# Desenvolvimento de um simulador de algoritmos quânticos utilizando a computação convencional

ANDERSON DE REZENDE ROCHA
ANTÔNIO MARIA PEREIRA DE RESENDE[1]
ANTONIO TAVARES DA COSTA JÚNIOR[2]

[1]ORIENTADOR
DCC - Departamento de Ciência da Computação
{undersun,tonio}@comp.ufla.br

[2]CO-ORIENTADOR
DEX - Departamento de Ciências Exatas
antc@stout.ufla.br

13 de agosto de 2003

Todo trabalho, por mais simples que seja, não é feito sem a ajuda de outras pessoas. Ao longo do desenvolvimento desta pesquisa tive muita ajuda. Desta forma, gostaria de agradecer ao meu orientador *Antônio Maria* pelas dicas, ao meu co-orientador *Antônio Tavares* (Toninho) pelo grande conhecimento passado sobre a teoria quântica, e aos meus novos colegas de pesquisa, *Matthew Hayward* e *Berhnard Ohpner*, que podem ser encontrados em `http://sds.cs.uiuc.edu/`, pelos toques na implementação dos algoritmos. No campo pessoal, agradeço a minha namorada *Aninha*, por ser a minha fonte de inspiração.

# Sumário

# Capítulo 1

# Introdução

Existem, atualmente, inúmeras atividades que, ao serem realizadas em computador, demandam enorme quantidade de processamento e tempo de execução. As tentativas de redução do tempo de execução destas atividades levam os pesquisadores a buscarem, cada vez mais, máquinas mais velozes. Entretanto chegará um momento em que os limites físicos impedirão esta busca por mais velocidade.

Segundo a lei de Moore [Int Media, 2002], os fabricantes de chips são capazes de duplicar, a cada 18 (dezoito) meses, o número de transistores contidos numa lâmina de silício do tamanho de uma unha. Por meio de feixes de raios ultravioleta, o silício cristalino é gravado com sulcos microscópicos. O diâmetro de um fio de cabelo é 500 (quinhentas) vezes maior que o diâmetro dos fios de um chip Pentium, cuja camada isolante tem apenas 25 (vinte e cinco) átomos de espessura [Kaku, 1998].

Contudo, segundo as leis da física, há um limite para a miniaturização. Os transistores ficarão tão diminutos que os componentes de silício terão tamanho quase molecular. Em distâncias tão minúsculas, os caprichos da mecânica quântica entram em ação, fazendo com que os elétrons pulem de um ponto a outro sem cruzar o espaço que os separa. Eles brotarão através de fios e isolantes de dimensões atômicas, como água vazando de uma mangueira, e provocarão curtos-circuitos fatais [Kaku, 1998].

Então o que ocorrerá quando os limites físicos forem atingidos? Muitas são as propostas de solução atualmente. Uma das mais fundamentadas é a Computação Quântica [Nielsen e Chuang, 2000] que promete revolucionar a computação nas próximas décadas.

Mesmo que a lei de Moore deixe, em algum momento, de ser aplicada a computação tradicional ainda sim a computação quântica continuará válida devido ao seu alto poder de processamento [Nielsen e Chuang, 2000]. Comprovando a real possibilidade de implementação de computadores quânticos, sabe-se que alguns processadores,

mesmo que, ainda muito simples, já estão sendo testados em laboratórios como os da IBM [Estadao, 2000].

A computação quântica é um modo fundamentalmente novo de processar informações através do uso de fenômenos da mecânica quântica (especialmente a interferência quântica [Nielsen e Chuang, 2000]).

A mecânica quântica é a teoria mais geral da física. É o arcabouço sobre o qual todas as outras teorias atuais, exceto a teoria geral da relatividade, são formuladas.

A promessa de revolução na computação por parte da computação quântica está no poder desta em aproveitar o chamado paralelismo quântico para tratar certos problemas comumente intratáveis na computação clássica visto que todos os computadores atuais obedecem às leis da mecânica clássica - as leis que regulam o mundo macroscópico.

Deste modo as pesquisas nesta área vêm despertando real interesse por parte de grandes empresas como Microsoft, AT&T, IBM entre outras. O próprio governo americano tem real interesse em financiar esta área [NYT, 2002].

Entretanto, de certo modo, fica sempre a questão da qual o futuro econômico e a prosperidade de muitos países talvez dependam: a tecnologia dos chips de silício manterá viva a lei de Moore [Int Media, 2002] depois de 2020? Há muitos anos, essa lei é o motor que impulsiona uma indústria trilionária. Graças a ela, pode-se receber, e depois jogar fora, um cartão musical de aniversário cuja capacidade de processamento é superior à todos os computadores das Forças Aliadas usados na Segunda Grande Guerra.

Procurando tornar este tema um pouco mais conhecido apresenta-se a seguir os principais conceitos da área, as possibilidades de implementação dos computadores quânticos e, finalmente, um simulador quântico com três algoritmos quânticos implementados. São eles: *Fatoração de Peter Shor, O Problema de Deutsch* e a *Transformada Quântica de Fourier*.

## 1.1 Motivação

Dado que exista um limite para a miniaturização, os transistores ficarão tão diminutos que os componentes de silício terão um comportamento completamente diferente. Em distâncias tão minúsculas, os caprichos da mecânica quântica entram em ação, fazendo com que os componentes eletrônicos convencionais deixem de funcionar adequadamente. Por outro lado, as leis quânticas que impedem uma maior redução de tamanho dos dispositivos convencionais, podem ser aproveitadas para projetar computadores que se comportariam de um modo drasticamente novo.

Devido ao chamado paralelismo quântico, a computação quântica promete uma revolução na maneira de lidar problemas comumente intratáveis na computação clássica. O funcionamento dos componentes dos computadores atuais é baseado nas pro-

priedades quânticas da matéria, contudo, os bits, unidades fundamentais de processamento, são clássicos, dado que podem estar apenas no estado 0 ou no estado 1. Em contraposição, os bits de um computador quântico, ou quBits, poderiam ser colocados em estados que são superposições coerentes do estado 0 e do estado 1.

Deste modo, para permitir um maior entendimento deste *admirável mundo novo* procurou-se, neste trabalho, levantar os principais termos e conceitos da área bem como construir um simulador didático para demonstrar o funcionamento de alguns algoritmos quânticos.

As simulações permitem visualizar propriedades interessantes da computação quântica, bem como o real poder dos algoritmos para ela desenvolvidos.

## 1.2 Mecânica quântica

A maioria dos fenômenos mecânico-quânticos são difíceis de entender desde que a maioria das experiências não são facilmente aplicáveis.

Tal como na matemática, a mecânica quântica é governada por um conjunto de axiomas. Tais axiomas podem, às vezes, levar a aparentes paradoxos: no efeito Comptom, por exemplo, é como se as conseqüências aparecessem antes das causas.

Segundo [Preskill, 2002], pode-se dizer, de forma geral, que a teoria quântica é o modelo matemático do mundo físico. De modo a caracterizar este modelo encontra-se, a seguir, definições de como representá-lo através dos *estados*, *observáveis*, *medições* e *dinâmica*.

### 1.2.1 Estados

Um estado é um descrição completa de um sistema físico. Na mecânica quântica, um estado é uma *linha* no *espaço de Hilbert*.

#### O *Espaço de Hilbert*

1. É um espaço vetorial sobre números complexos C. Os vetores são denotados por $|\psi\rangle$, conhecido como notação *ket* de Dirac.
2. Tem um produto interno $\langle\psi|\varphi\rangle$ que mapeia um par ordenado de vetores para C, definido pelas propriedades:
   (a) *Positividade:* $\langle\psi|\varphi\rangle > 0$ para $|\psi\rangle = 0$
   (b) *Linearidade:* $\langle\varphi|(a|\psi_1\rangle + b\,|\psi_2\rangle) = a\,\langle\varphi|\psi_1\rangle + b\,\langle\varphi|\psi_2\rangle$
   (c) *Simetria transversal:* $\langle\varphi|\psi\rangle = \langle\psi|\varphi\rangle^*$

3. É completo na *norma* $||\psi|| = \langle\psi|\psi\rangle^{\frac{1}{2}}$

### 1.2.2 Observáveis

Um observável é uma propriedade de um sistema físico que, em princípio, pode ser medido. Na mecânica quântica um observável é um *operador auto-adjunto*. Um operador toma um vetor e o mapeia para outro vetor:

$$\mathbf{A} : |\psi\rangle \rightarrow \mathbf{A}\,|\psi\rangle\,, \mathbf{A}(a\,|\psi\rangle + b\,|\psi\rangle) = a\mathbf{A}\,|\psi\rangle + b\mathbf{B}\,|\psi\rangle\,. \tag{1.1}$$

O adjunto do operator $\mathbf{A}$ é definido por:

$$\langle\varphi|\mathbf{A}\psi\rangle = \left\langle \mathbf{A}^{\dagger}\varphi|\psi\right\rangle, \tag{1.2}$$

para todos os vetores $|\varphi\rangle$, $|\psi\rangle$. $\mathbf{A}$ é auto-adjunto se $\mathbf{A} = \mathbf{A}^{\dagger}$.

### 1.2.3 Medições

Na mecânica quântica, o resultado numérico de uma medida do observável $\mathbf{A}$ é um autovalor de $\mathbf{A}$. Assim que a medição for feita, o estado quântico é um auto-estado de $\mathbf{A}$ com o autovalor medido. Se o estado quântico imediatamente anterior à medida é $|\psi\rangle$, então a saída $a_n$ é obtida através da probabilidade:

$$\mathbf{Prob}\,(a_n) = \|\mathbf{P_n}\,|\psi\rangle\,\|^2 = \langle\psi|\mathbf{P_n}|\psi\rangle\,. \tag{1.3}$$

Se a saída $a_n$ foi alcançada, então o estado quântico normalizado torna-se

$$\frac{\mathbf{P_n}\,|\psi\rangle}{(\langle\psi|\mathbf{Pn}|\psi\rangle)^{\frac{1}{2}}}. \tag{1.4}$$

Deve-se notar que se a mesma medida é imediatamente repetida, então, de acordo com esta regra, o mesmo resultado será obtido com probabilidade 1.

### 1.2.4 Dinâmica

A evolução no tempo de um estado quântico é unitária. Ela é gerada por um operator auto-adjunto, chamado de *Hamiltoniano* do sistema.

## 1.3 O quBit

A unidade indivisível da informação clássica é o *bit*, que pode tomar um de dois valores possíveis $\{0, 1\}$. A unidade correspondente na informação quântica é chamada de *quBit* ou *quantum bit*. Ele descreve o estado mais simples possível em um sistema quântico.

O menor espaço de Hilbert não trivial é bi-dimensional. Deste modo, pode-se denotar uma base ortonormal para espaço de um vetor bi-dimensional como $\{|0\rangle, |1\rangle\}$. Então, a maioria dos estados normalizados pode ser escrita como segue:

$$a|0\rangle + b|1\rangle, \tag{1.5}$$

onde $a$ e $b$ são números complexos que satisfazem $|a|^2 + |b|^2 = 1$.

Deste modo, pode-se fazer uma medida do quBit através de sua projeção na base $\{|0\rangle, |1\rangle\}$. Então pode-se obter a saída $|0\rangle$ com probabilidade $|a|^2$, e a saída $|1\rangle$ com probabilidade $|b|^2$. Além disso, exceto quando $a = b = 0$ a medida causa distúrbios no estado. Neste sentido os quBits diferem-se dos bits clássico que, por sua vez, podem ser medidos e manipulados sem sofrerem distúrbios.

# Capítulo 2

# Objetivos

De forma geral esta pesquisa procurou:

- propiciar ao autor um maior contato com as teorias da computação e mecânica quânticas, disciplinas não lecionadas no curso de graduação.
- dar continuidade à metodologia de desenvolvimento científico experimentado pelo autor quando de sua atual bolsa de pesquisa.
- estudar de técnicas atuais da computação quântica e seu embasamento na mecânica quântica.
- pesquisar algoritmos quânticos existentes.
- analisar o desempenho de algoritmos quânticos e seu real aproveitamento do processamento quântico.
- listar e classificar as perdas do desempenho computacional.
- buscar possíveis soluções para estas perdas na literatura quântica, matemática e física.
- desenvolver um simulador de algoritmos quânticos já propostos por pesquisadores da área. Estes algoritmos apresentam um novo paradigma de pensamento em relação à computação clássica. Deste modo pretende-se utilizar os meios clássicos de computação para simular estes algoritmos em funcionamento. Assim, quando uma pessoa comum procurar saber sobre algum algoritmo quântico não esbarre apenas em teorias e sim numa visualização de como atuaria aquele algoritmo. Esta visualização, mesmo que utilizando os meios clássicos de programação, com certeza, propiciará a esta pessoa um nível bem superior de entendimento do assunto.

- disponibilizar todo o conteúdo coletado em um site dedicado ao projeto de modo que todas as pessoas interessadas no assunto possam entender de forma mais simplificada em que nível está a computação quântica atualmente.
- formar uma equipe de pesquisa no departamento, quiçá na universidade, para atuar nesta área, visto que, ela engloba conhecimentos não só de computação, mas também de ciências exatas como a física.
- divulgar um pouco mais este tema, que, com certeza, não sairá das mídias informativas nos anos que estão por vir.

*Todos os conceitos julgados necessários ao entendimento deste trabalho são brevemente abordados ao longo do texto. Com certeza, materiais mais completos podem ser encontrados nas referências.*

# Capítulo 3

# Metodologia

Considerando que os conceitos básicos já foram introduzidos, doravante só serão explicados termos ainda não apresentados.

O autor realizou um pequeno curso de extensão para o conhecimento dos principais conceitos e leis da mecânica quântica, as bases as bases, a independência linear, operadores lineares e matrizes, produtos internos, produtos tensores, operações de funções, medida quântica, fases, etc.

Finda esta fase procurou-se levantar quais seriam os principais impactos da computação quântica no mundo. As mudanças que obrigatoriamente ocorreriam caso fosse anunciado de uma hora para outra a existência de um computador quântico

Após o levantamento deste impactos procurou-se estudar quais seriam as possíveis formas de implementação / realização de hardwares quânticos na atualidade. Com isso, chegou-se às 3 (três) principais formas que serão explicadas na primeira sessão dos resultados e discussão.

As três principais formas de realização física de hardwares quânticos incluem ressonância magnética nuclear, armadilha de íons e cavidades quânticas eletrodinâmicas.

Em seguida, escolheu-se três algoritmos quânticos: fatoração de Peter Shor, Transformada Quântica de Fourier e o Problema de Deutsch para implementação e um ambiente computacional. Escolheu-se a ferramenta de programação JAVA da Sun Microsystems devido à sua portabilidade.

*Na sessão de resultados e discussão serão explicados como foi feita cada implementação. Mais informações podem ser encontradas nos anexos.*

# Capítulo 4

# Resultados e discussão

Os resultados obtidos dividem-se em três partes. Na primeira parte apresentada na seção 4.1 descreve-se as principais tentativas de se desenvolver hardwares quânticos na atualidade. Na segunda parte, apresentada na seção 4.2, encontra-se toda a fundamentação teórica pesquisada e os formalismos para o desenvolvimento da última seção, 4.3, na qual encontra-se toda a descrição do SAQua (Simulador de Algoritmos Quânticos) desenvolvido neste projeto. Até o momento este simulador trabalha perfeitamente com três algoritmos: *Fatoração de Peter Shor*, *Transformação Quântica de Fourier* e o *Problema de Deutsch*.

## 4.1 Os computadores quânticos — realização física

Segundo [Preskill, 2002], vários estudos têm sido feitos no sentido de conseguir uma implementação plausível e útil de um computador quântico.

Para se conseguir construir o *hardware* para um computador quântico, é necessário uma tecnologia que possibilite a manipulação de quBits. O *hardware* precisará se adequar a alguns requisitos severos:

- **Armazenamento.** Os quBits precisam ser armazenados por períodos de tempo suficientes para completar computações interessantes.
- **Isolamento.** Os quBits precisam estar isolados do ambiente, para minimizar erros por *decoerência*.
- **Leitura.** Os quBits precisam permitir sua leitura de forma eficiente e confiável.
- **Portas lógicas.** É necessário a possibilidade de manipulação de quBits individuais. Deste modo, para permitir interações controladas entre quBits é necessário a construção de portas lógicas quânticas.

- **Precisão.** As portas lógicas quânticas precisam ser implementadas com alta precisão se o dispositivo for para cálculos confiáveis.

A seguir apresenta-se três abordagens para implementação de um computador quântico.

### 4.1.1 Armadilha de íons

Segundo [Nielsen e Chuang, 2000], um meio possível para atingir os requisitos citados foi proposto por Ignacio Cirac e Peter Zoller e tem sido continuado pelo grupo de pesquisas de Dave Wineland no *National Institute for Standards and Technology*, NIST, nos EUA. Neste esquema, cada quBit é carregado ou representado por um simples íon alimentado em uma *armadilha linear de Paul*.

O estado quântico de cada íon é uma combinação linear do estado base $|g\rangle$ (interpretado como $|0\rangle$) e um estado de longa vida particular metaestável e excitado $|e\rangle$ (interpretado como $|1\rangle$). Uma combinação linear coerente destes dois níveis,

$$a\,|g\rangle + be^{iwt}\,|e\rangle\,, \tag{4.1}$$

pode sobreviver por um tempo comparável a uma vida inteira de um estado excitado (apesar de que as fases relativas oscilam devido à divisão de energia $\hbar\omega$ entre os níveis). Os íons são tão bem isolados que decaídas espontâneas podem ser a forma dominante de decoerência.

É relativamente fácil *ler* os íons através de sua medida que faz a projeção sobre a base $\{|g\rangle\,, |e\rangle\}$. Um laser é direcionado para uma transição a partir do estado $|g\rangle$ para um estado excitado de vida curta $|e'\rangle$.

Quando o laser ilumina os íons, cada quBit com a valor $|0\rangle$ repetidamente absorve e reemite a luz do laser, assim ele fluoresce. Por outro lado, os quBits com o valor $|1\rangle$ permanecer escuros.

Devido à sua repulsão mútua de Coulomb, os íons são suficientemente bem separados de modo que eles podem ser individualmente endereçados por pulsos de laser. Se um laser é direcionado para a freqüência $\omega$ da transição e é focada no *n*-ésimo íon, então as oscilações de Rabi estão induzidas entre $|0\rangle$ e $|1\rangle$. Através da determinação correta do tempo, *timing*, do pulso do laser e pela escolha da fase apropriada para este, os pulsos de laser podem preparar qualquer combinação linear de $|0\rangle$ e $|1\rangle$.

No entanto, a parte mais complicada no projeto e construção de um *hardware* para computação quântica é tomar dois quBits e fazê-los interagir um com o outro. Na armadilha de íons, as interações aparecem devido às repulsões de Coulomb entre os íons. Devido à repulsão mútua de Coulomb, há um espectro de modos normais unidos de vibração para os íons pegos pela armadilha. Quando os íons absorvem ou emitem

um fóton laser, o centro de massa do íon recua. Todavia, se o laser é corretamente direcionado, então quando um simples íon absorve ou emite um fóton, um modo normal envolvendo muitos íons irá recuar coerentemente. Isto é conhecido como efeito Mössbauer.

O modo vibracional de freqüência mais baixa, freqüência $\nu$, é o modo de centro de massa (*cm*), em que os íons oscilam em modo fechado, *lockstep* na armadilha. Os íons podem ser resfriados pelo laser para uma temperatura muito menor que $\nu$, assim, cada modo vibracional irá, provavelmente, ocupar seu estado mecânico-quântico base. Agora imagine que o laser seja direcionado para a freqüência $(\omega - \nu)$ iluminando o *n*-ésimo íon. Para um tempo de pulso propriamente escohido o estado $|e\rangle_n$ irá rotacionar-se para $|g\rangle_n$, enquanto o oscilador de centro de massa fará uma transição do estado base $|0\rangle_{cm}$ para seu primeiro estado excitado $|1\rangle_{cm}$, assim um *phonon* é produzido. Entretanto, o estado $|g\rangle_n |0\rangle_{cm}$ não estará na ressonância para qualquer transição e, deste modo, não estará afetado pelo pulso. Assim, o pulso de laser induz uma transformação unitária atuando como:

$$\begin{aligned} |g\rangle_n |0\rangle_{cm} &\rightarrow |g\rangle_n |0\rangle_{cm} , \\ |e\rangle_n |0\rangle_{cm} &\rightarrow -i |g\rangle_n |1\rangle_{cm} . \end{aligned} \tag{4.2}$$

Esta operação remove um bit de informação que é inicialmente armazenado no estado interno do *n*-ésimo íon, e deposita este bit no estado *coletivo* em movimento de todos os íons.

Isto significa que o estado de movimento do *m*-ésimo íon $(m \neq n)$ foi influenciado pelo estado interno do *n*-ésimo íon. Neste sentido, houve sucesso na indução de interação entre os íons. Para completar a porta lógica quântica, precisa-se poder transferir a informação quântica do centro de massa de um *phonon* de volta para o estado interno de um dos íons. O processo poderia ser desenvolvido desde que o centro de massa do *modo* sempre retorna seu estado base $|0\rangle_{cm}$ na conclusão da implementação da porta lógica. Por exemplo, Cirac e Zoller mostraram que a porta lógica quântica *XOR*

$$|x, y\rangle \rightarrow |x, y \oplus x\rangle , \tag{4.3}$$

pode ser implementada em uma armadilha de íons com um total de 5 pulsos de laser.

**QUADRO-RESUMO**

1. *Representação do quBit.* Estados hiperfinos de um átomo, *spins nucleares*, e o menor nível vibracional, *phonons*, de átomos aprisionados nas armadilhas
2. *Evolução unitária.* Transformações arbitrárias são construídas a partir da aplicação de pulsos de laser que externamente manipulam o estado atômico, numa interação conhecida na literatura como *interação de Jaynes-Cummings*. A interação entre os quBits é feita através de estados de *phonons* compartilhados.
3. *Preparação do estado inicial.* Resfriar os átomos (por aprisionar e usar extração ótica) dentro de seus estados base de movimento, e estados base hiperfinos.
4. *Leitura.* Mede-se as populações dos estados hiperfinos.
5. *Desvantagens.*
   (a) A duração da vida dos *phonons* é muito pequena.
   (b) É muito difícil preparar os íons em seus estados base de movimento.
   (c) Os computadores quânticos feitos com armadilhas de íons são, por definição, feitos de dispositivos lentos. A velocidade destes dispositivos é limitada, no final das contas, pela incerta relação tempo-energia.

### 4.1.2 Cavidade quântica eletrodinâmica

Um projeto de *hardware* alternativo (proposto por Pellizari, Gardiner, Cirac e Zoller) está sendo prosseguido pelo grupo de Jeff Kimble no Caltech, *Instituto de tecnologia da Califórnia*. A idéia é aprisionar vários átomos neutros dentro de uma cavidade ótica altamente sutil. A informação quântica pode, então, ser armazenada dentro dos estados internos dos átomos. Contudo, aqui os átomos interagem devido a todo o seu emparelhamento com o modo normal do campo eletromagnético na cavidade (ao invés de modos vibracionais como na armadilha de íons). Novamente, através do direcionamento de transições com pulsos de laser, pode-se induzir a transição em um átomo que está condicionado no estado interno de outro átomo.

Outra possibilidade é armazenar um quBit, não no estado interno de um íon, mas na polarização de um fóton. Então, um átomo aprisionado pode ser utilizado com o intermediário capaz de fazer com que um fóton intereja com outro fóton (ao invés do fóton sendo utilizado juntar um átomo a outro).

O grupo de Kimble demonstrou a operação de duas portas lógicas quântica 2 (dois) anos atrás. Nesta demonstração, a polarização circular de um fóton influencia a *fase* de outro fóton.

$$\begin{aligned} |L\rangle_1 |L\rangle_2 &\rightarrow |L\rangle_1 |L\rangle_2 \\ |L\rangle_1 |R\rangle_2 &\rightarrow |L\rangle_1 |L\rangle_2 \\ |R\rangle_1 |L\rangle_2 &\rightarrow |R\rangle_1 |L\rangle_2 \\ |R\rangle_1 |R\rangle_2 &\rightarrow e^{i\Delta} |R\rangle_1 |L\rangle_2 \end{aligned} \tag{4.4}$$

onde $|L\rangle$, $|R\rangle$ denotam os estados do fóton com a polarização esquerda (*Left*) e direita (*Right*). Para atingir esta interação, um fóton é armazenado na cavidade, onde a polarização $|L\rangle$ não se emparelha ao átomo, mas a polarização $|R\rangle$ o faz fortemente. No segundo fóton, também, uma polarização interage dando preferência a um átomo. O pacote de onde do segundo fóton adquire uma fase particular deslocada $e^{i\Delta}$ apenas se ambos os fótons têm polarização $|R\rangle$. Devido ao fato de o deslocamento de fase ser condicionado nas polarizações de ambos os fótons, esta não é uma porta lógica quântica de dois quBits de fácil construção.

**QUADRO-RESUMO**

1. *Representação do quBit.* Ambientação de um simples fóton entre dois modos, $|01\rangle$ e $|10\rangle$, ou polarização.
2. *Evolução unitária.* Transformações arbitrárias são construídas a partir de deslocadores de fase (rotações $R_z$, divisores de luz (rotações $R_y$), e uma cavidade quântica eletrodinâmica, para que o campo ótico seja emparelhado.
3. *Preparação do estado inicial.* Cria simples estados de fóton, através da atenuação da luz laser.
4. *Leitura.* Detecta simples fótons utilizando tubos fotomúltiplos.
5. *Desvantagens.* O emparelhamento de dois fótons é mediado por um átomo, e deste modo, é desejável aumentar o emparelhamento do campo atômico. Entretanto, emparelhar os fótons dentro e fora da cavidade eletrodinâmica é difícil e limita o cascateamento.

### 4.1.3 Ressonância magnética nuclear

Segundo [Nielsen e Chuang, 2000], um terceiro esquema de hardware, inicialmente desecreditado e que mostrou-se promissor, apareceu há poucos anos e ultrapassou os

esquemas de armadilha de íons e cavidade quântica eletrodinâmica para se tornar o mais importante projeto de processamento quântico coerente. Este novo esquema utiliza a tecnologia de ressonância magnética nuclear, RMN. Agora os quBits são carregados em spins nucleares em moléculas particulares. Cada spin pode ser *alinhado* ($|\uparrow\rangle = |0\rangle$) ou *anti-alinhado* ($|\downarrow\rangle = |1\rangle$) com um campo magnético constante aplicado. Os spins demoram um longo tempo até relaxar ou tornarem-se decoerentes, assim, os quBits podem ser armazenados por um tempo razoável.

Pode-se também, ligar um campo magnético de rotação intermitente com freqüência $\omega$ (onde $\omega$ é a divisão de energia entre os estados *spin-up* e o *spin-down*), e induzir as oscilações de Rabi do spin. Através da deterinação adequada do tempo de pulso, pode-se fazer uma transformação unitária desejada sobre um simples spin (da mesma forma que discutido nas armadilhas de íons). Todos os spins na molécula são expostos ao campo de rotação magnética mas apenas aqueles sob ressonância respondem.

Além disso, os spins têm interações dipolo-dipolo, e este emparelhamento pode ser explorado para se fazer uma porta lógica. A divisão entre $|\uparrow\rangle$ e $|\downarrow\rangle$ para um spin depende dos estados dos spins vizinhos.

Tudo isto já era conhecido por químicos há décadas. Apesar disso, foi apenas no ano passado que Gershenfeld e Chuang mostraram que a ressonância magnética nuclear providenciava uma implementação útil para computação quântica. Isto não era óbvio por diversas razões. A mais importante é que os sistemas de RMN são muitos quentes. As temperaturas típicas dos spins pode ser da ordem de milhões de vezes maior que a energia de divisão entre $|0\rangle$ e $|1\rangle$. Isto significa que os estados quânticos deste computador (os spins em uma simples molécula) são muito *ruidosos* — estão sujeitos a flutuações de temperatura fortemente randômicas —. Este *ruído* irá distinguir as informações quânticas. Além disso, atualmente, pode-se fazer processamento não sobre uma molécula apenas mas fazer amostragens macroscópicas contendo na ordem de $10^{23}$ *computadores* e o sinal medido deste dispositivo é a media deste grupo. Todavia, os algoritmos quânticos são robabilísticos, devido à falta de método da medida quântica. Por conseguinte, calcular a média sobre o grupo não é equivalente a executar a computação sobre um simples dispositivo; o cálculo da média pode obscurecer os resultados.

Gershenfeld e Chuang explicaram como superar estas dificuldades. Eles descreveram como *estados efetivos puros* podem ser preparados, manipulados e monitorados fazendo-se operações adequadas na temperatura do grupo. A idéia é arranjar as propriedades de flutuação das moléculas para calcular a média quando o sinal for detectado, assim apenas as propriedades coerentes base são medidas.

Atualmente já se consegue desenvolver dispositivos de três quBits com esta tecnologia. Este é um resultado que nunca fora atingido.

**QUADRO-RESUMO**

1. *Representação do quBit.* Spins dos núcleos atômicos
2. *Evolução unitária.* Tranformações arbitrárias são construídas a partir pulsos de campos magnéticos aplicados aos spins em um campo magnético forte. Emparelhamento entre os spins são providenciados pelos limites químicos entre os átomos vizinhos.
3. *Preparação do estado inicial.* Polarizar os spins por colocá-los em um campo magnético forte, então usar técnicas de preparação de *estados efetivos puros*
4. *Leitura.* Medir o sinal de voltagem induzido pelo momento magnético precedente.
5. *Desvantagens.* Esquemas para preparação de *estados efetivos puros* reduzem o sinal exponencialmente no número de quBits, a menos que a polarização inicial seja suficientemente alta.

**QUADRO COMPARATIVO**

| DISPOSITIVO | PRINCIPAIS CARACTERÍSTICAS |
|---|---|
| Fótons simples | *Podem servir como bons quBits, usando* $\|01\rangle$ *e* $\|10\rangle$ *como 0 e 1 lógicos. Entretanto, materiais óticos não linerares convencionais que sejam suficientemente fortes para permitir interações simples entre os fótons inevitavelmente absorvem ou espalham estes.* |
| Cavidade quântica eletrodinâmica | *É uma técnica pela qual átomos únicos podem interagir fortemente com fótons únicos. Isto provê um mecanismo para se usar um átomo para mediarinterações entre fótons.* |
| Armadilha de íons | *Podem ser congelados de modo a permitir que seus estados nucleares de spin e eletrônicos possam ser controlados através da aplicação de pulsos de laser. Pelo emparelhamento de estados de spin através de* phonons *de centros de massa, portas lógicas entre diferentes íons podem ser construídas* |
| Spins nucleares | *Estão muito próximos dos quBits ideais, e moléculas simples poderiam estar próximas de computadores quânticos ideais se seus estados de spin pudessem apenas ser controlados e medidos. A ressonância magnética nuclear torna isto possível usando grandes grupos de moléculas, mas o custo de perda de sinal devido a um ineficiente procedimento de preparação é alto* |

Tabela 4.1: Realização física dos computadores quânticos proposta por [Nielsen e Chuang, 2000]

## 4.2 Os algoritmos estudados — fundamentos

### 4.2.1 O *Problema de Deutsch*

**Visão geral**

David Deutsch deu um exemplo mais concreto da idéia de Richard Feymann, sobre as possibilidades e vantagens de um computador quântico, em 1985. Deutsch enfatizou que um computador quântico poderia realizar melhor seu potencial computacional se utilizasse o chamado *paralelismo quântico*. Para entender o que isso significa, considere o exemplo a seguir.

Seguindo a idéia de Deutsch, imagine que se tenha uma *caixa preta* que computa uma função que mapeia um simples bit $x$ para um simples bit $f(x)$. Não se conhece o que ocorre dentro da caixa, mas suponha algo complicado o suficiente para que sua computação consuma 24 horas. Há quatro possibilidades de funções para $f(x)$ pois $f(0)$ pode ser $0$ ou $1$ e, por sua vez, $f(1)$ pode, também, ser $0$ ou $1$. Não se sabe o valor que a caixa preta está computando. Logo, gasta-se 48 horas para que $f(0)$ e $f(1)$ sejam calculadas.

Suponha, agora, que o objetivo seja saber se $f(x)$ é *constante* com $f(0) = f(1)$ ou *balanceada* com $f(0) \neq f(1)$. Considere agora que não se tenha 48 horas disponíveis para tal atingir tal objetivo; precisa-se de uma resposta em 24 horas.

Agora suponha que se tenha uma *caixa preta quântica* capaz de computar $f(x)$. De fato, $f(x)$ pode não ser uma função inversível, enquanto a ação deste *computador quântico* é unitária e precisa ser inversível. Assim, é preciso uma transformação $U_f$ que leve 2 quBits em outros dois quBits:

$$U_f : |x\rangle\,|y\rangle \rightarrow |x\rangle\,|y \oplus f(x)\rangle\,.$$

A operação $y \oplus f(x)$ é a operação de *xor* binária.

Esta máquina *flipa* o segundo quBit se $f$ atua no quBit 1 resulta em 1 e se $f$ atua no quBit 2 resulta em 0. Claramente, pode-se concluir que a a função $f(x)$ é *constante* ou *balanceada* utilizando-se a caixa-preta duas vezes. Mas isto ainda tomaria 48 horas para fazê-lo. Logo, isto não será feito. Este é o problema de Deutsch.

Pelo fato de a caixa preta ser um computador quântico, pode-se escolher a entrada como sendo uma *superposição* de $|0\rangle$ e $|1\rangle$. Se o segundo quBit for inicialmente preparado no estado: $\frac{1}{\sqrt{2}}\,(|0\rangle - |1\rangle)$, então:

$$U_f : |x\rangle\,\frac{1}{\sqrt{2}}\,(|0\rangle - |1\rangle) \quad \rightarrow \quad |x\rangle\,\frac{1}{\sqrt{2}}\,(|f(x)\rangle - |1 \oplus f(x)\rangle)$$

$$= \quad |x\rangle\,(-1)^{f(x)}\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

Deste modo, isolou-se a função $f$ em uma fase dependente de $x$. Agora suponha que o primeiro quBit tenha sido preparado no estado: $\frac{1}{\sqrt{2}}\,(|0\rangle + |1\rangle)$. Então a caixa preta atuará como:

$$\begin{array}{ll} U_f: & \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \rightarrow \\ & \frac{1}{\sqrt{2}}\left[(-1)^{f(0)}\,|0\rangle + (-1)^{f(1)}\,|1\rangle\right]\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \end{array}$$

Finalmente, pode-se fazer uma medida que projeta o primeiro quBit sobre a base:

$$|\pm\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle).$$

Evidentemente, sempre será obtido $|+\rangle$ se a função é *balanceada*, e $|-\rangle$ se a função é *constante*[1]

Com esta abordagem o problema de Deutsch foi resolvido e agora é possível fazer uma separação entre o que os computadores clássicos e os quânticos podem atingir. Um computador clássico precisa executar a *caixa preta* duas vezes para comparar duas funções unárias e classificá-las como *contantes* ou *balanceadas*. Por outro lado, um computador quântico faz a mesma tarefa em apenas um passo. Realmente um ganho considerável.

Isto é possível porque os computadores quânticos não são limitados a trabalhar com funções $f(0)$ ou $f(1)$. Eles podem atuar em uma superposição dos estados $|0\rangle$ e $|1\rangle$, e, através disso, extrair uma informação "global" sobre a função, informação esta que depende de $f(0)$ e $f(1)$ correlacionando-as. Isto é o *paralelismo quântico* [Preskill, 2002].

### 4.2.2 Algoritmo da *Transformada Quântica de Fourier*

#### Visão geral

De acordo com [Nielsen e Chuang, 2000], a *Transformação de Fourier* mapeia funções no domínio do tempo em funções no domínio da freqüência. Existe uma versão discreta da transformada de Fourier (TDF) que mapeia conjuntos finitos de pontos $\{(t_i, f_i)\}$ em

[1] Geralmente a medida final de uma computação quântica projeta cada quBit sobre a base $\{|0\rangle, |1\rangle\}$. Entretanto, aqui foi permitido uma medida em uma base diferente. Para proceder como anteriormente, sem perda de generalidade, basta aplicar-se uma mudança unitária da base para cada quBit antes de fazer a medida final.

pontos $\{(\nu_i, \tilde{f}_i)\}$. Estes pontos $\{(t_i, f_i)\}$ podem ser encarados como amostragens de uma função $f(t)$ tomadas em intervalos regulares.

A *Transformada Rápida de Fourier* (TRF) é um algoritmo muito eficiente para calcular a TDF. O algoritmo particiona o conjunto $\{f_i\}$ em subconjuntos de dois elementos. É necessário observar que o número de pontos do conjunto deve ser potência de 2: $N = 2^r$.

Por outro lado, a *Transformada Quântica de Fourier* (TQF) é uma operação sobre estados quânticos inspirada pela TDF. Seja um espaço de *Hilbert* $H$ de dimensão finita $N$. Seja $\{|x\rangle\}$ ($x = 0,\ 1,\ 2,\ 3,\ \ldots\ , N-1$) uma base neste espaço. Define-se a TQF como o operador linear que faz o mapeamento

$$U_{TQF}\,|x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{\frac{2\pi i x y}{N}}\,|y\rangle, \tag{4.5}$$

ou seja, a TQF leva um estado $|x\rangle$ da base numa combinação linear de estados da base.

Sabendo como determinada operação linear atua sobre um vetor da base, sabemos como ela atua sobre todo o espaço. Basta então defini-la através de sua atuação sobre um elemento da base de $H$.

Fisicamente a implementação da TQF depende de um sistema que possa ser submetido a uma evolução temporal que promova a transformação definida. Se o sistema for um conjunto de *spins*, por exemplo, pulsos eletromagnéticos cuidadosamente escolhidos podem realizá-la.

### 4.2.3 Algoritmo da *Fatoração de Shor*

#### Visão geral

Em 1994, inspirado pelos trabalhos de Daniel Simon, mais tarde re-publicado como [Simon, 1997], Peter Shor descobriu um algoritmo de tempo polinomial para fatoração de números de *n*-dígitos utilizando-se um *computador quântico*.

A pesquisa por algoritmos eficientes de fatoração data da década de 1970. Utilizando-se a computação clássica, atualmente, o algoritmo de fatoração mais eficiente é [Lenstra e Lenstra, 1993], o qual é exponencial em relação ao tamanho da entrada. Esta é o número de dígitos de um número *M* a ser fatorado. Muitas pessoas estavam confiantes de que um algoritmo polinomial não existia. Deste modo, ao publicar seu algoritmo quântico para fatoração, Shor chamou a atenção para a computação quântica e suas potencialidades. Além disso, a existência de um algoritmo eficiente de fatoração poderia destruir o mais eficiente sistema de criptografia da atualidade: *o sistema de criptografia de chave pública baseado no algoritmo RSA.*

Segundo [Rieffel e Polak, 2000], a maioria dos algoritmos de fatoração, inclusive o de Shor, usa uma redução padrão para o problema de achar o período de uma função sobre o conjunto dos inteiros. Shor utiliza o *parelelismo quântico* para obter uma representação de todos os valores de uma função em um passo. Esta representação é possível porque os estados quânticos referentes aos vários valores da função podem ser *superpostos*.

Ele, então, calcula a TQF. Com uma alta probabilidade, cerca de 40 por cento, a medição do estado resultará no período que, por sua vez, será utilizado para fatorar um número inteiro emphM. OBS.: o resultado da TQF só pode ser avaliado probabilisticamente. Sendo assim, pode ser necessário executar o algoritmo mais de uma vez para conseguir fatorar um número.

### O algoritmo em si

Maiores informações sobre o algoritmo podem ser encontradas em [Rieffel e Polak, 2000].

Restricões do algoritmo. O número *M* a ser fatorado:

1. precisa ser >= 15;
2. deve ser ímpar;
3. deve ser não-primo;
4. não deve ser uma potência de primos;

*Passo 0.* Leia M, o número a ser fatorado.

*Passo 1. Paralelismo quântico.* Escolha um inteiro *a* arbitrariamente. Se *a* não é coprimo de *M*, um fator de *M* foi achado. Caso contrário, aplica-se o resto do algoritmo.

Seja, *m*, tal que $M^2 \leq 2^m < 2M^2$. OBS.: esta escolha é feita de modo que a aproximação usada no *Passo 3*, para funções das quais o período não seja uma potência de 2, irá ser suficiente para o resto do algoritmo. Posteriormente, utilizando-se o *paralelismo quântico*, computa-se $f(x) = a^x \ mod \ M$ para todos os inteiros $x$ de $0 \ a \ 2^m - 1$. A função é então codificada no estado quântico:

$$\frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} |x\rangle \, , |f(x)\rangle. \tag{4.6}$$

Por exemplo, suponha $a = 11$ randomicamente escolhido. Desde que $M^2 = 441 \leq 2^9 < 882 = 2M^2$ acha-se $m = 9$. Deste modo, são necessários 14 bits quânticos, quBits, para calcular a equação de superposição 4.6. Serão necessários 9 quBits para representar $m$, que é a potência de 2 que satisfaz a condição $M^2 \leq 2^m < 2M^2$, em forma binária. Os 5 quBits restantes são necessários para a representação do número $M$ em forma binária.

*Passo 2. Ache um estado cuja amplitude tenha o mesmo período de f.* A transformação quântica de Fourier atua sobre a função de amplitude associada com o estado de entrada. Desta forma, para se usar a TQF para obter um período de *f*, um estado cuja função de amplitude tenha o mesmo período que *f*, é construído.

Para construir tal estado, mede-se os últimos $\lceil \log_2 M \rceil$ quBits do estado da esquação 4.6 que codifica $f(x)$. Um valor randômico $u$ é obtido. O valor $u$ em si não é interessante; apenas o efeito da medida sobre o conjunto de superposições é de particular interesse. Esta medida projeta o espaço de estados sobre o subspaço compatível com o valor medido, assim, o espaço de estados após a medição é:

$$C \sum_x g(x) \left|x, u\right\rangle, \tag{4.7}$$

para algum valor $C$ onde

$$g(x) = \begin{cases} 1 & \text{se } f(x) = u \\ 0 & \text{caso contrário.} \end{cases}$$

Note que os $x's$ que aparecem na soma, aqueles com $g(x) \neq 0$, diferem uns dos outros por múltiplos do período, assim $g(x)$ é a função que se está procurando. Infelizmente as leis da física quântica permitem apenas uma e somente uma medição.

*Passo 3. Aplique a transformada quântica de Fourier.* A parte $|u\rangle$ do estado não irá ser utilizada — assim, não se irá detalhá-la por hora —. Aplica-se a TQF para o estado obtido com o passo 2.

$$U_{TQF} : \sum_x g(x) \left|x\right\rangle \rightarrow \sum_c G(c) \left|c\right\rangle$$

A análise padrão de Fourier diz que quando o período $r$ da função $g(x)$ definida no passo 2 é uma potência de 2, o resultado da TQF é:

$$\sum_j c_j \left| j\frac{2^m}{r} \right\rangle,$$

onde a amplitude é zero exceto nos múltiplos de $\frac{2^m}{r}$. Quando o período $r$ não divide $2^m$ a TQF dá o período $r$ com uma probabilidade menor do que um. Deste modo, a maioria das amplitudes é formada por inteiros próximos dos múltiplos de $\frac{2^m}{r}$.

*Passo 4. Extraia o período.* Meça o estado na base padrão para a computação quântica, e chame o resultado $v$. No caso em que o período seja potência de 2, assim que a TQF resulte nos múltiplos de $\frac{2^m}{r}$, o período será facilmente extraído. Neste caso, $v = j\frac{2^m}{r}$ para algum j. Na maioria das vezes $j$ e $r$ serão *primos relativos*, no caso em que reduzir a fração $\frac{v}{2^m} = \frac{j}{r}$) para os menores termos ainda resulta uma fração cujo denominador $q$ é o período $r$. Quando o período não é uma potência de 2 uma boa suposição para o período pode ser obtida usando uma *expansão em frações continuadas* de $\frac{v}{2^m}$.

*Passo 5. Ache o fator de M.* Dada a suposição para o período $q$, usa-se o algoritmo de Euclides para checar se $(a^{\frac{q}{2}} + 1)$ ou $(a^{\frac{q}{2}} - 1)$ tem algum fator não trivial comum com M. Isto quer dizer que se $q$ é realmente o período de $f(x) = a^x \ mod \ M$, então $a^q = 1 \ mod \ M$ dado que $a^q a^x = a^x \ mod \ M$ para todo $x$. Se $q$ é par pode-se escrever

$$(a^{\frac{q}{2}} + 1)(a^{\frac{q}{2}} - 1) = 0 \ \textit{mod M}.$$

Assim, ou $(a^{\frac{q}{2}} + 1)$ e $(a^{\frac{q}{2}} - 1)$ não são múltiplos de $M$ ou um dos dois tem um fator comum não trivial com $M$.

*Passo 6. Repita o algoritmo, se necessário.* Vários problemas poderiam acontecer na execução dos passos anteriores do algoritmo, não resultando em um fator de $M$. Tais problemas poderiam ser:

1. O valor $v$ não é próximo o suficiente para um múltiplo $\frac{2^m}{r}$;
2. O período $r$ e o múltiplo $j$ poderiam ter um fator comum assim que o denominador $q$ fosse um fator do período e não o período em si;
3. O passo 5 resulta $M$ como um fator de $M$;
4. O período de $f(x) = a^x \ mod \ M$ é ímpar.

Shor demonstrou que poucas repetições desde algoritmo resolvem todos estes problemas.

### Exemplo de funcionamento

Entrada M = 15.

*Passo 1.*

1. Suponha que $a = 7$ foi randomicamente escolhido; Nesse caso, $a$ é co-primo de $M$. Co-primo quer dizer relativamente primo.
2. Em seguida acha-se $m = 8$ dado que $m$ é a potência de 2 tal que $M^2 \leq 2^m < 2$, i.e., $15^2 \leq 2^8 < 2 * 15^2$.
3. Cria-se o registrador quântico $\#1$, que conterá as superposições. Este terá 8 quBits.
4. Cria-se o registrador quântico $\#2$, que conterá os dados do universo do registrador $\#1$ superpostos. Este terá 4 posições dado que são necessários 4 bits para representar o número 15 em forma binária, $15_{10} = 1111_2$.
5. Preenche-se o registrador 1 com dados pares superpostos.

O resultado de 3 é:

$(0) \quad 0,00000000,\ 0,00000000i$
$(1) \quad 0,00000000,\ 0,00000000i$
$(2) \quad 0,00000000,\ 0,00000000i$
$(3) \quad 0,00000000,\ 0,00000000i$
$(4) \quad 0,00000000,\ 0,00000000i$
$(5) \quad 0,00000000,\ 0,00000000i$
$(6) \quad 0,00000000,\ 0,00000000i$
$(7) \quad 0,00000000,\ 0,00000000i$

O resultado de 4 é:

$(0) \quad 0,00000000,\ 0,00000000i$
$(1) \quad 0,00000000,\ 0,00000000i$
$(2) \quad 0,00000000,\ 0,00000000i$
$(3) \quad 0,00000000,\ 0,00000000i$

O resultado de 5 é:

$(0) \quad 0,06262243,\ 0,00000000i$
$(1) \quad 0,06262243,\ 0,00000000i$
$(2) \quad 0,06262243,\ 0,00000000i$
$(3) \quad 0,06262243,\ 0,00000000i$
$(4) \quad 0,06262243,\ 0,00000000i$
$(5) \quad 0,06262243,\ 0,00000000i$
$(6) \quad 0,06262243,\ 0,00000000i$
$(7) \quad 0,06262243,\ 0,00000000i$

Agora é feita a exponenciação modular sobre os elementos superpostos do registrador 1. Isto é, faz-se $x^a \ mod \ M$. Vale lembrar que, explorando-se o paralelismo quântico, em um computador quântico, isto poderia ser feito em um passo, ao invés de calcular cada valor possível e medir o valor no registrador $\#1$.

O resultado será armazenado no registrador $\#2$, que é emaranhado com o primeiro registrador. Isto significa que quando um for medido, e colapsar seus estados base, o outro precisa estar colapsado dentro de uma superposição de estados consistentes com o valor medido no primeiro registrador.

*Passo 2.*

Acha-se um estado cuja amplitude tenha o mesmo período de $f$. Neste caso, mede-se os últimos $\log_2 M$ quBits obtendo-se uma projeção do espaço de estados sobre o subspaço compatível. O resultado é:

$$
\begin{array}{ll}
(0) & 0,00000000,\ 0,00000000i \\
(1) & 0,00000000,\ 0,00000000i \\
(2) & 0,00000000,\ 0,00000000i \\
(3) & 1,00000000,\ 0,00000000i \\
(4) & 0,00000000,\ 0,00000000i \\
(5) & 0,00000000,\ 0,00000000i \\
(6) & 0,00000000,\ 0,00000000i \\
(7) & 1,00000000,\ 0,00000000i
\end{array}
$$

Em seguida, *normaliza-se* o resultado. O que produz:

$$
\begin{array}{ll}
(0) & 0,00000000,\ 0,00000000i \\
(1) & 0,00000000,\ 0,00000000i \\
(2) & 0,00000000,\ 0,00000000i \\
(3) & 0,12500000,\ 0,00000000i \\
(4) & 0,00000000,\ 0,00000000i \\
(5) & 0,00000000,\ 0,00000000i \\
(6) & 0,00000000,\ 0,00000000i \\
(7) & 0,12500000,\ 0,00000000i
\end{array}
$$

*Passo 3.*

Aplica-se a TQF. Isto resulta em:

$$
\begin{array}{ll}
(0) & 0,50000000,\ 0,00000000i \\
(1) & 0,00000000,\ 0,00000000i \\
(2) & 0,00000000,\ 0,00000000i \\
(3) & 0,00000000,\ 0,00000000i \\
(4) & 0,00000000,\ 0,00000000i \\
(5) & 0,00000000,\ 0,00000000i \\
(6) & 0,00000000,\ 0,00000000i \\
(7) & 0,00000000,\ 0,00000000i
\end{array}
$$

para o registrador $\#1$. E:

$(0) \quad 0,00000000,\ 0,00000000i$
$(1) \quad 0,50000000,\ 0,00000000i$
$(2) \quad 0,00000000,\ 0,00000000i$
$(3) \quad 0,00000000,\ 0,00000000i$

para o registrador 2.

*Passo 4.*

A extração do período. Faz-se uma medição no registrador $\#1$. Isto colapsa toda a informação dentro do registrador. Deste modo, tem-se $m = 64$, e $q = 256$ logo tem-se o valor $c = 0.25$ pois $c = \frac{m}{q}$. Em seguida calcula-se o denominador da melhor aproximação racional para $c$ com $denominador < q$. Desde que $c$ é $\lambda/r$ para algum inteiro $\lambda$, isto irá providenciar o PERÍODO $r$ da função, embora este seja HIPOTÉTICO, por enquanto.

*Passo 5.*

O cálculo de um fator de $M$. A partir do algoritmo de Euclides tem-se o primeiro fator de M como sendo $Fator_1 = 3$. Daí, trivialmente chega-se ao outro fator fazendo-se $Fator_2 = \frac{M}{Fator_1}$

## 4.3 SAQua - Simulador de Algoritmos Quânticos

O SAQua é composto, até o momento, de 17 (dezessete) classes. Até o momento há, aproximadamente, 5.000 (cinco mil) linhas de código construídas.

As classes são:

1. *public class* ***SAQua*** *extends JApplet*
2. *public class* ***Complexos***
3. *public class* ***QuBit***
4. *public class* ***FrameDeutsch*** *extends JInternalFrame*
5. *public class* ***FrameFourier*** *extends JInternalFrame*
6. *public class* ***FrameShor*** *extends JInternalFrame*
7. *public class* ***FrameSobre*** *extends JInternalFrame*
8. *public class* ***BaseFourier***
9. *public class* ***FourierTask***
10. *public class* ***Fourier*** *extends Operacoes implements Runnable*

11. *public class* ***ShorTask***
12. *public class* ***Shor*** *extends Operacoes implements Runnable*
13. *public abstract class* ***SwingWorker***
14. *public class* ***HandlerBotoes*** *implements ActionListener*
15. *public class* ***HandlerMenu*** *implements ActionListener*
16. *public class* ***Operacoes*** *extends Thread*
17. *public class* ***Registradores***

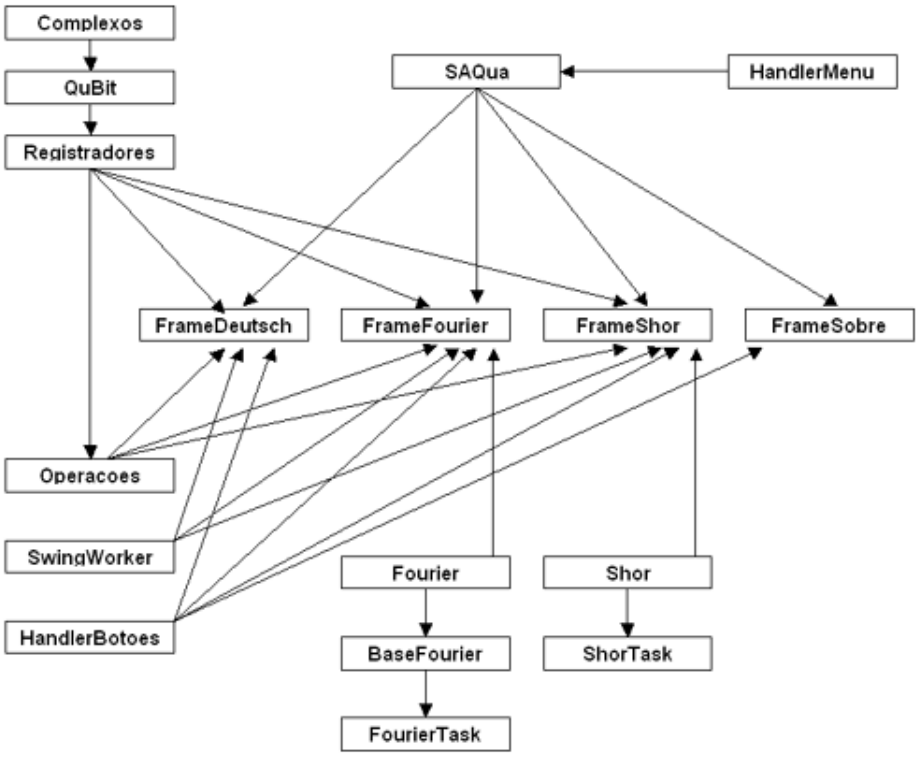


Figura 4.1: Diagrama de dependência entre as classes

### 4.3.1 Classe *SAQua.java*

Esta é a classe principal do simulador. É responsável por mostrar o ambiente de simulação na tela do micro-computador. Seus principais métodos são:

- *public void init()* — Inicializa todos os componentes gráficos do aplicativo;
- *private JToolBar criarToolBar* — Cria a barra de ferramentas localizada na parte superior esquerda da interface gráfica do simulador;
- *private void criarIcones* — Cria todos os ícones utilizados pela interface gráfica do simulador;
- *private JMenu criarMenus* — Cria o menu principal do simulador;
- *protected void criarFrameFourier* — Cria o *frame* de simulação do algoritmo da *Transformada Quântica de Fourier*;
- *protected void criarFrameSobre* — Cria o *frame* sobre os desenvolvedores do sistema;
- *protected void criarFrameShor* — Cria o *frame* de simulação do algoritmo da *Fatoração de Shor*;
- *protected void criarFrameDeutsch* — Cria o *frame* de simulação do *Problema de Deutsch*;
- *public static void main* — É o principal método do simulador. Este método é o primeiro a ser executado quando se lança o aplicativo SAQua;

### 4.3.2 Classe *Complexos.java*

Esta classe é a representação matemática de um número complexo. Um meio encontrado pelos físicos para representação quântica dos quBits foi através de números complexos. Desta forma, cada quBit pode ser representado por uma dupla de números complexos. Estes números são chamados *alfa* e *beta*. Formalmente, os quBits são representados na forma $[alfa\,|0\rangle + beta\,|1\rangle]$. O fator *alfa* representa a amplitude de probabilidade do estado $|0\rangle$ e beta a amplitude de probabilidade do estado $|1\rangle$. Assim, a classe *Complexos.java* é imprescindível ao funcionamento do simulador. Seus principais métodos são:

- *public Complexos()* — Aloca espaço na memória para a representação de um número complexo. É importante salientar que este número ainda não possui sua parte real e imaginária definidas.
- *public Complexos(double novoReal, double novoImaginario)* — Constrói um número complexo na forma $a + bi$ onde $a$ é sua parte real e $b$ é sua parte imaginária;
- *public void set(double novoReal, double novoImaginario)* — Dado um número complexo existente, este método permite mudar seu coeficiente real e imaginário;

- *public void setReal(double novoReal)* — Dado um número complexo existente, permite mudar seu coeficiente real;
- *public void setImaginaria(double novoImaginario)* — Dado um número complexo existente, permite mudar seu coeficiente imaginário;
- *public double getReal()* — Dado um número complexo existente, retorna o seu coeficiente real;
- *public double getImaginaria()* — Dado um número complexo existente, retorna o seu coeficiente imaginário;
- *public Complexos soma(Complexos cmp)* — Realiza a operação de soma entre dois números complexos;
- *public Complexos mult(Complexos cmp)* — Realiza a operação de multiplicação entre dois números complexos;
- *public Complexos recebe(Complexos cmp)* — Atribui um número complexo a outro;
- *public boolean igual(Complexos cmp)* — Verifica se um número complexo é igual a outro;

### 4.3.3 Classe *QuBit.java*

Define as propriedades de um quBit. Como já dito, um quBit é representado fisicamente por $[alfa\,|0\rangle + beta\,|1\rangle]$. Os seus principais métodos são:

- *public QuBit()* — Constrói um quBit básico da forma $[alfa\,|0\rangle + beta\,|1\rangle]$ com $alfa = 1$ e $beta = 0$;
- *public String espiar()* — Mede os estados de um quBit sem colapsá-los. Obviamente, em um computador quântico real, este método não seria realizável fisicamente dado que é impossível "ler" os estados quânticos sem colapsá-los;
- *public int medicao()* — Retorna o estado $|0\rangle$ com probabilidade $|alfa|^2$ ou o estado $|1\rangle$ com probabilidade $|beta|^2$. Corresponde a medir a componente Z do *spin* de um elétron;
- *public void setEstado(Complexos probZero, Complexos probUm)* — Seta um estado a partir de dois numeros complexos representando as amplitudes de probabilidades;
- *public void setMedia()* — Seta a componente $alfa$ e $beta$ do quBit para $\sqrt{2}$. Como não há mais a componente imaginária não há componente de fase;

- *public double modulo(int estado)* — Seja $alfa$ um número complexo e seja $alfa'$ seu complexo conjugado. Deste modo, esta operação faz a multiplicação de $(alfa * alfa')$ e retorna o valor obtido. Isto também é válido para a componente $beta$ do quBit. Vale lembrar que esta operação sempre retorna um número real;

### 4.3.4 Classe *FrameDeutsch.java*

Este classe contém a interface gráfica do *frame* de simulação do *Problema de Deutsch*. Por ser um algoritmo muito simples tudo relacionado a este foi implementado nesta classe. Desta forma, este algoritmo não é como a *Fatoração de Shor*, por exemplo, que precisa de várias classes para seu funcionamento. Seus principais métodos e/ou classes internas são:

- *public FrameDeutsch()* — Constrói o *frame* de simulação do algoritmo;
- *private Border criarBorda(String titulo)* — Cria todas as bordas necessárias para uma boa visualização deste *frame*;
- *private void resolverProblemaDeutsch()* — Este é o algoritmo propriamente dito. Por ser simples apenas este método foi necessário;
- *private class HandlerRadio implements ItemListener* — Esta classe privada é responsável por captar qualquer ação do usuário sobre este *frame* de simulação;

### 4.3.5 Classe *FrameFourier.java*

Este classe contém a interface gráfica do *frame* de simulação da *Transformada Quântica de Fourier*. Seus principais métodos são:

- *public FrameFourier()* — Constrói o *frame* para simulação do algoritmo da *Transformada Quântica de Fourier*;
- *private JScrollPane criarObservacoes()* — Cria o painel de observações sobre a execução deste algoritmo;
- *private Border criarBorda()* — Cria as bordas utilizadas no *frame*;
- *private void apresentarTabela()* — Cria as tabelas utilizadas no *frame*;
- *private void atualizarDados(JTable tabela)* — Atualiza os dados das tabelas a partir da execução do algoritmo;
- *private carregarBaseDados(String arquivoSelecionado)* — Carrega uma base numérica em arquivo onde estão os pontos na forma $(x, y)$ a serem transformados pelo algoritmo;

- *private void transformar()* — Realiza a *Transformada Quântica de Fourier* propriamente dita;
- *private void abrir()* — Apresenta uma caixa de diálogo onde o usuário pode selecionar um novo arquivo contendo um base de dados para ser transformado;
- *private Timer criarTimer()* — Cria o *timer* utilizado para sincronizar os passos do algoritmo e permitir ao usuário o seu acompanhamento
- *private void adicionarListeners()* — Adiciona os *listeners* responsáveis por capter qualquer ação do usuário neste *frame* de simulação;
- *public TableModel modelarTabela()* — Faz a modelagem das tabelas utilizadas por esse *frame*;

### 4.3.6 Classe *FrameShor.java*

Este classe contém a interface gráfica do *frame* de simulação do algoritmo da *Fatoração de Shor*. Seus principais métodos são:

- *public FrameShor()* — Constrói o *frame* para simulação do algoritmo da *Fatoração de Shor*;
- *private void setOpcoesCombo()* — Cria as opções do *ComboBox* do *frame* que será utilizado para fornecer opções de ação ao usuário;
- *private Timer criarTimer()* — Cria o *timer* utilizado para sincronizar os passos do algoritmo e permitir ao usuário o seu acompanhamento
- *private JPanel criarPainel(String titulo, Component cmps[], int x, int y, int tx, int ty)* — Todo o *frame* funciona através de painéis. Este método permite a sua criação de forma mais dinâmica e econômica;

### 4.3.7 Classe *FrameSobre.java*

Esta classe exibe um *frame* onde estão os créditos de desenvolvimento do simulador SAQua. Seus principais métodos são:

- *public FrameSobre()* — Constrói o *frame* sobre os desenvolvedores do SAQua;
- *private JEditorPane criarEditorPane(String pagina)* — Todo o *frame* funciona através da exibição de *Painéis editores*. Este método permite sua crição de forma eficiente, dado que todo este *frame* foi desenvolvido para mostrar os créditos em uma página *html*;

- *private void mostrarURL(URL url, JEditorPane ep)* — Permite o carregamento de uma página *html* especificada. A página será exibida no *JEditorPane* passado como parâmetro;

### 4.3.8 Classe *BaseFourier.java*

Esta classe é responsável por pegar um arquivo de base de dados, convencionado com a extensão (.FOU) de FOUrier, e carregá-lo para a memória fazendo o seu mapeamento dentro de um vetor de pontos na forma $(x, y)$. Seus principais métodos são:

- *public BaseFourier(String nomeArquivo)* — Constrói o objeto BaseFourier. Carrega uma base de dados de um arquivo (.FOU) para a memória. Em seguida, utiliza os métodos abaixo para mapear os dados do arquivo em um vetor de pontos;
- *public Vector getSequenciaNumeros()* — Retorna o vetor com os pontos já mapeados ou classificados;
- *private Vector classificarVetor(Vector v)* — Classifica os dados lidos do arquivo base para o formato $(x, y)$. Isto resulta em um vetor com todos os pontos que deverão ser transformados pelo algoritmo da *Transformada Quântica de Fourier*;

### 4.3.9 Classe *FourierTask.java*

Esta classe permite um controle sobre a execução do algoritmo da *Trasformada Quântica de Fourier*. Isto é importante para que cada passo do algoritmo possa ser acompanhado pelo usuário. Os seus principais métodos e/ou classes internas são:

- *public FourierTask()* — Cria a tarefa de execução do algoritmo, ou seja, deixa todas as variáveis preparadas;
- *public static void setStatus(double d)* — Seta o *status* da tarefa atual. Este valor representa a porcentagem concluída na execução do algoritmo. É um valor no intervalo $[0, 100]$;
- *public static void setMensagem(String msg)* — Seta uma mensagem a ser exibida ao usuário durante a simulação do algoritmo;
- *public String getMensagem()* — Retorna alguma mensagem previamente escolhida ao usuário;
- *public void setSequencia(Complexos seq[])* — Seleciona a seqüência de números a serem transformadas pelo algoritmo;
- *public Complexos[] getSequencia()* — Retorna a seqüência de números sendo transformadas pelo algoritmo;

- *public void go()* — Efetivamente executa o algoritmo;
- *public int getTamanhoTarefa()* — Retorna o tamanho desta tarefa. Este é um valor no intervalo $[0, 100]$;
- *public int getStatus()* — Retorna qual a porcentagem de execução do algoritmo já foi concluída;
- *public void stop()* — Termina a execução do algoritmo;
- *public void pause()* — Pause a execução do algoritmo;
- *public void resume()* — Dado que o algoritmo esteja pausado este método permite sua reativação;
- *public boolean done()* — Verifica se a tarefa atual já foi concluída;
- *private class ActualTask* — Permite criar tarefas específicas em relação ao algoritmo a ser executado;

### 4.3.10 Classe *Fourier.java*

Esta é a classe que permite a execução do algoritmo da *Transformada Quântica de Fourier*. Esta classe utiliza uma classe de utilitários gerais, *Operações.java*. Esta classe possui muitos algoritmos e funções úteis para todos os algoritmos implementados. Consulte sua especificação caso tenha alguma dúvida. Os principais métodos da classe *Fourier.java* são:

- *public Fourier(Complexos seq[])* — Constrói um objeto *Fourier* que irá permitir a execução do algoritmo. Isto é feito a partir de uma seqüência de números fornecida;
- *private boolean executarAlgoritmo()* — Dá a ordem à *Java Virtual Machine* para que o algoritmo seja realmente executado;
- *public Complexos[] getResultados()* — Retorna o resultado final da execução do algoritmo;
- *public void run()* — Dado que o objeto *Fourier* já tenha sido construído este método permite que a execução do algoritmo seja feita em uma *Thread*. Isto é importante para que outras tarefas no simulador possam ser executadas em parelelo à execução deste algoritmo;

### 4.3.11 Classe *ShorTask.java*

Esta classe permite um controle sobre a execução do algoritmo da *Fatoração de Shor*. Isto é importante para que cada passo do algoritmo possa ser acompanhado pelo usuário. Os seus principais métodos e/ou classes internas são:

- *public ShorTask()* — Cria a tarefa de execução do algoritmo, ou seja, deixa todas as variáveis preparadas;
- *public static void setEstadosObjeto(Object obj, String acao, int id)* — Este algoritmo trabalha com vários objetos tais como registradores, vetores de números .... Este método permite setar uma determinada ação para um dado objeto passado;
- *public String getEstadosObjeto(int id)* — Retorna o estado atual do objeto especificado por parâmentro. Ex.: se o identificador *id* for 1 será retornado o estado atual do registrador quântico de número 1;
- *public static void setStatus(double d)* — Seta o *status* da tarefa atual. Este valor representa a porcentagem concluída na execução do algoritmo. É um valor no intervalo $[0, 100]$;
- *public static void setMensagem(String msg)* — Seta uma mensagem a ser exibida ao usuário durante a simulação do algoritmo;
- *public String getMensagem()* — Retorna alguma mensagem previamente escolhida ao usuário;
- *public void go()* — Efetivamente executa o algoritmo;
- *public int getTamanhoTarefa()* — Retorna o tamanho desta tarefa. Este é um valor no intervalo $[0, 100]$;
- *public int getStatus()* — Retorna qual a porcentagem de execução do algoritmo já foi concluída;
- *public void stop()* — Termina a execução do algoritmo;
- *public void pause()* — Pause a execução do algoritmo;
- *public void resume()* — Dado que o algoritmo esteja pausado este método permite sua reativação;
- *public boolean done()* — Verifica se a tarefa atual já foi concluída;
- *private class ActualTask* — Permite criar tarefas específicas em relação ao algoritmo a ser executado;

### 4.3.12 Classe *Shor.java*

Este é o algoritmo da *Fatoração de Shor*, propriamente dito. É responsável por realizar todas as tarefas para se conseguir fatorar quanticamente um número. Seus principais métodos são:

- *public Shor(int num)* — Responsável por alocar todas as variáveis necessárias para a execução do algoritmo;
- *private boolean executarAlgoritmo()* — Executa o algoritmo;
- *public void run()* — Permite a execução do algoritmo em uma *Thread*. Isto é importante dado que este algoritmo é pesado. Com este recurso pode-se fazer outras tarefas no simulador enquanto um número está sendo fatorado, por exemplo;

### 4.3.13 Classe *SwingWorker.java*

Desenvolvida pela *Sun Microsystems*, esta classe permite que tarefas em uma interface gráfica sejam executadas em paralelo. Com isso, no SAQua, é possível simular mais de um algoritmo ao mesmo tempo. Mais detalhes podem ser encontrados em `http://java.sun.com/docs/books/tutorial/uiswing/misc/threads.html`
Os principais métodos desta classe são:

- *protected synchronized Object getValue()* — Retorna o valor da *thread* atual ou *null* caso nenhuma *thread* tenha sido construída ainda;
- *private synchronized void setValue(Object x)* — Especifica um valor para a *thread* atual;
- *public abstract Object construct()* — Cria um objeto *thread*;
- *public void finished()* — Método chamado quando uma *thread* tenha concluído seu trabalho;
- *public void interrupt()* — Interrompe a *thread* atual para executar um outro método mais importante;
- *public Object get()* — Retorna a *thread* atual caso já tenha sido construída;
- *public SwingWorker()* — Cria uma *thread* para alguma tarefa específica;
- *public void start()* — Inicia uma *thread* para alguma tarefa específica;
- *private static class ThreadVar* — Esta classe mantém uma referência à *thread* sendo executada na interface gráfica;

### 4.3.14 Classe *HandlerBotoes.java*

Classe tratadora dos eventos realizados pelo usuário sobre alguns *frames* do sistema. Seus principais métodos são:

- *HandlerBotoes(Component cpt, FrameShor ca)* — Cria um tratador de eventos para os botões presentes no *frame* de simulação do algoritmo da *Fatoração de Shor*;
- *HandlerBotoes(Component cpt, FrameSobre ca)* — Cria um tratador de eventos para os botões presentes no *frame* sobre os desenvolvedores do sistema;
- *public void actionPerformed(ActionEvent e)* — Responsável por capturar qualquer ação do usuário nos dois *frames* acima;

### 4.3.15 Classe *HandlerMenu.java*

Classe tratadora dos eventos realizados pelo usuário sobre o *menu* principal do SAQua. Seus principais métodos são:

- *HandlerMenu(Component cpt, SAQua ref)* — Cria um tratador de eventos para o *menu* presente na interface principal do SAQua;
- *public void actionPerformed(ActionEvent e)* — Responsável por capturar qualquer ação do usuário sobre o *menu* principal do sistema;

### 4.3.16 Classe *Operacoes.java*

Esta é uma classe de utilitários matemáticos para os algoritmos implementados. A operação de *Transformada Quântica de Fourier* foi especificada aqui pois é uma operação fundamental de grande parte dos algoritmos quânticos. Seus principais métodos são:

- *public boolean testarPrimalidade(int n)* — Esta função pega um inteiro de entrada e retorna 1 se este é um primo e 0, caso contrário;
- *public boolean testarPotenciaPrimos(int n)* — Esta função pega uma entrada inteira e retorna 1 se esta é uma potência de primos e 0, caso contrário;
- *public boolean testarPotenciaDois(int n)* — Esta função verifica se um dado número é uma potência de 2;
- *public int MDC(int a, int b)* — Computa o máximo divisor comum (MDC) entre dois inteiros. Como o módulo $0 \pmod 0$ de um número não é definido retorna-se $-1$ como um código de erro em qualquer tentativa deste tipo;

- *public int tamanhoRegistrador(int a)* — Esta função toma um argumento inteiro e retorna o número de *bits* necessários para sua representacao quântica;
- *public int GetQ(int n)* — Esta função retorna o valor $m$ que satisfaz a expressão $M^2 \leq 2^m < 2M^2$;
- *public int modexp(int x, int a, int n)* — Esta função toma três inteiros, $[x, a, n]$ e retorna $x^a \pmod n$. Este algoritmo é conhecido como: *Russian Peasant Method*;
- *public int denominador(double c, int qmax)* — Esta função acha um denominador $q$ do melhor número racional a partir da aproximação $\frac{p}{q}$ para $(c \mid q < qmax)$;
- *int maior(int a, int b)* — Retorna o maior entre dois números;
- *void transformacaoFourier(Registradores reg, int q)* — Computa a *Transformação Quântica de Fourier*. Faz o mapeamento de $q - 1$ entradas;

### 4.3.17 Classe *Registradores.java*

Esta classe representa formalmente um registrador quântico. Cada posição neste registrador é representada por um quBit. Os seus principais métodos são:

- *public Registradores()* — Constrói um registrador quântico sem nenhuma posição. Isto é útil para referenciar um registrador dado que ainda não existe um registrador real propriamente dito;
- *public Registradores(int nroPosicoes)* — Constrói um registrador quântico com o número de posições passadas. Cada posição será ocupada por um quBit;
- *public Complexos[] getEstados()* — Retorna os números que representam os quBits neste registrador quântico;
- *public void copiarRegistrador(Registradores antigo)* — Copia um registrador quântico para outro;
- *public Complexos getProbabilidade(int idEstado)* — Retorna a probabilidade de um dado estado. É usado, principalmente, na *Transformação Quântica de Fourier*. É interessante observar que num computador quântico real esta operação não seria possível devido à decoerência quântica. Um possível saída seria simplesmente construir um circuito quântico especializado em transformação de e então pegar apenas o resultado útil e não os cálculos intermediários. Contudo, isto foge ao escopo deste trabalho;
- *public void normalizar()* — Normaliza a amplitude de probabilidade, i.e, garante que a soma dos quadrados de todos os reais e imaginários seja sempre 1;

- *public int tamanho()* — Retorna o número de quBits neste registrador quântico;
- *public int medicao()* — Faz a medição de um estado e retorna o valor *decimal* medido. Além disso, faz uma espécie de *colapso* no estado quântico assim que a medição é feita. Este colapso é representado por setar as componentes reais dos quBits para 1 e suas componentes imaginárias para zero;
- *public void espiar(boolean chave)* — Imprime as informações a respeito do registrador. Não existe realização física deste método dado que não é possível observar um estado quântico sem colapsá-lo. Deste modo, este método serve apenas para efeito didático;
- *public void setEstado(Complexos nEstado[])* — Seta o estado do registrador a partir de um vetor de estados;
- *public void setSequencia(Complexos nEstado[])* — Dada uma seqüência de estados seta esta seqüência dentro do registrador quântico;
- *public void setMedia(int n)* — Seta os estados para uma superposição igual de inteiros. Nesta abordagem o coeficiente $beta$ dos quBits será sempre zero;

### 4.3.18 Algumas telas do SAQua


Figura 4.2: Tela inicial do SAQua


Figura 4.3: Os desenvolvedores do simulador

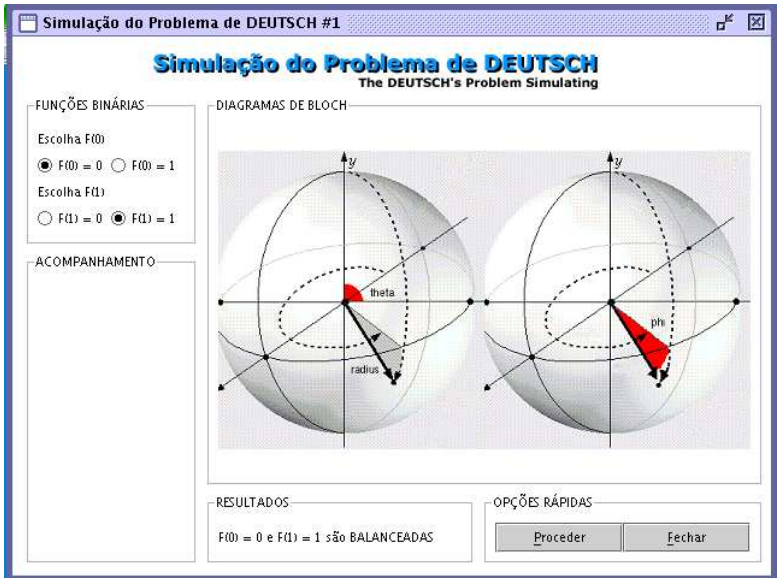


Figura 4.4: O *Problema de Deutsch*

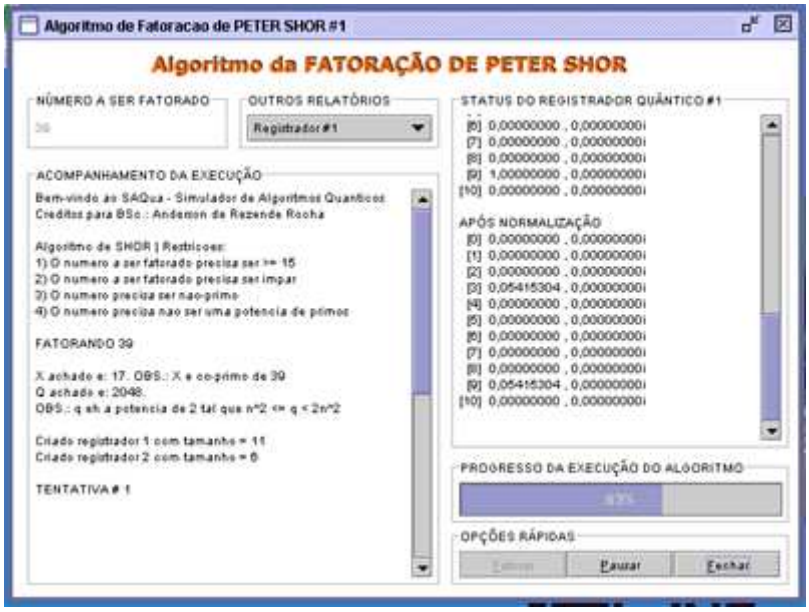


Figura 4.5: O algoritmo da *Fatoração de Shor*

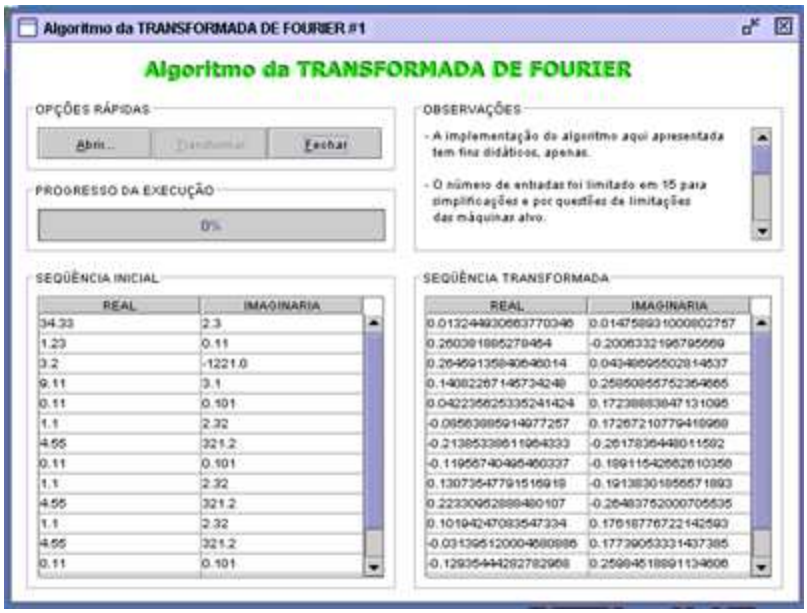


Figura 4.6: O algoritmo da *Transformada Quântica de Fourier*

### 4.3.19 Resultados

Foram realizados vários testes em diversas máquinas diferentes. Foi possível acompanhar a execução de cada algoritmo conforme era esperado. Nenhum dos algoritmos apresentou problemas ao ser simulado. Vale ressaltar que devido à sua complexidade o algoritmo da *Fatoração de Shor* não fatora números maiores de 170. Caso o usuário forneça número acima deste limite a execução do algoritmo ficará muito lenta devido ao enorme esforço computacional. Em relação ao algoritmo da *Transformada Quântica de Fourier* não se deve fornecer um conjunto de pontos acima de 100. Caso isto aconteça o algoritmo também terá sua execução muito carregada. Quanto à simulação do *Problema de Deutsch* não há nenhuma restrição, dado que este problema é muito simples de ser simulado.

# Capítulo 5

# Conclusões

A computação quântica é um novo e emergente campo de pesquisas. Ela tem mudado drasticamente a maneira pela qual se pensa a computação de forma geral, bem como a programação e a complexidade de seus algoritmos.

O desafio dos cientistas da computação e outros é desenvolver novas técnicas de programação apropriadas para os computadores quânticos. O *emaranhamento* quântico e o *cancelamento de fases* introduzem uma nova dimensão para a computação. Programar agora não consiste meramente em formular algoritmos passo-a-passo, mas sim, em novas técnicas para ajustes de fases, misturas e difusão de amplitudes, dentre outras, para se conseguir uma saída útil.

Neste trabalho, foi apresentado o estado da arte da computação quântica: algumas formas de implementação de computadores quânticos, suas vantagens e desvantagens entre outras coisas. Foi apresentado, também, o **SAQua** (Simulador de Algoritmos Quânticos) desenvolvido para simular alguns algoritmos quânticos já escritos por pesquisadores da área.

Obviamente, o material apresentado aqui é apenas uma ínfima parte do que está sendo estudado neste novo e instigante campo de pesquisas. Espera-se que os computadores quânticos realmente deixem a prancheta de projetos e cheguem ao mercado. Contudo, sabe-se que isso ainda é um objetivo um pouco distante. Desta forma, deve-se estar ciente das descobertas, pois, em um campo de pesquisas aberto como esse,

*É muito difícil fazer previsões, ainda mais sobre o futuro.* (Niels Bohr)

# Referências Bibliográficas


[Estadao, 2000] AGÊNCIA ESTADO. *Vem aí o computador quântico*. Artigo publicado em 16 de agosto de 2000. In: *O Estado de São Paulo*, www.estadao.com.br

[Int Media, 2002] INT MEDIA GROUP INCORPORATED. *Moores Law*. Webopedia.com, 25 de abril de 2002, http://www.webopedia.com/TERM/M/Moores_Law.html

[Kaku, 1998] KAKU, Michio. *Visions, How Science Will Revolutionize the* $21_{st}$ *Century*. New York, September, 1998.

[Lenstra e Lenstra, 1993] LENSTRA, A. e Lenstra, H. *The development of the number field sieve*. Vol. 1554 of *Lecture Notes in Mathematics*. Springer Verlag, 1993.

[Nielsen e Chuang, 2000] NIELSEN, Michel A. e Chuang, Isaac L. *Quantum Computation and Quantum Information*. Cambridge, UK, Cambridge University Press, 2000.

[NYT, 2002] THE NEW YORK TIMES. *Gauging the Limits of Quantum Computing*. 19 de abril de 2002, http://www.nytimes.com/library/national/science/030700sci-quantum-computing.html

[Preskill, 2002] PRESKILL, John. *Notas de aula do curso de computação quântica*. In www.theory.caltech.edu/people/preskill/ph219

[Rieffel e Polak, 2000] RIEFFEL, Eleanor e Wolfgang. *An introduction to quantum computing for non-physicists*. In: arXiv:quant-ph/9809016 vol. 2. 19 de janeiro de 2000.

[Simon, 1997] SIMON, D. R. *On the power of quantum computation*. Society for Industrial and Applied Mathematics Journal on Computing 26, 5.

# Apêndice A

# Problemas na simulação

Durante o desenvolvimento do simulador vários problemas apareceram. Os principais problemas foram:

1. *Permitir a execução do simulador a partir de qualquer máquina* — Como solução o simulador foi desenvolvido de forma a poder ser executado tanto quanto um aplicativo de *desktop* quanto um *applet*. Desta forma, este pode ser executado através da *internet* por qualquer *browser* que suporte Java;

2. *Como representar a saída da simulação do Problema de Deutsch* — Como solução utilizou-se a representação quântica das saídas através das esferas de *Bloch*;

3. *Como representar a saída da simulação da Fatoração de Shor* — Como solução criou-se um *Timer* que acompanhava a execução do algoritmo passo-a-passo, exibindo um relatório contante na tela de simulação;

4. *Como obter a base de simulação da Transformada Quântica de Fourier* — Como solução padronizou-se o formato do arquivo de entrada em uma extensão conhecida por (*.FOU). Desta forma qualquer arquivo texto com esta extensão e que tenha números na forma $(x, y)$ pode ser considerado uma entrada para o algoritmo;

5. *Alguns algoritmos implementados estavam lentos* — Para aumentar a velocidade dos algoritmos, *Fatoração de Shor* e *Transformada Quântica de Fourier*, foram tomadas algumas providências:

    - Armazenamento dos cálculos produzidos. Deste modo, ao se fatorar o número 21, por exemplo, os principais cálculos são armazenados para referências futuras. Os valores armazenados são descartados quando se fecha o simulador;

- A exibição de alguns passos intermediários está desabilitada. Isto aumentou drasticamente a velocidade de execução;
- Imposição de algumas restrições tais como o limite máximo para fatoração. Números acima de 170, por exemplo, tornam inviável a simulação do algoritmo de Shor;

# Apêndice B

# Andamento e divisão das tarefas

Os seguintes passos foram seguidos para se chegar ao estágio final da pesquisa:

1. *Coleta de material bibliográfico* — Num primeiro momento, foi realizada uma busca por artigos especializados, livros, *sites* na internet para montar-se uma pequena base bibliográfica de pesquisa;
2. *Desenvolvimento do site da pesquisa* — Foi desenvolvido um *site* para o acompanhamento da pesquisa. Tudo o que foi e ainda for desenvolvido, inclusive este relatório, está disponível para acesso público em `www.comp.ufla.br/~undersun/cq/`;
3. *Introdução à mecânica quântica* — Neste ponto, foi realizado uma espécie de introdução à mecânica quântica. O ministrante foi o co-orientador da pesquisa;
4. *Estudo da ligação entre a mecânica quântica e a computação quântica* — Alguns conceitos mais avançados em mecânica quântica foram introduzidos para que se pudesse abstrair as possibilidades de implementação dos computadores quânticos;
5. *Impactos da computação quântica no mundo* — Foram realizados alguns estudos de quais seriam os impactos no mundo atual caso fosse anunciado, repentinamente, a existência de supercomputadores quânticos;
6. *Possibilidades de implementação de computadores quânticos* — Foram estudadas as possibilidades de implementação de computadores quânticos na atualidade. Foram selecionados três maneiras expostas no item 4.1;
7. *Vantagens do computador quântico* — Foram estudadas as vantagens dos computadores quânticos. Estas se encontram distribuídas ao longo deste relatório;

8. *Estudo teórico de algoritmos quânticos* — Foram estudados alguns algoritmos quânticos. Logo após selecionou-se três: *Problema de Deutsch*, *Transformada Quântica de Fourier* e *Fatoração de Shor* para implementação em computador. Os dados obtidos podem ser verificados em 4.2;

9. *Implementação do simulador quântico* — Foi construído um ambiente num computador comum capaz de simular a execução de alguns algoritmos quânticos. O ambiente desenvolvido pode ser acessado de qualquer lugar do planeta através da internet. Para tal tarefa utilizou-se a linguagem de programação JAVA por ser bastante portável;

10. *Idéias para implementação dos algoritmos selecionados* — Foi procurado, na literatura relacionada, algumas formas de implementar os algoritmos selecionados;

11. *Implementação dos algoritmos* — Os algoritmos selecionados foram implementados no simulador construído para este fim;

12. *Análise dos algoritmos* — Os algoritmos foram analisados quanto ao seu custo computacional. Os resultados podem ser obtidos na seção 4.3;

13. *Possíveis soluções para as perdas de processamento* — Algumas soluções para perdas de desempenho foram apresentadas. Isto pode ser verificado em 4.3 e no apêndice A;

14. *Testes de verificação da implementação* — Esta fase ainda está sendo executada. Estão sendo feitos muitos testes de validação para caça a *bugs* e geração do produto final;

15. *Geração de documentação* — Todo o código do sistema desenvolvido está devidamente comentado e é de livre distribuição. Isto quer dizer que qualquer interessado em modificá-lo está livre para fazê-lo;

16. *Escrita do relatório final a ser entregue ao CNPq* — Esta fase foi iniciada com a elaboração deste relatório final;

# Apêndice C

# Publicações e trabalhos futuros

## C.1 Publicações

Esta pesquisa foi citada pelo *Centro para Computação Quântica* (QuBit.org) da *Universidade de Oxford e Cambridge*. Este reconhecimento é interessante porque, deste modo, a *Universidade Federal de Lavras* passa a ser uma, de três universidades brasileiras, fazendo alguma pesquisa relacionada à computação quântica. Para conferir a citação vá em `www.qubit.org` na seção *World Location*.

Além disso, um artigo científico sobre a pesquisa está sendo escrito para submissão em breve.

## C.2 Trabalhos futuros

A pesquisa relacionada à computação quântica é muito intrigante e promissora. O que apresenta-se nesta trabalho é algo ínfimo quando comparado à imensidão de publicações do gênero.

No simulador SAQua implementado, consegue-se simular 3 (três) algoritmos quânticos. Seria interessante, como proposta de trabalhos futuros:

- Estender o algoritmo de fatoração de Shor para trabalhar com processamento paralelo e distribuído. Neste caso, poder-se-ia colocar cada estado quântico em um computador em uma rede.
- Reescrever certos trechos de código de modo a otimizar a execução dos algoritmos. Isto permitiria, por exemplo, que o algoritmo de Shor fatorasse número maiores que 200.

- Implementar novos algoritmos no simulador. Alguns poderiam ser Busca em banco de dados de Groove ou mesmo o problema de Deutsch estendido para funções n-árias.
- Melhorar a execução do simulador como *applet* de modo que fique mais rápido e dependa menos de políticas *policies* que precisam ser configuradas pelo usuário deixando-o confuso.