Franklin de Lima Marquezino

A Transformada de Fourier Quântica Aproximada e sua Simulação

Petrópolis, RJ

Março, 2006

Franklin de Lima Marquezino

A Transformada de Fourier Quântica Aproximada e sua Simulação

Orientador:
Renato Portugal

Co-orientador:
Fernando Deeke Sasse

Laboratório Nacional de Computação Científica

Petrópolis, RJ

Março, 2006

MARQUEZINO, FRANKLIN DE LIMA

A Transformada de Fourier Quântica Aproximada e sua Simulação [Petrópolis] 2006

XII, 118p. 29,7cm (MCT/LNCC, M.Sc., Modelagem Computacional, 2006)

Dissertacão - Laboratório Nacional de Computação Científica, LNCC

1. Computação Quântica
2. Algoritmos quânticos
3. Simulação

I. MCT/LNCC II. Título (série)

A TRANSFORMADA DE FOURIER QUÂNTICA APROXIMADA E SUA SIMULAÇÃO

Franklin de Lima Marquezino

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DE FORMACÃO DE RECURSOS HUMANOS DO LABORATÓRIO NACIONAL DE COMPUTAÇÃO CIENTÍFICA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM MODELAGEM COMPUTACIONAL.

Aprovada por:

Dr. Renato Portugal
Orientador

Dr. Fernando Deeke Sasse
Co-orientador

Dr. Eduardo Lucio Mendes Garcia
LNCC/MCT

Dr. Gilson Antonio Giraldi
LNCC/MCT

Dr. José Abdalla Helayël-Neto
CBPF/MCT

Dr. Carlile Campos Lavor
UNICAMP

PETRÓPOLIS, RJ - BRASIL
MARÇO, 2006

Dedico a Deus.

Soli Deo Gloria.

Agradecimentos

Agradeço a Deus, a meus pais, amigos e parentes.

Aos orientadores, Renato Portugal e Fernando Sasse, pela atenção, pelo incentivo e pelos desafios.

Ao LNCC, a seus professores e funcionários com os quais tive o prazer de ter contato. Aos colegas do programa de pós-graduação em Modelagem Computacional. Ao grupo de Computação Quântica. A Leandro Gazoni, Rodolfo Dantas, Marcelo Collares, Fábio Borges e Eduardo Garcia, por ajudas valiosíssimas em questões técnicas que, de outra forma, poderiam ter prejudicado a realização das simulações.

Ao CBPF, ao Grupo de Física Teórica José Leite Lopes, e aos meus orientadores de Iniciação Científica, José Helayël e José Acebal. Com certeza, contribuíram enormemente para o meu amadurecimento científico.

À FAPERJ, pelo apoio financeiro.

Resumo da Dissertação apresentada ao MCT/LNCC como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M.Sc.)

A TRANSFORMADA DE FOURIER QUÂNTICA APROXIMADA E SUA SIMULAÇÃO

Franklin de Lima Marquezino

Março, 2006

Orientador: Renato Portugal
Co-orientador: Fernando Deeke Sasse

Modelagem Computacional

A Computação Quântica é uma área de pesquisa científica onde a teoria da Mecânica Quântica é usada para descrever um conceito mais geral que o da Máquina Universal de Turing clássica. Esta abordagem permite o desenvolvimento de algoritmos que podem ser consideravelmente mais rápidos que suas contrapartidas clássicas. Todos os algoritmos quânticos conhecidos até hoje que são exponencialmente mais rápidos que seus correspondentes clássicos utilizam a Transformada de Fourier Quântica (QFT) em alguma parte. Nesta dissertação, as versões exata e aproximada da QFT são construídas usando uma abordagem que generaliza o resultado fundamental de Coppersmith. O processo inicia com a representação matricial genérica da Transformada de Fourier Rápida (FFT) clássica, como descrita por Knuth, seguida por sua decomposição em termos de operadores quânticos universais. Tal decomposição também é alcançada por meio de uma abordagem recursiva. A simulação de computadores quânticos também é discutida. Experimentos computacionais são realizados com o objetivo de simular a QFT Aproximada sobre estados da base computacional e gatos de Schrödinger, e com diferentes níveis de aproximação. A qualidade das soluções e a complexidade computacional são estudadas, levando a resultados consistentes com a teoria.

Abstract of Dissertation presented to MCT/LNCC as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

THE APPROXIMATE QUANTUM FOURIER TRANSFORM AND ITS SIMULATION

Franklin de Lima Marquezino

March, 2006

Advisor: Renato Portugal
Co-advisor: Fernando Deeke Sasse

Computational Modelling

Quantum Computation is a field of scientific research where the theory of Quantum Mechanics is used to describe a concept more general than that of the classical Universal Turing Machine. This approach allows the development of algorithms that can be significantly faster than their classical counterparts. All quantum algorithms presently known which are exponentially faster than their classical counterparts use the Quantum Fourier Transform (QFT) in some part of it. In this dissertation both the exact and the approximate version of the QFT are built using an approach that generalizes the fundamental result of Coppersmith. The process starts with the construction of the general matrix representation of the classical Fast Fourier Transform (FFT), as described by Knuth, followed by its decomposition in terms of universal quantum operators. The decomposition is also achieved by means of a recursive approach. The simulation of quantum computers is also addressed. Computational experiments are performed in order to simulate the Approximate QFT over states of the computational basis and Schrödinger cats, and with different levels of approximation. The quality of the solutions and the computational complexity are investigated, leading to results consistent with those of the theory.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

$\boldsymbol{A_{jk}}$ Elemento da linha j e coluna k (indexando a partir de zero) de uma matriz A qualquer

$\boldsymbol{A_{m\times n}}$ Matriz A qualquer, de dimensão $m \times n$

$\mathbb{C}$ Conjunto dos números complexos

$\boldsymbol{F(\cdot,\cdot)}$ Fidelidade entre dois estados quânticos

$\boldsymbol{\hat{H}}$ Operador Hamiltoniano

$\hbar$ Constante de Plank dividida por 2π; tem-se $\hbar = \frac{h}{2\pi} \approx 1.5 \times 10^{-34}\ Js$

$\boldsymbol{i}$ Unidade imaginária, $\sqrt{-1}$

$\boldsymbol{j_s}$ Denota o $(s+1)$-ésimo bit menos significativo de $j \in \mathbb{Z}$, i.e., se j é um número inteiro de n bits então $j \equiv (j_{n-1} \cdots j_0)_2$

$\boldsymbol{k_B}$ Constante de Boltzmann, $k_B \approx 1.38 \times 10^{-23}\ J/K$

log Logaritmo na base dois

ln Logaritmo Neperiano

$\boldsymbol{O(\cdot)}$ Complexidade computacional no pior caso; diz-se que $f(x) = O(g(x))$ se $\exists C, x_0$ tal que $|f(x)| < Cg(x)$, $\forall x > x_0$

$\mathbf{proj}_{\boldsymbol{b}}\, \boldsymbol{a}$ Projeção do vetor a sobre o subespaço linear gerado por b

$\mathbf{tr}(\cdot)$ Traço de matriz

$\boldsymbol{X_j}$ Elemento de índice j (indexando a partir de zero) de um *array* X unidimensional

$\mathbb{Z}$ Conjunto dos números inteiros

$\mathbb{Z}_{\boldsymbol{N}}$ Grupo formado pelo conjunto $\{0, \cdots, N-1\}$ com soma módulo N

$\boldsymbol{\theta(\cdot)}$ Complexidade computacional no caso médio; diz-se que $f(x) = \theta(g(x))$ se $\exists c_1, c_2, x_0$, com $c_1 \neq 0$, $c_2 \neq 0$, tal que $c_1 g(x) < f(x) < c_2 g(x)$, $\forall x > x_0$

$\boldsymbol{\omega_N}$ Raiz principal da unidade, $\omega_N \equiv \exp\left(\frac{2\pi i}{N}\right)$

$\boldsymbol{(\cdot)^*}$ Complexo conjugado, e.g., A^*, z^*

$\lfloor \cdot \rfloor$ Maior número inteiro que seja menor ou igual a um número (*floor*)

$\neg$ Negação de um bit (*bit flip*)

$\| \cdot \|$ Norma de um vetor, definida por $\| \cdot \| = \langle \cdot | \cdot \rangle^{1/2}$

$\leftarrow$ Operação de atribuição em algoritmos

$\perp$ Ortogonalidade entre vetores

$\boldsymbol{|\psi_j\rangle \langle \psi_k|}$ Produto externo entre $|\psi_j\rangle$ e $|\psi_k\rangle$

$\boldsymbol{\langle \psi_j | \psi_k \rangle}$ Produto interno entre $|\psi_j\rangle$ e $|\psi_k\rangle$

$\boldsymbol{\langle \psi_j | A | \psi_k \rangle}$ Produto interno entre $|\psi_j\rangle$ e $A|\psi_k\rangle$. Equivalente ao produto interno entre $A^\dagger |\psi_j\rangle$ e $|\psi_k\rangle$

$\boldsymbol{|\psi_j\rangle \otimes |\psi_k\rangle}$ Produto tensorial entre $|\psi_j\rangle$ e $|\psi_k\rangle$

$\boldsymbol{|\psi_j\rangle\, |\psi_k\rangle}$ Produto tensorial entre $|\psi_j\rangle$ e $|\psi_k\rangle$ (notação compacta)

$\boldsymbol{|\psi_j \psi_k\rangle}$ Produto tensorial entre $|\psi_j\rangle$ e $|\psi_k\rangle$ (notação compacta)

$\boldsymbol{(\cdot)^{\otimes n}}$ Produto tensorial repetido n vezes, e.g., $I^{\otimes s} \equiv \underbrace{I \otimes \cdots \otimes I}_{s}$

$\boldsymbol{(\cdot)_2}$ Representação de um número inteiro em base binária

$\oplus$ Soma módulo dois, e.g., $a \oplus b \equiv a + b \pmod 2$

$\boldsymbol{(\cdot)^T}$ Transposto, e.g., A^T, z^T

$\boldsymbol{(\cdot)^\dagger}$ Transposto conjugado, e.g., $A^\dagger$, $z^\dagger$

$\boldsymbol{|\cdot\rangle}$ Vetor no espaço de Hilbert, em notação de Dirac; e.g., $|\psi_k\rangle$

$\boldsymbol{\langle \cdot |}$ Vetor dual (transposto conjugado) na notação de Dirac, e.g., $\langle \psi_j |$

Lista de Acrônimos

BDD *Binary Decision Diagram*

CNOT *Controlled NOT*, Porta NOT controlada

DFT *Discrete Fourier Transform*, Transformada de Fourier Discreta

FFT *Fast Fourier Transform*, Transformada de Fourier Rápida

FIRST *Fraunhofer Institut für Rechnerarchitektur und Softwaretechnik*

GB *Gigabyte*

GSL *GNU Scientific Library*

HSP *Hidden Subgroup Problem*, Problema do Subgrupo Escondido (ou Oculto)

KB *Kilobyte*

MB *Megabyte*

Mbps *Megabits* por segundo

MPI *Message Passing Interface*

OpenQUACS *Open-Source Quantum Computer Simulator*

QCE *Quantum Computer Emulator*

QCL *Quantum Computer Language*

QFT *Quantum Fourier Transform*, Transformada de Fourier Quântica

QSS *Quantum System Simulator*

VBLib *Villas-Boas Library*

1 Introdução

Uma área de pesquisa científica de grande crescimento nos últimos anos é a Computação Quântica. Ela aplica conceitos de Mecânica Quântica à Teoria da Computação, permitindo o desenvolvimento de algoritmos exponencialmente mais rápidos do que os correspondentes clássicos existentes. Um dos maiores propulsores das pesquisas em Computação Quântica na década de 1990 foi o artigo de Shor (1994), onde é apresentado um algoritmo eficiente para dois importantes problemas da Ciência da Computação: a fatoração de inteiros grandes e o logaritmo discreto. O problema da fatoração de inteiros, em particular, chamou muita atenção por possibilitar a quebra de certas chaves criptográficas. O método RSA, por exemplo, é amplamente utilizado e no entanto seria ineficaz se inteiros grandes pudessem ser fatorados com facilidade (RIVEST; SHAMIR; ADLEMAN, 1978). No momento de escrita da dissertação, o melhor algoritmo clássico para fatoração de inteiros é o *Number Field Sieve* (LENSTRA; LENSTRA JR., 1993; ZAYER, 1995), de complexidade computacional[1] $\exp \Theta(n^{1/3} \log^{2/3} n)$, enquanto a fatoração utilizando o algoritmo quântico de Shor possui complexidade $O(n^3)$.

O algoritmo de Shor utiliza uma versão quântica do algoritmo para Transformada de Fourier Discreta (DFT[2]) que é eficiente quando os fatores primos do número a ser fatorado não são grandes. Pouco tempo depois, Coppersmith (1994) desenvolveu um algoritmo eficiente para cálculo da DFT de vetores de comprimento igual a uma potência de dois, baseando-se no algoritmo da Transformada de Fourier Rápida (FFT[3]) descrito no clássico livro de Knuth (1981). Enquanto o algoritmo clássico de FFT possui complexidade $O(N \log N)$, com N correspondendo à quantidade de

[1] Complexidade computacional é o estudo de recursos, como tempo e memória, necessários para resolver um problema computacional. Um algoritmo é considerado eficiente quando os recursos mínimos necessários para sua execução crescem polinomialmente com o tamanho da entrada; e ineficiente quando os recursos necessários crescem mais rápido que qualquer polinômio.

[2] *Discrete Fourier Transform.*

[3] *Fast Fourier Transform.*

elementos do vetor de entrada, sua versão quântica é exponencialmente mais rápida, $O(\log^2 N)$. Cleve (1994) também chegou ao mesmo algoritmo, hoje conhecido como Transformada de Fourier Quântica (QFT[4]), utilizando uma abordagem recursiva. Nielsen e Chuang (2000) também mencionam um trabalho independente, porém não publicado, de David Deutsch, onde o algoritmo QFT é obtido. Esses artigos mostraram de forma clara a aplicabilidade da Computação Quântica, e desencadearam uma série de trabalhos na área, tanto teóricos como experimentais.

O trabalho de Coppersmith teve ainda outro resultado interessante: o desenvolvimento de uma Transformada de Fourier Quântica Aproximada. Este algoritmo possui complexidade $O(m \log N)$, onde $1 \leq m \leq \log N$ é um parâmetro que define o grau de aproximação — quanto mais perto de $\log N$, mais preciso. O parâmetro m normalmente é tomado próximo de $\log \log N$, de forma que a complexidade da QFT Aproximada torna-se $O(\log N \log \log N)$. Comparado à complexidade do algoritmo clássico de FFT, trata-se de uma diferença muito grande!

O algoritmo da QFT Aproximada traz outras vantagens. Barenco *et al.* (1996) utilizaram um modelo simplificado de descoerência — ruídos causados pela interação do sistema físico quântico com o meio — e demonstraram que, nestas condições, o algoritmo aproximado pode proporcionar resultados mais precisos que o algoritmo exato. Os mesmos autores ainda mostraram que a precisão da QFT exata pode ser atingida repetindo $O\left(\frac{\log^3 N}{m^3}\right)$ vezes a QFT Aproximada, parametrizada por m. O algoritmo de Shor, por exemplo, possui complexidade $O(\log^3 N)$, de forma que a QFT utilizada por ele poderia ser substituída por aplicações sucessivas da QFT Aproximada. Do ponto de vista experimental a QFT Aproximada também apresenta vantagens, pois requer portas lógicas[5] mais simples e em menor quantidade, sendo mais facilmente implementável. Portanto, o algoritmo quântico aproximado tem maiores chances de ser utilizado na prática, sendo por isso mais relevante que o algoritmo exato.

Apesar de muito importante, testes da QFT em um *hardware* quântico real ainda são inviáveis. O máximo que já se conseguiu realizar em laboratório, na área de Computação Quântica, foi a fatoração do número quinze através do algoritmo

[4] *Quantum Fourier Transform*.

[5] Na Computação Clássica, portas lógicas são funções $f : \{0,1\}^k \rightarrow \{0,1\}^l$, implementadas usualmente através de circuitos eletrônicos. Na Computação Quântica, as portas lógicas são operadores unitários.

de Shor. Para isso, foi utilizado um *hardware* de apenas sete *qubits*,[6] através da técnica de Ressonância Magnética Nuclear em uma solução líquida contendo uma molécula sintetizada especialmente para este fim (VANDERSYPEN *et al.*, 2001; BULNES, 2005). Não se sabe ainda como manipular estados quânticos suficientemente grandes, nem como mantê-los coerentes pelo tempo necessário para realização do cálculo. Enquanto os avanços tecnológicos não são alcançados no campo experimental, é importante simular os algoritmos quânticos em computadores clássicos, apesar destas simulações serem sempre extremamente ineficientes (FEYNMAN, 1982).

Nesta dissertação, é apresentada uma revisão sobre a Transformada de Fourier Quântica Aproximada de forma diferente daquelas usualmente encontradas na literatura. Em vez de mostrar o algoritmo pronto, fornecendo uma prova de corretude em seguida, o caminho inverso é percorrido. Tendo como base o algoritmo clássico FFT, mostra-se detalhadamente todos os passos que conduzem à versão quântica. Este resultado original é a maior contribuição da dissertação (MARQUEZINO; PORTUGAL; SASSE, 2006), e generaliza o método desenvolvido por Coppersmith. O algoritmo quântico também é construído por meio de uma abordagem recursiva original, a partir da representação matricial que implementa o método de Danielson-Lanczos. Espera-se, dessa forma, contribuir para uma melhor compreensão do processo de criação de um algoritmo quântico eficiente.

A simulação de computadores quânticos também é discutida. Após uma revisão detalhada sobre os simuladores de computadores quânticos, o simulador Feynman é apresentado. Alguns experimentos computacionais são realizados com o objetivo de ilustrar o funcionamento da QFT Aproximada. Primeiramente, são apresentados experimentos visando a verificação da qualidade dos resultados da transformada aproximada. Os resultados mostram que os gráficos das partes real e imaginária das amplitudes dos estados quânticos sofrem modificações muito pequenas conforme o parâmetro m diminui. Estes resultados são quantificados através da medida de fidelidade, constatando-se que para valores de m a partir de $\log \log N$ a fidelidade permanece muito próxima do valor máximo. São feitas medições de tempo em experimentos envolvendo as versões exata e aproximada da QFT, tendo como entradas estados da base computacional e estados emaranhados do tipo gato de Schrödinger. Neste caso, o simulador Feynman viabiliza uma análise mais detalhada que a

[6] Alguns autores, em português, preferem utilizar o termo q-bit. Ao longo desta dissertação será mantido o termo inglês.

maioria dos simuladores existentes, permitindo a identificação do tempo de processamento que realmente seria gasto no computador quântico. Os resultados obtidos são satisfatórios em todos os casos analisados.

No Capítulo 2 desta dissertação é feita uma revisão da Mecânica Quântica para Computação Quântica. No Capítulo 3, o conceito de Transformada de Fourier Discreta é revisado. Através de algumas manipulações algébricas, chega-se à DFT Aproximada. O algoritmo FFT é então revisado, de acordo com o algoritmo apresentado no livro de Knuth, e sua corretude é demonstrada. No Capítulo 4, inicialmente apresenta-se um caso particular de execução da FFT, ressaltando aspectos relevantes da representação matricial de cada passo do algoritmo. Em seguida apresenta-se, de forma detalhada e mais geral que aquela apresentada no artigo de Coppersmith, a passagem do algoritmo de FFT clássico para uma versão quântica, utilizando tanto a abordagem iterativa como a recursiva. Também são comentadas algumas propriedades úteis sobre a DFT e seu cálculo em computadores quânticos. No Capítulo 5 discutem-se as simulações de computadores quânticos, e apresenta-se uma revisão dos principais simuladores da atualidade. O simulador Feynman é introduzido (MARQUEZINO; MELLO JUNIOR, 2004c; MARQUEZINO *et al.*, 2005), e alguns experimentos computacionais são discutidos.

2 Mecânica Quântica e Computação Quântica

Por volta do início do século XX, começou-se a perceber conflitos entre os resultados experimentais e as previsões teóricas da Mecânica Newtoniana e do Eletromagnetismo Clássico, quando consideravam-se sistemas físicos em escala atômica ou sub-atômica. Nessa época a Mecânica Quântica começou a ser desenvolvida, e passou a fornecer um arcabouço matemático preciso para descrever muitos fenômenos físicos que não podem ser descritos pela Mecânica Clássica. Sabe-se, por exemplo, que toda carga elétrica sujeita a aceleração irradia energia. Portanto, os elétrons nos átomos, por estarem sujeitos à aceleração centrípeta, deveriam perder energia ao longo do tempo. Desta forma, o elétron deveria descrever rapidamente uma trajetória espiral até ser absorvido pelo núcleo — o que obviamente não corresponde à realidade. A Física Moderna apóia-se fortemente na Mecânica Quântica e na Teoria da Relatividade.

Neste Capítulo será apresentada uma revisão de Mecânica Quântica adequada ao estudo da Computação Quântica. Uma revisão mais detalhada pode ser obtida a partir de livros como Cohen-Tannoudji, Diu e Laloë (1977), Gasiorowicz (1974), Chaves (2001), Davydov (1976), dentre outros.

É conveniente revisar algumas notações utilizadas no estudo da Mecânica Quântica e da Computação Quântica (NIELSEN; CHUANG, 2000; PRESKILL, 1998). Um vetor em um espaço de Hilbert costuma ser denotado por $|\psi_i\rangle$, ou simplesmente por $|i\rangle$. Este símbolo chama-se *ket*, e faz parte da notação de Dirac. O vetor dual é denotado por $\langle i|$. O produto interno entre $|\psi_i\rangle$ e $|\psi_j\rangle$ é igual a $\langle\psi_i|\,|\psi_j\rangle$, mas pode ser denotado abreviadamente por $\langle\psi_i|\psi_j\rangle$.

O produto tensorial, ou produto de Kronecker (LOAN, 1992; PORTUGAL *et al.*, 2004), é uma forma de reunir espaços vetoriais para formar espaços maiores.

É denotado por $|\psi_i\rangle \otimes |\psi_j\rangle$, ou abreviadamente por $|\psi_i\rangle\,|\psi_j\rangle$, ou ainda $|\psi_i\psi_j\rangle$. O produto tensorial de duas matrizes $A_{m\times n}$ e $B_{p\times q}$,

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix},\ B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1q} \\ b_{21} & b_{22} & \cdots & b_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{p1} & b_{p2} & \cdots & b_{pq} \end{pmatrix}, \tag{2.1}$$

é a matriz $(A \otimes B)_{mp\times nq}$ definida por

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix}. \tag{2.2}$$

Há definições mais gerais para o produto tensorial, porém o tratamento matricial apresentado aqui é suficiente para os propósitos desta dissertação.

O produto de Kronecker satisfaz as seguintes propriedades:

1. Seja M um espaço vetorial de dimensão m, e seja N um espaço vetorial de dimensão n. Se $|\mu\rangle \in M$ e $|\nu\rangle \in N$, então $|\mu\rangle \otimes |\nu\rangle \in M \otimes N$, e $M \otimes N$ é um espaço vetorial de dimensão mn.

2. Para qualquer $|\mu\rangle \in M$ e $|\nu\rangle \in N$, e para qualquer escalar z, tem-se

$$z(|\mu\rangle \otimes |\nu\rangle) = (z\,|\mu\rangle) \otimes |\nu\rangle = |\mu\rangle \otimes (z\,|\nu\rangle),$$

3. Para quaisquer $|\mu_1\rangle, |\mu_2\rangle \in M$ e para qualquer $|\nu\rangle \in N$, tem-se

$$(|\mu_1\rangle + |\mu_2\rangle) \otimes |\nu\rangle = |\mu_1\rangle \otimes |\nu\rangle + |\mu_2\rangle \otimes |\nu\rangle,$$

4. Para qualquer $|\mu\rangle \in M$ e para quaisquer $|\nu_1\rangle, |\nu_2\rangle \in N$, tem-se

$$|\mu\rangle \otimes (|\nu_1\rangle + |\nu_2\rangle) = |\mu\rangle \otimes |\nu_1\rangle + |\mu\rangle \otimes |\nu_2\rangle.$$

5. Se $|\psi_A\rangle \in M$ e $|\psi_B\rangle \in N$, e se A e B são operadores lineares definidos nos espaços vetoriais M e N, respectivamente, então $(A \otimes B)(|\psi_A\rangle \otimes |\psi_B\rangle) = A\,|\psi_A\rangle \otimes B\,|\psi_B\rangle$.

As principais notações de Álgebra Linear utilizadas ao longo desta dissertação podem ser encontradas na Lista de Símbolos, na página x. Para a discussão da próxima seção, convém lembrar que um operador N é normal se $NN^\dagger = N^\dagger N$; um operador U é unitário se $U^\dagger U = I$; e um operador H é Hermitiano se $H = H^\dagger$. Conseqüentemente, todo operador Hermitiano é normal. Como boas referências em Álgebra Linear podemos citar, por exemplo, os livros de Hoffman e Kunze (1971), Lang (1987) e de Strang (1988).

2.1 Os postulados da Mecânica Quântica

Nesta seção os quatro postulados fundamentais da Mecânica Quântica são descritos (NIELSEN; CHUANG, 2000). O primeiro postulado trata da descrição matemática de um sistema quântico isolado. O segundo postulado trata da evolução dos sistemas físicos quânticos. O terceiro postulado descreve a forma como pode-se extrair informações de um sistema quântico através de medições. Finalmente, o quarto postulado descreve a forma como sistemas quânticos diferentes podem ser combinados.

2.1.1 Primeiro postulado: espaço de estados

Pode-se associar a qualquer sistema quântico isolado um espaço vetorial complexo $\mathbb{C}^n$ com produto interno, chamado espaço de Hilbert ou espaço de estados. O sistema físico pode ser descrito totalmente por meio de um vetor unitário deste espaço vetorial, chamado vetor de estado. Por analogia com um sistema físico clássico de dois níveis, capaz de representar um bit[1], pode-se pensar também em um sistema físico quântico de dois níveis, que seria capaz de representar um bit quântico.

Definição 2.1 (Qubit). Um *qubit*[2] ou bit quântico é um vetor unitário em $\mathbb{C}^2$.

Definição 2.2 (Estado). Um estado quântico de n *qubits* é um vetor unitário $|\psi\rangle$ em $\mathbb{C}^n$.

Estados quânticos com $n > 1$ *qubits* são freqüentemente chamados de registradores quânticos. A representação matemática destes registradores envolve o quarto

[1] *Binary digit.*
[2] *Quantum bit.*

postulado da Mecânica Quântica (combinação de estados), que será tratado na Seção 2.1.4.

No contexto da Computação Quântica existe uma base ortogonal para $\mathbb{C}^2$, chamada base computacional, cujos vetores são usualmente denotados por $|0\rangle$ e $|1\rangle$. Estes estados podem ser representados por vetores coluna $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ e $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ e correspondem aos bits clássicos 0 e 1, respectivamente. Portanto, um *qubit* arbitrário pode ser escrito como

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{2.3}$$

onde $\alpha, \beta \in \mathbb{C}$ são chamadas amplitudes e satisfazem $|\alpha|^2 + |\beta|^2 = 1$. Ao estado $|\psi\rangle$ dá-se o nome de superposição de estados $|0\rangle$ e $|1\rangle$, e sua interpretação física é a co-existência do *qubit* nestes dois estados.

Apesar de um *qubit* aparentemente requerer quatro números reais para ser completamente descrito, pode-se aproveitar a restrição dos valores de suas amplitudes e reescrever a Equação (2.3) como

$$|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle\right), \tag{2.4}$$

onde $\gamma, \phi, \theta \in \mathbb{R}$. A fase global $e^{i\gamma}$ pode ser ignorada, por não possuir efeito observável (NIELSEN; CHUANG, 2000). Assim, pode-se reescrever a equação como

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle. \tag{2.5}$$

Desta forma, um *qubit* pode ser representado por um ponto em uma esfera no $\mathbb{R}^3$, chamada esfera de Bloch (Figura 1). No entanto, a mesma representação gráfica não pode ser utilizada para sistemas de múltiplos *qubits*.

2.1.2 Segundo postulado: evolução dos estados

A evolução de um sistema quântico fechado é descrita por uma transformação unitária. Se em um instante inicial t_0 o estado do sistema quântico é $|\psi_0\rangle$, e em um instante final t_f o estado passa a ser $|\psi_f\rangle$, então existe um operador unitário U, dependente somente de t_0 e t_f, tal que

$$|\psi_f\rangle = U|\psi_0\rangle. \tag{2.6}$$

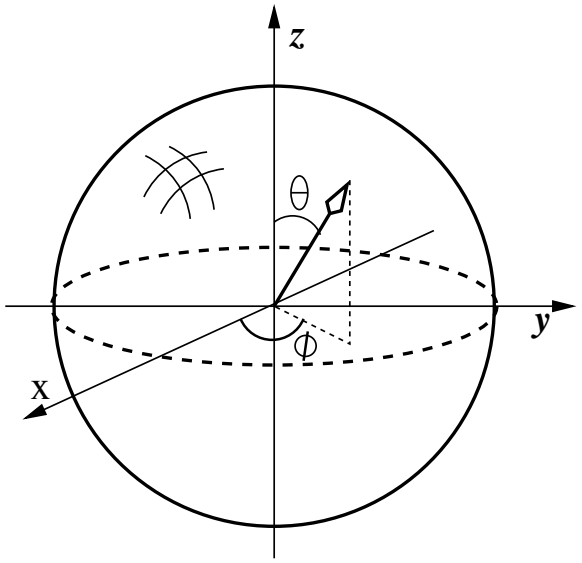


Figura 1: Esfera de Bloch

Equivalentemente, este postulado poderia ser enunciado em termos de equação de Schrödinger. A equação será vista no Capítulo 5, e maiores detalhes podem ser encontrados em livros de Mecânica Quântica (COHEN-TANNOUDJI; DIU; LALOË, 1977; GASIOROWICZ, 1974; DAVYDOV, 1976; CHAVES, 2001).

De acordo com o segundo postulado, operadores lineares unitários são potenciais algoritmos quânticos. No entanto, nem todo operador unitário pode ser imediatamente implementado através de um experimento físico no laboratório, sendo necessário antes expressá-lo através de operadores mais elementares. Se o número de operadores elementares requeridos cresce rapidamente em função do tamanho da entrada do algoritmo, o mesmo é ineficiente, e portanto sua implementação em um computador quântico não traria vantagens. A FFT é um exemplo de algoritmo que pode ser descrito como uma composição de poucos operadores unitários básicos, sendo por esse motivo eficiente em um computador quântico.

A evolução dos sistemas físicos quânticos é determinística. O caráter probabilístico da Mecânica Quântica — e, portanto, da Computação Quântica — surge apenas no momento da medição, conforme o postulado a seguir.

2.1.3 Terceiro postulado: medição de estados

Seja V um espaço vetorial de dimensão d, e W um subespaço vetorial em V, de dimensão k. É possível construir uma base ortonormal $|1\rangle, |2\rangle, \cdots, |d\rangle$ para V de tal

forma que $|1\rangle, |2\rangle, \cdots, |k\rangle$ seja uma base ortonormal para W (NIELSEN; CHUANG, 2000).

Um projetor no subespaço W pode ser definido como o operador Hermitiano

$$P \equiv \sum_{1 \leq i \leq k} |i\rangle \langle i|. \tag{2.7}$$

É fácil demonstrar que P satisfaz a equação $P^2 = P$.

Uma medição é caracterizada por um observável M, i.e., um operador Hermitiano no espaço de estados do sistema sendo observado. Como M é normal, pode-se fazer sua decomposição espectral,[3]

$$M = \sum_m m P_m, \tag{2.8}$$

de forma que P_m seja o projetor no autoespaço de M com autovalor m.

De acordo com este postulado da Mecânica Quântica, os possíveis resultados desta medição são os autovalores m de M. Se um estado $|\psi\rangle$ for medido, a probabilidade de um certo resultado m ocorrer é

$$p(m) = \langle \psi | P_m | \psi \rangle. \tag{2.9}$$

Após a medição, o estado sofre uma modificação irreversível, passando a valer

$$|\psi_f\rangle = \frac{P_m |\psi\rangle}{\sqrt{p(m)}}. \tag{2.10}$$

Como exemplo, pode-se citar a medição do observável

$$Z \equiv \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \tag{2.11}$$

cujos autovalores são $+1$ e -1, correspondendo aos autovetores $|0\rangle$ e $|1\rangle$. Portanto $Z = |0\rangle \langle 0| - |1\rangle \langle 1|$. Então, a medição do *qubit* genérico, definido na Equação (2.3), resulta em $+1$ com probabilidade $\langle \psi | 0 \rangle \langle 0 | \psi \rangle = |\alpha|^2$, e em -1 com probabilidade $\langle \psi | 1 \rangle \langle 1 | \psi \rangle = |\beta|^2$, justificando, portanto, a restrição $|\alpha|^2 + |\beta|^2 = 1$ para as ampli-

[3]Segundo o teorema da decomposição espectral (i) todo operador normal M em um espaço vetorial V é diagonal com relação a uma base ortonormal de V; e (ii) todo operador diagonalizável é normal. Maiores detalhes deste importante teorema e sua demonstração podem ser encontrados, por exemplo, no livro de Nielsen e Chuang (2000, pág. 72).

tudes dos *qubits*.

2.1.4 Quarto postulado: combinação de estados

O espaço de estados de um sistema composto é o produto tensorial dos espaços de estados de cada parte. Numerando os sistemas físicos de 1 até N, e chamando de $|\psi_\iota\rangle$ o estado do sistema ι, o sistema total tem o estado

$$\begin{aligned} |\psi\rangle &= |\psi_1\rangle \otimes |\psi_2\rangle \otimes \ldots \otimes |\psi_N\rangle \quad (2.12) \\ &= |\psi_1\rangle\, |\psi_2\rangle \ldots |\psi_N\rangle\,. \quad (2.13) \end{aligned}$$

Este postulado tem uma conseqüência interessante. Como os estados fazem parte de um espaço vetorial nada impede que um sistema quântico composto encontre-se em um estado que seja combinação linear, por exemplo, dos estados $|0\rangle\,|0\rangle = |00\rangle$ e $|1\rangle\,|1\rangle = |11\rangle$. Assim,

$$|\psi\rangle = \alpha\,|00\rangle + \beta\,|11\rangle\,, \quad (2.14)$$

com $\alpha, \beta \neq 0$, $|\alpha|^2 + |\beta|^2 = 1$. Para encontrar os estados das partes que compõem $|\psi\rangle$, pode-se fazer

$$\begin{aligned} |\psi\rangle &= (a\,|0\rangle + b\,|1\rangle) \otimes (c\,|0\rangle + d\,|1\rangle) \\ &= ac\,|00\rangle + ad\,|01\rangle + bc\,|10\rangle + bd\,|11\rangle \\ &= \alpha\,|00\rangle + \beta\,|11\rangle\,, \quad (2.15) \end{aligned}$$

de modo que

$$\begin{cases} ac &= \alpha \\ ad &= 0 \\ bc &= 0 \\ bd &= \beta. \end{cases} \quad (2.16)$$

Tomando-se, por exemplo, a segunda equação do sistema, tem-se que $a = 0$ (contradizendo a primeira equação) ou $d = 0$ (contradizendo a última equação). Desta forma, prova-se que o estado da Equação (2.14) não pode ser fatorado.

Definição 2.3 (Estado emaranhado). Um estado quântico, representado por $|\psi\rangle \in \mathbb{C}^{2^n}$, está emaranhado se não existe $|\psi_A\rangle \in \mathbb{C}^{2^a}$ e $|\psi_B\rangle \in \mathbb{C}^{2^b}$, com $a + b = n$, satisfazendo $|\psi\rangle = |\psi_A\rangle \otimes |\psi_B\rangle$.

Portanto, um estado composto emaranhado é aquele cujas partes constituintes não podem ser descritas isoladamente. O emaranhamento não possui análogo clássico, e parece desempenhar papel fundamental na Computação Quântica. Todos os algoritmos quânticos com ganho exponencial conhecidos até hoje aproveitam-se do emaranhamento de alguma forma.

2.1.5 Formalismo de operadores densidade

Os quatro postulados da Mecânica Quântica foram descritos utilizando o formalismo de vetores de estado. Existe, no entanto, outro formalismo matematicamente equivalente e especialmente útil quando o estado quântico não é conhecido com certeza. Se um estado quântico pode estar em qualquer dos estados $|\psi_\iota\rangle$, indexados por ι, com probabilidade p_ι, diz-se que $\{p_\iota, |\psi_\iota\rangle\}$ é um *ensemble* de estados puros. Neste caso, define-se o operador densidade — ou matriz densidade — do sistema, de acordo com a equação

$$\rho \equiv \sum_\iota p_\iota \, |\psi_\iota\rangle \, \langle\psi_\iota|. \tag{2.17}$$

No contexto de operadores densidade, chama-se de estado puro um sistema quântico cujo estado é conhecido com precisão. Neste caso, $\rho = |\psi\rangle \, \langle\psi|$, onde $|\psi\rangle$ é o estado do sistema no formalismo de vetores. Quando o estado não é puro, diz-se que ele é um estado misto, ou que é uma mistura de estados quânticos. Pode-se provar que se o estado for puro, $\mathrm{tr}(\rho^2) = 1$, e se for misto, $\mathrm{tr}(\rho^2) < 1$.

Todos os postulados da Mecânica Quântica podem ser reescritos em termos de operadores densidade (NIELSEN; CHUANG, 2000). Também é interessante utilizar este formalismo para definir uma importante medida de distância entre estados quânticos, chamada fidelidade.

Definição 2.4 (Fidelidade). A fidelidade F entre dois estados quânticos ρ e σ é definida como

$$F(\rho, \sigma) \equiv \mathrm{tr} \sqrt{\rho^{1/2} \sigma \rho^{1/2}}. \tag{2.18}$$

2.2 O teorema da não-clonagem

Além do emaranhamento, uma segunda diferença marcante entre Mecânica Newtoniana e Mecânica Quântica é o fato de que a informação clássica — a informação representada por um sistema físico clássico — pode ser copiada livremente, enquanto a informação quântica, em geral, não pode ser copiada perfeitamente sem que o estado original seja destruído. Este importante resultado, conhecido como *no-cloning theorem* na literatura em língua inglesa, foi descoberto por Wootters e Zurek (1982).

A demostração do teorema inicialmente supõe a possibilidade de criação de uma máquina cujas entradas sejam dois *qubits*, sendo o primeiro um estado $|\psi\rangle$ desconhecido, e o segundo um estado puro padrão, $|s\rangle$, funcionando como um papel em branco em uma máquina de fotocópia. O estado inicial da máquina é $|\psi\rangle \otimes |s\rangle$. Deseja-se um operador unitário U tal que

$$U(|\psi\rangle \otimes |s\rangle) = |\psi\rangle \otimes |\psi\rangle \,. \tag{2.19}$$

Além disso, para a máquina ser realmente útil, ela deve ser capaz de copiar mais de um estado diferente. Logo, U também deve satisfazer

$$U(|\phi\rangle \otimes |s\rangle) = |\phi\rangle \otimes |\phi\rangle. \tag{2.20}$$

O produto interno entre as Equações (2.19) e (2.20) é

$$\langle\psi|\phi\rangle = \langle\psi|\phi\rangle^2. \tag{2.21}$$

As únicas soluções a Equação (2.21) são $\langle\psi|\phi\rangle = 1$ e $\langle\psi|\phi\rangle = 0$, ou seja, respectivamente, quando $|\psi\rangle = |\phi\rangle$ ou quando $|\psi\rangle \perp |\phi\rangle$. Desta forma prova-se que, além do caso trivial, uma máquina de clonagem quântica somente é capaz de copiar **perfeitamente** estados ortogonais. É possível criar máquinas de clonagem quântica para atuarem sobre estados arbitrários, porém estas geram apenas cópias **aproximadas** (SCARANI *et al.*, 2005).

2.3 Histórico da Computação Quântica

Duas importantes teorias desenvolvidas no início do século XX foram marcantes. Uma delas foi a Mecânica Quântica, um arcabouço matemático que permitiu a

elaboração de teorias precisas para descrever sistemas físicos muito pequenos, tais como fótons e elétrons. Outra grande conquista foi o desenvolvimento da Teoria da Computação, com grande coolaboração de Turing (1936). Em seu artigo, motivado pelo *Entscheidungsproblem* de Hilbert, ele descreve a noção abstrata de uma máquina capaz de executar algoritmos: a Máquina de Turing. Trata-se de uma máquina teórica composta de: um programa; um controle de estados finitos, consistindo de um conjunto finito de estados internos; uma fita, desempenhando o papel de memória do computador; e uma cabeça de leitura e gravação (NIELSEN; CHUANG, 2000). Turing criou ainda a noção de computador programável, demonstrando a existência de uma Máquina Universal de Turing, capaz de simular qualquer outra Máquina de Turing. Outro passo importante para o estabelecimento da Ciência da Computação foi a tese de Church-Turing, — nomeada desta forma por ter sido desenvolvida originalmente por Church (1936) e posteriormente aprofundada pelo já mencionado matemático inglês — segundo a qual todo processo algorítmico que possa ser realizado na natureza pode ser descrito por uma Máquina de Turing (SHAPIRO, 1990).

Desde a invenção do transistor por Bardeen, Brattain e Shockley em 1948, um grande progresso foi observado no desenvolvimento dos computadores (TANENBAUM, 2001). Estes tornavam-se cada vez menores e mais velozes. Gordon Moore, em 1965, estabeleceu uma lei segundo a qual o número de transistores por unidade de área — e conseqüentemente, o poder de processamento dos computadores — dobraria aproximadamente a cada dois anos (MOORE, 1965). Esta lei ficou conhecida como lei de Moore, e de fato conseguiu prever razoavelmente a evolução dos computadores até o presente. No entanto, para que a lei de Moore seja válida, faz-se necessária uma constante miniaturização dos componentes dos computadores. Naturalmente, chegará o dia em que os componentes alcançarão o limite de indivisibilidade da matéria, e a lei de Moore deixará de ser válida. Mesmo antes disso, surgirão problemas quando os componentes dos computadores forem se aproximando de dimensões atômicas, devido ao aparecimento de efeitos quânticos. A fim de vencer o obstáculo imposto pela natureza, e dar continuidade ao avanço dos computadores, existem ao menos duas possibilidades. Uma delas envolve o aperfeiçoamento da própria computação clássica, como a introdução de modificações na arquitetura dos computadores ou a utilização de computação paralela, por exemplo. A outra possibilidade consiste na utilização daquilo que, inicialmente, parecia

o grande obstáculo: os efeitos quânticos da matéria. A segunda alternativa envolve grandes desafios tecnológicos, porém ao mesmo tempo viabiliza ganhos exponencialmente maiores.

De fato, a Computação está muito ligada à Física. Em 1961, por exemplo, Landauer publicou um importante trabalho onde é feito um estudo da relação entre consumo de energia e computação. Segundo Landauer, a energia dissipada por um computador quando este apaga um único bit de informação é maior ou igual a $k_B T \ln 2$, onde k_B é a constante de Boltzmann e T é a temperatura do ambiente do computador (LANDAUER, 1961). Outro resultado notável na Física da Computação foi o artigo de Bennett (1973), onde é demonstrada a possibilidade de realização de operações computacionais reversíveis. Como conseqüência deste trabalho, tem-se que é possível realizar operações computacionais sem dissipação de energia.[4]

A partir da década de 1980 surgiram alguns trabalhos muito importantes para a Computação Quântica. Uma contribuição importante foi o conceito de Máquina de Turing Quântica, desenvolvido por Benioff (1980), e posteriormente aprofundado por Deutsch (1985), Yao (1993) e Bernstein e Vazirani (1997). Destaca-se também o artigo de Feynman (1982), onde este físico norte-americano argumentava que a simulação de sistemas físicos quânticos por Máquinas de Turing seria um problema de complexidade exponencial, e que para simular eficientemente sistemas quânticos, seria necessário construir um computador baseado nos mesmos princípios da Mecânica Quântica. O trabalho desenvolvido independentemente por Manin (1980) também trazia conclusões semelhantes. Em 1989 o modelo de circuitos quânticos foi desenvolvido por Deutsch, e em 1993 foi aprofundado por Yao (DEUTSCH, 1989; YAO, 1993). Posteriormente, Barenco *et al.* (1995) demonstraram a universalidade das portas CNOT e portas atuando em um *qubit*.

Para se melhor compreender as motivações da Computação Quântica, é interessante fazer uma breve digressão sobre a tese de Church-Turing. Sua versão forte dizia que qualquer processo algorítmico da natureza pode ser simulado **eficientemente** por uma Máquina de Turing. A tese de Church-Turing normalmente é aceita sem muita hesitação, porém a sua versão forte, pelo simples acréscimo da palavra "eficientemente", já foi desafiada pela existência de algoritmos randômicos que não

[4]Uma revisão bastante didática deste tema pode ser obtida no artigo de Marquezino e Mello Junior (2004b).

parecem ter solução eficiente em Máquinas de Turing determinísticas[5]. Este problema pode ser solucionado com uma modificação na versão forte, passando a dizer que qualquer processo algorítmico da natureza pode ser simulado eficientemente por uma Máquina de Turing **probabilística**.

Dentro deste contexto, uma questão levantada pelo físico David Deutsch é se existe um modelo computacional, baseado nas leis da Física, com o qual seja possível estabelecer uma versão ainda mais forte da tese de Church-Turing. Este modelo computacional deveria ser capaz de simular **eficientemente um sistema físico arbitrário**. Na tentativa de responder essa questão, Deutsch utilizou a teoria da Mecânica Quântica. E, de fato, ele conseguiu desenvolver um algoritmo quântico mais rápido que qualquer algoritmo possível de ser implementado em um computador clássico (DEUTSCH, 1985). Para uma função $f(x) : \{0,1\} \rightarrow \{0,1\}$ o algoritmo de Deutsch verifica se $f(0) = f(1)$ ou $f(0) \neq f(1)$, avaliando $f(x)$ apenas uma vez.

O algoritmo de Deutsch não possui nenhuma aplicação prática. No entanto, vários pesquisadores conseguiram resolver eficientemente, utilizando o formalismo da Mecânica Quântica, alguns problemas computacionais de grande importância que não possuem solução eficiente conhecida em Máquinas de Turing — mesmo probabilísticas. Shor (1994) desenvolveu algoritmos para fatoração de inteiros grandes e cálculo de logaritmo discreto, utilizando para isso um algoritmo para cálculo de DFT em computadores quânticos, eficiente quando N é menor que $\log N$, onde N é a quantidade de elementos do vetor a ser transformado. No mesmo ano, motivado pelo trabalho de Shor, Coppersmith (1994) desenvolveu uma versão quântica da FFT quando N é potência de dois. Esta importante sub-rotina quântica é a QFT, discutida no Capítulo 4. A mesma sub-rotina foi desenvolvida independentemente por Cleve (1994), através de uma abordagem recursiva.

Kitaev (1995) utiliza a QFT para calcular ordem de elementos de um grupo. Simon (1997) desenvolveu um algoritmo quântico para resolver uma instância do Problema do Subgrupo Escondido (HSP[6]), quando $G = \mathbb{Z}_2^n$. Há ainda resultados mais recentes em algoritmos para Teoria de Grupos como, por exemplo, o trabalho de Hallgren, Russell e Ta-Shma (2000), onde se apresenta uma solução do HSP para

[5]Segundo Nielsen e Chuang (2000), o primeiro destes algoritmos foi o teste de primalidade desenvolvido por Solovay e Strassen (1976).

[6]*Hidden Subgroup Problem*. Dado um grupo G, por meio de um conjunto gerador, o HSP consiste em encontrar os geradores de um subgrupo H. Para isso, utiliza-se uma função, chamada oráculo, que diz se um elemento em G também pertence a H ou a um coset de H (LOMONT, 2004).

um subgrupo normal através da QFT, e os trabalhos de Mosca (1999), Watrous (2001), Cheung e Mosca (2001) e de Ivanyos, Magniez e Santha (2003) onde se discutem problemas como decomposição de grupos Abelianos e cálculo de ordem de grupos solúveis.

Todos os algoritmos mencionados nos parágrafos anteriores, que apresentam ganho exponencial de complexidade em relação aos algoritmos clássicos, utilizam a QFT. A Transformada de Fourier Quântica ainda foi utilizada no desenvolvimento de um algoritmo para soma, sem no entanto alcançar ganho exponencial de complexidade em relação ao algoritmo clássico (DRAPER, 2000; DRAPER *et al.*, 2004). Grover (1996) desenvolveu um algoritmo quântico para busca em listas não ordenadas, com ganho quadrático em relação ao algoritmo clássico.

Desta forma, a presença de efeitos quânticos nos computadores pode não ser um problema. Ao contrário, pode ser a oportunidade de desenvolver um modelo computacional muito mais eficiente que os atuais baseados na Mecânica Newtoniana.

2.4 Algoritmos e circuitos quânticos

O caráter contra-intuitivo da Mecânica Quântica exige atenção redobrada a qualquer que se proponha a fazer analogias entre a Computação Clássica e a Quântica. No entanto, certas analogias são, até certo ponto, bastante úteis.

No início deste Capítulo foi tratada a diferença entre bit clássico e quântico, e pôde-se perceber que o último, ao menos do ponto de vista matemático, é apenas uma generalização do primeiro. Dando prosseguimento aos conceitos, deve-se ressaltar o fato de que em Ciência da Computação o interesse não está no mero armazenamento de informação, mas também na sua manipulação, de forma a obter os resultados desejados.[7] As operações mais simples, evidentemente, são aquelas que atuam em apenas um bit, representadas na Tabela 1. No caso clássico, a única operação não-trivial deste tipo é a porta lógica NOT. Por outro lado, na Computação Quântica há infinitas portas lógicas não-triviais que podem atuar em um único *qubit*. De fato, qualquer matriz unitária de dimensão 2×2 representa uma porta lógica quântica sobre um *qubit*. As principais portas lógicas quânticas estão descritas na Tabela 2.

[7] Uma revisão bastante didática deste tema pode ser obtida no artigo de Marquezino e Mello Junior (2004a).

Entrada	Porta NOT	Identidade
0	1	0
1	0	1

Tabela 1: Operações lógicas sobre um bit

Nome	Matriz
Identidade	$I \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
Hadamard	$H \equiv \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
Pauli-X (NOT)	$\sigma_X \equiv X \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli-Y	$\sigma_Y \equiv Y \equiv \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli-Z	$\sigma_Z \equiv Z \equiv \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Fase	$S \equiv \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
$\pi/8$	$T \equiv \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/8} \end{pmatrix}$
$R^{(k)}$	$\begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}$

Tabela 2: Principais operações lógicas sobre um *qubit*

A representação de portas lógicas quânticas atuando sobre n *qubits* envolve matrizes unitárias de dimensão $2^n \times 2^n$. A aplicação simultânea de uma porta sobre o *qubit* $|\psi_p\rangle$ e de outra sobre o *qubit* $|\psi_q\rangle$ é representada pelo produto tensorial $P \otimes Q$ entre as matrizes que representam cada porta lógica. Além destas operações existe ainda, em Computação Quântica, uma classe de portas lógicas que atuam sobre dois ou mais *qubits* sem serem fatoradas como produtos tensoriais de matrizes 2×2. Casos particulares importantes de operações lógicas atuantes em mais de um *qubit* são as portas controladas e os *swaps*. Portas controladas são aquelas que atuam sobre certos *qubits* alvo somente se todos os *qubits* de controle valerem $|1\rangle$; e *swaps* são portas que atuam em dois *qubits* trocando seus valores. De todas as portas que atuam em mais de um *qubit*, a única que não pode ser fatorada é a porta NOT controlada (CNOT). As demais podem ser fatoradas em CNOTs e portas atuando em um *qubit* (BARENCO *et al.*, 1995). É fácil verificar que um *swap* pode ser implementado através de três portas CNOT, alternando controle e alvo.

De certa forma, a porta CNOT desempenha papel análogo ao dos estados emaranhados, discutidos anteriormente. E semelhantemente, todos os algoritmos quânticos de ganho exponencial descobertos até hoje utilizam não somente portas de um *qubit*, mas também portas que atuam de forma simultânea em dois *qubits*.

Ao analisar o caso das portas que atuam sobre mais de um bit, começam a surgir novas diferenças entre a Computação Clássica e a Quântica. Estas diferenças tornam-se mais claras através de um exemplo. Convém utilizar o formalismo dos circuitos clássicos em vez do formalismo de Máquinas de Turing. Um exemplo de circuito clássico pode ser encontrado na Figura 2. Neste circuito, o valor S da saída corresponde ao valor predominante das entradas A, B e C.

O circuito quântico é bastante parecido com o clássico. Nele, as portas lógicas são representadas por caixas e os fios representam o fluxo dos dados de uma porta até a outra. O circuito é sempre lido da esquerda para a direita. Nesta notação, o símbolo $\bullet$ é colocado sobre os fios correspondentes aos *qubits* de controle, e uma caixa com a identificação da porta lógica é colocada sobre o fio correspondente aos *qubits* alvo. A porta lógica quântica NOT pode ainda ser representada pelo símbolo $\oplus$. Portas de *swap* também recebem notação especial. Normalmente são representadas através de símbolos $\times$ ligados entre si, e posicionados sobre os fios correspondentes aos *qubits* que serão trocados. As notações utilizadas em circuitos quânticos

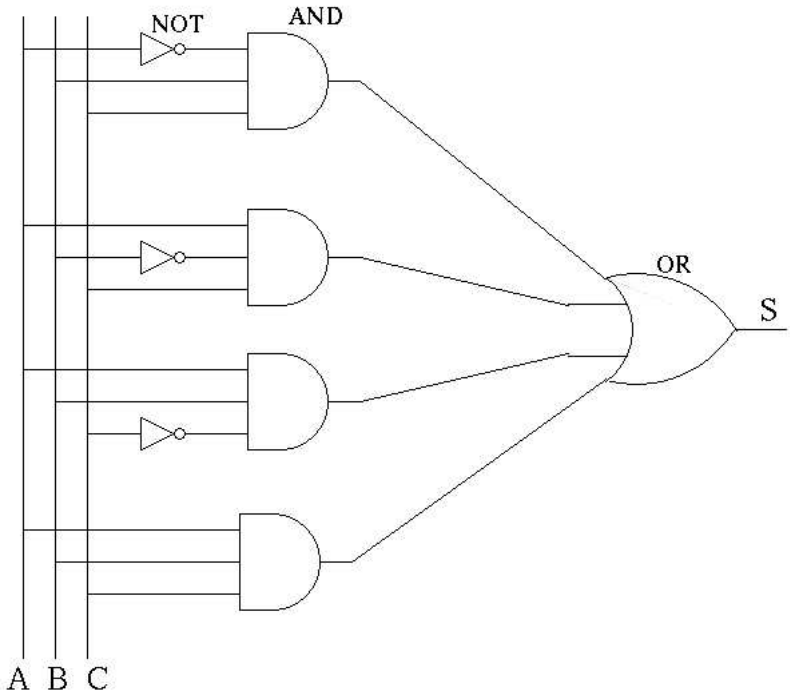


Figura 2: Exemplo de circuito clássico para voto majoritário

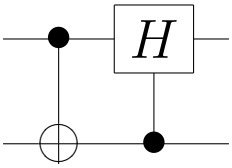
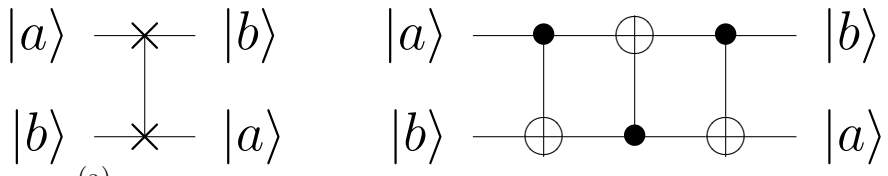


Figura 3: Porta NOT controlada (CNOT), com controle no primeiro qubit e alvo no segundo, seguida de uma porta Hadamard controlada, com controle no segundo qubit e alvo no primeiro

para representar portas controladas e *swaps* são exemplificadas pelas Figuras 3 e 4, respectivamente.

Com os elementos de Computação Quântica vistos até aqui, a passagem do circuito da Figura 2 para um equivalente quântico ainda não se torna evidente. Isto se dá pelo fato de a Mecânica Quântica ser reversível, tornando necessário que os circuitos quânticos sejam totalmente descritos através de portas lógicas reversíveis. As portas lógicas clássicas, em geral, não são reversíveis, pois nem sempre é possível identificar as entradas das mesmas com base em uma saída arbitrária. Um exemplo disso é a porta lógica AND: quando sua saída é 0, não é possível saber se as entradas foram $0-0$, $0-1$ ou $1-0$.



(a) Notação usual

(b) Utilizando três portas CNOT

Figura 4: Representação de uma porta swap

A	B	AND(A,B)	NAND(A,B)	OR(A,B)	NOR(A,B)	XOR(A,B)
0	0	0	1	0	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	0	1	0	0

Tabela 3: Principais operações lógicas sobre dois bits

Portanto, para um circuito clássico ser implementado em um computador quântico é necessário antes descrevê-lo de forma reversível. Um importante resultado na Computação Clássica foi a prova da possibilidade de construção de um computador reversível, por Bennett (1973). Mais tarde, Fredkin e Toffoli (1982) provaram que todo circuito clássico reversível pode ser descrito através de uma porta lógica universal, conhecida como porta de Fredkin. Como esta porta clássica possui equivalente quântico, — a porta de Toffoli — segue-se que qualquer circuito clássico pode ser implementado por computadores quânticos. O circuito da Figura 2, por exemplo, pode ser implementado através do circuito quântico da Figura 5. De fato, quando as entradas são estados projetados, a saída $|s\rangle$ é a idêntica à gerada pelo circuito clássico da Figura 2. Na prática, entretanto, isto não seria feito, pois o circuito quântico obtido desta forma não traria nenhum ganho de performance, e ainda seria muito mais complexo do ponto de vista tecnológico.

2.5 Paralelismo quântico

A propriedade que permite aos computadores realizarem cálculos distintos simultaneamente é chamada de paralelismo. Em computadores quânticos esta propriedade pode ser ilustrada através de um algoritmo muito simples, sem aplicações práticas, porém de grande importância conceitual (NIELSEN; CHUANG, 2000).

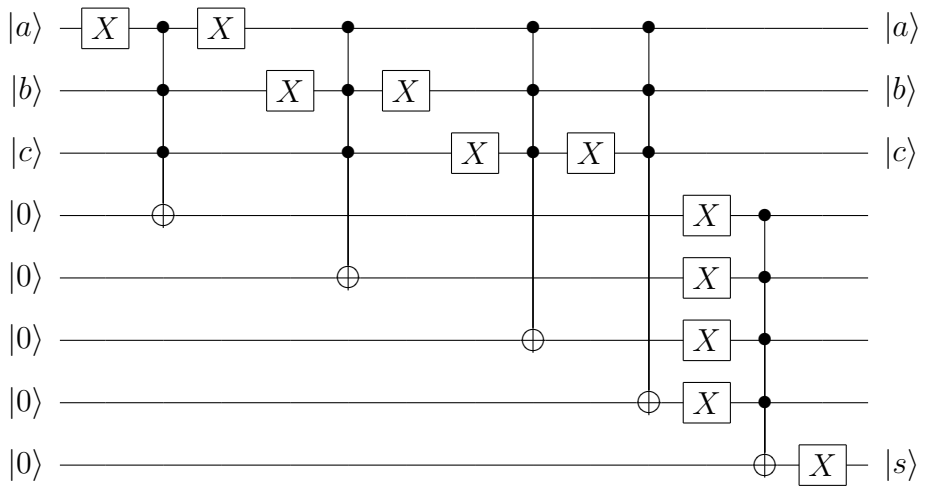


Figura 5: Exemplo de circuito quântico para voto majoritário

Inicialmente tem-se um registrador quântico $|\psi_0\rangle$ formado por dois *qubits* $|a\rangle$ e $|b\rangle$,

$$|\psi_0\rangle = |a\rangle\,|b\rangle\,. \tag{2.22}$$

Seja um operador unitário U, tal que

$$U\,|a\rangle\,|b\rangle = |a\rangle\,|b \oplus f(a)\rangle\,, \tag{2.23}$$

onde $\oplus$ denota soma módulo dois, $f(x) : \{0,1\} \to \{0,1\}$ e $a, b \in \{0,1\}$. Supondo que o valor inicial do *qubit* $|a\rangle$, na Equação (2.22), seja a superposição $\dfrac{|0\rangle + |1\rangle}{\sqrt{2}}$ e que $|b\rangle$ seja o estado da base computacional $|0\rangle$, tem-se

$$\begin{aligned} U\,|\psi_0\rangle &= U\frac{|0\rangle + |1\rangle}{\sqrt{2}}\,|0\rangle \\ &= \frac{|0\rangle\,|f(0)\rangle + |1\rangle\,|f(1)\rangle}{\sqrt{2}}. \end{aligned} \tag{2.24}$$

Portanto, a aplicação de uma porta lógica quântica permite a avaliação de uma função em pontos distintos simultaneamente — no exemplo, a função foi calculada nos pontos $x = 0$ e $x = 1$. Em um computador clássico seriam necessários múltiplos processadores ou execuções repetidas do algoritmo.

O paralelismo quântico, entretanto, esconde uma diferença sutil em relação ao seu correspondente clássico. De acordo com o terceiro postulado da Mecânica Quântica, os resultados obtidos, por exemplo, na Equação (2.24), não são totalmente acessíveis. De fato, quando o estado é medido obtém-se apenas o resultado

$f(0)$ ou $f(1)$, nunca os dois simultaneamente. Então, a princípio, pode-se ter a impressão de que o paralelismo quântico seja inútil, visto que o resultado do cálculo, apesar de ter sido obtido paralelamente, não torna-se totalmente disponível. Esta impressão demonstra-se falsa através de exemplos de utilização eficiente do paralelismo quântico, como no caso da Transformada de Fourier Quântica e seus subprodutos — o algoritmo para fatoração de inteiros, para cálculo de fase, para cálculo da ordem de grupo, dentre outros.

3 Transformada de Fourier Discreta Clássica

A DFT desempenha papel fundamental em diversos campos da ciência, como o processamento digital de sinais, a resolução de equações diferenciais parciais, bem como a multiplicação rápida de polinômios e inteiros grandes. Por muitos anos, entretanto, pensou-se que a DFT exigisse $O(N^2)$ operações, onde N é o número de pontos do vetor de entrada. Esta idéia parecia óbvia, já que a Transformada de Fourier pode ser vista como uma multiplicação de uma matriz $N \times N$ por um vetor.

A situação mudou drásticamente quando Cooley e Tukey (1965) descreveram um algoritmo para cálculo da DFT que realizava apenas $O(N \log N)$ operações. A este algoritmo, bem como às suas variantes, dá-se o nome de Transformada de Fourier Rápida (FFT). O trabalho de Cooley e Tukey, apesar de revolucionário, foi na verdade uma redescoberta de resultados anteriores, os quais curiosamente não chamaram a atenção da comunidade científica. De fato, apenas um ano após a publicação do artigo de Cooley e Tukey, Rudnick (1966) já apresentava um programa de computador para o cálculo da DFT, também de complexidade $O(N \log N)$, baseando-se em um método desenvolvido anteriormente por Danielson e Lanczos (1942). O trabalho de Danielson e Lanczos, por sua vez, baseava-se em trabalhos anteriores de Runge (1903, 1905). Até mesmo Gauss, no início do século XIX, já havia descoberto um método eficiente para cálculo de DFT em um caso particular.

Informações históricas mais detalhadas podem ser encontradas a partir do artigo de Cooley, Lewis e Welch (1967).

3.1 A transformada exata

Definição 3.1. As N-ésimas raízes complexas da unidade são os números complexos ω_N^k, $0 \leq k < N$, onde $\omega_N \equiv \exp\left(\frac{2\pi i}{N}\right)$ é chamado raiz principal da unidade.

Quando o contexto estiver claro pode-se omitir a indicação de N na raiz principal da unidade, de forma que $\omega \equiv \omega_N$. Antes de tratar da Transformada de Fourier Discreta, convém revisar algumas propriedades importantes das raízes da unidade. Em primeiro lugar, demonstra-se facilmente que a relação $\omega_{dn}^{dk} = \omega_n^k$ é válida para cada inteiro $k \geq 0$, $n > 0$ e $d > 0$. Como conseqüência, tem-se que $\omega_n^{n/2} = \omega_2 = -1$. Também é fácil demonstrar que, se $n > 0$ é par, então $(\omega_n^k)^2 = \omega_{n/2}^k$.

Lema 3.1. *Seja $\omega_N = \omega$ uma raiz principal da unidade. Então*

$$\sum_{k=0}^{N-1} \omega^{kj} = \begin{cases} N, & \text{se } j = 0 \pmod N \\ 0, & \text{caso contrário.} \end{cases} \tag{3.1}$$

Demonstração. Quando $j = 0 \pmod N$, basta substituir no lado esquerdo da Equação (3.1) e o resultado é imediato. Quando $j \neq 0 \pmod N$ tem-se que $\omega^j \neq 1$ e, portanto,

$$\begin{aligned} \sum_{k=0}^{N-1} \omega^{kj} &= \frac{1 - (\omega^N)^j}{1 - \omega^j} \\ &= 0. \end{aligned} \tag{3.2}$$

□

Ao longo desta dissertação convenciona-se que n seja um número inteiro positivo, e que a e c sejam inteiros de n bits. As representações binárias para a e c são denotadas respectivamente por

$$(a_{n-1}a_{n-2}\cdots a_0)_2 \text{ e } (c_{n-1}c_{n-2}\cdots c_0)_2,$$

de forma que

$$a = \sum_{j=0}^{n-1} a_j 2^j \text{ e } c = \sum_{j=0}^{n-1} c_j 2^j.$$

A indicação da base binária poderá ser omitida quando o contexto estiver claro. Diremos que a_s é o $(s+1)$-ésimo bit de um inteiro a, contando a partir do bit menos

significativo. A negação de um bit a_s (i.e., o *bit flip*) é aqui denotada por $\neg a_s$.

Sejam X e Y *arrays*[1] de números complexos, de comprimento $N = 2^n$. Estes *arrays* serão indexados pelos inteiros a e c, e a notação adotada será X_a. Freqüentemente o índice será utilizado na notação binária, como em $X_{(a_{n-1}a_{n-2}\cdots a_0)_2}$.

Definição 3.2 (DFT). A Transformada de Fourier Discreta toma como entrada um *array* de números complexos X e o converte em um *array* de números complexos Y, segundo a expressão

$$Y_c = \frac{1}{\sqrt{N}} \sum_{a=0}^{N-1} X_a \omega^{ac}. \tag{3.3}$$

Pode-se ainda reformular a Equação (3.3), utilizando as representações binárias de a e c. Reescrevendo ω^{ac}, obtém-se

$$Y_c = \frac{1}{\sqrt{N}} \sum_{a=0}^{N-1} X_a \exp\left(\frac{2\pi i}{N} \sum_{0 \le j,k \le n-1} a_j c_k 2^{j+k}\right). \tag{3.4}$$

Nota-se que

$$\omega^{(2^{j+k})} = \omega^{(2^n)2^{j+k-n}} = 1, \tag{3.5}$$

sempre que $j + k \geq n$, de modo que a Equação (3.4) pode ser reescrita como

$$Y_c = \frac{1}{\sqrt{N}} \sum_{a=0}^{N-1} X_a \exp\left(\frac{2\pi i}{N} \sum_{\substack{0 \le j,k \le n-1 \\ j+k \le n-1}} a_j c_k 2^{j+k}\right). \tag{3.6}$$

3.2 A transformada aproximada

Inicialmente, convém introduzir uma transformada muito importante, chamada Transformada de Hadamard.

Definição 3.3 (Transformada de Hadamard). A Transformada de Hadamard toma como entrada um *array* de números complexos X e o converte em um *array* de números complexos Y, de acordo com a expressão

$$Y_c = \frac{1}{\sqrt{N}} \sum_{a=0}^{N-1} X_a (-1)^{a \cdot c}, \tag{3.7}$$

onde $a \cdot c \equiv \sum_{j=0}^{n-1} a_j c_j$.

[1]Nesta dissertação o termo *array* irá sempre se referir ao conceito de estrutura de dados unidimensional.

Vamos mostrar que há semelhanças interessantes entre as Transformadas de Fourier e de Hadamard. Seja um inteiro b definido como a reversão de bits do inteiro c, isto é,

$$(b_{n-1}b_{n-2}\dots b_0)_2 \equiv (c_0c_1\dots c_{n-1})_2. \tag{3.8}$$

Utilizando a relação $\omega^{N/2} = -1$, e como $c_k = b_j$ para $j + k = n - 1$, pode-se fazer substituições na Equação (3.7), a fim de obter

$$\begin{aligned} Y_b &= \frac{1}{\sqrt{N}} \sum_{a=0}^{N-1} X_a (\omega^{N/2})^{a\cdot b} \\ &= \frac{1}{\sqrt{N}} \sum_{a=0}^{N-1} X_a \exp\left(\frac{2\pi i}{N} \sum_{0\le j\le n-1} a_j b_j 2^{n-1}\right) \\ &= \frac{1}{\sqrt{N}} \sum_{a=0}^{N-1} X_a \exp\left(\frac{2\pi i}{N} \sum_{\substack{0\le j,k\le n-1 \\ j+k=n-1}} a_j c_k 2^{j+k}\right). \end{aligned} \tag{3.9}$$

A Equação (3.9) define a Transformada de Hadamard indexada por b. A partir desta redefinição ainda é possível obter a Transformada de Hadamard usual através de uma simples reordenação dos elementos no *array* de saída.

Comparando as Equações (3.6) e (3.9) pode-se perceber que a única diferença entre a DFT e a Transformada de Hadamard indexada por b é o intervalo de $j + k$ no somatório. Na Transformada de Fourier o intervalo é $0 \le j + k \le n - 1$, enquanto na Transformada de Hadamard o intervalo é $n - 1 \le j + k \le n - 1$. Desta observação surge naturalmente uma nova transformada, parametrizada por um inteiro $1 \le m \le n$.

Definição 3.4 (DFT Aproximada). A Transformada de Fourier Aproximada, parametrizada por m, toma como entrada um *array* de números complexos X, e tem como resultado um *array* de números complexos Y, definidos pela equação

$$Y_c = \frac{1}{\sqrt{N}} \sum_{a=0}^{N-1} X_a \exp\left(\frac{2\pi i}{N} \sum_{\substack{0\le j,k\le n-1 \\ n-m\le j+k\le n-1}} a_j c_k 2^{j+k}\right). \tag{3.10}$$

Quando $m = 1$ a Transformada de Fourier Aproximada corresponde à Transformada de Hadamard indexada por b, e quando $m = n$, à DFT. Resta analisar o que ocorre quando o parâmetro m assume valores intermediários. Observando o

argumento na exponencial da Transformada de Fourier Aproximada e o argumento na exponencial da DFT, é fácil notar que a diferença entre os dois é dada por

$$i\epsilon = \frac{2\pi i}{N} \sum_{\substack{0 \leq j,k \leq n-1 \\ j+k<n-m}} a_j c_k 2^{j+k}. \tag{3.11}$$

Chamaremos esta diferença de erro.

Proposição 3.1. *A magnitude $|\epsilon|$ do erro descrito pela Equação (3.11) não é maior que $2\pi n 2^{-m}$.*

Demonstração. Basta notar que

$$\begin{aligned}
|\epsilon| &= \left| \frac{2\pi}{N} \sum_{\substack{0 \leq j,k \leq n-1 \\ j+k<n-m}} a_j c_j 2^{j+k} \right| \\
&\leq \frac{2\pi}{N} \sum_{0 \leq j \leq n-m} 2^j \sum_{0 \leq k < n-m-j} 2^k \\
&= \frac{2\pi}{N} \sum_{0 \leq j \leq n-m} 2^j \left(2^{n-m-j} - 1\right) \\
&= \frac{2\pi}{N} \left((n-m)2^{n-m} - 2^{n-m} + 1\right) \\
&\leq 2\pi n 2^{-m}.
\end{aligned}$$

□

Tanto a transformada exata quanto a aproximada podem, portanto, ser descritas através da multiplicação de uma matriz por um vetor, sendo que as entradas destas matrizes diferem entre si apenas por um fator multiplicativo $\exp(i\epsilon)$, onde $|\epsilon| \leq 2\pi n 2^{-m}$. Como $|\epsilon|$ decresce exponencialmente conforme m aumenta, tem-se que a Transformada de Fourier Aproximada fornece resultados muito próximos dos exatos, mesmo quando m não é tão próximo de n.

Corolário 3.1. *O menor valor de m necessário para garantir um erro de magnitude menor ou igual a $|\epsilon_{max}|$, na Transformada de Fourier Aproximada de um array com $N = 2^n$ elementos, é dado por*

$$m_{min} = \log \frac{2\pi}{|\epsilon_{max}|} + \log \log N. \tag{3.12}$$

Demonstração. Basta inverter a equação $|\epsilon_{max}| = 2\pi n 2^{-m}$ e substituir n por $\log N$. □

Portanto, fixando uma tolerância $|\epsilon_{max}|$ para os erros da Transformada de Fourier Aproximada, os resultados obtidos são satisfatórios quando um certo parâmetro maior ou igual a m_{min} é utilizado. Este parâmetro aumenta muito lentamente conforme aumenta o comprimento N do vetor de entrada.

Estes resultados foram utilizados por Coppersmith (1994) para desenvolver um algoritmo quântico eficiente para cálculo de DFT. Um discussão detalhada deste algoritmo será vista no Capítulo 4.

3.3 O algoritmo de Transformada de Fourier Rápida

Todos algoritmos conhecidos como FFT utilizam as propriedades das raízes da unidade, descritas no início do Capítulo, e calculam a DFT em tempo $O(N \log N)$, em vez do tempo $O(N^2)$ que seria obtido a partir da abordagem imediata. Este ganho de performance é obtido através de uma abordagem conhecida em Ciência da Computação como dividir-para-conquistar. Nesta abordagem, o problema inicial é dividido sucessivamente até que sejam atingidos cálculos elementares, quando então os resultados parciais são reagrupados para obtenção do resultado final. No caso da DFT, estas divisões são possíveis pelo Lema de Danielson-Lanczos. Análises detalhadas do algoritmo podem ser encontradas, por exemplo, em Cormen, Leiserson e Rivest (1990), Loan (1992), Meyer (2000), Oppenheim, Schafer e Buck (1999) e Press (1992).

Lema 3.2 (Danielson-Lanczos). *Sejam X e Y arrays de números complexos de N componentes obedecendo a Equação (3.3). Então*

$$Y_c = \frac{1}{\sqrt{2}} \left(Y_c^{par} + \omega_N^c Y_c^{impar} \right), \tag{3.13}$$

onde Y^{par} é o array resultante da DFT sobre um array de $N/2$ componentes, formado somente pelos termos de índices pares de X. Similarmente, Y^{impar} é definido pela DFT sobre o array formado pelos elementos de índices ímpares de X.

Demonstração. Basta notar que a Equação (3.3) pode ser reescrita como

$$\begin{aligned} Y_c &= \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{N/2}}\sum_{a=0}^{N/2-1} X_{2a}\omega_N^{2ac} + \frac{1}{\sqrt{N/2}}\sum_{a=0}^{N/2-1} X_{2a+1}\omega_N^{(2a+1)c}\right) \\ &= \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{N/2}}\sum_{a=0}^{N/2-1} X_{2a}\omega_N^{2ac} + \frac{\omega_N^c}{\sqrt{N/2}}\sum_{a=0}^{N/2-1} X_{2a+1}\omega_N^{2ac}\right). \end{aligned} \tag{3.14}$$

Agora, observando que $\omega_N^{2ac} = \omega_{N/2}^{ac}$ e fazendo

$$Y_c^{par} \equiv \frac{1}{\sqrt{N/2}}\sum_{a=0}^{N/2-1} X_{2a}\omega_{N/2}^{ac}, \tag{3.15}$$

$$Y_c^{impar} \equiv \frac{1}{\sqrt{N/2}}\sum_{a=0}^{N/2-1} X_{2a+1}\omega_{N/2}^{ac}, \tag{3.16}$$

recupera-se a Equação (3.13). □

Entrada: Um vetor $X \in \mathbb{C}^{2^n}$.
Saída: Um vetor $Y \in \mathbb{C}^{2^n}$ que é a Transformada de Fourier Discreta do vetor X.

1: {Inicialização, passo n}
2: **para todo** a tal que $0 \leq a \leq 2^n - 1$ **faça**
3: $\quad X^{(n)}_{(a_{n-1}a_{n-2}\ldots a_0)_2} \leftarrow X_{(a_{n-1}a_{n-2}\ldots a_0)_2}$.
4: **fim para**
5:
6: {Passo s}
7: **para** s de $n-1$ até 0, regressivamente **faça**
8: $\quad$ **para todo** $0 \leq b_{n-1}, \ldots, b_s, a_{s-1}, \ldots, a_0 \leq 1$ **faça**
9:

$$\begin{aligned} X^{(s)}_{(b_{n-1}\ldots b_s a_{s-1}\ldots a_0)_2} \leftarrow \frac{1}{\sqrt{2}} X^{(s+1)}_{(b_{n-1}\ldots b_{s+1}0a_{s-1}\ldots a_0)_2} + \\ + \frac{1}{\sqrt{2}}\omega^{(b_s b_{s+1}\ldots b_{n-1}0\ldots 0)_2} X^{(s+1)}_{(b_{n-1}\ldots b_{s+1}1a_{s-1}\ldots a_0)_2}. \end{aligned} \tag{3.17}$$

10: $\quad$ **fim para**
11: **fim para**
12:
13: {Reordenação}
14: **para todo** b tal que $0 \leq b \leq 2^n - 1$ **faça**
15: $\quad Y_{(b_{n-1}b_{n-2}\ldots b_0)_2} \leftarrow X^{(0)}_{(b_0 b_1 \ldots b_{n-1})_2}$.
16: **fim para**

Algoritmo 1: FFT clássica

O Lema de Danielson-Lanczos revela que para calcular Y, a DFT de um ve-

tor X, basta calcular duas DFTs sobre vetores de comprimento igual à metade do comprimento inicial. Cada um destes resultados parciais, Y^{par} e Y^{impar}, pode ser obtido através de nova aplicação do Lema 3.2. Isto leva ao cálculo de quatro novas DFTs, $Y^{par,par}$, $Y^{par,impar}$, $Y^{impar,par}$ e $Y^{impar,impar}$, cada uma sobre vetores de comprimento igual a um quarto do comprimento inicial. Repetindo este processo recursivamente, chega-se finalmente a vetores de comprimento um, cujas DFTs são a própria identidade — basta notar que na Equação (3.3), quando $N = 1$, obtém-se $Y = X$.

Pode-se, agora, analisar a complexidade computacional dos algoritmos FFT. Denotando por T_{2^n} o número aproximado de passos computacionais realizados pelo algoritmo, quando este calcula a DFT sobre um vetor de comprimento 2^n, pode-se escrever

$$T_{2^n} = 2T_{2^{n-1}} + 2^n, \tag{3.18}$$

para $n \geq 1$, com $T_1 = 0$. O primeiro termo do lado direito da Equação (3.18) refere-se aos passos executados no cálculo de duas DFTs, cada uma sobre vetores com 2^{n-1} elementos; e o segundo termo refere-se aos passos necessários para reagrupar os dois vetores resultantes de acordo com a Equação (3.13). Então,

$$\begin{aligned} \frac{T_{2^n}}{2^n} &= \frac{T_{2^{n-1}}}{2^{n-1}} + 1 \\ &= \frac{1}{2^{n-1}}\left(2T_{2^{n-2}} + 2^{n-1}\right) + 1 \\ &= \frac{T_{2^{n-2}}}{2^{n-2}} + 1 + 1 \\ &\vdots \\ &= n. \end{aligned} \tag{3.19}$$

Logo, tem-se $T_{2^n} = n2^n$, o que significa que os algoritmos FFT possuem complexidade $O(n2^n)$ ou, equivalentemente, $O(N \log N)$.

Um algoritmo FFT é dado pelo Algoritmo 1, sendo este o mesmo descrito no livro de Knuth (1981), com pequenas modificações em aspectos não-essenciais como notação e normalização. Denota-se por $X^{(s)}$ o estado do vetor em um passo s do cálculo.

Proposição 3.2 (Corretude do algoritmo FFT). *O vetor Y produzido pelo Algoritmo 1 é a DFT do vetor X, conforme Definição 3.2.*

Demonstração. Deseja-se demonstrar, por indução, que a relação

$$X^{(s)}_{(b_{n-1}b_{n-2}\ldots b_s a_{s-1}\ldots a_0)_2} = \frac{1}{\sqrt{2^{n-s}}} \sum_{0\leq a_{n-1},a_{n-2},\ldots a_s\leq 1} X_{(a_{n-1}\ldots a_0)_2}\omega^{(a_{n-1}\ldots a_s 0\ldots 0)_2(b_0 b_1\ldots b_{n-1})_2} \quad (3.20)$$

é válida para qualquer passo $0 \leq s < n$ do Algoritmo 1, supondo que no passo $s = n$ tenhamos $X^{(n)}_a = X_a$.

Primeiramente, nota-se que

$$\begin{aligned}(a_{n-1}0\ldots 0)_2(b_0 b_1\ldots b_{n-1})_2 &= 2^{n+1}\sum_{j=0}^{n-1} b_j 2^{n-1-j} \\ &= b_{n-1}2^{n-1} + \sum_{j=0}^{n-2} b_j 2^{n-1-j}2^{n-1}. \quad (3.21)\end{aligned}$$

Como $\sum_{j=0}^{n-2} b_j 2^{n-1-j}2^{n-1} \equiv 0 \pmod{2^n}$, tem-se que $\omega^{(a_{n-1}0\ldots 0)(b_0 b_1\ldots b_{n-1})} = \omega^{(b_{n-1}0\ldots 0)}$. Portanto, é fácil perceber que a Equação (3.20) é válida quando $s = n-1$:

$$\begin{aligned}&\frac{1}{\sqrt{2}} \sum_{0\leq a_{n-1}\leq 1} X_{(a_{n-1}\ldots a_0)}\omega^{(a_{n-1}0\ldots 0)(b_0 b_1\ldots b_{n-1})} \\ &\qquad = \frac{1}{\sqrt{2}} X^{(n)}_{(0\ a_{n-2}\ldots a_0)} + \frac{1}{\sqrt{2}} X^{(n)}_{(1\ a_{n-2}\ldots a_0)}\omega^{(b_{n-1}0\ldots 0)} \\ &\qquad = X^{(n-1)}_{(b_{n-1}a_{n-2}\ldots a_0)}. \quad (3.22)\end{aligned}$$

Em seguida, supondo que a Equação (3.20) seja válida para um passo arbitrário $s+1$, conclui-se que ela também é válida para o passo seguinte, s. Basta notar que

$$\begin{aligned}&\frac{1}{\sqrt{2^{n-s}}} \sum_{0\leq a_{n-1},\ldots,a_s\leq 1} X_{(a_{n-1}\ldots a_0)}\omega^{(a_{n-1}\ldots a_s 0\ldots 0)(b_0 b_1\ldots b_{n-1})} \\ &\qquad = \frac{1}{\sqrt{2^{n-s}}} \sum_{\substack{0\leq a_{n-1},\ldots,a_{s+1}\leq 1 \\ a_s=0}} X_{(a_{n-1}\ldots a_0)}\omega^{(a_{n-1}\ldots a_{s+1}0\ldots 0)(b_0 b_1\ldots b_{n-1})} + \\ &+ \frac{1}{\sqrt{2^{n-s}}} \sum_{\substack{0\leq a_{n-1},\ldots,a_{s+1}\leq 1 \\ a_s=1}} X_{(a_{n-1}\ldots a_0)}\omega^{(a_{n-1}\ldots a_{s+1}0\ldots 0)(b_0 b_1\ldots b_{n-1})}\omega^{(b_s b_{s+1}\ldots b_{n-1}0\ldots 0)}, \quad (3.23)\end{aligned}$$

onde utilizou-se a relação

$$\omega^{(a_{n-1}\ldots a_{s+1}1\ldots 0)_2(b_0 b_1\ldots b_{n-1})_2} = \omega^{(a_{n-1}\ldots a_{s+1}0\ldots 0)_2(b_0 b_1\ldots b_{n-1})_2}\omega^{(b_s b_{s+1}\ldots b_{n-1}0\ldots 0)_2}, \quad (3.24)$$

análoga à do passo anterior.

Por hipótese, o primeiro termo do lado direito da equação é o vetor $X^{(s+1)}$ avaliado em todos os índices onde $a_s = 0$. Analogamente, o segundo termo contém o vetor $X^{(s+1)}$ avaliado em todos os índices onde $a_s = 1$. Portanto, reescrevendo a Equação (3.23), obtém-se

$$\frac{1}{\sqrt{2}} X^{(s+1)}_{(b_{n-1}\ldots b_{s+1}\ 0\ a_{s-1}\ldots a_o)} + \frac{1}{\sqrt{2}} X^{(s+1)}_{(b_{n-1}\ldots b_{s+1}\ 1\ a_{s-1}\ldots a_0)} \omega^{(b_s b_{s+1}\ldots b_{n-1} 0\ldots 0)} = X^{(s)}_{(b_{n-1}\ldots b_s a_{s-1}\ldots a_0)}. \quad (3.25)$$

Fazendo $s = 0$ na Equação (3.20), correspondendo ao último passo do Algoritmo 1, conclui-se que a expressão de fato produz a DFT definida na Equação (3.3), exceto pelo uso do índice b — com bits revertidos — onde deveria ser c. Isto é facilmente corrigido através da reordenação, descrita no final do algoritmo. □

Agora que um exemplo concreto de algoritmo FFT foi apresentado no Algoritmo 1, sua complexidade pode ser analisada da seguinte forma. Em primeiro lugar, nota-se que a inicialização requer 2^n operações, assim como a reordenação no final do algoritmo. A parte principal do algoritmo consiste em dois laços de repetição. O laço externo repete n vezes. Já o laço interno repete 2^n vezes, realizando sempre uma operação de atribuição, a qual pode ser considerada $O(1)$. Portanto, o algoritmo tem complexidade $O(n2^n)$. Como será visto no Capítulo 4, o algoritmo quântico correspondente tem complexidade computacional $O(n^2)$.

Também será apresentado no Capítulo 4 um algoritmo para o cálculo da DFT Aproximada parametrizada por m, segundo a Definição 3.4. O algoritmo clássico aproximado (Algoritmo 3) não apresenta ganhos de complexidade em relação ao algoritmo exato (Algoritmo 1). No entanto, ele será utilizado como passo intermediário para obtenção do algoritmo quântico correspondente, de complexidade computacional $O(mn)$.

4 Transformada de Fourier Discreta Quântica

4.1 Exemplo de execução da FFT

Neste Capítulo será visto como o Algoritmo 1 pode ser convertido em algoritmo quântico. Antes disso, será considerado o caso particular em que $n = 3$, de modo que a transformada será sobre um vetor de $2^3 = 8$ entradas complexas. Ao longo desta exposição, serão feitos comentários referentes à representação matricial das operações de cada passo do algoritmo.

4.1.1 Inicialização

No passo de inicialização são realizadas as seguintes operações de atribuição:

$$\begin{aligned}
X^{(3)}_{(000)_2} &\leftarrow X_{(000)_2}, & X^{(3)}_{(001)_2} &\leftarrow X_{(001)_2},\\
X^{(3)}_{(010)_2} &\leftarrow X_{(010)_2}, & X^{(3)}_{(011)_2} &\leftarrow X_{(011)_2},\\
X^{(3)}_{(100)_2} &\leftarrow X_{(100)_2}, & X^{(3)}_{(101)_2} &\leftarrow X_{(101)_2},\\
X^{(3)}_{(110)_2} &\leftarrow X_{(110)_2}, & X^{(3)}_{(111)_2} &\leftarrow X_{(111)_2}.
\end{aligned} \tag{4.1}$$

Em um computador quântico, este passo corresponderia à preparação do sistema. Poderia ser, por exemplo, a preparação do estado

$$\begin{aligned}
|\psi_3\rangle &= \frac{1}{\sqrt{2^n}} \sum_{0\leq a_2,a_1,a_0\leq 1} X^{(3)}_{(a_2a_1a_0)_2} |a_2a_1a_0\rangle \\
&= \frac{1}{\sqrt{8}} \sum_{a=0}^{7} X_a |a_2a_1a_0\rangle.
\end{aligned} \tag{4.2}$$

Não se conhece método eficiente para preparar um estado quântico arbitrário. En-

tretanto, as aplicações da Transformada de Fourier Quântica tomam sempre como entrada um estado da base computacional, ou um estado já preparado por uma etapa anterior de um algoritmo. Portanto, a inicialização não impõe nenhuma dificuldade para a execução do algoritmo quântico.

Matematicamente, o estado do sistema em um passo s, denotado por $|\psi_s\rangle$, pode ser representado por um vetor coluna,

$$|\psi_s\rangle \equiv \frac{1}{\sqrt{8}} \left(X^{(s)}_{(000)_2}, X^{(s)}_{(001)_2}, \ldots, X^{(s)}_{(111)_2} \right)^T . \tag{4.3}$$

4.1.2 Passo dois

As operações realizadas no passo $s = 2$ do Algoritmo 1 são as seguintes:

$$\begin{aligned}
X^{(2)}_{(000)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(3)}_{(\mathbf{0}00)} + \omega^{(000)_2} X^{(3)}_{(\mathbf{1}00)_2} \right), \\
X^{(2)}_{(001)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(3)}_{(\mathbf{0}01)} + \omega^{(000)_2} X^{(3)}_{(\mathbf{1}01)_2} \right), \\
X^{(2)}_{(010)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(3)}_{(\mathbf{0}10)} + \omega^{(000)_2} X^{(3)}_{(\mathbf{1}10)_2} \right), \\
X^{(2)}_{(011)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(3)}_{(\mathbf{0}11)} + \omega^{(000)_2} X^{(3)}_{(\mathbf{1}11)_2} \right), \\
X^{(2)}_{(100)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(3)}_{(\mathbf{0}00)} + \omega^{(100)_2} X^{(3)}_{(\mathbf{1}00)_2} \right), \\
X^{(2)}_{(101)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(3)}_{(\mathbf{0}01)} + \omega^{(100)_2} X^{(3)}_{(\mathbf{1}01)_2} \right), \\
X^{(2)}_{(110)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(3)}_{(\mathbf{0}10)} + \omega^{(100)_2} X^{(3)}_{(\mathbf{1}10)_2} \right), \\
X^{(2)}_{(111)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(3)}_{(\mathbf{0}11)} + \omega^{(100)_2} X^{(3)}_{(\mathbf{1}11)_2} \right).
\end{aligned} \tag{4.4}$$

Percebe-se facilmente que as operações do passo $s = 2$ podem ser representadas

pela matriz

$$P^{(2)} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & \omega^4 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \omega^4 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \omega^4 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \omega^4 \end{pmatrix} \tag{4.5}$$

atuando no vetor coluna que representa o estado inicial do sistema — Equação (4.3), com $s = 3$. O leitor pode notar ainda que a matriz é unitária, de forma que, ao menos em princípio, poderia ser implementada em um computador quântico. De fato, observando que $\omega^4 = -1$, tem-se que

$$P^{(2)} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{4.6}$$

4.1.3 Passo um

As operações realizadas no passo $s = 1$ do Algoritmo 1 são as seguintes:

$$\begin{aligned}
X^{(1)}_{(000)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(2)}_{(0\mathbf{0}0)_2} + \omega^{(000)_2} X^{(2)}_{(0\mathbf{1}0)_2} \right) \\
X^{(1)}_{(001)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(2)}_{(0\mathbf{0}1)_2} + \omega^{(000)_2} X^{(2)}_{(0\mathbf{1}1)_2} \right) \\
X^{(1)}_{(010)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(2)}_{(0\mathbf{0}0)_2} + \omega^{(100)_2} X^{(2)}_{(0\mathbf{1}0)_2} \right) \quad \bullet \\
X^{(1)}_{(011)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(2)}_{(0\mathbf{0}1)_2} + \omega^{(100)_2} X^{(2)}_{(0\mathbf{1}1)_2} \right) \quad \bullet \\
X^{(1)}_{(100)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(2)}_{(1\mathbf{0}0)_2} + \omega^{(010)_2} X^{(2)}_{(1\mathbf{1}0)_2} \right) \\
X^{(1)}_{(101)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(2)}_{(1\mathbf{0}1)_2} + \omega^{(010)_2} X^{(2)}_{(1\mathbf{1}1)_2} \right) \\
X^{(1)}_{(110)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(2)}_{(1\mathbf{0}0)_2} + \omega^{(110)_2} X^{(2)}_{(1\mathbf{1}0)_2} \right) \quad \bullet \\
X^{(1)}_{(111)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(2)}_{(1\mathbf{0}1)_2} + \omega^{(110)_2} X^{(2)}_{(1\mathbf{1}1)_2} \right) \quad \bullet
\end{aligned} \tag{4.7}$$

Este passo do cálculo é equivalente à aplicação da matriz

$$P^{(1)} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & \omega^0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \omega^0 & 0 & 0 & 0 & 0 \\ 1 & 0 & \omega^4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \omega^4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & \omega^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \omega^2 \\ 0 & 0 & 0 & 0 & 1 & 0 & \omega^6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \omega^6 \end{pmatrix}, \tag{4.8}$$

ao vetor coluna obtido pelo passo $s = 2$. No contexto quântico, esta matriz é muito complexa para ser implementada por um único processo físico. Deve-se, portanto, utilizar alguma decomposição adequada ao desenvolvimento de circuitos quânticos, e obter matrizes mais simples. Mais adiante, quando for feita a generalização da QFT, será mostrado que a decomposição QR é bastante útil para este propósito. Por enquanto, serão apresentados apenas os resultados para o exemplo, a saber,

$$\begin{aligned} P^{(1)} &= M^{(1)} N^{(1)} \\ &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \omega^0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \omega^0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \omega^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \omega^2 \end{pmatrix}. \end{aligned} \tag{4.9}$$

Estas matrizes unitárias devem ser aplicadas da direita para a esquerda sobre o vetor coluna obtido no passo $s = 2$. Não é difícil ver que a matriz $N^{(1)}$, está, na verdade, realizando as seguintes operações:

$$\begin{aligned} X_{(010)_2} &\leftarrow \omega^{(000)_2} X^{(2)}_{(010)_2}, \\ X_{(011)_2} &\leftarrow \omega^{(000)_2} X^{(2)}_{(011)_2}, \\ X_{(110)_2} &\leftarrow \omega^{(010)_2} X^{(2)}_{(110)_2}, \\ X_{(111)_2} &\leftarrow \omega^{(010)_2} X^{(2)}_{(111)_2}. \end{aligned} \tag{4.10}$$

Comparando estas operações com as linhas marcadas com • no começo do passo $s = 1$, pode-se resumi-las por

$$X_{(b_2 b_1 a_0)_2} \leftarrow \omega^{(0 b_2 0)_2} X^{(2)}_{(b_2 b_1 a_0)}, \tag{4.11}$$

para $0 \leq b_2, a_0 \leq 1$ e $b_1 = 1$. Quando $b_2 = 0$, este procedimento deixa o estado inalterado. Portanto, a matriz multiplica por $\omega^{(0 b_2 0)_2}$ todos os componentes do *array* cujos índices possuem simultaneamente $b_1 = 1$ e $b_2 = 1$. Será visto mais adiante que a matriz $N^{(1)}$ corresponde a uma porta lógica quântica controlada.

Para completar o passo um deve-se finalmente aplicar a matriz $M^{(1)}$. Verifica-se que

$$M^{(1)} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{4.12}$$

Estas matrizes são facilmente implementáveis como portas lógicas quânticas, correspondendo a uma porta de Hadamard (vide Tabela 2) atuando no *qubit* 1.

4.1.4 Passo zero

As operações realizadas no passo $s = 0$ do Algoritmo 1 são as seguintes:

$$\begin{aligned}
X^{(0)}_{(000)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(1)}_{(00\mathbf{0})} + \omega^{(000)_2} X^{(1)}_{(00\mathbf{1})} \right) \\
X^{(0)}_{(001)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(1)}_{(00\mathbf{0})} + \omega^{(100)_2} X^{(1)}_{(00\mathbf{1})_2} \right) \quad \blacklozenge \\
X^{(0)}_{(010)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(1)}_{(01\mathbf{0})} + \omega^{(010)_2} X^{(1)}_{(01\mathbf{1})_2} \right) \\
X^{(0)}_{(011)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(1)}_{(01\mathbf{0})} + \omega^{(110)_2} X^{(1)}_{(01\mathbf{1})_2} \right) \quad \blacklozenge \\
X^{(0)}_{(100)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(1)}_{(10\mathbf{0})} + \omega^{(001)_2} X^{(1)}_{(10\mathbf{1})_2} \right) \\
X^{(0)}_{(101)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(1)}_{(10\mathbf{0})} + \omega^{(101)_2} X^{(1)}_{(10\mathbf{1})_2} \right) \quad \blacklozenge \\
X^{(0)}_{(110)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(1)}_{(11\mathbf{0})} + \omega^{(011)_2} X^{(1)}_{(11\mathbf{1})_2} \right) \\
X^{(0)}_{(111)_2} &\leftarrow \frac{1}{\sqrt{2}} \left(X^{(1)}_{(11\mathbf{0})} + \omega^{(111)_2} X^{(1)}_{(11\mathbf{1})_2} \right) \quad \blacklozenge
\end{aligned} \tag{4.13}$$

Este último passo da FFT pode ser representado pela matriz

$$P^{(0)} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & \omega^0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & \omega^4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \omega^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \omega^6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \omega & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \omega^5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \omega^3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \omega^7 \end{pmatrix} \tag{4.14}$$

atuando sobre um vetor coluna obtido após o passo $s = 1$. Novamente, esta matriz é muito complexa para ser diretamente implementada em um único processo físico quântico. Decompondo a matriz, os fatores obtidos são

$$\begin{aligned} P^{(0)} &= M^{(0)} N^{(0)} \\ &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \omega^0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \omega^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \omega & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \omega^3 \end{pmatrix}. \end{aligned} \tag{4.15}$$

Primeiramente é necessário aplicar a matriz $N^{(0)}$. Esta matriz pode ser analisada da mesma forma que a matriz $N^{(1)}$, da seção anterior. Desta vez, as operações relevantes estão marcadas com ♦ no início do passo zero. Através da comparação destas linhas com as da matriz $N^{(0)}$, pode-se resumi-las como

$$\begin{aligned} X_{(b_2 b_1 b_0)_2} &\leftarrow \omega^{(0 b_1 b 2)_2} X^{(2)}_{(b_2 b_1 b_0)_2} && (4.16) \\ &= \omega^{(0 b_1 0)_2} \omega^{(00 b 2)_2} X^{(2)}_{(b_2 b_1 b_0)_2}, && (4.17) \end{aligned}$$

para $0 \leq b_2, b_1 \leq 1$ e $b_0 = 1$.

Portanto, a matriz corresponde a multiplicar por $\omega^{(0 b_1 0)_2}$ todas as linhas do vetor cujos índices possuem simultaneamente $b_1 = 1$ e $b_0 = 1$; e por $\omega^{(00 b 2)_2}$ todas as linhas

do vetor cujos índices possuem simultaneamente $b_2 = 1$ e $b_0 = 1$. Mais adiante, será visto que estas matrizes correspondem a duas portas lógicas quânticas controladas.

Concluindo o passo $s = 0$, pode-se verificar facilmente que

$$M^{(0)} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \tag{4.18}$$

correspondendo, em Computação Quântica, à aplicação de uma porta de Hadamard (vide Tabela 2) no *qubit* menos significativo.

4.1.5 Reordenação

A etapa final da DFT move cada elemento do vetor coluna para a linha cujo índice seja o índice original com bits revertidos:

$$\begin{aligned} Y_{(000)_2} &\leftarrow X^{(0)}_{(000)_2} & Y_{(001)_2} &\leftarrow X^{(0)}_{(100)_2} \\ Y_{(010)_2} &\leftarrow X^{(0)}_{(010)_2} & Y_{(011)_2} &\leftarrow X^{(0)}_{(110)_2} \\ Y_{(100)_2} &\leftarrow X^{(0)}_{(001)_2} & Y_{(101)_2} &\leftarrow X^{(0)}_{(101)_2} \\ Y_{(110)_2} &\leftarrow X^{(0)}_{(011)_2} & Y_{(111)_2} &\leftarrow X^{(0)}_{(111)_2} \end{aligned} \tag{4.19}$$

A matriz para esta reordenação é

$$A^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.20}$$

Ficará claro durante o processo de generalização que a matriz $A^{(3)}$, além de unitária, corresponde a uma operação de *swap* da Computação Quântica.

Aplicando a seqüência de portas lógicas encontradas nesta seção, seria possível realizar a DFT em um vetor de números complexos cujas entradas fossem representadas através das amplitudes de um estado quântico de três *qubits*.

4.2 Da FFT clássica à quântica

No Capítulo anterior foi descrito um algoritmo clássico para o cálculo da DFT (Algoritmo 1). Baseando-se naquele algoritmo e tratando $X^{(s)}$ como um vetor coluna, pode-se descrever cada passo s como uma multiplicação de uma matriz por uma coluna, de forma que

$$X_j^{(s)} = \sum_{0 \leq k \leq N-1} P_{jk}^{(s)} X_k^{(s+1)}, \tag{4.21}$$

onde $P^{(s)}$ é uma matriz $N \times N$ para $0 \leq s \leq n-1$. De fato, as matrizes $P^{(s)}$ foram encontradas, na Seção 4.1, para um caso particular onde o vetor X possuía oito elementos. Agora, as matrizes serão generalizadas de forma a obter um algoritmo quântico para um número arbitrário de *qubits*, correspondendo ao algoritmo de Coppersmith (1994). Ao longo desta dissertação convenciona-se numerar de 0 até $N-1$ as linhas e colunas das matrizes, bem como as entradas dos vetores.

Observando a Equação (3.17), do Algoritmo 1, percebe-se que qualquer linha de um vetor coluna $X^{(s)}$ depende somente de duas linhas de $X^{(s+1)}$. Logo, cada linha das matrizes $P^{(s)}$ possui apenas duas entradas diferentes de zero. Estas entradas estão localizadas nas colunas $k = j - j_s 2^s$ e $k = j + (1 - j_s)2^s$. Portanto, cada entrada não-nula das matrizes $P^{(s)}$ situa-se na diagonal principal ou em uma subdiagonal, dependendo do valor do bit j_s de j.

Quando $k = j$, são obtidas as entradas da diagonal principal das matrizes. Se $j_s = 0$, então a observação do primeiro termo do lado direito da Equação (3.17) revela que estas entradas são $\frac{1}{\sqrt{2}}$. Se $j_s = 1$, então o segundo termo mostra que as entradas são $\frac{1}{\sqrt{2}}\omega^{(j_s j_{s+1} \cdots j_{n-1} 0 \cdots 0)_2}$.

Quando $k = j - 2^s$, são obtidas as entradas de uma subdiagonal inferior das matrizes. Se $j_s = 0$, então as entradas são iguais a zero, também de acordo com a Equação (3.17). Se $j_s = 1$, então a observação do primeiro termo do lado direito da equação permite concluir que estas entradas são iguais a $\frac{1}{\sqrt{2}}$.

Quando $k = j + 2^s$, são obtidas as entradas de uma subdiagonal superior das matrizes. Se $j_s = 0$, então a observação do segundo termo do lado direito da Equação (3.17) permite concluir que estas entradas são $\frac{1}{\sqrt{2}}\omega^{(j_s j_{s+1} \cdots j_{n-1} 0 \cdots 0)_2}$. Se $j_s = 1$, então as entradas são iguais a zero, também de acordo com a Equação (3.17).

Convém agora definir o conceito de fração binária como

$$0.j \equiv 0.j_0 j_1 \cdots j_{n-1} = \sum_{0 \leq t \leq n-1} \frac{j_t}{2^{t+1}}, \tag{4.22}$$

de modo que $\omega^{(j_s j_{s+1} \cdots j_{n-1} 0 \cdots 0)_2}$ possa ser reescrito de forma mais compacta, como $\omega^{(0.j)2^{n+s}}$. Para $x \in \mathbb{R}$, denota-se por $\lfloor x \rfloor$ o maior número inteiro menor ou igual a x. Observando que $\left\lfloor \frac{j}{2^s} \right\rfloor = (j_{n-1} j_{n-2} \cdots j_s)_2$, verifica-se que

$$j_s = \frac{1-(-1)^{\left\lfloor \frac{j}{2^s} \right\rfloor}}{2}, \tag{4.23}$$

e que

$$\neg j_s = \frac{1+(-1)^{\left\lfloor \frac{j}{2^s} \right\rfloor}}{2}, \tag{4.24}$$

de forma que as observações dos parágrafos anteriores podem ser resumidas através das matrizes genéricas

$$P_{jk}^{(s)} = \frac{1}{\sqrt{2}} \begin{cases} \omega^{j_s (0.j) 2^{n+s}}, & \text{se } k = j \\ \frac{1-(-1)^{\lfloor \frac{j}{2^s} \rfloor}}{2}, & \text{se } k = j - 2^s \\ \frac{1+(-1)^{\lfloor \frac{j}{2^s} \rfloor}}{2} \omega^{(0.j)2^{n+s}}, & \text{se } k = j + 2^s \\ 0, & \text{caso contrário.} \end{cases} \tag{4.25}$$

Estas matrizes representam os passos do Algoritmo 1. No Apêndice B são fornecidos códigos em Maple que podem ser utilizados para visualizar casos particulares das matrizes $P^{(s)}$, bem como das demais matrizes que ainda serão introduzidas ao longo deste Capítulo.

Na tentativa de converter o algoritmo clássico em quântico, a primeira etapa consiste em representá-lo matricialmente. Entretanto, decorre do segundo postulado da Mecânica Quântica que as matrizes $P^{(s)}$ somente poderão ser expressas em termos de portas lógicas quânticas se forem unitárias. De fato,

Proposição 4.1. *As matrizes $P^{(s)}$ são unitárias.*

Demonstração. Pode-se resolver

$$\left(P^{(s)}P^{(s)^\dagger}\right)_{jk} = \sum_{0\leq l\leq N-1} P^{(s)}_{jl}P^{(s)^*}_{kl}. \tag{4.26}$$

Quando $k = j$, tem-se que

$$\begin{aligned}\left(P^{(s)}P^{(s)^\dagger}\right)_{jj} &= P^{(s)}_{jj}P^{(s)^*}_{jj} + P^{(s)}_{j,j-2^s}P^{(s)^*}_{j,j-2^s} + P^{(s)}_{j,j+2^s}P^{(s)^*}_{j,j+2^s} \\ &= \frac{1}{2} + \frac{1}{2}\left(\frac{1-(-1)^{\left\lfloor\frac{j}{2^s}\right\rfloor}}{2}\right)^2 + \frac{1}{2}\left(\frac{1+(-1)^{\left\lfloor\frac{j}{2^s}\right\rfloor}}{2}\right)^2 \\ &= \frac{1}{2} + \frac{1}{2}j_s + \frac{1}{2}(\neg j_s) \\ &= 1. \end{aligned} \tag{4.27}$$

Para o restante da demonstração, convém ressaltar que $\omega^{(0.j)2^{n+s}} = -\omega^{(0.k)2^{n+s}}$ sempre que $k = j \pm 2^s$.

Quando $k = j - 2^s$, tem-se que

$$\begin{aligned}\left(P^{(s)}P^{(s)^\dagger}\right)_{j,j-2^s} &= P^{(s)}_{jj}P^{(s)^*}_{j-2^s,j} + P^{(s)}_{j,j-2^s}P^{(s)^*}_{j-2^s,j-2^s} + P^{(s)}_{j,j+2^s}P^{(s)^*}_{j-2^s,j+2^s} \\ &= \frac{1}{\sqrt{2}}\omega^{j_s(0.j)2^{n+s}}\frac{1+(-1)^{\left\lfloor\frac{j-2^s}{2^s}\right\rfloor}}{2\sqrt{2}}\omega^{*(0.k)2^{n+s}} + \\ &\quad + \frac{1-(-1)^{\left\lfloor\frac{j}{2^s}\right\rfloor}}{2\sqrt{2}}\frac{1}{\sqrt{2}}\omega^{*k_s(0.k)2^{n+s}} \\ &= \frac{j_s}{2}\omega^{j_s(0.j)2^{n+s}}\omega^{*(0.k)2^{n+s}} + \frac{j_s}{2}\omega^{*k_s(0.k)2^{n+s}}. \end{aligned} \tag{4.28}$$

O primeiro termo do lado direito da Equação (4.28) somente é diferente de zero quando $j_s = 1$. Neste caso, tem-se que $\omega^{j_s(0.j)2^{n+s}}\omega^{*(0.k)2^{n+s}} = -1$. Portanto,

$$\begin{aligned}\left(P^{(s)}P^{(s)^\dagger}\right)_{j,j-2^s} &= -\frac{j_s}{2} + \frac{j_s}{2}\omega^{*k_s(0.k)2^{n+s}} \\ &= 0. \end{aligned} \tag{4.29}$$

Quando $k = j + 2^s$, tem-se que

$$\begin{aligned}
\left(P^{(s)}P^{(s)\dagger}\right)_{j,j+2^s} &= P^{(s)}_{jj}P^{(s)*}_{j+2^s,j} + P^{(s)}_{j,j-2^s}P^{(s)*}_{j+2^s,j-2^s} + P^{(s)}_{j,j+2^s}P^{(s)*}_{j+2^s,j+2^s} \\
&= \frac{1}{\sqrt{2}}\omega^{j_s(0.j)2^{n+s}}\frac{1-(-1)^{\left\lfloor\frac{j+2^s}{2^s}\right\rfloor}}{2\sqrt{2}} + \\
&\quad + \frac{1+(-1)^{\left\lfloor\frac{j}{2^s}\right\rfloor}}{2\sqrt{2}}\omega^{j_s(0.j)2^{n+s}}\frac{1}{\sqrt{2}}\omega^{*k_s(0.k)2^{n+s}} \\
&= \frac{\neg j_s}{2}\omega^{j_s(0.j)2^{n+s}} + \frac{\neg j_s}{2}\omega^{j_s(0.j)2^{n+s}}\omega^{*k_s(0.k)2^{n+s}} \qquad (4.30)
\end{aligned}$$

O segundo termo do lado direito da Equação (4.30) somente é diferente de zero quando $j_s = 0$. Neste caso, tem-se que $\omega^{j_s(0.j)2^{n+s}}\omega^{*k_s(0.k)2^{n+s}} = -1$. Portanto,

$$\begin{aligned}
\left(P^{(s)}P^{(s)\dagger}\right)_{j,j+2^s} &= \frac{\neg j_s}{2}\omega^{j_s(0.j)2^{n+s}} - \frac{\neg j_s}{2} \\
&= 0, \qquad (4.31)
\end{aligned}$$

de modo que $\left(P^{(s)}P^{(s)\dagger}\right)_{jk} = \delta_{jk}$. □

Como as matrizes $P^{(s)}$ são unitárias, elas são potencialmente implementáveis em um computador quântico. O único problema de agora em diante é decompor estas matrizes em outras mais simples, que correspondam a CNOTs e a portas que atuem em um único *qubit* — as portas lógicas quânticas universais (BARENCO *et al.*, 1995). Um algoritmo quântico somente é considerado eficiente se a quantidade de portas universais não crescer muito rapidamente com o número de *qubits* na entrada. Além disso, é importante desenvolver uma versão eficiente do algoritmo quântico utilizando apenas um conjunto finito de portas lógicas quânticas, ou o algoritmo corre o risco de nunca ser implementado em um computador quântico real. Certamente, esta exigência pode implicar que o algoritmo tenha que ser aproximado. No entanto, existe um conjunto canônico de portas lógicas que pode ser utilizado para decompor qualquer porta universal com precisão arbitrária. Este conjunto é formado pelas portas Hadamard, Fase, CNOT e $\pi/8$. Existe ainda um conjunto alternativo, composto por Hadamard, Fase, CNOT e Toffoli, que pode ser utilizado para o mesmo fim (NIELSEN; CHUANG, 2000).

Para obter portas universais quânticas correspondentes ao algoritmo FFT, deve-se aplicar diversas decomposições às matrizes $P^{(s)}$. A primeira delas é a decomposição QR, descrita no Apêndice A. Para aplicar este método, em primeiro lugar

é necessário obter uma coluna arbitrária de uma matriz $P^{(s)}$, o que pode ser feito fixando um valor de k na Equação (4.25) e então fazendo j variar de 0 até $N-1$. A coluna k da matriz $P^{(s)}$ é

$$C_k = \begin{pmatrix} 0 \\ \vdots \\ -\dfrac{1-(-1)^{\left\lfloor \frac{k}{2^s} \right\rfloor}}{2}\omega^{(0.k)2^{n+s}} \\ \vdots \\ \omega^{k_s(0.k)2^{n+s}} \\ \vdots \\ \dfrac{1+(-1)^{\left\lfloor \frac{k}{2^s} \right\rfloor}}{2} \\ \vdots \\ 0 \end{pmatrix}. \tag{4.32}$$

onde as entradas não-nulas situam-se nas linhas $j = k - 2^s$, $j = k$ e $j = k + 2^s$, respectivamente.

As colunas das matrizes $P^{(s)}$ já são ortonormais. Agora, multiplicando-se cada coluna C_k arbitrária de $P^{(s)}$ pela constante

$$\alpha_k^{(s)} \equiv (-1)^{\left\lfloor \frac{k}{2^s} \right\rfloor}\omega^{*k_s(0.k)2^{n+s}}, \tag{4.33}$$

chega-se às matrizes ortogonais

$$M_{jk}^{(s)} = \frac{1}{\sqrt{2}} \begin{cases} (-1)^{\left\lfloor \frac{j}{2^s} \right\rfloor}, & \text{se } k = j \\ \dfrac{1-(-1)^{\left\lfloor \frac{j}{2^s} \right\rfloor}}{2}, & \text{se } k = j - 2^s \\ \dfrac{1+(-1)^{\left\lfloor \frac{j}{2^s} \right\rfloor}}{2}, & \text{se } k = j + 2^s \\ 0, & \text{caso contrário.} \end{cases} \tag{4.34}$$

A matriz $M^{(s)}$ de um certo passo s do algoritmo corresponde à primeira matriz do lado direito da Equação (A.5). Ainda na Equação (A.5) nota-se que $\|u_k\|$ corresponde a α_k^* e, como as colunas das matrizes $P^{(s)}$ já eram ortonormais no início do processo, $\langle e_j | a_k \rangle = 0$ para $j \neq k$. Portanto, a outra matriz obtida pela decomposição

QR de $P^{(s)}$, em um passo s arbitrário, é dada por

$$N_{jk}^{(s)} = \begin{cases} (-1)^{\left\lfloor \frac{j}{2^s} \right\rfloor} \omega^{j_s(0.j)2^{n+s}}, & \text{se } k = j \\ 0, & \text{caso contrário.} \end{cases} \tag{4.35}$$

Tal decomposição é confirmada pela seguinte Proposição:

Proposição 4.2. *Em qualquer passo s e para qualquer número de qubits, tem-se que*

$$P^{(s)} = M^{(s)} N^{(s)}, \tag{4.36}$$

onde as matrizes $M^{(s)}$ e $N^{(s)}$ são dadas pelas Equações (4.34) *e* (4.35)*, respectivamente.*

Demonstração. Como as matrizes $N^{(s)}$ são diagonais, tem-se que

$$\left(M^{(s)} N^{(s)}\right)_{jk} = M_{jk}^{(s)} N_{kk}^{(s)}. \tag{4.37}$$

Quando $k = j$,

$$\begin{aligned} M_{jj}^{(s)} N_{jj}^{(s)} &= \frac{(-1)^{\left\lfloor \frac{j}{2^s} \right\rfloor}}{\sqrt{2}} (-1)^{\left\lfloor \frac{j}{2^s} \right\rfloor} \omega^{j_s(0.j)2^{n+s}} \\ &= P_{jj}^{(s)}. \end{aligned} \tag{4.38}$$

Quando $k = j - 2^s$,

$$M_{j,j-2^s}^{(s)} N_{j-2^s,j-2^s}^{(s)} = \frac{1 - (-1)^{\left\lfloor \frac{j}{2^s} \right\rfloor}}{2\sqrt{2}} (-1)^{\left\lfloor \frac{k_s}{2^s} \right\rfloor} \omega^{k_s(0.k)2^{n+s}}. \tag{4.39}$$

O lado direito da Equação (4.39) somente é diferente de zero quando $j_s = 1$. Neste caso tem-se $k_s = 0$. Portanto,

$$\begin{aligned} M_{j,j-2^s}^{(s)} N_{j-2^s,j-2^s}^{(s)} &= \frac{1 - (-1)^{\left\lfloor \frac{j}{2^s} \right\rfloor}}{2\sqrt{2}} \\ &= P_{j,j-2^s}^{(s)}. \end{aligned} \tag{4.40}$$

Quando $k = j + 2^s$,

$$M_{j,j+2^s}^{(s)} N_{j+2^s,j+2^s}^{(s)} = \frac{1 + (-1)^{\left\lfloor \frac{j}{2^s} \right\rfloor}}{2\sqrt{2}} (-1)^{\left\lfloor \frac{k_s}{2^s} \right\rfloor} \omega^{k_s(0.k)2^{n+s}}. \tag{4.41}$$

O lado direito da Equação (4.39) somente é diferente de zero quando $k_s = 1$. Portanto,

$$\begin{aligned} M^{(s)}_{j,j+2^s} N^{(s)}_{j+2^s,j+2^s} &= \frac{1+(-1)^{\left\lfloor \frac{j}{2^s} \right\rfloor}}{2\sqrt{2}} \omega^{(0.j)2^{n+s}} \\ &= P^{(s)}_{j,j+2^s}, \end{aligned} \tag{4.42}$$

de modo que $\left(M^{(s)}N^{(s)}\right)_{jk} = P^{(s)}_{jk}$. □

Será considerada agora a decomposição das matrizes $M^{(s)}$. Começando por $M^{(0)}$, nota-se que

$$M^{(0)}_{jk} = \frac{1}{\sqrt{2}} \begin{cases} (-1)^j, & \text{se } j = k \\ \dfrac{1-(-1)^j}{2}, & \text{se } k = j-1 \\ \dfrac{1+(-1)^j}{2}, & \text{se } k = j+1 \\ 0, & \text{caso contrário.} \end{cases} \tag{4.43}$$

Verifica-se que

$$M^{(0)}_{2^n \times 2^n} = \begin{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} & & \\ & \ddots & \\ & & \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \end{pmatrix}. \tag{4.44}$$

Portanto,

$$M^{(0)}_{2^n \times 2^n} = I^{\otimes(n-1)} \otimes H, \tag{4.45}$$

onde $I^{\otimes(n-1)}$ denota o produto tensorial entre $n-1$ matrizes I.

Agora, prova-se que $M^{(s)}_{2^n \times 2^n} = M^{(s-1)}_{2^{n-1} \times 2^{n-1}} \otimes I$. Para isso, pode-se utilizar a fórmula

$$(A \otimes I)_{jk} = \begin{cases} A_{\left\lfloor \frac{j}{2} \right\rfloor \left\lfloor \frac{k}{2} \right\rfloor}, & \text{se } k = j \pmod 2 \\ 0, & \text{caso contrário} \end{cases} \tag{4.46}$$

para uma matriz genérica A. A fórmula pode ser obtida através da análise das entradas de $A \otimes I$.

Substituindo A por $M^{(s-1)}_{2^{n-1}\times 2^{n-1}}$ na Equação (4.46) obtém-se

$$\left(M^{(s-1)}_{2^{n-1}\times 2^{n-1}} \otimes I\right)_{jk} = \begin{cases} (-1)^{\left\lfloor \frac{\lfloor \frac{j}{2} \rfloor}{2^{s-1}} \right\rfloor}, & \text{se } \left\lfloor \frac{k}{2} \right\rfloor = \left\lfloor \frac{j}{2} \right\rfloor \\ & \text{e } k = j \pmod 2 \\ \frac{1-(-1)}{2}^{\left\lfloor \frac{\lfloor \frac{j}{2} \rfloor}{2^{s-1}} \right\rfloor}, & \text{se } \left\lfloor \frac{k}{2} \right\rfloor = \left\lfloor \frac{j}{2} \right\rfloor - 2^{s-1} \\ & \text{e } k = j \pmod 2 \\ \frac{1+(-1)}{2}^{\left\lfloor \frac{\lfloor \frac{j}{2} \rfloor}{2^{s-1}} \right\rfloor}, & \text{se } \left\lfloor \frac{k}{2} \right\rfloor = \left\lfloor \frac{j}{2} \right\rfloor + 2^{s-1} \\ & \text{e } k = j \pmod 2 \\ 0, & \text{caso contrário.} \end{cases} \tag{4.47}$$

Simplificando a Equação (4.47) obtém-se o lado direito da Equação (4.34). Portanto,

$$M^{(s)}_{2^n\times 2^n} = M^{(s-1)}_{2^{n-1}\times 2^{n-1}} \otimes I. \tag{4.48}$$

Proposição 4.3. *As matrizes $M^{(s)}$ podem ser decompostas como*

$$M^{(s)} = I^{\otimes(n-s-1)} \otimes H \otimes I^{\otimes s}. \tag{4.49}$$

Demonstração. Usando a Equação (4.48) recursivamente s vezes e substituindo n por $n-s$ na Equação (4.45) obtém-se a Equação (4.49). □

Ainda é necessário analisar a estrutura das matrizes $N^{(s)}$. De acordo com a Equação (4.35), essas matrizes são diagonais e suas entradas são iguais a um ou iguais a $\pm\omega^c$, para $0 \leq c \leq N$. Pode-se tentar escrever uma matriz $N^{(s)}$, para s fixo, como o produto de matrizes diagonais, cujas entradas sejam iguais a um ou iguais a $\pm\omega^c$, para um c fixo. De fato, pode-se utilizar as Equações (4.22) e (4.23) para reescrever a Equação (4.35) como

$$N^{(s)}_{jk} = \begin{cases} (-1)^{j_s} \prod_{t=0}^{n-1} \omega^{j_s j_t 2^{n+s-t-1}}, & \text{se } k = j, \\ 0, & \text{caso contrário.} \end{cases} \tag{4.50}$$

Como n e s são fixos os fatores procurados podem ser obtidos fixando t. Convém definir a matriz

$$R^{(s,t,u)} = \begin{cases} \omega^{j_s j_t 2^{n-u}}, & \text{se } k = j \\ 0, & \text{caso contrário.} \end{cases} \tag{4.51}$$

É importante notar que $R^{(s,t,u)} = R^{(t,s,u)}$. Além disso, por serem diagonais, as matrizes $R^{(s,t,u)}$ comutam.

Proposição 4.4. *As matrizes $N^{(s)}$ podem ser escritas como*

$$N^{(s)} = \prod_{t=s+1}^{n-1} R^{(s,t,u)}, \tag{4.52}$$

com $u = t - s + 1$, quando $s < n - 1$. Quando $s = n - 1$ tem-se que $N^{(n-1)} = I$.

Demonstração. Basta observar que $(-1)^{j_s} = \omega^{j_s 2^{n-1}}$. Segue-se, então, que,

$$\begin{aligned} \prod_{t=s+1}^{n-1} (-1)^{j_s}(-1)^{j_s}\omega^{j_s j_t 2^{n+s-t-1}} &= (-1)^{j_s}\omega^{j_s 2^{n-1}}\omega^{\sum_{t=s+1}^{n-1} j_s j_t 2^{n+s-t-1}} \\ &= (-1)^{j_s}\omega^{\sum_{t=s}^{n-1} j_s j_t 2^{n+s-t-1}}. \end{aligned} \tag{4.53}$$

Como $\sum_{t=0}^{s-1} j_t 2^{n+s-t-1} \equiv 0 \pmod{2^n}$, pode-se continuar o desenvolvimento da equação, fazendo

$$(-1)^{j_s}\omega^{\sum_{t=s}^{n-1} j_s j_t 2^{n+s-t-1}} = (-1)^{j_s}\omega^{\sum_{t=0}^{n-1} j_s j_t 2^{n+s-t-1}}, \tag{4.54}$$

recuperando facilmente a Equação (4.50), quando $s < n - 1$. Quando $s = n - 1$, basta substituir s por $n - 1$ na Equação (4.50) para verificar que $N^{(n-1)} = I$. □

As matrizes $R^{(s,t,u)}$ possuem uma estrutura importante. Além de serem diagonais, suas entradas apresentam, em uma linha j, os seguintes valores: um, quando j_s ou j_t for igual a zero; e $\omega^{2^{n-u}} = \exp\left(\frac{2\pi i}{2^u}\right)$, quando j_s e j_t forem simultaneamente iguais a um. Esta análise revela que a matriz $R^{(s,t,u)}$ é uma operação controlada, dada pela matriz

$$R^{(u)} \equiv \begin{pmatrix} 1 & 0 \\ 0 & \exp\left(\frac{2\pi i}{2^u}\right) \end{pmatrix}, \tag{4.55}$$

com controle no *qubit* s e alvo no *qubit* t (ou vice-versa, neste caso). Esta porta lógica pode ser vista como uma generalização das portas controladas atuando em dois *qubits*

na presença de outros *qubits*. Ou seja, em uma representação de circuito quântico os *qubits* alvo e controle da porta $R^{(s,t,u)}$ não são necessariamente adjacentes.

Resumindo os resultados desta Seção, a QFT pode ser expressa como

$$F_{2^n} = A^{(n)} \prod_{s=0}^{n-1} M^{(s)} N^{(s)}, \tag{4.56}$$

onde as matrizes $M^{(s)}$ e $N^{(s)}$ são dadas respectivamente pelas Equações (4.49) e (4.52). $A^{(n)}$ é uma matriz $2^n \times 2^n$ que implementa a reordenação do final do algoritmo. Na Seção 4.5 apresenta-se uma expressão algébrica para $A^{(n)}$.

4.3 Um algoritmo quântico para a transformada exata

É possível agora construir o algoritmo quântico para calcular a DFT. No início do algoritmo clássico (página 30) têm-se um vetor $X \in \mathbb{C}^{2^n}$. A inicialização é realizada através do comando $X^{(n)}_{(a_{n-1}a_{n-2}\ldots a_0)_2} \leftarrow X_{(a_{n-1}a_{n-2}\ldots a_0)_2}$, aplicado para todas as entradas do vetor. Esta etapa corresponde à obtenção de um estado quântico

$$|\psi_n\rangle = \sum_{0 \le a_{n-1}a_{n-2}\cdots a_0 \le 1} X^{(n)}_{(a_{n-1}a_{n-2}\cdots a_0)_2} |a_{n-1}a_{n-2}\cdots a_0\rangle, \tag{4.57}$$

desde que o vetor X seja unitário.[1] Portanto, a DFT pode ser calculada em um computador quântico através da aplicação das portas lógicas definidas na Equação (4.25), na seguinte ordem:

$$|\psi_0\rangle = P^{(0)} P^{(1)} \cdots P^{(n-1)} |\psi_n\rangle . \tag{4.58}$$

O estado $|\psi_0\rangle$ obtido é

$$|\psi_0\rangle = \sum_{0 \le c_{n-1}c_{n-2}\cdots c_0 \le 1} X^{(n)}_{(c_0c_1\cdots c_{n-1})_2} |c_{n-1}c_{n-2}\cdots c_0\rangle. \tag{4.59}$$

A aplicação das portas de *swap*, representadas pela matriz $A^{(n)}$, corrige a indexação dos coeficientes.

Apesar das matrizes $P^{(s)}$ serem, em geral, muito complexas para serem executadas diretamente em um computador quântico, foi demonstrado na Seção 4.2 que elas podem ser decompostas em matrizes $M^{(s)}$ e $N^{(s)}$, e que estas por sua vez podem

[1] Ainda que não seja unitário, ele pode facilmente ser normalizado.

ser decompostas em portas ainda mais simples, atuando em não mais que dois *qubits* simultaneamente.

Em cada passo s, a primeira porta a ser aplicada é a porta $N^{(s)}$. No passo $n-1$ tem-se que $N^{(n-1)} = I$. No entanto, cada matriz $N^{(s)}$ pode ser decomposta no produto de matrizes mais simples, de acordo com a Equação (4.52). Portanto, em vez de aplicar $N^{(s)}$, deve-se aplicar uma seqüência de portas $R^{(s,t,t-s+1)}$, para t começando em $n-1$, indo regressivamente até $s+1$. Deve-se lembrar que estas portas são, na verdade, operações $R^{(t-s+1)}$ com controle no *qubit* s e alvo no *qubit* t. Depois destas portas lógicas, deve-se ainda aplicar a matriz $M^{(s)}$, ou seja, uma porta de Hadamard atuando somente no *qubit* s. Após a aplicação de todas as matrizes, com s variando de $n-1$ até 0, deve-se aplicar a matriz $A^{(n)}$ (i.e., os *swaps*) a fim de corrigir a ordem da saída. Estes passos estão organizados no Algoritmo 2.

Entrada: Um vetor $X \in \mathbb{C}^{2^n}$.
Saída: Um estado quântico $|\psi\rangle$ cujas amplitudes correspondem aos elementos de $Y \in \mathbb{C}^{2^n}$, dados pela Transformada de Fourier Discreta de X.

{Inicialização, passo n}
prepare o estado do registrador quântico de n *qubits* como

$$|\psi_n\rangle = \sum_{k=0}^{N-1} X_k |k\rangle.$$

{Passo s}
para s de $n-1$ até 0, regressivamente **faça**
    para t de $n-1$ até $s+1$, regressivamente **faça**
        aplique operação unitária $R^{(s,t,t-s+1)}$
    fim para
    aplique uma porta de Hadamard somente no *qubit* s.
fim para

{Reordenação}
para t de 0 até $\lfloor n/2 \rfloor - 1$ **faça**
    aplique *swap* entre *qubits* t e $n-t-1$.
fim para

Algoritmo 2: QFT

A Figura 6 representa um circuito para QFT sobre quatro *qubits*, em termos de portas Hadamard, portas $R^{(u)}$ controladas e *swaps*. As portas $M^{(s)}$ e $N^{(s)}$ também são exibidas no topo. Pode-se obter diversos circuitos equivalentes através da utilização de certas propriedades da QFT. Por exemplo, o circuito utilizado no livro

de Nielsen e Chuang (2000) está representado na Figura 7, e pode ser obtido pela comutação de portas lógicas que não envolvem operações sobre o mesmo *qubit*.

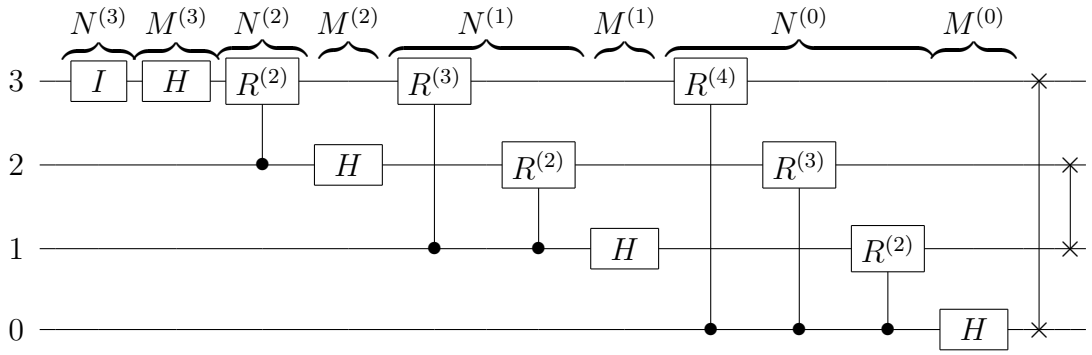


Figura 6: Um circuito para QFT com $n = 4$ *qubits* (nota-se que os *qubits* estão numerados de baixo para cima, começando em 0, indo até $n - 1$)

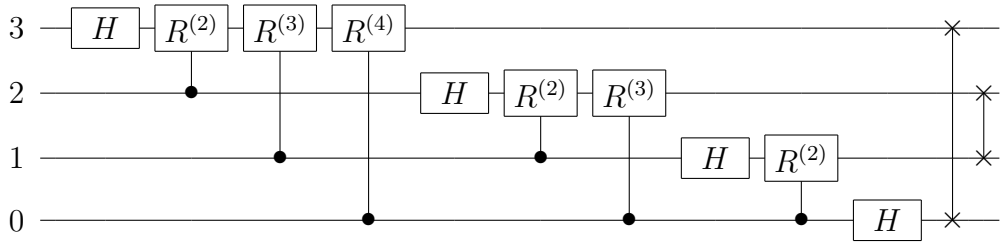


Figura 7: Um circuito equivalente para QFT com $n = 4$ *qubits*, obtido através da comutação de algumas operações lógicas

Deseja-se agora analisar a complexidade computacional do Algoritmo 2. Assume-se que a inicialização seja realizada em tempo desprezível. Esta suposição é bastante razoável, dado que as aplicações da QFT conhecidas até o presente sempre utilizam como entrada um estado da base computacional, ou a saída de um passo anterior de algum algoritmo. Na parte principal do algoritmo (o laço de repetição externo), são executados $1 + 2 + \cdots + n = n(n + 1)/2$ passos. Portanto, esta parte do algoritmo tem complexidade $O(n^2)$. A última parte do algoritmo requer $O(n)$ operações de *swap*. Conclui-se que, em termos de operações sobre um ou dois *qubits*, o algoritmo QFT possui complexidade $O(n^2)$ ou, equivalentemente, $O(\log^2 N)$. Ficará claro mais adiante que esta complexidade não se altera quando calculada em termos de portas lógicas universais.

4.4 Um algoritmo quântico para a transformada aproximada

O algoritmo clássico que calcula a Transformada de Fourier Aproximada, de acordo com a Definição 3.4, é similar ao Algoritmo 1, sendo dado pelo Algoritmo 3. A única diferença está no segundo termo do lado direito da Equação (3.17), onde deve-se substituir

$$\omega^{(b_s b_{s+1}\ldots b_{n-1}0\ldots 0)_2} = \exp\left(\frac{2\pi i}{N}\sum_{k=s}^{n-1} b_k 2^{n-1-k+s}\right) \tag{4.60}$$

por

$$\omega^{\left(b_s b_{s+1}\ldots b_{\min(s+m-1,n-1)}0\ldots 0\right)_2} = \exp\left(\frac{2\pi i}{N}\sum_{k=s}^{\min(s+m-1,n-1)} b_k 2^{n-1-k+s}\right). \tag{4.61}$$

Entrada: Um vetor $X \in \mathbb{C}^{2^n}$, e um parâmetro $1 \leq m \leq n$.
Saída: Um vetor $Y \in \mathbb{C}^{2^n}$ que é a DFT Aproximada do vetor X.

{Inicialização, passo n}
para todo a tal que $0 \leq a \leq 2^n - 1$ **faça**
  seja $X^{(n)}_{(a_{n-1}a_{n-2}\ldots a_0)_2} \leftarrow X_{(a_{n-1}a_{n-2}\ldots a_0)_2}$.
fim para

{Passo s}
para s de $n-1$ até 0, regressivamente **faça**
  para todo $0 \leq b_{n-1}, \ldots, b_s, a_{s-1}, \ldots, a_0 \leq 1$ **faça**

$$X^{(s)}_{(b_{n-1}\ldots b_s a_{s-1}\ldots a_0)_2} \leftarrow \frac{1}{\sqrt{2}} X^{(s+1)}_{(b_{n-1}\ldots b_{s+1}0 a_{s-1}\ldots a_0)_2} + \\ + \frac{1}{\sqrt{2}}\exp\left(\frac{2\pi i}{N}\sum_{k=s}^{\min(s+m-1,n-1)} b_k 2^{n-1-k+s}\right) X^{(s+1)}_{(b_{n-1}\ldots b_{s+1}1 a_{s-1}\ldots a_0)_2}. \tag{4.62}$$

  fim para
fim para

{Reordenação}
para todo b tal que $0 \leq b \leq 2^n - 1$ **faça**
  faça. $Y_{(b_{n-1}b_{n-2}\ldots b_0)_2} \leftarrow X^{(0)}_{(b_0 b_1\ldots b_{n-1})_2}$.
fim para

Algoritmo 3: FFT clássica aproximada

O algoritmo aproximado para DFT apresentado nesta dissertação não encontra-se descrito na literatura. Os algoritmos aproximados de DFT mais conhecidos são os de Shentov *et al.* (1995), Guo e Burrus (1996) e Edelman, McCorquodale e Toledo (1999). De fato, o algoritmo baseado na Definição 3.4 não proporciona ganho de complexidade na Computação Clássica — a análise de complexidade é análoga àquela realizada no final da Seção 3.3. Sua utilização neste trabalho é apenas um passo intermediário para se chegar ao algoritmo aproximado quântico, onde há ganho de complexidade considerável.

Proposição 4.5. *O vetor Y produzido pelo Algoritmo 3 é a DFT do vetor X, conforme Definição 3.4.*

Demonstração. Deseja-se demonstrar, por indução, que a relação

$$X^{(s)}_{(b_{n-1}b_{n-2}\ldots b_s a_{s-1}\ldots a_0)_2} = \frac{1}{\sqrt{2^{n-s}}} \sum_{0\leq a_{n-1},a_{n-2},\ldots a_s\leq 1} X_{(a_{n-1}\ldots a_0)_2} \exp\left(\frac{2\pi i}{N} \sum_{\substack{s\leq j\leq n-1\\ 0\leq k\leq n-1\\ n-m\leq j+k\leq n-1}} a_j b_{n-1-k} 2^{j+k}\right) \quad (4.63)$$

é válida para qualquer passo $0 \leq s < n$ do Algoritmo 3, supondo que o passo $s = n$ tenha feito $X^{(n)}_a = X_a$. Esta demonstração é muito semelhante à demonstração da Proposição 3.2.

Primeiramente, é fácil perceber que a Equação (3.20) é válida quando $s = n-1$.

$$\frac{1}{\sqrt{2}} \sum_{0\leq a_{n-1}\leq 1} X_{(a_{n-1}\ldots a_0)} \exp\left(\frac{2\pi i}{N} \sum_{\substack{j=n-1\\ 0\leq k\leq n-1\\ n-m\leq j+k\leq n-1}} a_j b_{n-1-k} 2^{j+k}\right) \quad (4.64)$$

$$= \frac{1}{\sqrt{2}} X^{(n)}_{(0\ a_{n-2}\ldots a_0)} + \frac{1}{\sqrt{2}} X^{(n)}_{(1\ a_{n-2}\ldots a_0)} \omega^{(b_{n-1}0\ldots 0)}$$

$$= X^{(n-1)}_{(b_{n-1}a_{n-2}\ldots a_0)}. \quad (4.65)$$

Em seguida, supondo que a Equação (4.63) seja válida para um passo arbitrário

$s+1$, conclui-se que ela também é válida para o passo seguinte, s. Basta notar que

$$\begin{aligned}
&\frac{1}{\sqrt{2^{n-s}}} \sum_{0\le a_{n-1},\ldots,a_s\le 1} X_{(a_{n-1}\ldots a_0)} \exp\left(\frac{2\pi i}{N} \sum_{\substack{s\le j\le n-1\\ 0\le k\le n-1\\ n-m\le j+k\le n-1}} a_j b_{n-1-k} 2^{j+k}\right) \\
&\quad = \frac{1}{\sqrt{2^{n-s}}} \sum_{\substack{0\le a_{n-1},\ldots,a_{s+1}\le 1\\ a_s=0}} X_{(a_{n-1}\ldots a_0)} \exp\left(\frac{2\pi i}{N} \sum_{\substack{s+1\le j\le n-1\\ 0\le k\le n-1\\ n-m\le j+k\le n-1}} a_j b_{n-1-k} 2^{j+k}\right) + \\
&\quad + \frac{1}{\sqrt{2^{n-s}}} \sum_{\substack{0\le a_{n-1},\ldots,a_{s+1}\le 1\\ a_s=1}} X_{(a_{n-1}\ldots a_0)} \exp\left(\frac{2\pi i}{N} \sum_{\substack{s+1\le j\le n-1\\ 0\le k\le n-1\\ n-m\le j+k\le n-1}} a_j b_{n-1-k} 2^{j+k}\right) \times \\
&\qquad\qquad \times \exp\left(\frac{2\pi i}{N} \sum_{\substack{0\le k\le n-1\\ n-s-m\le k\le n-s-1}} b_{n-1-k} 2^{s+k}\right). \quad (4.66)
\end{aligned}$$

Substituindo variáveis na segunda exponencial do segundo termo do lado direito da equação, e tirando esta exponencial do somatório, obtém-se

$$\begin{aligned}
&\frac{1}{\sqrt{2^{n-s}}} \sum_{0\le a_{n-1},\ldots,a_s\le 1} X_{(a_{n-1}\ldots a_0)} \exp\left(\frac{2\pi i}{N} \sum_{\substack{s\le j\le n-1\\ 0\le k\le n-1\\ n-m\le j+k\le n-1}} a_j b_{n-1-k} 2^{j+k}\right) \\
&\quad = \frac{1}{\sqrt{2^{n-s}}} \sum_{\substack{0\le a_{n-1},\ldots,a_{s+1}\le 1\\ a_s=0}} X_{(a_{n-1}\ldots a_0)} \exp\left(\frac{2\pi i}{N} \sum_{\substack{s+1\le j\le n-1\\ 0\le k\le n-1\\ n-m\le j+k\le n-1}} a_j b_{n-1-k} 2^{j+k}\right) + \\
&\qquad\qquad + \frac{1}{\sqrt{2^{n-s}}} \exp\left(\frac{2\pi i}{N} \sum_{\substack{s\le k\le s+m-1\\ 0\le k\le n-1}} b_k 2^{n-1-k+s}\right) \times \\
&\qquad \times \sum_{\substack{0\le a_{n-1},\ldots,a_{s+1}\le 1\\ a_s=1}} X_{(a_{n-1}\ldots a_0)} \exp\left(\frac{2\pi i}{N} \sum_{\substack{s+1\le j\le n-1\\ 0\le k\le n-1\\ n-m\le j+k\le n-1}} a_j b_{n-1-k} 2^{j+k}\right). \quad (4.67)
\end{aligned}$$

Por hipótese, o primeiro termo do lado direito da equação é o vetor $\frac{1}{\sqrt{2}}X^{(s+1)}$ avaliado em todos os índices onde $a_s = 0$. Analogamente, o segundo termo contém o vetor $\frac{1}{\sqrt{2}}X^{(s+1)}$ avaliado em todos os índices onde $a_s = 1$. Portanto, reescrevendo

a Equação (4.67), obtém-se

$$\frac{1}{\sqrt{2}}X^{(s+1)}_{(b_{n-1}\ldots b_{s+1}\ 0\ a_{s-1}\ldots a_o)}+ \\ + \frac{1}{\sqrt{2}}X^{(s+1)}_{(b_{n-1}\ldots b_{s+1}\ 1\ a_{s-1}\ldots a_0)} \exp\left(\frac{2\pi i}{N}\sum_{k=s}^{\min(s+m-1,n-1)} b_k 2^{n-1-k+s}\right) \\ = X^{(s)}_{(b_{n-1}\ldots b_s a_{s-1}\ldots a_0)}. \quad (4.68)$$

Fazendo $s = 0$ na Equação (4.63), correspondendo ao último passo do Algoritmo 3, conclui-se que a expressão de fato produz a DFT definida na Equação (3.10), exceto pelo uso do índice b — com bits revertidos — onde deveria ser c. Isto é facilmente corrigido através da reordenação, descrita no final do algoritmo. □

Calculando as matrizes genéricas $P^{(s)}$ do Algoritmo 3 e decompondo-as, chega-se à conclusão que a única diferença entre elas e as matrizes do algoritmo exato, é a fração binária $0.j$, que deve ser substituída por

$$0.j \equiv 0.j_0 j_1 \ldots j_{\min(s+m-1,n-1)} = \sum_{t=0}^{\min(s+m-1,n-1)} j_t/2^{t+1}. \quad (4.69)$$

Então, o processo de decomposição das matrizes da transformada aproximada é o mesmo das matrizes da transformada exata, exceto pelas matrizes $N^{(s)}$, que passam a ser escritas como

$$(N^{(s)}_{aprox})_{jk} = \begin{cases} (-1)^{j_s}\prod_{t=0}^{\min(s+m-1,n-1)} \omega^{j_s j_t 2^{n+s-t-1}}, & \text{se } k = j, \\ 0, & \text{caso contrário.} \end{cases} \quad (4.70)$$

Utilizando as matrizes $R^{(s,t,u)}$ definidas pela Equação (4.51), pode-se demonstrar que as matrizes $N^{(s)}_{aprox}$ são decompostas de forma muito semelhante àquela da Proposição 4.4.

Proposição 4.6. *As matrizes $N^{(s)}_{aprox}$ podem ser escritas como*

$$N^{(s)}_{aprox} = \prod_{t=s+1}^{\min(s+m-1,n-1)} R^{(s,t,u)}, \quad (4.71)$$

com $u = t - s + 1$, quando $s < n - 1$. Quando $s = n - 1$, tem-se que $N^{(n-1)} = I$.

Demonstração. Basta observar que $(-1)^{j_s} = \omega^{j_s 2^{n-1}}$. Segue-se, então, que,

$$\begin{aligned}\prod_{t=s+1}^{\min(s+m-1,n-1)} (-1)^{j_s}(-1)^{j_s}\omega^{j_s j_t 2^{n+s-t-1}} &= (-1)^{j_s}\omega^{j_s 2^{n-1}}\omega^{\sum_{t=s+1}^{\min(s+m-1,n-1)} j_s j_t 2^{n+s-t-1}} \\ &= (-1)^{j_s}\omega^{\sum_{t=s}^{\min(s+m-1,n-1)} j_s j_t 2^{n+s-t-1}}. \quad (4.72)\end{aligned}$$

Como $\sum_{t=0}^{s-1} j_t 2^{n+s-t-1} \equiv 0 \pmod{2^n}$, pode-se continuar o desenvolvimento da equação, quando $s < n-1$, fazendo

$$(-1)^{j_s}\omega^{\sum_{t=s}^{\min(s+m-1,n-1)} j_s j_t 2^{n+s-t-1}} = (-1)^{j_s}\omega^{\sum_{t=0}^{\min(s+m-1,n-1)} j_s j_t 2^{n+s-t-1}}, \quad (4.73)$$

e recuperando a Equação (4.70). Quando $s = n-1$, basta substituir s por $n-1$ na Equação (4.70) e verificar que $N^{(n-1)} = I$. □

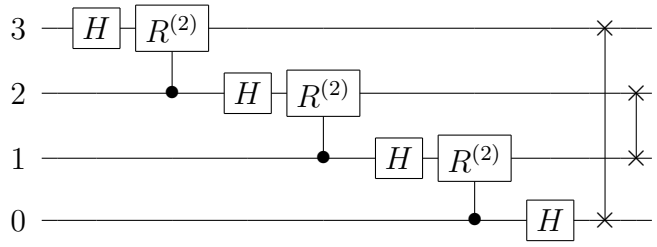


Figura 8: Um circuito aproximado para a QFT sobre quatro *qubits*, com parâmetro $m = 2$

Portanto, a Transformada de Fourier Aproximada pode ser realizada em um computador quântico de acordo com o Algoritmo 4. A expressão algébrica da QFT Aproximada é obtida substituindo $N^{(s)}$ por $N^{(s)}_{aprox}$ na Equação (4.56). A representação gráfica do algoritmo, no caso $n = 4$ e $m = 2$, é dada no circuito da Figura 8.

Percebe-se que o processo de aproximação da QFT realmente corresponde à eliminação das matrizes $R^{(s,t,u)}$ com $u > m$, conforme descrito por Coppersmith (1994). Isto se dá pelo fato de as matrizes $R^{(u)}$ realizarem mudanças de fase que tendem exponencialmente para zero enquanto u cresce apenas linearmente.

Agora, deseja-se calcular a complexidade computacional do Algoritmo 4. Novamente, assume-se que a inicialização seja realizada em tempo desprezível. Na parte

1: {Inicialização, passo n}
2: prepare o estado de um registrador quântico de n *qubits* como

$$|\psi\rangle = \sum_{k=0}^{N-1} X_k |k\rangle.$$

3: {Passo s}
4: **para** s de $n-1$ até 0, regressivamente **faça**
5: **para** t de $\min(s+m-1, n-1)$ até $s+1$, regressivamente **faça**
6: aplique transformação unitária $R^{(s,t,t-s+1)}$.
7: **fim para**
8: aplique uma porta de Hadamard somente no *qubit* s.
9: **fim para**
10:
11: {Reordenação}
12: **para** t de 0 até $\lfloor n/2 \rfloor - 1$ **faça**
13: aplique *swap* entre *qubits* t e $n-t-1$.
14: **fim para**

Algoritmo 4: QFT Aproximada

principal do algoritmo, tem-se

$$\overbrace{1+2+3}^{m} + \ldots + \overbrace{m+m+m}^{n-m} = \frac{m(2n-m+1)}{2} \tag{4.74}$$

passos. Portanto, esta parte do algoritmo tem complexidade $O(mn)$. A última parte do algoritmo consiste de $O(n)$ operações de *swap*. Conclui-se que, em termos de portas atuando em um e dois *qubits*, o algoritmo aproximado de QFT possui complexidade $O(mn)$ ou, equivalentemente, $O(m \log N)$. Normalmente utiliza-se um parâmetro $m \approx \log \log N$. As vantagens na utilização deste parâmetro, especialmente na presença de descoerência, estão descritas nos artigos de Barenco *et al.* (1996) e Cheung (2004).

Foi visto anteriormente que a Transformada de Hadamard é um caso particular da Transformada de Fourier Aproximada quando $m = 1$. Portanto, basta utilizar o parâmetro adequado no Algoritmo 4, e o Algoritmo 5 é facilmente encontrado. O resultado obtido, no entanto, corresponde ao da Transformada de Fourier indexada por b. Portanto, a fim de obter a Transformada de Hadamard usual deve-se também suprimir os *swaps* do final do algoritmo da QFT Aproximada.

Também é interessante notar que o efeito da QFT sobre o estado da base computacional $|00\cdots0\rangle$ é o mesmo de uma Transformada de Hadamard, ou seja, o

1: {Inicialização, passo n}
2: prepare o estado de um registrador quântico de n *qubit* como

$$|\psi\rangle = \sum_{k=0}^{N-1} X_k \, |k\rangle.$$

3: {Passo s}
4: **para** s de $n-1$ até 0, regressivamente **faça**
5: aplique a porta de Hadamard somente no *qubit* s.
6: **fim para**

Algoritmo 5: Transformada de Hadamard Quântica

resultado é uma superposição de todos os estados possíveis com probabilidade uniformemente distribuída.

4.5 Da FFT clássica à quântica em uma abordagem recursiva

Sabe-se, pelo Lema 3.2 (Danielson-Lanczos), que a DFT pode ser obtida pela abordagem dividir-para-conquistar, resolvendo primeiramente os índices pares do vetor, então os ímpares, e finalmente reunindo os resultados segundo a Equação (3.13). A representação matricial que implementa o método de Danielson-Lanczos é descrita, por exemplo, nos livros de Meyer (2000) e de Loan (1992):

$$F_{2^n} = \begin{pmatrix} F_{2^{n-1}} & D_{2^{n-1}} F_{2^{n-1}} \\ F_{2^{n-1}} & -D_{2^{n-1}} F_{2^{n-1}} \end{pmatrix} B^T, \tag{4.75}$$

onde

$$D_{2^n} = \begin{pmatrix} 1 & & & \\ & \omega_{2^n} & & \\ & & \ddots & \\ & & & \omega_{2^n}^{2^n-1} \end{pmatrix}, \tag{4.76}$$

e B^T é uma matriz de permutação que separa os componentes de índices pares e ímpares de um vetor. Por exemplo, no caso $n = 3$ tem-se que

$$B^T \begin{pmatrix} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{pmatrix}^T = \begin{pmatrix} x_0 & x_2 & x_4 & x_6 \mid x_1 & x_3 & x_5 & x_7 \end{pmatrix}^T. \tag{4.77}$$

Como $F_{2^n} = F_{2^n}^T$ — a simetria de F_{2^n} é discutida na Seção 4.7 — pode-se utilizar a identidade $(AB)^T = B^T A^T$ para reescrever a Equação (4.75) como

$$\begin{aligned} F_{2^n} &= B \begin{pmatrix} F_{2^{n-1}}^T & F_{2^{n-1}}^T \\ (D_{2^{n-1}} F_{2^{n-1}})^T & -(D_{2^{n-1}} F_{2^{n-1}})^T \end{pmatrix} \\ &= B \begin{pmatrix} F_{2^{n-1}} & F_{2^{n-1}} \\ F_{2^{n-1}} D_{2^{n-1}} & -F_{2^{n-1}} D_{2^{n-1}} \end{pmatrix}. \end{aligned} \tag{4.78}$$

Decompondo esta representação, tem-se que

$$F_{2^n} = B(I \otimes F_{2^{n-1}}) \begin{pmatrix} I_{2^{n-1}} & \\ & D_{2^{n-1}} \end{pmatrix} (H \otimes I_{2^{n-1}}). \tag{4.79}$$

Esta decomposição possui quatro termos, a serem analisados da direita para a esquerda. (i) A primeira matriz é um Hadamard atuando no *qubit* mais significativo. (ii) A segunda matriz é diagonal, e seus elementos são

$$\omega^{j_{n-1}(0 j_{n-2} j_{n-3} \cdots j_0)_2} = \prod_{t=0}^{n-2} \omega^{j_{n-1} j_t 2^t}. \tag{4.80}$$

Comutando j_{n-1} com j_t e usando a Equação (4.51), tem-se que

$$\begin{pmatrix} I_{2^{n-1}} & \\ & D_{2^{n-1}} \end{pmatrix} = \prod_{t=0}^{n-2} R^{(t,n-1,n-t)} \tag{4.81}$$

correspondendo à seqüência de portas lógicas controladas da Figura 9. (iii) A terceira matriz é a QFT atuando recursivamente sobre os $n-1$ *qubits* menos significativos. (iv) A última matriz é B, atuando em n *qubits*. Esta, será denotada por $B^{(n)}$ (vide Figura 9). É interessante notar que

$$B^{(n)} |a_{n-1} \cdots a_0\rangle = |a_{n-2} \cdots a_0 a_{n-1}\rangle . \tag{4.82}$$

Resolvendo a recursão e trocando alvos e controles, obtém-se o circuito descrito na Figura 7, exceto pelos *swaps*. Estes são substituidos por

$$A^{(n)} = B^{(n)} \left(I \otimes B^{(n-1)}\right) \left(I^{\otimes 2} \otimes B^{(n-2)}\right) \cdots \left(I^{\otimes (n-2)} \otimes B^{(2)}\right). \tag{4.83}$$

Proposição 4.7. *A matriz $A^{(n)}$ corresponde aos swaps no final do algoritmo de QFT, ou seja,*

$$A^{(n)} |a_{n-1} \cdots a_0\rangle = |a_0 \cdots a_{n-1}\rangle . \tag{4.84}$$

Demonstração. Seja

$$A^{(k)} = \left(I^{\otimes(n-k)} \otimes B^{(k)}\right) \cdots \left(I^{\otimes(n-1)} \otimes B^{(1)}\right), \tag{4.85}$$

para $1 \leq k \leq n$, onde $B^{(1)} = I$. Por indução em k verifica-se que a Equação (4.85) é válida para $1 \leq k \leq n$.

Quando $k = 1$ a análise é imediata. Agora, supõe-se que a relação seja válida também para para $k = m - 1$. Segue-se que, para $k = m$

$$\begin{aligned}\left(I^{\otimes(n-m)} \otimes B^{(m)}\right) \left(I^{\otimes(n-(m-1))} \otimes B^{(m-1)}\right) \cdots \\ \cdots \left(I^{\otimes(n-1)}\right) \otimes B^{(1)}\right) |a_{n-1} \cdots a_{m-1} a_{m-2} \cdots a_0\rangle \\ = \left(I^{\otimes(n-m)} \otimes B^{(m)}\right) |a_{n-1} \cdots a_{m-1} a_0 \cdots a_{m-2}\rangle . \end{aligned} \tag{4.86}$$

Usando a Equação (4.82), tem-se que

$$\left(I^{\otimes(n-m)} \otimes B^{(m)}\right) |a_{n-1} \cdots a_{m-1} a_0 \cdots a_{m-2}\rangle = |a_{n-1} \cdots a_m a_0 \cdots a_{m-1}\rangle . \tag{4.87}$$

Quando $k = n$, recupera-se a Equação (4.84). □

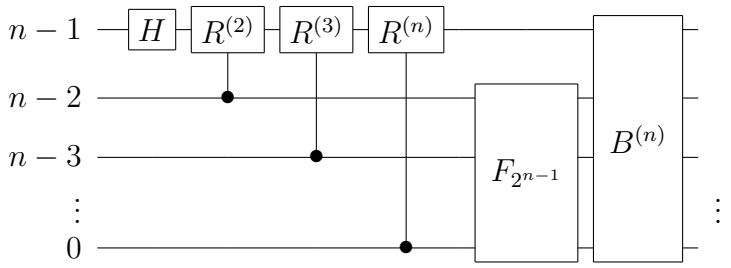


Figura 9: Circuito recursivo para a QFT

A expressão algébrica para a QFT recursiva é obtida substituindo $N^{(s)}$ por $N^{(s)}_{rec}$ na Equação (4.56), onde

$$N^{(s)}_{rec} = \prod_{t=0}^{s} R^{(t,n-1,n-t)}. \tag{4.88}$$

Formas recursivas da QFT foram discutidas nos artigos de Selinger (2004) e de Paradisi e Randriam (2004).

Ainda seria possível derivar um circuito recursivo equivalente, reescrevendo di-

retamente a Equação (4.75) sem calcular sua transposta. Neste caso, obter-se-ia

$$F_{2^n} = (H \otimes I_{2^{n-1}}) \begin{pmatrix} I_{2^{n-1}} & \\ & D_{2^{n-1}} \end{pmatrix} (I \otimes F_{2^{n-1}}) B^T. \tag{4.89}$$

Esta equação corresponde ao circuito da Figura 10. Ao resolver a recursão, pode-se demonstrar que as matrizes B^T são equivalentes aos *swaps*, desta vez aplicados no início do algoritmo. A demonstração desta afirmação é análoga à Proposição 4.7, bastando notar que que $B^T = B^{-1}$ e, portanto,

$$B^{(n)^T} |a_{n-1} \cdots a_0\rangle = |a_0 a_{n-1} \cdots a_1\rangle . \tag{4.90}$$

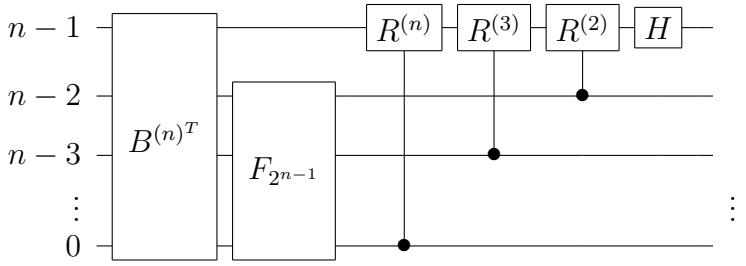


Figura 10: Circuito recursivo equivalente para a QFT

4.6 O algoritmo em termos de portas quânticas universais

Anteriormente, a complexidade computacional do algoritmo QFT foi calculada em $O(n^2)$, e de seu correspondente aproximado, em $O(mn)$. Entretanto, a análise de complexidade de algoritmos quânticos deve sempre levar em conta o algoritmo descrito em portas lógicas universais — CNOTs e portas atuando em um *qubit*. As portas controladas $R^{(s,t,u)}$ da QFT podem ser decompostas em portas universais de acordo com a equação

$$R^{(s,t,u-1)} = \left(R^{(u)} \otimes I\right) \cdot \mathrm{CNOT} \cdot \left(I \otimes R^{(u)^\dagger}\right) \cdot \mathrm{CNOT} \cdot \left(I \otimes R^{(u)}\right), \tag{4.91}$$

representada graficamente na Figura 11.

Portanto, a representação da QFT — tanto a exata quanto a aproximada — em termos de portas lógicas universais apenas multiplica o número de portas por uma constante, que desaparece assintoticamente. Desta forma, a complexidade da QFT

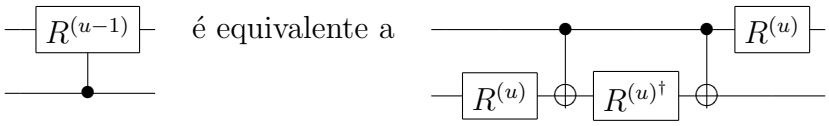


Figura 11: Decomposição da operação lógica $R^{(u)}$ em CNOTs e portas atuando em um *qubit*

permanece a mesma calculada anteriormente.

Também é importante aproximar o circuito utilizando um conjunto finito de portas lógicas universais: Hadamard, Fase, CNOT e $\pi/8$. Uma conseqüência do Teorema de Solovay-Kitaev (NIELSEN; CHUANG, 2000, pág. 197) é a possibilidade de um circuito com n CNOTs e portas de um *qubit* ser aproximado com precisão ϵ utilizando-se apenas $O\left(n \log^c \frac{n}{\epsilon}\right)$ operações do conjunto finito, onde c é uma constante próxima de dois. Portanto, após a utilização do conjunto discreto, a QFT passaria a ter complexidade $O\left(n^2 \log^c \frac{n^2}{\epsilon}\right)$ no caso exato, e $O\left(mn \log^c \frac{mn}{\epsilon}\right)$ no caso aproximado, onde c é constante. Esta complexidade ainda seria bastante razoável, e exponencialmente mais rápido que o melhor algoritmo clássico.

4.7 Propriedades adicionais da DFT

Algumas propriedades da Transformada de Fourier são úteis no desenvolvimento de algoritmos quânticos. Em primeiro lugar, é conveniente analisar a Equação (3.3) como uma operação matricial $F_N \equiv F$, tal que

$$F_N |j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{jk} |k\rangle, \tag{4.92}$$

de modo que $F_{jk} = \omega_N^{jk}$. Percebe-se de imediato que a matriz F_N é simétrica. Além disso, a decomposição da matriz F_N em portas unitárias, nas Seções anteriores, é uma prova de que a matriz F_N é também unitária.[2]

A forma canônica de inverter um circuito quântico é inverter a ordem das matrizes unitárias, — correspondendo às portas lógicas — e tomar o complexo conjugado de cada uma. Entretanto, no caso da Transformada de Fourier, nota-se que

[2]Naturalmente, esta afirmação também poderia ser demonstrada notando que o produto interno de duas colunas arbitrárias de F_N é igual a zero (MEYER, 2000).

$F_{jk}^{-1} = F_{jk}^{\dagger} = \omega_N^{-jk}$, de modo que

$$F_N^{-1} |j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{-jk} |k\rangle. \tag{4.93}$$

Portanto, a DFT inversa apenas toma o conjugado das raízes complexas da unidade, sem modificar a estrutura da matriz F_N. Esta análise permite construir um algoritmo quântico (ou, equivalentemente, um circuito quântico) apenas tomando o complexo conjugado de cada matriz envolvida, sem modificar a ordem. De fato, esta propriedade explica a equivalência entre os circuitos das Figuras 9 e 10. Basta inverter o circuito da Figura 9 duas vezes: primeiramente invertendo a ordem das portas e tomando o complexo conjugado de cada uma; e em seguida tomando o complexo conjugado das portas, sem modificar a ordem.

Define-se uma mudança de fase como uma operação P_k tal que $P_k |x\rangle = \omega^{xk} |x\rangle$, e uma translação como uma operação T_l tal que $T_l |x\rangle = |x + l \bmod N\rangle$. Pode-se demonstrar que

$$F_N T_k = P_k F_N. \tag{4.94}$$

Esta propriedade é importante no estudo do Problema do Subgrupo Escondido, e sua demonstração pode ser encontrada em artigos da área (LOMONT, 2004).

Seja r um inteiro que divide N. Então,

$$F_N \frac{1}{\sqrt{N/r}} \sum_{j=0}^{N/r-1} |rj\rangle = \frac{\sqrt{r}}{N} \sum_{y=0}^{N-1} \left(\sum_{j=0}^{N/r-1} \omega_N^{rkj} \right) |y\rangle. \tag{4.95}$$

Usando a identidade $\omega_N^r = \omega_{N/r}$, bem como o fato de que $\sum_{j=0}^{N/r-1} \omega_{N/r}^{kj}$ é igual a N/r se $j = 0 \pmod{N/r}$ e igual a zero caso contrário, chega-se a

$$F_N \frac{1}{\sqrt{N/r}} \sum_{j=0}^{N/r-1} |rj\rangle = \frac{1}{\sqrt{r}} \sum_{m=0}^{r-1} \left| \frac{N}{r} m \right\rangle. \tag{4.96}$$

Segundo esta propriedade, a Transformada de Fourier sobre um vetor cujas componentes não-nulas são periódicas com período r, é um novo vetor cujas componentes não-nulas são periódicas com período N/r.

5 Simulações

No ano de 1982, Richard Feynman já havia notado que a simulação de sistemas físicos quânticos — e conseqüentemente, computadores quânticos — seria um problema de complexidade exponencial (FEYNMAN, 1982). Portanto, um computador quântico extremamente simples, utilizando um único registrador de poucas dezenas de *qubits*, já não pode ser simulado nem mesmo pelos maiores supercomputadores da atualidade. Ainda assim, a simulação de computadores quânticos, mesmo limitada, é de grande importância.

Um simulador de computadores quânticos pode ajudar a verificar resultados, seja como ferramenta didática, ou como auxílio ao pesquisador durante o processo de desenvolvimento de um novo algoritmo. Pode também servir como ferramenta para verificar o desempenho de um algoritmo, não somente em situações ideais, mas também na presença de erros e descoerência (RAEDT; MICHIELSEN, 2004; BUTSCHER; WEIMER, 2003; ROSÉ *et al.*, 2004).

5.1 Modelo físico

A simulação de computadores quânticos tem como fundamentação teórica a equação de Schrödinger,

$$i\hbar\frac{\partial}{\partial t}\left|\psi(t)\right\rangle = \hat{H}\left|\psi(t)\right\rangle, \tag{5.1}$$

onde $\hbar$ é a constante de Plank dividida por 2π; e $\hat{H}$ é um operador positivo definido[1] e Hermitiano, conhecido como Hamiltoniano, e que está associado ao sistema físico discreto utilizado na construção do computador quântico.[2] Apesar de nenhuma

[1] Um operador A é positivo definido se $\langle v|A|v\rangle > 0$ para qualquer vetor $|v\rangle \neq 0$ (NIELSEN; CHUANG, 2000, pág. 71).

[2] O artigo de Lloyd e Braunstein (1999) discute a Computação Quântica através de sistemas físicos contínuos.

simulação realizada neste trabalho envolver a resolução explícita da equação de Schrödinger, convém estudá-la a fim de esclarecer a teoria envolvida.

Neste trabalho consideramos Hamiltonianos independentes do tempo. Neste caso, se $|\phi_j\rangle$ é um auto-estado de $\hat{H}$ com auto-valor de energia ε_j, então

$$\hat{H}\,|\phi_j\rangle = \varepsilon_j\,|\phi_j\rangle\,, \tag{5.2}$$

de modo que

$$|\psi(t)\rangle = \exp\left(-\frac{i\varepsilon_j t}{\hbar}\right)|\phi_j\rangle \tag{5.3}$$

é uma solução da Equação de Schrödinger com condições iniciais

$$|\psi(t=0)\rangle = |\phi_j\rangle\,. \tag{5.4}$$

O estado $|\psi(t)\rangle$ é estacionário,[3] pois o fator de fase não possui aqui efeito observável. Como qualquer estado inicial $|\psi(t=0)\rangle$ pode ser expresso como uma combinação dos auto-estados $|\phi_j\rangle$ de $\hat{H}$, o problema de valor inicial está resolvido, em princípio. Formalmente, a solução para $\hat{H}$ independente do tempo pode ser escrita como

$$\begin{aligned} |\psi(t)\rangle &= U(t)\,|\psi(t=0)\rangle \\ &\equiv \exp\left(-\frac{i\hat{H}t}{\hbar}\right)|\psi(0)\rangle\,, \end{aligned} \tag{5.5}$$

onde o operador de evolução temporal $U(t)$ pode ser interpretado por meio da representação espectral

$$\begin{aligned} U(t) &= \exp\left(-\frac{i\hat{H}t}{\hbar}\right) \\ &= \sum_j \exp\left(-\frac{i\varepsilon_j t}{\hbar}\right)|\phi_j\rangle\,\langle\phi_j|. \end{aligned} \tag{5.6}$$

Todos os auto-valores $\exp\left(-\frac{i\varepsilon_j t}{\hbar}\right)$ de $U(t)$ possuem módulo unitário. Portanto, pode-se demonstrar que $U(t)$ é um operador unitário.[4]

Como a dimensão da matriz que representa o operador $\hat{H}$ cresce exponencial-

[3] Estados estacionários, em Mecânica Quântica, são aqueles com energia bem definida (DAVYDOV, 1976, pág. 51).

[4] O livro de Cohen-Tannoudji, Diu e Laloë (1977, pág. 308) fornece a construção de um operador de evolução unitário para um Hamiltoniano geral, dependente do tempo.

mente com o número de *qubits* no estado $|\psi(t)\rangle$, a Equação (5.5) torna-se difícil de ser calculada exatamente, de forma que métodos numéricos aproximados fazem-se necessários sempre que a simulação envolva Hamiltonianos específicos. Dentre estes métodos pode-se citar a diagonalização total, Chebyshev, Lanczos e Suzuki-Trotter, todos descritos no artigo de Raedt e Michielsen (2004).

Nesta dissertação, entretanto, a ênfase do estudo é o algoritmo em si, não realizações físicas em particular. Portanto, as simulações consideradas são completamente gerais, de forma que nenhum *hardware* específico é assumido. Assim, a exponenciação do Hamiltoniano pode ser evitada através de sua substituição por um operador unitário U, de forma que

$$|\psi(t_f)\rangle = U\,|\psi(t_0)\rangle\,. \tag{5.7}$$

O estado inicial $|\psi(t_0)\rangle$ pode ser representado por um vetor de números complexos, e o operador U por uma matriz unitária. Estas representações seguem as convenções utilizadas pela literatura especializada, e revisadas no Capítulo 2.

Desta forma, a simulação de um computador quântico pode, a principio, ser realizada através de uma multiplicação de matrizes por vetores. O problema é o crescimento exponencial das dimensões das matrizes e dos vetores envolvidos. Uma solução paliativa é distribuir o cálculo entre diversos processadores. Entretanto, basta aumentar poucos *qubits* na simulação para que o problema torne-se inviável, mesmo com a utilização de vários supercomputadores. Além disso, a realização de uma simulação envolvendo apenas conceitos matemáticos abstratos fornece somente o resultado do cálculo. A fim de obter informações úteis acerca do algoritmo, e de como ele se comportaria em um computador quântico real, faz-se necessária uma simulação mais elaborada, levando em consideração a interpretação física dos cálculos efetuados.

5.2 Estado da arte dos simuladores

Nos últimos anos, a grande quantidade de artigos publicados em Computação Quântica, bem como a ausência de computadores quânticos reais com quantidade adequada de *qubits*, estimularam uma produção considerável de simuladores quânticos. Estes simuladores diferem em muitos aspectos, como objetivo, precisão

e forma de armazenamento dos dados. Nesta dissertação, segue-se a classificação utilizada no artigo de Raedt e Michielsen (2004). Uma revisão mais antiga sobre simulação de computadores quânticos pode ser encontrada no artigo de Wallace (1999), onde são descritos e comparados os diversos simuladores existentes na época — inclusive alguns que hoje estão indisponíveis, ou que não são mais atualizados.

5.2.1 Linguagens de programação para Computação Quântica

Linguagens de programação quânticas são uma tentativa de diminuir a distância entre os formalismos da Computação Clássica e da Computação Quântica, proporcionando meios adequados para descrever algoritmos quânticos complexos. Considera-se que a primeira proposta de linguagem de programação quântica tenha sido apresentada por Knill (1996). Outro trabalho inicial importante foi o Q-gol, desenvolvido por Baker (1996). A linguagem QCL,[5] cuja sintaxe é fortemente inspirada no C, foi desenvolvida por Ömer (1996, 1998, 2000, 2001, 2002, 2003).

Além da QCL, merece destaque a linguagem Q. Baseando-se parcialmente na linguagem Q, Chris Lomont desenvolveu a linguagem Q++. Esta linguagem, por sua vez, foi utilizada na construção do simulador SimQubit. O QDD é uma linguagem que utiliza a estrutura de *binary decision diagrams* (BDD) para representar os estados quânticos. O *site* onde a linguagem é disponibilizada fornece como exemplos o algoritmo de Shor, o teletransporte[6], a QFT, e algumas portas simples. Esta linguagem foi utilizada na preparação do simulador SHORNUF. Pode-se também mencionar o Quantum Lambda Calculus, cuja teoria é baseada nos artigos de Tonder (2003, 2004).

Muitos artigos discutem a implementação de linguagens quânticas em Haskell. O primeiro destes trabalhos foi de Mu e Bird (2001), posteriormente estendido por Sabry (2003). Vizzotto e Costa (2005) ainda estendem a idéia do artigo de Sabry, considerando desta vez a programação quântica concorrente.

O artigo de Selinger (2004) define uma linguagem de programação voltada para Computação Quântica, utilizando o paradigma de "controle clássico e dados quânticos". Altenkirch e Grattage (2005) define a linguagem QML, não implemen-

[5] *Quantum Computer Language.*

[6] O teletransporte quântico foi descoberto por Bennett *et al.* (1993) e realizado experimentalmente por Bouwmeester *et al.* (1997), Boschi *et al.* (1998), Furusawa *et al.* (1998) e Nielsen, Knill e Laflamme (1998).

tada, porém de grande importância teórica. Na linguagem QML, não apenas os dados, como também o controle pode ser quântico (GAY, 2005).

As principais linguagens desenvolvidas até o presente estão listadas na Tabela 4. Uma boa revisão sobre linguagens de programação para Computação Quântica pode ser encontrada no artigo de Simon Gay (2005), onde as linguagens aqui mencionadas são discutidas com maiores detalhes.

Nome	URL†	Linguagem	Última versão
Q	`http://sra.itc.it/people/serafini/qlang`	C++	0.5.8, 18/2/02
QCL	`http://tph.tuwien.ac.at/~oemer/qcl.html`	C++	0.6.1 (beta), 30/3/04
QDD	`http://thegreves.com/david/QDD/qdd.html`	C++	
QML	`http://www.arxiv.org/quant-ph/0409065`		
Quantum Entanglement	`http://www.cpan.org`	Perl	0.32
Quantum Fog	`http://www.ar-tiste.com`		1.6
Quantum Lambda Calculus	`http://www.het.brown.edu/people/andre/qlambda/`		
Quantum Superpositions	`http://www.cpan.org`	Perl	2.02, 22/4/03
QuBit	`http://www.bluedust.com/qubit/default.asp`	C++	
QULIB	`http://tph.tuwien.ac.at/~oemer`	C++	
Q++	`http://www.cybernet.com/programs/380/quantum/`	C++	
Q-gol	`http://www.ifost.org.au/~gregb/q-gol`	Tcl 7.5 CaML	

†URLs acessadas em 3 de janeiro de 2006.

Tabela 4: Principais linguagens de programação quânticas

5.2.2 Compiladores quânticos

Compiladores são softwares utilizados para converter um código escrito em linguagem de alto nível, em uma seqüência de instruções de baixo nível, possibilitando

sua execução por computadores. Dentre os simuladores de computadores quânticos, existem pelo menos dois softwares que podem ser classificados como compiladores de circuitos quânticos por converterem matrizes unitárias complexas em portas lógicas quânticas mais simples — ainda que não forneçam decomposições eficientes, em geral. São eles: o Qubiter e o GQC (vide Tabela 5).

Qubiter é um software escrito totalmente em C++ e que permite ao usuário decompor uma matriz unitária arbitrária em portas lógicas quânticas universais. O Qubiter pode ser utilizado juntamente com um software de preparação de circuitos em alto nível, como o Quantum Fog por exemplo. Após produzir a matriz unitária do circuito, pode-se decompô-la em portas mais simples, para serem eventualmente executadas em um computador quântico real. O artigo de Tucci (1999) aborda este assunto.

GQC é um compilador *on-line* que toma como entrada uma matriz unitária de dois *qubits*, que admita saídas emaranhadas a partir de entradas fatoradas, e a utiliza para construir um CNOT. O GQC serve para ilustrar um resultado teórico obtido pelos autores, no qual foi demonstrado que portas que atuam em dois *qubits* criando emaranhamento são universais na Computação Quântica, desde que assistidas por operações lógicas de um *qubit* (BREMNER *et al.*, 2002).

Nome	URL[‡]	Linguagem	Última versão
GQC	`http://www.physics.uq.edu.au/gqc/`		
Qubiter	`http://www.ar-tiste.com/qubiter.html`	C++	1.1

[‡]URLs acessadas em 3 de janeiro de 2006.

Tabela 5: Principais compiladores quânticos existentes

5.2.3 Softwares didáticos

Existem muitos softwares desenvolvidos com o objetivo exclusivo de servir como ferramenta para o ensino e o aprendizado de Computação Quântica. A maior parte destes simuladores está disponível na Internet, em *sites* pessoais. O Quantum Turing Machine Simulator, no entanto, está disponível apenas para assinantes do *The Mathematica Journal* (HERTEL, 2002).

Os simuladores didáticos mais relevantes estão listados na Tabela 6.

Nome	URL*	Linguagem
CS 596	`http://www.sci.sdsu/Faculty/Don.Short/QuantumC/cs662.htm`	MatLab
M-Fun for QC-Progs	`http://www.ar-tiste.com/m-fun/m-fun-index.html`	Octave / MatLab
QTM Simulator	`http://www.lix.polytechnique.fr/~durr/Attic/qtm`	C++
Optical Simulator	`http://www.cit.gu.edu.au/~s55086/qucomp`	Java applets
Quantum Turing Machine Simulator		Mathematica
Simulador de M. Hayward	`http://alumni.imsa.edu/~matth/quant`	
UH Quantum Computer	`http://www.cit.gu.edu.au/~s55086/qucomp`	Java
Virtual Quantum Mechanics	`http://www.pha.jhu.edu/~javalab/qubit/qubit.html`	Java

*URLs acessadas em 3 de janeiro de 2006.

Tabela 6: Principais softwares didáticos para Computação Quântica

5.2.4 Simuladores propriamente ditos

> "[Simuladores] são definidos como sendo programas de computador que podem ser executados em uma máquina clássica para simular as ações de um computador quântico. Estas simulações efetivamente envolvem o uso de máquinas que trabalham de acordo com as leis da Física Clássica Newtoniana para simular máquinas que trabalham de acordo com as leis da Mecânica Quântica." (WALLACE, 1999, pág. 1)

O Fraunhofer Quantum Computer Portal, do FIRST,[7] é um *web-site* que oferece serviços de simulação de computadores quânticos gratuitamente na Internet. Neste portal, o pesquisador pode submeter simulações através de uma interface gráfica bastante amigável, para que o sistema as distribua automaticamente por vários nós de um *cluster*. O portal de Fraunhofer permite simulações de circuitos quânticos e Hamiltonianos com até 27 *qubits*. Havendo necessidade, pode-se pedir autorização aos administradores para executar simulações com até 31 *qubits*. Informações mais detalhadas sobre este simulador podem ser encontradas no artigo de Rosé *et al.* (2004).

[7] *Fraunhofer Institut für Rechnerarchitektur und Softwaretechnik.*

Outro simulador muito interessante é o libquantum, uma biblioteca em C para simulação de circuitos quânticos arbitrários. Possui performance e consumo de memória otimizados, simula a descoerência dos estados quânticos e permite a implementação de códigos correção de erro quântico. O pacote já inclui os algoritmos de Shor, para fatoração, e de Grover, para busca em listas não-ordenadas. A documentação, juntamente com uma breve introdução aos simuladores de computadores quânticos pode ser obtida em um artigo dos desenvolvedores (BUTSCHER; WEIMER, 2003). A utilização da biblioteca está exemplificada no código abaixo, onde a função void quantum_qft(int width, quantum_reg *reg), nativa do libquantum, foi modificada para executar a Transformada de Fourier Aproximada.

```
void quantum_aqft(int width, int m, quantum_reg *reg){
    int i, j;
    for(i=width-1; i>=0; i--){
        for(j=width-1; j>i; j--)
            if(j-i<=m) quantum_cond_phase(j, i, reg);
        quantum_hadamard(i, reg);
    }
}
```

O software JaQuzzi foi escrito em Java e permite o desenvolvimento e simulação de circuitos quânticos em ambiente gráfico. O software pode ser obtido gratuitamente na Internet para execução *off-line*, ou pode-se optar pela execução na forma de *applet*. Trata-se de um simulador simples, de fácil utilização, e que atinge bons resultados.

O Quantum Computer Simulator, da empresa japonesa Senko, funciona em Windows ou em Machintosh. Ele permite a simulação de computadores quânticos através de uma interface gráfica amigável e de forma integrada ao Mathematica. As entradas são necessariamente estados da base computacional, mas expressões com coeficientes arbitrários podem ser tratadas no decorrer da simulação. Um versão de demonstração deste simulador está disponível na Internet.

O QCSim é um simulador de código-fonte aberto, disponível gratuitamente na Internet. Inclui exemplos de simulações do protocolo criptográfico BB84[8], teletransporte quântico, algoritmo de Grover, códigos de correção de erro, dentre outros.

[8] O protocolo quântico BB84 foi descoberto por Bennett e Brassard (1984). Uma revisão sobre o assunto, em nível bastante introdutório, encontra-se no artigo de Marquezino e Helayël-Neto (2004).

O OpenQUACS[9] é uma biblioteca escrita em Maple, permitindo a simulação de circuitos quânticos arbitrários. Este software pode ser obtido gratuitamente na Internet, juntamente com alguns exemplos e um tutorial. Este simulador foi resultado da tese de mestrado de McCubbin (2000).

O autor do Quantum Qudit Simulator propõe a simulação de computadores quânticos baseados em sistemas quânticos com mais de dois níveis. Este simulador opera com sistemas quânticos de no máximo dez níveis, devido ao alto consumo de tempo e memória que surge a partir deste ponto. O Quantum Qudit Simulator funciona em Windows 95 ou superior, e foi desenvolvido em Visual Basic.

O simulador Quantum eXpress tem como objetivo realizar simulações levando em consideração o tempo que seria gasto, de fato, em um computador quântico real. O uso do simulador é gratuito, porém apenas via Internet, devido ao código-fonte ser fechado. É o *software* que mais se aproxima do simulador Feynman, discutido na Sec. 5.3.

As idéias usadas no Finite State Machine Simulator estão expostas no artigo de Dunlavey (1998). O simulador QCS holandês foi desenvolvido por Nuyens (2002), autor de uma dissertação de mestrado sobre técnicas de simulação de computadores quânticos. O simulador LGP2 tem como objetivo utilizar técnicas de programação genética para desenvolver algoritmos quânticos melhores que os clássicos.

Simuladores que utilizam Hamiltonianos dependentes do tempo são chamados por Raedt e Michielsen (2004) de emuladores quânticos. Estes têm como principal objetivo emular um *hardware* específico em um computador clássico. Os principais emuladores quânticos são o QCE[10] e o QSS[11]. O QCE simula circuitos quânticos em condições experimentais realísticas, e pode ser obtido gratuitamente na Internet, juntamente com a documentação e alguns exemplos de algoritmos quânticos. O software QSS, desenvolvido em C++, para Windows, pode realizar uma simulação com dezoito *qubits* em cerca três horas, utilizando um computador de 2.4 GHz, com 1 GB de memória RAM. Este emulador está associado à dissertação de mestrado de Schneider (2000).

Estes e outros simuladores relevantes estão listados nas Tabelas 7 e 8.

[9] *Open-source Quantum Computer Simulator.*
[10] *Quantum Computer Emulator.*
[11] *Quantum System Simulator.*

Nome	URL▲	Linguagem	Última versão
CS20c	`http://www.cs.caltech.edu/~ayers/Progress.html`	Java	
Eqcs	`http://home.snafu.de/pbelkner/eqcs`		
Feynman		C++	0.4
Fraunhofer Quantum Computer Portal	`http://www.qc.fraunhofer.de`		
FSM Simulator		C	
jaQuzzi	`http://www.eng.buffalo.edu/~phygons/jaQuzzi/jaQuzzi.html`	Java	0.1
jQuantum	`http://jquantum.sourceforge.net`	Java	
LGP2	`http://hampshire.edu/lspector/code.html`		
Libquantum	`http://www.enyo.de/libquantum/`	C	0.2.4, 11/01/05
Libquantum para Mac OS	`http://libquantum.darwinports.com/`	C	0.2.4
OpenQUACS	`http://userpages.umbc.edu/~cmccub1/quacs/quacs.html`	Maple	22/5/00
QCAD	`http://acolyte.t.u-tokyo.ac.jp/~kaityo/qcad`		1.80
QCE	`http://rugth30.phys.rug.nl/compphys0/qce.htm`		8.1.1, 27/6/04
QCompute		Pascal	
QCS	`http://www.senko-corp.co.jp/qcs`	C++	1.1, 20/2/00
QCS	`http://www.cs.kuleuven.ac.be/~dirkn/thesis`	MatLab	

▲URLs acessadas em 3 de janeiro de 2006.

Tabela 7: Principais simuladores quânticos existentes - parte A

Nome	URL▲	Linguagem	Última versão
QCSim	`http://hissa.nist.gov/~black/Quantum/`	C++	
QDENSITY	`http://www.pitt.edu/~tabakin/QDENSITY`	Mathematica	
QDNS	`http://www.hit.bme.hu/people/imre/pages/QDNS`		
QGAME	`http://hampshire.edu/lspector/qgame.html`	Lisp	
Qsims	`http://qsims.sourceforge.net`		
QSS		C++	
QuaSi	`http://iaks-www.ira.uka.de/QIV/QuaSi`		2
Quantum algorithm designer	`https://www-users.cs.york.av.uk/~sok/QAD`		
Quantum eXpress	`https://www.research.ge.com/quantum/index.jsp`		Beta 1
Quantum-octave	`https://quantum-octave.sourceforge.net`	Octave	
Quantum Qudit Simulator	`http://www.programmersheaven.com`, ou `http://www.freedownloadscenter.com`	Visual Basic	1.1
Quack!	`http://www.physics.uq.edu.au/people/rohde/blog/?page_id=20`	MatLab	pi/2
QuCalc	`http://crypto.cs.mcgill.ca/QuCalc/`	Mathematica	
QuIDDPro	`http://vlsicad.eecs.umich.edu/Quantum/qp`		2.1.4
SHORNUF	`http://thegreves.com/david/QDD/qdd.html`		
SimQubit	`http://www.cybernet.com/programs/380/quantum/`	C++/Q++	
Zeno	`http://www.gia.dsc.ufcg.edu.br/zeno`	Java	

▲URLs acessadas em 3 de janeiro de 2006.

Tabela 8: Principais simuladores quânticos existentes - parte B

5.3 O simulador Feynman

5.3.1 Motivação

Apesar de não ser um fato demonstrado matematicamente, acredita-se que Richard Feynman estivesse certo ao sugerir que a simulação de sistemas físicos quânticos através de sistemas clássicos tenha complexidade inerentemente exponencial. Assim sendo, todos esforços dos programadores a fim de obter melhor desempenho nas simulações de computadores quânticos são insuficientes para que resultados consideráveis sejam atingidos. Um *software* que simule poucos *qubits* deve se tornar cerca de duas vezes mais eficiente para simular apenas um *qubit* a mais.

A proposta do simulador Feynman, portanto, é realizar simulações que se aproximem tanto quanto possível da realidade física (MARQUEZINO; MELLO JUNIOR, 2004c; MARQUEZINO *et al.*, 2005). Todas as operações realizadas nas simulações são separadas em operações com correspondente físico e sem correspondente físico. Contabilizando o tempo total e eliminando tudo o que não possui correspondente físico, é possível saber quanto tempo o cálculo levaria se executado em um computador quântico real.

Desta forma, pode-se utilizar o simulador Feynman para obter uma melhor compreensão dos fatores que levam à construção de algoritmos quânticos eficientes. Os esforços de programação deixam de visar desempenho computacional, e passam a ter como alvo a interpretação física dos resultados. Devido a esta abordagem o simulador Feynman é, em geral, mais lento que os demais simuladores.

5.3.2 Implementação

Alguns detalhes da implementação do simulador Feynman são expostos no Apêndice C. Nesta Seção faz-se um breve comentário sobre as idéias utilizadas no simulador.

Ao longo do texto alguns termos técnicos são utilizados, de modo que convém fazer uma breve apresentação dos mesmos. Um processo é uma instância de um programa em execução (BACON; HARRIS, 2003; TANENBAUM; WOODHULL, 2000). Um importante modelo de algoritmos paralelos é conhecido como mestre-escravo.

Neste modelo, há um processo conhecido como mestre, responsável por gerar trabalho para ser executado pelos processos conhecidos como escravos (GRAMA *et al.*, 2003). MPI[12] é um padrão para passagem de mensagens proposta para aplicações que utilizam um ambiente computacional distribuído (MARQUEZINO; MELLO JUNIOR, 2004c; SNIR *et al.*, 1996). A linguagem de programação da implementação foi o C++, que utiliza o paradigma da programação orientada a objetos (DEITEL; DEITEL, 2001). Portanto, pelo menos três conceitos são freqüentemente mencionados nesta dissertação: classe, método, atributo e objeto. Uma classe pode ser vista como um tipo definido pelo usuário. As classes possuem dados e um conjunto de funções que os manipulam. Os componentes de dados são chamados de atributos, enquanto os componentes de função da classe são chamados de métodos. Uma instância de uma classe é chamada de objeto.

Uma característica importante, que distingue os diferentes simuladores, é a representação dos estados quânticos. A maioria dos simuladores utiliza a representação na forma de um vetor de números complexos, formado a partir do produto tensorial de todas as sub-partes do sistema.[13] Em simuladores paralelos, o grande vetor de números complexos costuma ser distribuído entre diversos nós. A desvantagem desta abordagem é a perda de interpretação física, uma vez que durante a simulação não é possível isolar o cálculo referente a cada *qubit*.

O simulador Feynman, por outro lado, representa os estados em sua forma fatorada, isolando o estado dos *qubits* em processos distintos. Desta forma, pode-se distribuir o cálculo entre diversos processadores sem perder interpretação física. Sempre que não ocorre emaranhamento na simulação (por exemplo, no caso da QFT, ou da QFT inversa, com entradas projetadas) esta simulação é bastante eficiente, principalmente em relação ao consumo de memória. Quando um *qubit* está emaranhado ele é descrito pelo vetor de estado que agrupa todos os *qubits* emaranhados entre si. Naturalmente, cada *qubit* do emaranhamento precisa representar o mesmo estado, em uma redundância que pode parecer absurda do ponto de vista computacional, mas que é necessária para preservar a interpretação física.

Cada *qubit* da simulação possui um número de identificação. Como a imple-

[12] *Message Passing Interface.*

[13] Algumas exceções são o QDD, que utiliza representação através de *binary decision diagrams*, o Quantum Fog e o Qubiter, que utilizam representação através de redes Bayesianas (TUCCI, 1999; WALLACE, 1999).

mentação utilizou MPI, e cada *qubit* foi isolado em um único processo, é natural aproveitar o número do *rank*, i.e., a numeração dada pelo MPI para cada processo. O *qubit* da implementação possui ainda um vetor de números inteiros para armazenar os identificadores de todos os demais *qubits* com os quais ele está emaranhado.

Em relação às portas lógicas quânticas, a maioria dos simuladores as representa através de grandes matrizes de números complexos, de dimensão $2^n \times 2^n$, onde n é o número de *qubits* na simulação. As portas lógicas utilizadas no simulador Feynman, por outro lado, são sempre matrizes de dimensão 2×2. As portas possuem ainda um atributo indicando se a operação é controlada, e qual é o *qubit* de controle. Desta forma, é possível simular qualquer porta lógica, sem a necessidade de matrizes com dimensões maiores.

Os circuitos quânticos são implementados através de uma classe própria, com métodos, por exemplo, para preparação da seqüência de portas lógicas e para envio das mesmas aos respectivos *qubits*.

O programa principal do simulador Feynman, responsável pela simulação da QFT Aproximada inversa, está exposto no Apêndice C. Em um primeiro momento, são realizadas algumas operações de inicialização e de verificação. Logo após, finaliza-se eventuais processos excedentes. Em uma simulação com n *qubits* pelo simulador Feynman são necessários $n + 1$ processos: n escravos, representando *qubits*, e um mestre, representando o equipamento de controle. Em seguida, o processo mestre prepara o circuito para simulação. Cada processo escravo também passa por uma inicialização, onde faz a leitura do arquivo de entrada para extrair seu vetor de estado e, se necessário, a lista de *qubits* com os quais está emaranhado. O simulador Feynman permite a utilização de entradas superpostas, e até emaranhadas, ao contrário de simuladores como o QCS, da empresa japonesa Senko, ou o jaQuzzi.

Após as inicializações cada processo escravo fica parado, à espera de uma instrução via MPI. Quando o processo mestre termina de preparar o circuito, ele passa a enviar portas lógicas a estes processos, — que representam os *qubits* — aguardando sempre uma mensagem de confirmação de porta executada. O processo escravo, quando recebe uma porta, executa e envia uma mensagem de "pronto" para o processo mestre. A única peculiaridade ocorre quando o *qubit* está emaranhado, devido à necessidade de manter a interpretação física. Neste caso, ao receber a porta, ele

deve retransmiti-la aos demais *qubits* com os quais está emaranhado. Após efetuar seu cálculo, o processo do *qubit* emaranhado deve esperar confirmação de todos os demais, para somente então enviar a mensagem de "pronto" ao mestre.

Após o recebimento de uma porta lógica por um *qubit* existem duas possibilidades: o cálculo sobre estados fatorados e o cálculo sobre estados emaranhados. Na primeira situação, o *qubit* apenas calcula o produto entre a matriz da porta lógica e o vetor de estado, verificando antes o estado do *qubit* de controle, se houver. Na segunda situação, a aplicação da porta envolve a propriedade de linearidade da mesma. Por exemplo, o cálculo da porta de Hadamard no *qubit* menos significativo de um estado emaranhado pode ser realizado como

$$\left(I^{\otimes 2} \otimes H\right)\left(\alpha\left|000\right\rangle + \beta\left|111\right\rangle\right) = \alpha\left(\left|00\right\rangle \otimes H\left|0\right\rangle\right) + \beta\left(\left|11\right\rangle \otimes H\left|1\right\rangle\right), \tag{5.8}$$

de forma que qualquer porta pode ser calculada através de matrizes 2×2.

Os *swaps* podem ser realizados através de três portas CNOT (conforme Figura 4), ou através de método da classe Circuito. Neste último caso, pode-se realizar o *swap* entre *qubits* projetados, com um processo enviando seu estado para o outro, e efetuando a troca; ou entre *qubits* emaranhados, de modo similar ao artigo de Hsu (2002).

Para finalizar a simulação, o processo mestre envia uma mensagem para cada *qubit*, para que estes salvem o resultado parcial da simulação em um arquivo texto, e finalizem-se. Após a finalização de todos os processos escravos, o processo mestre agrupa os resultados parciais, adiciona algumas estatísticas, e salva tudo em um arquivo, finalizando logo em seguida. O formato do arquivo é descrito no Apêndice C.

Os tempos são contabilizados internamente através de comandos do MPI juntamente com métodos da classe Tempo. O comando MPI utilizado com este propósito foi o MPI::Wtime(). Todo tempo contabilizado é classificado como real ou virtual, sendo estes existentes somente na simulação, sem correspondente no computador quântico real; e aqueles existentes, de fato, em uma eventual realização física do computador quântico. O tempo virtual, portanto, deve ser desconsiderado na análise de desempenho do algoritmo quântico.

5.4 Metodologia das simulações

O simulador foi implementado em C++ com MPI, utilizando a versão MPICH-1.2.7p1, compilando através do mpiCC. Utilizou-se a opção de otimização do compilador, `-O3`. A execução do *software* deu-se através do comando `mpirun`. Quanto ao sistema operacional, utilizou-se a distribuição Linux Red Hat apenas em fases iniciais do desenvolvimento do software. Mais recentemente, a distribuição Linux SUSE foi utilizada. Além do MPICH, utilizou-se a biblioteca matemática GSL[14] 1.7, e a biblioteca VBLib, para manipulação de *strings*.

Os computadores utilizados nas simulações foram estações Intel Pentium 4, com CPU de 2.8 GHz, memória principal de 512 MB, memória cache de 1024 KB, e sistema operacional Linux SUSE 10. A rede utilizada operava a 100 Mbps. Apesar de inicialmente o *software* ter sido testado com muitos computadores, preferiu-se restringir o uso a apenas dois, durante as tomadas de tempo. Esta escolha teve como objetivo reduzir o tempo de comunicação entre os processos. Versões futuras do *software* Feynman poderão tratar de forma mais adequada as comunicações, permitindo simulações envolvendo mais computadores, sem contaminação dos resultados experimentais.

O algoritmo escolhido para as simulações foi a QFT inversa. A escolha do algoritmo inverso teve motivação puramente histórica, visto que o algoritmo de Shor, um dos mais importantes algoritmos quânticos já desenvolvidos, utiliza a transformada inversa. Os resultados obtidos, no entanto, possuem interpretação independente deste fato.

As entradas da simulação foram estados projetados e estados emaranhados do tipo gato de Schrödinger. A terminologia gato de Schrödinger é utilizada em artigos como o de Leibfried *et al.* (2005) e refere-se a estados

$$|\psi\rangle = \frac{|00\cdots0\rangle + |11\cdots1\rangle}{\sqrt{2}}, \tag{5.9}$$

isto é, superposições de dois estados maximalmente diferentes. A origem deste termo remonta ao início do século passado, época de intensas discussões filosóficas acerca da Mecânica Quântica.[15]

[14] *GNU Scientific Library*

[15] A mais famosa dessas discussões filosóficas foi o paradoxo EPR (EINSTEIN; PODOLSKY; ROSEN, 1935).

Os resultados da QFT Aproximada inversa foram analisados para o caso de $n = 9$ *qubits*, tanto no caso das entradas projetadas como das entradas emaranhadas, de forma a verificar a qualidade das soluções variando o parâmetro m. A visualização dos resultados deu-se através de gráficos das amplitudes dos estados obtidos, considerando-se partes real e imaginária separadamente. A qualidade dos resultados ainda foi quantificada através da medida de fidelidade, comparando o caso exato ($m = 9$) com os aproximados ($1 \leq m \leq 8$).

O objetivo da análise de qualidade das soluções é a compreensão do comportamento da QFT Aproximada com diversos parâmetros. O resultado esperado é a diminuição da fidelidade, de forma lenta para valores de m próximos à quantidade de *qubits*, podendo acentuar-se apenas quando m fica muito próximo de um. Espera-se ainda que os gráficos das soluções aproximadas sejam visualmente parecidos com os da solução exata, para valores de m próximos à quantidade de *qubits*.

Outro fator analisado foi o tempo de processamento consumido pela simulação. Neste caso as entradas também foram de dois tipos, projetadas e gatos de Schrödinger, porém variavam a quantidade de *qubits*. O primeiro tipo de entrada contemplou estados variando entre um e cem *qubits*. O segundo tipo de entrada, por envolver cálculos de complexidade muito maior, contemplou estados variando entre um e nove *qubits*. Os parâmetros utilizados foram $m = n$ e $m = \log n$ onde n é a quantidade de *qubits*.

As tomadas de tempo têm como objetivo verificar diferenças entre os tempos de processamento da QFT exata, e da aproximada através de um parâmetro típico. O resultado esperado é uma curva menos acentuada para o algoritmo aproximado, em relação ao algoritmo exato.

A utilização de entradas projetadas e emaranhadas permite verificar a importância do emaranhamento no desenvolvimento de algoritmos quânticos eficientes. Como as entradas projetadas nada mais são que estados clássicos, apenas representados pelo formalismo da Mecânica Quântica, espera-se que a simulação introduza pouco tempo virtual, de forma que o gráfico do tempo neste caso deve ser parecido com o gráfico da complexidade do algoritmo quântico. Por outro lado, as entradas emaranhadas não possuem correspondente clássico, e por isso o gráfico do tempo neste caso deve ser uma curva bem mais acentuada, com crescimento exponencial. Entretanto, o simulador Feynman deve permitir a extração do tempo que seria gasto

em um computador quântico real, (a menos de um fator multiplicativo dependente do tipo de *hardware* utilizado) e este deve corresponder à complexidade do algoritmo quântico, obtida pela teoria.

5.5 Resultados e discussões

5.5.1 Qualidade das soluções

A solução da QFT inversa (exata) sobre um estado projetado de nove *qubits* encontra-se no gráfico da Figura 12. O gráfico mostra apenas a parte real das amplitudes. O gráfico da parte imaginária não apresenta nenhuma diferença visual. Os resultados da QFT inversa atuando sobre o mesmo estado projetado, desta vez diminuindo o parâmetro m, encontram-se nos gráficos da Figura 13. Nota-se que os primeiros gráficos são praticamente idênticos ao gráfico da solução pelo algoritmo exato.

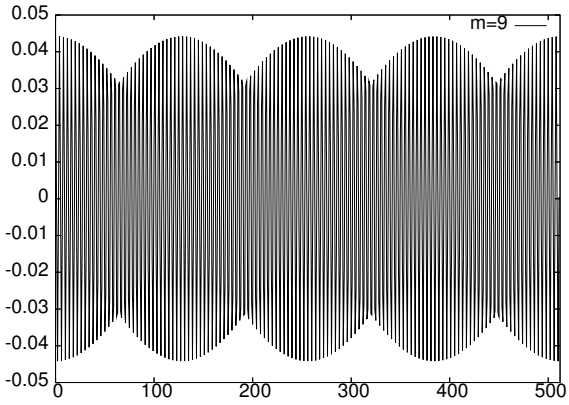
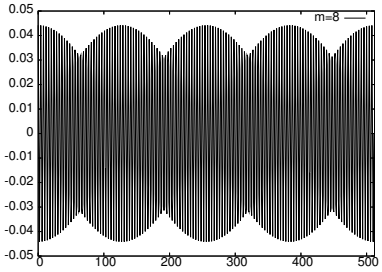


Figura 12: Resultado exato da QFT inversa (parte real) sobre o estado projetado $|127\rangle$

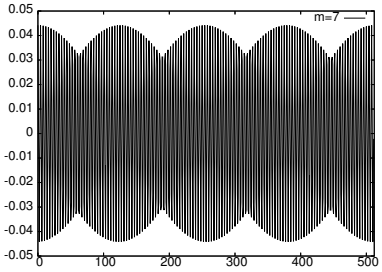
Os resultados são quantificados através da fidelidade, exposta no gráfico da Figura 14. A definição para fidelidade encontra-se na página 12. Nota-se que, de fato, o gráfico permanece muito perto da fidelidade máxima quando $5 \leq m \leq 9$.

Portanto, os resultados referentes à qualidade das soluções da QFT Aproximada inversa, para entradas projetadas, foram coerentes com as previsões teóricas.

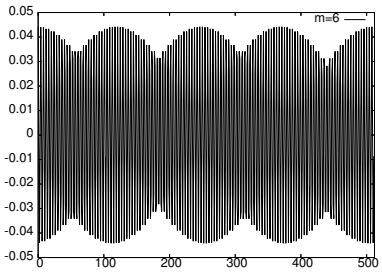
O resultado da QFT inversa atuando sobre um gato de Schrödinger de nove *qubits*, obtido pelo algoritmo exato, encontra-se nos gráficos das Figuras 15 e 17 — este contendo a parte imaginária da solução, e aquele contendo a parte real.



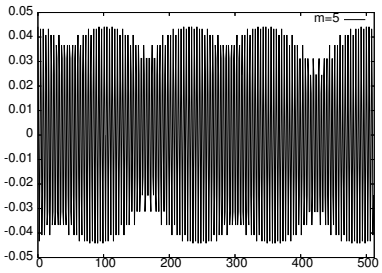
(a) Parâmetro $m = 8$



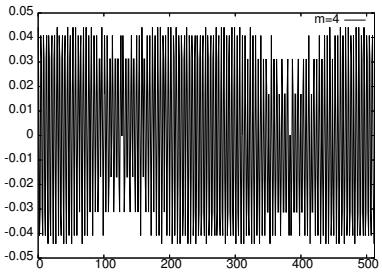
(b) Parâmetro $m = 7$



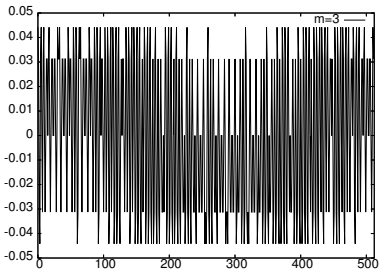
(c) Parâmetro $m = 6$



(d) Parâmetro $m = 5$

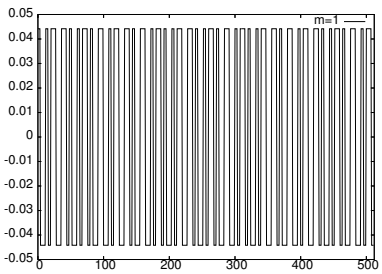


(e) Parâmetro $m = 4$



(f) Parâmetro $m = 3$

(g) Parâmetro $m = 2$



(h) Parâmetro $m = 1$

Figura 13: Resultados aproximados da QFT inversa (parte real) sobre o estado projetado $|127\rangle$

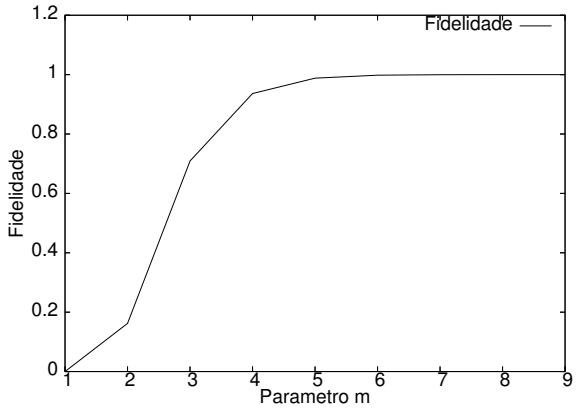


Figura 14: Fidelidade do resultado da QFT inversa sobre um estado projetado em função do parâmetro m utilizado

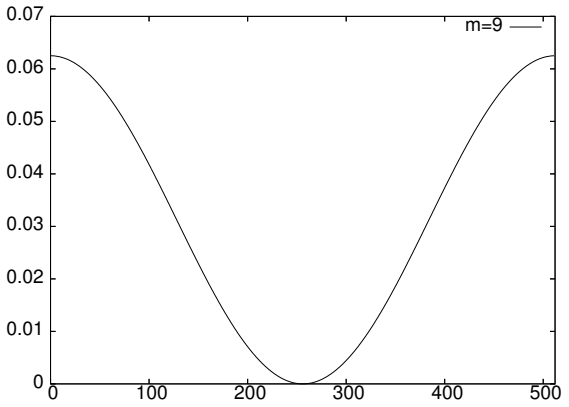
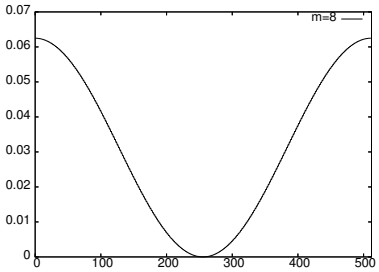


Figura 15: Resultado exato da QFT inversa (parte real) sobre o gato de Schrödinger de nove *qubits*

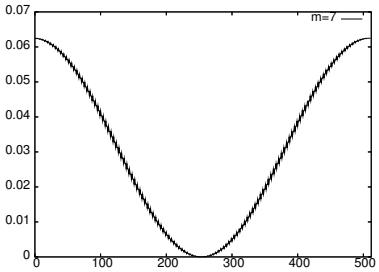
Os resultados da QFT inversa atuando sobre o mesmo estado emaranhado, porém diminuindo o parâmetro m, encontram-se nos gráficos das Figuras 16 e 18, este referente à parte imaginária, e aquele referente à parte real. Nota-se que os primeiros gráficos são praticamente idênticos ao gráfico da solução pelo algoritmo exato.

Os resultados são quantificados através da fidelidade, exposta no gráfico da Figura 19. Nota-se que, de fato, o gráfico permanece muito perto da fidelidade máxima quando $5 \leq m \leq 9$. A peculiaridade, neste caso, foram os resultados da fidelidade para valores de m menores que três. Em vez de continuar decrescendo, a fidelidade aumentou mais um pouco. Este resultado, apesar de imprevisto, não representa nenhuma inconsistência, tratando-se apenas de uma particularidade do estado escolhido para entrada.

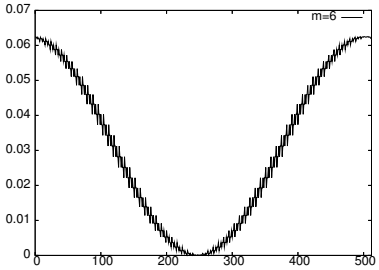
Portanto, os resultados referentes à qualidade das soluções da QFT Aproxi-



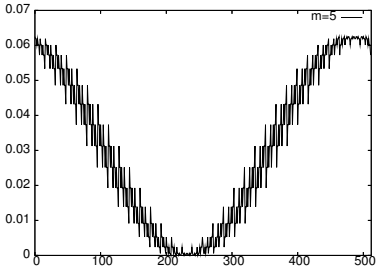
(a) Parâmetro $m = 8$



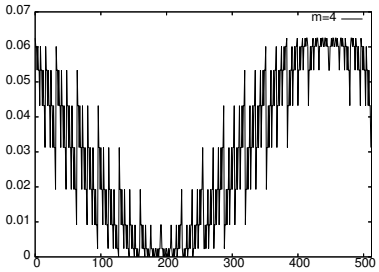
(b) Parâmetro $m = 7$



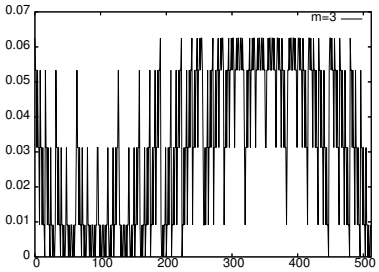
(c) Parâmetro $m = 6$



(d) Parâmetro $m = 5$

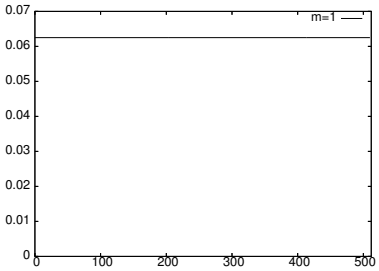


(e) Parâmetro $m = 4$



(f) Parâmetro $m = 3$

(g) Parâmetro $m = 2$



(h) Parâmetro $m = 1$

Figura 16: Resultados aproximados da QFT inversa (parte real) sobre o gato de Schrödinger de nove *qubits*

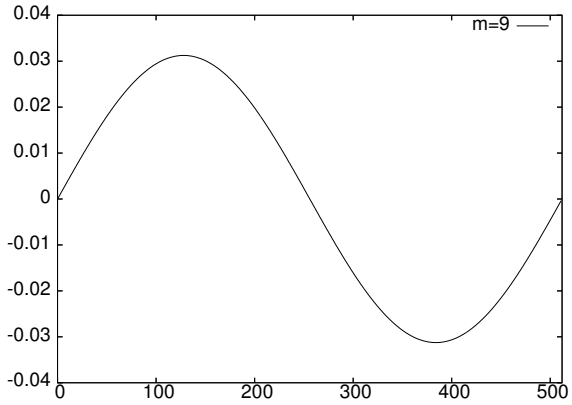


Figura 17: Resultado exato da QFT inversa (parte imaginária) sobre o gato de Schrödinger de nove *qubits*

mada inversa, para entradas emaranhadas, também foram coerentes com as previsões teóricas.

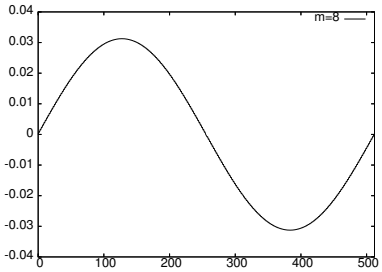
5.5.2 Medição de tempo

No gráfico da Figura 20 estão expostos os tempos totais de simulação da QFT inversa exata em dois casos diferentes: com entradas emaranhadas do tipo gato de Schrödinger e com entradas projetadas. No primeiro caso nota-se uma curva exponencial, enquanto no segundo caso a curva é bastante suave, ajustando-se a uma parábola. Neste gráfico não são exibidos os resultados com mais de quarenta *qubits*, permitindo uma comparação mais nítida entre o caso projetado e o emaranhado. Resultados para entradas projetadas com mais de quarenta *qubits*, no entanto, estão expostos na Figura 22(a).

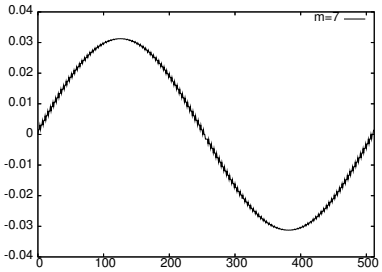
No gráfico da Figura 21 observam-se os tempos totais de simulação da QFT inversa aproximada, com parâmetro $m = \log n$. Foram consideradas as simulações com entradas do tipo gato de Schrödinger e entradas projetadas. Os resultados foram semelhantes aos da Figura 20, apenas com curvas um pouco mais suaves. No primeiro caso a curva ainda é claramente exponencial. No caso das entradas projetadas a curva que se ajustou bem aos dados[16] é da forma $an \log(bn) + c$, coerente portanto com a complexidade $O(nm) = O(n \log n)$ prevista pela teoria.

A Figura 22 faz uma comparação entre os resultados já mencionados anterior-

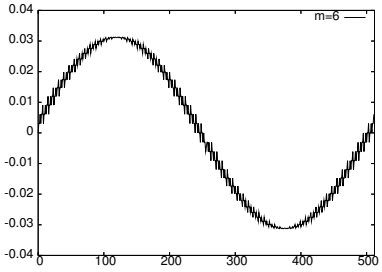
[16]Utilizou-se o método não-linear de mínimos quadrados Marquardt-Levenberg, implementado pelo gnuplot.



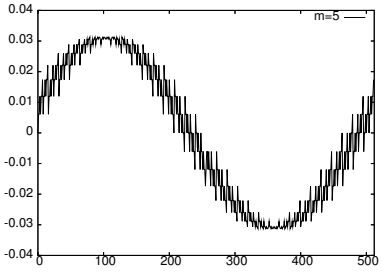
(a) Parâmetro $m = 8$



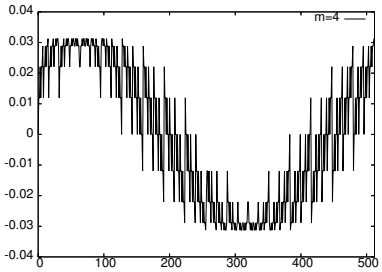
(b) Parâmetro $m = 7$



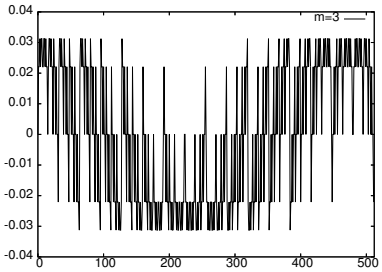
(c) Parâmetro $m = 6$



(d) Parâmetro $m = 5$

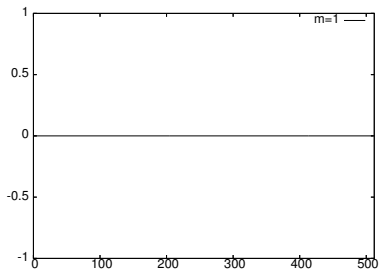


(e) Parâmetro $m = 4$



(f) Parâmetro $m = 3$

(g) Parâmetro $m = 2$



(h) Parâmetro $m = 1$

Figura 18: Resultados aproximados da QFT inversa (parte imaginária) sobre o gato de Schrödinger de nove *qubits*

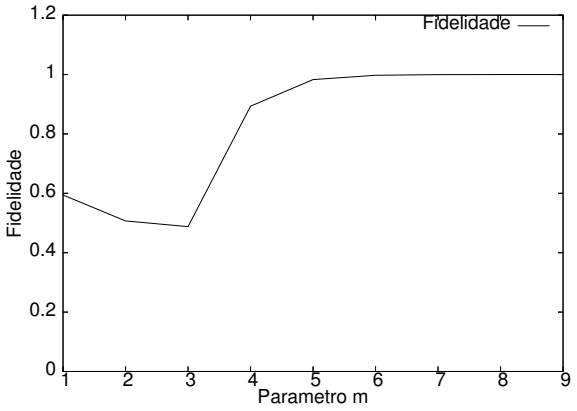


Figura 19: Fidelidade do resultado da QFT inversa sobre o gato de Schrödinger em função do parâmetro m utilizado

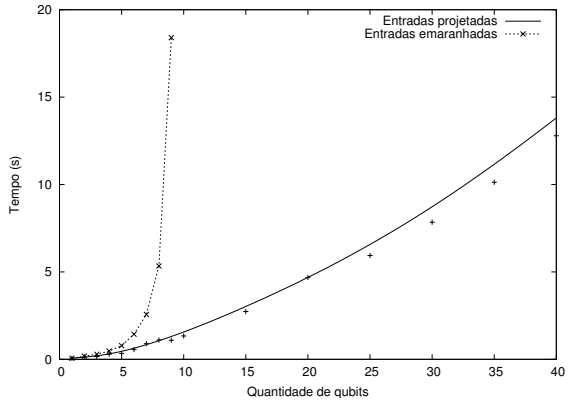


Figura 20: Tempo total gasto pela simulação da QFT inversa exata, sobre gatos de Schrödinger e entradas projetadas

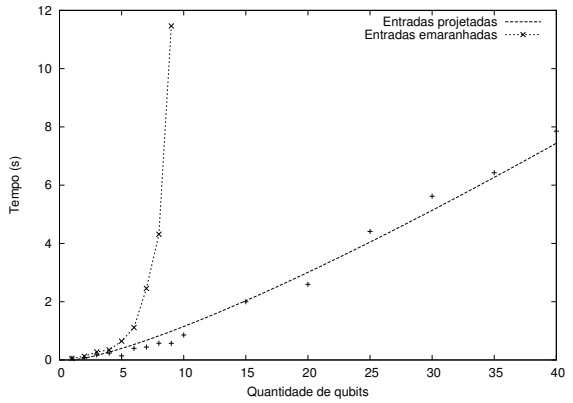
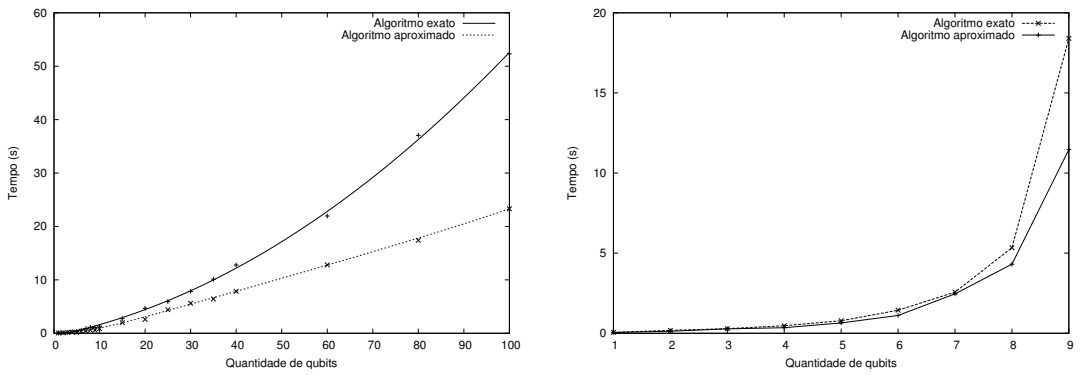


Figura 21: Tempo total gasto pela simulação da QFT inversa aproximada, sobre gatos de Schrödinger e entradas projetadas

mente, apenas destacando a diferença entre o algoritmo exato e o aproximado. No caso das entradas projetadas a diferença é bem nítida. Nas simulações envolvendo entradas emaranhadas, entretanto, o tempos totais foram quase idênticos. Para obter resultados mais expressivos no gráfico de tempo total seria necessário realizar simulações com número maior de *qubits* emaranhados, o que ainda não é possível na atual versão do simulador Feynman. Apesar desta limitação, as simulações realizadas permitiram observações interessantes quando o tempo virtual foi descartado. Estes resultados estão expostos na Figura 23. Percebe-se que os resultados experimentais estão de fato consistentes com as previsões teóricas, visto que o gráfico do algoritmo exato se ajusta a uma parábola, e do algoritmo aproximado, a uma curva $O(n \log n)$.



(a) Atuando em estados projetados

(b) Atuando em gatos de Schrödinger

Figura 22: Tempo total gasto pela simulação da QFT inversa, comparando o algoritmo exato e o aproximado

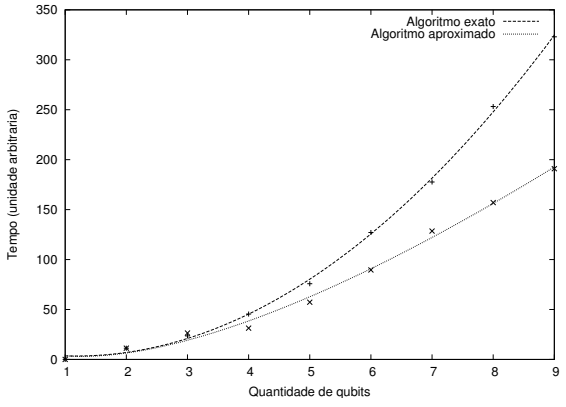


Figura 23: Tempo real gasto pelo computador quântico na execução da QFT inversa com entradas emaranhadas, comparando o algoritmo exato com o aproximado

6 Conclusões

Nesta dissertação, inicialmente foi feita uma revisão dos conceitos fundamentais da Computação Quântica e da DFT clássica. Em seguida, o algoritmo clássico de FFT foi descrito, conforme o livro de Knuth. Partindo da descrição matricial do algoritmo clássico, uma revisão da QFT foi apresentada de forma diferente da encontrada na literatura. Em vez de fornecer o algoritmo quântico pronto juntamente com uma demonstração de corretude, o caminho inverso foi percorrido, apresentando em detalhes o processo de construção do algoritmo quântico a partir do clássico.

Inicialmente foi possível identificar a forma matricial correspondente a cada passo do algoritmo iterativo de FFT, descrito por Knuth. Provou-se que estas matrizes são unitárias e, portanto, potencialmente realizáveis em um computador quântico. No entanto, as matrizes ainda eram muito complexas para serem realizadas em um único processo físico. Ao desenvolver um algoritmo quântico é necessário formulá-lo em termos de portas lógicas quânticas universais, caso contrário não há garantias de que ele possa ser implementado em um laboratório. As matrizes dos passos do algoritmo clássico foram então decompostas segundo a decomposição QR, obtendo-se matrizes ortogonais e diagonais.

As matrizes ortogonais puderam ser decompostas no produto tensorial de identidades e matrizes de Hadamard. Desta forma, ficou demonstrado que elas correspondem à aplicação de portas atuando em apenas um *qubit* por vez. Analisando a estrutura das matrizes restantes, diagonais, percebeu-se que elas correspondem à aplicação de portas lógicas controladas atuando em apenas dois *qubits* de cada vez.

Este processo permitiu a construção de um algoritmo quântico eficiente para cálculo da DFT, reproduzindo o algoritmo original de Coppersmith. Similarmente, partindo dos conceitos de DFT Aproximada e de Transformada de Hadamard, mostrou-se como são desenvolvidos algoritmos quânticos eficientes para estes

cálculos. Neste caso, constatou-se que a QFT Aproximada e a Transformada de Hadamard Quântica podem ser obtidas a partir da QFT exata, através da eliminação de portas controladas que efetuam mudanças de fase muito pequenas, reproduzindo também a conclusão de Coppersmith.

Como a construção do algoritmo quântico a partir do clássico vem sendo omitida pela literatura especializada, espera-se que a dissertação ajude a comunidade científica a desenvolver novos algoritmos quânticos eficientes, utilizando técnicas semelhantes.

Outro assunto abordado pela dissertação foi a simulação de algoritmos quânticos através de computadores clássicos. Apesar de extremamente ineficiente, este tipo de simulação desempenha papel fundamental dentro da Computação Quântica, já que para desenvolver novos algoritmos quânticos é necessário também testá-los. Simuladores, no entanto, não devem simplesmente fornecer o resultado do cálculo, mas devem permitir a extração de informações importantes acerca do algoritmo simulado. Portanto, um bom simulador de computadores quânticos pode ser uma valiosa ferramenta ao pesquisador, no processo de desenvolvimento de algoritmos quânticos eficientes. Dentro desta proposta foi apresentado o simulador Feynman (MARQUEZINO; MELLO JUNIOR, 2004c; MARQUEZINO *et al.*, 2005), atualmente na versão 0.4. Apesar de ainda estar em desenvolvimento, o simulador Feynman produz resultados consistentes com os obtidos por outros simuladores, além de permitir uma análise de tempo de processamento mais detalhada.

Algumas simulações foram realizadas e os resultados foram satisfatórios, de acordo com o objetivo inicial. O funcionamento da QFT Aproximada foi ilustrado, e pôde-se perceber que a qualidade dos resultados melhora quando parâmetros m a partir de $\log n$ são utilizados. Quando o parâmetro m é reduzido até valores muito baixos, próximos de um, percebe-se claramente a perda de qualidade nos resultados, indicando que estes se aproximam da Transformada de Hadamard. Além da análise visual das soluções, foi feita a medida da fidelidade entre resultados aproximados e exatos. Foram analisadas entradas emaranhadas, do tipo gato de Schrödinger, e entradas da base computacional.

O tempo também foi considerado nos experimentos computacionais. Neste caso, notou-se uma curva bem mais suave nos experimentos envolvendo entradas projetadas do que nos experimentos com entradas emaranhadas. Este resultado era pre-

visto, pois como a proposta do simulador Feynman é imitar, tão fielmente quando possível, a realidade dos sistemas físicos quânticos, as simulações com entradas emaranhadas deviam introduzir muito tempo virtual, ao contrário da simulações com entradas projetadas.

A comparação entre algoritmo exato e aproximado foi bem mais nítida nas simulações envolvendo entradas projetadas. Neste caso, o gráfico de esforço computacional na simulação do algoritmo exato foi próximo a uma parábola. Na simulação do algoritmo aproximado com parâmetro $m = \log n$ a curva foi consistente com $O(n \log n)$.

Nas simulações envolvendo gatos de Schrödinger a diferença foi mais sutil. O algoritmo aproximado foi mais rápido, porém na análise de tempo total de processamento a diferença foi pequena. Isto se deve ao fato de poucos *qubits* terem sido usados na simulação, de forma que o comportamento assintótico do gráfico não pôde ser observado com clareza. No entanto, a análise do tempo real, i.e., o tempo que seria gasto em um computador quântico, revelou resultados bastante nítidos, e coerentes com a teoria. Neste caso, foi possível observar uma diferença maior entre os tempos do algoritmo exato e do aproximado, indicando que o algoritmo teria complexidade $O(n^2)$ no primeiro caso, e $O(n \log n)$ no segundo. Apesar dos objetivos da presente dissertação terem sido alcançados mesmo com poucos *qubits* emaranhados, versões futuras do simulador Feynman poderão viabilizar experimentos mais interessantes.

Em trabalhos futuros pode-se aplicar a técnica aqui descrita a outros algoritmos clássicos, na tentativa de desenvolver novos algoritmos quânticos eficientes. Em relação às simulações, deve-se continuar o desenvolvimento do simulador Feynman. Em seguida, pode-se realizar experimentos com maior quantidade de *qubits* emaranhados, e envolvendo algum modelo de descoerência. Pode-se também estudar o comportamento da QFT Aproximada quando esta é utilizada em diferentes algoritmos.

APÊNDICE A - Decomposição QR

A decomposição QR decompõe uma matriz A quadrada como

$$A = QR, \tag{A.1}$$

onde Q é uma matriz unitária — ortogonal, se a matriz A for real — e R é uma matriz triangular superior. O conceito ainda poderia ser generalizado para matrizes retangulares, porém esta análise sairia do escopo da presente dissertação.

Há ao menos três métodos bem conhecidos para o cálculo da decomposição QR: as rotações de Givens, as transformações de Householder, e a decomposição por Gram-Schmidt. Na decomposição das matrizes $P^{(s)}$, no Capítulo 4, utilizou-se o último método.

Por simplicidade, pode-se considerar que a matriz A seja real. Pode-se aplicar o processo de ortogonalização de Gram-Schmidt às colunas da matriz

$$A = \left(a_0 | a_1 | \cdots | a_{n-1}\right), \tag{A.2}$$

de forma a obter um conjunto de vetores ortonormais, $\{e_0, e_1, \cdots, e_{n-1}\}$. Denotando por $\operatorname{proj}_e a$ a projeção do vetor a sobre o subespaço linear gerado por e, tem-se

$$\begin{aligned}
e_0 &= \frac{a_0}{\|a_0\|} = \frac{u_0}{\|u_0\|}, \\
e_1 &= \frac{a_1 - \operatorname{proj}_{e_0} a_1}{\|a_1 - \operatorname{proj}_{e_0} a_1\|} = \frac{u_1}{\|u_1\|}, \\
e_2 &= \frac{a_2 - \operatorname{proj}_{e_0} a_2 - \operatorname{proj}_{e_1} a_2}{\|a_2 - \operatorname{proj}_{e_0} a_2 - \operatorname{proj}_{e_1} a_2\|} = \frac{u_2}{\|u_2\|}, \\
&\vdots \\
e_k &= \frac{a_k - \sum_{j=0}^{k-1} \operatorname{proj}_{e_j} a_k}{\|a_k - \sum_{j=0}^{k-1} \operatorname{proj}_{e_j} a_k\|} = \frac{u_k}{\|u_k\|}.
\end{aligned} \tag{A.3}$$

Pode-se reescrever as Equações (A.3) em termos das colunas a_k.

$$\begin{aligned} a_0 &= e_0\|u_0\| \\ a_1 &= \text{proj}_{e_0} a_1 + e_1\|u_1\| \\ a_2 &= \text{proj}_{e_0} a_2 + \text{proj}_{e_1} a_2 + e_2\|u_2\| \\ &\vdots \\ a_k &= \sum_{j=0}^{k-1} \text{proj}_{e_j} a_k + e_k\|u_k\| \end{aligned} \tag{A.4}$$

A projeção de um vetor a_k sobre um vetor e_j é o produto interno entre os dois. Portanto, as Equações (A.4) podem ser reescritas em uma forma matricial que recupera a Equação (A.1).

$$A = (e_0|e_1|\cdots|e_{n-1}) \begin{pmatrix} \|u_0\| & \langle e_1, a_2\rangle & \langle e_1, a_3\rangle & \cdots \\ 0 & \|u_1\| & \langle e_2, a_3\rangle & \cdots \\ 0 & 0 & \|u_2\| & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{A.5}$$

A primeira matriz, de fato, é ortogonal, e a segunda é triangular superior. Na fatoração das matrizes complexas $P^{(s)}$, a única modificação foi a normalização nas Equações (A.3). Em vez de $\|u_k\|$, foram utilizados outros números complexos de mesma norma, de forma a obter vetores e_k sempre reais. Isto pôde ser feito devido a uma propriedade das matrizes $P^{(s)}$, a saber, que suas colunas são reais ou múltiplas de colunas reais.

APÊNDICE B - Programas em Maple

B.1 Inicialização

É recomendável inicializar a *worksheet* com os comandos abaixo. O primeiro deles, `restart`, serve para apagar o conteúdo das variáveis utilizadas anteriormente. Sua utilização evita a obtenção acidental de resultados errados. O segundo comando, `with(LinearAlgebra)`, serve para carregar o pacote LinearAlgebra, permitindo a utilização de comandos úteis. O terceiro comando, `interface(rtablesize=25)`, permite uma melhor visualização das matrizes obtidas. Se este comando não for utilizado, apenas casos com poucos *qubits* poderão ser testados.

```
> restart;
> with(LinearAlgebra):
> interface(rtablesize=25):
```

B.2 Procedimentos auxiliares

O comando abaixo define uma função que toma os valores de n e x, retornando a 2^n-ésima raiz principal da unidade elevada à potência x.

```
> w:=(x,n)->exp((2*Pi*I*x)/(2^n)):
```

Em certas situações é útil definir as raízes da unidade sem calcular a exponencial explicitamente. Esta definição alternativa é dada pelo comando a seguir.

```
> w:=proc(expo,n)
    local E1;
```

```
    E1 := (expo mod 2^n);
    return piecewise(E1<2^n, omega^E1, -omega^(E1-2^n));
  end proc:
```

Deve-se lembrar que apenas um dos comandos acima pode ser utilizado na *worksheet*.

O comando abaixo define uma função que toma como entradas os valores de s e $j \equiv (j_{n-1}j_{n-2}\cdots j_s\cdots j_0)_2$, e retorna j_s, i.e., o $(s+1)$-ésimo bit de j, contando a partir dos bits menos significativos. Nota-se que esta função vem diretamente da Equação (4.23).

```
> bitval:=(s,j)->(1-(-1)^floor(j/2^s))/2:
```

O comando abaixo define uma função que retorna fração binária $0.j$, conforme a Equação (4.22).

```
> binfrac:=(j,n)->add(bitval(K,j)/(2^(K+1)),K=0..n-1):
```

A seguir, apresenta-se um código para cálculo do produto tensorial, ou de Kronecker. Este código é disponibilizado gratuitamente para *download* no site de R.B. Israel (`http://www.math.ubc.ca/~israel/advisor`).

```
> kronprod := proc (A, B)
    options  'Maple Advisor Database 1.00 for Maple V Release 5',
  'Copyright (c) 1998 by Robert B. Israel.  All rights reserved';
    local Ap, Bp, i,j;
    if nargs > 2
       then RETURN(kronprod(kronprod(A,B),args[3..nargs])) fi;
    if type(A,\{vector,list(algebraic)\}) and
       type(B,\{vector,list(algebraic)\}) then
  # vector x vector = vector
      vector([seq(seq(A[i]*B[j], j=1..linalg[vectdim](B)),
                  i=1..linalg[vectdim](A))])
    else # otherwise result is matrix
      if type(A,matrix)
         then Ap:= A
```

```
      elif type(A,listlist)
           then Ap:= convert(A,matrix)
      elif type(A,list)
           then Ap:= matrix(map(t->[t],A))
      elif type(A,specfunc(list,transpose))
           then Ap:= matrix([op(A)])
      else Ap:= convert(A,matrix)
      fi;
      if type(B,matrix)
           then Bp:= B
      elif type(B,listlist)
           then Bp:= convert(B,matrix)
      elif type(B,list)
           then Bp:= matrix(map(t->[t],B))
      elif type(B,specfunc(list,transpose))
           then Bp:= matrix([op(B)])
      else Bp:= convert(B,matrix)
      fi;
      linalg[stackmatrix](seq(linalg[augment](
         seq(linalg[scalarmul](Bp,Ap[i,j]), j = 1 ..
         linalg[coldim](Ap))),i = 1 .. linalg[rowdim](Ap)));
    fi
  end;
```

B.3 Definição das matrizes genéricas

Primeiramente, define-se a matriz para um passo s arbitrário do algoritmo FFT, conforme Equação (4.25). O primeiro parâmetro de `P(s,n)` refere-se ao passo e o segundo parâmetro ao número de bits de cada elemento do vetor de entrada.

```
> P:=(s,n)->(2^(-1/2))*Matrix(2^n, 2^n,
    (j,k)->piecewise(
       j=k,w(add(bitval(s,j-1)*bitval(t,j-1)*2^(n-t+s-1),
             t=s..n-1),n),
       j=k+2^s,bitval(s,j-1),
```

```
        j=k-2^s,bitval(s,k-1)*w(add(bitval(t,j-1)*2^(n-t+s-1),
                                t=s+1..n-1),n),
        0) ):
```

Após definir a matriz P na *worksheet*, pode-se verificar que os resultados para $n = 3$ correspondem às matrizes obtidas no exemplo da Seção 4.1.

Em seguida, define-se primeira matriz obtida pela decomposição QR, conforme a Equação (4.34).

```
> M:=(s,n)->Matrix(2^n, 2^n,
    (j,k)->(2^(-1/2))*piecewise(
        j=k,(-1)^floor((j-1)/(2^s)),
        j=k+2^s,bitval(s,j-1),
        j=k-2^s, bitval(s,k-1) , 0) ):
```

O comando abaixo define a segunda matriz obtida pela decomposição QR, conforme a Equação (4.35) — ou, equivalentemente, conforme a Equação (4.50).

```
> N:=(s,n)->Matrix(2^n, 2^n,
    (j,k)->piecewise(
        j=k,w(add(bitval(s,j-1)*bitval(t,j-1)*2^(n-t+s-1),
              t=s+1..n-1),n),
        0) ):
```

Com as matrizes M e N definidas na *worksheet*, é possível verificar alguns casos particulares da Proposição 4.2.

Finalmente, o comando abaixo define a matriz $R^{(s,t,u)}$ conforme a Equação (4.51). Com esta matriz, é possível utilizar o Maple para testar alguns casos particulares da Proposição 4.4.

```
> R:=(s,t,n)->Matrix(2^n, 2^n,
   (j,k)->piecewise(
        j=k,w(bitval(s,j-1)*bitval(t,j-1)*2^(n-t+s-1),n),
        0) ):
```

APÊNDICE C – Descrição técnica do simulador Feynman

C.1 Comandos e tipos de dados do MPI

Os principais comandos MPI utilizados na implementação do simulador foram:

- MPI::Init(**int** ∗pargc, **char** ∗∗∗pargv). Inicia o ambiente de execução MPI.
  pargc - número de argumentos passados na linha de comando.
  pargv - ponteiro para o vetor de argumentos passados.
- MPI::Finalize(). Termina o ambiente de execução MPI
- MPI::COMM_WORLD.Get_rank(). Retorna um inteiro correspondendo à identificação do processo.
- MPI::COMM_WORLD.Recv(**void** ∗buf, **int** count, **const** Datatype& datatype, **int** src , **int** tag). Recebe uma mensagem de determinado processo.
  buf - ponteiro para a variável que se deseja receber
  count - número de elementos a receber
  datatype - Tipo de dado do elemento a ser recebido
  src - número do processo que enviou a mensagem
  tag - identificador da mensagem
- MPI::COMM_WORLD.Send(**void** ∗buf, **int** count, **const** Datatype& datatype, **int** dest , **int** tag). Envia uma mensagem para determinado processo.
  buf - ponteiro para a variável que se deseja enviar
  count - número de elementos a enviar
  datatype - Tipo de dado do elemento a ser enviado

dest - número do processo de destino
tag - identificador da mensagem

Os principais tipos de dados do MPI, e seus tipos respectivos em C/C++, são:

- MPI::INT, equivalente ao tipo **signed int**.
- MPI::FLOAT, equivalente ao tipo **float**.
- MPI::DOUBLE, equivalente ao tipo **double**.
- MPI::COMPLEX, equivalente ao tipo Complex<**float**>.
- MPI::DOUBLE_COMPLEX, equivalente ao tipo Complex<**double**>.
- MPI::CHAR, equivalente ao tipo **char**.
- MPI::UNSIGNED, equivalente ao tipo **unsigned int**.
- MPI::UNSIGNED_LONG, equivalente ao tipo **unsigned long int**.

C.2 Programa principal

A seguir são dados trechos do programa principal do simulador Feynman. Por simplicidade, foram retiradas todas as linhas não essenciais ao entendimento. Um descrição do algoritmo encontra-se na Seção 5.3.2. Os comandos do MPI foram resumidos na Seção anterior, e maiores informações podem ser encontradas em manuais especializados (SNIR *et al.*, 1996) e no site `http://www-unix.mcs.anl.gov/mpi`. Também foram utilizados alguns comandos da *GNU Scientific Library*. Apesar de serem comandos bastante intuitivos, pode-se consultar manuais disponíveis na Internet (`http://www.gnu.org/software/gsl`) em caso de dúvidas.

Primeiramente, observa-se a parte do código que é executada pelo nodo mestre. Antes do trecho exposto, existe uma etapa de inicialização, onde algumas informações são lidas do arquivo de entrada, e são realizadas algumas verificações.

```
int i;
AQFTI c(nmQubits,parametro);
c.prepara();
```

```
while(c.portasRestantes())
    c.proxPorta();

for(i=1; i<=((float)c.getnQubits()/2.); i++)
    c.solicitaSwap(i,c.getnQubits()-i+1);

for(i=1; i<=c.getnQubits(); i++)
    c.finalizaQubit(i);

c.salva(argv[3]);
```

Agora, observa-se parte do código executado pelos nodos escravos. O objeto qbt é da classe Qubit, e foi previamente inicializado com o estado lido a partir do arquivo de entrada.

```
int msg=0, origem=0;
MPI::Status status;
while(msg!=MSG_FINALIZA){
    MPI::COMM_WORLD.Recv(&msg,1,MPI::INT,MPI_ANY_SOURCE,
        TAG_NOVA_INSTRUCAO, status);
    origem=status.Get_source();
    switch(msg){
        case MSG_PORTA:  //Uma porta foi recebida
            Porta p;
            MPI::COMM_WORLD.Recv(&msg,1,MPI::INT,origem,
                TAG_PROX_PORTA_CTR);
            p.setControle(msg);
            MPI::COMM_WORLD.Recv(&msg,1,MPI::INT,origem,
                TAG_PROX_PORTA_ALV);
            p.setAlvo(msg);
            MPI::COMM_WORLD.Recv(&msg,1,MPI::INT,origem,
                TAG_PROX_PORTA_ORG);
            p.setOrigem(msg);

            gsl_matrix_complex *mtr;
            mtr=gsl_matrix_complex_alloc(2,2);

            gsl_complex z;
            MPI::COMM_WORLD.Recv(&z,sizeof(z),MPI::BYTE,origem,
                TAG_PROX_PORTA_MTR00);
            gsl_matrix_complex_set(mtr,0,0,z);
            MPI::COMM_WORLD.Recv(&z,sizeof(z),MPI::BYTE,origem,
```

```
                TAG_PROX_PORTA_MTR01);
            gsl_matrix_complex_set(mtr,0,1,z);
            MPI::COMM_WORLD.Recv(&z,sizeof(z),MPI::BYTE,origem,
                TAG_PROX_PORTA_MTR10);
            gsl_matrix_complex_set(mtr,1,0,z);
            MPI::COMM_WORLD.Recv(&z,sizeof(z),MPI::BYTE,origem,
                TAG_PROX_PORTA_MTR11);
            gsl_matrix_complex_set(mtr,1,1,z);

            p.setMatriz(mtr);
            if(mtr!=NULL){
                gsl_matrix_complex_free(mtr);
                mtr=NULL;
            }

            if(qbt.emaranhado() && (p.getOrigem()==MESTRE))
                qbt.reenviaPorta(p);

            qbt.efetua(p);

            if(qbt.emaranhado() && (p.getOrigem()==MESTRE))
                qbt.aguardaTodosOK();
            qbt.enviaOK(origem);
        break;

        case MSG_MANIPULA_ESTADOS_SWAP:
            int tam;
            if(!qbt.emaranhado()){
                //Agora eu recebo o estado do outro qubit
                MPI::COMM_WORLD.Recv(&tam,1,MPI::INT,origem,
                    TAG_ESTADO_TAM);
                gsl_matrix_complex *vet;
                vet= gsl_matrix_complex_alloc(tam,1);
                gsl_complex z;
                MPI::COMM_WORLD.Recv(&z,sizeof(gsl_complex),MPI::BYTE,
                    origem,TAG_ESTADO_VET0);
                gsl_matrix_complex_set(vet,0,0,z);
                MPI::COMM_WORLD.Recv(&z,sizeof(gsl_complex),MPI::BYTE,
                    origem,TAG_ESTADO_VET1);
                gsl_matrix_complex_set(vet,1,0,z);
                //Envio meu estado para o outro qubit
```

```
                qbt.enviaEstado(origem);
70              //Atribuo o estado recebido
                Estado novoEst(qbt.qtdEmar());
                novoEst.setVetor(vet);
                qbt.setEstado(novoEst);
                if(vet!=NULL){
75                  gsl_matrix_complex_free(vet);
                    vet=NULL;
                }
                //Envio pronto para o outro qubit
                int msg=MSG_NORMAL;
80              MPI::COMM_WORLD.Send(&msg,1,MPI::INT,origem,
                    TAG_PRONTO);
            }
        break;

85      case MSG_SWAP:
            int qb1, qb2;
            MPI::COMM_WORLD.Recv(&qb1,1,MPI::INT,origem,
                TAG_SOLICITA_SWAP0);
            MPI::COMM_WORLD.Recv(&qb2,1,MPI::INT,origem,
90              TAG_SOLICITA_SWAP1);
            if(qbt.emaranhado() && origem==MESTRE)
                qbt.reenviaSwap(qb1,qb2);
            qbt.efetuaSwap(qb1,qb2);
            if(qbt.emaranhado() && origem==MESTRE)
95          qbt.aguardaTodosOK();
            msg=MSG_NORMAL;
            MPI::COMM_WORLD.Send(&msg,1,MPI::INT,origem,
                TAG_PRONTO);
        break;
100
        case MSG_SOLICITA_ESTADO:
            Estado est(qbt.qtdEmar());
            est=qbt.getEstado();
            int tam=(int)pow(2.,(double)est.qtdQubits());
105         MPI::COMM_WORLD.Send(&tam,1,MPI::INT,origem,
                TAG_ESTADO_TAM);
            gsl_matrix_complex *vet;
            vet= gsl_matrix_complex_alloc(tam,1);
            vet= est.getVetor();
```

```
110                 gsl_complex z;
                    z=gsl_matrix_complex_get(vet,0,0);
                    MPI::COMM_WORLD.Send(&z, sizeof(gsl_complex),
                        MPI::BYTE, origem, TAG_ESTADO_VET0);
                    z=gsl_matrix_complex_get(vet,1,0);
115                 MPI::COMM_WORLD.Send(&z, sizeof(gsl_complex),
                        MPI::BYTE, origem, TAG_ESTADO_VET1);

                    if(vet!=NULL){
                        gsl_matrix_complex_free(vet);
120                     vet=NULL;
                    }
                break;

                case MSG_FINALIZA:
125                 qbt.salva(argv[3]);
                    MPI::COMM_WORLD.Send(&msg,1,MPI::INT,MESTRE,TAG_PRONTO);
                break;
            }//Fim-swicht
        }//Fim-enquanto
```

C.3 Código-fonte do método efetua

```
int Qubit::efetua(Porta porta){
    Estado estContr(1);
    int msg,tam;

 5  //Verifico se devo realmente executar a porta neste qubit
    if(getId()!=porta.getAlvo() && !emaranhado())
        return 1;

    if(emaranhado()){
10      setEstado(calculaLinearmente(porta,estado));
        return 0;
    }
    else{
        if(porta.getControle()==-1){
15      //Se nao for porta controlada
            setEstado(estado.produto(porta));
            return 0;
```

```
        }
        else{
20      //Se for porta controlada
            //Solicita estado via MPI
            msg=MSG_SOLICITA_ESTADO;
            MPI::COMM_WORLD.Send(&msg,1,MPI::INT,porta.getControle(),
                TAG_NOVA_INSTRUCAO);
25          MPI::COMM_WORLD.Recv(&tam,1,MPI::INT,porta.getControle(),
                TAG_ESTADO_TAM);
            gsl_matrix_complex *vetAux;
            vetAux=gsl_matrix_complex_alloc(tam,1);

30          gsl_complex z;
            MPI::COMM_WORLD.Recv(&z,sizeof(gsl_complex),MPI::BYTE,
                porta.getControle(),TAG_ESTADO_VET0);
            gsl_matrix_complex_set(vetAux,0,0,z);
            MPI::COMM_WORLD.Recv(&z,sizeof(gsl_complex),MPI::BYTE,
35              porta.getControle(),TAG_ESTADO_VET1);
            gsl_matrix_complex_set(vetAux,1,0,z);

            estContr.setVetor(vetAux);
            if(vetAux!=NULL){
40              gsl_matrix_complex_free(vetAux);
                vetAux=NULL;
            }

            if(estContr.um()){
45              setEstado(estado.produto(porta));
                return 0;
            }
            else
                return 0;
50      }
    }
    return 1;
}
```

C.4 Código-fonte do método calulaLinearmente

```
Estado Qubit::calculaLinearmente(Porta p, Estado est){
    int coef, pos;
```

```
    Estado estTemp(est.qtdQubits()), estBase(1), estAux(1);

    //Para cada coeficiente do estado
    for(coef=0; coef<pow(2.,(double)est.qtdQubits()); coef++){
        //Se o coef=0, nao precisa calcular essa iteracao
        if(gsl_complex_abs(est.getCoef(coef))>ERRO_COEF){
            //Para cada qubit do vetor de base
            for(pos=0; pos<est.qtdQubits(); pos++){
                if(ibin(coef,est.qtdQubits()-pos) == 0){
                    estBase.setCoef(0,gsl_complex_rect(1,0));
                    estBase.setCoef(1,gsl_complex_rect(0,0));
                }
                else{
                    estBase.setCoef(0,gsl_complex_rect(0,0));
                    estBase.setCoef(1,gsl_complex_rect(1,0));
                }
                if((pos+1)==p.getAlvo()){
                    if(ibin(coef,est.qtdQubits()-p.getControle()+1)==1
                            || p.getControle()==-1)
                        estBase = estBase.produto(p);
                }
                if(pos==0) estAux = estBase;
                else estAux = estAux.tensorial(estBase);
            }
            estAux = estAux.mulEscalar(est.getCoef(coef));
            estTemp = estTemp.soma(estAux);
        }
    }
    return estTemp;
}
```

C.5 Formato dos arquivos de entrada e saída

O arquivo de entrada para o simulador Feynman pode ser gerado em qualquer editor de textos em formato ASCII. O arquivo de saída respeita o mesmo padrão do arquivo de entrada, portanto o resultado de uma simulação pode ser usado como valor inicial de uma nova simulação.

Neste formato, pode-se destinar as primeiras linhas a comentários como data, autor, tipo de simulação, dentre outros. A primeira linha válida é aquela que começa

com `nqubits:` seguida do número de *qubits* da simulação. Não pode haver espaços.

Depois disso, cada *qubit* deve ter seu estado descrito. Indica-se que o estado de um determinado *qubit* começa a ser descrito através da linha `qubit:`, seguido do número de identificação dos mesmos[1]. Caso o estado esteja emaranhado, o procedimento é similar. Neste caso, todos os *qubits* emaranhados são listados após a palavra `qubit:` separados apenas pelo caracter dois-pontos (`:`). Em ambos os casos, a linha iniciada por `qubit:` deve ser finalizada por dois-pontos.

Para descrever o estado basta listar seus coeficientes complexos, um em cada linha, começando imediatamente abaixo da linha com a palavra `qubit:`. Até a versão 0.3 do simulador, os todos os coeficientes deviam ser listados, separando a parte real da imaginária através do caracter dois-pontos. A partir da versão 0.4, pode-se omitir os coeficientes iguais a zero. Entretanto, a partir dessa versão, deve-se informar em cada linha do arquivo o índice do coeficiente no vetor de estado. O formato, então, fica `<índice>:<parte-real>:<parte-imaginária>`. Finaliza-se a descrição indicando um índice negativo no vetor de estado.

Na Figura 24 encontra-se um exemplo de arquivo que utiliza o padrão descrito. Trata-se de uma simulação de três *qubits*, onde os *qubits* 2 e 3 estão emaranhados. Neste exemplo, o *qubit* 1 vale $|0\rangle$, e os *qubits* 2 e 3 valem $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

```
autor: Franklin de Lima Marquezino
comentario: Apenas um exemplo.
nqubits:3

qubit:1:
0:1:0:
-1:

qubit:2:3:
0:0.70710678118654752440084436210485:0
3:0.70710678118654752440084436210485:0
-1:
```

Figura 24: Exemplo de arquivo de entrada para o simulador

C.6 Diagrama de classes

[1]Note que o número de identificação deve começar com um.

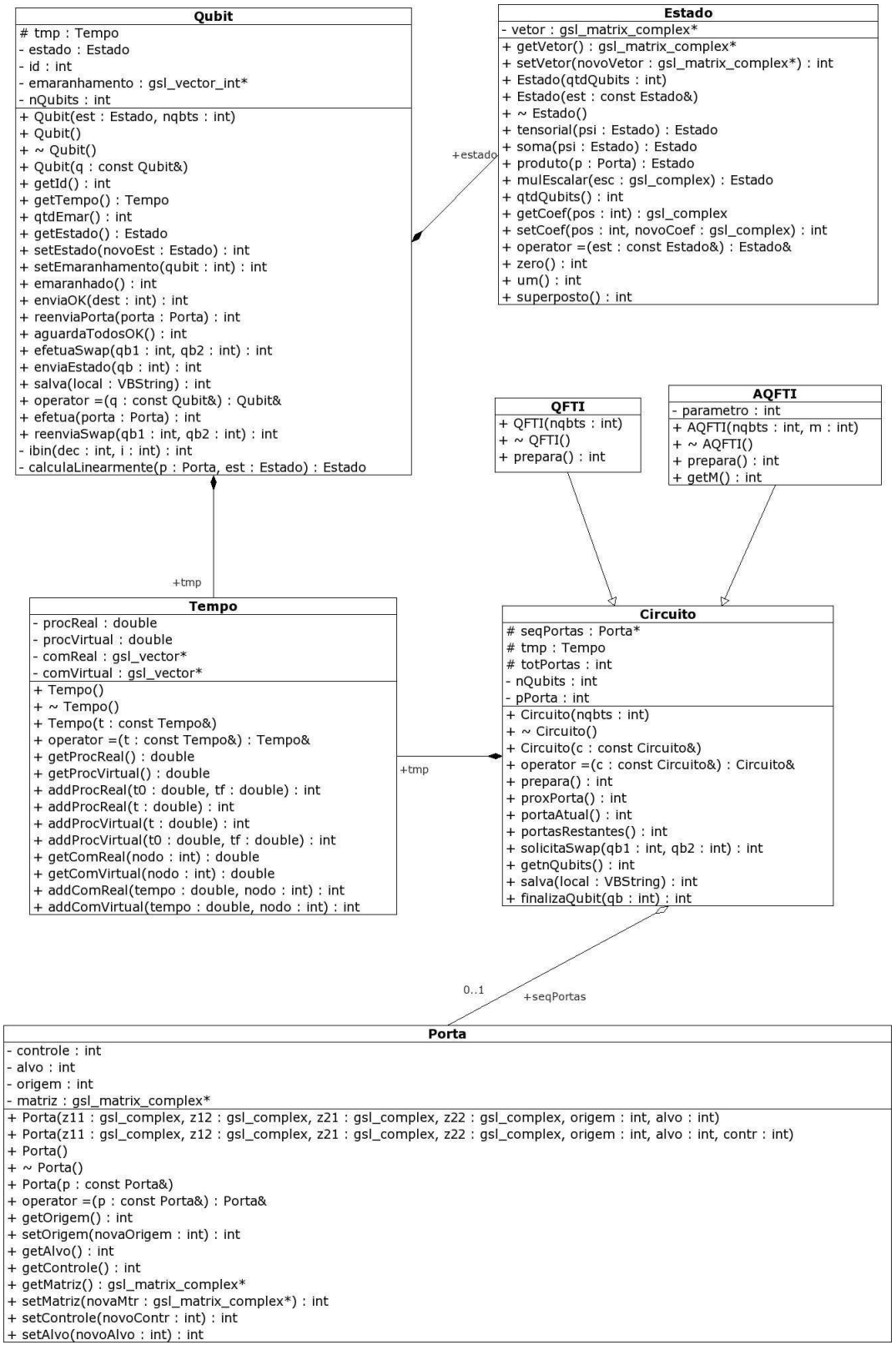


Figura 25: Diagrama de classes do simulador Feynman 0.4

Referências

ALTENKIRCH, T.; GRATTAGE, J. A functional quantum programming language. In: *Proceedings of the 20th Annual IEEE Symposium on Logic and Computer Science*. IEEE Computer Society, 2005. Disponível em: <http://www.arxiv.org/quant-ph/0409065>.

BACON, J.; HARRIS, T. *Operating Systems: concurrent and distributed software design*. [S.l.]: Addison Wesley, 2003.

BAKER, G. *Qgol: a system for simulating quantum computation: theory, implementation and insight*. Tese — Macquarie University, 1996. Disponível em: <http://www.ifost.org.au/ gregb/q-gol/QgolThesis.pdf>.

BARENCO, A. *et al.* Elementary gates for quantum computation. *Phys. Rev. A*, v. 52, p. 3457, 1995.

BARENCO, A. *et al.* Approximate quantum Fourier transform and decoherence. *Phys. Rev. A*, v. 54, p. 139–146, 1996.

BENIOFF, P. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *J. Stat. Phys.*, v. 22, p. 563–591, 1980.

BENNETT, C. Logical reversibility of computation. *IBM J. Res. Dev.*, v. 17, p. 5225, 1973.

BENNETT, C. H.; BRASSARD, G. Quantum cryptography: Public key distribution and coin tossing. In: *Proceedings of IEEE international Conference on Computers, Systems and Signal Processing, Bangalore, India*. New York: IEEE Press, 1984. p. 175.

BENNETT, C. H. *et al.* Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Phys. Rev. Lett.*, v. 70, p. 1895–1899, 1993.

BERNSTEIN, E.; VAZIRANI, U. Quantum complexity theory. *SIAM J. Comp.*, v. 26, p. 1411–1478, 1997.

BOSCHI, D. *et al.* Experimental realization of teleporting an unknown pure quantum state via dual classical and Einstein-Podolski-Rosen channels. *Phys. Rev. Lett.*, v. 80, p. 1121–1125, 1998.

BOUWMEESTER, D. *et al.* Experimental quantum teleportation. *Nature*, v. 390, p. 575–579, 1997.

BREMNER, M. J. *et al.* A practical scheme for quantum computation with any two-qubit entangling gate. *Phys. Rev. Lett.*, n. 89, p. 247902, 2002. Disponível em: <http://www.arxiv.org/quant-ph/0207072>.

BULNES, J. J. D. *Emaranhamento e separabilidade de estados em Computação Quântica por Ressonância Magnética Nuclear*. Tese (Doutorado) — Centro Brasileiro de Pesquisas Físicas, Rio de Janeiro, Setembro 2005.

BUTSCHER, B.; WEIMER, H. Simulation eines Quantencomputers. Universität Stuttgart. März 2003. Disponível em: <http://www.enyo.de/libquantum>. Acesso em: 13 jan. 2006.

CHAVES, A. *Física: Ondas, Relatividade e Física Quântica*. Rio de Janeiro: Reichmann & Affonso Editores, 2001.

CHEUNG, D. Improved bounds for the approximate QFT. March 2004. Disponível em: <http://www.arxiv.org/quant-ph/0403071>.

CHEUNG, K.; MOSCA, M. Decomposing finite abelian groups. *J. Quantum Inf. Comp.*, v. 1, n. 3, p. 26–32, 2001.

CHURCH, A. An unsolvable problem of elementary number theory. *Annals of Mathematics, second series*, v. 33, p. 346–366, 1936.

CLEVE, R. A note on computing Fourier transforms by quantum programs. 1994. Disponível em: <http://pages.cpsc.ucalgary.ca/~cleve>.

COHEN-TANNOUDJI, C.; DIU, B.; LALOË, F. *Quantum mechanics*. New York: Wiley, 1977.

COOLEY, J.; LEWIS, P.; WELCH, P. Historical notes on the Fast Fourier Transform. *IEEE Transactions on Audio and Electroacoustics*, AU-15, n. 2, p. 76–79, 1967.

COOLEY, J.; TUKEY, J. An algorithm for the machine calculation of complex Fourier series. *Math. of Comp.*, v. 19, p. 297–301, April 1965.

COPPERSMITH, D. *An Approximate Fourier Transform Useful in Quantum Factoring*. New York, July 1994. IBM Research Report 19642.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. *Introduction to Algorithms*. Cambridge, Massachussets: MIT Press, 1990.

DANIELSON, G.; LANCZOS, C. Some improvements in practical Fourier analysis and their application to X-ray scattering from liquids. *J. Franklin Inst.*, v. 233, p. 365–380 and 435–452, 1942.

DAVYDOV, A. S. *Quantum Mechanics*. 2nd. ed. Oxford: Pergamon, 1976. (International Series in Natural Philosophy, 1). Translated from the Russian by D. ter Haar.

DEITEL, H.; DEITEL, P. *C++ Como Programar*. 3. ed. Porto Alegre: Bookman, 2001.

DEUTSCH, D. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. R. Soc. Lond. A*, v. 400, p. 97–117, 1985.

DEUTSCH, D. Quantum computational networks. *Proc. R. Soc. of Lond. A*, v. 425, p. 73, 1989.

DRAPER, T. G. *Addition on a quantum computer*. August 2000. Disponível em: <http://www.arxiv.org/quant-ph/0008033>.

DRAPER, T. G. *et al.* A logarithmic-depth quantum carry-lookahead adder. June 2004. Disponível em: <http://www.arxiv.org/quant-ph/0406142>.

DUNLAVEY, M. R. Simulation of the finite state machines in a quantum computer. July 1998. Disponível em: <http://www.arxiv.org/quant-ph/9807026>.

EDELMAN, A.; MCCORQUODALE, P.; TOLEDO, S. The Future Fast Fourier Transform? *SIAM Journal on Scientific Computing*, v. 20, n. 3, p. 1094–1114, May 1999.

EINSTEIN, A.; PODOLSKY, B.; ROSEN, N. Can quantum-mechanical description of reality be considered complete? *Phys. Rev.*, v. 47, n. 10, p. 777–780, May 1935.

FEYNMAN, R. P. Simulating physics with computers. *Int. J. of Theor. Phys.*, v. 21, p. 467, 1982.

FREDKIN, E.; TOFFOLI, T. Conservative logic. *Int. J. Theor. Phys.*, v. 21, p. 219–253, 1982.

FURUSAWA, A. *et al.* Unconditional quantum teleportation. *Science*, v. 282, n. 5389, p. 706–709, 1998.

GASIOROWICZ, S. *Quantum Physics*. New York: John Wiley & Sons, Inc., 1974.

GAY, S. Quantum programming languages: survey and bibliography. *Bulletin of the European Association for Theoretical Computer Science*, v. 86, p. 176–196, June 2005.

GRAMA, A. *et al. Introduction to Parallel Algorithm Design*. 2. ed. [S.l.]: Addison Wesley, 2003.

GROVER, L. A fast quantum mechanical algorithm for database search. In: *Proc. 28th Annual ACM Symposium on the Theory of Computation*. New York, NY: ACM Press, New York, 1996. p. 212–219.

GUO, H.; BURRUS, C. Approximate FFT via the discrete wavelet transform. In: *Proceedings of SPIE Conference*. Denver: [s.n.], 1996.

HALLGREN, S.; RUSSELL, A.; TA-SHMA, A. Normal subgroup reconstruction and quantum computation using group representations. In: *Proceedings of the 32nd Symposium on Theory of Computing*. Portland, Oregon: [s.n.], 2000. p. 627–635.

HERTEL, J. Quantum Turing Machine Simulador. *The Mathematica Journal*, v. 8, n. 3, p. 440–457, 2002.

HOFFMAN, K.; KUNZE, R. *Linear Algebra*. Upper Saddle River, New Jersey: Prentice Hall, 1971.

HSU, E. Quantum Computing Simulation Optimizations and Operational Errors on Various 2-qubit Multiplier Circuits. August 2002. Disponível em: <http://www.arxiv.org/quant-ph/0208113>.

IVANYOS, G.; MAGNIEZ, F.; SANTHA, M. Efficient quantum algorithms for some instances of the non-Abelian hidden subgroup problem. *International Journal of Foundations of Computer Science*, v. 14, n. 5, p. 723–739, 2003.

KITAEV, A. *Quantum measurements and the abelian stabilizer problem*. 1995.

KNILL, E. *Conventions for Quantum Pseudocode*. [S.l.], 1996. Technical Report LAUR-96-2724.

KNUTH, D. E. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. 2nd. ed. Reading, Massachussets: Addison-Wesley, 1981.

LANDAUER, R. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, v. 5, p. 183–191, 1961.

LANG, S. *Linear Algebra*. 3rd. ed. New York: Springer-Verlag, 1987.

LEIBFRIED, D. *et al.* Creation of a six atom 'Schrödinger cat' state. *Nature*, v. 438, p. 639–642, December 2005.

LENSTRA, A. K.; LENSTRA JR., H. W. *The development of the Number Field Sieve*. New York: Springer-Verlag, 1993.

LLOYD, S.; BRAUNSTEIN, S. M. Quantum Computation over Continuous Variables. *Phys. Rev. Lett.*, v. 82, n. 8, p. 1784–1787, February 1999. Received 27 October 1998.

LOAN, C. V. *Computational frameworks for the Fast Fourier Transform*. Philadelphia: Society for Industrial and Applied Mathematics, 1992. (Frontiers in applied mathematics, 10).

LOMONT, C. The hidden subgroup problem: review and open problems. 2004. Disponível em: <http://www.arxiv.org/quant-ph/0411037>.

MANIN, Y. *Computable and Uncomputable*. Moscow: Sovetskoye Radio, 1980.

MARQUEZINO, F. L.; HELAYËL-NETO, J. A. Estudo Introdutório do Protocolo Quântico BB84 para Troca Segura de Chaves. *Revista Eletrônica de Iniciação Científica, Sociedade Brasileira de Computação*, 2004. Orientador: J.A. Helayël-Neto, Pré-print no Centro Brasileiro de Pesquisas Físicas, Série Monografias, CBPF-MO-001/03.

MARQUEZINO, F. L.; MELLO JUNIOR, R. R. An Introduction to Logical Operations on Classical and Quantum Bits. April 2004. Disponível em: <http://www.arxiv.org/physics/0404134>.

MARQUEZINO, F. L.; MELLO JUNIOR, R. R. Considerations on Classical and Quantum Bits. April 2004. Disponível em: <http://www.arxiv.org/physics/0404133>.

MARQUEZINO, F. L.; MELLO JUNIOR, R. R. *Simulação da Transformada de Fourier Quântica utilizando Computação Distribuída*. 63 p. Monografia (Graduação) — Faculdade de Informática, Universidade Católica de Petrópolis, Petrópolis, 2004.

MARQUEZINO, F. L. *et al.* Simulating the Quantum Fourier Transform with Distributed Computing. *III Workshop on Computational Grids and Applications*, Petrópolis, January 2005.

MARQUEZINO, F. L.; PORTUGAL, R.; SASSE, F. D. Quantum Fourier Transform Revisited. *Submetido para International Journal of Quantum Information*, 2006. Notas do LNCC 1/2006.

MCCUBBIN, C. B. *OpenQUACS, An Open-Source Quantum Computation Simulator in Maple*. Dissertação (Mestrado) — University of Maryland, 2000.

MEYER, C. *Matrix Analysis and Applied Linear Algebra*. Philadelphia: Society for Industrial and Applied Mathematics, 2000.

MOORE, G. Cramming more components onto integrated circuits. *Electronics*, v. 38, n. 8, April 1965. Disponível em: <ftp://download.intel.com/research/silicon/moorespaper.pdf>. Acesso em: 6 jan. 2003.

MOSCA, M. *Quantum Computer Algorithms*. Tese (Doutorado) — University of Oxford, 1999.

MU, S.-C.; BIRD, R. Functional quantum programming. In: *Proceedings of the 2nd Asian Workshop on Programming Languages and Systems*. Korea: [s.n.], 2001.

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

NIELSEN, M. A.; KNILL, E.; LAFLAMME, R. Complete quantum teleportation. *Nature*, v. 396, p. 52–55, 1998.

NUYENS, D. *Simulatietechnieken voor kwantumcomputers*. Dissertação (Mestrado), Mei 2002. Disponível em: <http://www.cs.kuleuven.ac.be/~dirkn/thesis>. Acesso em: 16 jan. 2006.

ÖMER, B. Simulation of quantum computers. 1996. Disponível em: <http://tph.tuwien.ac.at/~oemer/papers.html>.

ÖMER, B. *A procedural formalism for Quantum Computing*. Dissertação (Mestrado) — Technical University of Vienna, 1998.

ÖMER, B. *Quantum programming in QCL*. Dissertação (Mestrado) — Technical University of Vienna, 2000.

ÖMER, B. Procedural quantum programming. In: AMERICAN INSTITUTE OF PHYSICS. *Proceedings of the AIP Conference on Computing Anticipatory Systems*. [S.l.], 2001.

ÖMER, B. Classical concepts in quantum programming. 2002. Disponível em: <http://www.arxiv.org/quant-ph/0211100>.

ÖMER, B. *Structured Quantum Programming*. Tese (Doutorado) — Institute of Information Systems, Technical University of Viena, May 2003.

OPPENHEIM, A.; SCHAFER, R.; BUCK, J. *Discrete-time signal processing*. Upper Sadle River, NJ: Prentice Hall, 1999.

PARADISI, G.; RANDRIAM, H. A presentation of the quantum Fourier transform from a recursive viewpoint. 2004. Disponível em: <http://www.arvix.org/quant-ph/0411069>.

PORTUGAL, R. *et al. Uma introdução à Computação Quântica*. São Carlos, SP: Sociedade Brasileira de Matemática Aplicada e Computacional, 2004. (Notas em Matemática Aplicada, 8).

PRESKILL, J. *Lecture Notes on Physics 229: Quantum Information and Quantum Computation*. California Institute of Technology, USA, 1998. Disponível em: <http://www.theory.caltech.edu/~preskill/ph229>.

PRESS, W. *Numerical Recipes in C : the art of scientific computing*. 2nd. ed. Cambridge: Cambridge University Press, 1992.

RAEDT, H. D.; MICHIELSEN, K. Computational Methods for Simulating Quantum Computers. August 2004. Disponível em: <http://www.arxiv.org/quant-ph/0406210>.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method of obtaining digital signatures and public-key cryptosystems. v. 21, n. 2, p. 120–126, 1978.

ROSÉ, H. *et al.* The Fraunhofer Quantum Computing Portal: a web-based simulator of quantum computing processes. Junho 2004. Disponível em: <http://www.arxiv.org/quant-ph/0406089>.

RUDNICK, P. Note on the calculation of Fourier series. *Math. of Comp.*, v. 20, p. 429–430, July 1966.

RUNGE, C. *Zeit. für Math. und Physik*, v. 48, p. 443, 1903.

RUNGE, C. *Zeit. für Math. und Physik*, v. 53, p. 117, 1905.

SABRY, A. Modelling quantum computing in Haskell. In: *Proceedings of the ACM SIGPLAN Workshop on Haskell*. [S.l.]: ACM Press, 2003. p. 39–49.

SCARANI, V. *et al.* Quantum Cloning. *Reviews of Modern Physics*, v. 77, n. 4, October 2005.

SCHNEIDER, S. *Quantum System Simulator*. Dissertação (Mestrado) — Massachusetts Institute of Technology, July 2000.

SELINGER, P. Towards a quantum programming language. *Mathematical Structures in Computer Science*, v. 14, p. 527–586, 2004.

SHAPIRO, S. Church's Thesis. In: *Encyclopedia of Artificial Intelligence*. New York: John Willey & Sons, 1990. p. 99–100.

SHENTOV, O. *et al.* Subband DFT - Part I: definition, interpretation and extensions. *Signal Processing*, v. 41, n. 3, p. 261–277, February 1995.

SHOR, P. W. Algorithms for quantum computation: Discrete logarithms and factoring. In: GOLDWASSER, S. (Ed.). *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*. Los Alamitos, CA: IEEE Computer Society, 1994. p. 124–134.

SIMON, D. On the power of quantum computation. *SIAM J. Comp.*, v. 26, n. 5, p. 1474–1483, 1997.

SNIR, M. *et al. MPI: The Complete Reference*. Cambridge, MA: The MIT Press, 1996.

SOLOVAY, R.; STRASSEN, V. A fast Monte-Carlo test for primality. *SIAM J. of Comput.*, v. 6, p. 84–85, 1976.

STRANG, G. *Linear Algebra and its applications*. 3rd. ed. San Diego: Brooks Cole, 1988.

TANENBAUM, A. *Organização Estruturada de Computadores*. 4. ed. Rio de Janeiro: LTC, 2001.

TANENBAUM, A.; WOODHULL, A. *Sistemas Operacionais: projeto e implementação*. 2. ed. Porto Alegre: Bookman, 2000.

TONDER, A. van. Quantum Computation, Categorical Semantics and Linear Logic. 2003. Disponível em: <http://www.arxiv.org/quant-ph/0312174>.

TONDER, A. van. A Lambda Calculus for Quantum Computation. *SIAM J.Comput.*, n. 33, p. 1109–1135, 2004. Disponível em: <http://www.arxiv.org/quant-ph/0307150>.

TUCCI, R. R. A Rudimentary Quantum Compiler. February 1999. Disponível em: <http://www.arxiv.org/quant-ph/9902062>.

TURING, A. M. On computable numbers, with an application to the Entscheidungsproblem. *Proc. Lond. Math. Soc. 2*, n. 42, p. 230, 1936.

VANDERSYPEN, L. M. K. *et al.* Experimental realization of Shor's quantum factoring algorithm using Nuclear Magnetic Ressonance. *Nature*, v. 414, n. 6866, p. 883–887, 2001.

VIZZOTTO, J.; COSTA, A. Concurrent Quantum Programming in Haskell. VII Congresso Brasileiro de Redes Neurais. Seção de Computação Quântica. 2005.

WALLACE, J. Quantum computer simulators – a review. October 1999. Disponível em: <http://citeseer.ist.psu.edu/wallace99quantum.html>. Acesso em: 3 jan. 2006.

WATROUS, J. Quantum simulations of classical random walks and undirected graph connectivity. *Journal of Computer and System Sciences*, v. 62, n. 2, p. 376–391, 2001.

WOOTTERS, W. K.; ZUREK, W. H. A single quantum cannot be cloned. *Nature*, v. 299, n. 5886, p. 802–803, 1982.

YAO, A. Quantum circuit complexity. In: *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*. Los Alamitos, California: IEEE Press, 1993. p. 352–360.

ZAYER, J. *Faktorisieren mit dem Number Field Sieve*. Tese (Doutorado) — Technischen Fakultät, Universität des Saarlandes, Saarbrücken, Juni 1995.

Índice Remissivo