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

Dissertação

ReDId: Simulação de Computação Quântica baseada em Redução e Decomposição via Operador Identidade

Anderson Braga de Avila

Pelotas, 2016

Anderson Braga de Avila

ReDId: Simulação de Computação Quântica baseada em Redução e Decomposição via Operador Identidade

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação

Orientadora: Prof. Dr. Renata Hax Sander Reiser
Coorientador: Prof. Dr. Maurício Lima Pilla

Pelotas, 2016

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

A958r Avila, Anderson Braga de

ReDId : simulação de computação quântica baseada em Redução e Decomposição via operador Identidade / Anderson Braga de Avila ; Renata Hax Sander Reiser, orientadora ; Maurício Lima Pilla, coorientador. — Pelotas, 2016.

60 f.

Dissertação (Mestrado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2016.

1. Simulação de computação quântica. 2. Processamento paralelo. 3. GPU. 4. Algoritmo de Shor. I. Reiser, Renata Hax Sander, orient. II. Pilla, Maurício Lima, coorient. III. Título.

CDD : 005

Elaborada por Aline Herbstrith Batista CRB: 10/1737

Universidade Federal de Pelotas
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

ATESTADO

ATESTO para os devidos fins, que **ANDERSON BRAGA DE AVILA** foi aprovado pela Banca Examinadora da Defesa de Dissertação realizada em 3 de Março de 2016, intitulada: "ReDId: Quantum Computing Simulation based on Reduction and Decomposition via Identity Operator", sob a orientação do Prof. Dr. Renata Hax Sander Reiser. Realizadas as correções recomendadas pela Comissão Examinadora, o trabalho será encaminhado para homologação pelo Colegiado do Programa de Pós-Graduação em Computação. Igualmente, informamos que os demais requisitos necessários para a obtenção do grau de **Mestre em Ciência da Computação** pela Universidade Federal de Pelotas foram cumpridos.

Pelotas, 3 de Março de 2016

Ricardo Matsumura de Araujo
Coordenador do Programa de Pós-Graduação em Computação

RESUMO

AVILA, Anderson Braga de. **ReDId: Simulação de Computação Quântica baseada em Redução e Decomposição via Operador Identidade**. 2016. 60 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2016.

Um dos maiores obstáculos para a simulação de algoritmos quânticos é o crescimento exponencial das complexidades espaciais e temporais, devido à expansão das transformações e dos estados de leitura/escrita pelo uso do produto tensor em aplicações multi-dimensionais. A simulação destes sistemas é muito relevante para desenvolver e testar novos algoritmos quânticos. Para minimizar o problema gerado pela alta complexidade da simulação de algoritmos quânticos, este trabalho apresenta uma nova estratégia nomeada ReDId, provendo otimizações baseadas na redução e decomposição via operador Identidade. Na sequência, o trabalho considera a implementação do algoritmo que faz uso da estratégia ReDId, explorando os componentes VPE-qGM e VirD-GM que integram o ambiente D-GM. Para validação, considera-se a aplicação das otimizações via estratégia ReDId nas simulações de transformações Hadamard de 21 a 28 qubits e Transformadas de Fourier Quântica de 26 a 28 qubits. Estes algoritmos foram simulados sobre CPU, sequencialmente e em paralelo, e em *GPU*, mostrando redução da complexidade temporal e, consequentemente, menor tempo de simulação. Além disso, avaliações do algoritmo de Shor considerando o uso de $2n + 3$ qubits no algoritmo quântico para cálculo da ordem, foram simulados até 25 qubits. Ao comparar nossas implementações executando no mesmo hardware com o simulador LIQUi$|\rangle$ - *Language-Integrated Quantum Operations*, versão disponível pela QuArC - *Quantum Architectures and Computation Group* da *Microsoft Research*, o simulador via estratégia ReDId mostrou-se mais rápido.

Palavras-chave: Simulação de Computação Quântica, Processamento Paralelo, GPU, Algoritmo de Shor.

ABSTRACT

AVILA, Anderson Braga de. **ReDId: Quantum Computing Simulation based on Reduction and Decomposition via Identity Operator**. 2016. 60 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2016.

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. To minimize the problem created by the high complexity of the simulation of quantum algorithms, this paper presents a new strategy named ReDId, which provides optimizations based on the reduction and decomposition via Identity operator. Next, the paper considers the implementation of the algorithm that makes use of the ReDId strategy, exploring the components VPE-qGM and VirD-GM that integrate the D-GM environment. For evaluation, is considered the application of optimization via ReDId strategy in simulations of Hadamard transformations from 21 to 28 qubits and Quantum Fourier Transforms from 26 to 28 qubits. Those algorithms were simulated over CPU, sequentially and in parallel, and in GPU, showing reduced temporal complexity and, consequently, shorter simulation time. Moreover, evaluations of the Shor's algorithm considering $2n + 3$ qubits in the order-finding quantum algorithm were simulated up to 25 qubits. Comparing our implementations running on the same hardware with LIQUi$|\rangle$ - *Language-Integrated Quantum Operations*, release version by QuArC - *Quantum Architectures and Computation Group* from *Microsoft Research*, the simulator via ReDId strategy proved to be faster.

Keywords: Quantum Computing Simulation, Parallel Processing, GPU, Shor's Algorithm.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

D-GM	Distributed Geometric Machine
VirD-GM	Virtual Distributed Geometric Machine
VPE-qGM	Visual Programing Environment for the Quantum Geometric Machine Model
QCEdit	Quantum Circuit Editor
M-QCEdit	Mobile Quantum Ciruit Editor
TQ	Transformação Quântica
TFQ	Transformada de Fourier Quântica
qGM	Quantum Geometric Machine
qPE	Quantum Process Editor
qME	Quantum Memory Editor
qS	Quantum Simulator
CQ	Computação Quântica
MQ	Mecânica Quântica
MPP	Mixed Partial Process
QuIDD	Quantum Information Decision Diagram
MPQCS	Massive Parallel Quantum Computer Simulator
CPU	Central Processing Unit
GPU	Graphic Processing Unit
CUDA	Compute Unified Device Architecture

SUMÁRIO

1 INTRODUÇÃO

Neste trabalho é realizada a concepção da estratégia de simulação *ReDId*, que visa a redução da complexidade espacial e temporal inerente da simulação de aplicações quânticas, através do uso inteligente do operador Identidade na representação de transformações quânticas (TQs) e pela decomposição das transformações. Ao invés de executar uma TQ em um único passo, ela é dividida passos e só operadores diferentes do Identidade são armazenados.

Este trabalho está inserido no Projeto *ExPloreD-GM - Explorando o Paralelismo e a Distribuição do Modelo D-GM em Aplicações Científicas e Tecnologias Associadas*, buscando o estudo e aplicação de otimizações para simulação, sequencial e paralela, de algoritmos quânticos a partir de computadores clássicos.

O Projeto ExPloreD-GM busca a consolidação do ambiente D-GM (*Distributed Geometric Machine*), para modelagem e desenvolvimento de aplicações quânticas a partir de dois principais componentes de software: (i) o *VPE-qGM* (*Visual Programming Environment of the Quantum Geometric Machine Model*), ambiente de programação visual para modelagem de algoritmos quânticos; e (ii) o *VirD-GM* (*Virtual Distributed Geometric Machine*), ambiente de execução distribuída e gerenciamento destas aplicações.

1.1 Tema

O tema central desta dissertação é a simulação de algoritmos da Computação Quântica (CQ). A CQ é um paradigma fundamentado nos postulados definidos pela Mecânica Quântica (MQ), a qual provê interpretações para os comportamentos físicos incomuns que se fazem presentes quando da manipulação de elementos em escala atômica/subatômica, substituindo, dessa forma, as leis da Física Clássica (NIELSEN; CHUANG, 2003; PESSOA, 2003).

Aplicações envolvendo propriedades da MQ (emaranhamento, sobreposição, inversibilidade, não-clonagem) no contexto computacional, permitem, a concepção de uma nova classe de computadores (computadores quânticos), os quais são capa-

zes de apresentar um desempenho exponencialmente maior do que os computadores clássicos (ÖMER, 1998), principalmente em áreas que envolvem a modelagem, manipulação, transmissão e o processamento da informação quântica como na pesquisa em:

(i) criptografia, explorando a não-localidade quântica para transmitir mensagens com segurança absoluta (SHOR, 1997);

(ii) busca em listas não-ordenadas (GROVER, 1996);

(iii) teletransporte, transportando informação quântica de um lugar para outro sem que ocorra o deslocamento através de um meio físico (STEFFEN et al., 2013);

(iv) e ainda, algoritmos de fatoração e logaritmo discreto (SHOR, 1994).

No processamento da informação quântica, o estado-da-arte em *hardware* quântico ainda apresenta baixa escalabilidade devido a dificuldade de manipulação e controle de partículas elementares (elétrons, fótons, etc), as quais têm potencial para serem utilizadas como *qubits*. Assim, atualmente, o estudo e modelagem de algoritmos para *CQ* pode ser realizado de duas principais formas: através da especificação matemática do sistema ou do desenvolvimento dos circuitos quânticos(HEY, 1999; KNILL; NIELSEN, 2000) em *softwares* de simulação(RAEDT et al., 2006; NIWA; MATSUMOTO; IMAI, 2002).

1.2 Motivação

A simulação quântica é uma área de pesquisa básica, com muitos problemas para serem investigados, porém bem consolidada nos seus fundamentos. Contudo, a simulação de sistemas quânticos através de computadores clássicos mostra-se ineficiente, visto o alto custo computacional associado a aplicação das transformações que determinam a evolução temporal do sistema.

Nesse cenário, a adoção de soluções voltadas para o ganho de desempenho é essencial para simulação assim como a contínua pesquisa considerando a exploração de novas abordagens de otimização da complexidade espacial e temporal, tornam-se estratégias essenciais para simulação de aplicações quânticas complexas.

A introdução de diferentes soluções para lidar com a complexidade de simulação de algoritmos quânticos (HEY, 1999), a partir de computadores clássicos, tem contribuído para delinear as melhores abordagens, lidando com os diferentes problemas decorrentes do aumento exponencial no armazenamento e processamento, ambos requeridos por sistemas *multi-qubits*.

Uma atual e relevante abordagem, em consolidação neste trabalho, considera a integração de dois esforços: (i) a busca por otimização das estruturas associadas a transformações quânticos; e (ii) a paralelização das operações envolvidas na evolução do estado do sistema.

Esta integração constitue-se em um grande desafio, porém a perspectiva de bons resultados consiste na grande motivação que impulsiona esse tipo de pesquisa.

Neste contexto, no Projeto D-GM, integrando o ambiente de desenvolvimento *VPE-qGM* e o ambiente de gerenciamento *VirD-GM*, promove-se suporte desde a concepção até o desenvolvimento de aplicações, considerando uma arquitetura de software que contempla uma abordagem híbrida, viabilizando a computação (heterogênea) via *CPUs* ou *GPU*s para suporte a aplicações multi-qubits.

1.3 Objetivos

Este trabalho tem como objetivo geral a concepção, modelagem e aplicação de uma nova estratégia para tratamento das complexidades espacial e temporal, inerentes a simulação de algoritmos quânticos *multi-qubits* via software em computadores clássicos.

A partir desta estratégia, tem-se a possibilidade de aplicação de soluções eficientes para o ambiente D-GM, visando a extensão das suas capacidades de simulação em multicomputadores e/ou multiprocessadores.

Mais especificamente, considera-se os seguintes objetivos:

- concepção, modelagem e desenvolvimento de uma nova estratégia para simulação de computação quântica;
- implementação de um novo algoritmo execução para o ambiente D-GM, em linguagem C/C++ para as execuções em *CPU*, e em *CUDA* para as execuções em *GPUs*;
- validação das otimizações pela execução de aplicações provendo a simulação de algoritmos quânticos, e posterior avaliação dos resultados alcançados;
- divulgação na comunidade científica dos resultados obtidos na pesquisa por publicações em eventos/jornais especializados da área de simulação da computação quântica e de computação heterogênea via *CPUs* (*Central Processing Units*) e *GPUs* (*Graphics Processing Units*).

1.4 Organização do Texto

Na continuidade, este trabalho está estruturado da forma descrita logo a seguir.

No Capítulo 2 os principais fundamentos relacionados à CQ são descritos, incluindo uma breve discussão sobre duas aplicações quânticas simuladas neste trabalho.

No Capítulo 3, são discutidas as características encontradas em alguns simuladores quânticos mais relevantes, incluindo o simulador utilizada para comparação com este trabalho.

Uma das principais contribuições deste trabalho é descrita no Capítulo 4, contemplando a criação da nova estratégia de simulação de algoritmos quânticos, a Seção 4.1 abrange a otimização referentes as replicações e esparcialidades geradas pelo operador Identidade, a Seção 4.2 descreve como se dá a decomposição das transformações quânticos e a Seção 4.3 discute o uso da partição das memórias para proves escalabilidade as aplicações.

Na sequência, o Capítulo 5 contempla a descrição do ambiente *D-GM* e das implementações direcionadas a criação da nova biblioteca de execução com suporte à simulação em *CPUs* e *GPUs* usando a estratégia ReDId descrita neste trabalho.

No Capítulo 6 são mostrados os resultados obtidos com simulações de aplicações Hadamard, TQF e Shor usando a extensão da biblioteca qGM-Analyzer com suporte às otimizações propostas pelo ReDId, bem como uma comparação com o simulador LIQ$Ui| >$ para o algoritmo de Shor.

Por fim, no Capítulo 7, as principais conclusões obtidas a partir da realização deste trabalho, bem como propostas de continuidade do projeto D-GM são apresentadas.

2 COMPUTAÇÃO QUÂNTICA

O corrente capítulo contempla os principais conceitos de *CQ* e *MQ* que fundamentam este trabalho. Também são estudados alguns dos simuladores quânticos disponíveis atualmente.

2.1 Conceitos Básicos

A manipulação de partículas em escala atômica/subatômica compreende uma tarefa de alta complexidade, visto que nessas situações as partículas (fótons, elétrons e outras partículas de mesma escala) exibem comportamentos incomuns, não definidos pelas leis da física clássica. A *MQ* é a área da física que estuda tais comportamentos, apresentando teorias que definem de forma precisa os fenômenos que ocorrem em pequena escala.

Um dos principais fundamentos da *MQ* é a ocorrência de **superposição de estados** (PESSOA, 2003). Na mecânica clássica, é estabelecido que uma partícula possui um estado único e bem definido, como por exemplo, um elétron com um *spin* positivo. Na *MQ*, uma partícula pode ser definida a partir da coexistência de dois ou mais estados, ou seja, um elétron com *spin* positivo e negativo simultaneamente.

A *MQ* também contempla a interpretação do fenômeno da **dualidade onda-partícula**, exemplificado através do Experimento da Fenda Dupla (THE DOUBLE-SLIT EXPERIMENT, 2009), na qual uma partícula atômica é capaz de apresentar dois comportamentos distintos: (i) ondulatório, onde sua trajetória é descrita por uma superposição de ondas; (ii) corpuscular, com uma trajetória bem definida. Uma partícula quântica é definida através de uma **função de onda**, sendo constituída de uma grande variedade de estados possíveis. Porém, ao utilizar qualquer dispositivo para medir (observar) o estado dessa partícula, sua função de onda colapsa em uma das possibilidades, comportando-se, a partir desse ponto, como uma partícula com um estado bem definido.

Desse comportamento surge o **princípio da incerteza de Heisenberg** (PESSOA, 2003) que, em suma, estabelece que ao medir o estado de um objeto quântico,

instantaneamente seu estado será alterado, deixando de apresentar propriedades quânticas. Nota-se, portanto, a impossibilidade de determinar a trajetória de uma partícula quântica, visto que, ao medir a correspondente posição, seu estado colapsa, impedindo a medida de sua velocidade.

O princípio do **entrelaçamento** (PESSOA, 2003) considera que duas partículas criadas juntas são capazes de interagir de forma imediata mesmo quando separadas espacialmente por qualquer distância, de forma que, ao submeter uma partícula a um determinado efeito, a outra partícula entrelaçada a esta irá reagir instantaneamente. Interpretações detalhadas dos princípios da *MQ* podem ser obtidas em (PESSOA, 2003; PORTUGAL; LAVOR; MACULAN, 2004).

2.1.1 Postulados da Mecânica Quântica

Essas ponderações iniciais fundamentam a descrição do comportamento de sistemas quânticos, os quais são matematicamente especificados através de quatro postulados definidos pela *MQ*, permitindo a analogia com sistemas físicos:

- Espaço de Estados: Na *CQ*, o *qubit* é a unidade básica de informação, sendo o sistema quântico mais simples, definido por um vetor de estado, unitário e bidimensional, genericamente descrito, na notação de Dirac (NIELSEN; CHUANG, 2003), pela expressão $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Os coeficientes α e β são números complexos correspondentes às amplitudes dos respectivos estados, respeitando a condição de normalização $|\alpha|^2 + |\beta|^2 = 1$, garantindo a unitariedade do vetor de estado do sistema, representado por $v = (\alpha, \beta)^t$[1].

- Evolução do Sistema: A mudança de estado em um sistema quântico é feita por transformações quânticas unitárias, associadas a matrizes ortonormalizadas de ordem $2^N \times 2^N$, sendo N a quantidade de *qubits* da transformação.

- Medida Quântica: O processo de extração de informação de um sistema quântico, identificando qual o estado corrente mais provável, é estudado a partir da operação de medida. A medida quântica projetiva considera um conjunto de operadores de projeção, os quais aplicam diferentes processos de filtragem sobre o espaço de estados. O estado final do sistema depende do operador de projeção executado.

- Sistemas Compostos: O espaço de estados de um sistema quântico de múltiplos *qubits* é compreendido pelo produto tensorial do espaço de estados de seus sistemas componentes. Considerando um sistema quântico de dois *qubits*, $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ e $|\varphi\rangle = \gamma|0\rangle + \delta|1\rangle$, o correspondente espaço de estados é composto

[1] v^t denota a transposta de v

pelo produto tensor

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

A aplicação das propriedades da *MQ* no contexto computacional resulta no paradigma da *CQ*, permitindo o desenvolvimento de computadores quânticos com capacidade de processamento exponencialmente maior que os computadores clássicos, como demonstrado pela Tabela 1.

Tabela 1: Fatoração clássica x Fatoração quântica. Fonte:()

Tamanho do nº a ser fatorado	Algoritmo Clássico	Algoritmo Quântico
512 *bits*	4 dias	34 segundos
1024 *bits*	10^5 anos	4,5 minutos
2048 *bits*	10^{14} anos	36 minutos
4096 *bits*	10^{26} anos	4,8 horas

O modelo mais recorrente para descrição de aplicações quânticas é o modelo de circuitos quânticos (NIELSEN; CHUANG, 2003). Essa representação é uma das mais fundamentais da *CQ*, sendo caracterizada por uma notação gráfica intuitiva que remete ao modelo de circuitos digitais utilizados na computação clássica.

Os circuitos quânticos compreendem sincronizações e composições de portas (transformações) quânticas unitárias e operações de medidas, modelando qualquer tipo de algoritmo quântico, conforme apresentado na Figura 1. Algumas convenções são adotadas visando uma descrição homogênea dos algoritmos quânticos, sendo descritas a seguir:

- Linhas Horizontais: Cada linha representa um dos *qubits* que compõem o circuito, e a correspondente evolução temporal ocorre da esquerda para a direita;
- Linhas Verticais: Indicam que uma determinada transformação quântica atua sobre os *qubits* conectados através desta linha;
- Controle: Representado por um círculo sobre a linha de um *qubit*. Se o círculo for fechado, indica que é considerado o estado $|1\rangle$ do *qubit*; se for aberto, considera-se o estado $|0\rangle$;
- Portas Quânticas: Transformações unitárias que manipulam o *qubit* sobre o qual são aplicadas;
- Medida: No final de cada linha do circuito pode aparecer uma operação de medida, determinando o estado clássico do correspondente *qubit*.

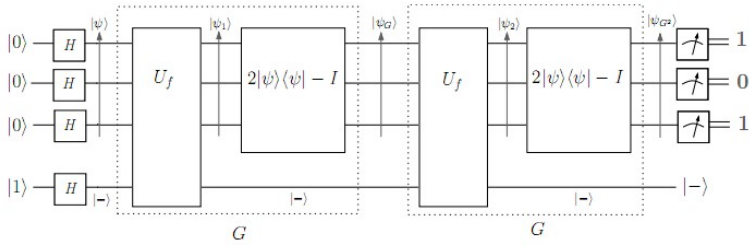


Figura 1: Circuito Quântico

2.1.2 Transformações Quânticas

As transformações unitárias quânticas são as operações responsáveis por manipular as amplitudes associadas aos estados do sistema. Essas transformações são definidas por matrizes unitárias quadradas de ordem 2^N, onde N representa a quantidade de *qubits* sobre os quais a transformação irá atuar. As principais transformações quânticas básicas são descritas na sequência.

- Hadamard: É a transformação responsável por gerar a superposição dos estados de um *qubit*. Sua definição matricial é:

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

  A aplicação de H sobre o vetor de estado do *qubit* genérico $|\psi\rangle$, definido na primeiro postulado da *MQ*, resulta em:

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

- Pauly X: Equivalente a porta clássica *NOT*, que inverte as amplitudes dos estados de um *qubit*. A correspondente definição matricial e aplicação sobre o vetor de estado de $|\psi\rangle$ é representado por:

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

- Pauly Y: Quando aplicada à $|\psi\rangle$, resulta em $Y|\psi\rangle = -i\beta|0\rangle + i\alpha|1\rangle$. A correspondente definição matricial é:

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

- Pauly Z: A matriz de transformação dessa operação é dada por:

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

  Sua função é realizar a inversão de fase do *qubit*, transformando o vetor de estado em $(\alpha, -\beta)^t$.

- Phase (*S*): Introduz uma fase relativa, ou seja, leva o *qubit* $|\psi\rangle$ para o estado $S|\psi\rangle = \alpha|0\rangle + i\beta|1\rangle$, onde a amplitude de $|0\rangle$ mantém-se inalterada, enquanto que a amplitude de $|1\rangle$ difere por um fator de fase igual à *i*. A matriz correspondente a porta Phase é descrita por:

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

- $\pi/8$: Transformação quântica associada a seguinte matriz unitária:

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

  A aplicação de T ao vetor de estado de $|\psi\rangle$ resulta em $(\alpha, exp(i\pi/4)\beta)^t$.

Para exemplificação do aumento exponencial nas transformações quânticas aplicadas a múltiplos *qubits*, considera-se, primeiramente, a transformação *Hadamard* (H) aplicada a um *qubit*. A seguinte representação matricial descreve tal cenário:

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

Considerando, agora, a aplicação simultânea de H à dois *qubits* de um sistema quântico, tem-se a matriz

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

a qual é obtida a partir da operação de produto tensorial ($\otimes$) entre as correspondentes matrizes de 1 *qubit*. O operador $\otimes$ gera um aumento exponencial na quantidade de elementos da matriz resultante, influenciando significativamente no custo de simulação das operações.

A aplicação de $H^{\otimes 2}$ ao sistema quântico com espaço de estados definido na Eq. 1, é dada por

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

Além das transformações de múltiplos *qubits*, obtidas pelo produto tensorial de transformações unitárias básicas, existem transformações controladas que modificam o estado de um ou mais *qubits* considerando o estado corrente dos demais. Dentre as transformações controladas, destacam-se:

CNOT (*Controled NOT*)

A transformação quântica *CNOT* recebe 2 *qubits* $|\psi\rangle$ e $|\varphi\rangle$ como entrada e aplica a transformação *NOT* (*Pauly X*) a um deles (*qubit* alvo), considerando o estado corrente do outro *qubit* (controle). Dessa observação, percebe-se a possibilidade de algumas configurações distintas para essa transformação:

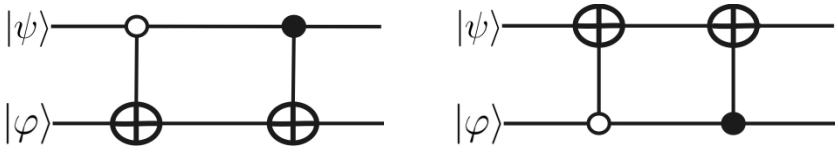
- Controle no *Qubit* $|\psi\rangle$: A modificação no *qubit* $|\varphi\rangle$ pode ser realizada quando o estado de $|\psi\rangle$ for $|0\rangle$ ou $|1\rangle$. No modelo de circuitos quânticos, essas operações são apresentadas de acordo com a Figura 2(a), respectivamente. Assim, têm-se os seguintes operadores matriciais que manipulam o vetor de estado do sistema:

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

- Controle no *Qubit* $|\varphi\rangle$: A modificação no *qubit* $|\psi\rangle$ pode ser realizada quando o estado de $|\varphi\rangle$ for $|0\rangle$ ou $|1\rangle$. Analogamente ao caso anterior, as correspondentes representações no modelo de circuitos quânticos são apresentadas na Figura 2(b), respectivamente. Os dois operadores matriciais que definem o comportamento dessa transformação são:

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} = \begin{pmatrix} \gamma \\ \beta \\ \alpha \\ \delta \end{pmatrix} \quad e \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} = \begin{pmatrix} \alpha \\ \delta \\ \gamma \\ \beta \end{pmatrix} \tag{13}$$

Toffoli



(a) Portas com controle em $|\psi\rangle$

(b) Portas com controle em $|\varphi\rangle$

Figura 2: Transformações quânticas controladas CNOT

A transformação controlada *Toffoli* é definida para três *qubits*, de forma a aplicar a transformação *Pauly X* ao terceiro *qubit* quando os estados dos dois primeiros *qubits* forem $|1\rangle$. A representação no modelo de circuitos quânticos é exemplificada pela Figura 3.

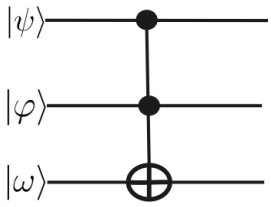


Figura 3: Porta *Toffoli* no modelo de circuitos quânticos.

Controlled-U

Na transformação controlada *CNOT*, o operador *Pauly X* é aplicado ao *qubit* alvo considerando o estado de um único *qubit* de controle. Porém, transformações controladas genéricas (*Controlled-U*) (NIELSEN; CHUANG, 2003) podem ser definidas, de forma a utilizar variadas configurações de *qubits* de controle e aplicar qualquer transformação unitária *U* ao(s) *qubit(s)* alvo.

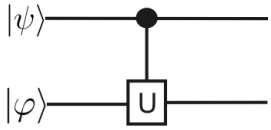


Figura 4: Porta *Controlled-U* no modelo de circuitos quânticos.

A partir da composição, sequencial e síncrona, de transformações quânticas, é possível realizar computações (GROVER, 1996; SHOR, 1997; KNILL; NIELSEN, 2000; AARONSON, 2007) que exploram os fenômenos particulares da mecânica quântica, obtendo ganhos frente aos melhores algoritmos clássicos conhecidos.

2.1.3 Transformada de Fourier Quântica

A execução da Transformada de Fourier Quântica (TFQ) se dá pela aplicação de operadores de descolocamento controlados e Hadamard e pode ser descrito como

uma operação de todo o sistema em termos de entradas e saídas dos correspondentes estados na base computacional de um estado quântico.

Ao tomar n qubits, $N = 2^n$ estados clássicos da base computacional $\{|j\rangle : j = 0, \ldots, N-1\}$, a TFQ aplicada a cada $|k\rangle$ pode ser expressada como:

$$DFT(|k\rangle) = |\phi_k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{\frac{2\pi ijk}{N}} |j\rangle \tag{14}$$

onde $\{|\phi_k\rangle : k = 0, \ldots, N-1\}$ denota a nova base ortogonal.

Na computação da Eq. 14, realizada passo por passo em cada estado na base ortogonal, o efeito de transformações controladas em cada qubit pode ser avaliada, no vetor de estado do sistema quântico. Baseado na propriedade de superposição da computação quântica, n qubits de entrada se transformam em $N = 2^n$ estados da base computacional.

No entanto, a execução de diferentes qubits da entrada pode ser realizada independentemente dos outros $N - 1$ componentes do vetor de estado num sistema quântico, fazendo com que este problema adequado para explorar *CPUs* multi-core e *GPUs*. Além disso, alcançar escalabilidade quando sistemas maiores como o algoritmo quântico de fatoração são simuladas.

2.1.4 Algoritmo de Shor para Fatoração Quântica

Considere o problema da fatoração de números primos: *"dado um inteiro positivo N (normalmente com várias centenas de dígitos), como descobrir seus fatores primos"*?

É bem conhecido que fatorar N pode ser reduzida a tarefa de escolher aleatoriamente um número inteiro a coprimo de N, e então a ordem r de a módulo N.

Esta abordagem para a fatoração permitiu que Shor construí-se seu algoritmo de fatoração para a computação quântica (SHOR, 1994, 1997). Ele consiste em um pré-processamento clássico, um algoritmo quântico para procurar a ordem, e um pós-processamento clássico (WECKER; SVORE, 2014).

O único uso de computação quântica no algoritmo de Shor é para procurar a ordem de a módulo N, onde N é um inteiro de n bits que queremos fatorar. Adicionalmente, o período r é o menor inteiro positivo que satisfaz $a^r \equiv 1(mod\ N)$.

Dado um número N para fatorar, o algoritmo segue os seguintes passos (BEAUREGARD, 2003):

1. Se N é par, retorne o fator 2.

2. Determine classicamente se $N = pq$ para $p \geq 1$ e $q \geq 2$ e se for o caso retorne o fator p (isto pode ser feito em tempo polinomial).

3. Escolha aleatoriamente um número a que satisfaça $1 < a \leq N - 1$. Usando o algoritmo de Euclides, determine se $gcd(a, N) > 1$ e se for o caso, retorne o fator $gcd(a, N)$.

4. Se r é ímpar ou r é par mas $a^{r/2} = -1(mod\ N)$, então vá para o passo 3. Caso contrário, calcule $gcd(a^{r/2} - 1, N)$ e $gcd(a^{r/2} + 1, N)$. Teste para saber se um desses é um fator não-trivial de N, e retorne o fator se for.

O algoritmo quântico para calcular a ordem usado neste trabalho é o descrito em (BEAUREGARD, 2003), usando $2n + 3$ qubits para fatorar um inteiro de n bits.

2.2 Considerações finais

Neste capítulo foram descritos os principais conceitos que fundamentam a CQ, com ênfase nas portas quânticas que estruturam os principais algoritmos utilizados na validação dos resultados desenvolvidos.

Foram brevemente estudas a evolução das transformações Hadamard, a caracterizada a geração de estados de sobreposição em sistemas multiqubits, com representação que faz uso de matrizes densas na modelagem do paralelismo quântico. Em contraposição, as portas controladas são representadas por matrizes esparsas geradas pela uso do operador Identidade.

Estas duas classes de transformações quânticas são discutidas nos próximos capítulos, e definem estratégias distintas quanto às otimizações propostas. A primeira estratégia se reporta ao uso da parcialidade dos processos para controle da granulosidade das computações concorrentes, enquanto que na segunda, consideram-se operações de decomposição e redução para minimizar as redundâncias na iteração do operador identidade pela aplicação do produto tensorial em sistemas multi-qubits.

Estes operadores são considerados quando da simulação do algoritmo de Shor(SHOR, 1994), que para cálculo da fatoração de um número inteiro N primeiramente considera o algoritmo quântico de cálculo da ordem de um inteiro x módulo N e na sequência, aplica a Transformação Quântica de Fourier Discreta para descobrir o período de uma função de forma eficiente.

No próximo capítulo, são reportados os principais simuladores estudados no contexto deste trabalho.

3 SIMULADORES QUÂNTICOS

Atualmente estão disponíveis simuladores quânticos com diversas abordagens. Dentre os mais relevantes, tem-se simuladores sequenciais que implementam otimizações para representação de transformações e de estados quânticos, reduzindo a quantidade de memória requerida durante a simulação, suportando sistemas quânticos com mais de 30 *qubits*.

Também são consideradas abordagens para simulação paralela de algoritmos quânticos, focados na redução do tempo necessário para aplicação das transformações quânticas pela exploração de *clusters* e *GPUs*.

A escolha dos simuladores considerados nesta sessão está embasada na descrição apresentada em (MARON, 2013).

3.1 QuIDDPro

O simulador *QuIDDPro*, proposto em (VIAMONTES, 2007), utiliza estruturas denominadas *QuIDDs* (*Quantum Information Decision Diagrams*) para representar eficientemente transformações e estados quânticos multidimensionais, os quais são definidos, matricialmente, por blocos de valores repetidos. Esses padrões de repetição ocorrem com frequência em vários algoritmos, sendo possível obter uma significativa redução no consumo de memória e no tempo de acesso às informações.

Um *QuIDD* é uma representação comprimida de matrizes e vetores, permitido que computações sejam realizadas diretamente sobre essa estrutura otimizada. Exemplificando, na Figura 5, tem-se a representação de diferentes estados quânticos utilizado *QuIDDs*. Na Figura 5(c), a aresta sólida saindo do vértice I_0 equivale a assinalar o valor 1 ao primeiro *bit* do índice de 2 *bits*. Já a aresta tracejada do vértice I_1 equivale a assinalar o valor 0 ao segundo *bit* do mesmo índice. Esses caminhos levam ao valor $-\frac{1}{2}$, o qual é a amplitude do estado da base computacional indexado por $|10\rangle$.

Dessa representação percebe-se que, para estados com amplitudes iguais, é obtido um *QuIDD* extremamente simples. Para estados com muitas amplitudes diferentes, não é possível obter compressão. Entretanto, tais estados não são usuais na

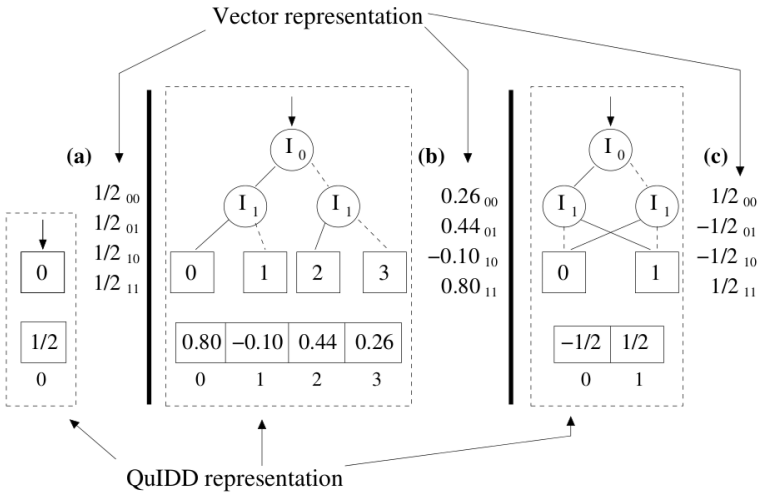
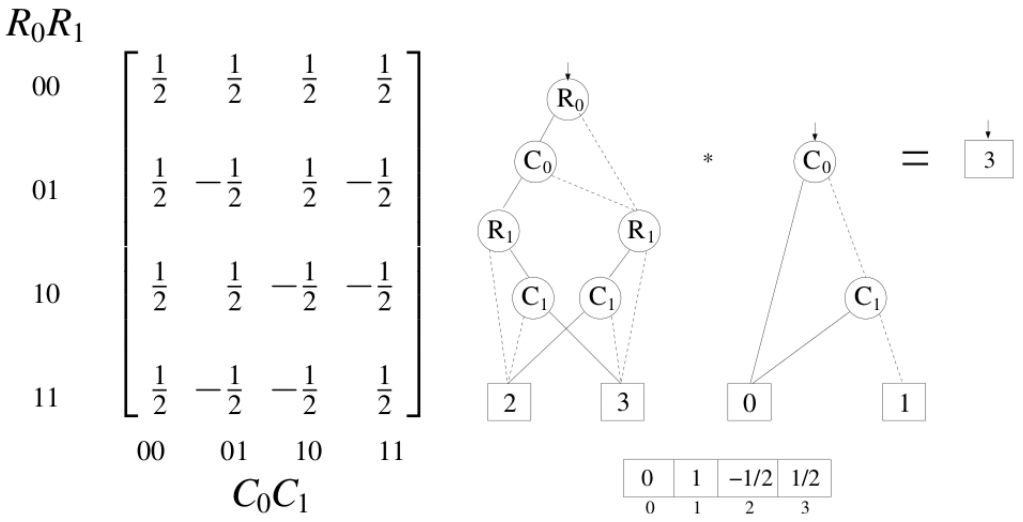


Figura 5: Exemplos de QuIDDs para diferentes estados quânticos. Fonte: (VIAMONTES, 2007)

CQ. Transformações quânticas são definidas de forma análoga, como visto nas Figuras 6(a) e 6(b).



(a) Definição matricial de $H^{\otimes 2}$

(b) Representação de $H^{\otimes 2}|00\rangle$ usando QuIDD

Figura 6: Transformações quânticas no QuIDDPro. Fonte: (VIAMONTES, 2007)

Vários resultados obtidos pelo *QuIDDPro* podem ser vistos em (GROUP, 2007). Como destaque, tem-se a simulação de instâncias do Algoritmo de Grover para sistemas de até 40 *qubits*, no qual o consumo de memória não ultrapassou $0,398$ MB, enquanto que demais pacotes de simulação ficaram limitados a sistemas de até 25 *qubits*. Entretanto, por se tratar de um processamento sequencial, a simulação demorou $8,23^4$ segundos.

3.2 Massive Parallel Quantum Computer Simulator

O *Massive Parallel Quantum Computer Simulator* (*MPQCS*) (RAEDT et al., 2006) consiste em um *software* para simulação paralela de computadores quânticos, podendo ser aplicado a máquinas paralelas *high end* ou a *clusters* de *desktops* comuns. Implementado em Fortran 90, o simulador suporta as transformações quânticas universais necessárias para descrição de qualquer algoritmo quântico. O algoritmo pode ser descrito a partir de um pseudo-código, demonstrado na Figura 7, ou por um circuito quântico, gerado a partir de uma *interface* gráfica desenvolvida para *MS Windows*. O circuito é interpretado e automaticamente gera o correspondente pseudo-código.

```
QUBITS 32                    ! The command QUBITS sets the size of the quantum computer
INITIAL STATE 0              ! The initial state of the quantum computer is set to |0...0>
MPIPROCESSES 32              ! Number of MPI processes is set to 32
                             ! (not used by the OpenMP code)
H 0                          ! Hadamard operation on qubit 0
.                            ! These lines are omitted here but contain commands for
.                            ! Hadamard operations carried out on qubits 1-25
.
H 26                         ! Hadamard operation on qubit 26
SWAP 1 0 27                  ! Command to swap 1 pair of qubits: qubits 0 and 27
H 27                         ! Hadamard operation on qubit 27
SWAP 1 27 28                 ! Command to swap qubits 27 and 28
```

Figura 7: Pseudocódigo para descrição de uma aplicação quântica no Massive Parallel Quantum Computer Simulator. Fonte: (RAEDT et al., 2006)

A técnica de paralelização do *MPQCS* consiste em distribuir o espaço de estados do sistema quântico através de todos os nodos do *cluster*, utilizando o modelo de programação *MPI* para comunicação entre os processos. A quantidade de processos (N) *MPI* necessários para construção do sistema quântico depende da razão $N = 2^L/2^M$, onde L representa a quantidade de *qubits* do sistema e M é a quantidade de *qubits* que cada processo é capaz de armazenar. Dessa forma, cada processo têm acesso direto à um determinado intervalo do espaço de estados do sistema quântico, enquanto que os demais estados são acessíveis através de comunicação com os correspondentes processos. A Figura 8 descreve um cenário com $L = 4$ e $M = 2$, associando os processos às respectivas amplitudes armazenadas.

	$\sigma_1 = \begin{pmatrix} 3 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$			
	00	**01**	**10**	**11**
00	a(0000)	a(0100)	a(1000)	a(1100)
01	a(0001)	a(0101)	a(1001)	a(1101)
10	a(0010)	a(0110)	a(1010)	a(1110)
11	a(0011)	a(0111)	a(1011)	a(1111)

Figura 8: Distribuição de amplitudes para 4 processos MPI. Fonte: (RAEDT et al., 2006)

Os valores **00**, **01**, **10** e **11** fazem alusão ao *rank* de cada processo MPI. Já *00*, *01*, *10* e *11* indexam as posições de memória de cada processo.

Considerando que a aplicação de uma transformação quântica sobre o sistema usualmente altera as amplitudes de todos os estados, percebe-se a necessidade de um método para obter as amplitudes armazenadas em processos diferentes. Exemplificando, considera-se a distribuição ilustrada pela Figura 8 e a aplicação da transformação quântica *NOT* ao *qubit* 2 do sistema. Para efetivação dessa operação, é necessário que cada processo *MPI* tenha conhecimento das amplitudes associadas aos estados $|0\rangle$ e $|1\rangle$ do *qubit* 2. Essa condição é satisfeita após a troca de amplitudes entre os processos, definida pela permutação σ_2, apresentada na Figura 9. Percebe-se, agora, que todos os processos possuem as amplitudes associadas aos estados do *qubit* 2. Assim, todos os processos podem aplicar, simultaneamente, a transformação quântica sobre esse *qubit*, caracterizando a evolução do estado global do sistema quântico. Descrições detalhadas de como essa troca é realizada podem ser obtidas em (RAEDT et al., 2006).

	$\sigma_1 = \begin{pmatrix} 3 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$				$\sigma_2 = \begin{pmatrix} 3 & 2 & 1 & 0 \\ 3 & 0 & 1 & 2 \end{pmatrix}$			
	00	**01**	**10**	**11**	**00**	**01**	**10**	**11**
00	a(0000)	a(0100)	a(1000)	a(1100)	a(0000)	a(0001)	a(1000)	a(1001)
01	a(0001)	a(0101)	a(1001)	a(1101)	a(0100)	a(0101)	a(1100)	a(1101)
10	a(0010)	a(0110)	a(1010)	a(1110)	a(0010)	a(0011)	a(1010)	a(1011)
11	a(0011)	a(0111)	a(1011)	a(1111)	a(0110)	a(0111)	a(1110)	a(1111)

Figura 9: Distribuição de amplitudes para N processos MPI considerando a permutação σ_2. Fonte: (RAEDT et al., 2006)

Os testes foram realizados em vários supercomputadores, como IBM BlueGene/L, Cray X1E, IBM Regatta p690+, dentre outros. Os principais resultados demonstram a capacidade de simulação de sistemas com até 36 *qubits*, exigindo aproximadamente 1 TB RAM e 4096 processadores. Esse alto custo se deve ao armazenamento explicito de todas as amplitudes que definem o estado do sistema quântico, implicando no armazenamento de 2^{36} valores complexos, usualmente representados por dois dados do tipo *float*.

Em 2010, o *MPQCS* fez uso do supercomputador *JUGENE* para executar uma instância do Algoritmo de Shor com 42 *qubits*, fatorando o número 15707 em 113×139. Para isso, foram necessários 262.144 processadores, porém o consumo de memória e o tempo de simulação requerido não foram divulgados.

Dada a constante necessidade de comunicação entre os processos, o simulador exige uma rede de intercomunicação de alta capacidade. Dessa forma, *clusters* comuns, formados por PCs conectados por uma rede *ethernet*, apresentarão uma limitação na quantidade de nodos que podem ser utilizados eficientemente, interfe-

rindo diretamente na dimensão dos sistemas quânticos suportados.

3.3 General-Purpose Parallel Simulator for Quantum Computing

A natureza inerentemente paralela associada à evolução de um sistema quântico é explorada, no *General-Purpose Parallel Simulator for Quantum Computing* (NIWA; MATSUMOTO; IMAI, 2002), a partir do particionamento da matriz associada à transformação unitária em sub-matrizes. Essas sub-matrizes são aplicadas, de forma paralela, a sub-vetores associados ao estado do sistema quântico.

Para a aplicação de uma transformação sobre o *qubit i*, como ilustrado na Figura 10, faz-se necessário a geração da matriz X a partir do produto tensor $X = (\otimes_{k=1}^{i-1} I) \otimes U \otimes (\otimes_{k=i+1}^{n} I)$.

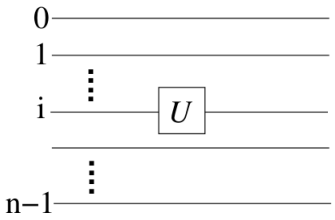


Figura 10: Transformação quântica de 1 *qubit*

A Figura 11 mostra uma representação genérica para a matriz X, na qual tem-se uma possível decomposição em termos de submatrizes S_k.

$$X = \underbrace{\begin{pmatrix} S_0 & & & & & 0 \\ & S_1 & & & & \\ & & \cdots & & & \\ & & & \cdots & & \\ & & & & S_{2^i-2} & \\ 0 & & & & & S_{2^i-1} \end{pmatrix}}_{2^n} \text{ where } S_k = \underbrace{\begin{pmatrix} u_{11} & \cdots & \mathbf{0} & u_{12} & \cdots & \mathbf{0} \\ & & & & & \\ \mathbf{0} & & u_{11} & \mathbf{0} & & u_{12} \\ u_{21} & & \mathbf{0} & u_{22} & & \mathbf{0} \\ & \cdots & & & \cdots & \\ \mathbf{0} & & u_{21} & \mathbf{0} & & u_{22} \end{pmatrix}}_{2^{n-i}} (0 \leq k < 2^i)$$

Figura 11: Matriz que define a evolução do sistema quântico. Fonte: (NIWA; MATSUMOTO; IMAI, 2002)

A metodologia para particionamento de X depende da quantidade de processadores disponíveis (2^P) no *cluster*. A matriz X é decomposta em um sequência de multiplicações de submatrizes-subvetores indexadas por $M_k(0 \leq k < 2^i)$. M_k é definido como sendo $S_k\phi_k$, ou seja, o produto $S_k(2^{n-i} \times 2^{n-i}) \times \phi_k(2^{n-i})$, onde ϕ_k é um vetor que armazena as amplitudes dos estados do sistema quântico. Como todos os produtos $M_k(0 \leq k < 2^i)$ são independentes, tem-se a possibilidade de execução simultânea de 2^{i-P} multiplicações de submatrizes-subvetores em cada processador, como exemplificado na Figura 12. Ao final da execução de cada produto, tem-se uma primitiva de sincronização para atualizar todas as amplitudes do vetor de estado do sistema quântico.

$$X|\phi\rangle = \begin{pmatrix} S_0 & & & & 0 \\ & S_1 & & & \\ & & \cdots & & \\ & & & S_{2^i-2} & \\ 0 & & & & S_{2^i-1} \end{pmatrix} \begin{pmatrix} \phi_0 \\ \phi_1 \\ \cdots \\ \phi_{2^i-2} \\ \phi_{2^i-1} \end{pmatrix} \begin{matrix} \} (processor\ 0) \\ \cdots \\ \cdots \\ \cdots \\ \} (processor\ 2^P) \end{matrix} \quad \text{where } \phi_k = \begin{pmatrix} \alpha_{k2^{n-i}} \\ \alpha_{k2^{n-i}+1} \\ \cdots \\ \alpha_{(k+1)2^{n-i}-2} \\ \alpha_{(k+1)2^{n-i}-1} \end{pmatrix} \quad (0 \le k < 2^i)$$

Figura 12: Decomposição da transformação quântica. Fonte: (NIWA; MATSUMOTO; IMAI, 2002)

Quando o número de submatrizes S_k é menor que a quantidade de processadores disponíveis ($2^i < 2^P$), ocorre um desbalanceamento no *cluster*, fazendo com que determinados processadores trabalhem de forma excessiva enquanto outros permanecem ociosos. Assim, as matrizes S_k devem ser decompostas para viabilizar a execução paralela. Essa decomposição se dá pela divisão de S_k em 2^{P+1} *chunks* de linhas, indexados por $R_j (0 \le j < 2^{P+1})$, as quais armazenam $2^{n-i-(P+1)}$ linhas de S_k.

As multiplicações de *chunks* indexados por R_j e R_{2^P+j} são mapeadas para o mesmo processador, conforme apresentado na Figura 13.

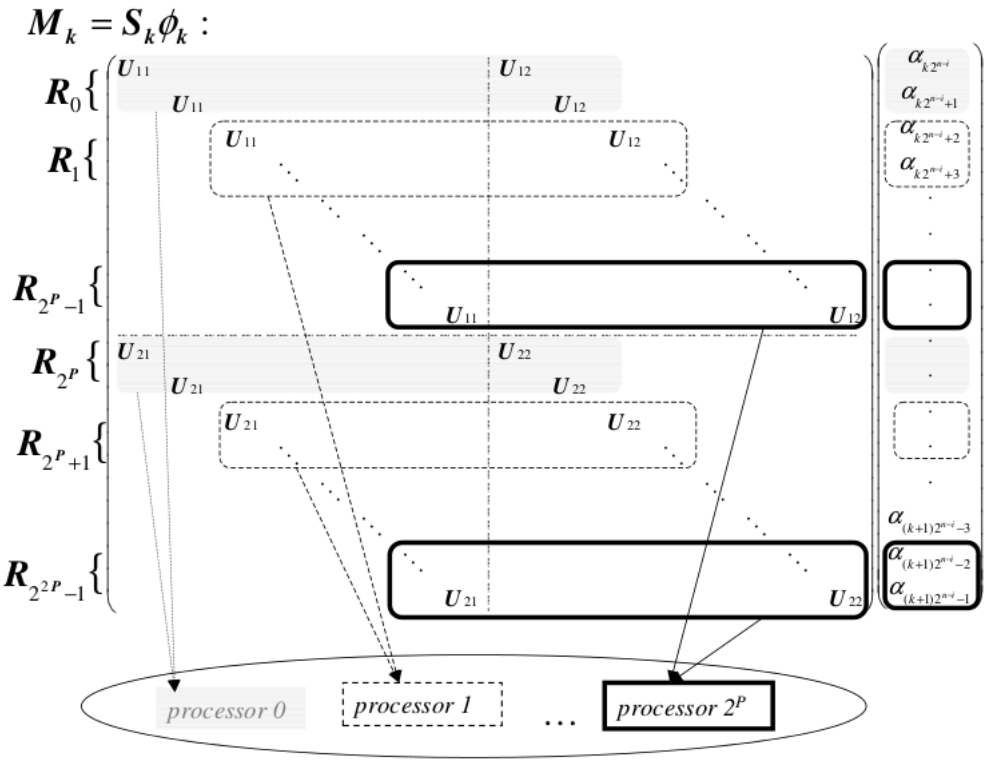


Figura 13: Decomposição de M_k e ϕ_k para casos em que $2^i \le 2^P$. Fonte: (NIWA; MATSUMOTO; IMAI, 2002)

Transformações quânticas também podem ser definidas a partir de matrizes de

rotação ($U_R(\theta)$) e de mudança de fase ($U_{P1}(\phi)$ e $U_{P2}(\phi)$), definidas por

$$U_R(\theta) = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}, \quad U_{P1}(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix} \; e \; U_{P2}(\phi) = \begin{pmatrix} e^{i\phi} & 0 \\ 0 & 1 \end{pmatrix}. \quad (15)$$

Por exemplo, a porta quântica *NOT* pode ser obtida por $U_R(\frac{\pi}{2})U_{P1}(\pi)$. Essa característica permite a implementação de um modelo de erros, considerando a decoerência dos sistemas quânticos, a qual é simulada a partir da inserção de pequenos desvios nos ângulos θ e ϕ, permitindo uma simulação coerente com as teorias físicas que fundamentam a Teoria da Informação Quântica.

O simulador foi desenvolvido sobre o computador paralelo Sun Enterprise (E4500), o qual possui 8 processadores UltraSPARC-II (400MHz), 1 MB cache, 10GB RAM e OS Solaris 2.8 (64 *bits*).

Dentre os resultados obtidos, destaca-se a simulação de transformações *Hadamard* em um sistema quântico de 29 *qubits*, conforme apresentado na Figura 14.

Qubits	Num. of Procs			
	1	2	4	8
20	2.38	1.18	0.76	0.40
22	10.85	5.73	3.20	1.35
24	46.94	24.96	13.40	9.58
26	205.81	109.97	58.83	38.71
28	887.40	467.71	253.82	167.31
29	2027.9	1081.1	592.08	395.81

Figura 14: Tempos, em segundos, para simulação de transformações *Hadamard*. Fonte: (NIWA; MATSUMOTO; IMAI, 2002)

O simulador fica limitado em aproximadamente 29 *qubits* devido ao custo de armazenamento do espaço de estados e da matriz de transformação do sistema, os quais crescem exponencialmente conforme são adicionados novos *qubits*.

3.4 Quantum Computer Simulation using CUDA

O simulador quântico descrito em (GUTIÉRREZ et al., 2010) utiliza as premissas do modelo de programação CUDA para exploração do paralelismo associado a evolução dos sistemas quânticos. Dessa forma, viabiliza-se a execução do cálculo das amplitudes associadas aos estados da base computacional a partir de um grande número de *threads*, as quais executam sobre as unidades de processamento das *GPUs*.

Essa abordagem considera o armazenamento explícito de todas as amplitudes que definem o estado do sistema quântico, de forma que uma grande quantidade de memória é necessária para suporte a simulação de algoritmos complexos. A obtenção das novas amplitudes dos estados não se dá por matrizes geradas a partir

da operação de produto tensor, reduzindo o custo de processamento e armazenamento da simulação. As novas amplitudes são definidas a partir das matrizes associadas às transformações quânticas universais, de um e dois *qubits*. Essas matrizes fornecem os coeficientes que serão multiplicados pelas amplitudes associadas aos estados do sistema. A descrição detalhada do cálculo das amplitudes pode ser estudado em (GUTIÉRREZ et al., 2010).

Para realização dos cálculos, é necessário copiar todo o vetor de amplitudes para a memória global da *GPU*. Visando um acesso mais rápido a esses dados durante a execução das *threads*, os conjuntos de amplitudes a serem modificadas são copiadas para a área de memória compartilhada, a qual possui menor tempo de acesso. Entretanto, faz-se necessário uma função de mapeamento que realize essa cópia de forma eficiente, garantindo um acesso coalescido a memória, aproveitando a largura de banda disponível no barramento e evitando a serialização das *threads*.

Cada bloco de *threads* é associado com determinados conjuntos de amplitudes (*closed groups*), os quais podem ser processados de forma independente. Cada *thread* do bloco realiza, simultaneamente, a computação associada a determinadas amplitudes contida nos conjuntos. Esse mapeamento depende da granulosidade do problema e da quantidade de recursos computacionais disponíveis. Dessa forma, pode ser definido previamente que uma *thread* compute apenas algumas amplitudes ou todas as amplitudes de vários conjuntos distintos. Na Figura 15 é apresentada uma visão simplificada dos principais mapeamentos a serem realizados.

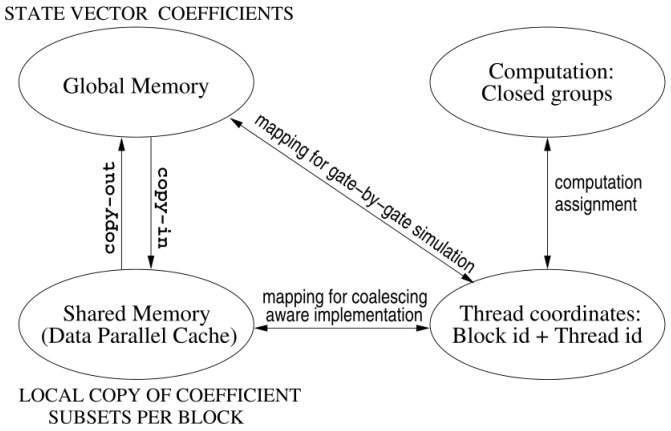


Figura 15: Principais mapeamentos a serem feitos para computação das amplitudes. Fonte: (GUTIÉRREZ et al., 2010)

A análise de desempenho considera uma *GPU* NVIDIA GeForce 8800GTX e CUDA 1.1. As principais características da *GPU* são descritas na Tabela 2.

Cada amplitude é um valor complexo representado por 2 *floats* de 32 *bits* cada, permitindo o armazenamento de até 26 *qubits* (2^{26} amplitudes) no espaço de memória global da *GPU*. Devido ao tamanho reduzido da memória compartilhada, é possível

Tabela 2: Principais características da GeForce 8800GTX

Atributo	Valor
Multiprocessadores	16
Processadores em cada multiprocessador	8
Frequência de *clock*	1,35 GHz
Memória global	768 MB

realizar a cópia de até 2^{10} amplitudes, limitando a quantidade de amplitudes a serem calculadas simultaneamente. Uma análise geral de desempenho é vista na Figura 16.

n	CUDA programming model on GPU						Sequential code on CPU		Speed-up t_{CPU}/t_{GPU}
	Gate-by-gate t (msec)	Coalescing-aware					Libquantum t (msec)	Optimized t (msec)	
		r	c	nstg	last	t (msec)			
Walsh									
15	0.32	10	4	2	5/6	0.11	4.42	1.76	16.3
16	0.46	10	4	2	6/6	0.20	10.57	3.82	18.9
17	0.78	9	4	3	3/5	0.42	32.42	7.90	18.7
18	1.39	9	4	3	4/5	0.81	83.07	18.82	23.1
19	2.68	9	4	3	5/5	1.63	179.72	47.89	29.4
20	5.41	9	5	4	3/4	3.48	380.81	100.45	28.9
21	11.07	9	5	4	4/4	7.15	825.84	210.78	29.5
22	22.82	9	5	5	1/4	15.41	1688.36	439.86	28.5
23	47.28	9	5	5	2/4	31.73	3364.75	919.95	29.0
24	98.26	9	5	5	3/4	65.39	7066.15	1927.47	29.5
25	203.69	9	5	5	4/4	135.15	15 077.27	4052.66	30.0
26	422.36	9	5	6	1/4	290.08	32 729.87	8774.42	30.2
QFT									
15	0.46	10	4	2	5/6	0.13	5.35	10.58	82.7
16	0.63	10	4	2	6/6	0.24	13.07	18.86	77.0
17	0.98	9	5	3	4/4	0.52	42.67	40.09	77.8
18	1.62	9	4	3	4/5	0.98	120.39	86.90	88.6
19	2.99	9	4	3	5/5	1.96	254.19	187.59	95.8
20	5.78	8	4	4	4/4	4.14	520.33	389.25	93.9
21	11.60	9	5	4	4/4	8.57	1107.78	816.98	95.3
22	23.71	8	4	5	2/4	18.26	2264.26	1715.42	93.9
23	48.84	8	4	5	3/4	37.67	4631.77	3588.86	95.3
24	101.04	9	4	4	5/5	76.60	9752.44	7492.01	97.8
25	209.12	9	5	5	4/4	161.58	22 878.99	15 642.21	96.8
26	433.03	8	4	6	2/4	343.30	55 325.69	32 692.91	95.2

Figura 16: Tempos de simulação para transformações Walsh-Hadamard e TFQ. Fonte: (GUTIÉRREZ et al., 2010)

As simulações sequenciais realizadas consideram um PC com Intel Core2Duo 6400 @ 2,13GHz com 2GB RAM, utilizando a biblioteca *libquantum* para simulação quântica. Como principais conclusões, destaca-se que o tamanho da memória física da *GPU* limita de forma significativa a quantidade de *qubits* suportados. Entretanto, para sistemas com até 26 *qubits*, um *speedup* de até $95\times$ pode ser obtido, caracterizando um excelente ganho de desempenho.

3.5 Language-Integrated Quantum Operations– LIQ$Ui|>$

LIQ$Ui|>$ é um novo projeto de pesquisa na Microsoft (WECKER; SVORE, 2014) que fornece uma arquitetura de software para computação quântica independente de hardware. Ele contém uma linguagem embutida, de domínio específico projetado para algoritmos de programação quântica, com F# como a linguagem *host*. Também per-

mite a extração de uma estrutura de dados de circuitos que pode ser utilizado para otimizações, gerando uma versão compacta, altamente otimizada para simulação.

Dois ambientes de simulação diferentes estão disponíveis para o usuário que permitem que um *trade-off* entre o número de qubits e classe das operações.

O simulador LIQ$Ui|>$ é altamente otimizado, tirando vantagem de várias técnicas disponíveis, incluindo gerenciamento personalizado de memória, paralelização, *"gate growing"*, e virtualização (execução na nuvem).

O simulador possui uma versão disponível para download no GitHub, porém ela é limitada para simulações de no máximo 23 qubits, e permite a definição e simulação de circuitos usando a linguagem F#. LIQ$Ui|>$ é um programa executado por linha de comando e possui algoritmos já incluídos para simulação, um exemplo de saída para a execução do comando *Liquid.exe __Shor(21,true)* que executa o algoritmo de Shor otimizado para o número 21 pode ser visto na Figura 17.

```
0:0000.0/=============================================================================================
0:0000.0/=    The Language-Integrated Quantum Operations (LIQUi|>) Simulator                         =
0:0000.0/=        Copyright (c) 2015,2016 Microsoft Corporation                                      =
0:0000.0/=        If you use LIQUi|> in your research, please follow the guidelines at               =
0:0000.0/=        https://github.com/msr-quarc/Liquid for citing LIQUi|> in your publications.       =
0:0000.0/=============================================================================================
0:0000.0/
0:0000.0/=============== Logging to: Liquid.log opened ================
0:0000.0/======== Doing Shor Round =========
0:0000.0/       21 = N = Number to factor
0:0000.0/        2 = a = coPrime of N
0:0000.0/        5 = n = number of bits for N
0:0000.0/       32 = 2^n
0:0000.0/       13 = total qubits
0:0000.0/       30 = starting memory (MB)
0:0000.0/   96.00% = prob of random result (983/1024)
0:0000.0/   43.07% = prob of Shor (worst case)
0:0000.0/          - Compiling circuit
0:0000.0/0.001156 = mins for compile
0:0000.0/    13890 = cnt of gates
0:0000.0/     3873 = cache hits
0:0000.0/      121 = cache misses
0:0000.0/       30 = compiled memory (MB)
0:0000.0/          - Wrapping circuit pieces
0:0000.1/             8 = wires have possibles:        66 (prv=  0GB did=     0 big=     0)
0:0000.1/             9 = wires have possibles:        57 (prv=  0GB did=     9 big=     4)
0:0000.1/            10 = wires have possibles:        52 (prv=  0GB did=    14 big=    39)
0:0000.1/            11 = wires have possibles:        51 (prv=  0GB did=    15 big=    81)
0:0000.1/            12 = wires have possibles:        48 (prv=  0GB did=    18 big=   124)
0:0000.1/            13 = wires have possibles:        47 (prv=  0GB did=    19 big=   166)
0:0000.1/       14 = Ran out of wires
0:0000.1/            MM: g:    19 b:   212  12=1 11=3 10=1 9=5 8=9
0:0000.1/0.118196 = mins for growing gates
0:0000.1/      470 = cnt of gates
0:0000.1/       53 = grown memory (MB)
0:0000.1/          Bit:    9 [MB:    41 m=1]
0:0000.2/          Bit:    8 [MB:    54 m=1]
0:0000.4/          Bit:    7 [MB:    42 m=0]
0:0000.5/          Bit:    6 [MB:    42 m=1]
0:0000.6/          Bit:    5 [MB:    43 m=0]
0:0000.6/... compiling MB=     200 cache(19785,126) GC:5611
0:0000.7/          Bit:    4 [MB:    44 m=1]
0:0000.9/          Bit:    3 [MB:    44 m=0]
0:0001.0/          Bit:    2 [MB:    44 m=1]
0:0001.1/          Bit:    1 [MB:    44 m=0]
0:0001.1/... compiling MB=     200 cache(35697,126) GC:4476
0:0001.2/          Bit:    0 [MB:    44 m=1]
0:0001.2/0.026858 = mins for running
0:0001.2/ 74.3236 = Total Elapsed time (seconds)
0:0001.2/       13 = Max Entangled
0:0001.2/        0 = Gates Permuted
0:0001.2/      371 = State Permuted
0:0001.2/      128 = None  Permuted
0:0001.2/      683 = m = quantum result
0:0001.2/0.666992 = c = 683/1024 =~ 2/3
0:0001.2/         Odd denominator, expanding
0:0001.2/        3 = 6/2 = exponent
0:0001.2/        9 = 2^3 + 1 mod 21
0:0001.2/        7 = 2^3 - 1 mod 21
0:0001.2/        7 = factor = max(3,7)
0:0001.2/CSV N a m den f1 f2 good,21,2,683,6,7,3,1
0:0001.2/GOT:   21=   7x   3 co=    2 n,q=21,13 mins=1.24 SUCCESS!!
0:0001.2/=============== Logging to: Liquid.log closed ================
```

Figura 17: Saída de uma execução no simulador LIQ$Ui|>$.

LIQ$Ui|>$ apresenta simulações de até 30 qubits em uma única maquina de 32 GB RAM limitado somente por memória e *threads*, resto da arquitetura não especificado. A Figura 18 mostra os tempos de simulação do algoritmo de Shor apresentados

em (WECKER; SVORE, 2014), sendo a linha com os menores tempos a última versão do simulador após inserida todas as otimizações. O maior número fatorado até o momento possui 13 bits, sendo necessário 27 qubits, meio milhão de portas e 5 dias de execução. O circuito utilizado para a parte quântica do algoritmo de Shor é o mesmo citado neste trabalho.

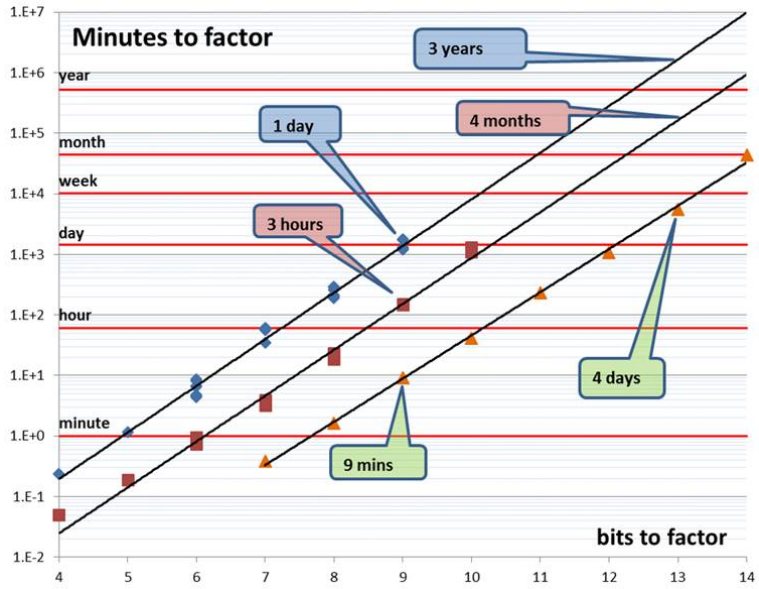


Figura 18: Tempos de simulação para o algoritmo de Shor no simulador LIQ$Ui|>$. Fonte: (WECKER; SVORE, 2014)

3.6 Considerações Finais

A proposta considerada pelo *QuIDDPro* é significante pela capacidade de redução no consumo de memória, viabilizando a simulação de sistemas com até 40 *qubits*. O *QuIDDPro* apresenta bons resultados quando existem padrões repetidos na estrutura matricial, ou seja, é eficiente em determinadas classes de algoritmos e pode não apresentar a mesma eficiência para casos genéricos. Apesar do ganho de memória obtido, em determinadas configurações de transformações quânticas o tempo de simulação ainda permanece alto. Esse obstáculo pode ser superado através da paralelização das tarefas.

Os simuladores paralelos descritos neste capítulo representam transformações e estados quânticos explicitamente a partir de matrizes e vetores, respectivamente. Dessa forma, as possibilidades de simulação são limitadas pela quantidade de memória disponível do *cluster* utilizado. Entretanto, é possível obter ganho no tempo de simulação pela paralelização do produto matriz-vetor que caracteriza a evolução de estado do sistema quântico. A distribuição das amplitudes dos estados ao longo dos nodos também é uma estratégia adotada, porém apresenta elevado custo de

comunicação.

O simulador LIQ$Ui| >$ apresenta diversos tipos de otimizações, é o mais atual entre os simuladores citados e disponibiliza a ferramenta para simulações. Além destas caracterizações, acrescenta-se que este simulador considera o mesmo circuito quântico para o algoritmo de Shor que foi utilizado nas implementações para validação do ReDId. Justifica-se portanto a escolha do simulador LIQ$Ui| >$ para realizar comparações com os resultados alcançados no desenvolvimento deste trabalho, as quais são apresentadas nos próximos capítulos.

4 REDID: ESTRATÉGIA PARA REDUÇÃO DA COMPLEXIDADE ESPACIAL E TEMPORAL

Em trabalhos anteriores (MARON et al., 2012; MARON; REISER; PILLA, 2013; AVILA et al., 2014), processos relacionados a TQs n-dimensionais eram definidos por um conjunto de matrizes básicas de baixa ordem para reduzir a memória usada nas simulações devido ao crescimento exponencial da sua representação por uma única matriz ($2^n \times 2^n$).

Por exemplo, considere a TQ $H^{\otimes 8}$, ao invés de armazenar uma matriz de $2^8 \times 2^8$, o produto tensor poderia ser realizado somente entre pares de operadores gerando um conjunto com 4 matrizes de $2^2 \times 2^2$, como mostrado na Figura 19.

$$\frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \quad \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \quad \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \quad \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Figura 19: Matrizes Básicas para a Transformação $H^{\otimes 8}$

Os elementos necessários na computação da simulação da TQ eram gerados em tempo de execução por iterações sobre estas matrizes, modelando a expansão exponencial da memória decorrente da aplicação do produto tensorial. No entanto, o tempo total computacional gasto nestas iterações é alto quando se dá a execução de TQs compostas por muitos operadores densos, como no caso das transformações Hadamard.

As principais otimizações propostas pela estratégia ReDId estão principalmente relacionadas à redução da complexidade espacial e temporal associada a simulação de uma TQ, pelo uso inteligente do operador Identidade.

Abordagens distintas foram usadas para alcançar bons resultados, e elas são descritas nas seções a seguir.

4.1 Replicações e Esparcialidade do Operador Identidade

A primeira otimização explora o comportamento do operador Identidade (I) quando aplicado o produto tensorial com outros operadores. Nestes casos, a diagonal principal do operador Identidade composta por valores unitários (1) gera a replicação de dados referentes aos outros operadores e a diagonal secundária composta por valores nulos (0) introduz a esparcialidade na matriz resultante.

Como um exemplo deste comportamento, considere a Eq. 16 para a transformação $I \otimes H$, ao analisar a matriz resultante do produtor tensor entre esse operadores pode-se facilmente verificar que os valores da matriz correspondente ao operador H são replicados (duas vezes) e a quantidade de valores zeros gerados é igual a duas vezes o número de elementos do operador H.

$$I \otimes H = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \tag{16}$$

Assim, quanto maior for a quantidade de operadores I presentes em uma TQ, maior será também a quantidade de elementos replicados e da esparcialidade na matriz que representa esta TQ.

Por exemplo, uma TQ de 20 qubits com apenas um operador diferente de I, cada elemento deste operador seria replicado 2^{19} vezes e 2^{21} zeros seriam gerados.

Quando considerado o método desenvolvido em trabalhos anteriores, as replicações e esparcialidades inseridas pelo operador I não são tão significantes em relação a complexidade espacial, pois o operador I não altera o resultado da computação na composição com outro operador. Porém, são inseridas inúmeras iterações pelas replicações, o que aumenta significativamente o número de operações de uma computação, incrementando a complexidade temporal.

Considerando as argumentações acima, o trabalho considerou otimizar a representação das TQs ao armazenar somente a expansão do produto tensorial entre operadores diferentes de I gerando assim uma matriz reduzida (MR), evitando a replicação de dados e inserção de esparcialidade gerada pelo operador I, resultando então na diminuição da complexidade espacial das TQs assim caracterizadas.

Porém ao utilizar esta forma de representação não é possível considerar a forma convencional de cálculo das novas amplitudes do estado quântico, caracterizada pela multiplicação entre matriz/vetor (processo-TQ/estado-vetor), uma vez que a ordem da MR é sempre menor, no máximo igual, a dimensão do estado. Portanto, para usar esta otimização é preciso adotar uma abordagem diferente que será explicada na

sequência.

Informações relevantes referentes a nova abordagem de cálculo da multiplicação matricial:

- Considera-se a representação binária das linhas/colunas da MR e das posições (leitura e escrita) das amplitudes nos estados quânticos.
- A quantidade de bits utilizada para cada estrutura é a mínima necessária, ou seja, para linhas/colunas é o equivalente ao número de operadores diferentes de I na MR, e para as posições das amplitudes estados é o equivalente ao número de qubits da aplicação.
- Cada bit das posições das amplitudes está relacionado a um operador da TQ. O mais significativo, ao operador do primeiro qubit; segundo mais significativo, ao operador do segundo qubit, e assim sucessivamente até esgotar a dimensão do operador.
- Bits relacionados a operadores distintos do operador I são considerados bits-ativos.

Utilizando as informações resumidas citadas, os passos para realização do cálculo de cada nova amplitude são descritos abaixo:

- **(i) Identificação da linha da matriz MR à ser usada** - este processo é feito pela concatenação dos bits-ativos da posição referente à nova amplitude, sendo o número resultante a linha desejada;
- **(ii) Multiplicação dos elementos desta linha por amplitudes do estado de leitura** - a amplitude pela qual cada elemento será multiplicado é determinada substituindo os bits da coluna deste elemento na MR pelo bits-ativos da posição da nova amplitude;
- **(iii) Atribuição do valor da nova amplitude** - este valor resulta da soma das multiplicações realizadas no passo anterior.

Como exemplo, considera-se a aplicação de um operador genérico ao primeiro qubit e ao segundo qubit de um estado bi-dimensional dados pelas Eqs. 17 e 18, respectivamente. Os elementos da matriz MR estão descritos na forma $\mathbf{m_{ij}}$, onde i e j são sua linha e coluna, respectivamente. As amplitudes dos estados estão descritas na forma a_b, onde b é a posição da amplitude no estado. Nos estados, os bits-ativos estão em outra cor, para facilitar a visualização gráfica e, consequente compreensão da metodologia adotada.

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

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

Para um exemplo mais detalhado do cálculo de uma nova amplitude, considere o cálculo do novo valor da amplitude referente à posição $\mathbf{a_{0000}}$ ao aplicar a TQ genérica $U \otimes I \otimes U \otimes I$ à um estado 4-dimensional. A MR gerada é $U \otimes U$, pois a TQ possui operadores diferentes de I no primeiro e no terceiro qubit, e está descrita na Eq. 19.

Seguindo os passos apresentados anteriormente, tem-se que
(i) ao concatenar os bits-ativos da posição da amplitude, os quais estão identificados com vermelho, temos o valor $\mathbf{10}$ que será a linha a ser utilizada;
(ii) cada elemento da linha $\mathbf{10}$ é multiplicado por uma amplitude – $(\mathbf{m_{1000}} \times \mathbf{a_{0000}})$ – $(\mathbf{m_{0001}} \times \mathbf{a_{0010}})$ – $(\mathbf{m_{1010}} \times \mathbf{a_{1000}})$ – $(\mathbf{m_{0011}} \times \mathbf{a_{1010}})$; e
(iii) atribui-se a soma das multiplicações à posição $\mathbf{0000}$ do estado de escrita.

$$U \otimes U = \begin{pmatrix} \mathbf{m_{0000}} & \mathbf{m_{0001}} & \mathbf{m_{0010}} & \mathbf{m_{0011}} \\ \mathbf{m_{0100}} & \mathbf{m_{0101}} & \mathbf{m_{0110}} & \mathbf{m_{0011}} \\ \mathbf{m_{1000}} & \mathbf{m_{0001}} & \mathbf{m_{1010}} & \mathbf{m_{0011}} \\ \mathbf{m_{1100}} & \mathbf{m_{0001}} & \mathbf{m_{1110}} & \mathbf{m_{0011}} \end{pmatrix} \tag{19}$$

Esta otimização também pode ser aplicada aos operadores controlados. Nestes casos, somente as posições do estado de escrita que satisfazem o controle são calculadas, as que não satisfazem são atribuídas o valor no estado de leitura correspondente a sua posição. Como exemplo, considere a aplicação de um operador controlado genérico ao primeiro qubit, com controle em 1 no segundo qubit, o cálculo ficaria como descrito na Eq.20.

$$\begin{pmatrix} \mathbf{m_{00}} & \mathbf{m_{01}} \\ \mathbf{m_{10}} & \mathbf{m_{11}} \end{pmatrix} \times \begin{pmatrix} \mathbf{a_{00}} \\ \mathbf{a_{01}} \\ \mathbf{a_{10}} \\ \mathbf{a_{11}} \end{pmatrix} = \begin{pmatrix} \mathbf{a_{00}} \\ \mathbf{a_{01}} \times \mathbf{m_{00}} + \mathbf{a_{11}} \times \mathbf{m_{01}} \\ \mathbf{a_{10}} \\ \mathbf{a_{01}} \times \mathbf{m_{10}} + \mathbf{a_{11}} \times \mathbf{m_{11}} \end{pmatrix} \tag{20}$$

Embora este conceito otimize de forma significativa a representação de TQs envolvendo operadores I, nem todas apresentam operadores I ou uma quantidade suficiente que torne possível representá-las através de uma única matriz na memória.

Assim, para superar esta limitação, a próxima otimização considera a decomposição de TQs.

4.2 Decomposição de TQs

Uma TQ n-dimensional pode ser decomposta aumentando o número de passos para o cálculo da mesma, distribuindo os operadores diferente de I ao longo destes passos, preservando o comportamento e propriedades da TQ. Está decomposição permite o controle (e/ou incremento) da quantidade de operadores I presentes em cada etapa de uma aplicação quântica.

Na Figura 20, mostra-se a decomposição da transformação $H \otimes H$ em duas etapas, $H \otimes I$ e $I \otimes H$, mantendo o mesmo comportamento da aplicação independentemente da ordem de composição destas etapas.

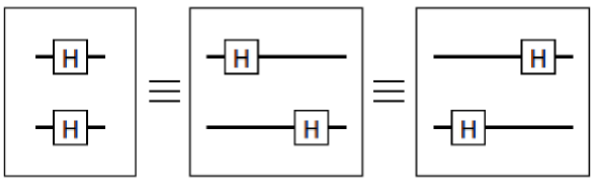


Figura 20: Transformação não-controlada decomposta

TQs controladas também podem ser decompostas, desde que os operadores conservem os controles associados a eles, como mostra a Figura 21.

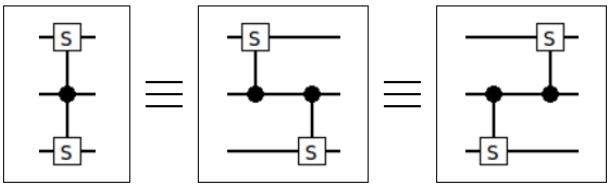


Figura 21: Transformação controlada decomposta

Utilizando este conceito, pode-se maximizar o impacto da otimização descrita na

seção anterior, pois poderá ser usada mesmo quando a quantidade de operadores I da TQ for pequena ou até mesmo ausente.

Estas duas otimizações viabilizam a simulação de um algoritmo quântico de foma eficiente, os quais não fazem uso de matrizes de menor ordem para simular o comportamento do produto tensor e geração, por simulação via computadores convencionais, dos novos estados quânticos.

No caso das otimizações por reduções, a complexidade espacial pode ser reduzida, limitando o número de operadores diferentes de I presentes em cada passo da decomposição e tornando possível a representação na memória de cada passo por uma única matriz reduzida.

Além deste resultado, tem-se também a redução da complexidade temporal. Pois, mesmo com o aumento do número de passos, não se tornam mais necessárias iterações, o que diminui muito o número total de acessos a memória e de multiplicações envolvendo números complexos.

4.3 Processos Mistos Parciais

Apesar de ser possível modelar TQs com baixa complexidade espacial, usando ambas abordagens mostradas nas seções anteriorores, o tamanho dos estados de leitura/escrita podem se tornar um fator limitante para simulação de TQs n-dimensionais, pois estes também crescem de forma exponencial (2^n). Por exemplo, uma TQ multidimesional, na ordem de 28 qubits, precisa 4 GiB de memória para armazenar ambos estados.

Uma vez que a memória das *GPUs* normalmente são menores que a memória RAM principal, é necessário adotar uma abordagem que forneça escalabilidade para a simulação de TQs multi-qubits.

O conceito de Processos Mistos Parciais (MPP), apresentando em (AVILA et al., 2015), provê controle sobre o tamanho dos estados de leitura/escrita no cálculo de uma TQ, contribuindo para o aumento da escalabilidade.

Baseando-se neste conceito, TQs com mais qubits que o limite suportado pela memória da *GPU*, podem ter seus estados particionados em 2^p sub-estados, onde p indica o número de qubits acima do limite, tornando possível a simulação da TQ.

Usando as otimizações descritas anteriormente, o número de sub-estados de leituras que cada sub-estado de escrita precisa acessar para realizar o cálculo de suas amplitudes é 2^r, sendo r o número de operadores afetados pelo particionamento.

Por operadores afetados entende-se o número de operadores diferentes de I presentes nos p primeiros qubits da etapa da computação. Logo, etapas que não possuam operadores afetados precisam somente do correspondente sub-estado de leitura, o que as torna totalmente independentes.

Para manter a consistência do resultado das computações concorrentes na aplicação em simulação, cada etapa incluindo operadores afetados precisa calcular os todos os sub-estados antes de prosseguir para o próxima etapa, respeitando as dependências geradas.

As etapas envolvendo operadores não-afetados podem ser calculadas da mesma forma ou podem ter cada sub-estado calculado de forma iterativa, pois não há dependências, ou seja, cada sub-estado de escrita precisa somente do correspondente sub-estado de leitura para realização do cálculo.

No caso de envolverem operadores afetados, somente os sub-estados que satisfaçam tais controles precisam ser calculados e acessados para leitura.

4.4 Considerações Finais

Este capítulo descreveu as principais otimizações propostas pela estratégia ReDId para a concepção e modelagem de um novo algoritmo visando a redução da complexidade espacial e temporal das simulações quânticas.

Fez-se uso de construtores como replicação e esparcialidade para viabilizar uma representação de TQs otimizada pelo uso inteligente do operador Identidade.

Além destes, a escalabilidade das aplicações pode ser incrementada ao considerar o conceito de decomposição de operações concorrentes.

No Capítulo 5, apresentam-se exemplos dos conceitos que modelaram a concepção do novo algoritmo, e validaram a simulação de algoritmos quânticos como TQ de Hadamard, TQF e o algoritmo de Shor.

5 ESTUDO DE CASO: IMPLEMENTAÇÃO DA ESTRATÉGIA REDID NO D-GM

Este capítulo descreve o ambiente D-GM e seus principais componentes e a implementação da estratégia ReDId neste ambiente, considerando uma arquitetura de software heterogênea, viabilizando a computação via *CPUs* e *GPUs*.

5.1 Ambiente D-GM

O projeto D-GM propõe um *framework* (AVILA et al., 2014) para simulação de algoritmos quânticos, com interfaces gráficas para modelagem, desenvolvimento e implementação de simulações, sequencialmente ou em paralelo, usando *GPUs* e *CPUs*. A Figura 22 apresenta a organização do D-GM em seus diferentes níveis.

Cada nível diferente tem sua própria funcionalidade no sentido da unificação da modelagem e simulação de aplicações:

Quantum Circuit Level: provê a descrição de aplicações no modelo de circuitos quânticos através do uso das ferramentas para modelagem e edição qCEdit (*Quantum Circuit Editor*) e sua versão para plataformas móveis M-qCEdit (GNUTZMANN, 2013), as quais fornecem a opção de automaticamente exportar as aplicações para uma representação de acordo com o modelo qGM (Quantum Geometric Machine).

qGM Level: contendo o ambiente *VPE-qGM* que tem o objetivo de oferecer suporte a modelagem e simulação de algoritmos quânticos considerando as abstrações do modelo *qGM*. Dentre os componentes arquiteturais do ambiente, os seguintes são destacados:

- *qPE* (Quantum Process Editor): IDE para o desenvolvimento gráfico dos algoritmos;
- *qME* (Quantum Memory Editor): interface para definição do estado inicial do sistema quântico. Um grid de memória armazena cada estado básico e a correspondente, modelando um vetor de estado de forma análoga à descrita na Eq. 1;

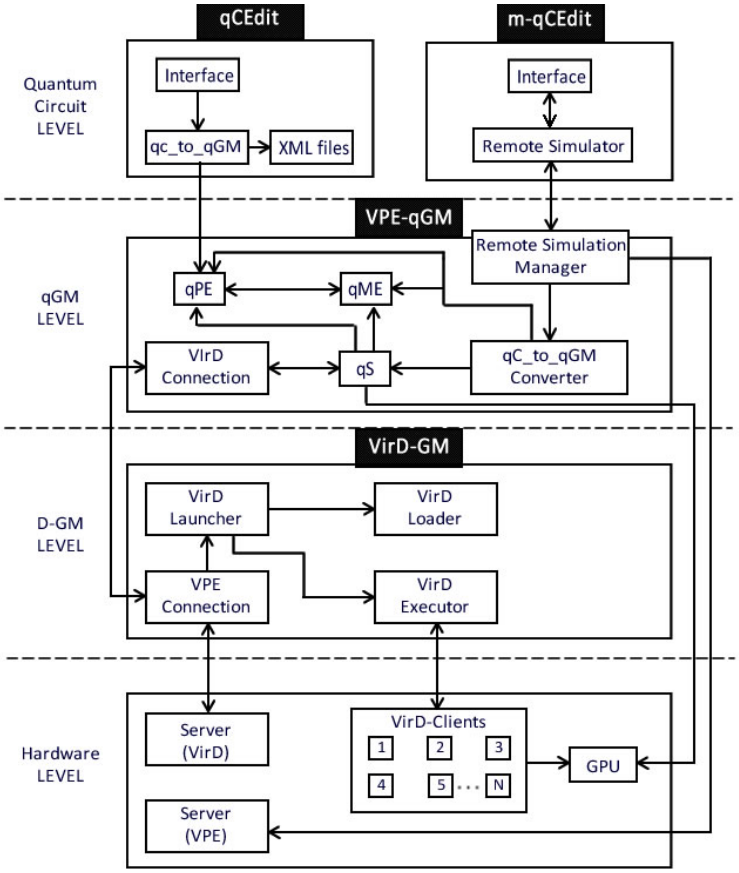


Figura 22: *Framework* do Simulador D-GM. Fonte: (AVILA et al., 2014)

- *qS* (Quantum Simulator): A partir das estruturas definidas nas interfaces do *qPE* e *qME*, o *qS* realiza a simulação do algoritmo quântico, suportando duas abordagens: (i) simulação passo-a-passo, a qual é relevante para viabilizar uma análise detalhada da evolução da simulação; (ii) simulação distribuída;

- *qGM-Analyzer*: Biblioteca para realização das computações associadas aos componentes que descrevem cada passo do algoritmo quântico, estando integrada à interface *qS*. O suporte para aceleração por *GPU* é estabelecida nesta biblioteca.

D-GM Level: módulo de gerenciamento de simulações distribuídas, *Virtual Distributed Geometric Machine* (*VirD-GM*) (AVILA et al., 2014), que se encarrega pela comunicação, escalonamento e sincronização quando uma simulação distribuída é requisitada pelo *VPE-qGM*. A execução distribuída de aplicações a partir do *VirD-GM* se dá através do uso de Processos Mistos Parciais. Os seus principais componentes são descritos abaixo.

- *VirD-Loader*: responsável pela interpretação de arquivos descritores contendo o algoritmo a ser simulado e seu vetor de estado inicial.

- *VirD-Launcher*: realiza o escalonamento e controle de fluxo de execução das tarefas.
- *VirD-Exec* controla a comunicação e transferência de dados entre os clientes de execução, nomeados *Vird-Clients*.

5.2 Implementações

Nesta seção, descreve-se a implementação de uma nova biblioteca de execução para o ambiente D-GM em linguagem C/C++ que forneça suporte a simulação de algoritmos quânticos multi-qubits, sequencialmente e em paralelo, em *CPUs* e *GPUs*. Esta implementação utiliza a estratégia ReDId em acordância com a descrição já apresentada no Capítulo 4, que visa a redução da complexidade espacial e temporal nas simulações e aplicações.

A estratégia adotada para realizar a decomposição de TQ no ambiente D-GM pode ser dividida em duas partes:

(1) **classificação da TQ em grupos**, separando operadores não-controlados e com controles distintos.

(2) **definição dos passos da TQ**, cada um é formado por operadores que pertençam ao mesmo grupo e atuem em qubits consecutivos respeitando o limite de operadores por passo estabelecido. Em caso de particionamento da memória, operadores afetados e não-afetados não podem fazer parte do mesmo passo.

Um exemplo é mostrado na Figura 23 para uma TQ de 9 qubits considerando limites de 3 operadores por passo e de 8 qubits para execução. Seguindo os passos descritos anteriormente, a TQ primeiro é dividida em 3 grupos e então em 5 passos. O grupo 1 gera dois passos apesar de ter 3 operadores consecutivos, pois o primeiro qubit é afetado pelo particionamento da memória.

Nas sub-seções a seguir são discutidas as implementações do algoritmo para *CPU* e *GPU*.

5.2.1 Computação em CPU

Os melhores resultados em *CPU* usando a abordagem de decomposição descrita acima foram alcançados quando o limite de operadores por passo de 1 e 2 foram considerados. Por isso, foi optado por calcular a TQ operador por operador, o mesmo que limite de operadores de 1, à fim de realizar otimizações direcionados ao tipo de operador calculado. Os operadores podem ser classificados em dois tipos:

(i) **Denso** - operadores definidos por matrizes que não possuem valores nulos. Estes operadores não permitem a aplicação de otimizações mais agressivas; e

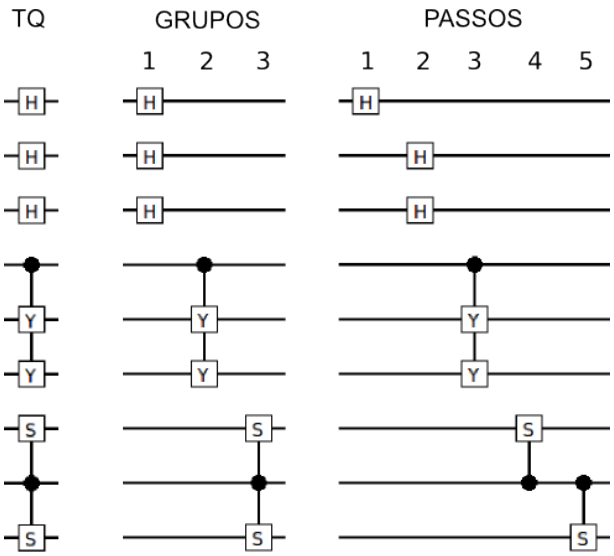


Figura 23: Exemplo de decomposição para uma TQ.

(ii) **Esparsos** - operadores definidos por matrizes com valores não-nulos somente na diagonal principal ou na diagonal secundária. Nesses casos, otimizações descartando computações envolvendo valores nulos podem ser aplicadas.

Para exemplificar estas otimizações, considere a aplicação de operadores genéricos dos dois tipos ao primeiro qubit de um estado 2-dimensional. O cálculo de operadores densos ocorre como descrito em Eq. 17. Para operadores esparsos, Eqs. 21 e 22 definem como cada amplitude pode ser calculada usando um único valor da matriz e do estado, enquanto operadores densos precisam de dois valores de cada estrutura.

$$\begin{pmatrix} \mathbf{m_{00}} & \mathbf{0} \\ \mathbf{0} & \mathbf{m_{11}} \end{pmatrix} \times \begin{pmatrix} \mathbf{a_{00}} \\ \mathbf{a_{01}} \\ \mathbf{a_{10}} \\ \mathbf{a_{11}} \end{pmatrix} = \begin{pmatrix} \mathbf{a_{00}} \times \mathbf{m_{00}} \\ \mathbf{a_{01}} \times \mathbf{m_{00}} \\ \mathbf{a_{10}} \times \mathbf{m_{11}} \\ \mathbf{a_{11}} \times \mathbf{m_{11}} \end{pmatrix} \tag{21}$$

$$\begin{pmatrix} \mathbf{0} & \mathbf{m_{01}} \\ \mathbf{m_{10}} & \mathbf{0} \end{pmatrix} \times \begin{pmatrix} \mathbf{a_{00}} \\ \mathbf{a_{01}} \\ \mathbf{a_{10}} \\ \mathbf{a_{11}} \end{pmatrix} = \begin{pmatrix} \mathbf{a_{10}} \times \mathbf{m_{01}} \\ \mathbf{a_{11}} \times \mathbf{m_{01}} \\ \mathbf{a_{00}} \times \mathbf{m_{10}} \\ \mathbf{a_{01}} \times \mathbf{m_{10}} \end{pmatrix} \tag{22}$$

A execução de uma TQ se dá operador por operador. Para cada um, é identificado o seu tipo e então o correspondente laço é executado a fim de produzir as novas amplitudes. As implementações paralelas em *CPU* foram implementadas em

OpenMP (OpenMP Architecture Review Board, 2015), replicando o código da versão sequencial explicada acima e adicionando a diretiva “parallel for” nos laços onde as novas amplitudes são calculadas.

5.2.2 Computação em GPU

Depois de realizar a decomposição da TQ de acordo com os parâmetros recebidos, o cálculo dos passos é realizado.

Como descrito na Seção 4.3: passos afetados são calculados um por um, sendo realizada uma chamada de kernel para cada combinação entre sub-estados de escritas e sub-estados de leitura necessários para o cálculo. Todos os passos não-afetados são calculados um sub-estado por vez, realizando chamadas de kernel de forma iterativa a fim de reduzir a comunicação entre host e *GPU*, uma vez que o sub-estado de escrita resultante do cálculo de um passo pode ser mantido na memória da *GPU* e servir como sub-estado de leitura para o próximo passo.

Cada chamada do kernel CUDA recebe os seguintes parâmetros para execução:

- Estados (sub-estados) de leitura/escrita;
- Matriz reduzida do passo a ser computado;
- Valores e posições dos controles (se existirem);
- Informações de acesso as estruturas que possuem os dados citados acima.

A computação do kernel CUDA é dividida em 5 passos, descritos a seguir.

Passo 1: Identificação do $lineId$ de cada *thread*, calculado usando informação sobre a *thread* atual e do controle. O $lineId$ define qual amplitude será calculada por cada *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;
```

Passo 2: Inicialização de variáveis locais a cada *thread*.

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

Passo 3: Computação da nova amplitude parcial usando as variáveis do passo anterior, que definem acessos à matriz e estados.

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

Passo 4: Armazenamento e acumulação da nova amplitude, calculada e escrita na memória global da *GPU*.

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

Passo 5: Cópia das amplitudes das posições complementares aos controles do estado de leitura para o estado de escrita.

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

6 RESULTADOS

A principal contribuição deste trabalho pode ser avaliado através da simulação de três classes de TQs:

(1) simulações Hadamard de 21 até 28 qubits;

(2) simulações de Transformadas de Fourier Quântica de 26 até 28 qubits;

(3) simulações do algoritmo de Shor considerando números de 6 até 12 bits, ou seja, aplicações de 15 até 25 qubits.

Simulações sequenciais e paralelas até 4 Threads foram realizadas em *CPU*, os parâmetros considerados para as simulações em *GPU* foram limite de operadores por passo de 1 até 6 e limite de qubits para execução de 26 até 28 a fim de avaliar o comportamento do novo algoritmo do ambiente D-GM.

Os testes foram realizados em um desktop com processador Intel Core i7-3770, 8 GiB RAM, *GPU* NVidia GTX Titan X. Os experimentos foram executados no sistema operacional Ubuntu Linux 14.04, 64 bits, e CUDA TOOLKIT 7.0.

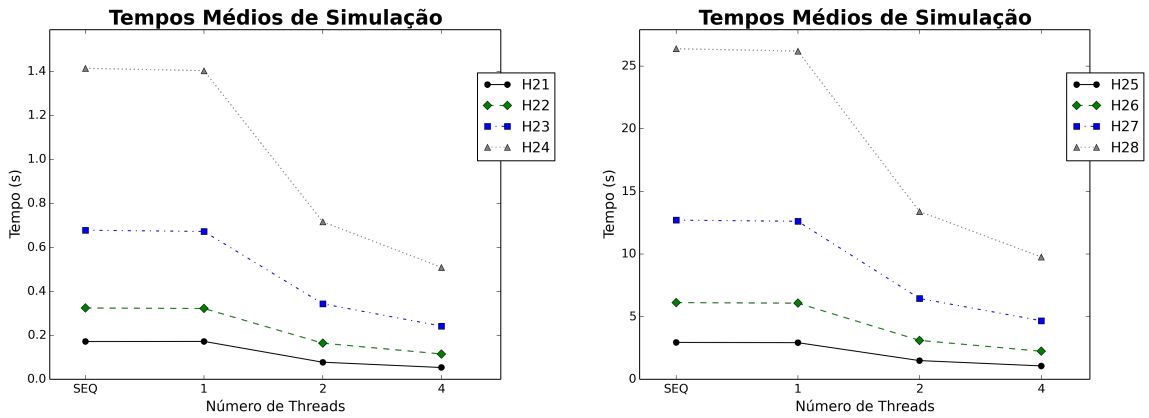
Durante a simulação, os tempos médios de simulação para Hadamard e QFT foram obtidos depois de 30 execuções de cada aplicação, descartando os 5 menores e 5 maiores tempos de execução, e para o algoritmo de Shor depois de 10 execuções por ser uma aplicação com tempo de execução elevado.

Nas seções à seguir são mostrados os resultados das simulações, salienta-se que diferentes escalas foram usados nos gráficos para melhor visualização dos resultados.

6.1 Transformação Hadamard

Tempos de simulação para transformações Hadamard em *CPU* podem ser visto na Figura 24. Nota-se que o tempo de simulação diminui para todas as Hadamard conforme o número de threads aumenta. No entanto, se afasta do ideal devido ao aumento de *cache misses*.

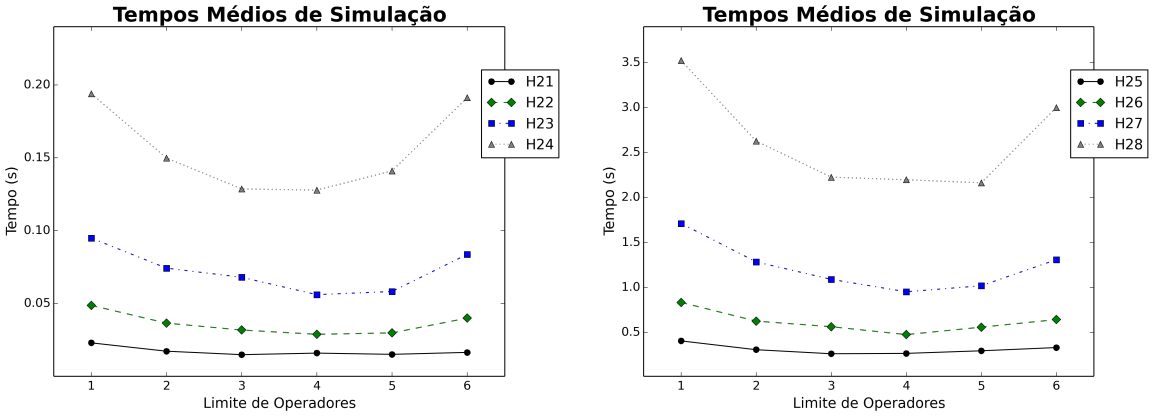
Tempos de simulação para transformações Hadamard em *GPU*, sem considerar limite de qubits para execução, são mostrados na Figura 25. Nota-se que o tempo



(a) 21-24 Qubits (b) 25-28 Qubits

Figura 24: Tempos de execução para a Transformação Hadamard em *CPU*, variando o número de *threads*

mínimo de execução ocorre sempre com limite de operadores por passo de 4 ou 5, mostrando que simulações com estes limites alcançam o melhor desempenho neste *hardware* para estas transformações.



(a) 21-24 Qubits (b) 25-28 Qubits

Figura 25: Tempos de execução para a Transformação Hadamard em *GPU*, variando o limite de operadores

A Figura 26 mostra os tempos de simulação em *GPU* de 26 até 28 qubits, considerando limite de qubits para execução. Como esperado as implementações usando MPPs permitem a simulação mesmo quando o limite de qubits para execução é menor que o número de qubits da transformação sendo calculada. No entanto, o tempo de execução aumenta conforme a quantidade de qubits passados do limite, pois haverá mais operadores afetados pelo particionamento do estado.

Em comparação com o método anterior (AVILA et al., 2015), tempos médios de simulação foram medidos para transformações Hadamard de 21 e 22 qubits com tempos de 110,407 s e 395,951 s respectivamente. O speedup relativo obtido comparando

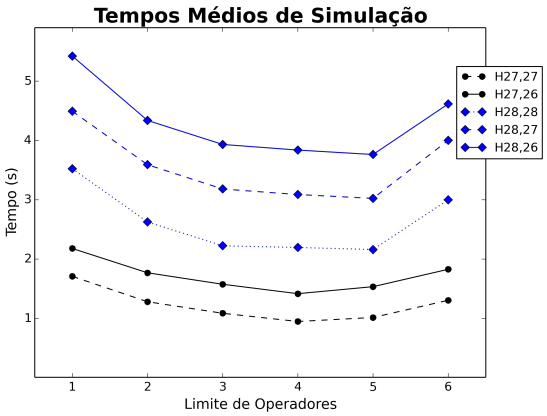


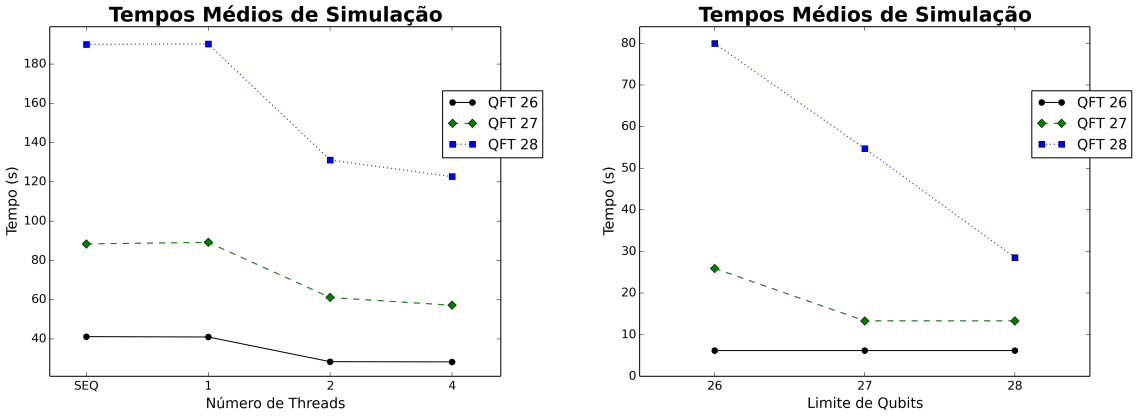
Figura 26: Transformação Hadamard, 27-28 Qubits, 26-28 Limite, *GPU*

nosso melhor resultado neste trabalho com o método anterior foi de $\approx 10.829\times$.

Este speedup tende a escalar com o número de qubits, pois a taxa de crescimento do tempo conforme o aumento do número de qubits é de $\approx 2\times$, enquanto no método anterior é de $\approx 3,6\times$.

6.2 Transformada de Fourier Quântica

A Figura 27(a) mostra os tempos de simulação em *CPU* para TFQ, mostrando ganho de desempenho com o aumento do número de threads. Por ser uma aplicação composta basicamente por operadores esparsos controlados, a maioria dos cálculos se dão pelos laços de execução otimizados promovendo um menor impacto no ganho das simulações paralelas se comparado aos resultados da transformação Hadamard.



(a) *CPU* (b) *GPU*, 26-28 Limite

Figura 27: Tempos de execução para a Transformada de Fourier Quântica, 26-28 qubits.

O limite de operadores por etapa não é relevante para a TFQ pois ela possui somente um operador por passo. Na Tabela 3 é mostrado o número de passos afetados

pela relação entre o número de qubits (etapas) da aplicação e o limite de qubits para execução, e com base nestes dados pode ser observado na Figura 27(b) o ganho de desempenho das simulações da TFQ em *GPU* conforme o número de etapas afetadas pelo limite de qubits diminui.

Tabela 3: Número de etapas afetadas pelo limite de qubits em uma TFQ.

QB(ET) \ LIM	26	27	28
26 (351)	0	0	0
27 (378)	27	0	0
28 (406)	55	28	0

QB (ET) - Qubits (Etapas). LIM - Limite de qubits para execução.

6.3 Algoritmo de Shor

Tempos de simulação para o algoritmo de Shor usando os simuladores LIQ$Ui|\rangle$ e D-GM podem ser vistos na Tabela 4. Tempos de execução do LIQ$Ui|\rangle$ foram obtidos à partir do "minutes for running" encontrado na saída da execução da versão *built-in* do algoritmo de Shor com a opção de otimização em "true".

Salienta-se que o simulador D-GM apresentou melhores resultados para todos os números executados independente do tipo de execução realizada. As execuções em *CPU* mostraram ganho de performance conforme o número de *threads* aumenta.

Assim como esperado, as execuções em *GPU* obtiveram os melhores tempos em todos os casos quando a parte quântica do algoritmo de Shor se torna o gargalo da simulação ao invés do processamento clássico necessário antes e depois desta parte, que seria para fatoração de números com 7 ou mais bits (17 ou mais qubits).

Tabela 4: Tempos Médios de Simulação para o Algoritmo de Shor, em segundos.

			LIQ$Ui\|\rangle$	*D-GM*				
Número	*Bits*	*Qubits*		Seq.	1 Thread	2 Threads	4 Threads	*GPU*
57	6	15	11,01	1,49	1,53	0,92	0,87	0,99
119	7	17	47,00	9,26	9,36	5,01	3,41	2,64
253	8	19	201,39	57,10	58.02	30.27	17,70	10,17
485	9	21	1.166,36	358,89	348,49	239,69	211,64	50,84
1.017	10	23	5.905,24	2.076,11	2.055,28	1.389,96	1.249,54	280.67
2.045	11	25	LS	NE	NE	NE	NE	1.623,87

Seq. - Sequencial. LS - Limite do Simulador. NE - Não Executado.

6.4 Considerações Finais

As otimizações propostas pela estratégia ReDId para melhorar o desempenho da simulação de computação quântica foram implementadas no *framework* D-GM, mas

não se restringe a este ambiente, podendo ser implementada em outras plataformas e/ou simuladores realizando as adaptações que virem a ser necessárias.

Pela exploração das características do operador Identidade e uso da decomposição de TQ, foi possível criar um algoritmo mais eficiente, permitindo a simulação de aplicações com um grande número de qubits em uma única máquina.

O uso do particionamento dos estados proporcionou escalabilidade às simulações em *GPU*, permitindo desenvolvimento de aplicações com uma quantidade de memória maior que a disponível na *GPU*, Experimentos mostraram simulações de até 28 qubits nesta arquitetura.

Quando comparado com nosso método anterior (AVILA et al., 2015), o melhor *speedup* relativo foi obtido para a Hadamard de 22 qubits, sendo $10,829\times$ mais rápido usando esta arquitetura otimizada.

Simulações do algoritmo de Shor mostraram melhores resultados no *framework* D-GM quando comparado ao simulador LIQUi$|\rangle$. Para o maior número fatorado (10 bits, 23 qubits), a execução sequencial em *CPU* foi $2,84\times$ mais rápida, a execução paralela $4,72\times$ mais rápida com 4 *threads*, e a execução em *GPU* $21,03\times$ mais rápida.

7 CONCLUSÃO

A simulação de algoritmos quânticos em computadores clássicos é uma tarefa que requer alta capacidade de processamento, mas tem-se tornado uma das alternativas viáveis para o estudo e desenvolvimento de aplicações, modelando e simulando o comportamento de sistemas multi-qubits de forma simplificada.

Entretanto, explorar isoladamente os recursos computacionais providos por arquiteturas paralelas (*clusters*, grids, *GPUs*) é insuficiente quando da simulação de algoritmos quânticos complexos, uma vez que sistemas quânticos exigem recursos de processamento e memória exponencialmente maiores.

Para colaborar com novas estratégias de enfrentar tais problemas, este trabalho foca nas possibilidades de extensão da capacidades de simulação do ambiente *D-GM* (AVILA et al., 2014), abrangendo o estudo e implementação de otimizações para representação e simulação sequencial/paralela de transformações quânticas.

Foram implemnetadas técnicas capazes de reduzir de forma significativa o custo (memória e processamento) de simulação, viabilizando o desenvolvimento de aplicações quânticas complexos (como algoritmos de fatoração).

Destacam-se, na sequência, os principais resultados.

(i) Estudo e concepção de nova estratégia de simulação, denominada ReDId:

- minimizando a replicação e explorando a esparcialidade dos operadores Identidade o projeto provê uma redução significativa da complexidade espacial e, quando combinado a decomposição de transformações quânticas, têm-se também a redução da complexidade temporal das simulações;
- considerando a parcialidade na definição dos operadores e estados, os resultados mostram incremento na escalabilidade das computações durante a simulação de aplicações, o que leva ao incremento da dimensão dos sistemas multi-qubits simulados (AVILA et al., 2015).

(ii) Modelagem e implementação do algoritmo de execução que considera estas estratégias na linguagem C/C++ para as execuções em *CPU*, e em *CUDA* para as

execuções em *GPUs*, a fim de obter o maior desempenho possível em ambas arquiteturas;

(iii) Validação das implementações, considerando:

- simulações de operadores Hadamard de 21 até 28 qubits, sendo $10,829\times$ mais rápido que a versão anterior (AVILA et al., 2015);
- simulações de Transformada de Fourier Quântica de 26 até 28 qubits (AVILA et al., 2015);
- simulações do algoritmo de Shor considerando números de 6 até 12 bits, ou seja, aplicações de 15 até 25 qubits, e quando comparada com $LIQUi|\rangle$, sua execução sequencial em *CPU* foi $2,84\times$ mais rápida, a execução paralela $4,72\times$ mais rápida com 4 *threads*, e a execução em *GPU* $21,03\times$ mais rápida (AVILA; REISER; PILLA, 2016a,b).

(iv) Introduz-se os primeiros passos para a concepção e o desenvolvimento de uma arquitetura de software híbrida para o projeto D-GM, viabilizando a computação distribuída (via *CPUs*) sob arquiteturas massivamente paralelas (via *GPUs*). Obteve-se os seguintes resultados:

- especificação de um algoritmo otimizado para simulação de aplicações quânticas fazendo uso da estratégia ReDId, incluindo sua implementação no ambiente D-GM e seus principais componentes, o *VPE-qGM* e o *VirD-GM*.
- modelagem e simulação distribuída de algoritmos quânticos, apresentando as construções e evolução dos sistemas quânticos a partir de um conjunto de *interfaces* gráficas.

7.1 Trabalhos Futuros

A maior contribuição deste trabalho está associada à integração desses dois esforços: (i) a busca por otimizações das estruturas associadas a transformações e estados quânticos; e (ii) a paralelização das operações de evolução do estado em sistema multi-qubits considerando tais otimizações.

A consolidação desta integração e novas otimizações constituem um grande desafio de pesquisa, porém a perspectiva de bons resultados é uma grande motivação na continuidade do projeto e busca por soluções mais eficientes para simulação de algoritmos quânticos a partir de computadores clássicos.

Os principais trabalhos futuros, focados no projeto D-GM, são brevemente descritos a seguir:

(1) Potencialização da dinâmica de execução do ambiente D-GM e consolidação de seus principais componentes:

   (i) o *VPE-qGM* e o *VirD-GM*, com potencial para viabilizar a simulação de algoritmos quânticos *multi-qubits*, Investiga-se a grande capacidade de processamento que pode ser obtida pela exploração de sistemas híbridos que exploram a computação com multicomputadores e/ou multiprocessadores;

   (ii) o ShareD-GM, estendendo a proposta de memória compartilhada e distribuída a partir de novas estratégias de armazenamento, controle e acesso de estados quânticos. A modelagem e implementação de uma nova forma de representação e/ou armazenamento dos estados *multi-qubits*. Investigação de recursos de controle e monitoramento do inerente crescimento exponencial associado ao número de qubits das aplicações quânticas, aliados à capacidade de simulação distribuída já consolidada no ambiente D-GM.

(2) A exploração das potencialidades disponíveis no ambiente D-GM para execução de sistemas fuzzy e investigação de como ambientes de simulação quântica são capazes de codificar conjuntos fuzzy (CFs) e suas operações lógicas a partir de operadores e transformações quânticas *multi-qubits*.

   (i) Extensão da biblioteca de métodos qGM-Analyzer para descrição de conectivos fuzzy via operadores quânticos;

   (ii) Modelagem de conjuntos fuzzy por registadores quântico e operadores fuzzy por transformações quânticas, considerando a incerteza tanto na expressão das variáveis linguísticas quanto a incerteza inerente aos fenómenos físicos presentes nos sistemas reais.

REFERÊNCIAS

AARONSON, S. **Shtetl-Optimized**: Shor, I'll do It. 2007. Disponível em http://scottaaronson.com/blog/?p=208.

AVILA, A. B. de; REISER, R. H. S.; PILLA, M. L. **Quantum computing simulation through reduction and decomposition optimizations with a case study of Shor's algorithm**. Concurrency and Computation Practice and Experience, 2016. (Submetido).

AVILA, A. B. de; SCHUMALFUSS, M. F.; REISER, R. H. S.; PILLA, M. L.; MARON, A. K. Optimizing Quantum Simulation for Heterogeneous Computing: a Hadamard Transformation Study. **Journal of Physics: Conference Series**, v.649, n.1, p.012004, 2015.

AVILA, A. B.; REISER, R. H. S.; PILLA, M. L. **Reduction and decomposition optimizations in quantum computing simulation applied to the Shor's algorithm**. Gramado: CNMAC, 2016. 1-7p. XXXVI Congresso Nacional de Matemática Aplicada a Computação (Submetido).

AVILA, A. B.; SHMALFUSS, M. F.; REISER, R. H. S.; PILLA, M. L. Otimização de Simulação de Computação Quântica Através da Redução e Decomposição Baseados no Operador Identidade. In: XVI SIMPÓSIO EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO (WSCAD), 2015, Florianópolis. **Anais...** SBC, 2015. p.1–12.

AVILA, A. B.; SHMALFUSS, M. F.; REISER, R. H. S.; PILLA, M. L.; MARON, A. K. Simulação Distribuída de Algoritmos Quânticos via GPUs. In: XV SIMPÓSIO EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO (WSCAD), 2014, São José dos Campos. **Anais...** SBC, 2014. p.1–12.

AVILA, A. B.; SHMALFUSS, M. F.; REISER, R. H. S.; PILLA, M. L.; YAMIN, A. Salable quantum simulation by reductions and decompositions through the id-operator. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, 30., 2015, Salamanca. **Proceedings...** ACM, 2015. p.1–3. (SAC '15).

AVILA, A.; MARON, A.; REISER, R.; PILLA, M.; YAMIN, A. GPU-aware Distributed Quantum Simulation. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, 29., 2014, New York, NY, USA. **Proceedings...** ACM, 2014. p.860–865. (SAC '14).

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

GNUTZMANN, P. **m-qCEdit**: Um Ambiente para Modelagem e Simulação de Circuitos Quânticos via Plataformas Móveis. Comment: 39 páginas, Monografia de Graduação PPGC/UFPEL, 2013.

GROUP, Q. C. **QuIDDPro**: High-Performance Quantum Circuit Simulator. The University of Michigan, 2007. Disponível em http://vlsicad.eecs.umich.edu/Quantum/qp/results.html (mar.2016).

GROVER, L. K. A Fast Quantum Mechanical Algorithm for Database Search. In: TWENTY-EIGHTH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, 1996, New York, NY, USA. **Proceedings...** ACM, 1996. p.212–219. (STOC '96).

GUTIÉRREZ, E.; ROMERO, S.; TRENAS, M. A.; ZAPATA, E. L. Quantum computer simulation using the CUDA programming model. **Computer Physics Communications**, v.181, p.283–300, Feb. 2010.

HEY, T. Richard Feynman and computation. **Contemporary Physics**, v.40, n.4, p.257–265, 1999.

KNILL, E. H.; NIELSEN, M. A. **Theory of quantum computation**. Kluwer, 2000. Comment: 5 pages.

MARON, A. **Explorando as Possibilidades de Otimização da Simulação de Algoritmos Quânticos no VPE-qGM**. Comment: 71 páginas, Dissertação de Mestrado PPGC/UFPEL, 2013.

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

MARON, A.; REISER, R.; PILLA, M.; YAMIN, A. Quantum Processes: A New Interpretation for Quantum Transformations in the VPE-qGM Environment. In: CLEI, 2012. **Anais...** IEEE Computer Society - Conference Publishing Services, 2012. p.1–10.

NIELSEN, M. A.; CHUANG, I. L. **Computação Quântica e Informação Quântica**. Bookman, 2003.

NIWA, J.; MATSUMOTO, K.; IMAI, H. General-Purpose Parallel Simulator for Quantum Computing. In: THIRD INTERNATIONAL CONFERENCE ON UNCONVENTIONAL MODELS OF COMPUTATION, 2002, London, UK, UK. **Proceedings...** Springer-Verlag, 2002. p.230–251. (UMC '02).

ÖMER, B. A. Procedural Formalism for Quantum Computing. In: ACM SIGPLAN 2003 HASKELL WORKSHOP, 1998. **Proceedings...** ACM, 1998.

OpenMP Architecture Review Board. **The OpenMP API specification for parallel programming**. Disponível em http://openmp.org/wp/openmp-specifications.

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 Matemáitica Aplicada - SBMAC, 2004.

RAEDT, K. D.; MICHIELSEN, K.; RAEDT, H. D.; TRIEU, B.; ARNOLD, G.; RICHTER, M.; LIPPERT, T.; WATANABE, H.; ITO, N. **Massive Parallel Quantum Computer Simulator**. Quantum Physics, 2006. http://arxiv.org/abs/quant-ph/0608239.

SHOR, P. W. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, 35., 1994, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 1994. p.124–134.

SHOR, P. W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. **SIAM J. Comput.**, Philadelphia, PA, USA, v.26, n.5, p.1484–1509, Oct. 1997.

STEFFEN, L.; SALATHE, Y.; OPPLIGER, M.; KURPIERS, P.; BAUR, M.; LANG, C.; EICHLER, C.; PUEBLA-HELLMANN, G.; FEDOROV, A.; WALLRAFF, A. Deterministic quantum teleportation with feed-forward in a solid state system. **Nature**, v.500, p.319–322, 2013.

THE Double-Slit Experiment. 2009. Disponível em http://www.doubleslitexperiment.com/.

VIAMONTES, G. **Efficient Quantum Circuit Simulation**. 2007. Phd Thesis — The University of Michigan.

WECKER, D.; SVORE, K. M. **LIQ**$Ui|\rangle$: A Software Design Architecture and Domain-Specific Language for Quantum Computing. 2014. Disponível em http://arxiv.org/abs/1402.4467.