** - for dense operators, (ii) **MAIN** - for main diagonal operators, (iii) **SEC** - for secondary diagonal operators. These corresponding computations are described by Algorithms 1, 2 and 3.

As can be observed the computations of Dense and Secondary Diagonal operators has use a temporary variable since the computation is performed in place and the amplitudes are used on each other calculus, and Primary Diagonal and Secondary Diagonal operator needs to perform only 2 multiplications instead of 4 since the multiplications involving the zeros in their matrix could be removed.

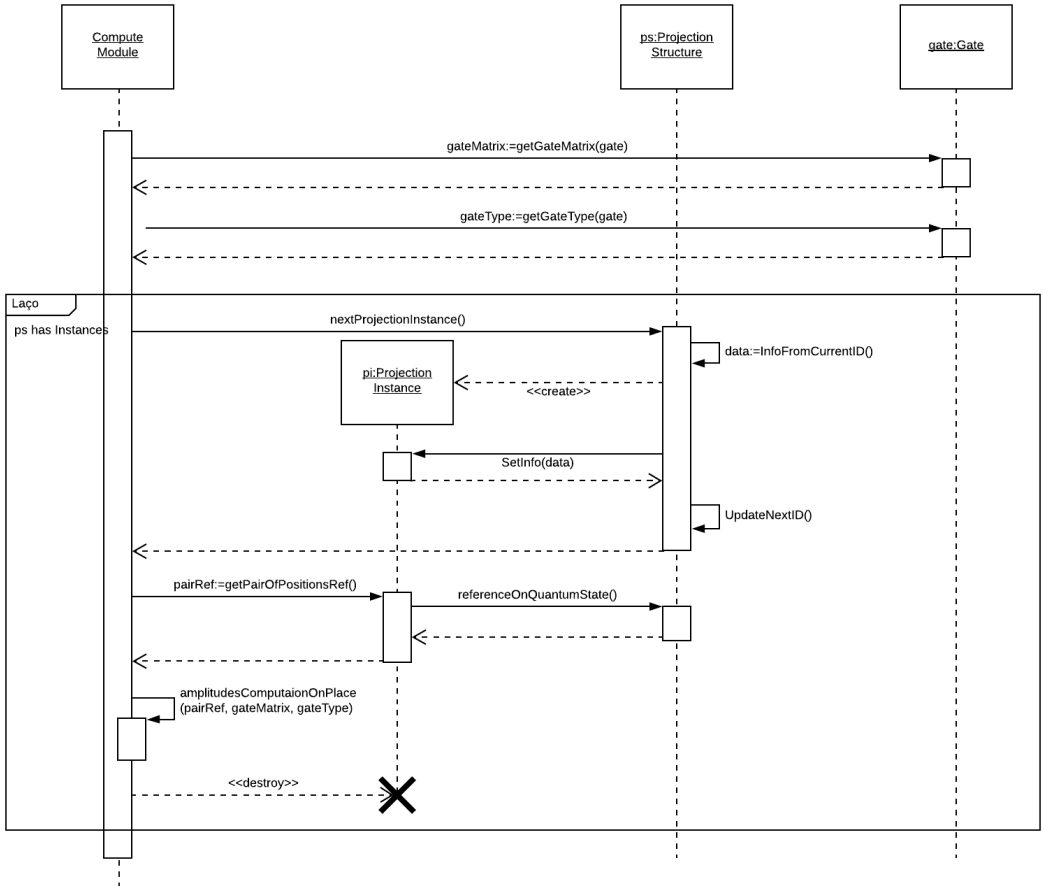


Figure 28 – Flow diagram for the Comp Gate module.

Algorithm 1: Dense

1: $tmp \leftarrow a_0$
2: $a_0 \leftarrow a_0 \times M[0][0] + a_1 \times M[0][1]$
3: $a_1 \leftarrow tmp \times M[1][0] + a_1 \times M[1][1]$

Algorithm 2: Primary Diagonal

1: $a_0 \leftarrow a_0 \times M[0][0]$
2: $a_1 \leftarrow a_1 \times M[1][1]$

Algorithm 3: Secondary Diagonal

1: $tmp \leftarrow a_0$
2: $a_0 \leftarrow a_1 \times M[0][1]$
3: $a_1 \leftarrow tmp \times M[1][0]$

6.2.4.2 GPU Layers

Figure 29 shows the flow diagram for the Kernel Layer responsible for requesting the GPU computation of a projection, already having its quantum state in the GPU's memory. It perform the following steps:

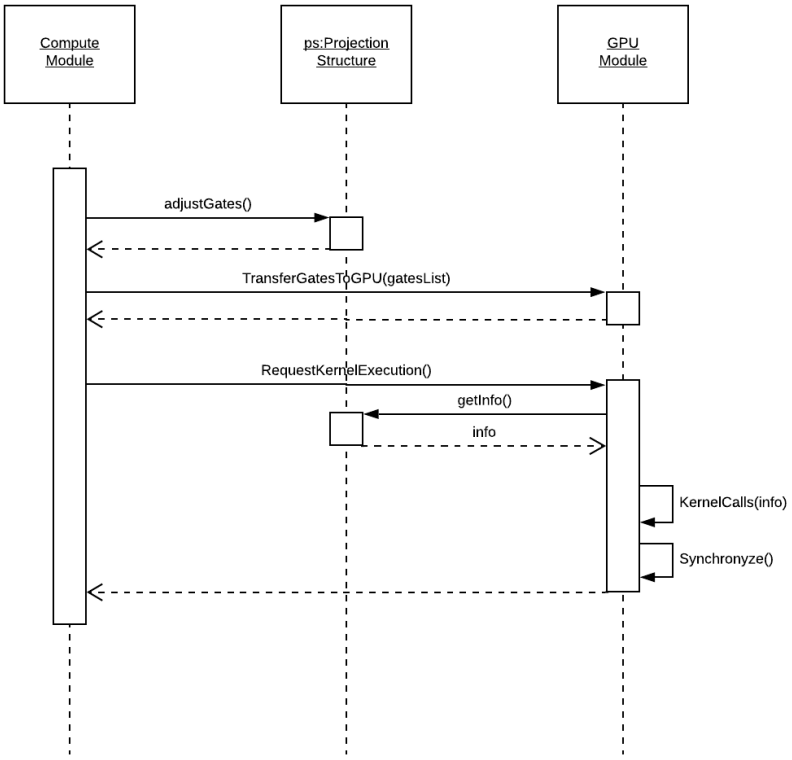


Figure 29 – Flow diagram related to the Kernel Layer.

(i) Firstly, the quantum gates to be computed have their target-qubits and projection-controls mapped and transferred to the GPU.

(ii) In sequence, their execution is requested to the GPU module;

(iii) The GPU module retrieves the projection structure information;

(iv) And then invokes the GPU kernel execution on each device, awaiting for their return.

For multiple GPU executions just homogeneous hardware were considered in the current state of this work.

The execution flow for the GPU kernel is showed in Algorithm 4 considering the point of view of a GPU block, with n_1 representing the number of projection instances associated to the block. Detailed descriptions for a GPU kernel are given in the next chapter.

6.3 Execution Approaches

This section shows how the projection manager combines projection and execution layers to allows the simulation-types considered in this work.

Algorithm 4: Presenting flow data of a GPU kernel in a block point of view.

```
1 Project n_1 instances of the shared memory ;
2 foreach Gate in QuantumGatesList do
3 |   Define the projection p_2 for the quantum gate structure;
4 |   Distributes the the instances from p_2 for block threads;
5 |   Perform thread-computations over their instances of p_2;
6 end
7 Immerse n_1 instances in the global GPU-memory;
```

For all cases, the structures generated by the preprocessing steps are considered as input, that is, a projection structure containing a single instance and a list of Gate Structures representing, respectively, the quantum state and the quantum algorithm to be simulated.

The approach to a single core simulation in the HybriD-GM model can be observed in Figure 30 referred as SINGLE CORE EXEC, containing four layers as described in the following:

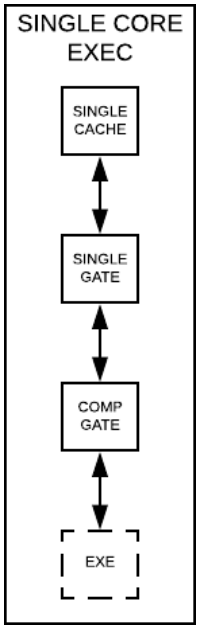


Figure 30 – Layers composition for single core simulations.

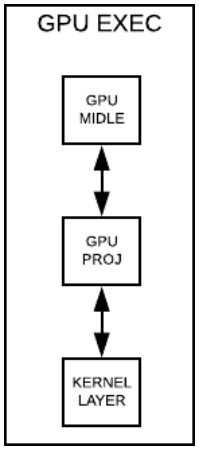


Figure 31 – Layers composition for GPU simulations.

(i) SINGLE CACHE - ensuring that the amplitudes accessed for computing an instance will remain in the cache until the end of the computation;

(ii) SINGLE GATE - ensuring that computations can be carried out gate by gate;

(iii) COMP GATE - generating ordered instances and then ensuring the exploration of the spatial locality of the lowest cache level, since each pair of amplitudes is subsequent to the previous pair;

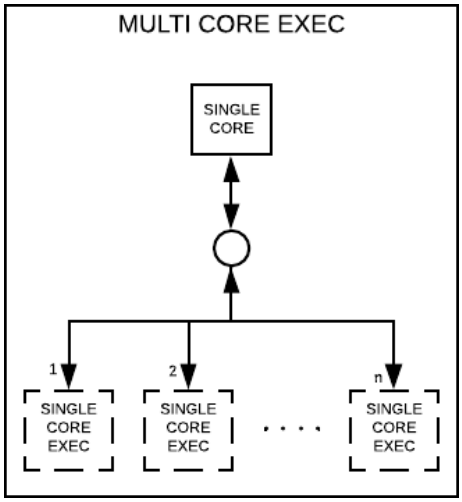


Figure 32 – Layers composition for multi core simulations.

(iv) EXE - this layer refers to one of the follows CPU execution layers, DENSE, MAIN and SEC, depending on the quantum gate-type.

The simulation in a multicore execution approach is presented in Figure 32 referred as MULTI CORE EXEC, containing two steps characterized as follows:

(i) SINGLE CORE - the projection structures generated are passed to n nodes, with n being the number of cores being used in the simulation;

(ii) SINGLE CORE EXEC - each node is this level reports to a single core simulation, since it follows the same flow of projection/computations.

The flow simulation of a GPU approach is presented in Figure 31. This structure is performed based on the next layers:

1. GPU MIDDLE - generating projection structures which match the GPU memory size and transferring it to GPU-memory;

2. GPU PROJ - generating structures compatible to the shared memory size per GPU block;

3. KERNEL LAYER - performing kernel calls to accomplish the computation tasks.

Multiple GPU quantum simulations follow the same approach, but in the first layer the quantum state from the projection structure is transferred in equal parts to each GPU, and their global memory spaces have to be visible to each other.

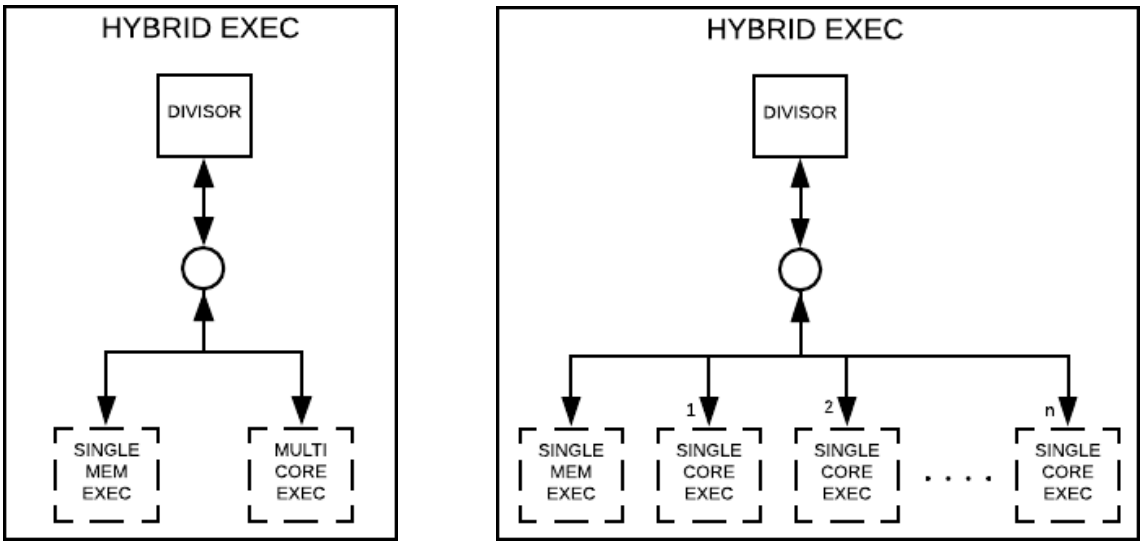
There exist two approaches for hybrid simulations, corresponding to distinct layers, which are considered in two steps, as presented in Figures 33a and 33b.

The first layer consisted by the DIVISOR layer is presented in both approaches providing projection structures for their child nodes, which is what differs in each approach. The second level are described bellow:

(i) consisted by a pair of nodes, given by GPU EXEC and MULTI CORE EXEC; and

(ii) the other approach has $n + 1$ child-nodes referring to GPU EXEC and n SINGLE CORE EXEC.

Those difference in approach's will result will results in different granularity associated to each CPU core.



(a) Approach 1 (b) Approach 2.

Figure 33 – Layers composition for hybrid simulations.

6.4 Summarizing

This chapter describes the structure of HybriD-GM model, considering the level of their structured architecture, Preprocessing, Projection Manager, Projections Layers, Execution Layers also detailing how those the former tow Layers are combined to construct the types of simulations considered in this work.

7 HYBRID-GM PROPOSAL: EVALUATING APPLICATIONS

The HybriD-GM model proposed in this work can be applied to any of the simulators presented. For validation and evaluation, an extension of the D-GM framework, presented in the section 3, was implemented. This extension occurred as follows:

- Addition of a module with the structures and functions present in the model that refer to the representation and manipulation of projections, it should be noted that all logical functions were implemented using bitwise operations to minimize their overhead;
- Re-implementation of computing functions in CPU and GPU following the HybriD-GM model, referring to what was presented in Figures 32 and 31 respectively;
- Implementation of the Hybrid simulation approach combining CPU and GPU, we chose to follow the form described in 33b as it does not have nested parallelism and therefore facilitates the implementation.

The next section will describe the GPU kernel implemented, and the following section present the results achieved.

7.1 GPU Kernel

The CUDA kernel is responsible for the execution of a projection structure, and its characteristic, functionalities and computation methods are described in this section. The quantum state and the list of gate structures are already in the GPU memory when the execution is requested.

The CUDA kernel computation may be divided into 3 steps, as described below:

Step 1: Identify each block's projection instances and copy their amplitudes to the shared memory

In order to perform step 1, first each thread determines the projection instances basis using its CUDA block unique identifier. Then, for each projection instance, each thread copies two of its amplitudes from the state in global memory to

shared memory. Since they are accessed for the execution of each operator, keeping it on the GPU's global memory would have a very negative impact on performance.

The pseudo-code for this step is shown in Algorithm 5.

Algorithm 5: GPU Kernel Step 1.

```
1  bId ← GETBLOCKID();
2  thId ← GETTHREADID();
3  foreach s in BlockProjInstances do
4  |  piId ← GETPROJINSTANCEID(s, bId, ps);
5  |  s[thId] ← state[GETGLOBALPOS(piId, ps, thId)];
6  |  s[thId + BS] ← state[GETGLOBALPOS(piId, ps, thId + BS)];
7  end
8  SYNCTHREADS()
```

Step 2: Execution of the list of gate structures This step goes through the list of quantum gates and performs their execution, at each iteration each CUDA thread projects a pair of amplitudes according to the target qubit of the current quantum gate. Then, it calculates those positions for all sub-states (if conditions defining the quantum gate controls are satisfied). Barriers are used to synchronize these threads within the same block before passing to the next quantum gate, guaranteeing that all amplitudes on the shared memory are up-to-date.

The pseudo code for this step is shown in Algorithm 6.

Algorithm 6: GPU step 2

```
1  foreach gate in GatesList do
2  |  p0, p1 ← GETPINSTANCEPAIR(thId, gate.qubit) ;
3  |  foreach s in BlockSubStates do
4  |  |  if MATCHCONTROLS(p0, gate) then
5  |  |  |  tmp ← s[p0] × gate.matrix[0][0] + s[p1] × gate.matrix[0][1];
6  |  |  |  s[p1] ← s[p0] × gate.matrix[1][0] + s[p1] × gate.matrix[1][1];
7  |  |  |  s[p0] ← tmp;
8  |  |  end
9  |  end
10 |  SYNCTHREADS();
11 end
```

Step 3: Update the quantum state.

After calculating all quantum gates, the result stored in the shared memory are copied to the quantum state on the GPU global memory.

The pseudo code for this step is shown in Algorithm 7.

Algorithm 7: GPU step 3

```
1 foreach s in BlockProjInstances do
2 |   piId ← GETPINSTANCEID(s, bId, ps);
3 |   state[GETGLOBALPOS(piId, ps, thId)] ← s[thId];
4 |   state[GETGLOBALPOS(piId, ps, thId + BS)] ← s[thId + BS];
5 end
```

The auxiliary functions used on the GPU pseudo-codes are described in the following:

- GETBLOCKID - providing the unique ID of the CUDA block;
- GETTHREADID - getting the ID of the CUDA thread within the block;
- GETPINSTANCEID - obtaining the ID for a given projection instance using the block ID and projection structure information;
- GETGLOBALPOS - obtaining the position of an amplitude in the global memory corresponding to a specific position in projection instance.
- GETPINSTANCEPAIR - providing a closed pair of positions in a gate projection structure for a given thread ID and target qubit;
- MATCHCONTROLS - verifying if a position satisfies the controls of a quantum gate.

In the next section, will be described and discussed the results obtained by simulating Shor's and Grover's algorithms in the extension of the D-GM framework.

7.2 Results

All tests on this paper were performed on a desktop with an Intel Core i9-7900X processor with 10 cores, 32 GB RAM, and 2 NVidia GTX Titan X GPU. The experiments were executed over Ubuntu Linux version 17.04, 64 bits, and CUDA Toolkit 9.0. Average simulation times were calculated from 30 executions, and all cases presented a standard deviation smaller than 1%.

Tables 2 and 3 shows LIQUi$|\rangle$ and ProjectQ parallel performances for simulations of Shor's and Grover's algorithm in the architecture mentioned above, those results will be used later to make comparisons with the HybriD-GM results. Other simulators were not used because they are not available.

7.2.1 CPU Results

In this subsection is presented the results considering simulations only in CPU, both algorithms were simulated sequentially and in parallel on CPU using the previous

Table 2 – LIQUi|⟩ and ProjectQ simulation times for Shor's algorithm, in seconds.

Qubits	LIQUi|⟩	*ProjectQ*
15	16.90	1.43
17	54.91	8.15
19	77.22	16.00
21	430.39	66.27
23	2192.49	574.786

Table 3 – LIQUi|⟩ and ProjectQ simulation times for Grover's algorithm, in seconds.

Qubits	LIQUi|⟩	ProjectQ
15	22.93	1.82
17	67.78	4.93
19	331.60	24.29
21	1879.58	656.16
23	–	5664.10

version of the D-GM and the extension HybriD-GM, varying the number of threads from 1 to 10 and considering a range of qubits from 15 to 25.

7.2.1.1 Shor

Table 4 shows the execution times obtained for simulations of the Shor's algorithm in the previous version of the D-GM.

Table 4 – Simulation times for Shor's algorithm over CPU for previous implementation, in seconds.

Qubits	Seq.	1 Thread	2 Threads	4 Threads	8 Threads	10 Threads
15	4.66	4.66	2.44	1.29	0.70	0.59
17	30.24	30.24	15.65	7.96	4.14	3.38
19	186.81	186.81	96.52	48.98	24.85	20.06
21	1108.21	1108.21	573.53	290.98	157.48	132.35
23	13223.30	13223.30	6348.58	3276.57	1678.93	1134.27
25	35278.31	35278.27	18142.40	9290.41	6403.29	6000.35

Table 5 shows the execution times obtained for Shor's algorithm in the HybriD-GM. The speedups over the senquential simulation are shown in Figure 34, as can be seen, all simulations with multiple threads showed performance gains, presenting better scalability for simulations with a higher number of qubits reaching $9.18\times$ for a 21 qubit simulation. This occurs because simulations with 15 and 17 qubits have a very low execution time due to the small size of the quantum state (256 and 1024 kilobytes), and thus operations other than amplitudes computations have a impact in the total time of simulation.

HybriD-GM speedup over the previous D-GM are presented on Figure 35, showing superior performance for all simulations increasing with the number of qubits and

Table 5 – Simulation times for Shor's Algorithm over CPU for HybriD-GM, in seconds.

Qubits	Seq.	1 Thread	2 Threads	4 Threads	8 Threads	10 Threads
15	0.64	0.64	0.37	0.21	0.14	0.14
17	3.99	3.99	2.12	1.14	0.64	0.57
19	23.87	23.87	12.38	6.40	3.35	2.79
21	140.20	140.20	71.09	36.17	18.98	16.12
23	1138.27	1138.27	574.74	288.99	149.10	123.86
25	6320.18	6320.18	3167.27	1590.55	831.10	696.75

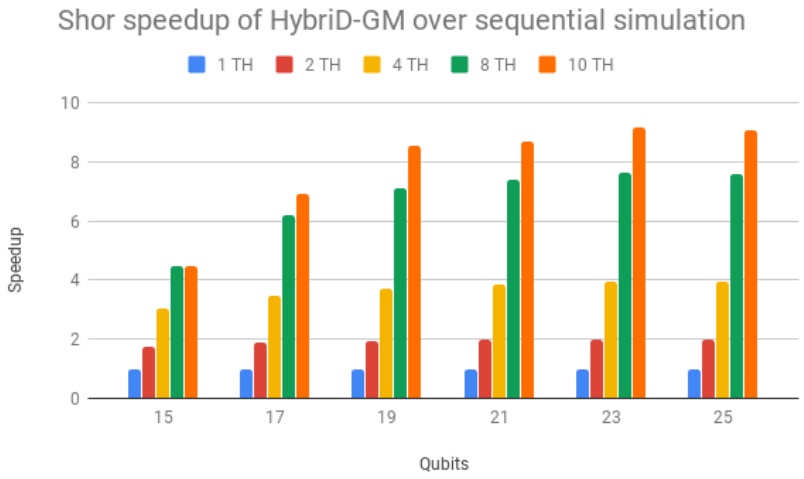


Figure 34 – Shor's algorithm speedups for HybriD-GM parallel simulation over sequential simulation.

achieving performances of $18\times$ faster for 25 qubits simulations.

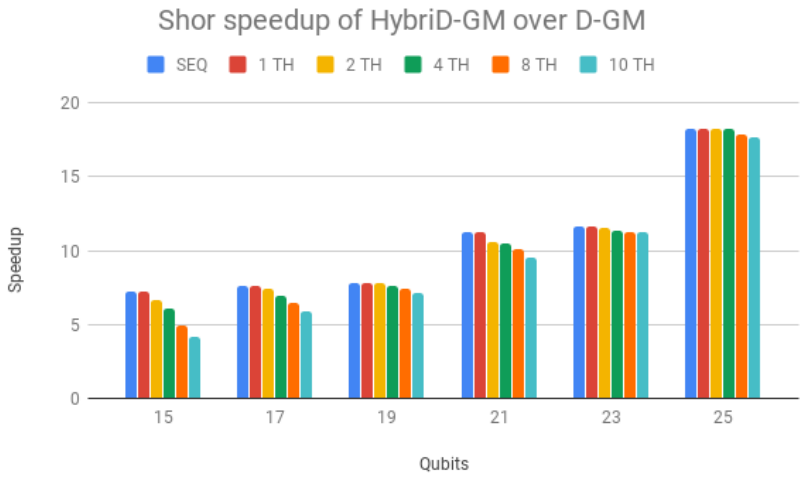


Figure 35 – Shor's algorithm speedup for HybriD-GM over D-GM.

Speedups over LIQUi$|\rangle$ and ProjectQ are shown in Figure 36, using a logarithmic scale for better visualization. Speedups over LIQUi$|\rangle$ started high, $117\times$ for 15 qubits, but declined and stabilized with the increase in the number of qubit, down to $17.7\times$.

For ProjectQ, the speedup presented a similar behaviour but on a small scale, with a maximum of $14.2\times$ with 17 qubits. HybriD-GM presented better performance for all qubits values, achieving higher values for low qubits as a consequence of being more efficient than these simulators for those executions.

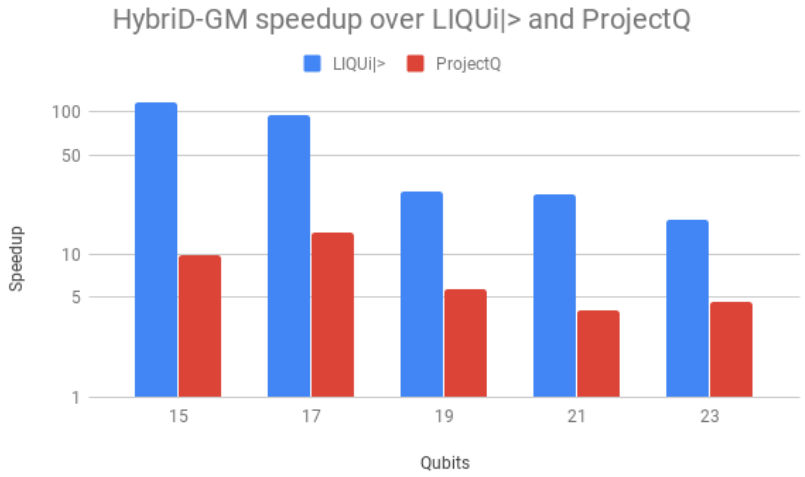


Figure 36 – Shor's algorithm speedup for HybriD-GM with 10 threads over LIQUi$|\rangle$ and ProjectQ.

7.2.1.2 Grover

Table 4 shows the execution times obtained for simulations of Grover's algorithm in the previous version of the D-GM.

Table 6 – Simulation times for Grover's algorithm over CPU for previous implementation, in seconds.

Qubits	Seq.	1 Thread	2 Threads	4 Threads	8 Threads	10 Threads
15	2.04	2.04	1.03	0.52	0.27	0.22
17	18.77	18.77	9.39	4.70	2.37	1.91
19	168.73	168.73	84.59	42.36	21.21	17.00
21	1504.03	1504.03	753.44	378.17	191.32	153.82
23	13199.90	13199.90	6619.73	3324.60	1701.69	1391.06
25	115317.00	115317.00	57759.60	28991.90	14820.80	12288.80

Table 7 shows the execution times obtained for the CPU simulations of the Shor algorithm in HybriD-GM. In the Figure. 37 speedups are presented in relation to sequential simulation. Simulations with multiple threads also showed performance gains for all cases with a scalability similar to Shor's, reaching up to $9.51\times$ for the 21 qubit simulation.

HybriD-GM speedup over D-GM are shown in Figure. 38, presenting a steady scalability for 21-qubit simulations with a performance gain around $7.9\times$.

Table 7 – Simulation times for Grover's algorithm over CPU for new implementation, in seconds.

Qubits	Seq.	1 Thread	2 Threads	4 Threads	8 Threads	10 Threads
15	0.27	0.27	0.15	0.08	0.05	0.05
17	2.43	2.43	1.27	0.66	0.35	0.31
19	21.68	21.68	11.28	5.82	2.97	2.50
21	192.03	192.03	96.52	49.40	24.92	20.19
23	1696.62	1696.62	850.74	434.20	219.12	180.72
25	14778.00	14778.00	7534.92	3780.28	1942.71	1616.72

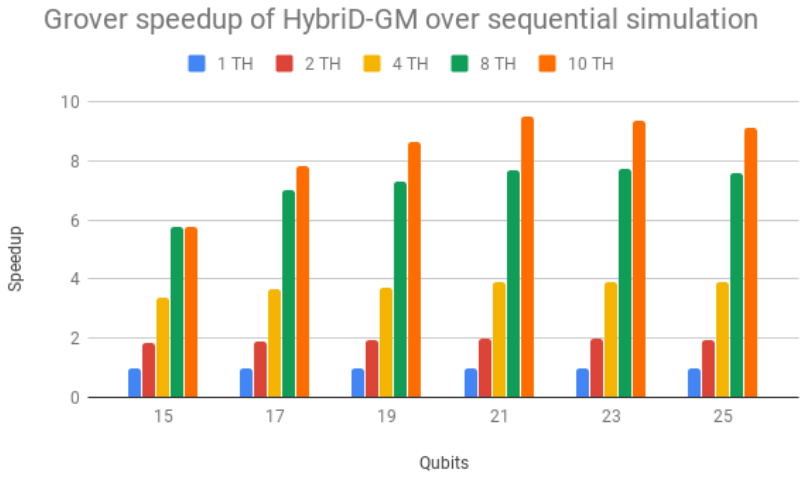


Figure 37 – Grover's algorithm speedups for HybriD-GM parallel simulation over sequential simulation.

Shor's algorithm presented a higher gain in performance cause it is benefited by the most improvements acting on simulating controlled operators, since the Grover's simulation algorithm have less controlled operations.

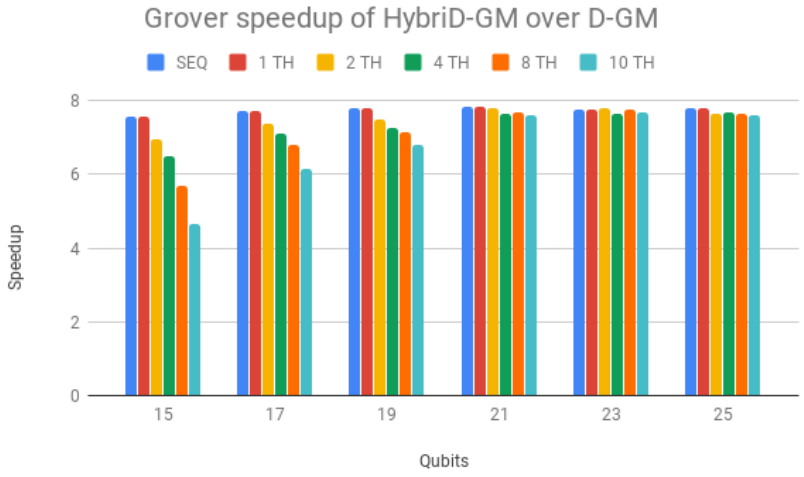


Figure 38 – Grover's algorithm speedup for HybriD-GM over D-GM.

Speedups over LIQUi|⟩ and ProjectQ are shown in Figure 39, using a logarithmic scale for better visualization. Speedup over LIQUi|⟩ also started high, $491.1\times$ for 15 qubits, decaying and stabilizing as the number of qubits is increased, down to $93.1\times$. For ProjectQ, the speedups went down from $39\times$ with 15 qubits to $9.7\times$ with 19 qubits, and then raised up to $32\times$ for 21 and 23 qubits, this implies that ProjectQ loses performance in algorithms that have a little amount of controlled gates. HybriD-GM presented a even better performance than for Shor's algorithm.

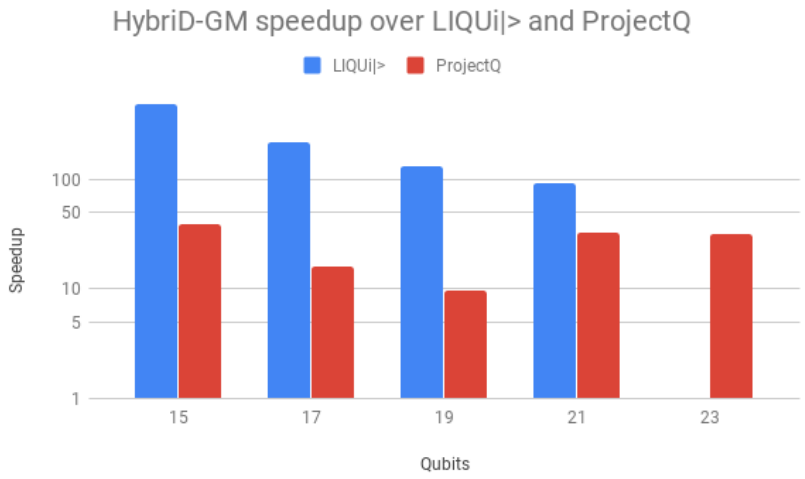


Figure 39 – Grover's algorithm speedup for HybriD-GM with 10 threads over LIQUi|⟩ and ProjectQ.

7.2.2 GPU Results

In this Section, first results with isolated Hadamard gates are presented. Afterwards, Shor's and Grover's Algorithms are simulated and compared with the previous kernel presented in Section 3, then LIQUi|⟩ and ProjectQ. Closing up with a discussion of results.

7.2.2.1 Hadamard gates

In order to understand the impact in performance of coalescence and multiple operators per projection, simulations containing a large amount of Hadamard gates (100,000) were performed. This large number ensures that other factors such as initialization and memory allocation do not impact on the total time.

The Coalescing Factor was varied from 0 to 6 (quantum state projection with 2^0 to 2^6 contiguous amplitudes) and the number of operators per each projection (i.e., per each kernel call) between 10, 50, 100, 150 and 200, for 3 types of execution:

(1) *1 GPU* - using only 1 GPU;

(2) *2 GPU global* - using 2 GPU and gates operating over the last qubit, thus resulting in communication between GPU when doing the steps 1 and 3 of the kernel; and

(3) *2 GPU local* - using 2 GPU and gates not operating over the last qubit, thus not needing communication between GPU.

Simulation results for each of the structures described before are presented in Figures 40 to 44, where each figure presents time in the vertical axis and the coalescing factor in the horizontal axis. Each subsequent figure increases the number of gates per projection. Simulations using controlled gates would show similar results since only target qubits matters for the communication presented on steps 1 and 3 of the kernel.

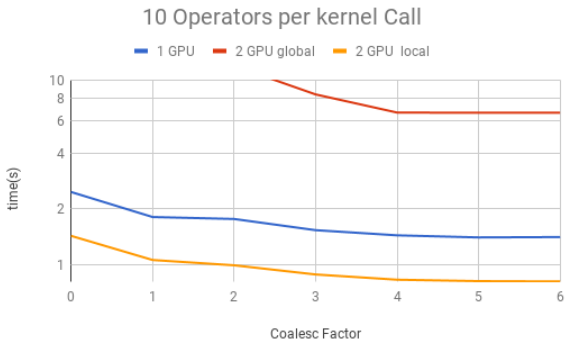


Figure 40 – Projections with 10 operators.

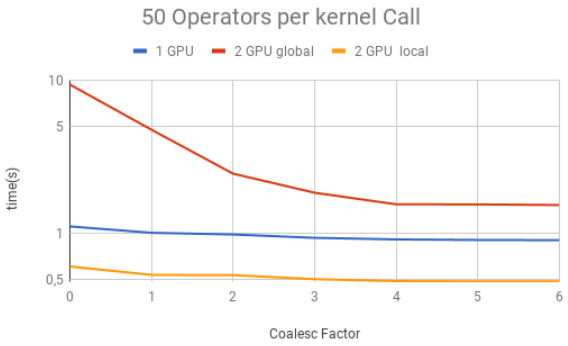


Figure 41 – Projections with 50 operators.

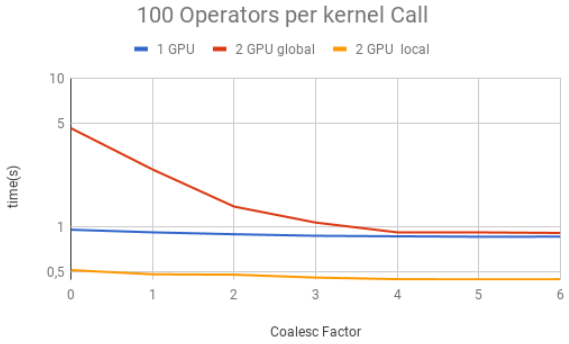


Figure 42 – Projections with 100 operators.

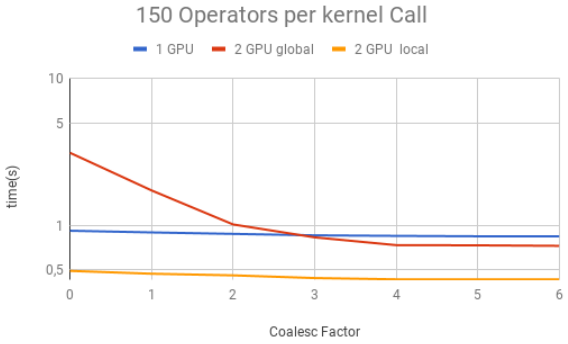


Figure 43 – Projections with 150 operators.

The results for the experiments in Figures from 40 to 44 show that:

(1) For all types, increasing the coalescing factor up to 4 the simulation time decreases but remaining the same after that. Such factor 4 determines that projection instances have 16(2^4) coalesced amplitudes resulting in coalesced memory access of 128(2^8) bytes (8 bytes per amplitude), which is the minimum size per

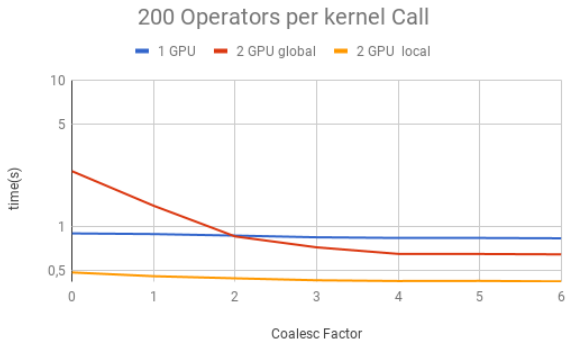


Figure 44 – Projections with 200 operators.

memory request in this hardware, for both its own and to another GPU within a computation node. Thus, any factor smaller than 4 will produce less efficient memory requests; and larger ones will be break into multiple requests;

(2) Two GPU local was $\approx 2\times$ faster than one GPU; this was expected since it has the double of computational power and no communication between GPU is required;

(3) Two GPU global is slower than one GPU for small number of operators per projection, because the communication between GPU has a high cost. From 150 operators per call and coalesced factor 4, it was $\approx 1.3\times$ faster, showing that as long the time spent computing operators (step 2 of the kernel) is high enough it can absorb some of the overhead inserted by the communication between GPU thus achieving some improvement in performance for this type of execution as well;

7.2.2.2 Shor's Algorithm

Simulation times for Shor's algorithm over the D-GM and the HybriD-GM are presented in Table 8. Speedups of the HybriD-GM over th D-GM are shown on Figure 46- showing improvements, for 1 and 2 GPU respectively, up to 35.74$\times$ and 61.90$\times$.

Regarding multi GPU simulations on the Hybrid-GM, executions with 2 GPU did not show performance gain for small number of qubits due to the small amount of computation, but from 19 qubits it started to show growing speedups up to $1.73\times$.

Table 8 – Average simulation times for Shor's Algorithm in seconds.

Qubits	*Previous*	*1 GPU*	*2 GPU*
15	0.76	0.081	0.088
17	2.48	0.198	0.174
19	11.83	0.571	0.492
21	63.84	2.116	1.480
23	363.39	11.01	6.648
25	2015.53	56.38	32.56

SL - Simulator Limit.

7.2.2.3 *Grover's Algorithm*

Simulation time for Grover's algorithm using HybriD-GM and D-GM can be seen in Table 9.

Speedups of the HybriD-GM over the D-GM are shown on Figure 45, also showing improvements up to 38.32× and 31.76× faster for 1 and 2 GPU respectively.

Regarding multi GPU simulation performance, it achieved better speedups than Shor's for all number of qubits when using 1 GPU, it happens because the circuit for Grover's algorithm has a lower average number of operators per projection, consequently making this application more bound and thus the memory coalescing access feature has a bigger impact on performance. For that same reason the execution with two GPU was slower than with a single GPU, hence despite having coalesced access, communication between GPU is slow and ends up having a large impact in runtime.

Table 9 – Average simulation times for Grover's Algorithm in seconds.

Qubits	*Previous*	*1 GPU*	*2 GPUs*
15	0.19	0.015	0.019
17	1.04	0.051	0.063
19	6.97	0.210	0.0294
21	56.31	1.469	2.015
23	489.52	13.215	15.770
25	4245.33	114.723	133.629

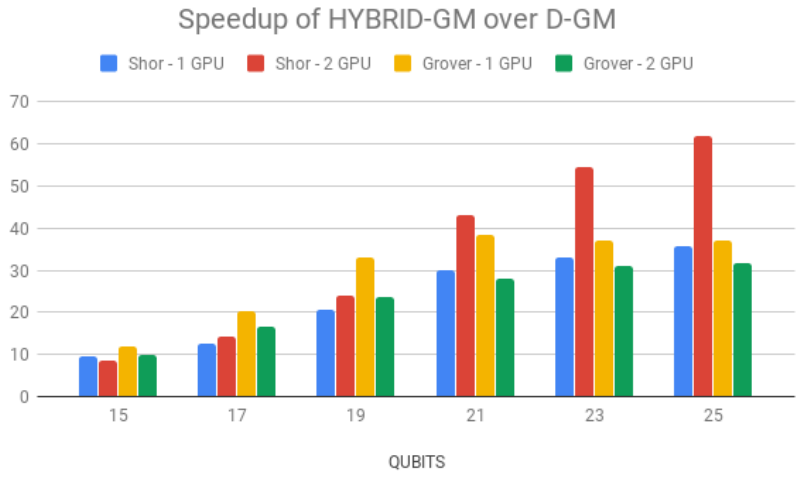


Figure 45 – HybriD-GM speedup over D-GM for GPU simulations.

7.2.2.4 *Hybrid*

In this subsection will be observed the results when performing a hybrid simulation in CPU and GPU. For that, the GPU memory was limited for the next simulations in

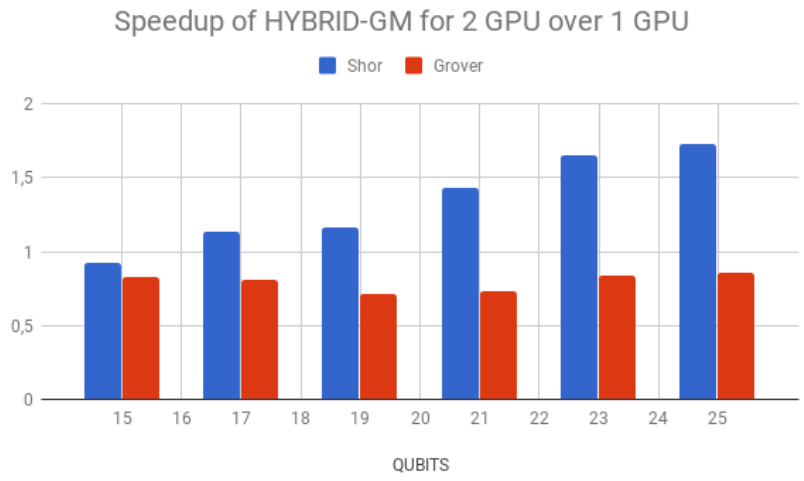


Figure 46 – Speedup of simulations with 2 GPU over 1 GPU.

order to represent a scenario were the GPU in not able to store the entire quantum state as on previous GPU simulations.

Simulations of Shor's and Grover's algorithm were performed for a 25 qubit simulation and the results obtained when limiting the GPU memory to 20 qubits are shown in Table 10.

As can be observed, simulations using only GPU increased in time for both algorithms when compared to their times with no memory limitation, Shor's algorithm only increased $1.17\times$ while Grover's increased $25,54\times$, this happens because for Shor's algorithm is possible to executes more operators per projection.

Reflecting on that, the hybrid simulation for Shor's algorithm decreased the performance with the increase in number of threads, once the GPU had to wait for the threads end their computation to pass to the next projection. And for the Grover's algorithm was the opposite, the performance increased with the number of threads as the projections were smaller not making the GPU idle.

Table 10 – Simulation times for a hybrid execution of 25-qubits Shor's and Grover's algorithm with a limitation of 20 qubits for GPU memory, measured in seconds.

	Only GPU	1 Thread	2 Threads	4 Threads	8 Threads	10 Threads
Shor	66.23	140.27	141.53	146.34	159.67	176.60
Grover	2930.19	2272.11	1835.71	1144.69	920.08	919.78

Those results showed that, depending on the algorithm, is possible to gain performance using a hybrid approach when the quantum state cant be fully stored on the GPU. And even for Shor's algorithm, it would probably gain performance as well if architecture used for the simulations had a slower GPU.

7.3 Summarizing

In this chapter, the GPU kernel is explained as follows:

- describing parameters integrating the block projections and shared memory;
- dynamic of synchronous computations of gates structures performing over amplitude states;
- updating threads in the shared memory;
- exploiting auxiliary functions applied in GPU code.

In addition, an application of LIQUi$|\rangle$ and ProjectQ which is related to parallel performances over Shor's and Grover's algorithms are reported and compared to parallel executions (10 threads) of the HybriD-GM, showing better results for the HybriD-GM achieving higher speedups for simulations with low qubits, up to $117\times$ for Shor's and $491\times$ for Grover's, and good speedups for simulations with more qubits, $4.64\times$ for Shor's and $32\times$ for Grover's.

Results are also presented to simulations of CPU for the above classes of algorithms, and the comparison of the hybrid D-GM extension with the previous D-GM version showing improvements for Shor's algorithm up to $18\times$ and for Grover's algorithm up to $7.9\times$. These improvements are justified by specifications of HybriD-GM, which are potentially more expressive over space matrices of controlled gates more present in Shor's than Grover's algorithms.

For simulations over GPU, results proved that setting the projection coalescence to match the GPU memory requests size will improve performance, as suggested by the HybriD-GM model. When compared to the previous D-GM version, simulations of Shor's algorithm showed gains up to $35.74\times$ and of Grover's algorithm up to $38.32\times$. For simulations with 2 GPUs, only Shor's algorithms was able to show improvements, up to $1.73\times$, as it presents a low amount of operators acting on the qubit that implies the necessity of communication between GPUs.

8 CONCLUSION

The simulation of QC is an important application for HPC, as spatial and temporal complexity increases exponentially with the dimension of quantum algorithms simulated.

This research work proposed the HybriD-GM model, as a hybrid computational approach, aiming to explore the use of projection operators to control computational granularity and better utilize hardware resources.

8.1 Relevance of construction of HybriD-GM Model

The study of quantum computing foundations and general purpose HPC simulators presented in the related works showed that they all have the principle of partitioning the computations to perform the quantum computing simulations, with approaches that have some similarities but with optimizations aimed to a specific architecture. Furthermore, none of them explore a hybrid approach considering CPU and GPU.

From that, it was considered the creation of a computational model that can provide support for quantum computing simulation in any architecture and that also seeks to optimize their resources.

The HybriD-GM model was then conceived, integrating HPC and QC, structured for simulations on hybrid architecture, which was applied to extend the D-GM environment. Those contributions characterize the originality of this work and intend to support the development of the quantum computing research area.

8.2 Main Contributions

This work contributed to the development of the HybriD-GM model for quantum computing simulation that explores the potentialities of High Performance Computing providing functionalities that make use of projection operators acting on quantum structures, state and transformations, to manipulate the granularity and distribution of the computation.

The HybriD-GM model strategy to define how computations will be performed is

structured as a tree, where intermediate nodes represent projection layers and final nodes represent execution layers. Such structure is configured to optimize the hardware resources for the proposed scenarios, which at the current state of the model are based on simulations in CPU, GPU and hybrid approaches.

Validation and evaluation of the HybriD-GM model by extending the D-GM environment, so that its simulations take place in the form proposed by the model. Adding a module with the structures and functionalities necessary to deal with projections, and implementing functions to perform the computation of projections in CPU and/or GPU.

Simulations testing the HybriD-GM model's efficiency were performed on the D-GM extension mainly taking Shor's and Grover's quantum algorithms, and then comparing the results obtained with the previous version of D-GM, and with LIQUi$|\rangle$ and ProjectQ simulators.

Simulations of Shor's and Grover's algorithm achieved performance gains over the previous version of $21\times$ and $9.5\times$ for parallel CPU simulations and of $61.9\times$ and $38.32\times$ for GPU simulations, respectively.

Comparing the parallel simulation to LIQUi$|\rangle$ and ProjectQ showed higher speedups for simulations with low qubits, up to $117\times$ for Shor's and $491\times$ for Grover's, and good speedups for simulations with more qubits, $4.64\times$ for Shor's and $32\times$ for Grover's.

Hybrid simulations showed that is possible to increase performance for some classes of algorithms when the size of the application does not allow full storage in GPU, improving the simulation of Grover's algorithm up to $3.18\times$ over the simulation only with GPU.

The results obtained showed that the application of model impacted positively on all types of simulations, consolidating the D-GM environment as quantum computing simulator and validating the HybriD-GM for modelling computation.

Additionally, the proposal HybriD-GM model supports new extensions, allowing the addition of optimizations steps, scenarios of projection/computation and others types of simulations. Also meaning that higher levels of quantum simulation demand a growing development of computer technology and can be accompanied by a corresponding software evolution from hardware components to digital code. Moreover, such proposal of a hybrid software architecture for quantum computing is conceived as independent of hardware, where the computations can be performed from regular desktops for sequential simulations of multi-qubits quantum applications to clusters with multiple GPU.

8.2.1 Reporting the main publications

In the following, the main themes related to the topic of this doctoral thesis are listed below. In addition, other publications related to collaborators in the research groups of FFMMFCC/UFPEL, LUPS/UFPEL achieved during the doctoral period are also presented.

8.2.1.1 Contributions reported in journal publications

1. AVILA, A.B. de; REISER, R.; PILLA, M. L., Improving in situ GPU simulation of quantum computing in the D-GM environment. INTERNATIONAL JOURNAL OF HIGH PERFORMANCE COMPUTING APPLICATIONS. , v.1, p.109434201882325 - , 2019.

2. AVILA, A.B. de; YAMIN, A.C.; PILLA, M. L.; REISER, R., State-of-the-Art Quantum Computing Simulators: Features, Optimizations, and Improvements for D-GM. NEUROCOMPUTING. , v.1, p.1 - 36, 2019.

3. AVILA, A. B.; Reiser, R.; PILLA, M. L., Quantum computing simulation through reduction and decomposition optimizations with a case study of Shor's algorithm. Concurrency and Computation. , v.3961, p.1 - 15, 2016.

8.2.1.2 Contributions reported in events and congress proceedings

1. AVILA, ANDERSON; Reiser, Renata H.S.; YAMIN, ADENAUER C.; PILLA, MAURICIO L. Efficient In-Situ Quantum Computing Simulation of Shor's and Grover's Algorithms In: 2017 International Symposium on Computer Architecture and High Performance Computing Workshops (SBACPADW), 2017, Campinas.

2. AVILA, A.B. de; REISER, RENATA H S; YAMIN, A.C.; PILLA, M.L. Parallel Simulation of Shor's and Grover's Algorithms in the Distributed Geometric Machine In: The 2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD 2017), 2017, Guilin, China.

3. AVILA, A.B. de; REISER, R.; PILLA, MAURICIO; YAMIN, A.C. Optimizing D-GM Quantum Computing by Exploring Parallel and Distributed Quantum Simulations Under GPUs Arquitecture In: WICC 2016 - IEEE CEC 2016, Congress on Evolutionary Computation, 2016, Vancouver.

4. AVILA, A.B. de; REISER, R.; PILLA, M. L. Reduction and Decomposition Optimizations in Quantum Computing Simulation Applied to the Shor's Algorithm In: CNMAC 2016 Congresso Nacional de Matemática Aplicada e Computacional, 2016, Gramado.

5. AVILA, A.B. de; REISER, R.; YAMIN, A.C.; PILLA, M. L. Scalable quantum simulation by reductions and decompositions through the Id-operator In: ACM/SIGAPP Symposium On Applied Computing (SAC), 2016, Pisa.

6. AVILA, A.B. de; REISER, R.; PILLA, M.L. Otimização de Simulação de Computação Quântica em GPUs In: ERAD/RS 2016 - XVI Escola Regional de Alto Desempenho, 2016, Porto Alegre.

8.2.1.3 Awards

The above evolution works were awarded.

- Selected papers in congresses as WAMCA, WSCAD, ICNC and WEIT.

8.3 Further Work

Once able to think "coherently", QC will be able to outperform any other calculating machine by orders of magnitude and new studies based on classical simulation need to be investigated integrated to results in HPC.

In the context of this research work, further work in the HybriD-GM project consists on many research topics, see some ones listen below.

1. **Consolidating the HybriD-GM model**

   Other main optimizations, related to speed and parallelism, can be performed on the D-GM computational environment for massively parallel and non-standard simulation of quantum computations, which are listed below:

   - continuing to study the projection properties to improve even more the HybriD-GM model;
   - extending functionalities of HybriD-GM model as heterogeneous computing;
   - adding more optimizations steps;
   - exploring other architectures such as distributed systems;
   - developing quantum applications by exploring the potential of extended D-GM model; and
   - extending the validation to other classes of quantum algorithms.

In LUPS/UFPEL, some research projects reflecting ongoing works have been developed, as descried in the two following perspectives:

2. **Improvements over memory access and memory use in simulation of QA**

   One of the main problems of quantum computing comes from the memory, representing a bottleneck not just for simulations, but for computer science research as a whole. In order to mitigate the effects from memory limitations, in (NASCIMENTO et al., 2019/08) a strategy to work around these problems is presented, providing improvements over memory access and memory use performed over simulations of dense operators, primary diagonal, and secondary diagonal operators (simple and controlled), showing a reduced amount in the number of memory access, and an improvement over the speed of execution. The methodology behind this work can be extended to be used in further work as well, even though

it can not able to fix memory performance issues, it can manage to enhance quantum simulations in the hybriD-GM model .

3. **Simulation of intuitionistic fuzzy algorithms based on quantum computing**
   Computer systems based on intuitionistic fuzzy logic are capable of generating a reliable output even when handling inaccurate input data by applying a rule based system, even with rules that are generated with imprecision. The main contribution in (AVILA et al., 2019) is to show that quantum computing can be used to extend the class of intuitionistic fuzzy sets with respect to representing intuitionistic fuzzy Xor operators. This paper describes a multi-dimensional quantum register using aggregations operators such as t-(co)norms based on quantum gates allowing the modeling and in

This work can also be addressed to study new classes of quantum algorithms in quantum computing topics as cryptography, code error correction or in development of nanotechnologies.

“This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001”

REFERENCES

AARONSON, S.; CHEN, L. Complexity-Theoretic Foundations of Quantum Supremacy Experiments. In: COMPUTATIONAL COMPLEXITY CONFERENCE (CCC 2017), 32., 2017, Dagstuhl, Germany. **Anais...** Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. p.22:1–22:67. (Leibniz International Proceedings in Informatics (LIPIcs), v.79).

AHARONOV, D.; LANDAU, Z.; MAKOWSKY, J. The quantum FFT can be classically simulated. **Computing Research Repository (CoRR)**, Cornell University, v.91, p.147902, Oct 2007.

ALVES, C.; SANTOS, L. P.; BASHFORD-ROGERS, T. A Quantum Algorithm for Ray Casting using an Orthographic Camera. In: INTERNATIONAL CONFERENCE ON GRAPHICS AND INTERACTION, ICGI 2019, FARO, PORTUGAL, NOVEMBER 21-22, 2019, 2019. **Anais...** IEEE, 2019. p.56–63.

AMARAL, R. B.; REISER, R. H. S.; COSTA, A. C. R. Interpretações do Interferômetro de Mach Zenhder no Modelo qGM. **TEMA Tendência em Matemática Aplicada e Computacional**, SBMAC, v.10, n.2, p.111–124, 2009.

AVILA, A. B. de; REISER, R.; PILLA, M. L.; YAMIN, A. C. Interpreting Xor Intuitionistic Fuzzy Connectives from Quantum Fuzzy Computing. In: INTERNATIONAL JOINT CONFERENCE ON COMPUTATIONAL INTELLIGENCE, IJCCI 2019, VIENNA, AUSTRIA, SEPTEMBER 17-19, 2019, 11., 2019. **Proceedings...** ENTCS, 2019. p.288–295.

AVILA, A. et al. GPU-aware Distributed Quantum Simulation. In: ANNUAL ACM SYMP. ON APPLIED COMPUTING (SAC), 29., 2014, New York. **Proceedings...** ACM, 2014. p.860–865. (SAC '14).

AVILA, A. et al. Optimizing Quantum Simulation for Heterogeneous Computing: a Hadamard Transformation Study. **Journal of Physics: Conference Series**, Bristol, UK, v.649, n.1, p.012004, 2015.

AVILA, A.; REISER, R.; PILLA, M. Quantum computing simulation through reduction and decomposition optimizations with a case study of Shor's algorithm. **Concurrency and Computation: Practice and Experience**, New Jersey, USA, p.1–14, 2016. cpe.3961.

AVILA, A.; REISER, R.; YAMIN, A.; PILLA, M. Scalable quantum simulation by reductions and decompositions through the Id-operator. In: ACM SYMPOSIUM ON APPLIED COMPUTING (SAC), 31., 2016, New York, USA. **Proceedings...** ACM, 2016. p.1255–1257.

AVILA, A.; REISER, R.; YAMIN, A.; PILLA, M. Optimizing D-GM Quantum Computing by Exploring Parallel and Distributed Quantum Simulations Under GPUs Arquitecture. In: IEEE CONGRESS ON EVOLUTIONARY COMPUTATION (CEC), 2016., 2016, New York, USA. **Proceedings...** IEEE, 2016. p.1–6.

AVILA, A.; SHMALFUSS, M.; REISER R.AND PILLA, M.; YAMIN, A. Scalable quantum simulation by reductions and decompositions through the id-operator. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING (SAC), 30., 2015, New York, USA. **Proceedings...** ACM, 2015. p.1–3.

BARENCO, A. et al. Elementary gates for quantum computation. **Phys. Rev. A**, Cornell University, v.52, p.3457–3467, Nov 1995.

BEAUREGARD, S. Circuit for Shor's Algorithm using 2N+3 Qubits. **Quantum Info. Comput.**, Paramus, NJ, v.3, n.2, p.175–185, 2003.

BENNETT, C. H. Time/Space Trade-Offs for Reversible Computation. **SIAM Journal on Computing**, Society for Industrial and Applied Mathematics, v.18, n.4, p.766–776, 1989.

BENNETT, C. H.; SHOR, P. W. Quantum Information Theory. **IEEE Transactions on Information Theory**, IEEE Xplore, v.44, n.6, p.2724–2742, 1998.

BISWAS, R. et al. A NASA Perspective on Quantum Computing. **Parallel Comput.**, Amsterdam, The Netherlands, The Netherlands, v.64, n.C, p.81–98, May 2017.

BOIXO, S. et al. Characterizing quantum supremacy in near-term devices. **Nature Physics**, Nature Publishing Group, v.14, n.6, p.595–600, 2018.

BRASSARD, G.; HØYER, P.; TAPP, A. Quantum Counting. In: INTL. COLLOQUIUM ON AUTOMATA, LANGUAGES AND PROGRAMMING (ICALP), 25., 1998, Berlin, DE. **Proceedings...** Springer Berlin Heidelberg, 1998. p.820–831.

BUTSCHER, B.; H., W. **Libquantum library**. Disponível em `http://www.libquantum.de`. Accessado em: Nov. 2017.

CAFARO, C.; MANCINI, S. On Grover's search algorithm from a quantum information geometry viewpoint. **Physica A: Statistical Mechanics and its Applications**, Amsterdam, NL, v.391, n.4, p.1610–1625, 2015.

DE RAEDT, H. et al. Massively parallel quantum computer simulator, eleven years later. **Computer Physics Communications**, Elsevier, v.237, p.47–61, 2019.

DENCHEV, V. S. et al. What is the Computational Value of Finite Range Tunneling. **Physical Review X**, American Physical Society, p.1–17, 2016.

DEUTSCH, D. Quantum theory, the Church–Turing principle and the universal quantum computer. **Proceedings A. Mathematical and Physical Sciences**, Royal Society of London, v.400, n.1818, p.97–117, 1985.

DO, M. et al. Planning for Compilation of a Quantum Algorithm for Graph Coloring. **Computing Research Repository (CoRR)**, Cornell University, v.abs/2002.10917, 2020.

FEYMANN, R. P. Simulating Physics with Computers. **Journal of Theory Physics**, v.11, n.21, p.467–488, 1982.

GIRARD, J.; LAFONT, Y.; TAYLOR, P. M. **Visual Object-Oriented Programming**. Cambridge: Cambridge University Press, 1989. 176p.

GROVER, L. A Fast Quantum Mechanical Algorithm for Database Search. **Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing**, ACM Digital Library, p.212–219, 1996.

GROVER, L. K. A fast quantum mechanical algorithm for database search. In: TWENTY-EIGHTH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, 1996. **Anais...** ACM Press New York, 1996. p.212–219.

GUTIERREZ, E.; ROMERO, S.; TRENAS, M.; ZAPATA, E. Quantum Computer Simulation Using the CUDA Programming Model. **Computer Physics Communications**, Elsevier, p.283–300, 2010.

HANER, T.; STEIGER, D. S. 0.5 Petabyte Simulation of a 45-Qubit Quantum Circuit. In: INTERNATIONAL CONFERENCE FOR HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS, 2017, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2017. (SC '17).

HäNER, T.; STEIGER, D. S.; SMELYANSKIY, M.; TROYER, M. High Performance Emulation of Quantum Circuits. In: INTERNATIONAL CONFERENCE FOR HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS, 2016, Piscataway, NJ, USA. **Proceedings...** IEEE Press, 2016. p.74:1–74:9. (SC '16).

HILLMICH, S.; ZULEHNER, A.; WILLE, R. Concurrency in DD-based Quantum Circuit Simulation. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE (ASP-DAC), 2020., 2020. **Anais...** IEEE Xplore, 2020. p.115–120.

HIRVENSALO, M. **Quantum Computing**. Natural Computing Series: Springer Verlag, 2001.

JEFFERY, S.; MAGNIEZ, F.; WOLF, R. Optimal parallel quantum query algorithms. In: EUROPEAN SYMP. ON ALGORITHMS (ESA), 22., 2014. **Proceedings...** Springer, 2014. p.592–604. (Lecture Notes in Computer Science, v.8737).

KAYE, P.; LAFLAMME, R.; MOSCA, M. **An Introduction to Quantum Computing**. USA: Oxford University Press, Inc., 2007.

KNILL, E. et al. Introduction to Quantum Information Processing. , Cornell University, 2002. http://arxiv.org/abs/quant-ph/0207171.

KNILL, E.; LAFLAMME, R. Power of one bit of quantum information. **Physical Review Letters**, American Physical Society, v.81, n.25, p.5672, 1998.

KNILL, E.; NIELSEN, M. Theory of quantum computation. **arXiv preprint quant-ph/0010057**, Cornell University, 2000.

LI, K.; YANG, W.; LI, K. Performance analysis and optimization for SpMV on GPU using probabilistic modeling. **IEEE Transactions on Parallel and Distributed Systems**, IEEE Xplore, v.26, n.1, p.196–205, 2015.

LU, T.-C.; JUANG, J.-C. Quantum-inspired space search algorithm (QSSA) for global numerical optimization. **Applied Mathematics and Computation**, Amsterdam, NL, v.218, n.6, p.2516–2532, 2011.

MARON, A.; REISER, R. H. S.; PILLA, M.; YAMIN, A. Expanding the VPE-qGM Environment Towards a Parallel Quantum Simulation of Quantum Processes Using GPUs. **CLEI Electronic Journal**, CLEI, v.16, n.3, p.1–18, 2013.

MARON, A.; REISER, R. H. S.; PILLA, M.; YAMIN, A. Quantum Processes: A Novel Optimization for Quantum Simulation. **TEMA: Trends in Applied and Computational Mathematics**, SBMAC, v.14, n.3, 2013.

MARON, A.; REISER, R.; PILLA, M. Correlations from Conjugate and Dual Intuitionistic Fuzzy Triangular Norms and Conorms. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2013, New York, USA. **Anais...** IEEE, 2013. p.1–8.

NASCIMENTO, M.; AVILA, A.; REISEN, R.; PILLA, M. Towards Memory Access Optimization in Quantum Computing. In: CONFERENCE OF THE EUROPEAN SOCIETY FOR FUZZY LOGIC AND TECHNOLOGY (EUSFLAT 2019), 11., 2019/08. **Anais...** Atlantis Press, 2019/08. p.467–473.

NIELSEN, M. A.; CHUANG, I. L. **Quantum Computation and Quantum Information**. Cambridge: Cambridge University Press, 2000.

OpenMP Architecture Review Board. **The OpenMP API specification for parallel programming**. Available at `http://openmp.org/wp/openmp-specifications`. Access in: Oct. 2015.

PESSOA, O. **Conceitos de Física Quântica**. SP: Editora Livraria da Física, 2003.

PORTUGAL, R.; LAVOR, C.; MACULAN, N. **Uma introdução à Computação Quântica**. SP: Notas em Matemática Aplicada - SBMAC, 2004.

WANG, G.; ZOMAYA, A.; MARTINEZ, G.; LI, K. (Ed.). **Quantum Computer Simulation on Multi-GPU Incorporating Data Locality**. Cham: Springer International Publishing, 2015. p.241–256.

SCOTT, D. Some definitional suggestions for automata theory. **Journal of Computer and System Sciences**, Elsevier, v.28, n.1, p.187–212, 1967.

SHOR, P. W. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, SANTA FE, NEW MEXICO, USA, 20-22 NOVEMBER 1994, 35., 1994. **Anais...** IEEE Xplore, 1994. p.124–134.

SHOR, P. W. Scheme for reducing decoherence in quantum computer memory. **Physical Review A**, American Physical Society, v.52, p.R2493–R2496, Oct 1995.

SHOR, P. W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. **Society for Industrial and Applied Mathematics Review**, SIAM, v.41, n.2, p.303–332, 1999.

SILVA, A. J. da; OLIVEIRA, W. R. de; LUDERMIR, T. B. Weightless neural network parameters and architecture selection in a quantum computer. **Neurocomputing**, Elsevier, v.183, p.13–22, 2016.

SMELYANSKIY, M.; SAWAYA, N. P. D.; ASPURU-GUZIK, A. qHiPSTER: The Quantum High Performance Software Testing Environment. , Cornell University, 2016.

STEIGER, D. S.; HäNER, T.; TROYE, M. ProjectQ: an open source software framework for quantum computing. **Quantum**, Vienna, v.2, 2018.

STOLLENWERK, T. et al. Quantum Annealing Applied to De-Conflicting Optimal Trajectories for Air Traffic Management. **IEEE Transactions on Intelligent Transportation Systems**, IEEE Xplore, v.21, n.1, p.285–297, 2020.

Texas Advanced Computing Center (TACC). **Stampede Supercomputer**. Disponível em: `https://www.tacc.utexas.edu/stampede`. Acesso em: Nov. 2017.

THAKER, D. D. et al. Quantum memory hierarchies: Efficient designs to match available parallelism in quantum computing. In: INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE (ISCA'06), 33., 2006, IEEE. **Anais...** IEEE Explore, 2006. p.378–390.

TRIEU, D. B. **Large-Scale Simulations of Error-Prone Quantum Computation Devices**. 2009. Dr. (Univ.) — Univ. Diss. Wuppertal, Jülich. Record converted from VDB: 12.11.2012; Wuppertal, Univ. Diss., 2009.

VERDON, G. et al. Learning to learn with quantum neural networks via classical neural networks. **Computing Research Repository (CoRR)**, Cornell University, v.abs/1907.05415, 2019.

VIDAL, G. Efficient Classical Simulation of Slightly Entangled Quantum Computations. **Phys. Rev. Lett.**, Stanford University, EUA, v.91, p.147902, Oct 2003.

WECKER, D.; SVORE, K. M. LIQUi|>: A Software Design Architecture and Domain-Specific Language for Quantum Computing. **Computing Research Repository (CoRR)**, Cornell University, v.abs/1402.4467, 2014.

YANG, W.; LI, K.; MO, Z.; LI, K. Performance Optimization Using Partitioned SpMV on GPUs and Multicore CPUs. **IEEE Transactions on Computers**, IEEE Xplore, v.64, n.9, p.2623–2636, Sept 2015.

YOUNG, T. **The Double-Slit Experiment**. Disponível em: `http://www.doubleslitexperiment.com`. Acesso em: Nov. 2017.