Universidade Federal Rural de Pernambuco
Departamento de Estatística e Informática
Programa de Pós-Graduação em Informática Aplicada

LinDCQ: Uma linguagem para descrição de circuitos quânticos que possibilita o cálculo das operações na GPU utilizando JOCL

Mouglas Eugênio Nasário Gomes

Recife

Outubro de 2015

Mouglas Eugênio Nasário Gomes

LinDCQ: Uma linguagem para descrição de circuitos quânticos que possibilita o cálculo das operações na GPU utilizando JOCL

Orientador: Prof. Dr. Wilson Rosa de Oliveira Júnior
Coorientador: Prof. Dr. Tiago Alessandro Espínola Ferreira

Dissertação de mestrado apresentada ao Curso de Pós-Graduação em Informática Aplicada da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do grau de Mestre em Ciências da Computação.

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES

Recife

Outubro de 2015

Ficha Catalográfica

G366l Gomes, Mouglas Eugênio Nasário
LinDCQ: Uma linguagem para descrição de circuitos quânticos que possibilita o cálculo das operações na GPU utilizando JOCL / Mouglas Eugênio Nasário Gomes. – Recife, 2015.
114 f.: il.

Orientador(a): Wilson Rosa de Oliveira Júnior.
Dissertação (Programa de Pós-graduação em Informática Aplicada) – Universidade Federal Rural de Pernambuco, Departamento de Estatística e Informática, Recife, 2015.
Inclui apêndice(s), anexo(s) e referências.

1. Computadores – Circuitos 2. Computação 3. GPU 4. LinDCQ (Linguagem de Programação de Computador) 5. Linguagem de progração (Computadores) 6. Circuitos quânticos I. Oliveira Júnior, Wilson Rosa de, orientador II. Título

CDD 005.13

Universidade Federal Rural de Pernambuco

Departamento de Estatística e Informática

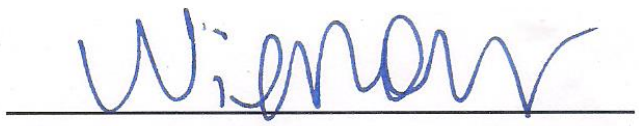
Programa de Pós-Graduação em Informática Aplicada

LinDCQ: Uma linguagem para descrição de circuitos quânticos que possibilita o cálculo das operações na GPU utilizando JOCL

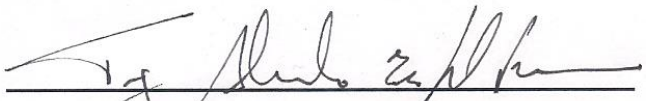
Mouglas Eugênio Nasário Gomes

Dissertação julgada adequada para obtenção do título de Mestre em Ciências da Computação, defendida e aprovada por unanimidade em 27/07/2015 pela Banca Examinadora.

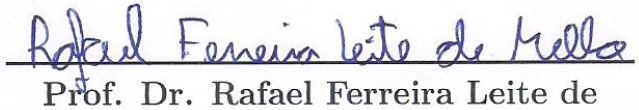
Banca Examinadora:



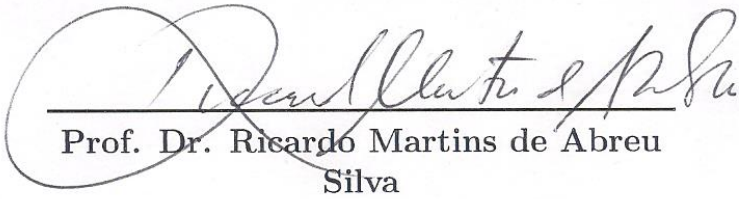
Prof. Dr. Wilson Rosa de Oliveira Júnior (orientador)
Universidade Federal Rural de Pernambuco



Prof. Dr. Tiago Alessandro Espínola Ferreira (coorientador)
Universidade Federal Rural de Pernambuco



Prof. Dr. Rafael Ferreira Leite de Mello
Universidade Federal Rural de Pernambuco



Prof. Dr. Ricardo Martins de Abreu Silva
Universidade Federal de Pernambuco

Agradecimentos

Agradeço primeiramente a Deus, e a minha Família por sempre terem acreditado em mim. Agradeço também a todos os meus colegas de Pós-graduação, tanto alunos, quanto professores, pelo companheirismo que tivemos no decorrer do curso. E por fim agradeço, também, a o meu orientador Prof. Dr. Wilson Rosa de Oliveira Junior, por ter me apresentado este campo de estudo, que é a computação quântica.

Resumo

Este trabalho apresenta a ferramenta LinDCQ — uma linguagem de descrição e programação de circuitos quânticos — a qual possibilita a criação de circuitos quânticos com cálculo das operações realizados de forma paralela na GPU, utilizando JOCL. A ferramenta também permite a geração do circuito de forma gráfica. Utiliza gramáticas como mecanismo na geração de linguagens e autômatos como mecanismo reconhecedor de linguagens e de expressões regulares. Nesse contexto é apresentada uma discussão sobre as fases dos compiladores e sobre a computação quântica, assim como uma explanação sobre as principais tecnologias utilizadas para o desenvolvimento de circuitos quânticos. A ferramenta LinDCQ é composta de: gramática no formato BNF (Backus-Naur-Form), compilador que verifica a incidência de erros no código a ser executado, de uma interface gráfica com características facilitadoras à programação que permite a construção do circuito de forma gráfica e de algoritmos paralelos em JOCL para executar as operações que requerem maior custo computacional na GPU. Ao final é realizado um experimento com o intuito de aferir a usabilidade da ferramenta, para, deste modo, garantir um maior um nível de aceitação do usuário, facilitando a interação do mesmo com a ferramenta desenvolvida nesta dissertação.

Palavras-chave: Circuitos, Computação, GPU, Linguagens, Quântica.

Abstract

This paper presents the LinDCQ tool — a description language and programming quantum circuits — which enables the creation of quantum circuits with calculus of operations performed in parallel on the GPU, using JOCL. The tool also allows the generation of graphically circuit. Used as a mechanism to generate grammars of languages and automata as language recognizer and the regular expression engine. In this context a discussion of the phases of compilers and on quantum computation is presented as well as an explanation of the main technologies used for the development of quantum circuits. LinDCQ The tool consists of: grammar in BNF form (Backus-Naur-Form), the compiler verifies that the incidence of errors in the code to be executed, a graphical interface to facilitate the programming features that allow the construction of the circuit graphically and parallel algorithms JOCL to perform operations that require greater computational cost in the GPU. At the end of an experiment is performed in order to assess the usability of the tool, to thereby ensure a higher level of user acceptance, facilitating interaction thereof with the tool developed in this work.

Keywords: Circuits, Computer, GPU, Languages, Quantum.

Sumário

Lista de Abreviaturas

AFD	*Autômato Finito Determinístico*
ALU	*Arithmetic Logic Unit*
BNF	*Backus Naur Form*
CPU	*Central Processing Unit*
CQ	*Computação Quântica*
CUDA	*Compute Unified Device Architecture*
ER	*Expressões Regulares*
GALS	*Gerador de Analisadores Léxicos e Sintáticos*
GNU	*General Public License*
GPU	*Graphics Processing Unit*
GPGPU	*General Purpose Graphics Processing Unit*
IDE	*Integrated Development Environment*
JCUDA	*Java Compute Unified Device Architecture*
JOCL	*Java bindings for OpenCL*
LinDCQ	*Linguagem para Descrição de Circuitos Quânticos*
LL	*Left-to-rigth Left linear*
LP	*Linguagem de Programação*
LR	*Left-to-rigth Rightmost derivation*
MEF	*Máquinas de Estados Finitos*
MQ	*Mecânica quântica*
OpenCL	*Open Computing Language*
RMN	*Ressonância Magnética Nuclear*
SAUSP	*Sistema Avaliador de Usabilidade em Softwares Pedagógicos*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

1.1 Motivação

É presumível que os sistemas quânticos são exponencialmente poderosos! Por exemplo, imagine que tem-se um pequeno sistema quântico composto por centenas de partículas subatômicas, como elétrons ou fótons, assim, pode-se dizer que tem-se um sistema com algo em torno de 500 partículas. Se fosse possível aproveitar todo este poder computacional, inerente ao que a mecânica quântica promete, para construção de um computador quântico, poderia-se realizar 2^{500} passos computacionais por segundo. Mas o quão grande é 2^{500}? Segundo Vazirani (2013), é muito maior do que a estimativa para o número total de partículas no universo; é também muito maior do que a estimativa para a idade do universo em femtossegundo (1fs corresponde a 10^{-15} segundos, ou seja, um milionésimo de um bilionésimo de segundo). Na realidade é muito maior do que o produto destas duas quantidades. Então, apesar dos vários desafios, a construção de um computador quântico, que explore todo esse poder, seria uma evolução marcante no mundo da computação.

Apesar da computação quântica tirar proveito de certas propriedades da mecânica quântica para permitir a realização de tarefas de forma mais eficiente do que seria possível na computação clássica, tal método utiliza um processo de superposição de bits para aumentar a capacidade de processamento de dados. Assim, ainda existem muitos desafios em termos práticos, teóricos e físicos para serem resolvidos.

É notável que em diversas áreas do conhecimento humano a simulação desempenha um importante papel. Simuladores são utilizados, por exemplo, como ferramenta de apoio ao ensino. Para o ensino de matérias relacionadas à computação, existem ferramentas para simulação de arquiteturas de computadores clássicos, autômatos finitos e máquinas de Turing (Andrade Barbosa, 2007).

Feynman (1982) observou que nenhum sistema clássico pode simular um sistema quântico de maneira eficiente, isso acontece, principalmente, devido à natureza paralela do mundo quântico (Feynman, 1985). A partir desta observação, Feynman propôs que apenas um sistema quântico pode simular eficientemente outro sistema quântico. Dessa forma a criação de ferramentas que ofereçam um desempenho mais rápido no processo de simulação de circuitos quânticos pode tornar o trabalho com alguns algoritmos quânticos menos exaustivo para o computador, em virtude de tais cálculos utilizam muitas operações com alto custo

computacional, e a programação com GPUs é mais apropriada para se realizar cálculos com matrizes, assim uma ferramenta que dê, a opção de o usuário simular na GPU algumas operações, pode ser de grande ajuda, também é importante notar que muitas das ferramentas existentes na literatura são muito difíceis de utilizar. Deste modo, uma ferramenta de fácil utilização e instalação, tendo uma curva de aprendizagem pequena e que permita automatizar os cálculos com circuitos quânticos, — provendo tanto a simulação na CPU, quanto na GPU — é de extrema importância.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo deste dissertação é promover o desenvolvimento de uma ferramenta para programação de circuitos quânticos que possibilita a execução dos cálculos na GPU.

1.2.2 Objetivos Específicos

Para alcançar o objetivo geral, têm-se os seguintes objetivos específicos:

1. Desenvolver uma gramática no padrão BNF;
2. Desenvolver um compilador de acordo com a gramática citada em (1);
3. Desenvolver o módulo de geração de código;
4. Desenvolver as funções para calcular o circuito na CPU;
5. Desenvolver as funções para calcular o circuito de forma paralela em JOCL na GPU;
6. Desenvolver um módulo que permita gerar o circuito programado, de forma gráfica;
7. Determinar um IDE de programação composto de funcionalidades para facilitar no processo de desenvolvimento de circuitos quânticos; e
8. Promover um experimento visando validar a ferramenta LinDCQ (APÊNDICE A – (LinDCQ)).

1.3 Descrição do Trabalho

Para evidenciar a concretização dos objetivos citados, nesta seção são resumidas as características dos seus desenvolvimentos. Para o desenvolvimento do compilador, primeiramente definiu-se a sua gramática, construída a partir do modelo BNF (*Backus-Naur-Form*). Para a edição da gramática da ferramenta aqui apresentada e da geração das classes utilizadas para compor o programa de interpretação (analisador léxico e sintático) utilizou-se o GALS (Gerador de Analisadores Léxicos Sintáticos) (GESSER, 2003).

A ferramenta possui um algoritmo para a geração de código (importante camada da estrutura de um compilador) e para o desenvolvimento do seu IDE utilizou-se a linguagem de programação Java, em virtude de ser uma linguagem independente da plataforma (DEITEL, 2005), garantindo a possibilidade de seu uso em diversos sistemas operacionais. As funções desenvolvidas para realizar os cálculos na GPU utilizam JOCL, uma biblioteca que permite trabalhar com Java e OpenCL. Para validar a usabilidade da ferramenta proposta, foi realizado um experimento por meio das técnicas objetivas e prospectivas por meio de questionário. Os dados produzidos pelo experimento foram submetidos a uma análise estatística.

1.4 Organização do Trabalho

Esta dissertação está organizada da seguinte forma:

- Capítulo 1 - aborda-se a motivação que inspirou a realização desta pesquisa, o objetivo geral e os objetivos específicos, a descrição sobre o desenvolvimento dos objetivos, organização desta dissertação e uma descrição sobre os capítulos;
- Capítulo 2 - apresentam-se alguns conceitos de teoria da computação, gramáticas e expressões regulares, bem como as aplicações e utilidades das mesmas;
- Capítulo 3 - apresenta-se uma explanação sobre os compiladores, um dos elementos essenciais para o desenvolvimento desta dissertação. Também são apresentadas as fases que compõem os compiladores, bem como a aplicação de cada uma delas. Ao final, são abordados alguns conceitos básicos com relação as linguagens de programação, apresentando critérios para avaliar uma linguagem de programação;
- Capítulo 4 - aborda-se a programação em GPGPU, discutindo um pouco a respeito da grande evolução que as placas de vídeo tiveram, com uma comparação entre estas e as CPU's. Apresentando, no final, alguns pacotes para programação em JAVA utilizando CUDA e OpenCL;
- Capítulo 5 - apresentam-se alguns conceitos da mecânica quântica e da computação quântica, com uma pequena explanação do surgimento desta. Támbém são apresentados os circuitos quânticos, juntamente com algumas ferramentas para simulação de tais circuitos;
- Capítulo 6 - discuti-se a arquitetura da ferramenta descrita nesta dissertação, suas funcionalidades e as tecnologias utilizadas para o seu desenvolvimento, discutindo-se a gramática utilizada, as funções utilizadas para a etapa de geração de código, as técnicas para calcular o circuito na CPU e na GPU e alguns exemplos de circuitos quânticos construídos na ferramenta;
- Capítulo 7 - explana a validação da usabilidade da ferramenta LinDCQ, discutindo-se as metodologias utilizadas para tal finalidade e os testes estatísticos realizados;
- Capítulo 8 - apresentam-se as conclusões desta dissertação, bem como as considerações finais, as contribuições e propostas para trabalhos futuros; e
- Ao final são apresentadas as referências bibliográficas, seguidas de apêndice e anexos que complementam a dissertação.

Capítulo 2

Teoria da Computação e Expressões Regulares

Neste capítulo apresentam-se conceitos da computação e que servem de base no projeto de um compilador: a Teoria da Computação com suas máquinas abstratas (autômatos) e as expressões regulares. Na Seção 2.1 defini-se o que vem a ser um programa e uma linguagem na computação. Na Seção 2.2 são apresentados alguns conceitos fundamentais da Teoria da Computação, como, por exemplo, alfabeto, símbolo, cadeia, subpalavra e algumas operações. Na Seção 2.3 é apresentado o conceito de gramática com destaque a BNF e autômatos. Também são apresentadas as expressões regulares. A Seção 2.4 resume o capítulo.

2.1 Programa e Linguagem

Segundo MENEZES (2005), um programa pode ser definido como sendo um "procedimento efetivo", que pode ser descrito usando qualquer formalismo com determinadas características, ou seja, qualquer formalismo que possa descrever todos os procedimentos possíveis a serem executados por um computador. Pode-se definir um "procedimento efetivo" como sendo um programa que é uma sequência finita de instruções que podem ser executadas mecanicamente em um tempo finito. Já uma linguagem é um dos conceitos fundamentais no estudo da teoria da computação, pois se trata de uma forma precisa de expressar problemas. Permitindo um desenvolvimento formal adequado ao estudo da computabilidade.

2.2 Conceitos Básicos

Um **alfabeto** é um conjunto finito de símbolos ou caracteres, e se o conjunto for infinito ou vazio, então não será um alfabeto. Alguns exemplos de alfabetos são ilustrados abaixo:

$$\text{alfabeto} = \{a, b, c\}$$
$$\text{alfabeto} = \{1, 2, 3\}$$
$$\text{alfabeto} = \{a, 1, b, 3\}$$

Como exemplo de conjunto de símbolos que não constituem alfabeto pode-se citar o conjunto dos números Naturais N = {0, 1, 2, 3,...} e vários outros conjuntos numéricos, como os inteiros

e os reais, assim como um conjunto vazio (MENEZES, 2005).

Símbolos (átomos ou letras) são elementos indivisíveis do alfabeto, um símbolo de um alfabeto é uma entidade básica, podendo representar números, letras ou desenhos, como por exemplo: a, b, 1, #, ∗.

Cadeias de palavras ou **sentenças** dizem respeito a qualquer concatenação finita de símbolos de um alfabeto, onde uma cadeia de símbolos finitos é usualmente denominada de palavra. Tem como características:

- ϵ - Denota a cadeia vazia ou uma palavra vazia.
- Σ - Representa um conjunto de símbolos, ou seja, um alfabeto.
- Σ^* - O conjunto de todas as palavras possíveis sobre Σ (o alfabeto).
- Σ^+ - se refere a todas as palavras possíveis sobre o alfabeto menos a cadeia vazia, por exemplo $\Sigma^+ = \Sigma^* - \{\epsilon\}$.

O comprimento ou tamanho de uma palavra pode ser representado por uma palavra entre duas barras verticais, como, por exemplo: $|aaba| = 4$ e $|654\#jL| = 6$. Existe um caso especial de sentença, de tamanho zero, chamada de sentença vazia, a qual é denotada por 'ϵ', muito utilizada na teoria de linguagens, bem como durante a análise e processamento destas (FREITAS, 2006).

A **concatenação de palavras** é uma operação binária, que é definida sobre uma determinada linguagem, a qual adiciona a cada par de palavras uma palavra formada pela justaposição da segunda, que pode não resultar em uma palavra pertencente à linguagem. Por exemplo, considerando as palavras z = aab e w = 18, é possível formar algumas concatenações possíveis para essas palavras: zw = aab18, wz = 18aab e wzw = 18aab18.

A **concatenação sucessiva de uma palavra** é definida pela concatenação de uma dada palavra com ela mesma, que pode ser representada na forma de um expoente. Supondo que p^n seja uma palavra, onde: p é a palavra e n como sendo o número de concatenações sucessivas que devem ser aplicadas a palavras, então para $p \neq \lambda$, tem-se $p^0 = \lambda$; para $p = \lambda$, tem-se $p^0 =$ indefinido, $p^1 = p, p^2 = pp, ..., p^n = pppp...p$.

Um **prefixo** é qualquer sequência de símbolos iniciais de uma palavra, enquanto que um **sufixo** é qualquer sequência de símbolos finais de uma palavra, e uma **subpalavra** é qualquer sequência de símbolos contígua de uma palavra (MENEZES, 2005). A Tabela 2.1 ilustra prefixos, sufixos e subpalavras.

O **reverso de uma cadeia** pode ser obtido pela ordem inversa de símbolos da cadeia, como, por exemplo, a palavra w = abcd, então seu reverso será w r = dcba.

Uma **linguagem** é um conjunto de palavras sobre um alfabeto, tem uma sintaxe bem definida de forma que dada uma sentença, seja sempre possível saber se, ela pertence ou não a linguagem, com uma semântica bem precisa, a exemplo de Java, C, HTML, Pascal, entre outras. As linguagens podem ser definidas, de maneira formal, através da enumeração de suas sentenças (mas isto apenas no caso de linguagens finitas), ou por meio de gramáticas, construções mais sofisticadas, onde é possível representar de maneira finita, linguagens

Alfabeto	Palavra	Prefixo	Sufixo	Subpalavras
		a, aa,	c, ac,	
		aab,	bac,	Todos os
{a, b, c}	aabbac	aabb,	bbac,	prefixos e sufixos são também
		aabba,	abbac,	subpalavras.
		aabbac	aabbac	

Tabela 2.1: *Prefixos, sufixos e subpalavras para um determinado alfabeto*

eventualmente infinitas. Segundo Sebesta (2002), as gramáticas também são a base para o reconhecimento de linguagens, isto é, analisar uma sentença (por meio de uma analise sintática) e dizer se ela pertence ou não a uma, determinada, linguagem.

2.3 Gramática

As gramáticas são usadas com o intuito de representar uma linguagem, também podemos dizer que as gramáticas são um dispositivo de geração de sentenças que geram uma linguagem (GESSER, 2003).

Uma gramática é uma quádrupla $G = (V, T, S, P)$ onde:

- V - é um conjunto finito de símbolos variáveis (ou não terminais);
- T - é um conjunto finito de símbolos terminais (ou alfabeto);
- S - elemento de V, denominado variável inicial;
- P - conjunto finito de pares, denominado regras de produção tal que o primeiro componente é uma cadeia de $(V \cup T)^+$ e o segundo componente é uma cadeia de $(V \cup T)^*$.

Uma regra de produção (α, β) é representada $\alpha \rightarrow \beta$, que definem as condições de geração das palavras da linguagem, e caso as sequências de regras apresentarem a forma $\alpha \rightarrow \beta_1$, $\alpha \rightarrow \beta_2$, $\alpha \rightarrow \beta_3$, ..., $\alpha \rightarrow \beta_n$, poderão ser reduzidas para $\alpha \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid ... \beta_n$.

Suponha como exemplo, a seguinte gramática: $G = (\{\alpha\}, \{0, 1\}, \alpha, \{(\alpha, 0\alpha1), (\alpha, \lambda)\})$. Então $V = \{\alpha\}, T = \{0, 1\}, S = \{\alpha\}$ e $P = \{(\alpha, 0\alpha1), (\alpha, \lambda)\}$, ou seja, $\alpha \rightarrow 0\alpha1, \alpha \rightarrow \lambda$ e, portanto $\alpha \rightarrow 0\alpha1 \mid \lambda$.

Para mostrar que 000111 faz parte da linguagem e está associada à gramática G, tem-se a seguinte sequência de derivação:

- Aplica a regra de produção $\alpha \rightarrow 0\alpha1$, se obtêm: $0\alpha1$,
- Aplica a regra de produção $\alpha \rightarrow 0\alpha1$, se obtêm: $00\alpha11$,
- Aplica a regra de produção $\alpha \rightarrow 0\alpha1$, se obtêm: $000\alpha111$,

- Aplica a regra de produção $\alpha \rightarrow \lambda$, se obtêm: 000111.

Seja $G = (V, T, S, P)$ uma gramática. A linguagem gerada pela gramática G denotada por $L(G)$ ou $GERA(G)$, é composta por todas as cadeias símbolos terminais deriváveis a partir do símbolo inicial S, ou seja: $L(G) = \{w \in T^* \mid S \Rightarrow^+ w\}$.

A critério de exemplificação seja $G = (\{S\}, \{a, b\}, S, P)$, onde $P = \{S \rightarrow aSb \mid \lambda\}$, então: $S \Rightarrow \lambda, S \Rightarrow aSb \Rightarrow ab, S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb, S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$. Logo $L(G) = \{\lambda, ab, aabb, aaabbb, ...\} = \{a^n b^n, n \geq 0\}$.

2.3.1 Gramática BNF

As linguagens livres de contexto compreendem uma abordagem mais ampla do que as linguagens regulares, tratando de forma apropriada as questões de balanceamentos entre parênteses e blocos de programas. Os seus algoritmos são simples e possuem uma boa eficiência; a gramática é um método formal de geração de linguagens utilizado para descrever a sintaxe das mesmas.

A BNF é uma notação natural para descrever a sintaxe, tornou-se e ainda continua a ser o método mais popular para descrever concisamente a sintaxe de uma linguagem de programação; é considerada uma metalinguagem, ou seja, uma linguagem utilizada para descrever outras linguagens (Sebesta, 2002).

Para descrever uma estrutura sintática na BNF utiliza-se uma palavra entre os sinais de "<" e ">", respectivamente, para fazer uma derivação é utilizado o sinal "=>", que também pode ser chamada de regra de produção. A Figura 2.1, traz um exemplo de uma gramática BNF, que aceita a instrução $A = B * (A + C)$:

```
<atribuição> = > <id> = <expr>
             = > A = <expr>
             = > A = <id> * <expr>
             = > A = B * <expr>
             = > A = B * ( <expr> )
             = > A = B * ( <id> + <expr> )
             = > A = B * (A + <expr> )
             = > A = B * (A + <id> )
             = > A = B * ( A + C )
```

Figura 2.1: *Exemplo de uma gramática BNF*

As gramáticas descrevem estruturas sintáticas hierárquicas das linguagens que elas definem, tais estruturas são chamadas de árvores de análise (*parse trees*), a árvore de análise da Figura 2.2 se refere à gramática da Figura 2.1:

2.3.2 Autômatos

As máquinas de estados finitos (MEF) são máquinas abstratas que capturam as partes essenciais de alguma máquina concreta, as máquinas concretas são aquelas capazes de realizar

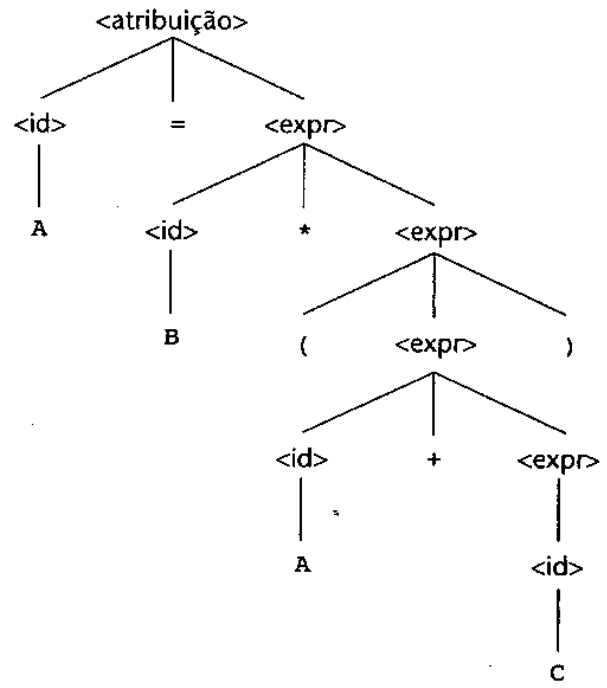


Figura 2.2: *Exemplo da arvore de análise de uma gramática BNF*

alguma tarefa. Existem basicamente dois tipos de MEF: transdutores (que são máquinas com entrada e saída) e reconhecedores de linguagem (que são máquinas com apenas duas saídas possíveis: aceita a entrada ou rejeita a entrada).

As linguagens reconhecidas por um MEF são denominadas linguagens regulares, uma das características das MEF é que sua memória é limitada e organizada em torno do conceito de "estados". As MEF do tipo reconhecedor de linguagem são conhecidas como autômatos finitos. Um autômato é um modelo abstrato de um computador composto por um arquivo de entrada, um arquivo de saída, memória e unidade de controle, pode-se dizer que um autômato finito é uma estrutura matemática constituída por três tipos de entidades: um conjunto finito de estados, um alfabeto e um conjunto finito de transições (MENEZES , 2005).

Para MENEZES (2005), sendo um autômato finito um formalismo operacional reconhecedor de linguagens eles podem ser classificados em:

- **Determinístico** - dependendo do estado e do símbolo lido o sistema só poderá assumir um estado; e
- **Não determinístico** - dependendo do estado e sem ler um símbolo o sistema pode assumir um novo ou um conjunto de estados.

Um autômato finito determinístico (AFD) pode ser visto como uma máquina constituída de três partes:

- **Fita** - um dispositivo de entrada;
- **Unidade de controle** - reflete o estado corrente da máquina; e

- **Programa, função programa ou função de transição** - uma função que comanda as leituras e define o estado da máquina.

O arquivo de entrada armazena uma cadeia sobre um alfabeto. O mecanismo de leitura lê a cadeia da esquerda para a direita, um símbolo por vez. O arquivo de saída armazena uma cadeia sobre um alfabeto produzido pelo autômato. A memória é um número finito de células que armazenam símbolos de um alfabeto (que podem ser distintos do alfabeto de entrada e saída). O autômato pode ler e alterar o conteúdo da memória. A unidade de controle pode estar em qualquer um dos estados internos do autômato, em um dado instante. A unidade de controle pode mudar de estado dependendo das funções de transições definidas.

O AFD é uma quíntupla ordenada: $M = (\Sigma, Q, \delta, q_0, F)$, onde:

- Σ - é um alfabeto de símbolos de entrada, ou simplesmente alfabeto de entrada.
- Q - é um conjunto de estados possíveis do autômato conjunto finito.
- δ - é a função de transição ou programa: que é uma função parcial como, por exemplo, se δ é definida para um estado "p" e um símbolo "a", resultando num estado "q", então: $\delta(p, a) = q$ é uma transição do autômato.
- q_0 - é um símbolo distinguido de Q, denominado estado inicial.
- F - é um subconjunto de Q, denominado conjunto de estados finais.

O autômato pode ser definido através de um diagrama esquemático, que de certa forma é um grafo, no qual:

- Estados são nodos;
- Transições são arestas ligadas aos nodos;
- Estados iniciais e finais são representados de forma distinta dos demais estados; e
- Transições paralelas são representadas por arestas duplas.

A Figura 2.3 apresenta um grafo, o qual demonstra a transição de estados de um autômato, este autômato se encontra primeiramente no estado "p", onde lendo o símbolo "a" ocorre uma mudança de estado, passando para o estado "q".

Existe outra forma alternativa para representarmos um AFD através de uma tabela de dupla entrada, como, por exemplo, se dada uma transição do tipo $\delta(p, a) = q$, tem-se a Tabela 2.2.

2.3.3 Expressões Regulares

Segundo MENEZES (2005), a expressão regular é uma maneira de descrever os conjuntos regulares. Usa-se a expressão regular em construção de compiladores, editores, sistemas operacionais, protocolos, etc. Trata-se de um formalismo denotacional, também considerado

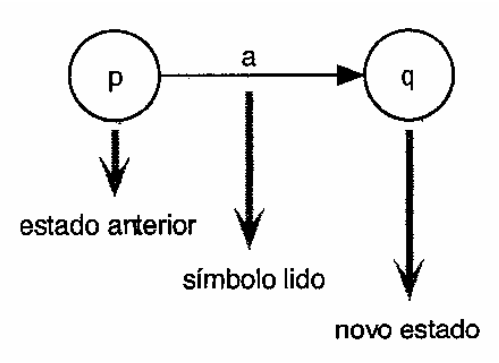


Figura 2.3: *Diagrama de transição de um AFD*

δ	a
p	q
q	

Tabela 2.2: *Função programa de um AFD na forma de tabela*

gerador, pois se pode inferir como construir ou gerar as palavras de uma linguagem. Uma expressão regular é definida a partir de conjuntos básicos e operações de concatenação e união. Toda linguagem regular pode ser descrita por uma expressão regular, assim como é definida a partir de conjuntos de linguagens básicas e que possuem uma álgebra definida.

2.3.4 Operadores de Expressão Regular

São considerados operadores de expressão regular: união, concatenação e operador de Kleene, onde:

- Denota-se a união das linguagens L e M como $L \cup M$, ou seja, é o conjunto de strings que estão em L ou M ou ambas. Também é denotado por $L + M$;
- Na concatenação de duas linguagens L e M, tem-se o conjunto de todas as strings que podem ser formados tomando qualquer string em L e concatenado com qualquer string em M. Denota-se por $L.M = LM = L(M).L(M)$; e
- Em lógica matemática e ciência da computação, fecho de Kleene ou operador de Kleene é uma operação unária em linguagens ou alfabetos, a aplicação do fecho de Kleene num conjunto V é escrito como V^*. É uma expressão no contexto em que foi introduzida por Stephen Kleene[1] para caracterizar certos tipo de autômatos.

A Tabela 2.3 ilustra algumas expressões regulares com as respectivas linguagens geradas:

Os operadores também possuem precedências, a saber:

- O operador de Estrela (fecho de Kleene) é o de procedência mais alta;

[1]Disponível em: http://www-history.mcs.st-and.ac.uk/Biographies/Kleene.html

ER	Linguagem representada aa
aa	Somente a cadeia aa
ba^*	Cadeias que iniciam com b, seguida por zero ou mais a
$(a+b)^*$	Todas as cadeias sobre $\{a, b\}$
$(a+b)^*aa(a+b)^*$	Todas as cadeias contendo aa como subcadeia
$a^*ba^*ba^*$	Todas as cadeias contendo exatamente dois b
$(a+b)^*(aa+bb)$	Todas as cadeias que terminam com aa ou bb
$(a+\lambda)(b+ba)^*$	Todas as cadeias que não possuem dois a consecutivos

Tabela 2.3: *Expressões regulares e as correspondentes linguagens geradas*

- Concatenação; e
- Uniões.

Se r é uma expressão regular, então $L(r)$, a linguagem sobre r, é uma linguagem regular. Por definição, uma linguagem é regular se, e somente se, é possível construir um autômato finito (determinístico ou não), que reconheça a linguagem. A Tabela 2.4 apresenta algumas expressões regulares e os respectivos autômatos finitos.

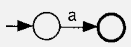
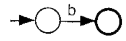
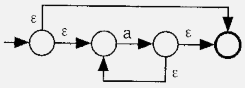
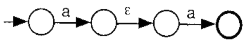
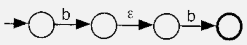
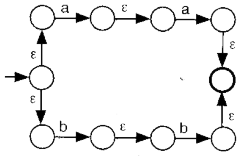
ER	Autômato finito correspondente
a	
b	
a^*	
aa	
bb	
$(aa+bb)$	

Tabela 2.4: *Expressões regulares (ER) e os autômatos finitos correspondentes*

2.4 Resumo do Capítulo

Este capítulo descreveu os principais conceitos da teoria da computação, tais como: alfabeto, símbolo e cadeia. Apresentou também a gramática que é um mecanismo gerador de linguagens. Foram abordados os autômatos, que nada mais são que um mecanismo reconhecedor

de linguagens, assim como as expressões regulares que são uma maneira de descrever os conjuntos regulares.

Capítulo 3

Compiladores e Linguagens

Neste capítulo explana-se sobre compiladores, elemento essencial ao desenvolvimento do trabalho. Na Seção 3.1 discuti-se o funcionamento de uma linguagem compilada. Na Seção 3.2 é abordado o funcionamento de uma linguagem puramente interpretada, enquanto que a Seção 3.3 aborda o funcionamento de uma linguagem de interpretação hibrida. Na Seção 3.4 introduzem-se compiladores, descrevendo arquitetura e o papel desempenhado por cada um de seus componentes. Na Seção 3.5 é apresentada uma breve explanação sobre Linguagens de programação. A Seção 3.6 resume o capítulo.

3.1 Compilação

Para o entendimento do processo de compilação é preciso saber que programas podem ser traduzidos para linguagem de máquina, o qual pode ser executado diretamente no computador. Esse método de implementação é chamado de compilação, cujo processo é ilustrado na Figura 3.1. A vantagem do método compilado é que o programa tem uma velocidade de execução muito rápida depois do processo de tradução ser concluído, um exemplo de linguagem compilada é a linguagem de programação C (Sebesta, 2002).

3.2 Interpretação Pura

Os programas podem ser interpretados por outros programas chamados interpretadores, essa técnica tem a vantagem de permitir uma depuração mais fácil, porque o processo de verificação de erros que ocorre em tempo de execução pode emitir mensagens de algumas unidades do código, como, por exemplo, o acesso a um elemento de uma matriz que possua índice fora do intervalo definido, resultando em uma mensagem contendo a linha do erro. A Figura 3.2 ilustra o fluxograma de uma interpretação pura.

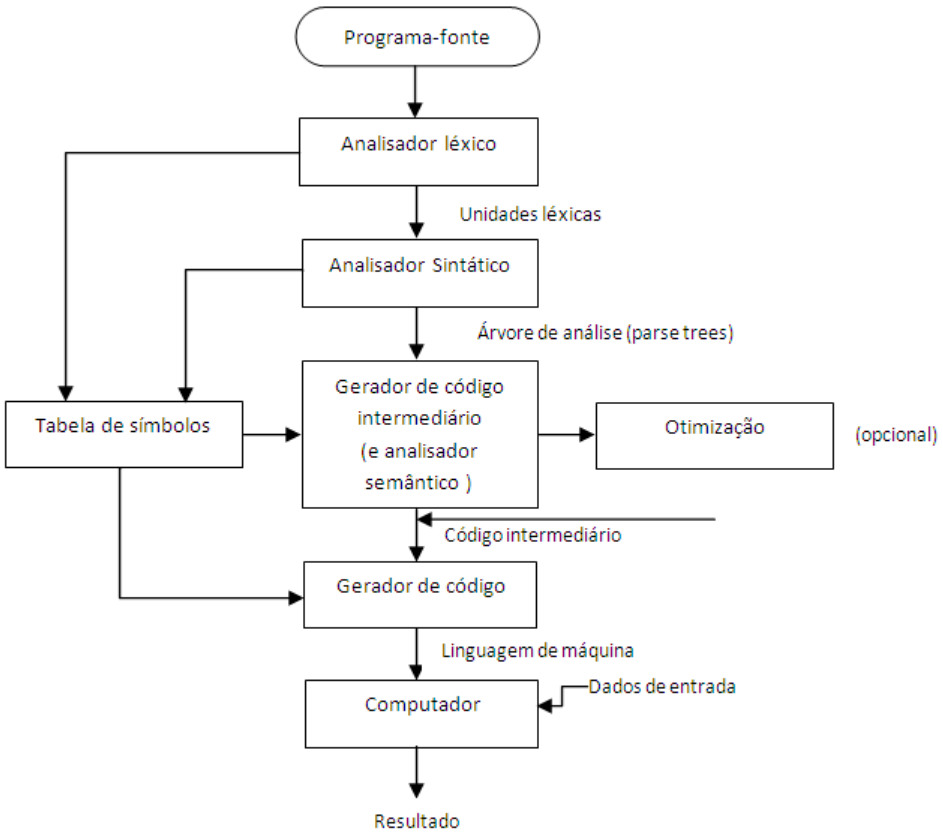


Figura 3.1: *O processo de compilação*

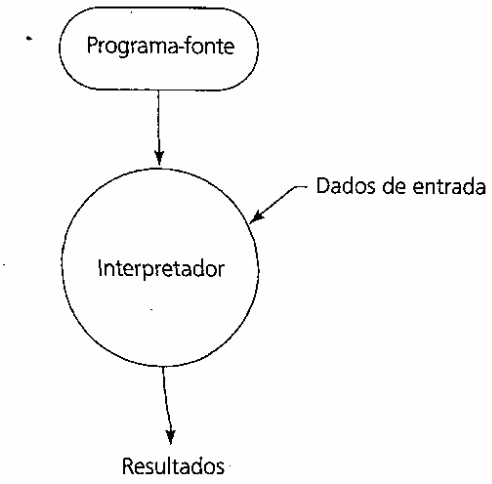


Figura 3.2: *Interpretação pura*

3.3 Interpretação Hibrida

Segundo Sebesta (2002), algumas linguagens apresentam características tanto das linguagens compiladas, quanto das interpretadas, são as chamadas híbridas. Elas traduzem programas escritos em linguagens de alto nível para linguagens intermediárias, projetada para produzir fácil interpretação.

Ao invés de traduzir código em linguagem intermediária para código de máquina, o código de máquina simplesmente é interpretado. A Figura 3.3 ilustra o passo a passo do sistema de

implementação híbrida:

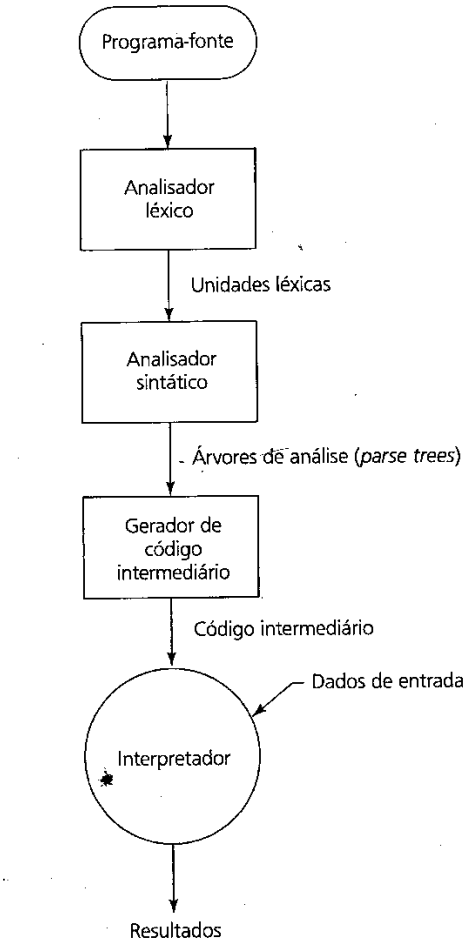


Figura 3.3: *Sistema de implementação hibrido*

3.4 Introdução aos Compiladores

Segundo Sethi *et al.* (2008), o propósito básico das linguagens de programação é gerar instruções para que o computador possa executar o processamento. Existe uma grande necessidade de facilitar a comunicação dos seres humanos e os computadores, visto que os computadores operam em nível atômico em linguagem de máquina, mais especificamente em base binária o que é extremamente complexo e muito sujeito a erros para os seres humanos, sendo preferível, nesse caso, uma linguagem mais próxima do seu modo de comunicação. A Figura 3.4 ilustra a estrutura geral dos compiladores, tendo suas principais etapas descritas nas próximas subseções.

3.4.1 Análise Léxica

O analisador léxico é a primeira fase que compõe o processo de compilação de um texto fonte (source), a qual cumpre uma série de tarefas, sendo a principal delas a de fragmentar

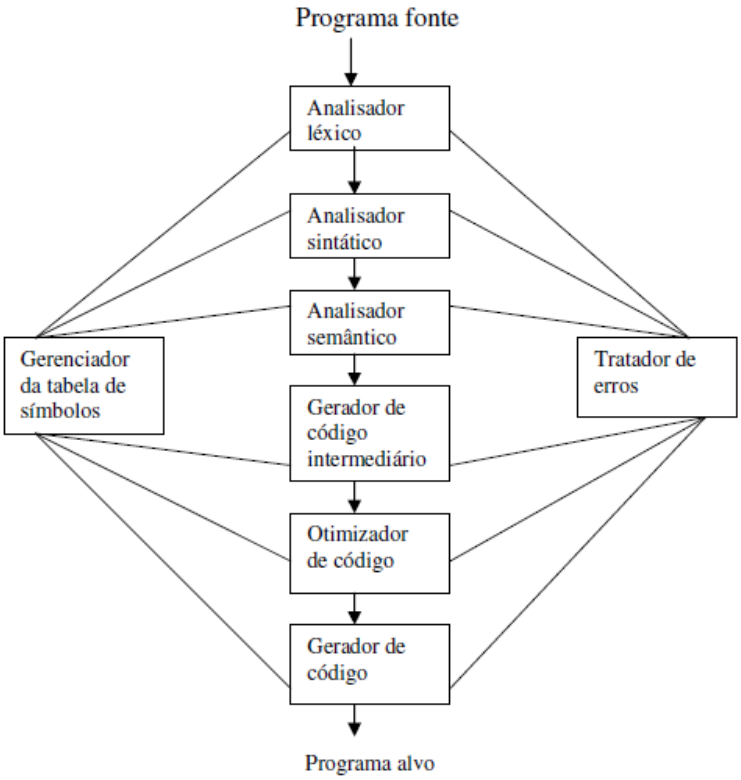


Figura 3.4: *Estrutura de um compilador*

o programa fonte de entrada em trechos elementares completos com identificação própria, ou seja, com um tipo de ID (Alencar Price e Toscani, 2000).

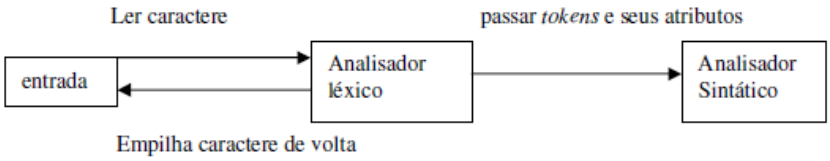


Figura 3.5: *Estrutura do analisador léxico*

A Figura 3.5 ilustra a comunicação envolvida no analisador léxico, o qual lê os caracteres de entrada e produz uma sequência de *tokens* que os *parser* utilizarão na análise sintática. O analisador léxico do compilador tem como função varrer o programa fonte da esquerda para a direita, agrupando os símbolos de cada item léxico e determinando a sua classe (Figura 3.6).

As relações das funções internas do analisador léxico são: extração e classificação de átomos; eliminação de delimitadores e comentários; identificação de palavras reservadas e; recuperação de erros.

Em geral existe um conjunto de cadeias de entrada para os quais o mesmo *token* é produzido como saída. Esse conjunto de cadeias é descrito por uma regra chamada de um padrão associado ao *token* de entrada, e o padrão tem a função de reconhecer cada cadeia do conjunto. Um *lexema* é um conjunto de caracteres no programa-fonte que é reconhecido pelo padrão de algum *token*, a exemplo de uma declaração e inicialização de uma variável em C: **int** $x = 10$, onde **int** é um *lexema* para o *token* do tipo numérico (Sethi *et al.*, 2008).

Os *tokens* são tratados como símbolos terminais nas gramáticas para a linguagem fonte,

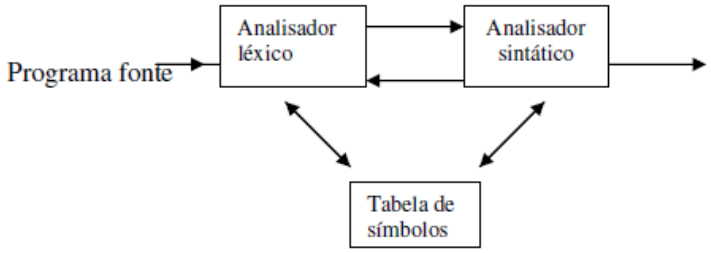


Figura 3.6: *Estrutura do analisador léxico*

usando nomes em negrito para representá-los. A Tabela 3.1 exemplifica alguns *tokens* com seus respectivos *lexemas*, explicitando uma descrição do padrão formal associado a cada *token*.

Token	Lexemas exemplo	Descrição formal do padrão
Const	Const	Const
IF	IF	IF
Relação	<, <=, <, >=, =, <>	<, <=, <, >=, =, <>, respectivamente
Id	PI, contador, d2	Letra seguida por letra e/ou dígito
Num	3.25, 14.8, 7, 8	Qualquer constante numérica
Literal	"conteúdo da memória"	Qualquer caractere entre aspa (exceto a própria aspa)

Tabela 3.1: *Tokens com os respectivos Lexemas*

3.4.2 Analisador Sintático

O módulo de análise sintática é o segundo passo de analise dos compiladores, e tem como função a promoção da análise da sequência com que os átomos componentes do código fonte se apresentam, a partir da qual se efetua a síntese da árvore sintática, com base na linguagem fonte.

Segundo Sethi *et al.* (2008), existem várias técnicas de analise sintáticas, como, por exemplo:

- LL - o primeiro "L" significa varredura da entrada da esquerda para a direta (*left-to-rigth*), e o segundo a produção de uma derivação mais a esquerda (*left linear*).
- LR - "L" significa varredura da entrada da esquerda para a direta (*left-to- rigth*), e o "R" significa construir uma derivação mais a direita (*rightmost derivation*).

A análise sintática cuida exclusivamente da forma das sentenças da linguagem, e procura, com base na gramática, levantar a estrutura das mesmas. A Figura 3.7 ilustra a comunicação no processo da análise sintática (Alencar Price e Toscani, 2000).

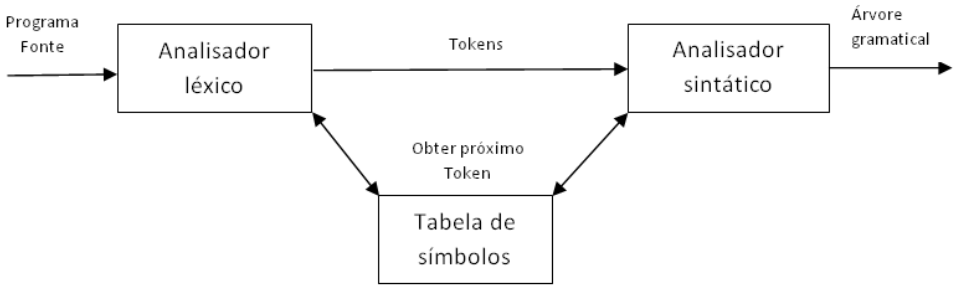


Figura 3.7: *Estrutura do analisador sintático*

3.4.3 Análise Semântica

Segundo Sethi *et al.* (2008), a análise semântica compõe a terceira etapa do processo de compilação e refere-se à tradução do programa fonte em programa objeto, tendo como principal objetivo a captação do significado das ações a serem tomadas no texto fonte. Existem duas notações para associar regras semânticas às produções: definições dirigidas pela síntese e esquemas de tradução. As definições dirigidas pela sintaxe são especificações de alto nível para as traduções. Escondem muitos detalhes de implementação e liberam o usuário de especificar exatamente a ordem na qual as traduções têm lugar. Os esquemas de tradução indicam a ordem a qual as regras semânticas são avaliadas e assim permitem que alguns detalhes de implementação sejam evidentes.

Conceitualmente, com os dois esquemas de definições dirigidas pela sintaxe e de tradução, é analisado sintaticamente o fluxo de *tokens* de entrada, é construída a árvore gramatical e em seguida, percorre-a da forma necessária, avaliando as regras semânticas a cada nó, processo este que está ilustrado na Figura 3.8. A avaliação das regras semânticas pode gerar códigos, salvar informações numa tabela de símbolos, emitir mensagens de erro ou realizar quaisquer outras das regras semânticas (Sethi *et al.*, 2008).

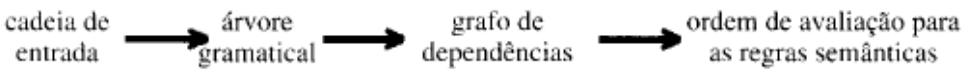


Figura 3.8: *Estrutura do analisador semântico*

A fase da análise semântica é de difícil formalização, exigindo notações complexas, e devido a isso a maioria das linguagens de programação adota especificações mais simplificadas, com bases informais, através de linguagens naturais.

A principal função da análise semântica é criar uma interpretação a partir do texto fonte, expressa em alguma notação adequada que é, geralmente, uma linguagem intermediária do compilador. Esta operação é realizada com base nas informações existentes nas tabelas construídas pelos outros analisadores, tabelas de palavras reservadas, mapas e saídas dos demais analisadores. Deve ser observada a fundamental importância de uma análise semântica bem realizada, visto que toda a síntese estará embasada na interpretação gerada por este analisador. Entre as ações semânticas são tipicamente encontradas a representação dos tipos de dados e manter informações sobre o escopo dos identificadores (GESSER, 2003).

3.4.4 Geração de Código

Segundo Sethi *et al.* (2008), a entrada para uma rotina de geração de código é um programa em linguagem intermediária, e a saída do gerador de código é o programa-objeto. Esta saída pode apresentar-se numa variedade de formas de código: linguagem de máquina absoluta, linguagem de máquina relocável, linguagem montadora (assembly), ou então outra linguagem qualquer de programação. A produção de uma linguagem de alto nível simplifica ainda mais a geração de códigos. Este tipo de implementação atua como pré-processador, e transfere os problemas de geração do código de máquina para outro compilador.

As técnicas de otimização de código são geralmente aplicadas depois da análise sintática, usualmente antes e durante a geração de código. Essas técnicas consistem em detectar padrões no programa e substituí-los por construções equivalentes, porém mais eficientes. Estes padrões podem ser locais ou globais, e a estratégia de substituição pode ser dependente ou independente da máquina (Alencar Price e Toscani, 2000). Segundo Sethi *et al.* (2008), o programa-fonte como entrada para um compilador é uma especificação de uma ação. O programa-objeto, a saída do compilador, é considerado como sendo outra especificação da mesma ação. Para cada programa-fonte, existem infinitos programas objeto que implementam a mesma ação, ou seja, produzem a mesma saída para uma mesma entrada.

3.5 Linguagens

As linguagens de programação são estudadas essencialmente pelos cientistas da computação, existe uma vasta quantidade de profissionais que usam as linguagens de programação como instrumento de trabalho. Muitos dos usuários das linguagens de programação atuais têm o conhecimento de várias linguagens, ou ainda recursos relacionados ao ambiente da qualidade de uma linguagem de programação, desde a forma como representa as informações do "mundo real" como às ferramentas disponibilizadas para facilitar o tratamento dessas informações. Segundo Sebesta (2002), no intuito de simplificar o desenvolvimento, para o bom uso de uma linguagem, o programador deve conhecer as características da linguagem, bem como deve conhecer também as limitações da mesma.

Hoje em dia os computadores são usados para uma infinidade de áreas diferentes, que vão desde o controle de uma usina elétrica ao armazenamento de registro de talões de cheques pessoais (Sebesta, 2002). Em virtude dessa grande diversidade de finalidades, várias linguagens de programação com metas diferentes têm sido desenvolvidas cada uma com uma finalidade especifica, cada uma se enquadrando melhor em um projeto.

De forma geral, as linguagens de programação devem atender aos requisitos que lhe são atribuídos de forma bastante eficiente, reduzindo ao máximo a complexidade e a incidência de erros, e a linguagem deve ser representada de uma forma única, e seus elementos devem ter um significado único (Sethi *et al.*, 2008). Desse modo deve-se garantir que a linguagem seja definida tanto sintaticamente (como é escrito cada um dos símbolos de uma linguagem) como semanticamente (o que significa cada um dos símbolos de uma linguagem).

Segundo Sebesta (2002), a sintaxe tem como objetivo determinar a forma de manipular um programa em uma linguagem, sua construção tem o intuito de facilitar essa manipulação levando em consideração critérios como: simplicidade, expressividade e legibilidade

(discussão apresentada na Seção 3.5.2).

A semântica determina de certa forma a interpretação pretendida, ou seja, o que se deseja para cada um dos elementos sintáticos, para as linguagens de programação de forma geral a semântica é caracterizada sob três aspectos segundo Sethi *et al.* (2008):

- Semântica axiomática - descrita através de um conjunto de axiomas equacionais que relacionam diferentes expressões sintáticas na linguagem, determinando o significado de algumas expressões através de "sinônimos" como os dicionários fazem com alguns termos na língua português;
- Semântica operacional - é descrita pelas operações realizadas pelas expressões e seus resultados. A semântica operacional caracteriza o significado de cada expressão em um programa pelo que ela faz propriamente dita. O significado do programa é o comportamento deste quando executado em uma máquina; e
- Semântica denotacional - descrita pelo conjunto de dados associados a cada expressão, considerando o programa como uma máquina que transforma dados, então o significado de cada expressão pode ser baseado nos dados que ela transforma. O significado de um programa é análogo ao de uma função matemática que mapeia dados de entrada para suas respectivas saídas.

3.5.1 Aplicações de Linguagens de Programação

As linguagens de programação podem ser utilizadas para fins específicos, e sua escolha dependerá diretamente da aplicação a ser desenvolvida. Pode ser utilizado para fins científicos o que desencadeou o surgimento dos primeiros computadores na década de 40, e as aplicações cientificas necessitavam de estruturas de dados simples, mas exigiam uma grande capacidade de computação aritmética, uma vez que, trabalhavam com números grandes, e envolviam, por exemplo, cálculo de matrizes, tabelas de senos, cossenos e logaritmos. Para esses cálculos os programadores usavam tipicamente a linguagem assembly, mas a primeira linguagem para fins científicos foi a FORTRAN (Sebesta, 2002).

A disseminação de computadores com finalidades comerciais começou a emergir na década de 50, onde as linguagens de baixo nível como assembly e linguagem de máquina não se mostraram eficientes para esses propósitos, assim foram surgindo às linguagens de alto nível, sendo COBOL (ANSI, 1985) a primeira linguagem de alto nível a surgir.

A inteligência artificial é uma área muito abrangente das aplicações de computadores, que tem como uma das principais características o uso de computações simbólicas ao invés da numérica como de costume, ou seja, usar símbolos, ao invés de números. A primeira linguagem de programação desenvolvida para aplicações de inteligência artificial foi a funcional LISP (McCarthy, 1960).

3.5.2 Critérios para Avaliação de Linguagens de Programação

Para Sebesta (2002), existem vários critérios que definem uma boa linguagem de programação, de certa forma definir tais critérios geram controvérsias, tendo em vista que o que

pode ser bom para um propósito pode não ser bom para outro, ou ser desnecessário para os demais, assim só algumas poucas características são realmente necessárias na maioria das linguagens de programação, algumas dessas características são apresentadas na Tabela 3.2.

Características	Legibilidade	Capacidade de escrita	Confiabilidade
Simplicidade	*	*	*
Estrutura de Controle	*	*	*
Tipos de dados e estruturas	*	*	*
Projeto de sintaxe	*	*	*
Suporte a abstrações		*	*
Expressividade		*	*
Verificação de tipo			*
Manipulação de Exceções			*
Apelido restrito			*

Tabela 3.2: *Critérios de Avaliação da linguagem e as características que afetam*

Dentre as principais características segundo Sebesta (2002) destacam-se:

- **Legibilidade** - uma linguagem de programação para ser bem aceita pelos usuários deve ser de fácil escrita e de fácil manutenção, deve possuir estruturas lógicas fáceis de serem entendidas sem grandes esforços por quem se destina a ler o código. Quanto mais fácil é escrever um programa, maior é a probabilidade de está correto, linguagens que não utilizam algoritmos que não podem ser escritos em linguagem natural são mais difíceis de serem entendidos aumentando a probabilidade de erros;

- **Simplicidade global** - uma linguagem com um grande número de componentes é bem mais difícil de ser entendida que outra com poucos componentes. Na maioria das linguagens os programadores aprendem apenas uma parte dos recursos oferecidos pelas linguagens e deixam o restante dos recursos de lado. Outro problema a multiplicidade de recursos, como, por exemplo, em C um programador pode incrementar uma variável de quatro formas diferentes: $cont = cont + 1; cont += 1; cont++; ++cont$;

- **Instruções de controle** - Com o passar do tempo o uso da instrução "goto" caio em desuso, pois estruturas desse tipo prejudicam muito a legibilidade do código. O desuso desse tipo de instrução se deve também ao aperfeiçoamento de instruções como for e $while$;

- **Tipos de dados e estruturas** - uma linguagem que apresente tipos de dados e outras estruturas de dados pode facilitar muito sua legibilidade, a exemplo do retorno de uma função que poderia ser $return$ 1, mas se a linguagem tiver o tipo booleano ficaria mais nítido se o retorno fosse definido como $return\ true$;

- **Considerações sobre sintaxe** - muitas considerações sobre a sintaxe devem ser levadas em conta, como, por exemplo, restringir o tamanho dos identificadores tornando-os muito pequenos, pois palavras muito pequenas podem não ser suficientes para representar algum sentido. Também é de extrema importância projetar instruções para que a aparência indique sua finalidade, não requerendo esforços de interpretação do programador;

- **Capacidade de escrita** - tem como finalidade explicitar o quanto facilmente uma linguagem pode ser usada para criar programas para um determinado tipo de problema, por exemplo, algumas linguagens são melhores para programar sistemas operacionais como a linguagem C, já outras são mais ideais para sistemas comerciais como Visual BASIC;

- **Simplicidade e ortogonalidade** - caso uma linguagem possua um grande número de maneiras diferentes de implementar determinadas construções, poderá resultar em construções inadequadas na solução de um determinado problema e deixar outros recursos mais eficientes de lado;

- **Suporte a abstração** - o conceito de abstração em linguagens de programação é basicamente a capacidade de criar estruturas complexas e depois utilizá-las desprezando muitos de seus detalhes;

- **Expressividade** - a expressividade está relacionada com a possibilidade de representar informações de forma simplificada, otimizando o desenvolvimento de programas computacionais, a exemplo da linguagem C que existe uma forma simples e menor de incrementar valores: $cont++$, ao invés de $cont = cont + 1$;

- **Confiabilidade** - um programa pode ser definido como confiável se ele se comporta de acordo com as especificações as quais foi projetado;

- **Verificação de tipos** - testa o programa em busca de erros de tipos pelo compilador ou em tempo de execução;

- **Manipulação de exceção** - uma linguagem que manipula exceções oferece a capacidade de um programa tratar erros em tempo de execução, ou seja, encontrando esses erros e oferecendo medidas para corrigi-los, dando continuidade ao programa;

- **Apelidos** - devem ser usados com cuidado, pois é um recurso poderoso nas linguagens de programação e pode gerar erros graves se manipulados de forma errada. Permitem que um programa tenha dois ou mais métodos ou nomes diferentes que fazem referência à mesma célula na memória, por exemplo; e

- O custo da criação de uma linguagem de programação envolve várias variáveis a ser consideradas, como, por exemplo, o custo de pessoas envolvidas no desenvolvimento, o custo das ferramentas que serão necessárias para tal propósito, assim como do treinamento do pessoal que vai utilizá-la.

3.6 Resumo do Capítulo

Nesse capítulo apresentou-se o fato de que as linguagens de programação podem ser compiladas, interpretadas e hibridas, e expôs a arquitetura dos compiladores, abordando todas as suas fases: a análise léxica, análise sintática, análise semântica, geração de código intermediário, otimizador de código e geração de código objeto. Também foram expostos os principais critérios para a avaliação de uma linguagem de programação, como o custo de desenvolvimento de uma linguagem.

Capítulo 4

Programação em GPGPU

Neste capítulo apresentam-se alguns conceitos básicos da programação para GPGPU que servem de base para a compreensão deste trabalho. Na Seção 4.1 defini-se brevemente o que é a programação para placa de vídeo e quando a mesma é mais apropriada. Na Seção 4.2 são apresentadas algumas definições sobre a programação em CUDA. A Seção 4.3 explana sobre a programação em JCUDA, basicamente alguns pacotes para programação em CUDA utilizando JAVA. Na Seção 4.4 explana-se sobre OpenCL, um padrão aberto para o uso de GPUs, explanando, também, sobre JOCL, basicamente um modo de trabalhar de forma interoperável entre OpenCL e Java. E na Seção 4.5 é apresentado um resumo do capítulo.

4.1 Programação para a Placa de Vídeo

A Figura 4.1 faz uma comparação entre a capacidade de processamento das GPUs da Nvidia e dos processadores da Intel, com relação às operações de ponto flutuante. Pode-se observar que a partir de novembro de 2006, as GPUs da Nvidia estabeleceram uma larga vantagem no poder computacional em comparação com as CPUs da Intel (VASCONCELLOS, 2009).

Com o advento do crescimento da complexidade do processamento gráfico, mais especificamente nas aplicações de jogos 3D, as placas de vídeo passaram por um contundente progresso tecnológico. Hoje em dia, as placas de vídeo são pesadamente paralelas. Dispondo de vários núcleos dirigidos por largura de banda de memória, é importante salientar que esta largura de banda é muito alta. Desse modo, as GPUs de hoje oferecem recursos incríveis para processamento gráficos e não-gráficos.

A principal razão, a ser observada, por trás de tal evolução é que a GPU é especializado e construída, principalmente, para uso intensivo de computação, ou seja, computação altamente paralela — exatamente o que é a renderização gráfica sobre o processamento — e, portanto, é projetada de tal forma que mais transistores são dedicados ao processamento de dados, em vez de cache de dados e controle de fluxo, pode-se observar tal característica no esquema ilustrado pela Figura 4.2. O lado esquerdo apresenta a CPU e o direito a GPU, como é possível notar, as CPU's têm uma preocupação maior com a memória *cache* e menos preocupação com a ALU (*Arithmetic Logic Unit*); o contrário ocorre com as GPU's, ou

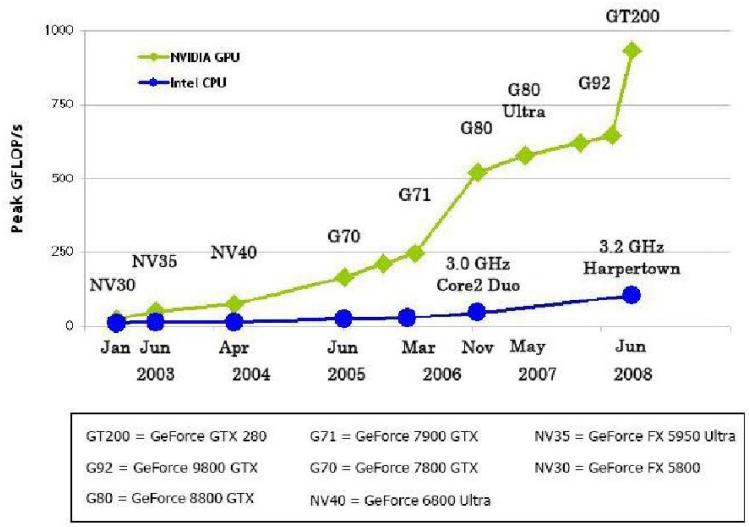


Figura 4.1: *Comparação das GPUs e CPUs (NVIDIA, 2009)*

seja, mais preocupação com a ALU e menos preocupação com a memória *cache*.

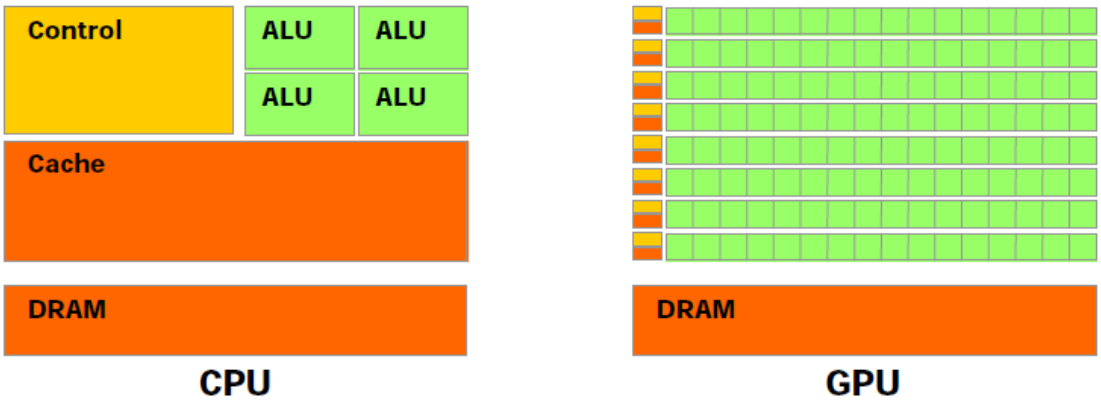


Figura 4.2: *A GPU dedicada com mais transistores para processamento de dados (NVIDIA, 2008)*

Mais especificamente, a GPU é bem adaptada para resolver os problemas que podem ser expressos, como cálculos paralelos de dados — o mesmo programa executa em diversos tipos de dados em paralelo, com alta intensidade aritmética. No que concerne as operações aritméticas e operações de memória, o mesmo programa é executado para cada elemento de dados, existindo uma necessidade menor para controles sofisticados de fluxo; tal fato ocorre porque, como salientado, o programa é executado em muitos elementos de dados e tem alta intensidade aritmética, a latência de acesso à memória pode ser escondida com cálculos, em vez de grandes caches de dados (NVIDIA, 2008).

Outro importante aspecto — levando em consideração a analise do poder de processamento de um hardware e a largura de banda da memória (memory bandwidth) — é o fato de, as GPUs da Nvidia também ultrapassaram as CPUs da Intel, neste quesito, conforme pode ser visto na Figura 4.3.

Tendo em vista esse grande potencial, a NVidea tornou publico em 15 fevereiro de 2007 uma nova plataforma de software chamada CUDA, com ela é possível explorar o potencial

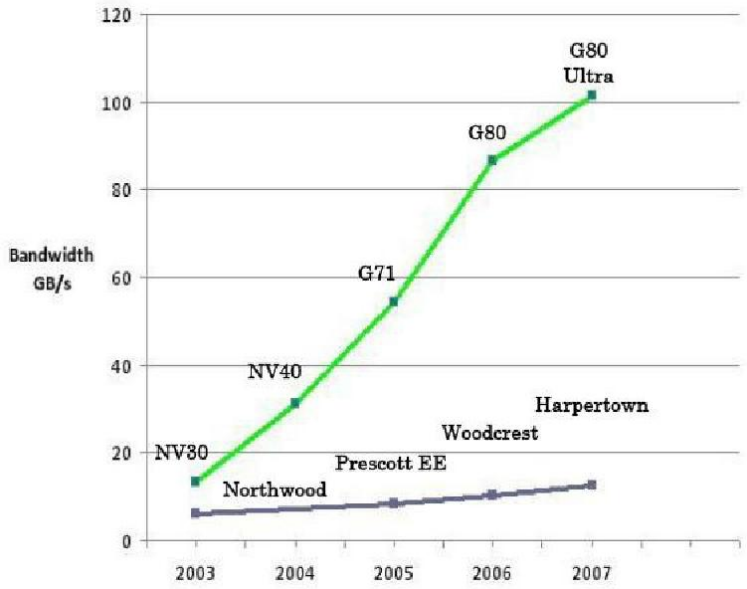


Figura 4.3: *Evolução da BANDWIDTH das GPUs e CPUs (NVIDIA, 2009)*

computacional das GPUs para computação geral (NVIDIA, 2008).

A arquitetura CUDA permite que cada unidade lógica aritmética (ULA) sobre o chip possa ser empacotada por um programa com a intenção de realizar cálculos de propósitos gerais. A NVIDIA destina esta nova família de processadores gráficos para ser usado para computação de propósito geral, estas ULAs foram construídas para cumprir com os requisitos da IEEE[1] para precisão simples de aritmética de ponto flutuante e foram projetados para usar uma instrução conjunta, adaptadas para computação geral e não especificamente para gráficos. Além disso, foram autorizadas as unidades de execução na GPU, que realizam leitura arbitrária e acesso à memória, bem como o acesso a uma cache gerenciada pelo software conhecido e escrever como memória compartilhada. Todas essas características da arquitetura CUDA foram adicionadas a fim de criar uma GPU que se destacasse em computação, além de executar bem as tarefas gráficas tradicionais (J. e E, 2010).

4.2 Programação em CUDA

Apesar de, no passado recente, a computação paralela ter sido concebida como uma área "exótico" e, normalmente, ser abordada como uma especialidade, dentro do campo da ciência da computação. Segundo J. e E (2010), essa percepção mudou profundamente de forma, nos últimos anos. O mundo da computação mudou a tal ponto que, longe de ser um exercício esotérico, quase todo programador aspirante precisa de treinamento em programação paralela para ser totalmente eficaz em ciência da computação. Aprendendo CUDA C este, estará bem posicionada para escrever aplicações de alto desempenho para plataformas heterogêneas que contêm unidades centrais de processamento gráfico.

Foi em novembro de 2006, que a NVIDIA revelou o primeiro DirectX 10 GPU da indústria, a GeForce 8800 GTX. A GeForce 8800 GTX, também foi a primeira GPU a ser construído com arquitetura CUDA da NVIDIA. Esta arquitetura incluiu vários novos com-

[1]Disponível em: http://standards.ieee.org/findstds/standard/754-2008.html

ponentes destinados estritamente para computação na GPU e destinados, também, a aliviar muitas das limitações que impediram que os processadores gráficos anteriores fossem, explicitamente, úteis para computação de propósito geral (J. e E, 2010). Algumas informações extras podem ser obtidas facilmente no Site[1].

4.2.1 Exemplo de aplicação em CUDA

O Código 4.1, a seguir, apresenta um programa simples, desenvolvido em CUDA, a estrutura é de um programa em C padrão, com um conjunto de bibliotecas complementares para trabalhar com a GPU. O programa, apenas, inicializa dois vetores a e b para depois armazenar o resultado da soma, dos respectivos indexes destes, no vetor c.

```
1  #include "../common/book.h"
2
3  #define N   20
4
5  __global__ void add( int *a, int *b, int *c ) {
6      int tid = blockIdx.x; //numero de thread para o processamento
7      if (tid < N)
8          c[tid] = a[tid] + b[tid];
9  }
10
11 int main( void ) {
12     int a[N], b[N], c[N];
13     int *dev_a, *dev_b, *dev_c;
14
15     // aloca a memoria na GPU
16     HANDLE_ERROR( cudaMalloc( (void**)&dev_a, N * sizeof(int) ) );
17     HANDLE_ERROR( cudaMalloc( (void**)&dev_b, N * sizeof(int) ) );
18     HANDLE_ERROR( cudaMalloc( (void**)&dev_c, N * sizeof(int) ) );
19
20     // preenche as matrizes 'a' e 'b' na CPU
21     for (int i=0; i<N; i++) {
22         a[i] = i+i;
23         b[i] = i * i;
24     }
25
26     // copia as matrizes 'a' e 'b' para a GPU
27     HANDLE_ERROR( cudaMemcpy( dev_a, a, N * sizeof(int),
          cudaMemcpyHostToDevice ) );
28     HANDLE_ERROR( cudaMemcpy( dev_b, b, N * sizeof(int),
          cudaMemcpyHostToDevice ) );
29
30     add<<<N,1>>>( dev_a, dev_b, dev_c );
31
32     // copia a matriz 'c' de volta da GPU para a CPU
33     HANDLE_ERROR( cudaMemcpy( c, dev_c, N * sizeof(int),
          cudaMemcpyDeviceToHost ) );
34
35     // exibir os resultados
36     for (int i=0; i<N; i++) {
```

[1]Disponível em: http://www.nvidia.com.br/object/cuda_home_new_br.html

```
37          printf( "%d + %d = %d\n", a[i], b[i], c[i] );
38      }
39
40      // libera a memoria alocada na GPU
41      HANDLE_ERROR( cudaFree( dev_a ) );
42      HANDLE_ERROR( cudaFree( dev_b ) );
43      HANDLE_ERROR( cudaFree( dev_c ) );
44
45      return 0;
46  }
```

Código 4.1: *Exemplo de programa desenvolvido em CUDA*

A linha 1 importa uma biblioteca para o tratamento de erros no código fonte, na linha 5 é declarada uma função que soma dois números, sendo esta a função executada na GPU (quando uma função for executada na GPU põe-se a palavra reservada *__global__* no início da declaração), este mecanismo alerta o compilador que a função deve ser compilado para rodar em um *device* (GPU), em vez de ser executada no *host* (CPU). A função recebe três ponteiros e coloca a soma, do conteúdo, dos dois primeiros no terceiro. A linha 6 contém o valor dos índices para qualquer bloco do código que está sendo executado no *device*. Na linha 12 declarou-se três vetores que serão utilizados no cálculo, enquanto que na linha 13 são declarados os respectivos ponteiros para esses vetores. Da linha 16 até a 18, foi reservado o espaço de memória no *device* que será utilizado durante a realização das operações, a função *HANDLE_ERROR* trata possíveis erros que possam surgir, já a função *cudaMalloc* aloca o espaço na memória do *device*.

Nas linhas 27 e 28 as matrizes *a* e *b* são copiadas do *host* para o *device* e a função *add<<<N,1>>>* executa a operação, propriamente dita, de somar os vetores no *device*, o primeiro parâmetro representa o número de *blocks*, enquanto o segundo representa o de *threads* que são lançados para realizar os cálculos. Após realizar os cálculos, a linha 33 copia o vetor que está no *device* para o *host* e depois imprime seus valores, como pode ser visto entre as linhas 36 e 38. Por fim, as linhas 41, 42 e 43 liberam a memória que foi alocada no *device*.

4.3 Programação em JCUDA

JCuda é uma API que oferecida a capacidade de manipular códigos em CUDA, a partir de funções escritas em Java. Existem vários pacotes com funções prontas, que serão discutidos nesta Seção 4.3.

- **JCublas** é uma biblioteca que torna possível a implementação dos pacotes básicos de Álgebra Linear com subprogramas em aplicações Java, suportando *single precision* e *double precision*, no entanto, é importante ressaltar que nem todos os dispositivos capazes de executar CUDA suportam os cálculos de *double precision*.
- **JCufft** é uma implementação da Transformada Rápida de Fourier em aplicações Java. Pode-se tanto procurar a documentação da API JCufft neste Site[1], ou baixar a docu-

[1]Disponível em: http://www.jcuda.org/jcuda/jcufft/doc/index.html

mentação JCufft em um arquivo ZIP[2].

- **JCurand** é uma biblioteca que torna possível a utilização de um gerador de números aleatórios, para ser utilizado em aplicações Java. Pode-se obter a última versão do JCurand e o código fonte no Site[3]. JCurand permite aos programadores criarem números estruturados de forma pseudo-aleatórios e quase aleatórios, com grande eficiência e de alta qualidade, a partir de programas Java — para os casos em que *java.util.Random* não garante uma grande precisão na aleatoriedade dos números gerados.

- **JCusparse** é uma biblioteca que torna possível calcular matrizes esparsas, em aplicações Java. Pode-se obter a última versão do JCusparse e o código fonte no Site[4].

- **JCudpp** permite que as aplicações em Java usem uma biblioteca para operações paralela de dados primitivos em CUDA, contendo métodos para matrizes esparsas; manipulação de vetores e multiplicação; varreduras paralelas; e classificação, bem como métodos para manter uma tabela de Hash em um dispositivo. Esta biblioteca pode ser compilado a partir do código-fonte que está disponível na home page Cudpp[5].

- **JNpp** oferece os métodos de processamento de imagem e de processamento de sinal. Algumas funções (como cálculos multi-nível de histograma) ainda não são totalmente suportados. É importante notar que a versão atual do JNpp, é uma versão inicial beta, e sua API ainda pode mudar. Pode-se obter a última versão do JNpp e o código fonte no Site[6].

Como em todas as bibliotecas JCuda, os métodos seguem as mesmas convenções de nomenclatura dos métodos nativos, e só formam uma camada fina em torno da API original do C.

4.3.1 Exemplo de aplicação em JCUDA

O Código 4.2 a seguir exemplifica uma aplicação simples em JCuda, que multiplica duas matrizes.

```
1  package consulta;
2
3  import jcuda.*;
4  import jcuda.jcublas.JCublas;
5
6  public class ProdutoMatrixJCublas {
7    // Implementa multiplicacao de matrizes usando JCublas
8    private static void sgemmJCublas(int n, float alpha, float A[], float
       B[],
9        float beta, float C[]) {
10     int nn = n * n;
11
12     // Inicializa JCublas
```

[2]Disponível em: http://www.jcuda.org/downloads/downloads.html#JCufft
[3]Disponível em: http://www.jcuda.org/downloads/downloads.html#JCurand
[4]Disponível em: http://www.jcuda.org/downloads/downloads.html#JCusparse
[5]Disponível em: http://code.google.com/p/cudpp/
[6]Disponível em: http://www.jcuda.org/downloads/downloads.html#JNppDownload

```
13      JCublas.cublasInit();
14
15      // Aloca memoria na GPU
16      Pointer d_A = new Pointer();
17      Pointer d_B = new Pointer();
18      Pointer d_C = new Pointer();
19      JCublas.cublasAlloc(nn, Sizeof.FLOAT, d_A);
20      JCublas.cublasAlloc(nn, Sizeof.FLOAT, d_B);
21      JCublas.cublasAlloc(nn, Sizeof.FLOAT, d_C);
22
23      // Copia resultado da CPU para a GPU
24      JCublas.cublasSetVector(nn, Sizeof.FLOAT, Pointer.to(A), 1, d_A, 1);
25      JCublas.cublasSetVector(nn, Sizeof.FLOAT, Pointer.to(B), 1, d_B, 1);
26      JCublas.cublasSetVector(nn, Sizeof.FLOAT, Pointer.to(C), 1, d_C, 1);
27
28      // Executa sgemm
29      JCublas.cublasSgemm('n', 'n', n, n, n, 1, d_A, n, d_B, n, 0, d_C, n)
            ;
30
31      // Copia resultado da GPU para a CPU
32      JCublas.cublasGetVector(nn, Sizeof.FLOAT, d_C, 1, Pointer.to(C), 1);
33
34      // Imprime valores com JCublas
35      // JCublas.printMatrix(n, d_A, n);
36      // JCublas.printMatrix(n, d_B, n);
37      JCublas.printMatrix(n, d_C, n);
38
39      // Limpa vetores
40      JCublas.cublasFree(d_A);
41      JCublas.cublasFree(d_B);
42      JCublas.cublasFree(d_C);
43
44      JCublas.cublasShutdown();
45    }
46
47    public static void main(String args[]) {
48      int n = 100;
49
50      float alpha = 1.0f;
51      float beta = 1.0f;
52      int nn = n * n;
53
54      float h_A[] = new float[nn];
55      float h_B[] = new float[nn];
56      float h_C[] = new float[nn];
57      for (int i = 0; i < nn; i++) {
58        h_A[i] = i;
59        h_B[i] = i * i;
60        h_C[i] = i + i;
61      }
62
63      System.out.println("Performance java com JCublas...");
64      sgemmJCublas(n, alpha, h_A, h_B, beta, h_C);
65    }
66  }
```

Código 4.2: *Exemplo de programa desenvolvido em JCUDA*

As linhas 3 e 4 importam as bibliotecas JCuda que serão utilizadas. O método *sgemmJCublas* linha 6, prepara os dados para serem executados na GPU. A linha 13 inicializa JCublas e da linha 16 até a 21, aloca-se a memória que será utilizada no *device*. Após a memória ter sido alocada os valores que estão no *host* são copiados para o *device*, tal processo pode ser visto nas linhas 24, 25 e 26.

A linha 29 executa a função no *device*, propriamente dita, depois de receber os respectivos parâmetros e após terminar o processamento no *device*, os dados são trazidos de volta para para o *host*, como deseja-se mostrar, apenas, o resultado do cálculo da matriz, apenas o valor da matriz C foi copiado de volta para o *host* como pode-se ver na linha 32. Para imprimir o valor da matriz C foi utilizado o método *printMatrix* que recebe como parâmetro a matriz e o comprimento da mesma, linha 37. Por fim, depois de terminadas as operações, é preciso liberar a memória que foi alocado no *device* é, como é possível ver nas linhas 40, 41 e 42.

Bseado no que foi explanado neste capitulo pode-se notar que JCuda oferece uma boa alternativa para quem deseja trabalhar com cálculos, pesadamente paralelos, utilizando a GPU por meio de programas escritos em JAVA, algumas informações extras podem ser obtidas facilmente no Site[1]. JCuda é publicado sob a licença do MIT[2].

4.4 Programação em OpenCl

O OpenCL™(Open Computing Language) é definido como um padrão aberto, atualmente está sendo mantido pelo Khronos Group®, que permite o uso de GPUs para desenvolvimento de aplicações paralelas. Ele também permite que os desenvolvedores escrevam códigos de programação heterogêneos, fazendo com que estes programas consigam aproveitar tanto os recursos de processamento das CPUs quanto das GPUs. Por ser um framework aberto e padronizado, independente de fabricante, OpenCL aparece como uma alternativa conveniente para sistemas computacionais que demandem desempenho e portabilidade. Além disso, segundo Tsuchiyama *et al.* (2010), permite a programação paralela usando paralelismo de dados e de tarefas.

Apesar de se tratar de um sistema aberto, o Khronos Group® é responsável pela padronização de alguns parâmetros. O Khronos Group® anunciou recentemente uma versão atualizada do OpenCL. O OpenCL 2.0 é a mais recente evolução do padrão OpenCL, projetado para simplificar ainda mais a programação multiplataforma, permitindo uma variedade de algoritmos e padrões de programação para serem facilmente acelerados (GROUP, 2014). Ainda, poderá fornecer uma série de vantagens interessantes para os desenvolvedores.

O OpenCL também fornece uma linguagem e várias APIs, permitindo que os programadores acelerem aplicações por meio da exploração de paralelismo de dados ou tarefas. Os dispositivos em OpenCL podem ou não compartilhar memória com a CPU e, normalmente, têm um conjunto de instruções de máquina diferente (Stone *et al.*, 2010). As APIs fornecidas pelo OpenCL incluem funções para enumerar os dispositivos disponíveis (CPU, GPU

[1]Disponível em: http://www.jcuda.org/
[2]Disponível em: http://www.jcuda.org/License.txt

e outros aceleradores), gerenciar as alocações de memória, realizar transferências de dados entre CPU e GPU, invocar *kernels* para serem executados nos núcleos da GPU, e verificar erros.

Alguns programadores afirmam que CUDA é mais "maduro"(eficiente) e contém APIs de alto nível, as quais são mais convenientes. Entretanto, isso pode mudar haja vista que o OpenCL vem se aprimorando (NVIDIA, 2009). Uma vantagem do OpenCL é o fato de ele permitir que qualquer fornecedor implemente suporte OpenCL para seus produtos. O modelo de programação em OpenCL é semelhante ao utilizado em CUDA.

Enquanto CUDA é mantido e aprimorado apenas pela NVIDIA®, o OpenCL é suportado por fabricantes como, por exemplo: AMD®, NVIDIA®, APPLE®, INTEL® e IBM®. No entanto, apesar de se tratar de um modelo heterogêneo, permitindo o gerenciamento para portabilidade em multiplataformas. Diversos trabalhos já utilizaram CUDA ou OpenCL para solucionar vários tipos de problemas paralelizáveis. Outros trabalhos apresentaram uma comparação de desempenho computacional entre ambos os modelos. Entretanto, com base na revisão bibliográfica realizada e em alguns trabalhos como o de (Martins de Paula , 2014), não foi encontrado algum artigo completo e recente que realizasse uma comparação teórica, destacando claramente qual modelo, de fato, pode ser considerado mais adequado.

4.4.1 Interoperabilidade entre OpenCL e Java

Um dos principais motivos para a escolha de OpenCL, foi o fato desta tecnologia também oferecer uma maneira de trabalhar utilizando o poder computacional das GPUs, de forma fácil e intuitiva, com aplicações Java por meio da biblioteca JOCL (Java bindings for OpenCL), esta biblioteca oferece ligações de Java para OpenCL que são muito semelhantes aos da API original do OpenCL. As funções são fornecidas como métodos estáticos, a semântica e assinatura destes métodos foi mantida de acordo com as funções das bibliotecas originais, exceto para as limitações específicas, impostas pela linguagem Java.

A API OpenCL pode ser muito detalhada em alguns pontos, e isso não está oculto ou simplificado, mas é explicitamente evidente nos códigos implementados em JOCL. O objetivo da biblioteca JOCL é fornecer uma abstração orientada a objeto do OpenCL para Java, algumas informações extras podem ser obtidas facilmente no Site [1].

4.4.2 Exemplo de aplicação em JOCL

```
1  import static org.jocl.CL.*;
2
3  import org.jocl.*;
4
5  public class JOCLMultiplica {
6    /**
7     * Programa a ser executado no kernel
8     */
9    private static String programaFonte = "__kernel void "
10       + "multiplicaKernel(__global const float *a,"
```

[1] Disponível em: http://www.jocl.org/

```
      + "              __global const float *b,"
      + "              __global float *c)" + "{"
      + "    int gid = get_global_id(0);"
      + "    c[gid] = a[gid] * b[gid];" + "}";

  public static void main(String args[]) {
    // Cria dados de entrada e saida
    int n = 10;
    float entradaA[] = new float[n];
    float entradaB[] = new float[n];
    float saidaC[] = new float[n];
    for (int i = 0; i < n; i++) {
      entradaA[i] = i;
      entradaB[i] = i;
    }
    Pointer entA = Pointer.to(entradaA);
    Pointer entB = Pointer.to(entradaB);
    Pointer saiC = Pointer.to(saidaC);

    // Plataforma, numero e tipo do dispositivo que vai ser utilizada
    final int platformIndex = 0;
    final long deviceType = CL_DEVICE_TYPE_ALL;
    final int deviceIndex = 0;

    // Habilitar excecoes e, posteriormente, omiti verificacoes de erro
       nesta amostra
    CL.setExceptionsEnabled(true);

    // Obtem o numero da plataforma
    int numPlatformsArray[] = new int[1];
    clGetPlatformIDs(0, null, numPlatformsArray);
    int numPlatforms = numPlatformsArray[0];

    // Obtem o numero do ID da plataforma
    cl_platform_id platforms[] = new cl_platform_id[numPlatforms];
    clGetPlatformIDs(platforms.length, platforms, null);
    cl_platform_id platform = platforms[platformIndex];

    // Inicializa as propriedades de contexto
    cl_context_properties contextProperties = new cl_context_properties
       ();
    contextProperties.addProperty(CL_CONTEXT_PLATFORM, platform);

    // Obtem o numero do dispositivos para a plataforma
    int numDevicesArray[] = new int[1];
    clGetDeviceIDs(platform, deviceType, 0, null, numDevicesArray);
    int numDevices = numDevicesArray[0];

    // Obtem o ID do dispositivo
    cl_device_id devices[] = new cl_device_id[numDevices];
    clGetDeviceIDs(platform, deviceType, numDevices, devices, null);
    cl_device_id device = devices[deviceIndex];

    // Cria um contexto para o dispositivo selecionado
    cl_context context = clCreateContext(contextProperties, 1,
```

```
64          new cl_device_id[] { device }, null, null, null);
65
66      // Cria um comando da fila para o dispositivo selecionado
67      cl_command_queue commandQueue = clCreateCommandQueue(context, device
            ,
68          0, null);
69
70      // Aloca os objetos da memoria para os dados de entrada e de saida
71      cl_mem memObjects[] = new cl_mem[3];
72      memObjects[0] = clCreateBuffer(context, CL_MEM_READ_ONLY
73          | CL_MEM_COPY_HOST_PTR, Sizeof.cl_float * n, entA, null);
74      memObjects[1] = clCreateBuffer(context, CL_MEM_READ_ONLY
75          | CL_MEM_COPY_HOST_PTR, Sizeof.cl_float * n, entB, null);
76      memObjects[2] = clCreateBuffer(context, CL_MEM_READ_WRITE,
77          Sizeof.cl_float * n, null, null);
78
79      // Cria o programa a partir do codigo fonte
80      cl_program program = clCreateProgramWithSource(context, 1,
81          new String[] { programaFonte }, null, null);
82
83      // Cria o programa
84      clBuildProgram(program, 0, null, null, null, null);
85
86      // Cria o kernel
87      cl_kernel kernel = clCreateKernel(program, "multiplicaKernel", null)
            ;
88
89      // Define os argumentos para o kernel
90      clSetKernelArg(kernel, 0, Sizeof.cl_mem, Pointer.to(memObjects[0]));
91      clSetKernelArg(kernel, 1, Sizeof.cl_mem, Pointer.to(memObjects[1]));
92      clSetKernelArg(kernel, 2, Sizeof.cl_mem, Pointer.to(memObjects[2]));
93
94      // Define as dimensoes do item de trabalho
95      long global_work_size[] = new long[] { n };
96      long local_work_size[] = new long[] { 1 };
97
98      // Execute o kernel
99      clEnqueueNDRangeKernel(commandQueue, kernel, 1, null,
            global_work_size,
100         local_work_size, 0, null, null);
101
102     // Le os dados de saida
103     clEnqueueReadBuffer(commandQueue, memObjects[2], CL_TRUE, 0, n
104         * Sizeof.cl_float, saiC, 0, null, null);
105
106     // Lanca o kernel, os objetos de programa, a memoria
107     clReleaseMemObject(memObjects[0]);
108     clReleaseMemObject(memObjects[1]);
109     clReleaseMemObject(memObjects[2]);
110     clReleaseKernel(kernel);
111     clReleaseProgram(program);
112     clReleaseCommandQueue(commandQueue);
113     clReleaseContext(context);
114
115     if (n <= 10) {
```

```
116         System.out.println("Resultado: "
117             + java.util.Arrays.toString(saidaC));
118       }
119     }
120 }
```

Código 4.3: *Exemplo de programa desenvolvido em JOCL*

O Código 4.3, referente a uma aplicação desenvolvida em JOCL para calcular a multiplicação dos elementos de dois vetores, cada elemento de um vetor é multiplicado pelo elemento de outro vetor de modo que os índices sejam iguais. Como é possível ver na linha 9, o código OpenCL — que será executado na GPU — é embutido em uma *String*, a qual recebeu o nome *programaFonte*, esta *String* contém a função que calcula o quadrado dos números de forma paralela.

Da linha 19 até a linha 28 são configurados os dois vetores de entrada, os quais contêm os números que serão multiplicados e o vetor de saída onde os resultados das multiplicações serão inseridos. Entre as linhas 36 e 68 são realizadas algumas configurações relacionas a plataforma e ao dispositivo *device*. Da linha 71 até a linha 77 é realizada a alocações de memória, as linhas 80 e 84 criam o programa a partir do código contido em *programaFonte*, enquanto a linha 87 cria a função que será executada na GPU. As linhas 90, 91 e 92 definem os argumentos que a função do *kernel* deverá receber, respectivamente. A linha 103 lê os dados de saída do *kernel* para serem impressos na linha 116.

4.5 Resumo do Capítulo

Este capítulo explanou sobre a programação para a placa de vídeo, realizando algumas comparações entre uma GPU e uma CPU, evidenciando as principais características de cada uma. Foi discutido sobre a programação em CUDA e alguns pacotes para trabalhar com JAVA e com algumas funções algébricas que demandam maior poder computacional. No final do capitulo explanou-se com relação à OpenCL, juntamente com suas principais características e diferenças entre CUDA, foi demonstrado, também, um exemplos de JOCL, uma biblioteca que promove interoperabilidade entre OpenCL e Java.

Capítulo 5

Computação quântica

Neste capítulo apresentam-se conceitos da computação quântica que servem de base para a compreensão deste trabalho. Na Seção 5.1 é realizada uma pequena explanação com relação a mecânica quântica e as características desta área que a tornam tão intrigante. Enquanto que na Seção 5.2 defini-se brevemente o que é a computação quântica e são apresentados alguns conceitos fundamentais, juntamente com as dificuldades enfrentadas em construir-se um computador quântico. Na Seção 5.3 são apresentadas algumas definições sobre circuitos quânticos. E na Seção 5.4 explana-se sobre algumas das ferramentas para construção de circuitos quânticos existentes, evidenciando as principais características de cada uma. A Seção 5.5 é um resumo do capítulo

5.1 Mecânica quântica

A computação quântica surge com a promessa de realizar tarefas que a computação clássica, na maioria das vezes, não consegue solucionar de forma satisfatória, devido a limitações impostas por fenômenos físicos na esfera subatômica, nesta perspectiva, surgem várias propostas para a construção de um computador quântico. A princípio é constatado que o computador quântico é uma construção teórica, no entanto, se não houvesse a possibilidade de implantação na natureza de máquinas que processem a informação quântica, este campo de pesquisa seria apenas uma curiosidade matemática (Nielsen *et al.*, 2005), e não despertaria nenhum interesse por parte da comunidade cientifica.

É necessário frisar que implantar experimentalmente circuitos quânticos e sistemas de comunicação, tem se tornado um grande desafio, na medida em que é difícil controlar um sistema quântico individual e é de vital importância a manipulação de tais sistemas para que de fato a computação quântica seja uma realidade (Rodrigues, 2011).

No entanto, antes de falar sobre computação quântica é importante conhecer alguns dos princípios que regem a física quântica, segundo Bernstein e Vazirani (1993):

- O principio da superposição explica como uma partícula pode ser sobreposta em dois estados ao mesmo tempo;

- O principio da medição nos diz como medir uma partícula que muda de estado, e quanta informação pode-se acessar da mesma;
- E o principio da evolução unitária rege a forma como o estado do sistema quântico evolui no tempo.

O formalismo da MQ (Mecânica Quântica) é a estrutura matemática em que são postulados os conceitos primitivos pelos quais a teoria quântica é fundamentada. A descrição de um sistema físico é baseada em dois conceitos fundamentais: estado e observável.

5.1.1 Estado Físico de um sistema

A MQ é fundamentada no fato de que um vetor de estado $|\psi\rangle$, descreve totalmente um sistema físico. Este vetor de estado contém, dentro das limitações impostas pelo Princípio de (Heisenberg, 2013), a informação máxima sobre o sistema quântico. De acordo com o Postulado 1: *A todo sistema físico existe associado um espaço vetorial complexo com produto interno, chamado espaço de Hilbert, conhecido como espaço dos estados do sistema, cujos vetores de estados descrevem completamente os estados do sistema.*

Os vetores de estados pertencem a um espaço vetorial de dimensão infinita, onde é definido um produto escalar:

$$\langle u \mid v\rangle = \langle v \mid u\rangle^*, \tag{5.1}$$

No que concerne a teoria quântica, é importante para sua interpretação probabilística que todos os vetores da MQ tenham métrica unitária, ou seja, é necessário que $|\Psi\rangle$ seja normalizado. Assim, exceto para o *ket* nulo, um vetor $|u\rangle$ pode ser colocado na forma normalizada:

$$|\ u'\rangle = \frac{1}{\sqrt{\langle u \mid u\rangle}}|\ u\rangle, \tag{5.2}$$

que tem a propriedade de ortonormalidade,

$$\langle u' \mid u'\rangle = 1, \tag{5.3}$$

Ainda, tratando da interpretação probabilística da MQ, os vetores necessitam ser ortogonais,

$$\langle u \mid v\rangle = 0. \tag{5.4}$$

Assim, os vetores da base $\{|n\rangle\}$ são, simultaneamente, *ortonormais e ortogonais*: "A probabilidade de o vetor $|u\rangle$ ser ele próprio é 100% e de ser um vetor arbitrário $|v\rangle$ é zero".

Essa interpretação pode ser representada por um delta de Kronecker:

$$\langle u \mid v \rangle = \delta_{uv} = \begin{cases} 1, & u = v, \\ 0, & u \neq v \end{cases} \tag{5.5}$$

A norma tem valor positivo definido,

$$\langle u \mid v \rangle \geqslant 0. \tag{5.6}$$

5.1.2 Observáveis

É dito observável, toda quantidade física que se pode medir em um dado sistema, como, por exemplo: a posição, o momento, a energia e a componente do spin em uma direção arbitrária. O conceito de operador está estritamente relacionado à medida do observável. Na constituição do formalismo da Mecânica Quântica, os observáveis são representados por operadores hermitianos definidos sobre o espaço de Hilbert. De acordo com o Postulado 2: *Qualquer quantidade física* **A** *mensurável é descrita matematicamente por um operador* **A** *que atua sobre o espaço de estados do sistema.* Matematicamente, representamos o operador, como:

$$A|\psi\rangle = |\psi'\rangle, \tag{5.7}$$

onde **A** é o operador que atua em $|\psi\rangle$ produzindo um outro vetor $|\psi'\rangle$ no mesmo espaço.

5.1.3 Evolução de um sistema quântico

É importante saber como o sistema evolui no tempo. A lei ou equação de movimento que rege a evolução do vetor de estado no tempo para situações não relativísticas é a equação de (Schrödinger, 1992). De acordo com o Postulado 3: *A evolução temporal do estado de um sistema quântico é descrita pela equação de Schödinger*:

$$i\hbar \frac{d|\psi\rangle}{dt} = H|\psi\rangle. \tag{5.8}$$

Sendo H o operador hamiltoniano do sistema que está associado à energia total do sistema físico em questão. Assim, a equação, por meio de H, depende do sistema físico em consideração.

Se o hamiltoniano de um sistema for conhecido, sua dinâmica será completamente determinada (em princípio). Geralmente, descobrir o hamiltoniano de um sistema particular é um pouco complicado. É importante salientar que, dado o hamiltoniano de um sistema

isolado, a MQ nos ensina como calcular os observáveis físicos em qualquer instante de tempo posterior, mas ela não diz como encontrar o hamiltoniano de um sistema (Nielsen *et al.* , 2005).

Sendo o hamiltoniano um operador hermitiano, ele possui uma decomposição espectral

$$H = \sum_E E|E\rangle\langle E|, \tag{5.9}$$

com autovalores E e autovetores $|E\rangle$.

No caso de um q-bit, qualquer operador unitário pode ser implementado em sistemas físicos reais. Assim, podemos dizer que a evolução de um sistema quântico é descrita por uma transformação unitária U:

$$|\psi'\rangle = U|\psi\rangle. \tag{5.10}$$

5.2 Computação quântica

Por volta da década de 1970 avanços experimentais em diversas áreas permitiram que experimentos pudessem ser feitos em números cada vez menores de partículas, deste modo, tornando "visíveis" os efeitos quânticos mais fundamentais (Oliveira, 2005).

Neste contexto se desenvolveu a Computação Quântica (CQ), talvez a mais espetacular proposta de aplicação prática da MQ. Para fins didáticos denomina-se Informação Quântica (IQ) a identificação e o estudo dos recursos quânticos utilizáveis na área da informação, e Computação Quântica a aplicação direta desses recursos em chaves lógicas, algoritmos, etc. A Tabela 5.1 exposta no trabalho de (Oliveira e Sarthour, 2004), resume o desenvolvimento desta área desde os seus primórdios até os dias atuais.

Notoriamente existem várias outras descobertas de grande impacto na Computação Quântica, no entanto, a Tabela 5.1 evidência as descobertas que pavimentaram as bases desta área.

5.2.1 Qubit's

Na computação clássica utiliza-se uma base numérica binária que usualmente é representada por 0 e 1, no entanto, como na computação quântica utiliza-se o estado das partículas para tal representação, os bit's 0 e 1 da computação clássica são substituídos por vetores unitários no espaço de Hilbert complexo $\mathbb{C}^2$ onde:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ e} \tag{5.11}$$

Ano	Fato
1973	- Demonstrada a possibilidade de computação (clássica) reversível por Charles Bennett.
1982	- Proposta de computador quântico por Paul Benioff baseado no trabalho de Charles Bennett de 1973.
1984	- Charles Bennett e Gilles Brassard descobrem o protocolo de criptografia quântica BB84.
1985	- David Deutsch cria o primeiro algoritmo quântico.
1993	- Peter Shor cria o algoritmo de fatoração. Neste ano é também descoberto o teleporte quântico por Charles Bennett e colaboradores.
1994	- Lov Grover cria o algoritmo de busca.
1996	- Um grupo da IBM demonstra experimentalmente o BB84 utilizando fótons enviados por fibras comerciais de telecomunicações.
1997	- Neil Gershenfeld e Isaac Chuang descobrem os estados pseudo-puros e fazem eclodir a CQ por Ressonância Magnética Nuclear (RMN).
1998	- Este foi o ano da Computação Quântica por RMN. Implementações de várias chaves lógicas quânticas são demonstradas através dessa técnica. São demonstrados também por RMN os algoritmos de busca e de teleporte.
2001	- É demonstrado o algoritmo de Shor por RMN.
2003	- Demonstração de emaranhamento entre os spins do núcleo e de um elétron na mesma molécula combinando-se as técnicas de RMN e RPE (Ressonância Paramagnética Eletrônica).

Tabela 5.1: *Desenvolvimento da Computação Quântica*

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{5.12}$$

Chamados de *qubit's* notação de Dirac.

Notoriamente, o conjunto $\{|0\rangle, |1\rangle\}$ forma uma base no espaço de Hilbert de duas dimensões, chamada de base computacional. No caso de um spin $\frac{1}{2}$ representar o q-bit, pode-se identificar a seguinte relação: $|0\rangle \equiv |\uparrow\rangle$ e $|1\rangle \equiv |\downarrow\rangle$.

Fisicamente, q-bits são representados por qualquer objeto quântico que possua dois autoestados bem distintos. Os exemplos mais comuns são: estados de polarização de um fóton (horizontal ou vertical), elétrons em átomos de dois níveis (o que é uma aproximação), elétrons em poços quânticos, e spins nucleares.

Se o estado da partícula está isolado e não houver nenhuma medição, o estado da mesma é descrito por uma combinação linear dos vetores representados por: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ onde, α e β são números complexos e a soma das magnitudes $|\alpha|^2$ e $|\beta|^2$ deve ser menor igual a 1 (Portugal, 2010).

5.2.2 Estados de Sistemas Compostos

A expressão "entrelaçamento" foi elaborada por Schrödinger (1935), também chamada de emaranhamento quântico, é, portanto, um evento estudado pela Mecânica Quântica. Segundo esta teoria, dois ou mais "objetos" podem estar conectados de tal forma que um "objeto" não pode ser analisada de forma adequada sem que o outro "objeto", conectado, seja diretamente afetado, mesmo que ambos estejam localizados em dimensões espaciais distintas (Cornelio, 2008). Dessa forma, mesmo que uma partícula esteja neste Planeta e sua partícula conectada esteja situada em outro, separados anos-luz de distância, se em uma ocorrer alguma interação na outra também ocorrerá, independentemente do tempo que a luz levar para viajar de um lugar a outro.

O espaço de estados de um sistema composto é o produto tensorial dos espaços de estados dos componentes. Se $|\psi_1\rangle, ..., |\psi_n\rangle$ descrevem os estados de n sistemas quânticos isoladamente, o estado do sistema composto é $|\psi_1\rangle \otimes ... \otimes |\psi_n\rangle$.

Um exemplo de sistema composto é a memória de um computador quântico de *n qubit's*. Usualmente, a memória é dividida em conjunto de *qubit's*, chamado de *registradores*. O espaço de estados da memória do computador é o produto tensorial dos espaços de estados dos registradores que, por sua vez, são obtidos pelo produto tensorial repetido do espaço de Hilbert $\mathbb{C}^2$ de cada *qubit*.

O espaço de estados da memória de um computador quântico de 2 *qubit's* é $\mathbb{C}^4 = \mathbb{C}^2 \otimes \mathbb{C}^2$. Portanto, qualquer vetor unitário de $\mathbb{C}^4$ representa o estado quântico de 2 *qubit's*. Por exemplo, o vetor:

$$|0,0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{5.13}$$

que pode ser escrito como $|0\rangle \otimes |0\rangle$, representa o estado de 2 elétrons ambos com spin para cima. Interpretação análoga se aplica a $|0,1\rangle$, $|1,0\rangle$, $|1,1\rangle$. Considere agora o vetor unitário de $\mathbb{C}^4$ dado por:

$$|\psi\rangle = \frac{|0,0\rangle + |1,1\rangle}{\sqrt{2}} \tag{5.14}$$

Para saber qual é o estado de spin de cada elétron nesse caso, é preciso fatorar $|\psi\rangle$ da seguinte forma:

$$\frac{|0,0\rangle + |1,1\rangle}{\sqrt{2}} = (a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle) \tag{5.15}$$

Podemos expandir o lado direito e igualar os coeficientes montando um sistema de equa-

ções para achar a, b, c e d. O estado do primeiro *qubit* será $a|0\rangle + b|1\rangle$ e do segundo $c|0\rangle + d|1\rangle$. Porém, há um problema: o sistema de equações não tem solução, ou seja, não existem coeficientes a, b, c e d, que satisfaçam à Equação 5.15. Todo estado de um sistema composto que não pode ser fatorado é chamado de emaranhado (Steane, 1998). Esses estados são bem definidos quando olhamos o sistema composto como um todo, porém não podemos atribuir estados para as partes.

5.2.3 Processo de Medida

Em geral, medir um sistema quântico que se encontra no estado $|\psi\rangle$, tem como intuito obter informações clássicas, no que diz respeito, a esse estado. Na prática, a medida é feita no laboratório usando instrumentos como lasers, magnetos, escalas e cronômetros. Na teoria, descrevemos o processo matematicamente de modo que haja correspondência com o que ocorre na prática. Medir um sistema físico que se encontra em um estado desconhecido, em geral, perturba esse estado de forma irreversível. Não tem como recuperar ou conhecer o estado antes da execução da medida. Se o estado não foi perturbado, então não foi possível obter qualquer informação sobre ele. Matematicamente, a perturbação é descrita por um *projetor*. Se esse projetor for sobre um espaço unidimensional, então diz-se que o estado quântico *projetor* passa a ser descrito pelo vetor unitário pertencente ao espaço unidimensional. No caso geral, a projeção é sobre um espaço vetorial de dimensão maior que 1, e assim, diz-se que o colapso é parcial ou, no caso extremo, não há alteração no estado quântico do sistema.

Uma *medida projetiva* é descrita por um operador hermitiano O, chamado de observável, no espaço de estados do sistema, que está sendo medido. O observável O tem uma *representação diagonal*:

$$O = \sum_{\lambda} \lambda P_{\lambda} \tag{5.16}$$

onde P_{λ} é o projetor no auto-espaço de O associado ao autovalor λ. Os possíveis resultados da medida correspondem aos autovalores λ do observável. Se o estado do sistema no momento da medida for $|\psi\rangle$, a probabilidade de se obter o resultado λ será:

$$P_{\lambda} = \langle\psi|P_{\lambda}|\psi\rangle \tag{5.17}$$

Se o resultado da medida for λ, o estado do sistema quântico imediatamente após a medida será (Portugal, 2010):

$$\frac{1}{\sqrt{P_{\lambda}}} P_{\lambda}|\psi\rangle \tag{5.18}$$

5.3 Circuitos quânticos

Segundo McMahon (2007), um conjunto de portas lógicas podem ser ligadas entre si para construir circuitos digitais. Um exemplo simples de porta é a NOT que inverte o valor de um bit de entrada, por exemplo:

Entrada	**Saida**
0	1
1	0

Tabela 5.2: *Entrada e Saida da porta NOT*

No caso quântico pode-se representar as portas por matrizes. Considerando que a evolução unitária de um *qubit* acontece em um espaço de Hilbert $\mathbb{C}^2$, especificada pelo mapeamento das bases $|0\rangle$ e $|1\rangle$ ortogonais para os estados $|v_0\rangle = a|0\rangle + b|1\rangle$ e $|v_1\rangle = c|0\rangle + d|1\rangle$. Especificados pela transformação linear no $\mathbb{C}^2$.

$$U = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Dado seu conjugado transposto:

$$U^T = \begin{pmatrix} *a & *c \\ *b & *d \end{pmatrix}$$

É possível notar que as operações $U^T U = UU^T = I$ são equivalentes. Os portões apresentados em seguida operam em um único *qubit*, são alguns dos mais utilizados na computação quântica:

- Porta de Hadamard;

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{5.19}$$

- Porta de Rotação;

$$U = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{5.20}$$

- Porta NOT;

$$NOT = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{5.21}$$

- Porta de modificação de fase.

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{5.22}$$

Um dos portões mais básicos de dois *qubit's* é o CNOT, representado pela matriz (DiVincenzo, 1995):

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{5.23}$$

Seu circuito quântico é exibido na Figura 5.1.

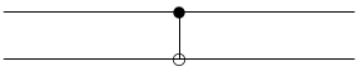
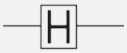


Figura 5.1: *Circuito da porta CNOT*

A Tabela 5.3 exibe algumas portas de um *qubit*:

Nome	**Símbolo**	**Matariz**
Hadamard		$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
Pauli-X		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli-Y		$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$

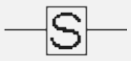
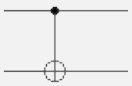
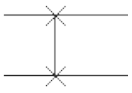
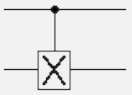
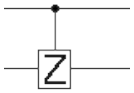
Pauli-Z		$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Fase		$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
$\frac{\pi}{8}$		$\begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix}$
Medida		Projeção sobre $\|0\rangle$ e $\|1\rangle$

Tabela 5.3: *Portas lógicas em circuitos quânticos de um qubit*

Enquanto a Tabela 5.3 exibe as portes de um *qubit* a Tabela 5.4 mostra as portas de dois *qubit's*:

Nome	**Símbolo**	**Matariz**
Não-Controlado		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
Troca		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
X-Controlado		$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$
Z-Controlado		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$

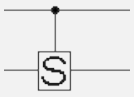
Fase-Controlada		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{pmatrix}$

Tabela 5.4: *Portas lógicas em circuitos quânticos de dois qubit's*

A Tabela 5.5 exibe algumas portas de três *qubit's*:

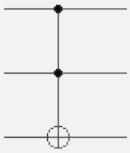
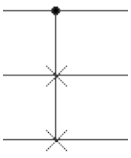
Nome	**Símbolo**	**Matariz**
Toffoli		$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$
Fredkin		$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$

Tabela 5.5: *Portas lógicas em circuitos quânticos de três qubit's*

Todas as portas apresentadas nas Tabelas (5.3, 5.4 e 5.5) com as respectivas matrizes que as representam estão presentes no LinDCQ, ferramenta desenvolvida nesta dissertação.

5.4 Ferramentas para construção de circuitos quânticos

O modelo de computador quântico universal só passa a ser valorizado, de certa forma, quando Deutsch cria a linguagem de circuitos quânticos (Deutsch, 1989), que passa a ser amplamente aceita pela comunidade, pelo fato de ser similar a linguagem de circuitos clássica. Vários algoritmos quânticos foram criados, com o intuito de ilustrar as vantagens da computação quântica, por exemplo: os algoritmo de (Bernstein e Vazirani, 1993), e o algoritmo de (Simon, 1997). No entanto, somente com o trabalho de (Shor, 1994a) (algoritmo que realiza fatoração de números inteiros em tempo polinomial) que a computação quântica ganho mais notoriedade. Assim, a computação quântica deixou de ser vista somente como uma curiosidade acadêmica.

Inerente à enorme importância adquirida pelos sistemas de criptografia de chave pública, o algoritmo de Shor fomento o aprofundamento no estudo da computação quântica, tanto no que diz respeito à construção de computadores quânticos, quanto no desenvolvimento de algoritmos. Os algoritmos de (Grover, 1996), para busca em bases de dados desordenadas em $O(n)$ e de (Jozsa, 1998), que exibe um algoritmo de fatoração alternativo, são exemplos de outros algoritmos.

Com relação à construção de máquinas quânticas, alguns projetos desenvolveram sistemas quânticos com poucos *qubit's*, dentre os quais podem ser citados (Jones e Mosca, 1998), (Jones *et al.*, 1998), (Chuang *et al.*, 1998) e recentemente (D-Wave, 2007). Contudo, os sistemas implementados não são suficientes para realizar computações úteis, sendo assim, pode-se afirmar que ainda não existe um computador quântico efetivo.

O trabalho de (Andrade Barbosa, 2007), cita um simulador de circuitos quânticos. O mesmo aponta que uma das dificuldades em tal simulação se baseia no fato de o cálculo de algumas operações exigir razoável poder computacional do computador. No entanto, existem várias outras ferramentas para se trabalhar com circuitos quânticos, como, por exemplo:

- *Libquantum*, Figura 5.2, é uma biblioteca para a linguagem C criado na Alemanha (B. Butscher, 2013), que pode ser utilizado para simular circuitos e algoritmos quânticos, está disponível para download na Web[1]. É eficiente e possui alguns algoritmos já pré-implementados. No entanto é executado na linha de comandos e é necessário saber programar para o utilizar. Desse modo, a utilização desta biblioteca para simular circuitos quânticos torna-se bastante complexa.

- *Quantum Computer Simulator*, Figura 5.3, programa criado na Universidade de Patras da Grécia (Kyriakos Sgarbas, 2010), para utilizar a aplicação é necessário acessar à pagina Web[2]. Pode ser utilizado em qualquer sistema operativo, uma vez que é executado a partir do servidor do site. No entanto é de difícil utilização, já que não possui uma interface gráfica avançada, e oferece um leque pequeno de portas quânticas para utilizar.

- O programa *QCAD*, Figura 5.4, foi criado nas universidades de Tokyo e de Nagoya no Japão (H. Watanabe, 2011), está disponível para download na Web[3]. É uma aplicação utilizável apenas no sistema operativo Windows, no entanto tem uma boa interface

[1]Disponível em: http://www.libquantum.de/
[2]Disponível em: http://www.wcl.ece.upatras.gr/ai/resources/demo-quantum-simulation
[3]Disponível em: http://qcad.sourceforge.jp/

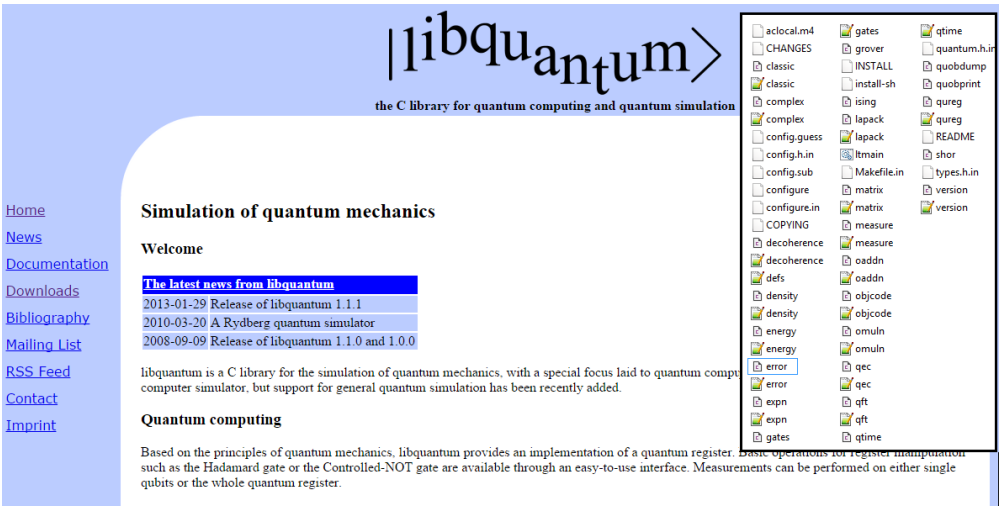


Figura 5.2: *Libquantum*

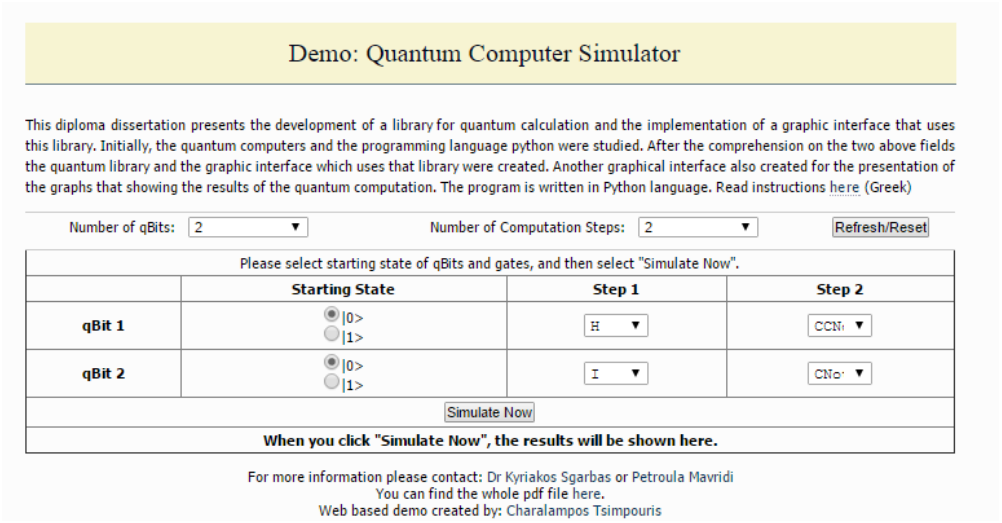


Figura 5.3: *QCS*

gráfica e várias portas quânticas. Porém, não é possível visualizar os resultados na forma matricial e não tem opção de criar portas personalizadas pelo utilizador.

- QCS (Simulador de Circuitos Quânticos), Figura 5.5, foi desenvolvido na Faculdade de Engenharia da Universidade do Porto, por (Figueiredo, 2012). Para utilizar a aplicação é necessário acessar à pagina Web[4]. O QCS tem uma interface gráfica, de fácil usabilidade, onde o usuário pode construir e visualizar o seu circuito. No entanto, como ele foi implementado em flash, na linguagem Actionscript, a simulação torna-se muito lenta quando o número de qubit's cresce e está limitado a um pequeno número de portas, tornando-se difícil simular alguns circuitos mais complexos. Também não é possível simular o protocolo de teletransporte quântico, por causa do modo como a medição é executada.

- O simulador *jQuantum*, Figura 5.6, foi criado na linguagem de programação Java (Vries, 2010), desta forma pode ser utilizado em qualquer sistema operacional desde que este já tenha instalado o plug-in necessário. O jQuantum pode ser baixado

[4]Disponível em: http://quark.fe.up.pt/qcs

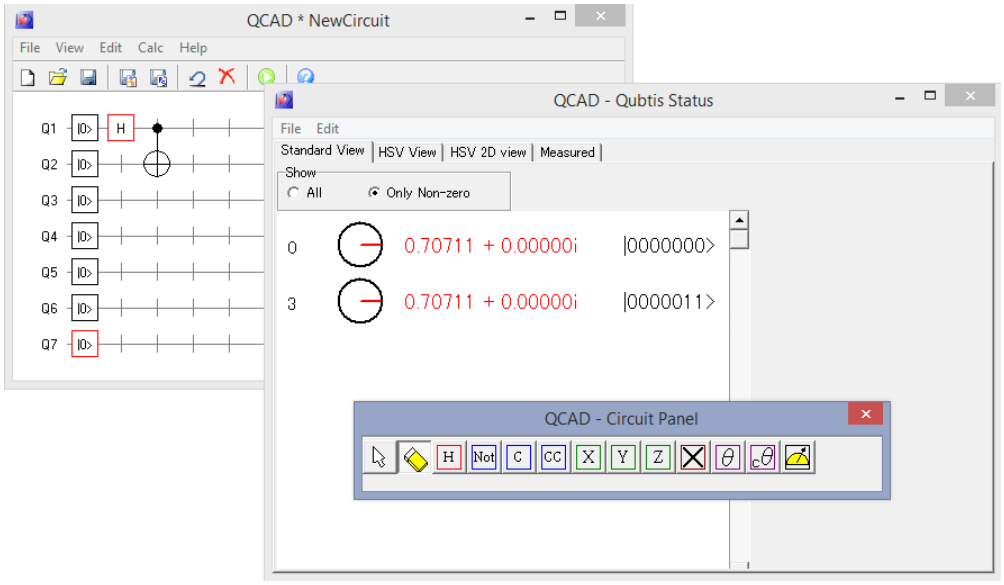


Figura 5.4: *QCAD*

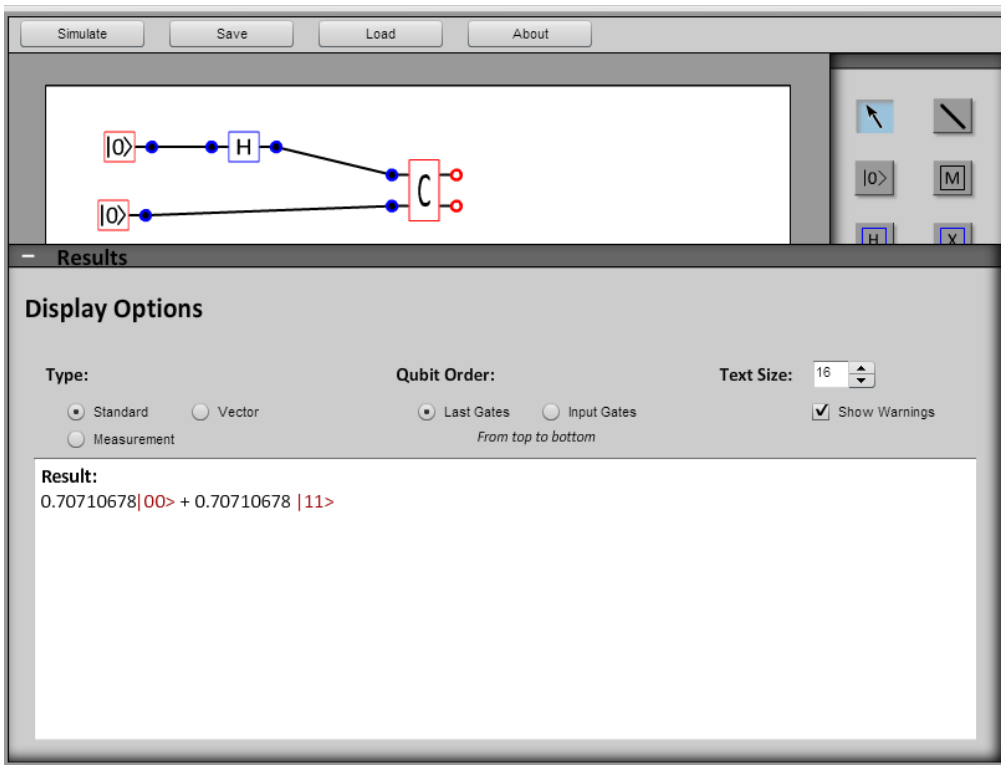


Figura 5.5: *Simulador de Circuitos Quânticos*

facilmente na Web[5], onde, também, é possivel encontrar a documentação dele. Este programa, tal como o *QCAD*, possui uma boa interface gráfica e é um dos melhores programas, atuais, para simulação de circuitos quânticos, tem uma grande variedade de portas quânticas predefinidas para utilizar e é eficiente na simulação. No entanto, não possui a opção de adicionar portas personalizadas, e é difícil visualizar os resultados, especialmente quando estes têm diferentes fases.

[5]Disponível em: http://jquantum.sourceforge.net/

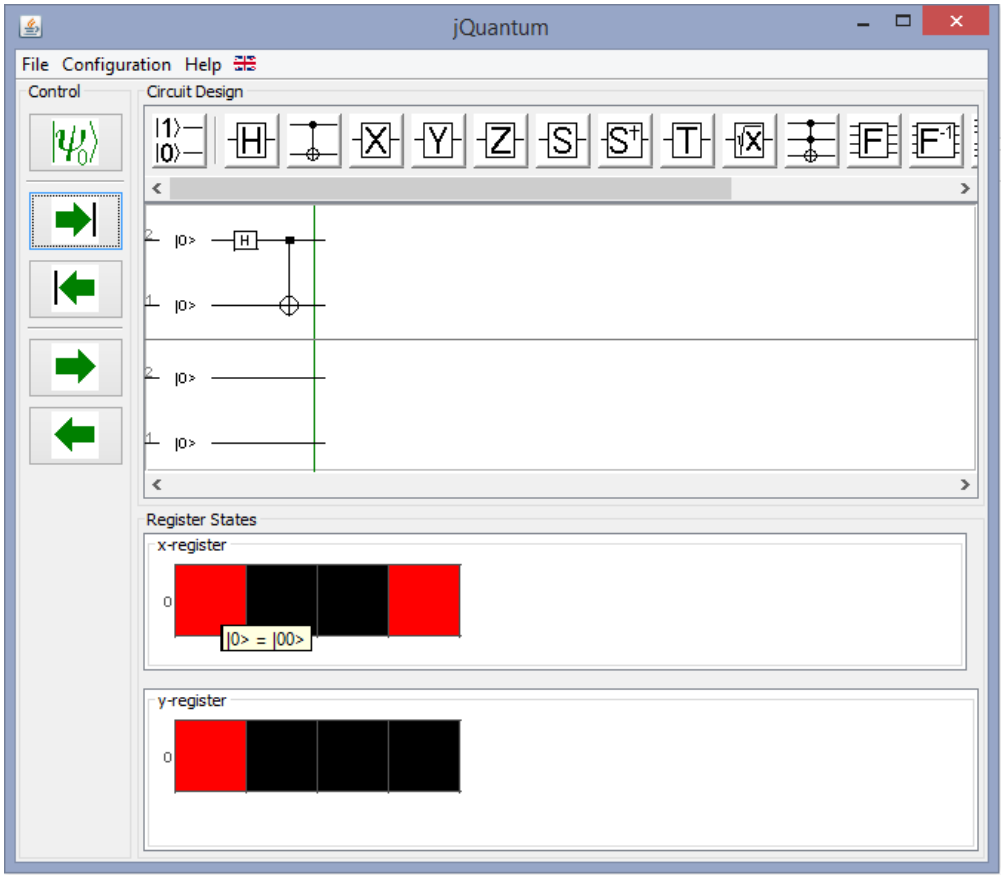


Figura 5.6: *JQuantum*

A Tabela 5.6 apresenta as principais características do LinDCQ, evidenciando suas vantagens e desvantagens em relação as outras ferramentas, as células da tabela com o "X" apresentam possíveis desvantagens, enquanto que as células com o "V", se referem as possíveis vantagens.

5.5 Resumo do Capítulo

Este capítulo iniciou com uma pequena explanação da mecânica quântica, descrevendo seus princípios. Conseguinte, foi exposta a computação quântica com as descobertas que pavimentaram as bases desta área, discutiu-se também com relação aos circuitos quânticos. Ao final, foi realizada uma revisão da literatura para evidenciar as principais ferramentas, existentes, para construção de circuitos quânticos.

Características	Libquantum	Quantum Computer Simulator	QCAD	QCS	jQuantum	LinDCQ
Executado na linha de comandos	✗					
Necessário saber programar	✗					✗
Qualquer sistema operativo		✓				✓
Leque pequeno de portas quânticas		✗		✗		
Resultados na forma matricial			✓	✓		✓
Criar portas personalizadas	✓			✓	✓	✓
Tem interface gráfica		✓	✓	✓	✓	✓
Resultado da medição no circuito		✓	✓	✓	✓	✓

Tabela 5.6: *Tabela comparativa entre as características de LinDCQ e demais ferramentas*

Capítulo 6

LinDCQ

Neste capítulo apresenta-se a ferramenta LinDCQ e uma pequena explanação sobre sua construção. Na Seção 6.1 fala-se brevemente sobre o desenvolvimento do compilador e das funções complementares, as quais compõem a fase de geração de código. A Seção 6.2 evidência a arquitetura das classes envolvidas no núcleo do seu desenvolvimento. Na Seção 6.3 são apresentadas algumas características do IDE, explanando as principais funcionalidades. Enquanto que na Seção 6.4 são mostrados alguns exemplos de circuitos desenvolvidos em LinDCQ, exibindo o fluxo que o circuito passa até ser executado. E na Seção 6.6 é mostrada uma pequena comparação de desempenho do LinDCQ, na CPU e GPU. A Seção 6.7 é um resumo do capítulo.

6.1 Desenvolvimento do Compilador

Este capítulo mostra uma visão geral do desenvolvimento da ferramenta LinDCQ, começando das partes fundamentais, primeiro explanando com relação à construção do seu compilador, em seguida do código fonte e sua arquitetura, e por fim, explanando alguns exemplos práticos de circuitos que foram desenvolvidos em LinDCQ. Assim, pode-se perceber que inicia-se explanando do nível mais baixo, até o nível mais alto da estrutura do LinDCQ.

Para o desenvolvimento do LinDCQ, primeiramente foi proposta a gramática, desenvolvida seguindo o modelo BNF. O Código 6.1 apresenta os *tokens* — os quais se encontram antes dos dois pontos — e os *lexemas* do compilador da aplicação, os conceitos de *tokens* e *lexemas* são discutidos na subseção 3.4.1:

```
1  //folhas
2  OP_par_esq:"("
3  OP_par_dir:")"
4
5  OP_col_esq:"["
6  OP_col_dir:"]"
7
8  OP_chav_esq:"{"
9  OP_chav_dir:"}"
10
```

```
OP_enquanto:"Enquanto"
OP_menor:"<"
OP_maior:">"
OP_fim:"Fim"

OP_fio:"Fio.ket"
OP_fio0:"Fio.ket0"
OP_fio1:"Fio.ket1"

OP_h:"H"
OP_x:"X"
OP_y:"Y"
OP_z:"Z"
OP_s:"S"
OP_t:"T"
OP_cnot:"CNOT"
OP_troca:"Troca"
OP_toffoli:"Toffoli"
OP_fredkin:"Fredkin"
OP_m:"M"

OP_virgula:","
OP_ponto_virgula:";"
OP_igual:"="

numeros:{N}*
palavras:{L}*
: {ws}
```

Código 6.1: *Tokens e Lexemas do compilador do LinDCQ*

É importante salientar que o compilador do LinDCQ é *case-sensitive*, ou seja, maiúsculas são diferentes de minúsculas, deste modo, "a" é diferente de "A". Este fato é evidenciado nas palavras reservadas, tais palavras sempre começam com a primeira letra maiúsculas, como foi possível ver em alguns *lexemas* no Código 6.1. Também foram definidas algumas ER (Expressões Regulares), com o intuído de trabalhar com *string* de números, palavras e alguns caracteres especiais. Pode-se ver tais ER no Código 6.2 que segue:

```
//exprecoes regulares
L: [a-z0-9]
N: [0-9]

ws: [\ \+\n\r\t]
```

Código 6.2: *Expressões regulares do compilador do LinDCQ*

6.1.1 Gerador de Analisadores Léxicos e Sintáticos

Os analisadores léxicos e sintáticos são as duas primeiras partes do projeto de um compilador. A construção desses dois módulos sem a ajuda de uma ferramenta de auxílio pode levar a erros graves no projeto do compilador, além de ser um esforço desnecessário, pois

existem várias ferramentas com o propósito de auxiliar a construção dos analisadores léxicos e sintáticos.

Desta forma, para o desenvolvimento dos módulos de análise léxica e análise sintática foi utilizado como auxílio o GALS, e um dos principais motivos para a escolha do GALS está relacionado à sua flexibilidade, podendo gerar o código objeto em linguagens como C++, Delphi e Java; assim como o fato de ser uma ferramenta de código fonte aberto, e seu código fonte é liberado sob a licença pública GNU[1] e pode ser obtida facilmente na Internet[2].

A maioria dos geradores pesquisados gera ou o analisador léxico ou o sintático acarretando, algumas vezes, imprecisão na integração dos códigos gerados, e muitas vezes é necessário à utilização de mais de uma ferramenta, pois algumas são voltadas ao analisador léxico e outras ao sintático, e o GALS é uma ferramenta para geração de analisadores tanto léxicos quanto sintáticos, eliminando eventuais esforços de aprender a manipular duas ferramentas diferentes. O GALS foi desenvolvido na linguagem de programação Java (GESSER, 2003).

No GALS, a geração dos analisadores léxicos e sintáticos é realizada através de especificações léxicas, baseadas em expressões regulares, e especificações sintáticas, baseadas em gramáticas livres de contexto, respectivamente. Os seguintes menus estão disponíveis para que o usuário possa utilizá-los:

- Arquivo - permite operações básicas, como abrir, salvar e criar novos arquivos, além de importar especificações sintáticas do GALS;
- Ferramentas - possibilita modificar opções, verificar erros, simular os analisadores e gerar o código dos analisadores léxico e sintático;
- Documentação - exibe as tabelas de análise; e
- Ajuda - ajuda para o usuário.

6.1.2 Criação dos Analisadores Léxicos e Sintáticos

No menu ferramentas do GALS, em opções no painel gerar, o usuário pode escolher se quer gerar o código apenas como: analisador léxico, analisador sintático ou analisador léxico e sintático. Para o desenvolvimento da ferramenta proposta foi escolhida a opção analisador léxico e sintático, com a geração dos analisadores seguindo a técnica de análise LR. É através deste ambiente que o usuário fornece as especificações para a geração dos analisadores. O Código 6.3 a seguir, mostra as variáveis (ou não terminais) utilizados no compilador da gramática da ferramenta:

```
1 //variaveis
2 <programa>
3
4 <operacoes>
5 <mais_operacoes>
6
```

[1]Disponível em: http://www.gnu.org
[2]Disponível em: http://gals.sourceforge.net

```
<tipo_fio>
<fio>
<fio_psi>
<mais_fio>

<sentenca_ini>
<sentenca_fin>

<dupla_ref>
<dupla_tripla_ref>
<tripla_ref>
<quarta_ref>

<lab_porta_2>
<lab_porta_2_3>
<lab_porta_3>
<lab_porta_4>
<porta_2>
<porta_2_3>
<porta_3>
<porta_4>

<ope_logico>
<enquanto>
<mais_enquanto>
<estrut_corpo>

<num>
<mais_num>
<posicao>
```

Código 6.3: *Variáveis do compilador do LinDCQ*

O compilador foi projetado para receber o código que gerará o circuito quântico, escrito em alto nível que esteja de acordo com as regras sintáticas expostas em sua gramática. A função do compilador é fazer toda a análise de código — reportando os possíveis erros ao usuário — e gerar o código correspondente para que possar ser executado e gerado graficamente. O Código 6.4 abaixo, estruturado segundo o padrão BNF, mostra a gramática utilizada para a criação das fases da análise léxica e sintática do compilador:

```
//gramatica BNF
//declara o corpo do programa
<programa>::=<fio> | <fio> <operacoes> | <enquanto> | <fio> <enquanto> |
    <fio> <operacoes> <enquanto> |
<fio> <enquanto> <operacoes> | <fio> <enquanto> <operacoes> <enquanto>;

//tipo fio
<tipo_fio>::=<fio_psi> | OP_fio0 | OP_fio1;

<fio_psi>::=OP_fio OP_par_esq numeros OP_virgula numeros OP_virgula
   numeros OP_virgula numeros OP_par_dir;

//fio
<fio>::=<tipo_fio> palavras OP_col_esq numeros OP_col_dir
   OP_ponto_virgula <mais_fio>;
```

```
<mais_fio>::=<fio> | λ;

<posicao>::=numeros | palavras;

<sentenca_ini>::=OP_par_esq palavras OP_virgula;
<sentenca_fin>::=palavras OP_virgula <posicao> OP_par_dir
   OP_ponto_virgula;

<dupla_ref>::= <sentenca_ini> <posicao> OP_par_dir OP_ponto_virgula;
<tripla_ref>::=<sentenca_ini> <sentenca_fin>;
<quarta_ref>::=<sentenca_ini> palavras OP_virgula <sentenca_fin>;
<dupla_tripla_ref>::=<dupla_ref> | <tripla_ref>;

<operacoes>::= <porta_2> | <porta_2_3> | <porta_3> | <porta_4>;
<mais_operacoes>::= <operacoes> | λ;

//(u, h, m, y e troca)com 2
<porta_2>::=<lab_porta_2> <dupla_ref> <mais_operacoes>;
<lab_porta_2>::= OP_h | OP_m | OP_y | OP_troca;

//(x,z, s e t)com 2 ou 3
<porta_2_3>::=<lab_porta_2_3> <dupla_tripla_ref> <mais_operacoes>;
<lab_porta_2_3>::=OP_x | OP_z | OP_s | OP_t;

//cnot e fredkin
<porta_3>::=<lab_porta_3> <tripla_ref> <mais_operacoes>;
<lab_porta_3>::=OP_cnot | OP_fredkin;

//toffoli
<porta_4>::=<lab_porta_4> <quarta_ref> <mais_operacoes>;
<lab_porta_4>::= OP_toffoli;

<ope_logico>::=OP_menor | OP_maior;

//enquanto
<enquanto>::=OP_enquanto OP_par_esq <posicao> <ope_logico> numeros
   OP_par_dir
<estrut_corpo> OP_fim <mais_enquanto>;
<mais_enquanto>::=<enquanto> | λ;

<estrut_corpo>::=<fio> | <operacoes>;
```

Código 6.4: *Gramática BNF do compilador do LinDCQ*

6.1.3 Criação do Gerador de Código

O compilador faz a análise dos códigos, verificando a existência de erros léxicos e de erros sintáticos, os quais se existirem são exibidos ao usuário. Após tal verificação são iniciadas as atividades do módulo de geração de código, a partir do código lexicamente e sintaticamente correto, faz-se a geração de um código que pode ser simulado pela ferramenta.

O código ilustrado abaixo, contém uma das partes principais do algoritmo utilizado

na geração de código para o compilador, no qual se encontra um laço que percorre todo o código a ser compilado, então esse código é inserido em uma pilha e cada lexema é analisado e convertido em um equivalente para o código final, e assim poderá ser utilizado para simulação na ferramenta. Conforme mencionado anteriormente, após a geração do código realizada com sucesso, uma mensagem é exibida confirmando o sucesso do procedimento.

O trecho de Código 6.5 a seguir, ilustra o algoritmo de geração de código, aplicado a analise do lexema *CNOT*:

```
try {
      while ((tk = lex.nextToken()) != null) {

        if (tk.getLexeme().equals("CNOT")) {
          tk = lex.nextToken();
          tk = lex.nextToken();
          String ini = "" + tk.getLexeme();
          tk = lex.nextToken();
          tk = lex.nextToken();
          String fi = "" + tk.getLexeme();
          tk = lex.nextToken();
          tk = lex.nextToken();
          int pos = Integer.parseInt(tk.getLexeme());
          tk = lex.nextToken();
          tk = lex.nextToken();

          claCNot = new CNot(ini, fi, pos);
          cnots.add(claCNot);
        }
            }
 }catch (Exception e2) {
      e2.printStackTrace();
    }
```

Código 6.5: *Parte da função para geração de código do compilador do LinDCQ*

6.2 Arquitetura do código fonte de LinDCQ

As classes mostradas na Figura 6.1 estão contidas no pacote *analisadorLinDCQ*, o qual contém 12 classes responsáveis por fazer a analise léxica e sintática do código fonte dos circuitos desenvolvidos em LinDCQ, reportando ao usuário os erros que venha a ocorrer em cada tipo de analise.

Para cada uma das portas quânticas, foi criada uma classe[1], com os respectivos atributos e matrizes associadas, ao todo foram criadas 16 classes de portas quânticas e uma classe com as operações relacionadas a tais portas, estas classes estão localizadas no pacote *matrizesOperacoes*. Há um relacionamento de 1 para 1 da classe *Operacoes* com a classe *M* (medida), como é possível ver na Figura 6.2.

[1]Em orientação a objetos, uma classe é uma estrutura que abstrai um conjunto de objetos com características similares. Uma classe define o comportamento de seus objetos através de métodos e os estados possíveis destes objetos através de atributos (DEITEL, 2005).

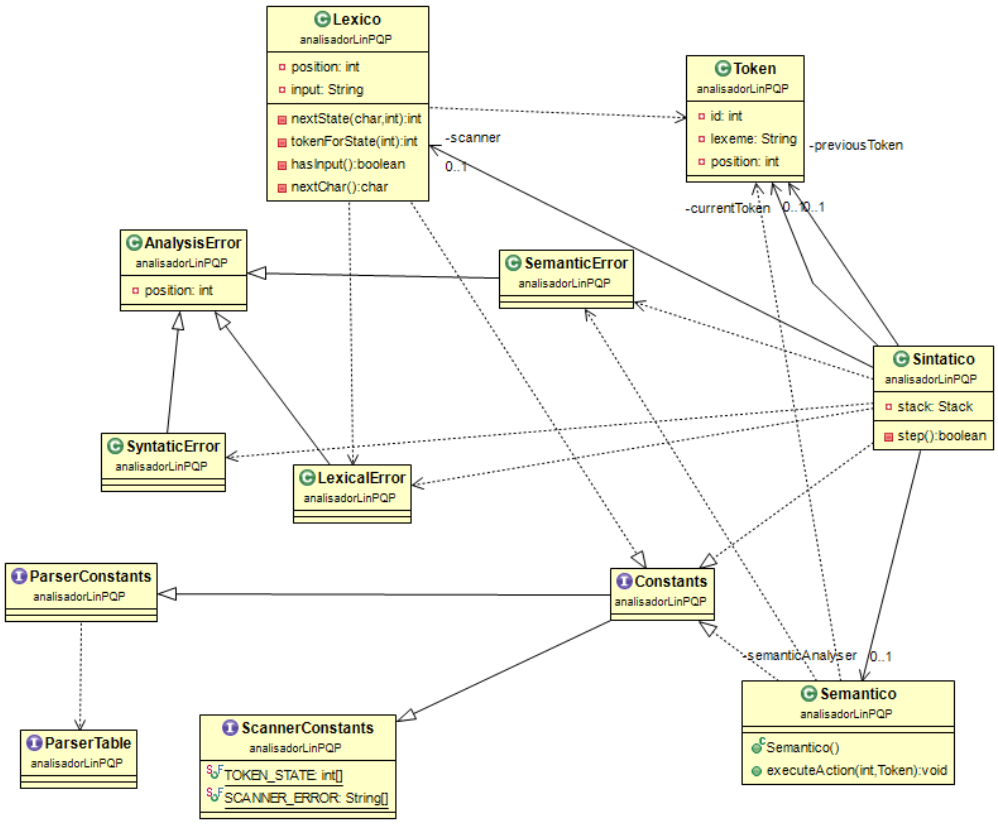


Figura 6.1: *Diagrama de classe dos analisadores*

O pacote responsável por fazer a comunicação com os demais pacotes do código fonte é chamado de *principal*, este pacote pode ser visto na Figura 6.3, tal pacote é de substancial importância para a aplicação. Neste pacote estão presentes as classes responsáveis por:

1. regras de visão;
2. geração de código intermediário;
3. construção da parte gráfica do circuito;
4. chamadas as funções de verificação de erros; e
5. funções de auxílio ao desenvolvimento dos circuitos.

6.3 Funcionalidades do IDE

O IDE do LinDCQ visa facilitar o trabalho do compilador, preparando o código para que este seja executado, o IDE também visa melhorar a usabilidade da ferramenta, a Figura 6.4 ilustra a aparência da ferramenta.

Para uma melhor visualização, os componentes do IDE da Figura 6.4 foram enumerados de 1 a 8. A barra de menu (1), apresenta as opções "Arquivo", "Configurações" e "Ajuda". O menu "Arquivo" possui o submenu "Códigos" que permite acessar as funcionalidades de "Relatório", "Erros", "Compilar", "Passo-a-Passo" e "Salvar". Também possui o submenu

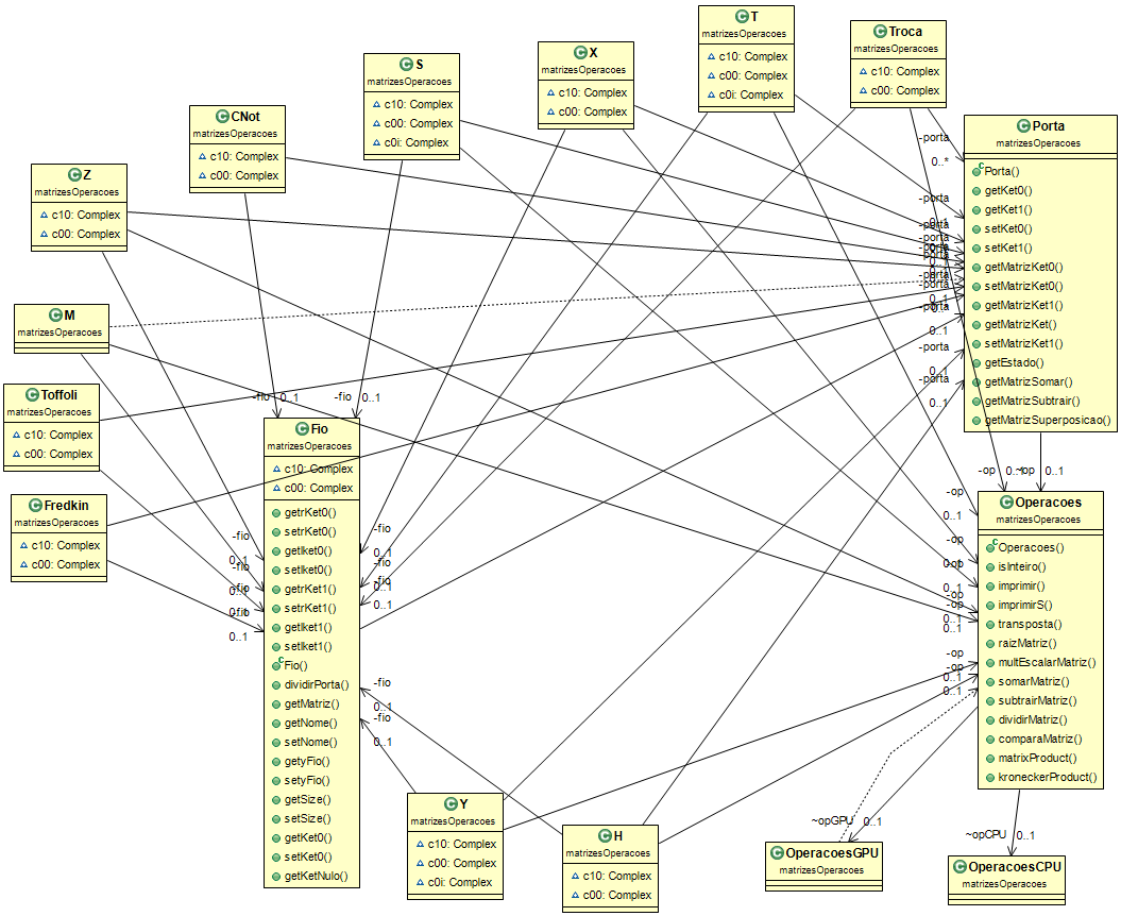


Figura 6.2: *Diagrama de classe das portas quânticas*

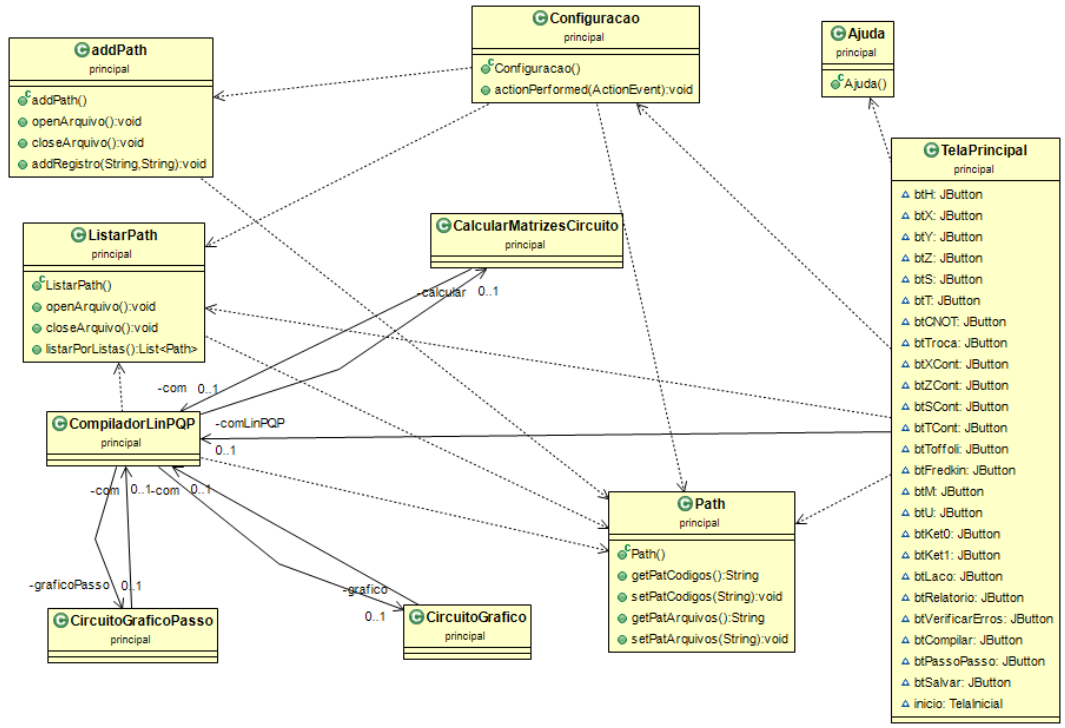


Figura 6.3: *Diagrama de classe do pacote principal*

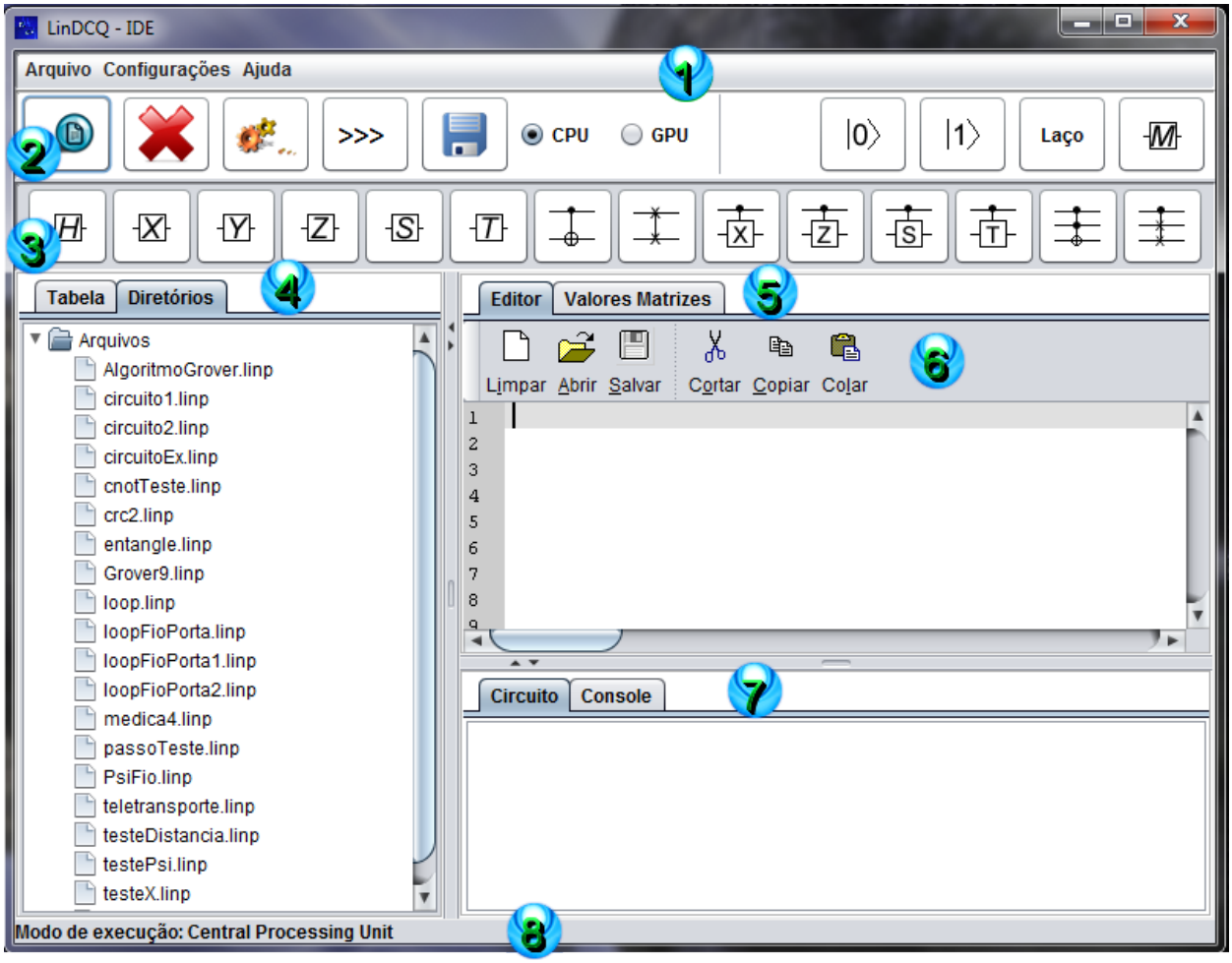


Figura 6.4: *Aparência do IDE*

"Sair" que é utilizado para fechar a aplicação. O submenu "Relatório" exibe uma tabela com todos os *tokens*, *lexemas* e as respectivas posições. O submenu "Erros" realiza a verificação de erros léxicos e sintáticos, e caso algum erro seja encontrado, uma mensagem é disparada ao usuário informando o tipo de erro (léxico ou sintático) e sua posição. O submenu "Compilar" gera o código final que poderá ser salvo utilizando o submenu "Salvar". E o submenu "Passo-a-Passo" executa o circuito desenvolvido passo a passo, como o próprio nome diz, movendo uma linha vertical, com a cor verde, da esquerda para a direta, esse processo pode ser visto na Figura 6.7.

Ainda, no menu "Arquivo", encontram-se:

- "Limpar tabela" - serve para limpar a tabela na qual são mostrados os relatórios de tokens e lexemas, situada no painel (4);
- "Limpar console" - serve para limpar o console que exibe as mensagens em geral de advertência ou de erros, situadas no painel (7);
- "Limpar códigos" - é utilizado para limpar as telas de códigos, ou seja, a tela de edição de códigos (Editor) e a tela de exibição dos cálculos das matrizes do circuito (5);
- "Limpar circuito" - é responsável por limpar o painel onde aparece o circuito graficamente, após o código ser compilado; e

- "Limpar tudo" - é responsável por limpar todos os painéis e as áreas de edição de código.

No menu "Configurações" é possível alterar algumas configurações básicas do IDE, nele são encontrados as opções de cor, fonte, tamanho e alterar path:

- Em "Cor" é aberta uma tela para escolher uma preferência de cor, a ser aplicada no plano de fundo do IDE;
- Em "Fontes" são exibidas as funções de mudar a fonte das palavras do IDE. Estão disponíveis: Arial, Calibre ou Times New Roman;
- "Tamanho" escolhe-se o tamanho das palavras do IDE: 12, 14 ou 16; e
- O botão "Alterar path" é utilizado para alterar o local onde os códigos são salvos ou alterar o local para a estrutura de diretórios exibidas no painel (4).

O menu "Ajuda" possui duas opções, a primeira trata-se da própria "Ajuda" que exibe uma tela com um manual de orientação ao usuário, enquanto que a segunda opção, "Sobre", diz respeito as informações concernentes a ferramenta. O LinDCQ também possui duas barras de ferramentas (uma situada no painel (2) e a outra no painel (3)):

- A barra de botões com as opções de ("Relatório", "Erros", "Compilar", "Passo-a-Passo", "Salvar", "Alternar entre CPU e GPU", "Ket-Zero", "Ket-Um", "Laço" e "Medida") está situada no painel (2). Os primeiros cinco botões têm as mesmas funções dos botões encontrados no menu "Arquivo" do submenu "Códigos", a Figura 6.5 exibe a barra de ferramentas de botões de funcionalidades.

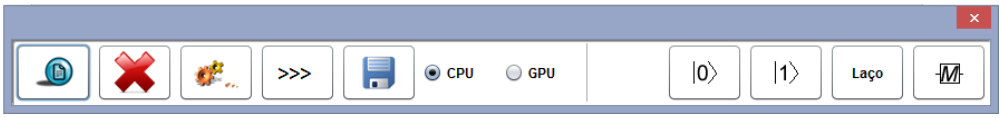


Figura 6.5: *Barra de ferramentas de botões de funcionalidades*

- A barra de botões com as opções de adicionar as portas de um *Qubit* ("Hadamard", "Pauli-X", "Pauli-Y", "Pauli-Z", "Pauli-S" e "Pauli-T"), as de dois *Qubit's* ("Não-Controlado", "Troca", "X-Controlado", "Z-Controlado" e "S-Controlado") e as portas de três *Qubit's* ("Toffoli" e "Fredkin") está situada no painel (3). A Figura 6.6 exibe a barra de ferramentas de botões de portas quânticas.

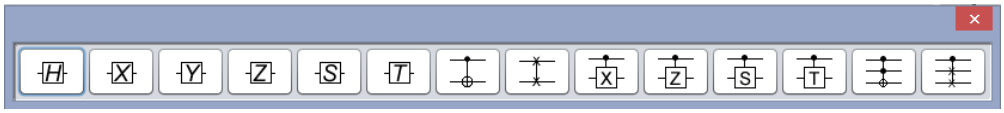


Figura 6.6: *Barra de ferramentas de botões de portas quânticas*

O IDE é dividido da seguinte forma:

- Do lado esquerdo se encontram a aba de "Diretórios" e a aba de "Tabela" (painel (4)). A aba da "Tabela" exibe os Tokens, Lexemas e Posições dos mesmos. A aba de "Diretórios" exibe os arquivos criados, e ao clicar com o botão direito do mouse pode-se: "Abrir arquivo", "Criar arquivo", "Deletar arquivo" e "Salvar arquivo".
- O lado direito é dividido em duas partes:
  - Na parte superior (painel (5)) encontram-se as abas "Editor" e "Valores Matrizes". Em "Editor" está a barra de ferramentas com as opções referentes à área de edição de código: "Novo", "Abrir", "Salvar", "Cortar", "Copiar" e "Colar" painel (6). Em "Valores Matrizes" exibem-se os valores numéricos das matrizes do circuito.
  - Na parte inferior (painel (7)) encontram-se as abas "Circuito" e "Console". "Circuito" é utilizado para exibir a representação gráfica do circuito quântico depois de construído e compilado, já em "Console" são exibidas as mensagens de advertência ou de erros ao usuário.
- Na parte inferior há um painel que informa o estado da ferramente com relação ao modo de execução, o qual pode estar no modo CPU — onde os cálculos são processados na CPU (*host*) — ou no modo GPU — onde os calculos são executados na GPU *device* — dependendo da opção que o usuário escolha.

A Figura 6.7, exibe a tela gráfica de construção dos circuítos, na mesma é possível inserir tanto fios quanto portas quânticas, para isso é necessário posicionar o cursor do mouse na área a qual deseja-se inserir o componente e clicar com o botão direto, para em seguida escolher o componente desejado.

6.4 Exemplos de circuitos desenvolvidos em LinDCQ

Os circuitos desenvolvidas em LinDCQ, utilizam a gramática BNF da ferramenta, descrita na subseção 6.1.1. O Código 6.6 a seguir, exemplifica a derivação do comando — *Fio.ket0 a[2]; H(a,1);* — o qual encontra-se enumerado da linha 1 até a 25.

```
1  <programa>::=<fio> <operacoes>,
2  ::=<fio> <operacoes>,
3  ::=<tipo_fio> palavras OP_col_esq numeros OP_col_dir OP_ponto_virgula <
       mais_fio> <operacoes>,
4  ::=OP_fio0 palavras OP_col_esq numeros OP_col_dir OP_ponto_virgula <
       mais_fio> <operacoes>,
5  ::=Fio.ket0 palavras OP_col_esq numeros OP_col_dir OP_ponto_virgula <
       mais_fio> <operacoes>,
6  ::=Fio.ket0 a OP_col_esq numeros OP_col_dir OP_ponto_virgula <mais_fio>
       <operacoes>,
7  ::=Fio.ket0 a [ numeros OP_col_dir OP_ponto_virgula <mais_fio> <
       operacoes>,
8  ::=Fio.ket0 a [ 2 OP_col_dir OP_ponto_virgula <mais_fio> <operacoes>,
9  ::=Fio.ket0 a [ 2 ] OP_ponto_virgula <mais_fio> <operacoes>,
10 ::=Fio.ket0 a [ 2 ] ; <mais_fio> <operacoes>,
11 ::=Fio.ket0 a [ 2 ] ; <operacoes>,
```

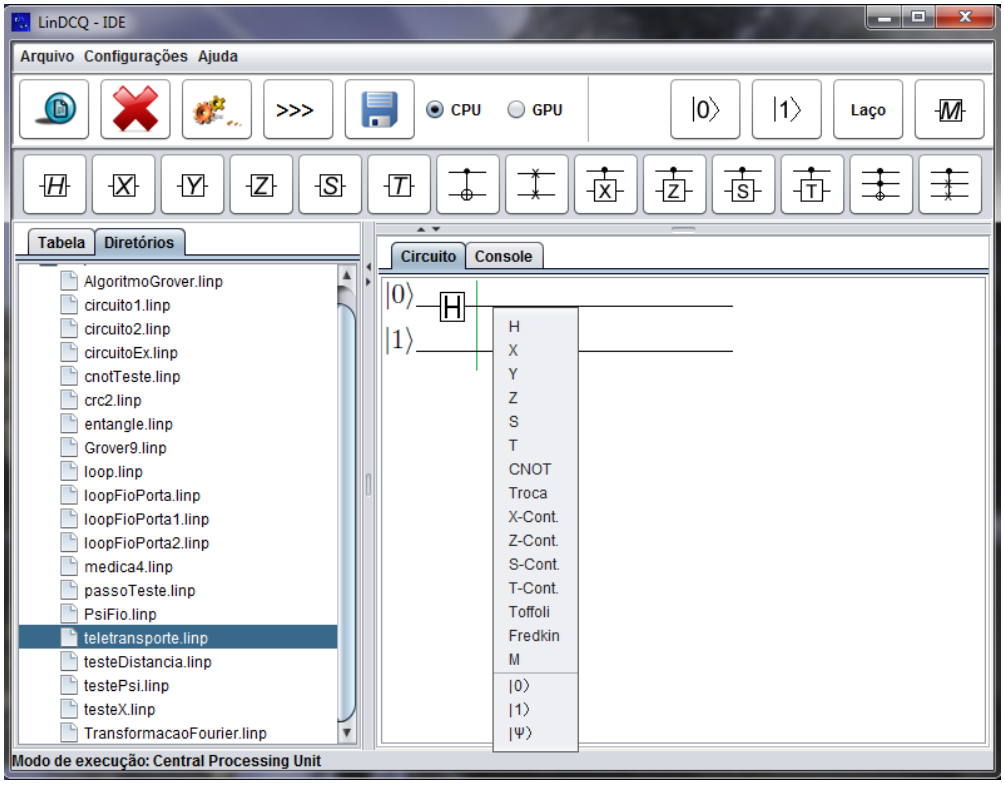


Figura 6.7: *Linha vertical do passo a passo e inserção dos componentes por meio da interface gráfica*

```
12 ::=Fio.ket0 a [ 2 ] ; <porta_2>,
13 ::=Fio.ket0 a [ 2 ] ; <lab_porta_2> <dupla_ref> <mais_operacoes>,
14 ::=Fio.ket0 a [ 2 ] ; OP_h <dupla_ref> <mais_operacoes>,
15 ::=Fio.ket0 a [ 2 ] ; H <dupla_ref> <mais_operacoes>,
16 ::=Fio.ket0 a [ 2 ] ; H <sentenca_ini> <posicao> OP_par_dir
       OP_ponto_virgula <mais_operacoes>,
17 ::=Fio.ket0 a [ 2 ] ; H OP_par_esq palavras OP_virgula <posicao>
       OP_par_dir OP_ponto_virgula <mais_operacoes>,
18 ::=Fio.ket0 a [ 2 ] ; H ( palavras OP_virgula <posicao> OP_par_dir
       OP_ponto_virgula <mais_operacoes>,
19 ::=Fio.ket0 a [ 2 ] ; H ( a OP_virgula <posicao> OP_par_dir
       OP_ponto_virgula <mais_operacoes>,
20 ::=Fio.ket0 a [ 2 ] ; H ( a , <posicao> OP_par_dir OP_ponto_virgula <
       mais_operacoes>,
21 ::=Fio.ket0 a [ 2 ] ; H ( a , numeros OP_par_dir OP_ponto_virgula <
       mais_operacoes>,
22 ::=Fio.ket0 a [ 2 ] ; H ( a , 1 OP_par_dir OP_ponto_virgula <
       mais_operacoes>,
23 ::=Fio.ket0 a [ 2 ] ; H ( a , 1 ) OP_ponto_virgula <mais_operacoes>,
24 ::=Fio.ket0 a [ 2 ] ; H ( a , 1 ) ; <mais_operacoes>,
25 ::=Fio.ket0 a [ 2 ] ; H ( a , 1 ) ;
```

Código 6.6: *Exemplo de derivação*

A Figura 6.8 ilustra a derivação realizada no código anterior na perspectiva de uma árvore de derivação, também chamada de árvore sintática. O símbolo λ encontrado nas derivações representa vazio. As palavras entre os sinais de "<" e ">" que estão com a cor bordô,

são as variáveis ou "não terminais", enquanto que as palavras em azul, são os *tokens* e as que estão com a cor preta são os *lexemas*, também chamados de "terminais" ou "folhas". Como discutido na subseção 3.4.2 a técnica de análise utilizada na gramática foi a LR, assim a gramática teve uma derivação mais acentuada a direta, como é possível perceber na Figura 6.8.

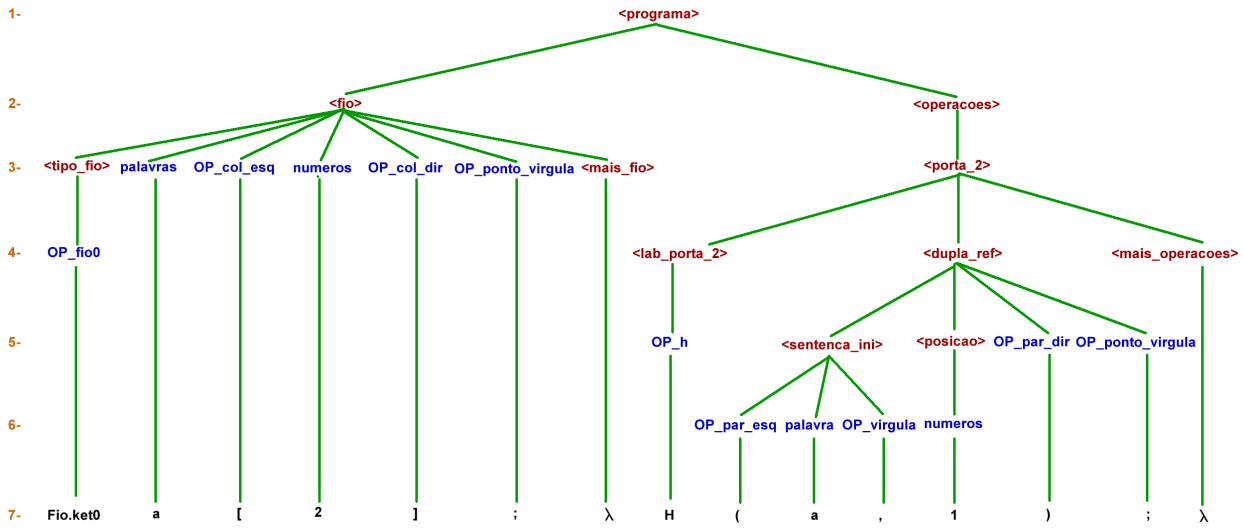


Figura 6.8: *Arvore de derivação*

O Código 6.7 define dois *Qubit's* $|\ 0\rangle$ e $|\ 1\rangle$, respectivamente. A cada um deles está associado um fio de duas posições, como pode ser visto nas linhas 1 e 2. Na linha 4 foi aplicada a porta *CNOT* do fio *a* para o fio *b* na posição 2. No fio *a*, na posição 1, foi aplicada a porta de *Hadamard*, esta operação pode ser vista na linha 5.

```
1 Fio.ket0 a[2];
2 Fio.ket1 b[2];
3
4 CNOT(a,b,2);
5 H(a,1);
```

Código 6.7: *Exemplo de circuito básico desenvolvido em LinDCQ*

O Código 6.8 exibe um fio *a* declarado na linha 1 e outros dez fios declarados dentro de um laço (cinco correspondentes ao *a* e cinco correspondentes ao *aux*), ambos de tamanho 9 respectivamente, o laço da linha 3 enumera estes fios em ordem crescente ponto um número inteiro na frente dos mesmos, por exemplo, o fio *aux* será enumerado da seguinte forma: *aux0*, *aux1*, ..., *auxN*. No último laço linha 8, são declaradas duas portas, onde a porta *Pauli-H* é aplicada no fio *aux* que está dentro do laço, em seguida aplica-se a porta *CNOT* do fio *a* para o fio *aux*.

```
1 Fio.ket0 a[9];
2
3 Enquanto(aux<5)
4   Fio.ket0 a[9];
5   Fio.ket1 aux[9];
```

```
6  Fim
7
8  Enquanto(aux<5)
9    H(aux,1);
10   CNOT(a,aux,6);
11 Fim
```

Código 6.8: *Exemplo de circuito com laço desenvolvido em LinDCQ*

No Código 6.9 seguinte, depois de definir dois *Qubit's* associados com os fios a e b, foi aplicada a porta *Pauli-T* no fio b na posição 1, e a porta *CNOT* do fio a para o b na posição 2. A Figura 6.9 mostra a representação gráfica deste circuito na ferramenta *LinDCQ* e a Figura 6.10 mostra os cálculos que foram realizados durante a execução do circuito. Os cálculos foram inseridos em uma estrutura de dados, exibidos seguindo a notação de Dirac e, ao final, é mostrado o *Kronecker Product* do estado quântico completo.

```
1 Fio.ket0 a[3];
2 Fio.ket1 b[3];
3
4 T(b,1);
5 CNOT(a,b,2);
```

Código 6.9: *Exemplo de circuito com a porta Pauli-T*

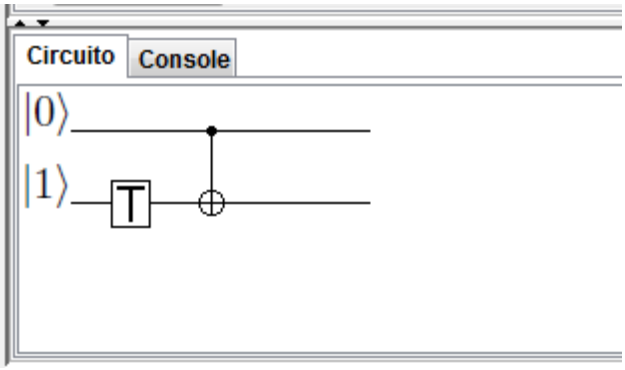


Figura 6.9: *Exemplo do circuito*

No Código 6.10, é exibido um exemplo de medição em circuito, nas linhas 1 e 2 são declarados dois fios de tamanho 3. Na posição 1 do fio b aplica-se, a porta de Hadamard, enquanto que na segunda e terceira posição, respectivamente, dos fios a e b ocorre uma medição, como é possível ver nas linhas 5 e 6.

```
1 Fio.ket0 a[3];
2 Fio.ket1 b[3];
3
4 H(b,1);
5 M(a,2);
6 M(b,3);
```

Código 6.10: *Exemplo de Medição em circuito*

Para analisar o resultado da medição é preciso selecionar a aba "Console", como é possível ver na Figura 6.11 primeiro é informado o nome e a posição, entre parentes, do fio que ocorreu

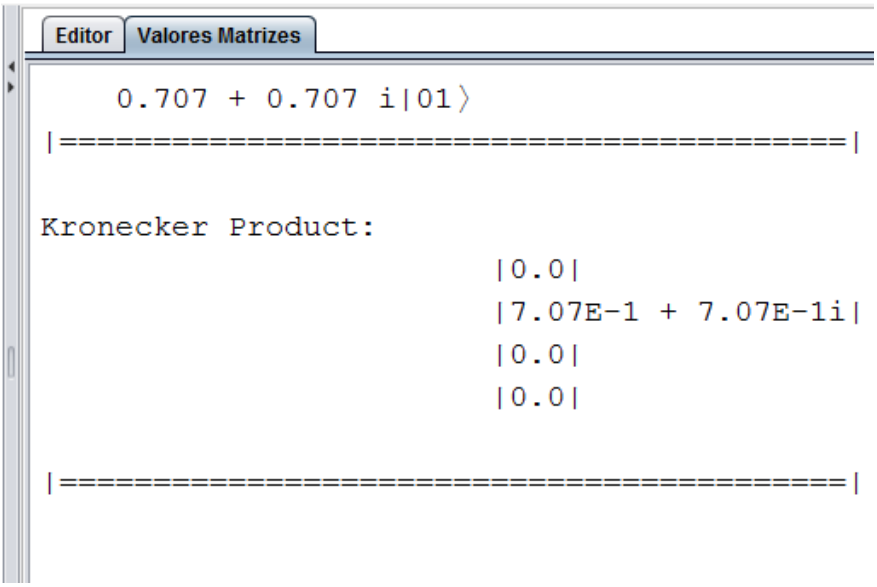


Figura 6.10: *Cálculos do circuito*

a medição, logo em seguida é informado o valor do resultado obtido, seguido da probabilidade obtida, após realizar a medição.

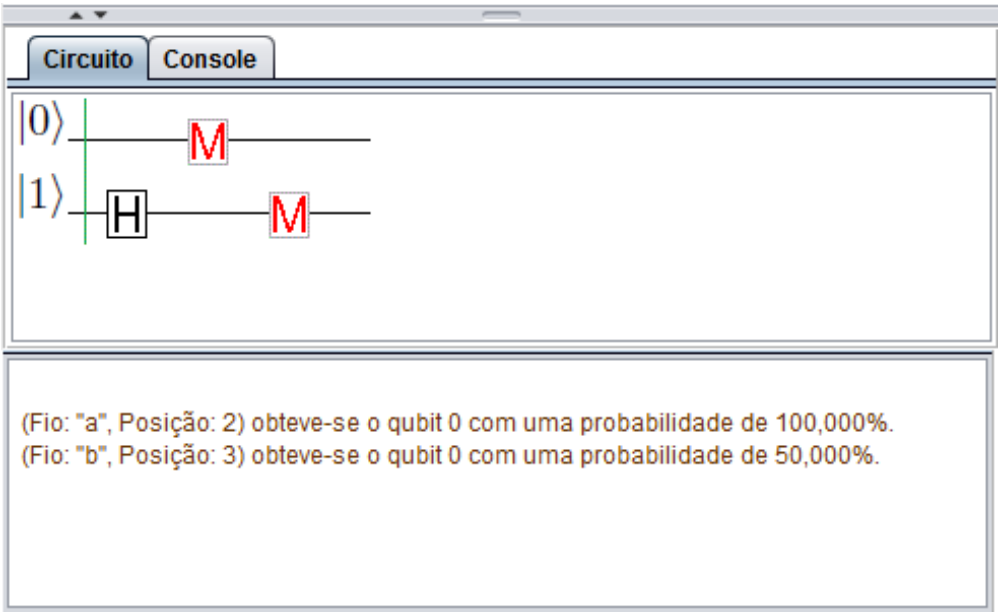


Figura 6.11: *Exemplo de Medição em circuito*

No Código 6.11, é exibido um exemplo de dois *qubit's* ambos em sobreposição, nas linhas 1 e 2 são declarados dois fios de tamanho 1, os dois primeiros parâmetros do fio são referentes ao *ket* zero e os dois últimos ao *ket* um, sendo primeiro inserido o valor real e depois o imaginário, respectivamente.

```
1 Fio.ket(0.552,0,0.834,0) a[1];
2 Fio.ket(0.707,0,0.707,0) b[1];
```

Código 6.11: *Exemplo de qubit's em sobreposição*

Na Figura 6.12 é mostrada a interface para inserir os valores do *qubit*. Primeiro é inserido o valor real e depois o valor imaginário, antes do simbolo de cada *ket*, logo em seguida basta clicar em "Salvar" para confirmar a operação.

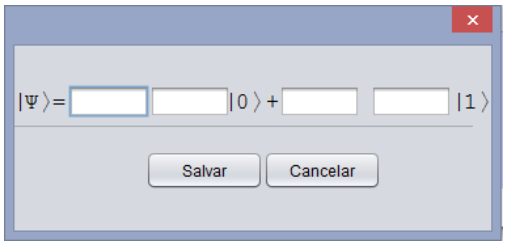


Figura 6.12: *Tela de inserir qubit's em sobreposição*

6.4.1 Teletransporte

O Código 6.12 ilustra uma aplicação desenvolvida na ferramenta LinDCQ. Da linha 1 até a linha 3 são declarados 3 três fios de tamanho 6, cada fio recebe um nome para ser referenciado (nesta aplicação foram nomeados de a, b e c), antecedido da palavra reservada Fio. Da linha 5 até a linha 10 foram declaradas as operações a serem realizadas, por exemplo, na porta $CNOT(a,\ b,\ 1)$; indica que existe uma ligação do $Fio\ a$ para o $Fio\ b$ na posição 1. Essa ideia é igual para as demais operações, deste modo, o restante do procedimento é intuitivo. Este código é referente ao circuito para o teletransporte quântico.

```
1  Fio.ket0 a[6];
2  Fio.ket1 b[6];
3  Fio.ket1 c[6];
4
5  CNOT(a,b,1);
6  H(a,2);
7  M(b,3);
8  M(a,4);
9  X(b,c,5);
10 Z(a,c,6);
```

Código 6.12: *Circuito do Teletransporte quântico*

Na Figura 6.13 do lado esquerdo encontra-se os lexemas da aplicação depois de compilada e do lado direito, na parte inferior, o circuito representado graficamente, após também ter sido compilado, o circuito desta imagem é referente ao Código 6.12 da aplicação do exemplo anterior.

6.4.2 Algoritmo de Grover

Proposto em 1996, o algoritmo de (Grover, 1996) é utilizado para pesquisa em listas ou base de dados não ordenados, mas não se limita, apenas, a este objetivo. Podendo ser também aplicado à grande maioria dos algoritmos clássicos que utilizam algum tipo de busca, conseguindo, deste modo, um aumento de eficiência quadrática $O(\sqrt{n})$.

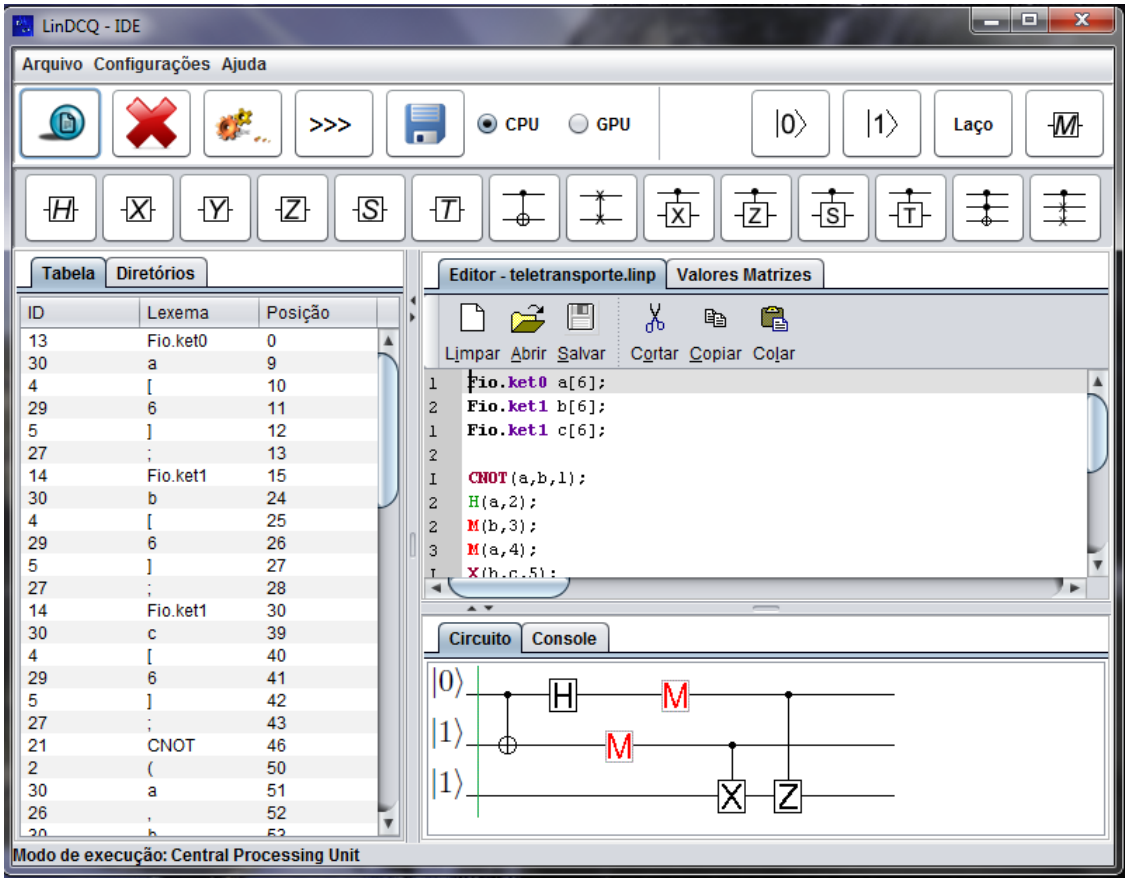


Figura 6.13: *Circuito gráfico gerado a partir do código*

É importante salientar que este algoritmo é probabilístico, isto é, determina o resultado correto com uma determinada probabilidade de sucesso, esta probabilidade pode ser melhorada com sucessivas iterações.

O Código 6.13 corresponde ao algoritmo de Grover:

```
1  Fio.ket0 a[16];
2  Fio.ket0 b[16];
3  Fio.ket1 c[16];
4
5  H(a,1); H(b,2); H(c,3);
6  Toffoli(a,b,c,4);
7  H(a,5); X(a,6);
8  H(b,7); X(b,8);
9  H(b,9); CNOT(a,b,10); H(b,11);
10 X(a,12);X(b,13);
11 H(a,14);H(b,15);H(c,16);
```

Código 6.13: *Circuito do Algoritmo de Grover*

Como é possível ver no código anterior, utilizaram-se três fios, a porta *Toffoli* aplicada na linha 6, funciona como uma função Oráculo. A Figura 6.14 exemplifica o circuito quântico do algoritmo gerado graficamente.

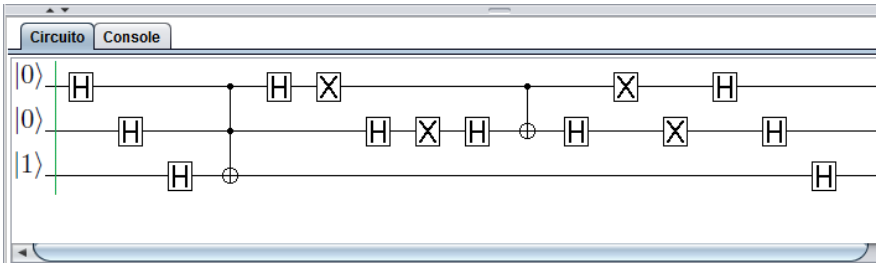


Figura 6.14: *Circuito gráfico do Algoritmo de Grover*

6.4.3 Transformada de Fourier

Publicado em 1994, o algoritmo de (Shor, 1994b) é o mais famoso algoritmo quântico da atualidade, em virtude de sua capacidade de fatoração de números de forma eficiente, este algoritmo pode ser executado classicamente, no entanto, encontrar o período r continua a ser um problema que evolui exponencialmente em n. Mas utilizando a transformada de Fourier quântica, que pode ser implementada eficientemente na computação quântica, é possível encontrar o período r em tempo polinomial e consequentemente executar a fatoração de um número n de forma eficiente.

Enquanto um algoritmo convencional leva uma complexidade de tempo exponencial para fatorar um número, o algoritmo de Shor cresce como um polinômio, ou seja, ele tem complexidade $log(n)$. O código 6.14 gera o circuito quântico que simula a transformada de Fourier quântica.

```
1  Fio.ket0 a[10];
2  Fio.ket1 b[10];
3  Fio.ket0 c[10];
4
5  H(a,1);
6  S(b,a,2);
7  T(c,a,3);
8  H(b,4);
9  S(c,b,5);
10 H(c,6);
11
12 Troca(a,7);
13 Troca(b,8);
14 Troca(a,9);
```

Código 6.14: *Circuito da Transformada de Fourier*

A transformada de Fourier quântica é o elemento chave para a fatoração e também para vários outros algoritmos quânticos. A Figura 6.15 ilustra o circuito da transformada de Fourier quântica graficamente:

A Figura 6.16 mostra o resultado após a aplicação da transformada de Fourier, como a mesma utiliza portas quânticas que operam com números complexos, os resultados foram exibidos seguindo a notação convencional para números complexos, separando a parte real da parte imaginaria pelo sinal de mais, com a parte real ficando do lado esquerdo e a parte imaginária do lado direito.

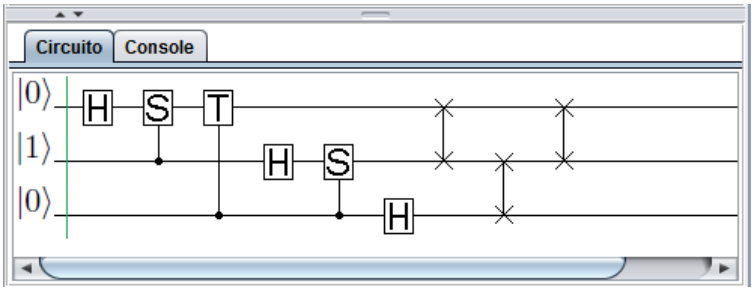


Figura 6.15: *Circuito gráfico da Transformada de Fourier*

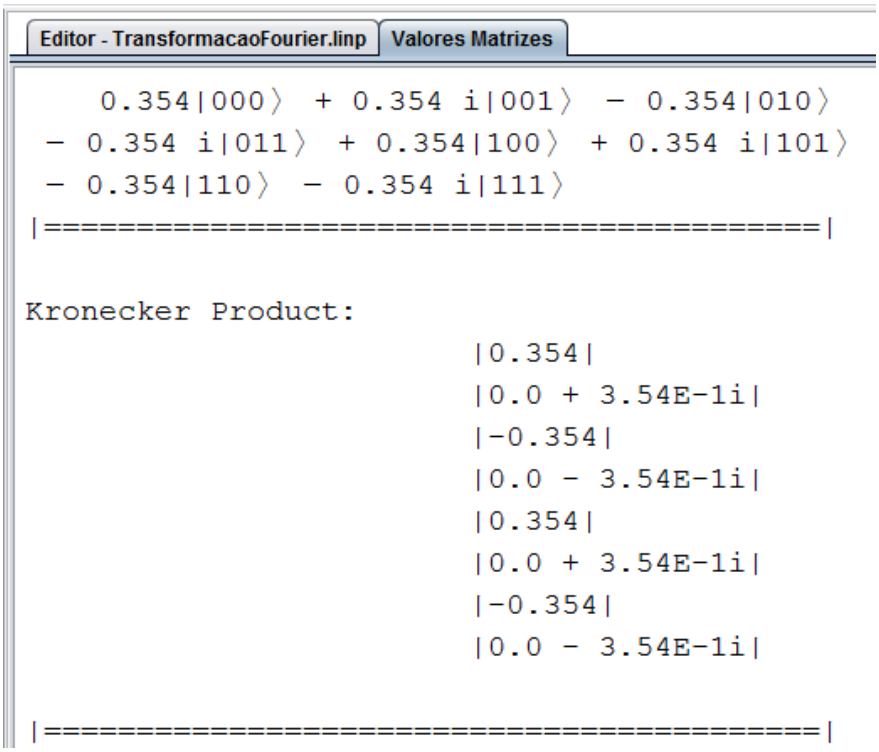


Figura 6.16: *Resultado da Transformada de Fourier*

6.5 Fluxo de um circuitos desenvolvidos em LinDCQ

A Figura 6.17 evidência o fluxo básico de um circuito desenvolvido na ferramenta *LinDCQ*. O código fonte é inserido no editor de códigos do IDE, após esse código ser obtido à função de análise léxica é chamada. Se não ocorrerem erros, então a função de análise sintática é chamada, caso não ocorram erros após a análise sintática então chama-se a função de geração de código, durante essas etapas, caso ocorram erros, eles são tratados e exibidos ao usuário. Após a fase de geração de código completar sua tarefa, sem a incidência de erros, o código processado é armazenado em uma estrutura de dados, a partir da qual pode-se montar o circuito graficamente para ser exibido ao usuário e calcular os valores das matrizes do circuito.

Durante a etapa de calcular os valores das matrizes o usuário tem duas opções, ou ele calcula os valores na CPU, ou calcula os valores na GPU, dependendo das configuração do seu computador, no final os valores calculados são exibidos ao usuário. O diagrama da Figura 6.17, exibe todo esse processo em detalhes, existem várias outras etapas especificas no processamento do circuito, mas as discutidas neste tópico são as mais importantes para

o entendimento deste processo.

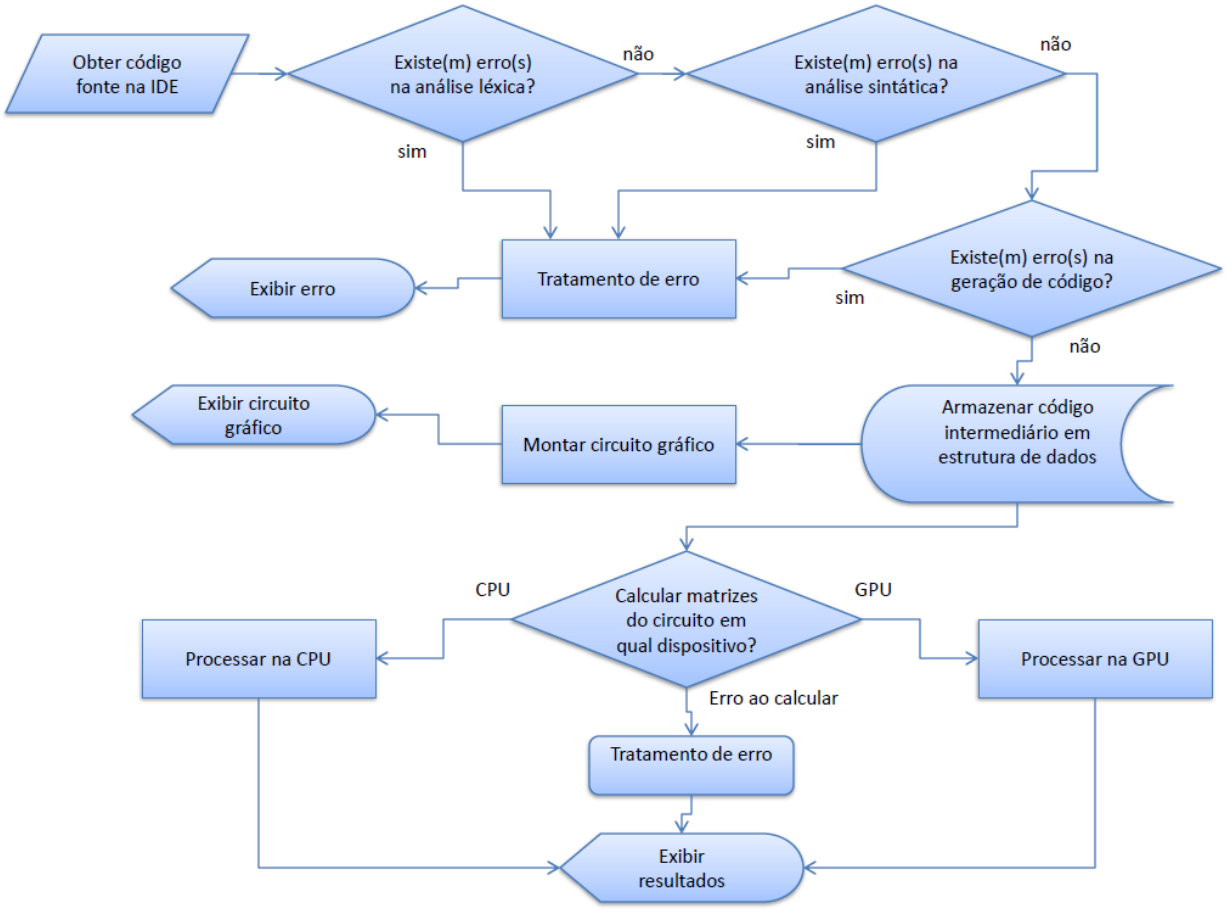


Figura 6.17: *Fluxo de um circuito executado em LinDCQ*

6.6 Comparação do desempenho na CPU e na GPU

6.6.1 Especificação do Hardware

Para a implementação da comparação de desempenho foi utilizado o IDE (*Integrated Development Environment*): do LinDCQ v 1.0, as características do computador, onde as simulações foram realizadas, são as seguintes:

- Processador: AMD Athlon(tm) II X2 270 com 3.40 GHz;
- Memória RAM : 4 Gb DDR3;
- Placa de vídeo: NVidia GeForce GTX 650 1 Gb de RAM DDR5 CUDA 3.0;
- Sistema operacional: Ubuntu 12.04 arquitetura 32bits.

Durante este trabalho, quando a palavra *device* for citada é, apenas, uma referência a placa de vídeo (GPU); e quando a palavra citada for *host* está se referenciando ao processador (CPU).

6.6.2 Comparação do desempenho

Para fazer a comparação dos desempenhos dos circuitos construídos, na CPU e na GPU, foi necessário implementar um código que medisse o tempo de execução de tais circuitos, pertencente ao pacote *System.currentTimeMillis()* do Java.

As comparações foram realizadas pondo-se os estados de alguns *Qubit's* em sobreposição com a porta quântica de *Hadamard*. A Figura 6.18 exibe o gráfico relacionado a comparação do desempenho, no qual, no eixo horizontal é exibido o número de *Qubit's* e no eixo vertical o tempo decorrido em segundos. No gráfico a linha com a cor azul corresponde ao número de *Qubit's*, a linha com a cor bordo ao tempo decorrido no *host* e a linha com a cor verde ao tempo de corrido no *device*. Como é possível ver, houve um ganho de desempenho com relação a execução no *device*.

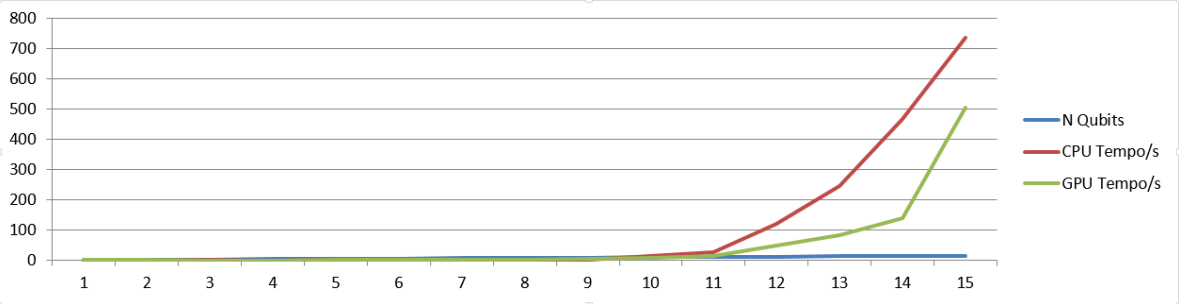


Figura 6.18: *Desempenho CPU X GPU*

6.7 Resumo do Capítulo

Nesta Seção foi explicada a arquitetura do compilador do LinDCQ e as etapas envolvidas em sua construção. Foi apresentado o GALS, uma ferramenta utilizada para gerar o analisador léxico e sintático do compilador, explanando-se também com relação à criação da etapa de geração de código e alguns diagramas de classes dos pacotes de códigos desenvolvidos.

Também foram descritas as funcionalidades do IDE e exibidas suas principais características e funcionalidades. Conseguinte, foram demonstrados alguns exemplos de circuitos quânticos desenvolvidos em LinDCQ. E ao final do capítulo, foi exposto o fluxo do código fonte de um circuito que é submetido à ferramenta do início até o final.

Capítulo 7

Validação da Ferramenta LinDCQ

Neste capítulo apresenta-se a validação da ferramenta LinDCQ. Na Seção 7.1 apresenta-se a justificativa quanto à necessidade de validar a ferramenta. Na Seção 7.2 apresentam-se as técnicas de avaliação de usabilidade. Na Seção 7.3 é apresentada uma discussão sobre a construção do questionário de avaliação da ferramenta. Na Seção 7.4 é explanado sobre os resultados da aplicação da ferramenta a partir de testes estatísticos. A Seção 7.5 é um resumo do capítulo.

7.1 Introdução

A experimentação pode ser considerada o centro do processo científico. Somente experimentos verificam as teorias, oferecendo um modo sistemático, disciplinado, computável e controlado para avaliação da atividade humana. É importante salientar que nenhum experimento oferece prova com certeza absoluta. Os experimentos verificam a previsão teórica de encontro à realidade.

Outro fato importante de ser frisado é que para realizar uma aferição da qualidade de uso, não se pode utilizar uma metodologia genérica que sirva para todos os tipos de públicos, uma vez que os resultados obtidos de tal avaliação não serão precisos. A avaliação da usabilidade de um software está relacionada a uma séria de variáveis, como, por exemplo, ao tipo de aplicação em questão, perfil dos usuários, contexto de utilização e outros (Winckler e Pimenta , 2002). A Norma ISO (1998) define usabilidade como "a capacidade de um produto ser usado por usuários específicos para atingir objetivos específicos com eficácia e satisfação em um contexto especifico de uso".

Existem algumas soluções disponibilizadas para mensuração da qualidade de uso, como as metodologias de (Schilling, 2009), (Santos, 2007) e (Oliveira Jr, 2007), além dos questionários de satisfação de usuário já desenvolvidos, mas que não levam em consideração os aspectos pedagógicos nos softwares (Abreu, 2010). Nesta dissertação são utilizados os seguintes aspectos pedagógicos: Usabilidade geral e usabilidade pedagógica. Tais aspectos são discutidos na Seção 7.2.

7.2 Técnicas de avaliação de usabilidade

Apesar da existência de técnicas de avaliação de usabilidade, uma das classificações utilizadas é quanto ao seu objetivo. Para esta classificação, existem três classes distintas de técnicas para avaliação de usabilidade (Cybis, 2003). No entanto, para a avaliação da usabilidade da ferramenta descrita neste trabalho, serão utilizadas apenas às técnicas de avaliação interpretativa e prospectiva. A terceira técnica de avaliação de usabilidade (a diagnóstica) foi descartada pelo fato de ser necessário a existência de um especialista na área de IHC (Interação Homem Computador), e não se pôde encontrar nenhum disponível.

- Técnicas Objetivas (ou interpretativas) - O avaliador faz uma simulação de uso do aplicativo com os usuários finais, monitorando-os e logo após os dados coletados devem ser interpretados. Pode-se citar como exemplo: o teste Empírico Tradicional - consiste em observar e monitorar a interação do usuário com o sistema, em um ambiente parcialmente controlado (normalmente em um laboratório), através da execução de uma bateria de atividades de determinada funcionalidade da aplicação (Prates e Barbosa , 2003);
- Técnicas Prospectivas - Nesta técnica é realizada uma prospecção das opiniões subjetivas dos usuários, baseadas na aplicação de questionários ou entrevistas com o usuário para avaliar sua satisfação em relação ao sistema e sua operação. Sua dificuldade está relacionada às taxas de devolução, cerca de 30% de retorno. Os questionários de satisfação dos usuários são úteis na avaliação da interação entre o usuário e a aplicação, permitindo conhecer as experiências, opiniões e preferências dos usuários, coletando informações sobre a qualidade da interface. Tais dados são tão (ou mais) importantes do que o desempenho do sistema, e só podem ser obtidas perguntando aos usuários (Winckler e Pimenta, 2002); e
- Técnicas Preditivas (ou diagnósticas) - Inspeções realizadas através de especialistas na interface, destinando-se a prover problemas que os usuários tenham.

Dentre as técnicas mencionadas para avaliar a usabilidade, existe um atrativo na utilização de questionário como ferramenta para avaliação: os achados produzidos pela investigação, independentemente dos sistemas, usuários e tarefas considerados, são passíveis de descrição e análise estatística (Nielsen, 1994). Isto torna tal técnica uma ferramenta para sondagem de aplicação rápida, o que reduz, por conseguinte, os custos envolvidos com a administração e computação dos resultados.

7.3 Questionário e planejamento do estudo

O questionário desenvolvido para avaliar o grau de usabilidade da ferramenta apresentada nesta dissertação teve como base os trabalhos de (Abreu, 2010) e (ISO-9241-10, 2010).

O SAUSP (Sistema Avaliador de Usabilidade em Softwares Pedagógicos) objetiva auxiliar professores a avaliar softwares educacionais a serem utilizados como ferramenta de apoio à aprendizagem de alunos nos laboratórios de informática de escolas. Destina-se a oferecer um mecanismo a professores de laboratórios de informática que necessitem mensurar a qualidade

de uso de materiais de aprendizagem, levando em consideração os fatores pedagógicos e técnicos da usabilidade (Abreu, 2010).

A ISONORM é uma norma internacionalmente reconhecida (Cybis, 2003) que provê questões relativas à usabilidade geral, tais questões estão relacionadas em sete grupos, conforme os princípios de diálogo da parte 10 da norma (ISO-9241-10, 2010), conforme explana (Rodrigues, 2010). Ainda, segundo Nokelainen (2006) seus princípios são:

- **Adequação à tarefa** - uma interface é adequada para uma tarefa quando dá apoio ao usuário de modo que a tarefa seja concluída de forma efetiva e eficiente;
- **Autodescrição** - quando os passos do diálogo na interface são imediatamente compreensíveis por meio de resposta do sistema ou é explicado, sob demanda, ao usuário;
- **Controlabilidade** - permite que o usuário inicie e controle a direção e o ritmo da interação até seu objetivo ser atingido;
- **Conformidade com as expectativas do usuário** - a interface é consistente e corresponde às características do usuário (conhecimento da tarefa, educação e experiência) e às convenções por ele normalmente aceitas;
- **Tolerância a erros** - apesar dos erros de entrada do usuário, o resultado da aplicação esperado pode ser obtido com pouca ou nenhuma ação corretiva do usuário;
- **Adequação à individualização** - capacidade de modificação para adequação das necessidades da tarefa, preferências individuais e habilidades do usuário; e
- **Adequação ao aprendizado** - orienta e dá apoio ao usuário para as estratégias em aprender a usar o sistema, permitindo sua compreensão e memorização.

O motivo para escolher este questionário, deve-se ao fato do mesmo refletir fielmente aos princípios da (ISO-9241-10, 2010) e ser de domínio público.

7.3.1 Desenvolvimento do questionário utilizado

Para a avaliação da ferramenta LinDCQ, as perguntas do questionário elaborado foram extraídas dos trabalhos: (i) Sistema Avaliador de Usabilidade em Softwares Pedagógicos (baseado no trabalho de (Abreu, 2010)) e (ii) Usabilidade Geral (baseado nas questões do modelo de questionário (ISO-9241-10, 2010)). Entretanto, eliminaram-se perguntas semelhantes e alteraram-se algumas perguntas visando adequação ao contexto do problema abordado.

Desse modo espera-se minimizar a quantidade de questões, contribuindo para reduzir o tempo de preenchimento do formulário (Mack e Nielsen, 1994), gerando mais atratividade por seus utilizadores e, consequentemente, um maior número de respostas aos questionários (Hollingsed e Novick, 2007).

O questionário desenvolvido nesta proposta adotou como medida de usabilidade a satisfação do usuário, uma vez que juntamente com a eficiência e a efetividade, pela definição

da *International Organization for Standardization* (Rodrigues, 2010), são métricas de usabilidade que permite utilizar um produto alcançando seus objetivos específicos em um contexto de uso específico (ISO-9241-10, 2010).

Visando a organização das respostas, a melhor alternativa para esse tipo de avaliação é dispô-las em escalas, e consequentemente tabulá-las para obtenção de dados quantitativos do nível de satisfação dos usuários (Kotler, 2003); (SANTOS, 2004). O questionário tem o intuito de medir a satisfação do usuário através da aferição da usabilidade da ferreamente e sua tabulação está sob uma escala. As respostas estão dispostas em uma escala (Likert , 1932), com pontuação de 1 a 5, representando as opiniões de Concordo Totalmente (5 pontos) até Discordo Totalmente (1 ponto). O questionário é apresentado no (APÊNDICE B – Questionário de satisfação de uso).

7.3.2 Objetivo global

Definir se a ferramenta LinDCQ oferece um nível de usabilidade agradável para os seus usuários.

7.3.3 Objetivos da medição

Tendo como base as questões relativas à usabilidade geral da ISONORM, conforme os princípios de diálogo da parte 10 da norma (ISO-9241-10, 2010); (Rodrigues, 2010) e as questões de usabilidade pedagógicas referentes ao trabalho de (Abreu, 2010), pretende-se comparar: (i) Qual a aceitação dos usuários que desenvolveram atividades utilizando a ferramenta LinDCQ e (tiveram treinamento para utilizá-la) e (ii) Qual a aceitação dos usuários que desenvolveram atividades utilizando a ferramenta LinDCQ e (não tiveram treinamento para utilizá-la).

7.3.4 Objetivo do estudo

Analisar a usabilidade geral e pedagógica da ferramenta LinDCQ
Com o propósito de comparar
Com respeito às métricas da (ISO-9241-10, 2010) e de (Abreu, 2010)
Do ponto de vista do aprendiz desenvolvedor de circuitos quânticos
No contexto de alunos de curso de Computação.

7.3.5 Definição das hipóteses

Esta seção apresenta sobre como o estudo foi desenvolvido, fornecendo elementos para execução de outros estudos que poderão utilizar o mesmo planejamento para obtenção de resultados. Estudos futuros poderão definir novas variáveis a fim de avaliar a ferramenta LinDCQ.

Em um experimento são levantadas evidências sobre a plausibilidade de uma hipótese idealizada. A principal hipótese é a nula e no estudo tenta-se rejeitar tal hipótese a favor de uma hipótese alternativa.

As hipóteses referentes à usabilidade da ferramenta são:

- **Hipótese nula** - o grau de usabilidade oferecido pela ferramenta **não** é considerado aceitável para os usuários que **não** tiveram treinamento, se comparado com os que tiveram treinamento; e
- **Hipótese da pesquisa** - O grau de usabilidade oferecido pela ferramenta é considerado aceitável para os usuários que **não** tiveram treinamento, se comparado com os que tiveram treinamento.

7.3.6 Seleção dos indivíduos

Para participar do experimento foram selecionados alunos do curso de Bacharelado em Ciência da Computação da Faculdade de Tecnologia do Recife (FATEC-PE). No experimento realizado supõe o processo off-line, uma vez que os alunos não estão sendo entrevistados durante todo o tempo do aprendizado, apenas em certo instante.

7.3.7 Variáveis

São variáveis dependentes:

- As respostas do questionário.

São variáveis independentes:

- Implementação Interativa e Incremental;
- Tecnologias especificas para implementar o estudo: LinDCQ, JRE, JDK, OpenCl; e
- Exercícios solicitados aos alunos: (i) um circuito que coloque dois *qubit's* em superposição; (ii) implementar o circuito anterior e depois adicionar a porta CNOT e fazer uma mediação no primeiro *qubit*; (iii) criação de um circuito que simule o teletransporte quântico.

7.3.8 Descrição da instrumentação

O questionário elaborado (APÊNDICE B – Questionário de satisfação de uso) possui um total de vinte e três perguntas, as quais estão apresentadas na subseção 7.4.1. Tais perguntas foram aplicadas aos indivíduos selecionados para o experimento e estes foram organizados em dois grupos: indivíduos que tiveram treinamento na ferramenta LinDCQ e os que não tiveram treinamento na ferramentas.

Os grupos foram formados a partir de seleção aleatória de indivíduos, os quais foram divididos em dois grupos: o primeiro com 16 indivíduos e o segundo com 15 indivíduos, respectivamente. Cada grupo respondeu aos exercícios descritos na subseção 7.3.7.

É importante salientar que a maioria dos alunos não possuíam conhecimentos em nenhuma tecnologia utilizada para construir circuitos quânticos, mas todos tinham conhecimentos de lógica de programação e programavam em, pelo menos, uma linguagem. Dessa forma, antes da realização do experimento ofertou-se um minicurso de noções básicas de computação quântica.

Para responder o questionário, foram indicadas possíveis respostas. A Tabela 7.1 apresenta a legenda para o significado das possíveis respostas.

Valor	Legenda
1	Discordo totalmente
2	Discordo parcialmente
3	Talvez
4	Concordo parcialmente
5	Concordo totalmente

Tabela 7.1: *Legenda para as escalas*

7.4 Resultados do estudo e aplicação dos testes estatísticos

7.4.1 Resultados do estudo

A Tabela 7.2 apresenta a frequência das respostas do questionário dos indivíduos do experimento, contendo as perguntas elaboradas para a avaliação da usabilidade da ferramenta LinDCQ. Segundo Levin e Fox (2004), as frequências observadas correspondem ao conjunto de respostas obtidas em uma distribuição efetiva, ou seja, do resultado efetivo de uma pesquisa ou um estudo. Nas questões a palavra material refere-se ao tipo de ferramenta que o aluno utilizou durante o experimento, ou seja, LinDCQ e demais programas de apoio.

Nº	Questão	Treinados					Não treinados				
		1	2	3	4	5	1	2	3	4	5
1	Este material possui funções necessárias, disponibilizadas de forma eficiente para as tarefas correspondentes? (Por exemplo: ajuda, configurações, salvar, etc.)	0	0	5	7	4	0	0	9	5	1
2	Sente-se à vontade com as funções deste material, ou somente algumas lhes são familiares?	0	1	3	9	3	3	4	7	1	0
3	Consegue visualizar os recursos disponíveis neste material?	0	0	4	5	7	2	3	3	6	1
4	Entende com facilidade as palavras, nomes, abreviaturas ou símbolos que estão neste material?	0	0	5	7	4	0	0	3	6	6
5	Permite alternar facilmente entre os menus ou telas?	0	0	6	6	4	0	0	5	2	8

6	Não impõe qualquer interrupção desnecessária em seu ritmo de estudo? (Por exemplo, mensagens explicativas só aparecem quando você espera que apareçam.)	0	1	5	5	5	0	2	2	4	7
7	As telas deste material com seus textos, botões e figuras tem formato de fácil reconhecimento?	0	2	1	5	8	2	0	4	8	1
8	Esse material pode ser entendido e usado por qualquer aluno, com pouca ou muita experiência com programação?	0	3	9	2	2	1	2	8	3	1
9	Depois de ambientado ao uso do material, não precisa de esforços para lembrar-se onde encontrar uma informação ou da funcionalidade de um botão?	0	1	4	7	4	0	3	7	4	1
10	Foi fácil aprender a usá-lo? Você não precisou de ajuda depois de entender como o sistema funciona.	1	1	7	5	2	1	3	6	5	0
11	Possui controle sobre as tarefas realizadas? (O sistema não dita a sequência de passos, você tem controle na ordem para finalização de suas tarefas)	1	0	4	5	6	1	0	4	6	4
12	Ao usar este material, sente que ele foi projetado para você? (As tarefas não são consideradas nem fáceis nem difíceis)	0	1	4	7	4	2	1	7	2	3
13	Você teria interesse de aprender os tópicos deste material mais profundamente? Se houvesse mais tempo, gostaria usar mais este material para aprender mais?	0	1	4	6	5	1	1	3	4	6
14	É rápido aprender um novo tópico ou recapitular um tópico anterior neste material?	1	1	5	7	2	2	1	7	4	1
15	A linguagem usada é natural. Os termos, frases, conceitos são similares àqueles usados no dia-a-dia ou no ambiente de estudo?	0	0	2	8	6	1	0	3	5	6
16	Não fico confuso na utilização dos símbolos, ícones e imagens.	0	0	5	6	5	0	4	8	2	1
17	Ao errar a solução de uma tarefa, o programa me envia um aviso amigável.	0	2	5	9	0	0	1	2	5	7
18	Este material de aprendizagem ensina habilidades e conhecimentos que serão necessários no futuro?	3	2	7	3	1	2	0	5	6	2
19	Exibe mensagem de erros cometidos.	0	1	5	2	8	0	0	3	2	10
20	As caixas de diálogo proporcionam informações adequadas ao desempenho das tarefas?	0	0	4	6	6	0	0	2	10	3
21	Os recursos de ajuda são fáceis de usar?	0	3	3	6	4	0	0	6	2	7
22	Este material me encoraja a aprender sobre o assunto abordado?	0	1	4	7	4	0	3	4	3	5
23	Possui interesse no assunto abordado neste material?	0	1	4	8	3	1	3	7	2	2

Tabela 7.2: *Frequência das questões respondidas*

A Tabela 7.3 apresenta os dados do questionário do perfil dos participantes do experimento. Este questionário é apresentado no (APÊNDICE C – Questionário do Perfil do participante).

Número do Participante	**Você já havia utilizado alguma ferramenta para trabalhar com circuitos quânticos**	**Como você classifica sua experiência com programação**	**Ano de ingresso**	**Entrada**
1	não	baixa	2011	2º
2	não	media	2011	2º
3	não	media	2011	2º
4	não	media	2011	2º
5	não	baixa	2011	1º
6	não	media	2011	2º
7	não	baixa	2011	2º
8	não	media	2011	2º
9	sim	baixa	2011	2º
10	não	baixa	2011	2º
11	não	baixa	2011	1º
12	sim	baixa	2012	2º
13	não	media	2012	1º
14	não	baixa	2011	2º
15	não	media	2012	1º
16	não	media	2011	2º
17	não	baixa	2012	2º
18	não	media	2011	2º
19	não	media	2012	2º
20	não	baixa	2012	2º
21	não	media	2012	2º
22	não	alta	2013	2º
23	não	media	2013	2º
24	não	alta	2013	1º
25	não	baixa	2013	1º
26	não	media	2012	1º
27	não	media	2012	2º
28	não	baixa	2013	2º
29	não	alta	2013	1º
30	não	alta	2013	2º
31	não	baixa	2012	1º

Tabela 7.3: *Perfil dos participantes*

7.4.2 Análise e interpretação dos Resultados

Nesta Seção serão realizadas análises do ponto de vista da estatística descritiva e aplicação do teste de *Qui-quadrado*.

7.4.3 Estatística descritiva

Nesta estatística são realizadas medidas de tendência central. Como os valores correspondentes a "Discordo Totalmente", "Discordo Parcialmente", "Talvez", "Concordo parcialmente" e "Concordo totalmente" são de escala ordinal, então é possível definir as métricas media e moda.

Para o grupo que foi treinado com a ferramenta LinDCQ, os dados foram divididos em duas tabelas para permitir uma melhor visualização. A Tabela 7.4 apresenta as questões de 1 a 12 e a Tabela 7.5 apresenta as questões de 13 a 23. Nestas são expostas as medias e modas de cada uma das questões, sendo que na última coluna da Tabela 7.5 são mostradas a media e a moda total de todas as frequências das perguntas do questionário.

Questões com usuários treinados												
Nº	1	2	3	4	5	6	7	8	9	10	11	12
Media	3,94	3,88	4,19	3,94	3,88	3,88	4,19	3,19	3,88	3,38	3,94	3,88
Moda	4,00	4,00	5,00	4,00	4,00	4,00	5,00	3,00	4,00	3,00	5,00	4,00

Tabela 7.4: *Media e Moda das questões*

Questões com usuários treinados												**Total**
Nº	13	14	15	16	17	18	19	20	21	22	23	
Media	3,94	3,50	4,25	4,00	3,44	2,81	4,06	4,13	3,69	3,88	3,81	3,81
Moda	4,00	4,00	4,00	4,00	4,00	3,00	5,00	4,00	4,00	4,00	4,00	4,00

Tabela 7.5: *Media e Moda das questões*

Da mesma forma, para o grupo de indivíduos que não receberam o treinamento na ferramenta LinDCQ, os dados foram divididos em duas tabelas. As Tabelas 7.6 e 7.7 apresentam a moda e media da frequência das respostas.

Questões com usuários não treinados												
Nº	1	2	3	4	5	6	7	8	9	10	11	12
Media	3,47	2,40	3,07	4,20	4,20	4,07	3,40	3,07	3,20	3,00	3,80	3,20
Moda	3,00	3,00	4,00	4,00	5,00	5,00	4,00	3,00	3,00	3,00	4,00	3,00

Tabela 7.6: *Media e Moda das questões*

	Questões com usuários não treinados											Total
Nº	13	14	15	16	17	18	19	20	21	22	23	
Media	3,87	3,07	4,00	3,00	4,20	3,40	4,47	4,07	4,07	3,67	3,07	3,56
Moda	5,00	3,00	5,00	3,00	5,00	4,00	5,00	4,00	5,00	5,00	3,00	3,00

Tabela 7.7: *Media e Moda das questões*

7.4.4 Aplicação dos testes estatísticos

Para os cálculos realizados neste tópico foi utilizado o teste *Qui-quadrado* de dois critérios. Segundo Levin e Fox (2004), esse teste tem a finalidade de testar se uma distribuição de frequências observadas difere significativamente de outra distribuição observada. Com base nos resultados desse teste é que será possível verificar as hipóteses definidas na subseção 7.3.5.

Para o teste *Qui-quadrado* utilizou-se a frequência dos dados das questões respondidas pelos alunos, ou seja, os resultados expressos sob a escala de 1 a 5. A Tabela 7.8 apresenta a frequência das escalas referente à usabilidade.

Métrica	Não Treinados	Treinados	Total
Discordo totalmente	19	6	25
Discordo parcialmente	31	22	53
Talvez	115	105	220
Concordo parcialmente	97	138	235
Concordo totalmente	83	97	180
Total	345	368	713

Tabela 7.8: *Frequência das escalas de usabilidade*

Na Tabela 7.9 calculou-se a frequência esperada para cada célula Tabelas 7.8, uma vez que, essa frequência esperada é essencial para o teste do *Qui-quadrado*. Para a realização

desse cálculo, aplicou-se à seguinte fórmula (Levin e Fox, 2004):

$$f_e = \frac{t_l * t_c}{t_t} \tag{7.1}$$

Onde:

- f_e Frequência esperada
- t_l Soma do valor total da respectiva linha
- t_c Soma do valor total da respectiva coluna
- t_t Soma do valor total da tabela

Por exemplo, para calcular o valor da primeira célula da Tabela 7.9, foi necessário o seguinte procedimento:

$$\begin{aligned} f_e &= \frac{25 * 345}{713} \\ f_e &= 12,097 \end{aligned} \tag{7.2}$$

O procedimento anterior é repetido para as demais células da Tabela 7.9 a qual apresenta os cálculos realizados para obtenção da frequência esperada em cada uma das células.

Métrica	Não Treinados	Treinados
Discordo totalmente	19(12,097)	6(12,903)
Discordo parcialmente	31(25,645)	22(27,355)
Talvez	115(106,452)	105(113,548)
Concordo parcialmente	97(113,710)	138(121,290)
Concordo totalmente	83(87,097)	97(92,903)

Tabela 7.9: *Frequências esperadas de usabilidade*

Tomando a Tabela 7.9 como referência e dispondo todos os seus valores (cálculos finais) nas colunas apropriadas da Tabela 7.10[1].

Para cada célula são reportadas as frequências observadas e esperadas, subtraídas as frequências esperadas das frequências observadas, elevadas as diferenças ao quadrado, dividida pela frequência esperada, e por fim somado esses quocientes, a fim de obter o valor

[1]Legenda: esquerdo = usuários não treinados com LinDCQ; direto = usuários treinados com LinDCQ.

do *Qui-quadrado*.

Métrica	f_0	f_e	$f_0 - f_e$	$(f_0 - f_e)^2$	$\frac{(f_0-f_e)^2}{f_e}$
Discordo totalmente esquerdo	19	12,097	-6,903	47,655	3,939
Discordo totalmente direito	31	25,645	-5,355	28,674	1,118
Discordo parcialmente esquerdo	115	106,452	-8,548	73,075	0,686
Discordo parcialmente direito	97	113,710	16,710	279,213	2,455
Talvez esquerdo	83	87,097	4,097	16,784	0,193
Talvez direito	6	12,903	6,903	47,655	3,693
Concordo parcialmente esquerdo	22	27,355	5,355	28,674	1,048
Concordo parcialmente direito	105	113,548	8,548	73,075	0,644
Concordo totalmente esquerdo	138	121,290	-16,710	279,213	2,302
Concordo totalmente direito	97	92,903	-4,097	16,784	0,181
				X^2	16,260

Tabela 7.10: *Cálculo do Qui-quadrado de usabilidade*

Para determinar o grau de liberdade, aplicou-se a fórmula (Levin e Fox, 2004):

$$gl = (l - 1) * (c - 1) \tag{7.3}$$

Onde:

- gl: Grau de liberdade
- l: Número de linhas
- c: Número de colunas

Fazendo o cálculo temos:

$$gl = (5 - 1) * (2 - 1)$$

$$gl = (4) * (1) \tag{7.4}$$

$$gl = 4$$

Comparando o valor do Qui-quadrado obtido com o valor apropriado do Qui-quadrado (ANEXO A – Tabela Qui-quadrado), tem-se:

- Valor obtido de usabilidade: $X^2 = 16,260$

- Tabelado: $X^2 = 9,488$

Para rejeitar-se uma hipótese nula ao nível de significância 0,05 com 4 graus de liberdade, o valor calculado do *Qui-quadrado* deve ser no mínimo 9,488. Como se obtive um valor de 16,260 para o cálculo do *Qui-quadrado* referente à usabilidade, então a hipótese nula é rejeitada e é aceita a hipótese alternativa.

O resultado sugere que o grau de usabilidade oferecido pela ferramenta é considerado aceitável para os usuários que não tiveram treinamento, comparando estes com os que tiveram o treinamento para usar LinDCQ. Deste modo, a ferramenta mostrou-se eficiente com um resultado positivo para o teste do *Qui-quadrado*, neste contexto de uso específico.

7.5 Resumo do capítulo

Neste capítulo foram explicadas as metodologias utilizadas para a aferição do grau de usabilidade da ferramenta LinDCQ proposta neste trabalho. Foi discutido sobre a validação da ferramenta com a aplicação de alguns testes estatísticos. Também foram apresentados os resultados juntamente com uma analise dos dados coletados no experimento.

Capítulo 8

Conclusões

Neste capítulo apresentam-se as considerações finais sobre o trabalho desenvolvido nesta dissertação. Na Seção 8.1 apresentam-se as considerações finais deste trabalho. Na Seção 8.2 descrevem-se as contribuições desta dissertação e na Seção 8.3 algumas propostas para trabalhos futuros.

8.1 Considerações Finais

Este trabalho explana sobre o desenvolvimento de uma ferramenta para a programação de circuitos quânticos que podem ser simulados tanto na CPU quanto na GPU. Dependendo, apenas das configurações realizadas pelo usuário. Com a conclusão do trabalho, espera-se que o LinDCQ seja uma ferramenta ideal para quem deseja construir circuitos que demandam enorme custo computacional, uma vez que, os cálculos podem ser realizados na GPU. Os principais componentes desenvolvidos para a ferramenta LinDCQ foram: o IDE (para facilitar o trabalho na programação dos circuitos); um compilador; um módulo para geração do circuito gráfico; um modulo para calcular o circuito com números complexos; e os algoritmos paralelos em LinDCQ, utilizando JOCL para simulação na GPU.

Vale ressaltar que novos experimentos precisam ser realizados visando constatar o resultado da análise realizada, uma vez que o resultado obtido é dependente das métricas utilizadas, ou seja, da (ISO-9241-10, 2010) e as de (Abreu, 2010). Sendo assim, a utilização de novas métricas e novos indivíduos poderá apresentar novos resultados, que servirão de base para novas propostas quanto à melhoria de LinDCQ.

No que concerne à utilização da ferramenta, pode-se dizer que LinDCQ é satisfatória, tento em vista o sucesso de utilização no experimento realizado. E que suas funcionalidades quanto à usabilidade podem ser consideradas, a princípio, eficientes. Os aspectos relacionados ao desenvolvimento de LinDCQ não apresentaram falhas durante a utilização da ferramenta. Dessa forma, pode-se dizer que tanto a gramática definida para a ferramenta, quanto o seu compilador são funcionais.

8.2 Contribuições deste trabalho

Com a concretização dos objetivos explanados na Seção 1.2, pode-se concluir o desenvolvimento de LinDCQ, almeja-se que LinDCQ se torne uma ferramenta ainda mais valiosa, suas principais contribuições destacam-se nos contextos da pesquisa e ensino. No caso da pesquisa, a ferramenta deverá facilitar o desenvolvimento de algoritmos, simplificando a construção dos mesmos e oferecendo um modo rápida de executá-los na GPU. No contexto de ensino, tanto professores quanto alunos deverão se beneficiar, tendo em vista que, os professores terão mais uma ferramenta didática que os ajudará a ilustrar os conceitos trabalhados em sala e os alunos poderão utilizar a ferramenta para norteá-los na solução de exercícios ou na verificação de resultados alcançados.

A gramática proposta poderá servir de base ou ser reutilizada em outras ferramentas ou aplicações destinadas a descrição de circuitos quânticos. A validação da ferramenta, apresentada na Seção 7, poderá servir de modelo para novos experimentos envolvendo a usabilidade de ferramentas tanto no contexto geral, quanto educacional.

8.3 Proposta para Trabalhos Futuros

Como trabalho futuro espera-se, ampliar a gramática da ferramenta para que, assim, o compilador possa aceitar estruturas de código mais complexas e realizar algumas modificações na ferramenta para permitir que a mesma aceite:

- Estruturas de controle mais sofisticadas;
- Criação de funções;
- Chamadas recursivas; e
- Otimização dos cálculos do circuitos, como, por exemplo, tratar matrizes esparsas.

Ainda, diante do resultado obtido referente a usabilidade, novos experimentos deverão ser realizados, tanto no aumento da amostra de indivíduos visando reforçar o resultado obtido no experimento desta dissertação. Para que desta forma, a partir da identificação de características negativas, poderão ser implementadas novas funcionalidades e/ou melhorar as já existentes, visando atender aos aspectos de usabilidade.

Essas sugestões para trabalhos futuros têm o intuito de criar uma ferramenta para programação de circuitos quânticos, mais completos quanto possível, similar a uma linguagem de programação de alto nível.

Referências Bibliográficas

Abreu (2010) A. C. B. de Abreu. Avaliação de usabilidade em softwares educativos. Dissertação de Mestrado, Universidade Estadual do Ceará Instuto Federal de eduacação, ciência e tecnologia do Ceará, Fortaleza, CE. Citado na pág. 72, 73, 74, 75, 85

Alencar Price e Toscani (2000) Ana Maria Alencar Price e Simão Sirineo Toscani. *Implementação de linguagens de programação: compiladores*. Sagra-Luzzatto. Citado na pág. 16, 17, 19

Andrade Barbosa (2007) Alexandre Andrade Barbosa. Um simulador simbólico de circuitos quânticos. Citado na pág. 1, 46

ANSI (1985) ANSI. *American National Standard Programming Language COBOL. ANSI X3.23-1985*. Nova York. Citado na pág. 20

B. Butscher (2013) H. Weimer B. Butscher. libquantum. the c library for quantum computing and quantum simulation, 2013. URL <http://www.libquantum.de>. Último acesso em 21/01/2014. Citado na pág. 46

Bernstein e Vazirani (1993) Ethan Bernstein e Umesh Vazirani. Quantum complexity theory. Em *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, páginas 11–20. ACM. Citado na pág. 35, 46

Chuang *et al.* (1998) Isaac L Chuang, Neil Gershenfeld e Mark Kubinec. Experimental implementation of fast quantum searching. *Physical review letters*, 80(15):3408. Citado na pág. 46

Cornelio (2008) Marcio Fernando Cornelio. *Estados emaranhados quânticos tri-partidos com um qubit*. Tese de Doutorado, Universidade de São Paulo. Citado na pág. 40

Cybis (2003) Walter de Abreu Cybis. Engenharia de usabilidade: uma abordagem ergonômica. *Florianópolis: Labiutil*. Citado na pág. 73, 74

D-Wave (2007) D-Wave. World's first commercial quantum computer demonstrated, Novembro 2007. URL <http://www.dwavesys.com/index.php?mact=News,cntnt01,detail,0&cntnt01articleid=4&cntnt01origid=15&cntnt01returnid=21>. Último acesso em 20/01/2014. Citado na pág. 46

DEITEL (2005) H. M DEITEL. *Java: como programar*. Ed. Pearson Education, 6º ed. Citado na pág. 3, 56

Deutsch (1989) David Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 425(1868):73–90. Citado na pág. 46

DiVincenzo (1995) David P DiVincenzo. Two-bit gates are universal for quantum computation. *Physical Review A*, 51(2):1015. Citado na pág. 43

Feynman (1985) Richard P Feynman. Quantum mechanical computers1. *Foundations of Physics*, 16(6):986. Citado na pág. 1

Feynman (1982) Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488. Citado na pág. 1

Figueiredo (2012) Filipe Figueiredo. Simulador de circuitos quânticos. Citado na pág. 47

FREITAS (2006) R. L FREITAS. Compiladores, 2006. URL <http://compila.googlecode.com/files/Apostila%20de%20Compiladores%20EComp.pdf>. Último acesso em 20/03/2014. Citado na pág. 5

GESSER (2003) C. E. GALS GESSER. Gerador de analisadores léxicos e sintáticos. *Universidade Federal de Santa Catarina–UFSC, Florianópolis*. Citado na pág. 2, 6, 18, 53

GROUP (2014) KHRONOS GROUP. The open standard for parallel programming of heterogeneous systems, 2014. URL <http://www.khronos.org/opencl/>. Último acesso em 04/12/2014. Citado na pág. 30

Grover (1996) Lov K Grover. A fast quantum mechanical algorithm for database search. Em *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, páginas 212–219. ACM. Citado na pág. 46, 66

H. Watanabe (2011) M. Suzuki e J. Yamazaki H. Watanabe. Qcad: Gui environment for quantum computer simulator, 2011. URL <http://qcad.sourceforge.jp>. Último acesso em 22/01/2014. Citado na pág. 46

Heisenberg (2013) Werner Heisenberg. *The physical principles of the quantum theory*. Courier Dover Publications. Citado na pág. 36

Hollingsed e Novick (2007) Tasha Hollingsed e David G Novick. Usability inspection methods after 15 years of research and practice. Em *Proceedings of the 25th annual ACM international conference on Design of communication*, páginas 249–255. ACM. Citado na pág. 74

ISO (1998) WD ISO. 9241-11. ergonomic requirements for office work with visual display terminals (vdts). *The international organization for standardization*. Citado na pág. 72

ISO-9241-10 (2010) ISO-9241-10. Ergonomic requirements for office work with visual display terminals (vdt's), 2010. URL <http://www.inf.ufsc.br/~cybis/ine5624/ISO9241parte10.pdf>. Último acesso em 03/07/2014. Citado na pág. 73, 74, 75, 85

J. e E (2010) Sanders J. e Kandrot E. *CUDA by Example, An Introduction to General Purpose GPU Programming*. ISBN 978-0-13-138768-3. Citado na pág. 25, 26

Jones e Mosca (1998) Jonathan A Jones e Michele Mosca. Implementation of a quantum algorithm on a nuclear magnetic resonance quantum computer. *The Journal of chemical physics*, 109:1648. Citado na pág. 46

Jones *et al.* (1998) Jonathan A Jones, Michele Mosca e Rasmus H Hansen. Implementation of a quantum search algorithm on a quantum computer. *Nature*, 393(6683):344–346. Citado na pág. 46

Jozsa (1998) Richard Jozsa. Quantum algorithms and the fourier transform. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):323–337. Citado na pág. 46

Kotler (2003) Philip Kotler. *Marketing de A a Z: 80 conceitos que todo profissional precisa saber*. Elsevier Brasil. Citado na pág. 75

Kyriakos Sgarbas (2010) Petroula Mavridi e Charalampos Tsimpouris Kyriakos Sgarbas. Demo: Quantum computer simulator, 2010. URL <http://www.wcl.ece.upatras.gr/ai/resources/demo-quantum-simulation>. Último acesso em 19/01/2014. Citado na pág. 46

Levin e Fox (2004) Jack Levin e James Alan Fox. Estatística para ciências humanas. Em *Estatística para ciências humanas*. Pearson. Citado na pág. xiv, 77, 81, 82, 83, 97

Likert (1932) Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*. Citado na pág. 75

Mack e Nielsen (1994) Robert L Mack e Jakob Nielsen. *Usability inspection methods*. Wiley & Sons. Citado na pág. 74

Martins de Paula (2014) Lauro Cássio Martins de Paula. Cuda vs. opencl: uma comparação teórica e tecnológica. *ForScience*, 2(1):31–46. Citado na pág. 31

McCarthy (1960) John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, 3(4):184–195. Citado na pág. 20

McMahon (2007) David McMahon. *Quantum computing explained*. Wiley. com. Citado na pág. 42

MENEZES (2005) P. B MENEZES. *Linguagens formais e Autômatos*. Porto Alegre: Bookman, 5º ed. Citado na pág. 4, 5, 8, 9

Nielsen (1994) Jakob Nielsen. Usability inspection methods. Em *Conference companion on Human factors in computing systems*, páginas 413–414. ACM. Citado na pág. 73

Nielsen *et al.* (2005) Michael A Nielsen, Isaac L Chuang e Ivan S Oliveira. *Computação quântica e informação quântica*. Bookman. Citado na pág. 35, 38

Nokelainen (2006) Petri Nokelainen. An empirical assessment of pedagogical usability criteria for digital learning material with elementary school students. *Educational Technology & Society*, 9(2):178–197. Citado na pág. 74

NVIDIA (2009) NVIDIA. *OpenCL Programming Guide for the CUDA Architecture*. 2.3 ed. Citado na pág. xiii, 24, 25, 31

NVIDIA (2008) NVIDIA. *Nvidia cuda compute unified device architecture, programming guide*. 2.0 ed. Citado na pág. xiii, 24, 25

Oliveira (2005) Ivan S Oliveira. *Física Moderna para iniciados, inter. e afic. vol 1*, volume 1. Editora Livraria da Fisica. Citado na pág. 38

Oliveira e Sarthour (2004) Ivan S. Oliveira e Roberto S. Sarthour. *Computação quântica e informação quântica*. Centro Brasileiro de Pesquisas Físicas. Citado na pág. 38

Oliveira Jr (2007) JAG Oliveira Jr. Apoio a avaliação de usabilidade na web: desenvolvimento do useweb [dissertação de mestrado]. *Campinas, SP: Instituto de Computação, Universidade Estadual de Campinas*. Citado na pág. 72

Portugal (2010) R. Portugal. Algoritmos quânticos de busca. Citado na pág. 39, 41

Prates e Barbosa (2003) Raquel Oliveira Prates e Simone Diniz Junqueira Barbosa. Avaliação de interfaces de usuário–conceitos e métodos. Em *Anais do XXIII Congresso Nacional da Sociedade Brasileira de Computação*. sn. Citado na pág. 73

Rodrigues (2010) G. T. M Rodrigues. Análise de usabilidade em ambientes virtuais de aprendizagem: O ambiente solar na perspectiva do aluno. Dissertação de Mestrado, Mestrado Profissional em Computação Aplicada do Centro de Ciência e Tecnologia da Universidade Estadual do Ceará. Citado na pág. 74, 75

Rodrigues (2011) Wilton Albuquerque Rodrigues. O computador quântico óptico. Citado na pág. 35

Santos (2007) R. C. Santos. Desenvolvimento de uma metodologia para avaliação de usabilidade de sistemas utilizando a lógica fuzzy baseado na iso. 2007. 115 p [dissertação de mestrado]. *Rio de Janeiro: Faculdade de Economia e Finanças IBMEC*. Citado na pág. 72

SANTOS (2004) Robson SANTOS. Alguns conceitos para avaliar usabilidade. 2003, 2004. Citado na pág. 75

Schilling (2009) A Schilling. Favihc framework de avaliação da interação humano computador. Dissertação de Mestrado, Mestrado em Informática Aplicada, UNIFOR-Universidade de Fortaleza. Fortaleza. Citado na pág. 72

Schrödinger (1935) Erwin Schrödinger. Discussion of probability relations between separated systems. Em *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 31, páginas 555–563. Cambridge Univ Press. Citado na pág. 40

Schrödinger (1992) Erwin Schrödinger. *What is life?: With mind and matter and autobiographical sketches*. Cambridge University Press. Citado na pág. 37

Sebesta (2002) Robert W Sebesta. *Concepts of programming languages*, volume 4. Addison Wesley. Citado na pág. 6, 7, 13, 14, 19, 20, 21

Sethi *et al.* (2008) Ravi Sethi, Jeffrey D Ullman e monica S. Lam. *Compiladores: princípios, técnicas e ferramentas*. Pearson Addison Wesley. Citado na pág. 15, 16, 17, 18, 19, 20

Shor (1994a) Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. Em *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, páginas 124–134. IEEE. Citado na pág. 46

Shor (1994b) Peter W Shor. Polynomial time algorithms for discrete logarithms and factoring on a quantum computer. Em *Algorithmic Number Theory*, páginas 289–289. Springer. Citado na pág. 68

Simon (1997) Daniel R Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483. Citado na pág. 46

Steane (1998) Andrew Steane. Quantum computing. *Reports on Progress in Physics*, 61(2): 117. Citado na pág. 41

Stone *et al.* **(2010)** John E Stone, David Gohara e Guochun Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66. Citado na pág. 30

Tsuchiyama *et al.* **(2010)** Ryoji Tsuchiyama, Takashi Nakamura, Takuro Iizuka, Akihiro Asahara, Satoshi Miki e Satoru Tagawa. The opencl programming book. *Fixstars Corporation*, 63. Citado na pág. 30

VASCONCELLOS (2009) F.B. VASCONCELLOS. Programando com gpus: Paralelizando o método lattice-boltzmann com cuda. Citado na pág. 23

Vazirani (2013) Umesh Vazirani. Quantum mechanics and quantum computation, 2013. URL <https://courses.edx.org/courses/BerkeleyX/CS-191x/2013_August/courseware/>. Último acesso em 05/11/2014. Citado na pág. 1

Vries (2010) A. Vries. jquantum - quantum computer simulation applet, 2010. URL <http://jquantum.sourceforge.net/jQuantumApplet.html>. Último acesso em 24/01/2014. Citado na pág. 47

Winckler e Pimenta (2002) Marco Winckler e Marcelo Soares Pimenta. Avaliação de usabilidade de sites web. *Disponível por WWW em "http://lis.univ-tlse1.fr/winckler/publications.html".(11 Maio 2002)*. Citado na pág. 72, 73

APÊNDICE A – (LinDCQ)

APÊNDICE B – Questionário de satisfação de uso

As perguntas deste questionário tem a finalidade de avaliar a facilidade de uso do software que você acabou de utilizar. Fique à vontade para responder, porque o que está sendo avaliado é este material e não você.

1. Este material possui funções necessárias, disponibilizadas de forma eficiente para as tarefas correspondentes? (Por exemplo: ajuda, configurações, salvar, etc.)

   1 2 3 4 5

   Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

2. Sente-se à vontade com as funções deste material, ou somente algumas lhes são familiares?

   1 2 3 4 5

   Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

3. Consegue visualizar os recursos disponíveis neste material?

   1 2 3 4 5

   Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

4. Entende com facilidade as palavras, nomes, abreviaturas ou símbolos que estão neste material?

   1 2 3 4 5

   Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

5. Permite alternar facilmente entre os menus ou telas?

   1 2 3 4 5

   Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

6. Não impõe qualquer interrupção desnecessária em seu ritmo de estudo? (Por exemplo, mensagens explicativas só aparecem quando você espera que apareçam.)

   1 2 3 4 5

   Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

7. As telas deste material com seus textos, botões e figuras têm formato de fácil reconhecimento?

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

8. Esse material pode ser entendido e usado por qualquer aluno, com pouca ou muita experiência com programação?

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

9. Depois de ambientado ao uso do material, não precisa de esforços para lembrar-se onde encontrar uma informação ou da funcionalidade de um botão?

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

10. Foi fácil aprender a usá-lo? Você não precisou de ajuda depois de entender como o sistema funciona?

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

11. Possui controle sobre as tarefas realizadas? (O sistema não dita a sequência de passos, você tem controle na ordem para finalização de suas tarefas)

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

12. Ao usar este material, sente que ele foi projetado para você? (As tarefas não são consideradas nem fáceis nem difíceis)

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

13. Você teria interesse de aprender os tópicos deste material mais profundamente? Se houvesse mais tempo, gostaria usar mais este material para aprender mais?

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

14. É rápido aprender um novo tópico ou recapitular um tópico anterior neste material?

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

15. A linguagem usada é natural. Os termos, frases, conceitos são similares àqueles usados no dia-a-dia ou no ambiente de estudo?

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

16. Não fico confuso na utilização dos símbolos, ícones e imagens.

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

17. Ao errar a solução de uma tarefa, o programa me envia um aviso amigável.

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

18. Este material de aprendizagem ensina habilidades e conhecimentos que serão necessários no futuro?

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

19. Exibe mensagem de erros cometidos.

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

20. As caixas de diálogo proporcionam informações adequadas ao desempenho das tarefas?

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

21. Os recursos de ajuda são fáceis de usar?

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

22. Este material me encoraja a aprender sobre o assunto abordado?

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

23. Possui interesse no assunto abordado neste material?

1 2 3 4 5

Discordo totalmente ○ ○ ○ ○ ○ Concordo totalmente

APÊNDICE C – Questionário do Perfil do participante

1. Nome completo:

2. Você já havia utilizado alguma ferramenta para trabalhar com circuitos quânticos:
   - Sim
   - Não

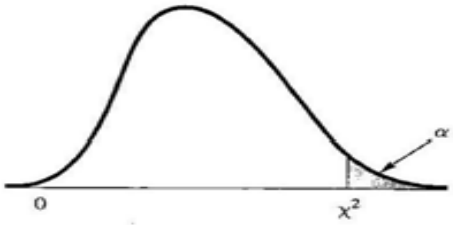
3. Como você classifica sua experiência com programação:
   - Baixa
   - Media
   - Alta

4. Ano de ingresso:

5. Entrada:
   - 1º
   - 2º

ANEXO A – Tabela Qui-quadrado

Valores críticos do qui-quadrado nos níveis de significância de 0,05 e 0,01 (α)



gl	α 0,05	α 0,01	gl	α 0,05	α 0,01
1	3,841	6,635	16	26,296	32,000
2	5,991	9,210	17	27,587	33,409
3	7,815	11,345	18	28,869	34,805
4	9,488	13,277	19	30,144	36,191
5	11,070	15,086	20	31,410	37,566
6	12,592	16,812	21	32,671	38,932
7	14,067	18,475	22	33,924	40,289
8	15,507	20,090	23	35,172	41,638
9	16,919	21,666	24	36,415	42,980
10	18,307	23,209	25	37,652	44,314
11	19,675	24,725	26	38,885	45,642
12	21,026	26,217	27	40,113	46,963
13	22,362	27,688	28	41,337	48,278
14	23,685	29,141	29	42,557	49,588
15	24,996	30,578	30	43,773	50,892

Fonte 1: *(Levin e Fox, 2004)*