JAI CSBC 2019

Centro de Convenções da Amazônia | Belém • PA | 14 a 18 de julho de 2019
Computação e Responsabildiade Socioambiental

39º Congresso da Sociedade Brasileira de Computação
38ª Jornada de Atualização em Informática (JAI)

Realização | Organização

Belém, 2019

JAI CSBC 2019

Centro de Convenções da Amazônia | Belém • PA | 14 a 18 de julho de 2019
Computação e Responsabildiade Socioambiental

38ª Jornada de Atualização em Informática (JAI)

EDITORA
Sociedade Brasileira de Computação (SBC)

COORDENAÇÃO GERAL
Denis Rosário (UFPA)
Marcelle Mota (UFPA)

COMISSÃO ORGANIZADORA
Marcelino Silva da Silva (UFPA)
Filipe Saraiva (UFPA)
Marcos Seruffo (UFPA)
Eduardo Cerqueira (UFPA)
Carlos Gustavo Resque (UFPA)
Dionne Cavalcante Monteiro (UFPA)
Paulo Henrique Bezerra (IFPA)

COORDENAÇÃO DA JAI
Soraia Musse (PUCRS)
Fabio Kon (USP)

APOIO LOCAL
Gustavo Pinto (UFPA)

REALIZAÇÃO
Sociedade Brasileira de Computação (SBC)

ORGANIZAÇÃO
Universidade Federal do Pará (UFPA)

Realização | Organização

SBC Sociedade Brasileira de Computação

Belém, 2019

Mensagem da Coordenação Geral do XXXIX Congresso da Sociedade Brasileira de Computação

Sejam bem-vindos ao XXXIX Congresso da Sociedade Brasileira de Computação (CSBC 2019), realizado na acolhedora cidade das mangueiras - Belém/Pará, de 14 a 18 de Julho de 2019. Organizar uma edição do maior evento acadêmico de Computação da América Latina pela segunda vez no Norte do Brasil é um desafio e um privilégio por podermos contribuir com a comunidade de Computação do Brasil e do exterior, pois o CSBC se destaca como um importante ambiente para a discussão, troca de conhecimento e apresentação de trabalhos científicos de qualidade. O CSBC 2019 teve como tema central: "Computação e Responsabilidade Socioambiental". Tendo em vista que Belém, a cidade sede do evento, é porta de entrada para a Amazônia, maior floresta tropical do mundo, é fundamental envolver questões ambientais nos debates científicos.

Estes anais registram os trabalhos apresentados durante o CSBC 2019, que teve 10 eventos base, 12 eventos satélites, o selo de inovação, e 5 reuniões e a Assembleia geral da Sociedade Brasileira de Computação (SBC). Desta forma, a programação do CSBC 2019 contou com mais de 250 trabalhos científicos em diversas áreas da computação e discutiu temas relevantes no cenário nacional e internacional. A contribuição da comunidade científica brasileira foi de fundamental importância para manter a qualidade técnica dos trabalhos e fortalecer a ciência, tecnologia e inovação no Brasil. Além disso, o CSBC 2019 teve várias palestras e mesas-redondas com pesquisadores renomados de diversos setores da computação. O Congresso ainda abrigou a reunião do Fórum de Pós-Graduação em Ciência da Computação, a reunião do CNPq/CAPES, a reunião dos Secretários Regionais SBC, a reunião das Comissões Especiais e a reunião do Fórum IFIP/SBC.

O CSBC 2019 teve como objetivo proporcionar o intercâmbio de conhecimento na região e a integração entre pesquisadores, alunos, professores e profissionais da área de Computação de todo o Brasil. A realização do CSBC contou com o importante apoio do Comitê Gestor da Internet no Brasil (CGI.br), do CNPq, da CAPES, da FAPESPA e da PRODEPA. Ainda, gostaríamos de fazer um especial agradecimento à Universidade Federal do Pará (UFPA), à Universidade Federal do Sul e Sudeste do Pará (UNIFESPA), à Universidade Federal Rural da Amazônia (UFRA) e ao Instituto Federal do Pará (IFPA) pelo indispensável suporte à realização do evento. O sucesso do CSBC 2019 só foi possível devido à dedicação e entusiasmo de muitas pessoas, especialmente a comissão organizadora, aos coordenadores e apoio local dos 22 eventos, aos organizadores das 5 reuniões, a organização do selo de inovação e aos autores pelo envio de mais de 800 artigos científicos. Por fim, gostaríamos de expressar nossa gratidão ao Comitê Organizador, por sua grande ajuda em dar forma ao evento; e, em especial, à equipe da Sociedade Brasileira de Computação (SBC), por todo apoio.

Denis Lima do Rosário
Marcelle Pereira Mota
Coordenadores do CSBC 2019

Mensagem da Coordenação da XXXVIII Jornada de Atualização em Informática

Sejam bem-vindos à 38º Jornada de Atualização em Informática (JAI)! A JAI é um dos mais importantes eventos acadêmicos de atualização científica e tecnológica da comunidade de Computação do Brasil. Tradicionalmente realizada em conjunto com o Congresso da SBC, a JAI compreende trabalhos de pesquisadores sêniores da nossa comunidade, oferecendo uma oportunidade única para estudantes e profissionais de informática atualizarem-se em temas diversos, interagindo com líderes das mais diversas áreas de pesquisa no Brasil.

Em 2019, a JAI ocorrerá em Belém de 14 a 18 de julho. A JAI será formada por quatro mini-cursos em áreas avançadas do conhecimento em Computação. São elas: 1) Introdução à Programação de Computadores Quânticos; 2) Como programar aplicações de alto desempenho com produtividade; 3) Correntes de Blocos (Blockchain): Algoritmos de Consenso e Implementação na Plataforma Hyperledger Fabric e 4) Autenticação usando Sinais Biométricos: Fundamentos, Aplicações e Desafios. Este ano, recebemos 9 excelentes propostas de curso de JAI, sendo que após revisão do comitê, 4 foram escolhidas. Os critérios levaram em consideração os comentários dos revisores, a senioridade dos palestrantes, competência demonstrada na sua área de atuação, importância e abrangência do tema proposto e potencial para despertar o interesse de jovens estudantes, acadêmicos e profissionais da área.

A realização do evento contou com o importante apoio dos revisores que gentilmente avaliaram as propostas, bem como os capítulos de livro que serão disponibilizados para a comunidade. Agradecemos também a organização local da CSBC 2019 e ao apoio sempre presente da SBC.

Desejamos a todos um excelente evento com vários desdobramentos científicos!

Nos vemos em Belém!

Soraia Raupp Musse
Fabio Kon
Coordenadores do JAI 2019

Comitê de programa do JAI 2019

Andreia Formico (UNIFOR), Bruno Feijó (PUC-Rio),
Carla Negri Lintzmayer (UFABC), Cláudio Rosito Jung (UFRGS),
Ellen Francine Barbosa (ICMC-USP), Esteban Clua (UFF),
Francisco José da Silva e Silva (UFMA), Jussara Almeida (UFMG),
Luciano Barbosa (UFPE), Luciano Pereira Soares (INSPER),
Luiz Gustavo Fernandes (PUCRS), Rafael Bordini (PUC-RS),
Raphael Camargo (UFABC), Raquel Oliveira Prates (UFMG).

Editores

Denis Rosário (UFPA)

Fabio Kon (USP)

Marcelle Mota (UFPA)

Marcelino Silva (UFPA)

Soraia Musse (PUCRS)

C736j Sociedade Brasileira de Computação, Congresso, 2019

38º Jornada de Atualização em Informática (JAI), 1ª Edição, Editora – Sociedade Brasileira de Computação - SBC

Organizadores: Fabio Kon (USP) e Soraia Musse (PUCRS)

Belém-PA, 2019.

ISBN: 978-85-7669-471-7

1. Computação – Congresso. 2. Computação e Responsabilidade Socioambiental

CDD: 004

Sumário

Capítulo 1

Introdução à Programação de Computadores Quânticos

Renato Portugal, Franklin L. Marquezino

Resumo

A computação quântica é um novo paradigma computacional que explora o comportamento quântico para realizar um processamento mais eficiente do que o processamento baseado na lógica binária clássica. Este tutorial apresenta as bases da computação quântica de forma didática com foco no modelo de circuitos quânticos, usa o algoritmo de Grover como exemplo, descreve como implementar algoritmos quânticos e como programar computadores quânticos da IBM usando a interface gráfica IBM Q Experience e o kit de programação quântica Qiskit. Como pré-requisito, é necessário que o leitor tenha conhecimento básico de Álgebra Linear e Python.

Abstract

Quantum computation is a new computational paradigm that exploits the quantum behavior in order to perform a kind of processing that is more efficient than the processing based on classical binary logic. This tutorial presents the foundations of quantum computation in a didactic way with a focus on the quantum circuit model, uses the Grover algorithm as an example, describes how to implement quantum algorithms, and how to program IBM quantum computers using the graphical interface IBM Q Experience and the quantum kit Qiskit. As a prerequisite, it is necessary that the reader has basic knowledge of linear algebra and Python.

1.1. A base da computação clássica

A máquina de Turing é uma descrição abstrata de um modelo computacional útil para determinar o que um computador pode calcular. A máquina tem um fita infinita dividida em células onde podemos gravar 0, 1 ou deixar em branco. Ela tem um cabeçote que pode ler uma célula, mudar o valor e depois se mover para uma das células vizinhas obedecendo a

um programa. Este programa consiste de instruções básicas que determinam o movimento do cabeçote e a escrita na fita condicionado ao que foi lido nas últimas células visitadas e ao estado da máquina. O número de estados possíveis deve ser finito. O programa é executado recursivamente até que um comando de parada condicional é encontrado. Se a máquina de Turing não para, não há um resultado da computação. Existem problemas computacionais para os quais não é possível fazer uma máquina de Turing que pare em um tempo finito, como por exemplo o famoso problema da parada [11, 35, 39].

O modelo de circuitos lógicos se baseia em *portas lógicas*, que são dispositivos que implementam funções booleanas básicas, como AND, OR e NOT. Este modelo tem diversas vantagens do ponto de vista de implementação prática e desenvolvimento de computadores e, em especial, na análise do processamento computacional e cálculo da eficiência de um algoritmo. Um circuito lógico de n bits de entrada e 1 bit de saída é equivalente a uma tabela verdade de 2^n linhas. Uma vez que cada linha pode ter duas saídas (0 ou 1), um chip de n bits de um computador clássico pode calcular 2^{2^n} funções booleanas diferentes. Combinando funções booleanas, podemos construir algoritmos com saídas de mais de 1 bit. Uma função booleana genérica de n variáveis pode ser colocada na forma normal disjuntiva (FND) e pode ser convertida em um circuito lógico usando as portas AND, OR e NOT. Uma vez que esta decomposição é útil no contexto da implementação de algoritmos quânticos, vamos dar um exemplo de como obter a FND de uma tabela verdade específica. Fica evidente como se obtém o caso geral.

Seja $f(a,b,c)$ uma função booleana de 3 bits definida pela seguinte tabela verdade:

a	b	c	$f(a,b,c)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Para obter a FND, vamos focar nas saídas iguais a 1. Como há 2 saídas iguais a 1, vamos escrever uma expressão booleana que é disjunção de 2 cláusulas conjuntivas com todas as variáveis, isto é,

$$(a \wedge b \wedge c) \vee (a \wedge b \wedge c).$$

A primeira cláusula se refere a linha de entrada 001 e saída 1. Para que a cláusula tenha saída 1, devemos negar as variáveis a e b, da seguinte forma, $(\bar{a} \wedge \bar{b} \wedge c)$. A segunda cláusula se refere a linha de entrada 110 e saída 1. Para que a cláusula tenha saída 1, devemos negar a variável c, da seguinte forma, $(a \wedge b \wedge \bar{c})$. A FND da tabela acima é

$$f(a,b,c) = (\bar{a} \wedge \bar{b} \wedge c) \vee (a \wedge b \wedge \bar{c}).$$

Como este procedimento pode ser feito para uma tabela verdade genérica, concluímos que $\{\wedge, \vee, \bar{}\}$ é um conjunto universal. O circuito lógico desta expressão booleana está mostrado na figura 1.1.

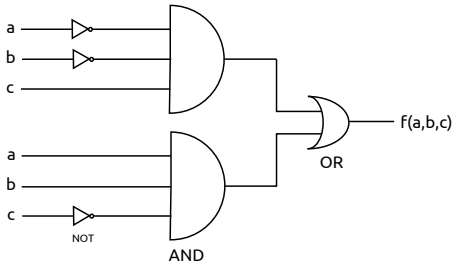


Figura 1.1. O circuito lógico da função booleana $f(a,b,c)$.

A entrada fica à esquerda do circuito e é representada pelos bits a, b e c. À medida que a informação se propaga para a direita, a porta NOT atua em 1 bit convertendo 0 em 1 e vice-versa. Cada porta AND e OR atua em mais de 1 bit gerando uma única saída à direita, reduzindo sistematicamente de n de bits até 1 bit de saída (0 ou 1), que é o valor de $f(a,b,c)$. A redução sistemática do número de bits requer que $(n-1)$ bits de informação sejam apagados e, pelas leis da termodinâmica, este processo gera calor por ser um processo irreversível [5, 21].

Entre algumas referências importantes no contexto do modelo de circuitos está a dissertação de mestrado de Claude Shannon [33] e as referências [7, 8, 40].

Nas próximas seções, entramos no contexto quântico. Para adiantar o tema um pouco e frisar algumas diferenças, a entrada e a saída de um circuito quântico têm o mesmo tamanho e o processamento é reversível. Para obter um circuito quântico que implementa uma tabela verdade, devemos usar portas lógicas reversíveis [38]. Como veremos adiante, esta tarefa pode ser realizada através da *porta Toffoli generalizada*.

1.2. Origens e novidades da computação quântica

Um momento marcante no início da computação quântica foi a palestra *Simulating Physics with Computers* de Richard Feynman na conferência *First Conference on the Physics of Computation* no MIT em 1981, quando ele incentivou os pesquisadores a buscarem um computador que usasse os princípios da mecânica quântica para realização de cálculos, em vez da lógica binária clássica. O principal argumento de Feynman se baseou no fato de que a simulação clássica de sistemas quânticos cresce exponencialmente em função do tamanho do sistema. A questão colocada foi que um computador cujo processamento e memória exploram os fenômenos quânticos poderia simular eficientemente sistemas quânticos. Ficou claro que existia uma difícil tarefa a ser realizada, pois todo o edifício da computação clássica teria que ser refeito a partir de uma nova base.

O primeiro andar ficou pronto em 1985 quando David Deutsch da Universidade de Oxford descreveu formalmente a primeira máquina de Turing universal quântica, que pode simular qualquer outro computador quântico [13]. Sabemos que a máquina de Turing clássica não é adequada para a implementação e o desenvolvimento de algoritmos complexos. O modelo de circuitos e portas lógicas é mais adequado e serve de orientação no momento do desenvolvimento de chips dos processadores. Esta observação também é válida no contexto quântico. O modelo de circuitos e portas lógicas quânticas foi desenvolvido a partir de 1985 [14] e culminou no aparecimento do algoritmo de Shor para

fatoração de números inteiros em tempo polinomial em 1994 [27, 30, 34].

A construção de computadores quânticos enfrenta enormes problemas de engenharia para evitar erros quando o número de qubits aumenta [1]. Diversos laboratórios de mecânica quântica construíram computadores quânticos universais com poucos qubits. No entanto, somente a partir de 2017 e 2018, empresas como a IBM, Google e Microsoft anunciaram a construção de computadores quânticos universais com dezenas de qubits. Em 2019, a IBM anunciou um computador quântico comercial de 20 qubits para ser usado em nuvem. Infelizmente, ainda não está claro se os erros estão sob controle. A curto prazo acredita-se que computadores universais baseados em portas lógicas com poucas centenas de qubits com ruídos estarão disponíveis [31]. Entre as principais aplicações no escopo destas máquinas podemos citar a simulação de sistemas quânticos, a simulação de química quântica, a solução de alguns problemas de otimização, a programação linear, a aprendizagem profunda quântica e a teoria de jogos quânticos [31]. Por exemplo, para simular o comportamento de uma molécula de cafeína, o computador clássico deve ter da ordem de 10^{48} bits e o computador quântico da ordem de 160 qubits[1]. Infelizmente, a área de algoritmos quânticos requer uma quantidade maior de qubits lógicos estáveis e com pouco ruído (mais de 1000 qubits já com correção de erros e tolerância a falhas). Isto deve demorar várias décadas para ser atingido. Por outro lado, as primeiras demonstrações de supremacia quântica devem ocorrer em breve [9].

1.3. Qubit, portas lógicas e circuitos quânticos

Antes de definir o conceito de *qubit* (*quantum bit*), vamos fazer uma breve revisão de álgebra linear. Existem diversas notações para mostrar que uma variável v é um vetor, por exemplo, $\vec{v}$. Na computação quântica, a notação mais usada é a *notação de Dirac*: $|v\rangle$. Uma sequência de vetores é denotada por $|v_0\rangle$, $|v_1\rangle$, $|v_2\rangle$ e assim por diante. Na computação quântica, é muito frequente abusar desta notação e denotar a mesma sequência por $|0\rangle$, $|1\rangle$, $|2\rangle$ e assim por diante. Referências adicionais que podem ser usadas para esta seção são [32] e [43].

1.3.1. Breve revisão de álgebra linear na notação de Dirac

A base de um espaço vetorial de duas dimensões é constituída de dois vetores. Na notação de Dirac, a *base canônica* é denotada por $\{|0\rangle, |1\rangle\}$, onde $|0\rangle$ e $|1\rangle$ são vetores bidimensionais ortogonais, isto é, eles têm duas entradas ou componentes e o produto interno entre $|0\rangle$ e $|1\rangle$ é 0. Estes vetores são dados por

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{e} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Na álgebra linear, esta base é chamada de *base canônica*. Na computação quântica, ela é denominada *base computacional*.

Note que $|0\rangle$ não é o vetor nulo, mas sim o primeiro vetor da base canônica. O vetor nulo é definido pelo vetor com todas as entradas nulas. No caso bidimensional ele é

[1] Veja a apresentação do prof. Ulisses Mello da IBM no Youtube.

dado por

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

sem nenhuma denominação especial na notação de Dirac.

Um vetor genérico do espaço vetorial bidimensional é obtido através de uma *combinação linear* dos vetores da base e denotado por

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

onde α e β são números complexos. Estes números são as entradas do vetor $|\psi\rangle$, como pode ser visto pela notação matricial

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

O *vetor dual* ao vetor $|\psi\rangle$ é denotado por $\langle\psi|$ e é obtido transpondo o vetor $|\psi\rangle$ e conjugando cada entrada. Usando a equação anterior, obtemos

$$\langle\psi| = \begin{pmatrix} \alpha^* & \beta^* \end{pmatrix},$$

que pode ser escrito como

$$\langle\psi| = \alpha^*\langle 0| + \beta^*\langle 1|,$$

onde

$$\langle 0| = \begin{pmatrix} 1 & 0 \end{pmatrix} \text{ e } \langle 1| = \begin{pmatrix} 0 & 1 \end{pmatrix}.$$

Podemos ver o vetor dual $\langle\psi|$ como uma matriz 1×2 e o vetor $|\psi\rangle$ como uma matriz 2×1. Suponha que $|\psi_1\rangle$ e $|\psi_2\rangle$ sejam dois vetores bidimensionais dados por

$$|\psi_1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \text{ e } |\psi_2\rangle = \begin{pmatrix} \gamma \\ \delta \end{pmatrix}.$$

O *produto interno* de $|\psi_1\rangle$ e $|\psi_2\rangle$ é um número complexo denotado por $\langle\psi_1|\psi_2\rangle$ e definido pelo produto matricial do vetor dual $\langle\psi_1|$ pelo vetor $|\psi_2\rangle$, da seguinte forma:

$$\langle\psi_1|\psi_2\rangle = \begin{pmatrix} \alpha^* & \beta^* \end{pmatrix} \begin{pmatrix} \gamma \\ \delta \end{pmatrix} = \alpha^*\gamma + \beta^*\delta.$$

Na notação de Dirac, o cálculo do produto interno é feito da seguinte maneira:

$$\langle\psi_1|\psi_2\rangle = (\alpha^*\langle 0| + \beta^*\langle 1|) \cdot (\gamma|0\rangle + \delta|1\rangle) = \alpha^*\gamma\langle 0|0\rangle + \beta^*\delta\langle 1|1\rangle = \alpha^*\gamma + \beta^*\delta.$$

A *norma* do vetor $|\psi_1\rangle$ é denotada por $\| |\psi_1\rangle \|$ e definida como

$$\| |\psi_1\rangle \| = \sqrt{\langle\psi_1|\psi_1\rangle} = \sqrt{|\alpha|^2 + |\beta|^2},$$

onde $|\alpha|$ é o *valor absoluto* de α, isto é

$$|\alpha| = \sqrt{\alpha \cdot \alpha^*}.$$

Se $\alpha = a + b\,i$, onde i é a unidade imaginária ($i = \sqrt{-1}$), a é a parte real e b a parte imaginária, então

$$|\alpha| = \sqrt{(a+b\,i)\cdot(a-b\,i)} = \sqrt{a^2+b^2}.$$

Em espaços vetoriais reais, o produto interno é chamado de *produto escalar* e é dado por

$$\langle \psi_1 | \psi_2 \rangle = \| \, |\psi_1\rangle \| \, \| \, |\psi_2\rangle \| \cos\theta,$$

onde θ é o ângulo formado pelos vetores $|\psi_1\rangle$ e $|\psi_2\rangle$.

Usando estas definições, podemos mostrar que a base de vetores $\{|0\rangle, |1\rangle\}$ é ortonormal, isto é, os vetores $|0\rangle$ e $|1\rangle$ são ortogonais e têm norma 1, ou seja

$$\langle 0|0\rangle = 1, \quad \langle 0|1\rangle = 0, \quad \langle 1|0\rangle = 0, \quad \langle 1|1\rangle = 1.$$

Uma forma algébrica de denotar ortonormalidade e de compactar as quatro últimas equações em uma única é

$$\langle k|\ell\rangle = \delta_{k\ell},$$

onde k e ℓ são bits e $\delta_{k\ell}$ é chamado de *delta de Kronecker* e definido como

$$\delta_{k\ell} = \begin{cases} 1, \text{ se } k = \ell, \\ 0, \text{ se } k \neq \ell. \end{cases}$$

Para maior aprofundamento na área de álgebra linear, sugerimos as seguintes referências: [2, 37, 22]. Para uma compilação de resultados relevantes para a computação quântica, sugerimos o apêndice A do livro [29] e a seção 2.1 do livro [27].

1.3.2. Qubit

A unidade básica de memória de uma computador clássico é o bit, que pode assumir os valores 0 ou 1. Usualmente, o bit é implementado usando o potencial elétrico, seguindo a convenção de que potencial elétrico nulo ou baixo representa o bit 0 e potencial elétrico alto representa o bit 1. No final de um cálculo, é necessário medir o potencial elétrico para determinar se a saída é o bit 0 ou o bit 1.

A unidade básica de memória de um computador quântico é o qubit. No final da computação, uma medição do qubit retorna o valor 0 ou 1 da mesma forma que o bit clássico. A diferença aparece durante a computação, pois o qubit admite a coexistência simultânea dos valores 0 e 1. Durante a computação, isto é, antes da medição, o *estado* de um qubit é representado por um vetor bidimensional de norma 1 e os estados de um qubit correspondentes aos valores 0 e 1 são $|0\rangle$ e $|1\rangle$. O termo *estado* pode sempre ser identificado com "um vetor de norma 1" e pode ser pensado como o "valor" do qubit antes da medição. A coexistência quântica é representada matematicamente por uma soma de vetores ortogonais da seguinte forma

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

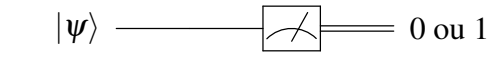
onde α e β são números complexos que obedecem ao vínculo

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

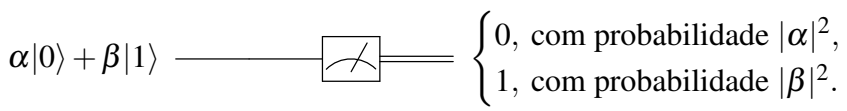
Isto é, o estado do qubit é o vetor $|\psi\rangle$ de norma 1 com as entradas α e β. Os números complexos α e β são chamados de *amplitudes* do estado $|\psi\rangle$.

A coexistência dos bits 0 e 1 não pode ser implementada em um sistema físico clássico, pois não é possível ter potencial elétrico baixo e alto ao mesmo tempo, como todo mundo sabe. Na mecânica quântica, por mais incrível que pareça, é possível ter um sistema quântico (usualmente microscópico) com potencial elétrico baixo e alto ao mesmo tempo. Esta coexistência só pode ser mantida se o sistema quântico estiver isolado do meio ambiente macroscópico ao redor. Quando medimos o sistema quântico para determinar o valor do potencial elétrico, o aparelho de medição inevitavelmente influencia irreversivelmente no valor do potencial elétrico gerando um resultado estocástico, que é potencial elétrico baixo ou alto, similar ao bit clássico. Ou seja, a coexistência só é mantida quando ninguém estiver observando. Note que a mecânica quântica é uma *teoria científica*, isto é, suas leis e resultados podem ser testados de forma objetiva em laboratório. Além disso, leis e afirmações desnecessárias são sumariamente descartadas. Portanto, a afirmação "a coexistência só é mantida quando ninguém estiver observando" tem consequências práticas e é uma afirmação testada e re-testada por mais de 100 anos em milhares de laboratórios de mecânica quântica no mundo. Por outro lado, teorias alternativas mais palatáveis classicamente, que poderiam ajudar na compreensão ou visualização da superposição quântica, foram descartadas por testes experimentais.

Do ponto de vista computacional, podemos ter um qubit em superposição e usar este fato em um circuito. Por exemplo, o circuito



indica que o "valor" inicial do qubit é $|\psi\rangle$ e esta informação foi transmitida inalterada da esquerda para a direita até que uma medição do potencial elétrico foi feita, como indicado pelo *medidor* (o visor de um voltímetro). O resultado da medição é 0 ou 1. A informação clássica é mostrada pela linha dupla. Se o estado do qubit for $|0\rangle$, uma medição resulta necessariamente em 0 e se o estado do qubit for $|1\rangle$, uma medição resulta necessariamente em 1. No caso geral, se o estado do qubit for $\alpha|0\rangle+\beta|1\rangle$, uma medição resulta em 0 com probabilidade $|\alpha|^2$ e em 1 com probabilidade $|\beta|^2$, como mostrado no circuito



A saída pode ser vista com um histograma da distribuição de probabilidades. É importante repetir o fato de que α e β são chamados de *amplitudes* do estado $\alpha|0\rangle+\beta|1\rangle$ e são números que podem ser negativos e ter parte imaginária. Por outro lado, $|\alpha|^2$ e $|\beta|^2$ são números reais positivos no intervalo $[0,1]$ e são chamados de probabilidades. Confundir amplitude com probabilidade gera erros imperdoáveis.

O estado de um qubit pode ser caracterizado por dois ângulos θ e φ da seguinte maneira:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle,$$

onde $0 \leq \theta \leq \pi$ e $0 \leq \varphi < 2\pi$. Esta notação mostra que existe uma correspondência biunívoca entre o conjunto de estados de 1 qubit e pontos na superfície de uma esfera de

raio 1, chamada *esfera de Bloch*. Os ângulos θ e φ são ângulos esféricos que descrevem a posição do estado $|\psi\rangle$, como mostrado na figura 1.2. O ponto na esfera de Bloch é descrito por um vetor tridimensional de entradas reais dado por

$$\begin{pmatrix} \sin\theta\cos\varphi \\ \sin\theta\sin\varphi \\ \cos\theta \end{pmatrix}.$$

A figura 1.2 mostra a posição dos estados $|0\rangle$ e $|1\rangle$, cujos ângulos esféricos são $(\theta,\varphi) = (0,0)$ e $(\theta,\varphi) = (\pi,0)$.

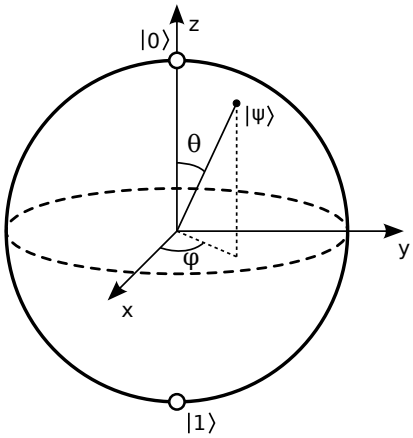


Figura 1.2. Esfera de Bloch (Fonte: Wikipedia)

1.3.3. Portas lógicas quânticas de 1 qubit

Uma porta lógica de 1 qubit é uma matriz quadrada[2] bidimensional unitária. Uma matriz de duas dimensões U é unitária se a multiplicação de U por um vetor de norma 1 arbitrário resulta em um vetor de norma 1. Mais formalmente, seja $|\psi'\rangle = U|\psi\rangle$, onde $|\psi\rangle$ é um vetor bidimensional de norma 1. Se U for uma matriz unitária, então $|\psi'\rangle$ tem norma 1. Por exemplo, a matriz de Hadamard

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

é unitária. Portanto, a multiplicação de H pelos vetores da base tem que dar vetores de norma 1. De fato,

$$H|0\rangle \;=\; H\begin{pmatrix} 1 \\ 0 \end{pmatrix} \;=\; \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \;=\; \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle.$$

Note que o vetor resultante tem norma 1. Vamos denotar este vetor por $|+\rangle$, isto é

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle.$$

[2]É muito comum o uso do termo *operador* no lugar de matriz quadrada.

A multiplicação de H por $|1\rangle$ resulta no vetor $|-\rangle$ definido como

$$|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle,$$

que também tem norma 1. Estes cálculos são importantes para saber a saída da porta H. Se a entrada for $|0\rangle$, a saída será $|+\rangle$. Se a entrada for $|1\rangle$, a saída será $|-\rangle$. Se a entrada for uma superposição de $|0\rangle$ e $|1\rangle$, a saída será a mesma superposição de $|+\rangle$ e $|-\rangle$.

Verificar que a multiplicação de H pelos vetores de uma base ortonormal resulta em vetores de norma 1 não é suficiente para mostrar que H é uma matriz unitária. Além disso, é necessário mostrar que os vetores resultantes são ortogonais, isto é, mostrar que $\langle - | + \rangle = 0$.

Um circuito quântico é uma forma pictórica de descrever um algoritmo quântico. O qubit de entrada fica à direita e a informação flui inalterada da esquerda para a direita até encontrar uma porta lógica. A porta lógica recebe a entrada pela esquerda, processa a informação, e ela sai pela direita e continua seu fluxo. O processamento da porta é feito pela multiplicação da matriz unitária que representa a porta pelo vetor que representa o estado do qubit na entrada. Por exemplo, a expressão $|+\rangle = H|0\rangle$ é representada pelo seguinte circuito:

$$|0\rangle \quad \text{---}\boxed{H}\text{---} \quad |+\rangle.$$

A entrada é o vetor $|0\rangle$, que é transportado inalterado pelo fio até a porta H, que age ou atua na entrada e a transforma em $|+\rangle$, que é transportado até a saída à direita. A ação ou atuação de uma porta é calculada pelo produto matricial da matriz que representa a porta pelo vetor de entrada. Portanto, o resultado da computação é o vetor $|+\rangle$. Se ao final da computação fizermos uma medição, a representação é

$$|0\rangle \quad \text{---}\boxed{H}\text{---}\boxed{}\!\!= \begin{cases} 0, \text{ com probabilidade } \frac{1}{2}, \\ 1, \text{ com probabilidade } \frac{1}{2}. \end{cases}$$

A medição de 1 qubit no estado $|+\rangle$ resulta em 0 com probabilidade 1/2 ou 1 com a mesma probabilidade. A figura 1.3 mostra o histograma da distribuição de probabilidades gerado no Qiskit com duas iterações.

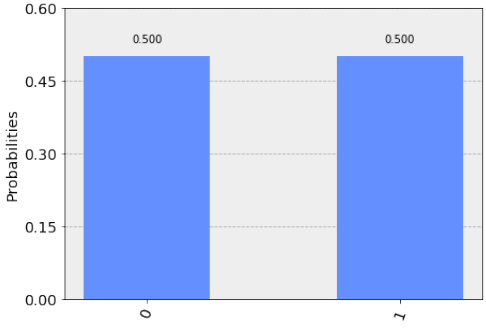


Figura 1.3. Histograma da distribuição de probabilidades da porta H

Neste ponto já podemos usar o simulador de circuitos quânticos (*composer*) do computador quântico IBM Q 5 Tenerife [ibmqx4] e semelhantes disponibilizado pela IBM. Após abrir a página eletrônica da IBM[3], o usuário deve se registrar para usar este simulador. O *composer* do IBM Q 5 é muito simples, pois podemos pegar as portas lógicas disponibilizadas e arrastar para o circuito. Vamos nos restringir a um uso bem básico no momento. A figura 1.4 mostra o circuito da porta H já pronto no *composer*. Este circuito pode ser facilmente feito, pegando a porta H no conjunto de portas à direita e arrastando para o primeiro qubit do circuito. Em seguida devemos arrastar o medidor e soltá-lo depois da porta H. A seta do medidor indica que a saída foi direcionada para um registrador clássico.

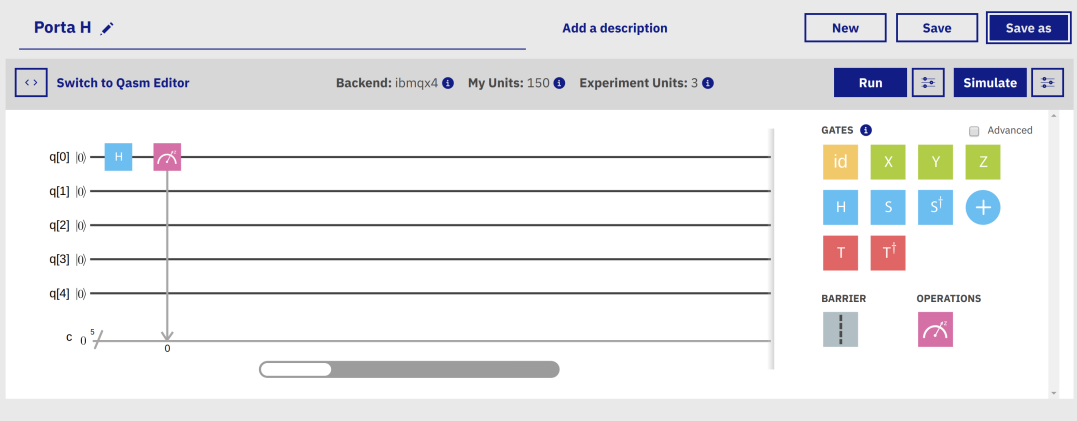


Figura 1.4. Exemplo de circuito usando a porta H e um medidor (Reprint Courtesy of IBM Corporation ©)

Depois do circuito pronto, temos duas opções: (1) rodar o programa no computador quântico clicando em Run ou (2) simular o programa clicando em Simulate. Para que o resultado saia rápido, é melhor usar a segunda opção. A saída da execução está mostrada na figura 1.5. O resultado 00001 foi obtido com probabilidade 0.520 e o resultado 00000 com probabilidade 0.480. Os quatro primeiros bits devem ser descartados, pois se referem aos qubits que não foram usados. A saída usa a ordem dos bits menos significativos à direita, como é usual na computação clássica. Por *default*, o *composer* roda o experimento 100 vezes e mostra o histograma da distribuição de probabilidades. O resultado 00001 foi obtido 52 vezes e 00000 foi obtido 48 vezes. Teoricamente, o resultado mais provável é 50 vezes cada, porém resultados próximos de 50 têm probabilidades não-desprezíveis e ocorrem com frequência. Para obter resultados mais próximos de 0.5, temos que aumentar o número de repetições (*shots*) clicando caixa  no lado direito da caixa Simulate.

Se o usuário optar pela opção Run, a tarefa entra na fila e pode demorar mais de 1 dia. Além disso, o usuário gasta unidades. A saída do computador quântico usualmente é diferente do simulador, pois os erros degradam os resultados. Aumentar o número de repetições não garante que a distribuição de probabilidades tenda para a distribuição correta. É importante usar o simulador antes para testar a corretude do circuito com algumas casas decimais.

[3] `https://quantumexperience.ng.bluemix.net/qx/editor`

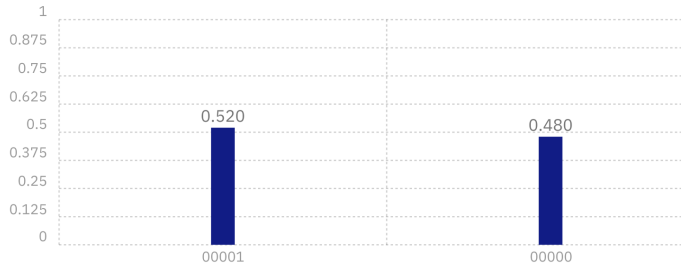
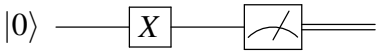


Figura 1.5. Saída do circuito usando a porta H e um medidor (Reprint Courtesy of IBM Corporation ©)

Um exemplo mais simples do que o anterior é a porta X dada pela matriz

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

X é a porta NOT quântica, pois $|1\rangle = X|0\rangle$ e $|0\rangle = X|1\rangle$. Podemos verificar estas equações através da multiplicação matricial de X com $|0\rangle$ e $|1\rangle$. De forma mais compacta, podemos escrever $|j \oplus 1\rangle = X|j\rangle$, onde $\oplus$ é a operação XOR ou soma módulo 2. Por causa disto, a porta X também é representada como $\oplus$. Um circuito usando a porta X é



1 com probabilidade 1.

A simulação no *composer* deverá resultar em uma única saída com probabilidade 1. Aqui vai uma dica sobre o *composer*: Se você quiser apagar uma porta colocada no circuito, clique na porta e arraste para a lixeira que aparece no momento em que começamos arrastar. Se não quiser aproveitar o circuito anterior, faça um novo.

Agora podemos fazer um experimento mais complexo. Como gerar uma superposição tal que as amplitudes do estado $\alpha|0\rangle + \beta|1\rangle$ sejam diferentes? Por exemplo, como gerar o estado $\alpha|0\rangle + \beta|1\rangle$ tal que $|\alpha|^2 = 25\%$ e $|\beta|^2 = 75\%$? A resposta é usar a porta U_3, que só aparece no *composer* depois de clicar na caixa Advanced. A expressão algébrica desta porta é

$$U_3(\theta,\phi,\lambda) = \begin{bmatrix} \cos\frac{\theta}{2} & -e^{i\lambda}\sin\frac{\theta}{2} \\ e^{i\phi}\sin\frac{\theta}{2} & e^{i(\lambda+\phi)}\cos\frac{\theta}{2} \end{bmatrix}.$$

Portanto, aplicando a porta $U_3(\theta,0,0)$ no estado $|0\rangle$, obtemos

$$U_3(\theta,0,0)|0\rangle = \cos\frac{\theta}{2}|0\rangle + \sin\frac{\theta}{2}|1\rangle.$$

Devemos escolher $\theta = 2\pi/3$, pois $\sin^2(\pi/3) = 3/4$. Para digitar os ângulos no *composer*, podemos usar π como entrada. Por exemplo, $\theta = 2\pi/3$ pode ser digitado como "2*pi/3". Os outros ângulos devem ser zero.

U_3 é uma porta coringa, pois ela substitui todas as outras portas de 1 qubit. Infelizmente, este porta é perfeita demais para ser verdade. Por exemplo, poderíamos a princípio escolher θ tão pequeno quanto desejarmos a um custo de uma única porta. No

mesmo contexto, o *composer* disponibiliza duas outras portas de 1 qubit usando ângulos arbitrários, que são

$$U_1(\lambda) = \begin{bmatrix} 1 & 0 \\ 0 & \mathrm{e}^{i\lambda} \end{bmatrix}, \qquad U_2(\phi,\lambda) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -\mathrm{e}^{i\lambda} \\ \mathrm{e}^{i\phi} & \mathrm{e}^{i(\lambda+\phi)} \end{bmatrix}.$$

Note que $U_1(\lambda) = U_3(0,0,\lambda)$ e $U_2(\phi,\lambda) = U_3(\pi/2,\phi,\lambda)$.

Para a determinação da complexidade computacional de um algoritmo, temos que nos restringir a um conjunto finito de portas de 1 qubit, por exemplo, ao conjunto de portas quando não clicamos na caixa Advanced. As portas de 1 qubit disponíveis no *composer* sem clicar na caixa Advanced são

$$\mathrm{id} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

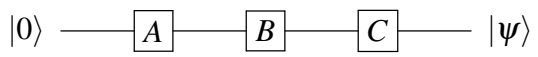
conhecidas como *matrizes de Pauli*,

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \quad S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 0 \\ 0 & \mathrm{e}^{\frac{i\pi}{4}} \end{bmatrix}, \quad T^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & \mathrm{e}^{-\frac{i\pi}{4}} \end{bmatrix}$$

conhecidas como porta Hadamard, porta fase, porta fase conjugada, porta $\pi/8$ ou porta T e sua conjugada. O símbolo † é a notação da matriz transposta conjugada (transponha a matriz e conjugue cada entrada). Os números complexos $\mathrm{e}^{\pm\frac{i\pi}{4}}$ são iguais a

$$\mathrm{e}^{\pm\frac{i\pi}{4}} = \frac{1 \pm i}{\sqrt{2}}.$$

Todo circuito quântico sem os medidores tem uma forma algébrica equivalente. Por exemplo, se A, B e C são portas de 1 qubit, o circuito

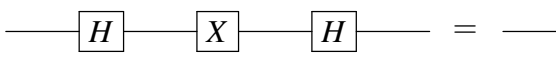


tem a seguinte expressão algébrica equivalente

$$|\psi\rangle = C \cdot B \cdot A \cdot |0\rangle,$$

onde a operação $\cdot$ é o produto matricial, que usualmente é omitido. Note que a expressão algébrica equivalente ao circuito tem a ordem inversa das portas. Portanto, o circuito acima também pode ser mostrado como

onde CBA é uma matriz 2×2. Por exemplo, os seguintes circuitos são equivalentes:

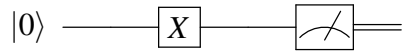


pois $Z = HXH$. A expressão algébrica equivalente pode ser usada para prever a saída do circuito.

Exercício 1. *Mostre que* $|\ell \oplus 1\rangle = X|\ell\rangle$, $(-1)^{\ell}|\ell\rangle = Z|\ell\rangle$, $(-1)^{\ell} i|\ell \oplus 1\rangle = Y|\ell\rangle$ *onde* ℓ *é um bit.*

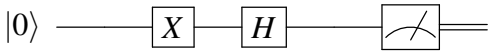
Exercício 2. *Calcule* $H \cdot |0\rangle$, $Z \cdot H \cdot |0\rangle$, $S \cdot H \cdot |0\rangle$, $Z \cdot S \cdot H \cdot |0\rangle$. *Ache os pontos na esfera de Bloch que correspondem a estes vetores.*

Exercício 3. *Usando o composer da IBM e a tecla* Simulate*, obtenha a saída do seguinte circuito:*



Mostre que o resultado está correto.

Exercício 4. *Usando o composer da IBM e a tecla* Simulate*, obtenha a saída do seguinte circuito:*



O resultado deveria dar necessariamente 0 com probabilidade 0.5 e 1 com probabilidade 0.5?

Exercício 5. *PARTE 1. Usando o composer da IBM e a tecla* Simulate*, tente mostrar que os seguintes circuitos não são equivalentes:*

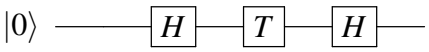
Atenção: você deve usar apenas resultados de simulações (não faça cálculos). Você concluiu que os circuitos não são equivalentes? Veja nota de rodapé[4]*.*

PARTE 2. Calcule algebricamente a saída de ambos circuitos sem o medidor e mostre que as distribuições de probabilidades são iguais.

PARTE 3. Coloque uma porta H no final de ambos circuitos da seguinte forma:

e conclua que os circuitos originais não são equivalentes usando o composer (sem fazer cálculos).

Exercício 6. *Mostre algebricamente que a saída do circuito*



é

$$\frac{\sqrt{2}+1+i}{2\sqrt{2}}|0\rangle + \frac{\sqrt{2}-1-i}{2\sqrt{2}}|1\rangle.$$

Compare a previsão teórica com o resultado do composer da IBM.

Exercício 7. *a) Obtenha a porta H a partir de* $U_2(\phi,\lambda)$ *e* $U_3(\theta,\phi,\lambda)$.

b) Obtenha as portas X, Y e Z a partir de $U_3(\theta,\phi,\lambda)$.

c) Obtenha as portas S, $S^{\dagger}$*, T e* $T^{\dagger}$ *a partir de* $U_1(\lambda)$.

[4]Se você concluiu que os circuitos são equivalentes ou não são equivalentes em qualquer caso a sua conclusão está errada, pois teoricamente as distribuições de probabilidades são iguais e não dá para concluir nada a partir dos resultados do composer.

1.3.4. Estados quânticos de 2 ou mais qubits

O estado de 2 qubits é descrito por um vetor de norma 1 que pertence a um espaço vetorial de 4 dimensões, pois após a medição dos qubits podemos ter 4 resultados: 00, 01, 10, 11. O primeiro bit se refere ao primeiro qubit e o segundo bit ao segundo qubit, ou seja, adotamos a convenção inversa de representar o bit mais significativo à esquerda[5]. Seguindo as leis da mecânica quântica, devemos indexar a base canônica com os possíveis resultados da seguinte forma:

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

No caso clássico, podemos ter dois bits com valores 00 ou 01 ou 10 ou 11 de forma exclusiva. No caso quântico, o estado de 2 qubits é uma superposição da forma

$$|\psi\rangle = a_0|00\rangle + a_1|01\rangle + a_2|10\rangle + a_3|11\rangle,$$

onde a_0, a_1, a_2 e a_3 são números complexos. A medição dos qubits resulta em 00 ou 01 ou 10 ou 11 de forma exclusiva e estocástica. Isto é, não há como saber qual será o resultado da medição mesmo conhecendo $|\psi\rangle$. Porém, se conhecemos $|\psi\rangle$ então sabemos quais são as probabilidades:

$$\begin{aligned} \text{prob}(00) &= |a_0|^2, \\ \text{prob}(01) &= |a_1|^2, \\ \text{prob}(10) &= |a_2|^2, \\ \text{prob}(11) &= |a_3|^2. \end{aligned}$$

Se não conhecemos o estado $|\psi\rangle$ de 2 qubits, uma única medição não permite a determinação de $|\psi\rangle$, isto é, a determinação dos coeficientes a_0, a_1, a_2 e a_3. Existe um teorema importante na computação quântica conhecido como *teorema da não-clonagem* [15, 28, 42], que é formulado da seguinte maneira:

Teorema. *Não é possível criar uma cópia idêntica de um estado quântico desconhecido.*

Este teorema restringe severamente qualquer possibilidade de determinação de $|\psi\rangle$ através de medições. No entanto, se pudermos gerar $|\psi\rangle$ novamente, por exemplo, através de um circuito, podemos repetir o processo todo diversas vezes e obter uma aproximação para $\text{prob}(00)$, $\text{prob}(01)$, $\text{prob}(10)$ e $\text{prob}(11)$. Por exemplo, repetindo 1000 vezes, podemos determinar estas probabilidades com 2 dígitos precisos. Infelizmente, ainda assim não podemos determinar $|\psi\rangle$ exatamente, pois conhecer $|a_0|^2$ não permite determinar a_0

[5] O gráfico de probabilidades da saída do *composer* da IBM adota a ordem usual (bits menos significativos à direita).

exatamente. Pode parecer que isto não merece importância—falso. Vamos dar um exemplo crítico. Suponha que

$$\text{prob}(00) = \frac{1}{2}, \quad \text{prob}(01) = 0, \quad \text{prob}(10) = 0, \quad \text{prob}(11) = \frac{1}{2}.$$

Temos pelos menos duas possibilidades para $|\psi\rangle$:

$$|\psi_1\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \quad \text{e} \quad |\psi_2\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}.$$

Note que $|\psi_1\rangle$ e $|\psi_2\rangle$ são ortogonais. Isto mostra que podemos cometer um erro sério e dificulta a verificação da equivalência de circuitos usando implementações no *composer*.

Neste ponto, a seguinte questão é relevante: Suponha que o estado $|\psi\rangle$ de 2 qubits é conhecido, é possível determinar o estado de cada qubit? A resposta é "depende de $|\psi\rangle$". Se $|\psi\rangle$ for um dos estados da base computacional então sabemos o estado de cada qubit. Por exemplo, suponha que $|\psi\rangle = |10\rangle$. Temos que fatorar $|\psi\rangle$ da seguinte forma:

$$|10\rangle = |1\rangle|0\rangle = |1\rangle \otimes |0\rangle,$$

onde $\otimes$ é chamado de *produto de Kronecker*. Quando a fatoração é bem sucedida, sabemos o estado de cada qubit. Neste caso, o primeiro qubit está no estado $|1\rangle$ e o segundo no estado $|0\rangle$. Quando escrevemos $|1\rangle|0\rangle$, o produto de Kronecker fica subentendido.

O produto de Kronecker[6] de dois vetores ou duas matrizes é definido da seguinte forma. Seja A uma matriz $m \times n$ e B uma matriz $p \times q$. Então

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ & \ddots & \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}.$$

A dimensão da matriz resultante é $mp \times nq$. O produto de Kronecker dos vetores bidimensionais $|1\rangle$ e $|0\rangle$ é calculado interpretando estes vetores como matrizes 2×1 e é dado por

$$|1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ \\ 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = |10\rangle.$$

Note que o produto de Kronecker não é comutativo. Por exemplo, $|1\rangle \otimes |0\rangle \neq |0\rangle \otimes |1\rangle$.

As leis da mecânica quântica afirmam que se o estado do primeiro qubit for $|\psi_1\rangle$ e o estado do segundo qubit for $|\psi_2\rangle$ então o estado $|\psi\rangle$ do sistema composto é

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle.$$

[6]Existe uma formulação mais abstrata do produto de Kronecker que é denominada *produto tensorial*. Neste curso, usamos estes termos como sinônimos. A notação $A \otimes B$ é lida da forma "A tensor B", no entanto os termos *tensor* e *produto tensorial* não têm nenhuma relação com tensores (generalização de matrizes) ou produto de tensores.

Sempre podemos obter o estado do sistema composto quando conhecemos os estados das partes. Porém, a direção inversa não é possível em geral. Por exemplo, suponha que o estado de 2 qubits seja

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

Queremos encontrar estados de 1 qubit $|\psi_1\rangle = \alpha|0\rangle + \beta|1\rangle$ e $|\psi_2\rangle = \gamma|0\rangle + \delta|1\rangle$ tal que

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} = (\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle).$$

Expandindo o lado direito e igualando os coeficientes, obtemos o seguinte sistema de equações:

$$\begin{aligned}
\alpha\gamma &= \frac{1}{\sqrt{2}}, \\
\alpha\delta &= 0, \\
\beta\gamma &= 0, \\
\beta\delta &= \frac{1}{\sqrt{2}}.
\end{aligned}$$

Como este sistema de equações não admite nenhuma solução, o estado $|\psi\rangle$ de 2 qubits não pode ser escrito como o produto de Kronecker de estados de 1 qubit. Aqui vai mais uma estranheza quântica:

Fato. *Um sistema quântico composto pode estar em um estado bem definido enquanto que as partes do sistema não têm nenhum estado definido.*

Um estado quântico de um sistema composto que não pode ser fatorado usando o produto de Kronecker é chamado *estado emaranhado*. Os estados emaranhados são muito importantes na computação quântica, pois sem eles o poder computacional do computador quântico seria o mesmo do computador clássico. No entanto, é importante frisar que a presença de um estado emaranhado em um algoritmo quântico para resolver um problema não garante que este algoritmo é mais eficiente do que um algoritmo clássico para o mesmo problema.

Podemos generalizar a discussão desta seção para n qubits, onde $n \geq 1$. A base computacional é constituída por vetores de 2^n entradas

$$|0\cdots00\rangle = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad |0\cdots01\rangle = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \quad \cdots, \quad |1\cdots11\rangle = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}.$$

Note que o número de cadeias de bits é 2^n. Cada cadeia pode ser escrita na notação decimal como

$$|0\cdots00\rangle = |0\rangle, \;\; |0\cdots01\rangle = |1\rangle, \;\; |0\cdots10\rangle = |2\rangle, \;\; \ldots \;\; |1\cdots11\rangle = |2^n - 1\rangle.$$

Um estado genérico pertence a um espaço vetorial de 2^n dimensões, e, na notação decimal, ele é escrito como

$$|\psi\rangle = a_0|0\rangle + a_1|1\rangle + a_2|2\rangle + \cdots + a_{2^n-1}|2^n-1\rangle,$$

onde

$$|a_0|^2 + |a_1|^2 + |a_2|^2 + \cdots + |a_{2^n-1}|^2 = 1.$$

Após uma medição dos n qubits, obtemos como resultado uma cadeia de n bits de forma estocástica. O resultado é a cadeia $0\cdots00$ com probabilidade $|a_0|^2$ ou é a cadeia $0\cdots01$ com probabilidade $|a_1|^2$ e assim por diante.

Um vetor da base computacional de n qubits pode ser escrito como produto de Kronecker de n vetores de 1 qubit. Por exemplo, no caso de $n = 3$ qubits, podemos obter o segundo vetor da base computacional através do produto de Kronecker da seguinte maneira:

$$|0\rangle \otimes |0\rangle \otimes |1\rangle = \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{pmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{pmatrix} = |001\rangle.$$

Na notação decimal, $|0\rangle$ pode ser confundido com o estado de 1 qubit. Para evitar a confusão, temos que saber qual é o número de qubits em questão. Por exemplo, se $|0\rangle$ se refere ao estado de 3 qubits na notação decimal, então a descrição deve ser convertida para $|000\rangle$.

Exercício 8. *Mostre que* $|0\rangle \otimes |1\rangle \neq |1\rangle \otimes |0\rangle$ *(*$\otimes$ *não é comutativo) e* $(|0\rangle \otimes |1\rangle) \otimes |1\rangle = |0\rangle \otimes (|1\rangle \otimes |1\rangle)$. *(*$\otimes$ *é associativo)*

Exercício 9. *Por definição,* $M^{\otimes n} = M \otimes \cdots \otimes M$ *(n termos), onde M é uma matriz de qualquer dimensão. Calcule* $I_2^{\otimes 3}$, $X^{\otimes 3}$, $I_2 \otimes X$, $X \otimes I_2$, $I_2 \otimes Z$, $Z \otimes I_2$.

Exercício 10. *Diga se é verdadeiro ou falso: a)* $\langle 01|10\rangle = 1$, *b)* $\langle 101|111\rangle = 1$, *c)* $\langle +|-\rangle = 1$, *d)* $\langle 0|+\rangle = 0$, *e)* $\langle 10|++\rangle = 1/2$, *f)* $\langle ++|+-\rangle = 0$, *g)* $\langle 111|--+\rangle = -1/2$.

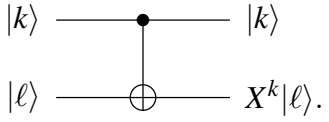
Exercício 11. *Seja* $*$ *o produto de números,* $\cdot$ *o produto matricial e* $\otimes$ *o produto de Kronecker. Diga se é verdadeiro ou falso ou inconsistente: a)* $\langle 01|10\rangle = \langle 0|1\rangle * \langle 1|0\rangle$, *b)* $|0\rangle\langle 1| = |0\rangle * \langle 1|$, *c)* $|0\rangle\langle 1| = |0\rangle \cdot \langle 1|$, *d)* $|0\rangle\langle 1| = |0\rangle \otimes \langle 1|$, *e)* $|+\rangle\langle -| = |-\rangle\langle +|$, *f)* $(|0\rangle\langle 1|) \cdot |1\rangle = |0\rangle(\langle 1|1\rangle)$, *g)* $H \cdot (|0\rangle\langle 1|) = |+\rangle \cdot \langle 1|$.

1.3.5. Portas lógicas quânticas de 2 qubits

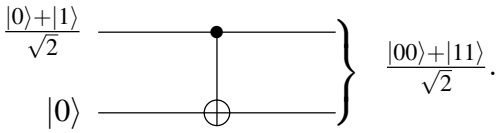
A porta de 2 qubits mais importante é a porta CNOT ou porta NOT controlada, também representada por $C(X)$ ou cX. Ela é definida pela expressão

$$\text{CNOT}|k\rangle|\ell\rangle = |k\rangle X^k|\ell\rangle,$$

e é representada pelo circuito



No circuito acima, $|k\rangle$ e $|\ell\rangle$ são estados da base computacional de 1 qubit. O estado do primeiro qubit não muda após a aplicação da porta CNOT. O estado do segundo qubit só muda se o valor de k for 1. Neste caso a saída é $X|\ell\rangle = |\ell \oplus 1\rangle$. Se $k = 0$ então $X^0 = I_2$ e $I_2|\ell\rangle = |\ell\rangle$, onde I_2 é a matriz identidade 2×2. A definição que acabamos de mostrar descreve a atuação da porta CNOT na base computacional de 2 qubits. Na álgebra linear, esta definição está completa, pois para saber a atuação da porta CNOT em um vetor que é uma superposição de vetores da base computacional usamos a linearidade desta porta. Por exemplo, no circuito abaixo a entrada do primeiro qubit está em superposição:



Como calcular a saída? A melhor maneira de determinar a saída é usar um cálculo algébrico. Após usar a distributividade do produto de Kronecker, a entrada pode se escrita como

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |0\rangle \;=\; \frac{1}{\sqrt{2}}|0\rangle \otimes |0\rangle + \frac{1}{\sqrt{2}}|1\rangle \otimes |0\rangle \;=\; \frac{|00\rangle + |10\rangle}{\sqrt{2}}.$$

Para calcular a atuação da porta CNOT em uma soma de vetores, usamos a linearidade do produto matricial (denotado por $\cdot$), isto é,

$$\text{CNOT} \cdot \left(\frac{|00\rangle + |10\rangle}{\sqrt{2}} \right) \;=\; \frac{1}{\sqrt{2}} \text{CNOT} \cdot |00\rangle + \frac{1}{\sqrt{2}} \text{CNOT} \cdot |10\rangle.$$

As notações $\text{CNOT} \cdot |00\rangle$ ou $\text{CNOT}|00\rangle$ denotam a multiplicação da matriz CNOT pelo vetor $|00\rangle$. A matriz CNOT é

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Podemos confirmar que a saída é

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

Como este resultado é um estado emaranhado, não podemos fatorá-lo e portanto não podemos escrever uma saída para cada qubit. Após a medição, a saída é 00 com probabilidade 1/2 ou 11 com probabilidade 1/2.

Para gerar o circuito acima no *composer* da IBM, vamos usar o *backend ibmqx2*, cujo mapa de conectividade está mostrado no lado direito da figura 1.6. O usuário pode selecionar o *backend* no *composer* da IBM ao criar uma nova simulação. O mapa descreve como os qubits interagem e deve ser usado no momento de fazer uma porta CNOT controlada. É possível construir um CNOT com controle no qubit 0 e alvo no qubit 1, porém

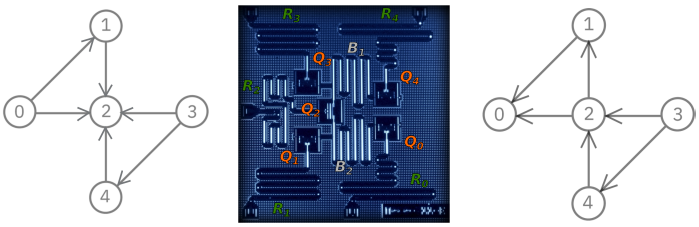


Figura 1.6. Mapa de conectividade do chip do IBM Q 5 Yorktown [ibmqx2] (esquerda), Tenerife [ibmqx4] (direita) e o chip físico (centro) com cerca de 5mm

não é possível inverter esta ordem diretamente[7]. Certamente não é possível construir um CNOT entre qubits que não são vizinhos.

Como os qubits do ibmqx2 estão inicialmente no estado $|0\rangle$, temos que usar uma porta H no primeiro qubit para gerar a superposição $(|0\rangle + |1\rangle)/2$. Portando, devemos implementar a expressão

$$\text{CNOT} \cdot (H \otimes I)$$

como mostrado na figura 1.7.

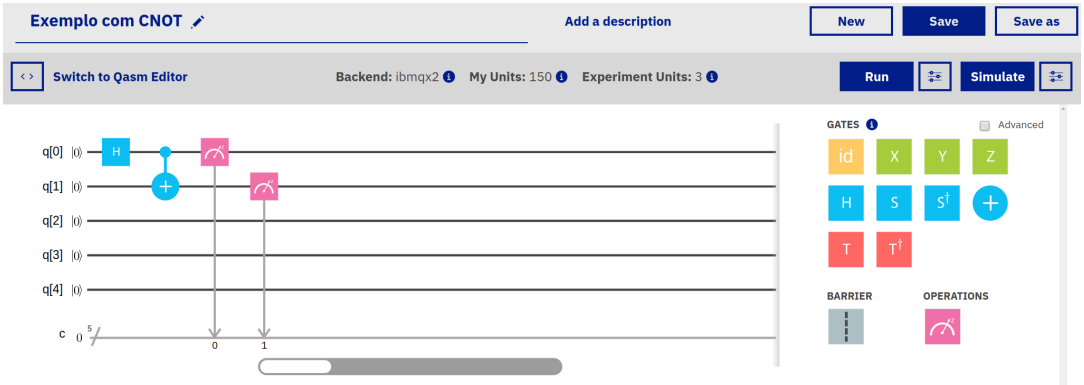


Figura 1.7. Circuito usando as portas H e CNOT (Reprint Courtesy of IBM Corporation ©)

O controle da porta CNOT é o qubit q[0] e o alvo é o qubit q[1]. Para fazer um CNOT arrastamos a porta $\oplus$ até o segundo qubit (q[1]) e levamos o controle até o primeiro qubit (q[0]). É importante usar o *backend ibmqx2*, pois não é possível fazer um CNOT com controle no primeiro qubit no *backend ibmqx4*. No final do algoritmo, devemos colocar as portas de medição representadas pelo visor do voltímetro. Automaticamente é gerado um programa *Qasm* (Quantum assembly language) que pode ser visto clicando em $\langle\,\rangle$ ou em *Switch to Qasm Editor*. Em diversas situações é melhor lidar com o programa *Qasm* em vez da interface gráfica, por exemplo, quando queremos introduzir uma porta no começo do circuito, pois arrastar muitas portas na interface é trabalhoso.

A saída de uma simulação com 100 repetições está mostrada na figura 1.8. Como vimos antes, o resultado é um histograma da distribuição de probabilidades que é gerado

[7]Existe um truque baseado na fórmula $\text{CNOT}_{01} = (H \otimes H)\,\text{CNOT}_{10}(H \otimes)$, que inverte a ordem do controle e alvo, onde a notação CNOT_{01} é um CNOT com controle no qubit 0 e alvo no qubit 1.

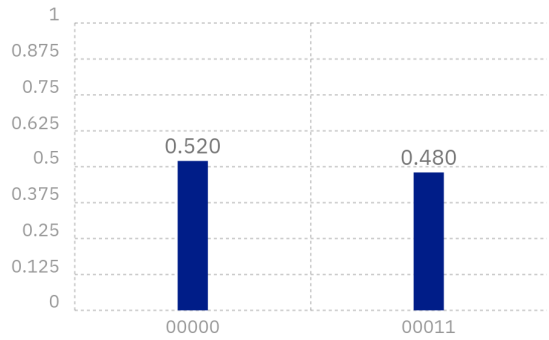


Figura 1.8. Saída do circuito usando as portas H e CNOT (Reprint Courtesy of IBM Corporation ©)

por 100 repetições. A figura mostra que o resultado q[4]q[3]q[2]q[1]q[0]=00011 foi obtido com probabilidade 0.48 e o resultado q[4]q[3]q[2]q[1]q[0]=00000 foi obtido com probabilidade 0.52. No nosso caso, devemos descartar os qubits q[2], q[3] e q[4]. Portanto, o resultado mostra que o algoritmo retornou 00 com probabilidade 0.52 e 11 com probabilidade 0.48, que são resultados próximos de 1/2 como era esperado em função do cálculo analítico.

O próximo exemplo é mais simples do que o anterior. Considere o seguinte circuito sem a medição ao final:

$$|0\rangle \text{ ——} \boxed{H} \text{——} |+\rangle$$
$$|0\rangle \text{ ——} \boxed{H} \text{——} |+\rangle.$$

Como calcular a saída? Usualmente, a forma algébrica é a mais simples. A forma algébrica do circuito acima é $(H \otimes H)|00\rangle$. O cálculo é executado da seguinte forma:

$$(H \otimes H)|00\rangle \;=\; (H \otimes H) \cdot (|0\rangle \otimes |0\rangle) \;=\; (H|0\rangle) \otimes (H|0\rangle) \;=\; |+\rangle \otimes |+\rangle.$$

Na segunda igualdade, usamos a seguinte propriedade do produto de Kronecker:

$$(A \otimes B) \cdot (|\psi_1\rangle \otimes |\psi_2\rangle) \;=\; (A|\psi_1\rangle) \otimes (B|\psi_2\rangle),$$

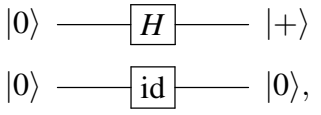
que é válida para quaisquer matrizes A e B e vetores $|\psi_1\rangle$ e $|\psi_2\rangle$ desde que as dimensões dos vetores sejam compatíveis com as dimensões correspondentes das matrizes. Se A é uma matriz $k \times k$ e B uma matriz $\ell \times \ell$ então $|\psi_1\rangle$ tem que ter k entradas e $|\psi_2\rangle$ tem que ter ℓ entradas. Portanto a saída do circuito é

$$|+\rangle \otimes |+\rangle \;=\; \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} \;=\; \frac{|0\rangle + |1\rangle + |2\rangle + |3\rangle}{2}.$$

Vamos ver mais um exemplo simples. Considere o seguinte circuito sem a medição ao final:

$$|0\rangle \text{ ——} \boxed{H} \text{——} |+\rangle$$
$$|0\rangle \text{ ————————} |0\rangle.$$

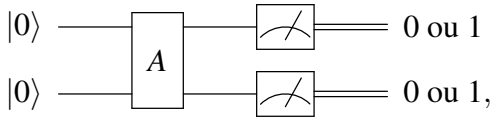
Como calcular a saída usando a expressão algébrica equivalente? A dica é usar o seguinte circuito equivalente:



onde $\boxed{\text{id}}$ é a matriz identidade de duas dimensões também representada como I_2. O cálculo algébrico é feito da seguinte forma:

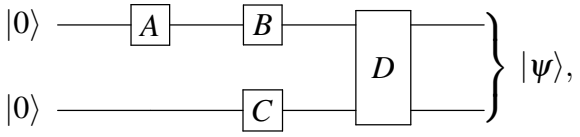
$$(H \otimes I_2)|00\rangle \;=\; (H|0\rangle) \otimes (I_2|0\rangle) \;=\; |+\rangle \otimes |0\rangle \;=\; \frac{|00\rangle + |10\rangle}{\sqrt{2}}.$$

O circuito de 2 qubits mais geral possível é



onde A é uma matriz unitária 4×4. Relembrando, a matriz A é unitária se a ação de A em um vetor de norma 1 genérico resulta em um vetor de norma 1. Existem várias maneiras algébricas de verificar se A é uma matriz unitária: (1) As colunas de A tem que ser vetores de norma 1 ortogonais entre si, ou (2) a multiplicação de A pela sua transposta conjugada tem que dar a matriz identidade, isto é, $A \cdot A^{\dagger} = I$.

Quando convertemos um circuito quântico na sua expressão algébrica equivalente, devemos usar o produto de Kronecker para portas na mesma coluna e o produto matricial para portas na mesma linha ou em sequência, no entanto, devemos inverter a ordem das portas no segundo caso. Por exemplo, a expressão algébrica equivalente ao circuito



onde A, B e C são portas de 1 qubit e D é uma porta de 2 qubits, é

$$|\psi\rangle \;=\; D \cdot (B \otimes C) \cdot (A \otimes I_2) \cdot |00\rangle.$$

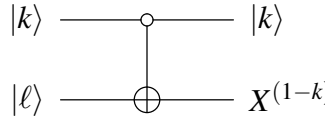
Podemos simplificar um pouco esta expressão e escrever

$$|\psi\rangle \;=\; D\,(BA \otimes C)|00\rangle.$$

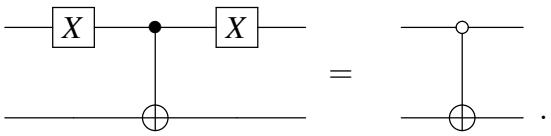
A porta CNOT é tão importante que vamos descrever uma variante chamada porta CNOT ativada pelo 0. Ela é definida pela expressão

$$|k\rangle|\ell\rangle \longrightarrow |k\rangle X^{(1-k)}|\ell\rangle,$$

e é representada pelo circuito



Note que o qubit de controle é denotado pelo círculo vazio indicando que o controle só é ativado se o qubit está no estado $|0\rangle$. Esta porta pode ser obtida a partir do CNOT usual por conjugação pela porta $(X \otimes I_2)$, como mostra a seguinte equivalência de circuitos:



Esta porta é representada por uma matriz em blocos da forma

$$\begin{bmatrix} X & \\ & I_2 \end{bmatrix}.$$

Exercício 12. *Calcule $(X^{\otimes 3}) \cdot |010\rangle$ e compare com $(X \cdot |0\rangle) \otimes (X \cdot |1\rangle) \otimes (X \cdot |0\rangle)$ (pense num vetor como uma matriz 2×1) e expresse o resultado na base computacional usando a notação de Dirac (não use o formato matricial nas respostas finais). Por que as respostas são iguais? Qual cálculo é mais simples?*

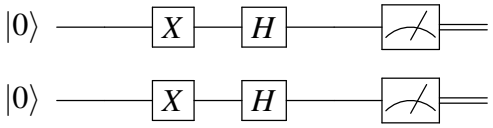
Exercício 13. *Calcule $H^{\otimes 2}$. Calcule $(H^{\otimes 2}) \cdot |11\rangle$ e compare com $(H \cdot |1\rangle)^{\otimes 2}$. Por que as respostas são iguais?*

Exercício 14. *Considere os seguintes estados de 2 qubits:*

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle), \;\; |\psi_2\rangle = \frac{1}{\sqrt{3}}(|00\rangle + |01\rangle) + \frac{1}{\sqrt{6}}(|10\rangle + |11\rangle), \;\; |\psi_3\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle).$$

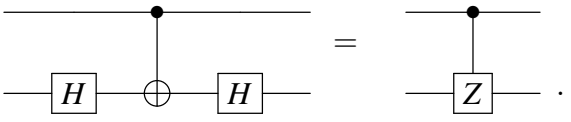
PARTE 1. Mostre que $|\psi_1\rangle$ e $|\psi_2\rangle$ não são emaranhados e $|\psi_3\rangle$ é emaranhado. PARTE 2. Implemente no composer da IBM os circuitos que geram os estados $|\psi_1\rangle$, $|\psi_2\rangle$ e $|\psi_3\rangle$.

Exercício 15. *Implemente o seguinte circuito no composer da IBM:*



Use os resultados do exercício 13 para mostrar que a distribuição de probabilidades gerada pelo composer é igual ou próxima do esperado segundo os cálculos analíticos.

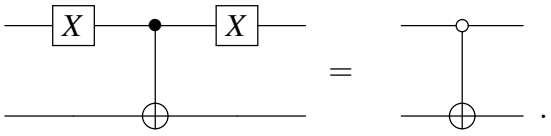
Exercício 16. *Mostre a equivalência algébrica dos seguintes circuitos*



Exercício 17. *Sabendo que a porta NOT controlada pelo 0 é representada por uma matriz em blocos da forma*

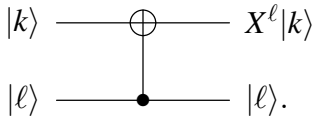
$$\begin{bmatrix} X & \\ & I_2 \end{bmatrix},$$

mostre a equivalência algébrica dos seguintes circuitos

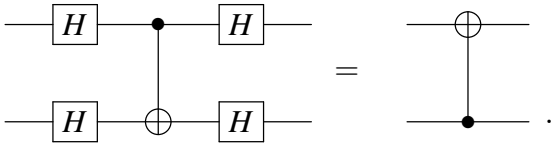


usando matrizes em blocos. [Dica: converta a expressão $(X \otimes I_2) \cdot CNOT \cdot (X \otimes I_2)$ *para o formato matricial usando matrizes em blocos e faça a multiplicação.]*

Exercício 18. *Obtenha a representação matricial da porta NOT com os qubits invertidos, mostrada no seguinte circuito:*



Exercício 19. *Mostre a equivalência algébrica dos seguintes circuitos*



Exercício 20. *Seja U uma porta de 1 qubit. Mostre que a porta U controlada de 2 qubits é uma matriz diagonal em blocos dada por*

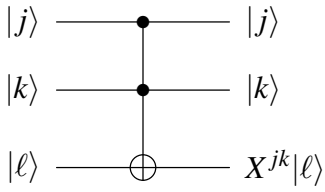
$$C(U) = \begin{bmatrix} I_2 & \\ & U \end{bmatrix}.$$

1.3.6. Portas lógicas quânticas de 3 ou mais qubits

A porta de 3 qubits mais importante é a porta Toffoli[8] designada como CCNOT ou $C^2(X)$, definida pela expressão

$$C^2(X)\,|j\rangle|k\rangle|\ell\rangle = |j\rangle|k\rangle X^{jk}|\ell\rangle,$$

e representada pelo circuito



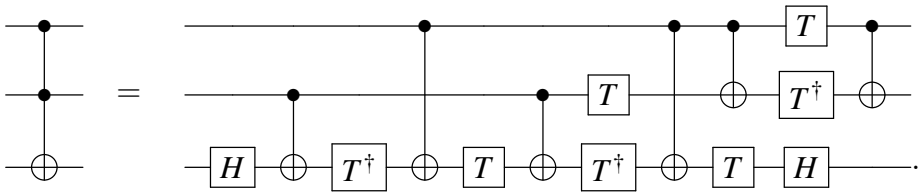
Esta porta tem 2 controles e 1 alvo. A porta X atua no alvo se, e somente se, os valores dos bits de controle forem ambos iguais a 1. Se um dos bits de controles for 0, o valor de terceiro qubit não se altera. Esta porta pode ser vista como a versão quântica da porta clássica AND, pois se $\ell = 0$, a saída do terceiro qubit é $(j \text{ AND } k)$. A saída do terceiro qubit também pode ser escrito como $(j \text{ AND } k) \oplus \ell$. A matriz da porta Toffoli é uma

[8] A pronúncia é *tófoli*.

matriz diagonal em blocos dada por

$$C^2(X) = \begin{bmatrix} I_2 & & & \\ & I_2 & & \\ & & I_2 & \\ & & & X \end{bmatrix}.$$

A porta Toffoli não está disponível no *composer* da IBM (Ela está disponível na próxima geração do *composer*—veja a versão beta). A sua implementação depende da decomposição em CNOT e portas de 1 qubit (H, T), como descrito pela equivalência dos seguintes circuitos:



A implementação do circuito da porta Toffoli no *backend* ibmqx2 está mostrado na figura 1.9. Para reproduzir esta implementação, basta abrir um novo experimento no *backend* ibmqx2 e copiar o código da nota de rodapé[9], colar no editor *Qasm* a partir da quarta linha e adicionar os medidores. Outra opção é baixar o arquivo *Toffoli_ibmqx2.qasm* da conta *programaquantica*[10] do GitHub e depois usar o comando *Import QASM* do *composer* para fazer o carregamento do arquivo.

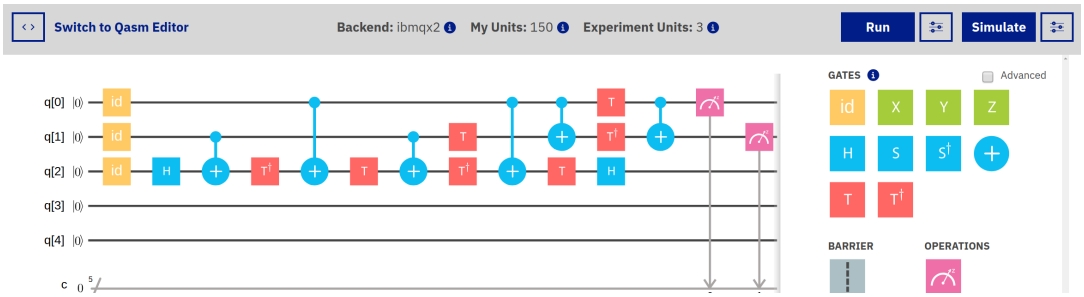


Figura 1.9. Circuito da porta Toffoli no *backend* ibmqx2 (Reprint Courtesy of IBM Corporation ©)

Para verificar que esta decomposição está correta é necessário testar cada linha da

[9] id q[0]; id q[1]; id q[2]; h q[2]; cx q[1],q[2]; tdg q[2]; cx q[0],q[2]; t q[2]; cx q[1],q[2]; t q[1]; tdg q[2]; cx q[0],q[2]; cx q[0],q[1]; t q[2]; t q[0]; tdg q[1]; h q[2]; cx q[0],q[1];

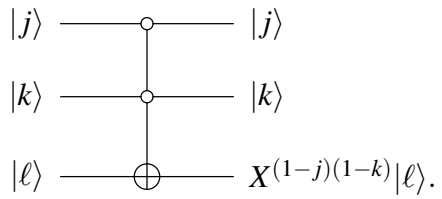
[10] https://github.com/programaquantica

tabela a seguir:

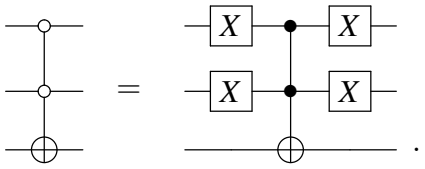
entrada			saída		
q[0]	q[1]	q[2]	q[0]	q[1]	q[2]
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0\rangle$	$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|0\rangle$	$\|1\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|0\rangle$	$\|1\rangle$	$\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|0\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

As entradas podem ser implementadas trocando as portas $\boxed{\text{id}}$ no começo do circuito por portas $\boxed{\text{X}}$ e a probabilidade de saída colocando medidores no final. No entanto, note que isto não serve de prova da equivalência dos circuitos.

Existem variantes da porta Toffoli ativadas pelo 0. Por exemplo, o circuito



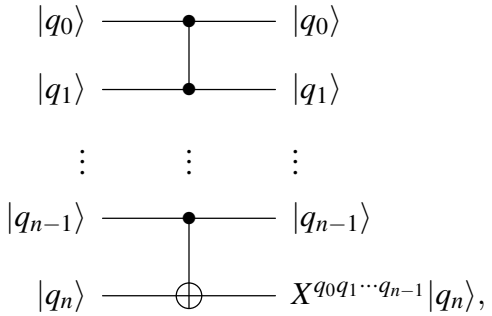
implementa uma porta que inverte o valor do terceiro qubit se, e somente se, os dois primeiros qubits estão no estado $|0\rangle$. Esta porta pode ser implementada usando uma porta Toffoli usual e a porta X, como mostra a equivalência de circuitos a seguir:



A porta Toffoli generalizada $C^n(X)$ é uma porta de $(n+1)$ qubits com n qubits de controle e um alvo. Ela é definida pela expressão

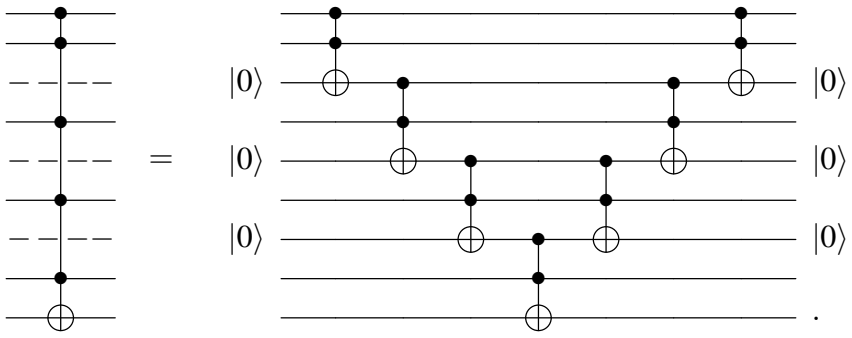
$$C^n(X)|q_0\rangle|q_1\rangle\cdots|q_{n-1}\rangle|q_n\rangle = |q_0\rangle|q_1\rangle\cdots|q_{n-1}\rangle X^{q_0q_1\cdots q_{n-1}}|q_n\rangle,$$

e é representada pelo circuito



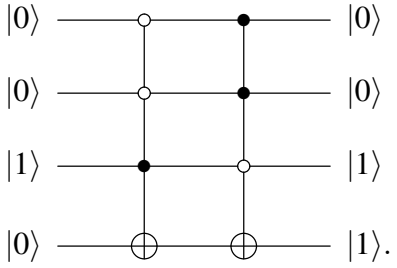
onde $q_0 q_1 \cdots q_{n-1}$ é o produto dos bits q_0, q_1, ..., q_{n-1}. Portanto, o estado do qubit alvo inverte se, e somente se, todos os qubits de controle estiverem no estado $|1\rangle$. O estado de cada qubit de controle não muda quando descrevemos esta porta atuando na base computacional. A atuação em um vetor genérico é obtida escrevendo o vetor na base computacional e usando linearidade.

A maneira mais simples de decompor a porta Toffoli generalizada em termos de portas Toffoli usuais é usando $(n-2)$ qubits de rascunho chamados *ancillas*. Os qubits ancillas são introduzidos de maneira intercalada com os qubits de controle, sendo que o primeiro qubit ancilla deve ser inserido entre o segundo e terceiro qubits. A melhor maneira de explicar a decomposição é mostrar um exemplo. Considere a porta $C^5(X)$, cuja decomposição requer 3 *ancillas*, como mostra a seguinte equivalência de circuitos:



A porta Toffoli generalizada também pode ser ativada pelo 0. Neste caso, o qubit de controle é mostrado com um círculo vazio e pode ser obtido através porta Toffoli generalizada ativada pelo 1 com os controles conjugados por portas X. Para n qubits, temos 2^n portas Toffoli generalizadas que podem implementar qualquer função booleana de n bits, como descrevemos no próximo parágrafo.

Agora vamos mostrar como obter o circuito quântico de uma tabela verdade. Para mostrar que as portas Toffoli generalizadas ativadas pelo 0 e 1 podem ser usadas para implementar uma função booleana em um computador quântico, vamos tomar a função booleana $f(a,b,c)$ descrita na seção 1.1 como exemplo. Fica evidente como se obtém o caso geral. Como f tem 3 bits de entrada, vamos usar portas Toffoli generalizadas de 3 controles. A saída da função f é a saída de uma medição do qubit alvo. Como a função f tem 2 cláusulas, vamos usar 2 portas Toffoli generalizadas. A primeira porta deve ser ativada pela entrada $|001\rangle$ e a segunda pela entrada $|110\rangle$. O seguinte circuito implementa a função f:

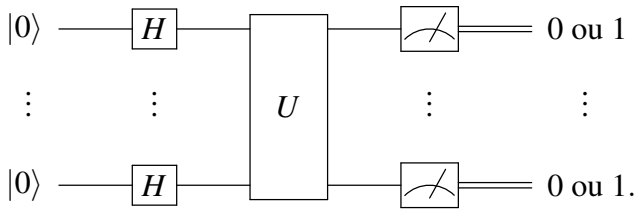


Isto mostra que o computador quântico pode calcular qualquer função booleana de n bits usando uma porta Toffoli generalizada de $(n+1)$ bits para cada saída 1 da tabela

verdade associada. É claro que não faz sentido construir um máquina muito mais cara para executar algoritmos clássicos. No entanto, a implementação que acabamos de descrever pode ser usada em entradas superpostas, algo não permitido no computador clássico.

1.4. Paralelismo quântico e o modelo padrão da computação quântica

O modelo padrão da computação quântica é descrito pelo circuito



O estado inicial de cada qubit é $|0\rangle$. Depois, a porta Hadamard H é aplicada a cada qubit, preparando para o *paralelismo quântico*. Na sequência, a matriz unitária U é aplicada em todos os qubits. Finalmente é realizada a medição de todos os qubits retornando uma cadeia de bits.

O paralelismo quântico é a execução simultânea de um algoritmo com mais do que uma entrada. Para entender este conceito, vamos focar apenas na matriz U sem considerar as portas H. A matriz U e as medições do modelo padrão podem ser interpretados como um algoritmo randomizado no sentido clássico. Se x é uma cadeia de n bits e U é uma matriz $2^n \times 2^n$, então $U|x\rangle$ é uma superposição de 2^n vetores da base computacional. Após a medição, uma cadeia y de n bits é retornada. Isto é, se a entrada de U é x, a saída após a medição é y. O "algoritmo" U foi executado para uma única entrada x. Agora vamos colocar as portas H de volta.

O primeiro passo do modelo padrão é executar o seguinte cálculo: $(H|0\rangle) \otimes \cdots \otimes (H|0\rangle)$, isto é

$$\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) \otimes \cdots \otimes \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right).$$

Após expandir este produto, obtemos

$$\frac{1}{\sqrt{2^n}}\big(|0\cdots0\rangle + |0\cdots01\rangle + |0\cdots10\rangle + \cdots + |1\cdots1\rangle\big).$$

Todos as possíveis cadeias de n bits aparecem na soma acima. Ou seja, todas a entradas possíveis de um algoritmo clássico com n bits estão representadas na soma acima. O próximo passo do modelo padrão é aplicar a matriz U. Pela linearidade da álgebra linear, a matriz U deve ser aplicada a cada termo do soma acima simultaneamente. Portanto, é possível realizar 2^n cálculos simultâneos no computador quântico de n bits. Note porém que os resultados destes cálculos estão misturados em uma nova soma e, após a medição, o resultado é uma única cadeia de n bits.

1.5. Algoritmo de Grover

O algoritmo de Grover [16] é um algoritmo de busca em bancos de dados não-ordenados. Do ponto de vista prático é mais interessante ordenar o banco de dados e depois realizar a

busca. Em função disto, vamos descrever nesta seção uma outra formulação do algoritmo de Grover que mostra de forma mais clara a sua importância [29]. Outras referências que apresentam o algoritmo de Grover são [3, 4, 6, 12, 17, 18, 19, 20, 23, 25, 26, 27, 36, 41].

1.5.1. Formulação do problema

Seja N uma potência de 2, isto é, $N = 2^n$ para algum número inteiro n. Suponha que $f : \{0,\ldots,N-1\} \to \{0,1\}$ é uma função booleana tal que $f(x) = 1$ se, e somente se, $x = x_0$ para algum valor fixo x_0. Assim,

$$f(x) = \begin{cases} 1, & \text{se } x = x_0, \\ 0, & \text{caso contrário.} \end{cases}$$

Suponha que x_0 não é conhecido. Como podemos achar x_0 usando f? Do ponto de vista computacional, queremos desenvolver um algoritmo que avalie f o número mínimo de vezes.

Classicamente, o algoritmo mais eficiente consulta f um número de vezes proporcional a N, isto é, a complexidade de consultas é $O(N)$. Como é feito na prática? Temos que pedir a uma pessoa a qual confiamos para gerar um número aleatório x_0 de n bits. Esta pessoa esconde x_0 e faz uma subrotina compilada da função f. Podemos usar a subrotina quantas vezes desejarmos, porém não podemos examinar a subrotina e tentar descobrir x_0 *hackeando* o código. O algoritmo clássico que resolve este problema é um iteração que consulta $f(j)$ para j indo de 0 a $2^n - 1$. Assim que $f(j)$ retorna o valor 1, o programa é interrompido e x_0 é encontrado.

Quanticamente, é possível melhorar a complexidade de consultas para $O(\sqrt{N})$. Como é feito na prática? Temos que pedir de novo a uma pessoa de confiança para gerar um número aleatório x_0 de n bits. Esta pessoa, que usualmente é chamada de *oráculo*, implementa a função f através de uma matriz unitária F_{x_0}. Nós podemos usar F_{x_0} no computador quântico, porém não podemos ver os detalhes da implementação de F_{x_0}. Cada vez que usarmos F_{x_0}, adicionamos uma unidade na contagem. Podemos usar portas adicionais que obviamente não dependem de x_0.

Temos um mesmo problema que pode ser resolvido por algoritmos que são executados de duas formas diferentes. No primeiro caso, um computador clássico de $O(n)$ bits é usado e a solução é obtida após $O(N)$ passos. No segundo caso, um computador quântico de $O(n)$ qubits é usado e a solução é obtida após $O(\sqrt{N})$ passos. Este ganho de complexidade justifica o investimento em um hardware quântico e no estudo de técnicas para desenvolver algoritmos quânticos, que necessariamente têm que explorar a superposição de estados. No caso do algoritmo de Grover, a matriz F_{x_0} atuando na superposição de estados avalia a função f em mais de uma entrada ao mesmo tempo com os mesmos recursos disponíveis. Esta tarefa só pode ser executada no computador clássico através de um número exponencial de processadores.

1.5.2. Como implementar uma função em um computador quântico?

O primeiro passo para desenvolver um algoritmo quântico que resolve o problema acima é a implementação da função f. Como f é uma função booleana cuja tabela verdade tem uma única linha com saída 1, f pode ser implementada com uma porta Toffoli ge-

neralizada ativada pelo x_0, como descrito no final da subseção 1.3.6. Esta porta tem uma matriz unitária associada que denominamos F_{x_0}, que é definida pela sua atuação na base computacional como

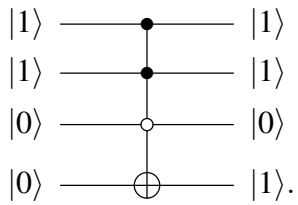
$$F_{x_0}|x\rangle|i\rangle = |x\rangle|i \oplus f(x)\rangle,$$

onde x é uma cadeia de n bits e i é um bit. O conjunto dos n primeiros qubits é chamado de primeiro registrador e o último qubit de segundo registrador. Note que se tomarmos $i = 0$, a equação acima se reduz à

$$F_{x_0}|x\rangle|0\rangle = \begin{cases} |x_0\rangle|1\rangle, & \text{se } x = x_0, \\ |x\rangle|0\rangle, & \text{caso contrário,} \end{cases}$$

que descreve o que a porta Toffoli generalizada faz quando ativada por x_0 (e somente por x_0). O resultado do cálculo de $f(x)$ é armazenado no segundo registrador enquanto que o valor do primeiro registrador fica inalterado.

Por exemplo, o circuito que implementa F_{x_0} no caso $N = 8$ e $x_0 = 6$, isto é, $f(110) = 1$ e $f(j) = 0$ se $j \neq 110$, é



Os três primeiros qubits formam o primeiro registrador e o quarto qubit é o segundo registrador. Note que o estado do segundo registrador só muda de $|0\rangle$ para $|1\rangle$ se a entrada do primeiro registrador for $|110\rangle$, pois a cadeia de bits 110 ativa os 3 controles e qualquer outra cadeia de 3 bits não ativa.

No algoritmo de Grover, o segundo registrador está sempre no estado

$$|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

Usando a linearidade, a atuação da matriz F_{x_0} é dada por

$$F_{x_0}|x\rangle|-\rangle = \begin{cases} -|x_0\rangle|-\rangle, & \text{se } x = x_0, \\ |x\rangle|-\rangle, & \text{caso contrário.} \end{cases}$$

Neste caso, a atuação da matriz F_{x_0} não altera o estado do segundo registrador.

1.5.3. Descrição do algoritmo

O algoritmo de Grover usa uma matriz adicional definida como

$$G = \left(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\right) \otimes I_2,$$

onde

$$|\mathrm{d}\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle.$$

A notação "$|\mathrm{d}\rangle\langle\mathrm{d}|$" é chamada de *produto externo* entre o vetor $|\mathrm{d}\rangle$ (matriz $N \times 1$) e o vetor dual $\langle\mathrm{d}|$ (matriz $1 \times N$). O produto externo é equivalente ao produto matricial. A multiplicação de uma matriz $N \times 1$ por uma matriz $1 \times N$ resulta em uma matriz $N \times N$. Portanto, $|\mathrm{d}\rangle\langle\mathrm{d}|$ é uma matriz $N \times N$, dada por

$$|\mathrm{d}\rangle\langle\mathrm{d}| = \frac{1}{N}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix},$$

e a matriz G é o produto tensorial da matriz

$$\left(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\right) = \frac{1}{N}\begin{bmatrix} (2-N) & 1 & \cdots & 1 \\ 1 & (2-N) & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & (2-N) \end{bmatrix}$$

por I_2. A matriz $\left(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\right)$ é chamada matriz de Grover.

Estamos prontos para descrever o algoritmo:

Algoritmo 1: Algoritmo de Grover

Input: um inteiro N e uma função $f : \{0,...,N-1\} \to \{0,1\}$ tal que $f(x) = 1$ somente para um ponto $x = x_0$ no domínio.

Output: com probabilidade igual ou maior que $1 - \frac{1}{N}$, retorna x_0.

1 Prepare o estado inicial $|\mathrm{d}\rangle|-\rangle$;

2 Aplique $(GF_{x_0})^t$, onde $t = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$;

3 Faça uma medição do primeiro registrador na base computacional.

1.5.4. Circuito (não-econômico) do algoritmo de Grover

Para obter um circuito do algoritmo de Grover, temos que fazer uma manipulação algébrica com a expressão da matriz de Grover $\left(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\right)$. Note que

$$|\mathrm{d}\rangle = H^{\otimes n}|0\rangle$$

onde $|0\rangle$ está na notação decimal e $H^{\otimes n} = H \otimes \cdots \otimes H$. Transpondo a equação acima, obtemos

$$\langle\mathrm{d}| = \langle 0|H^{\otimes n}.$$

Podemos verificar que $(H^{\otimes n}) \cdot (H^{\otimes n}) = (H \cdot H)^{\otimes n} = (I_2)^{\otimes n} = I_N$. Usando estes resultados, obtemos

$$\left(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\right) = H^{\otimes n}\left(2\,|0\rangle\langle 0| - I_N\right)H^{\otimes n},$$

onde

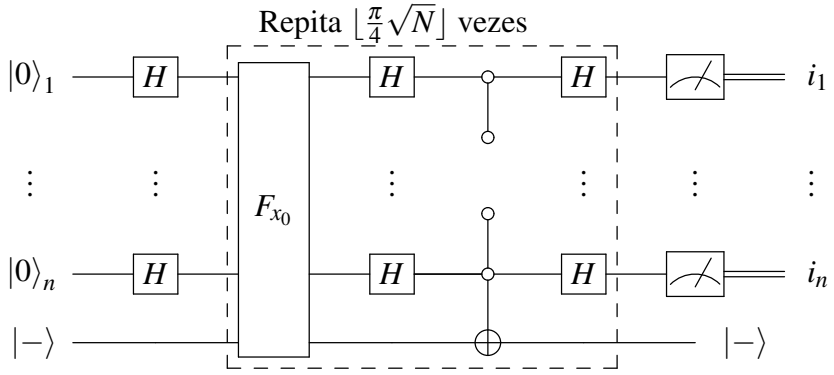
$$\left(2\,|0\rangle\langle 0| - I_N\right) = \begin{bmatrix} -1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}.$$

A matriz $\left(2|0\rangle\langle 0| - I_N\right)$ atua apenas no primeiro registrador. No entanto, é mais simples implementar esta matriz usando ambos os registradores. Vamos mostrar que ela é implementada por uma porta Toffoli generalizada ativada por n bits zero. De fato, a atuação da matriz $\left(2|0\rangle\langle 0| - I_N\right)$ em um estado $|x\rangle$ da base computacional do primeiro registrador, onde x é uma cadeia de n bits, é

$$\left(2|0\rangle\langle 0| - I_N\right)|x\rangle = \begin{cases} -|0\rangle, & \text{se } x = 0, \\ |x\rangle, & \text{caso contrário.} \end{cases}$$

Portanto, a atuação da matriz $\left(2|0\rangle\langle 0| - I_N\right)$ (no primeiro registrador) é igual à atuação da matriz F_{x_0} (em ambos registradores) descrita na subseção 1.5.2 quando $x_0 = 0$ e quando o segundo registrador está no estado $|-\rangle$.

Usando estes resultados algébricos discutidos acima, concluímos que um circuito para implementar o algoritmo de Grover é

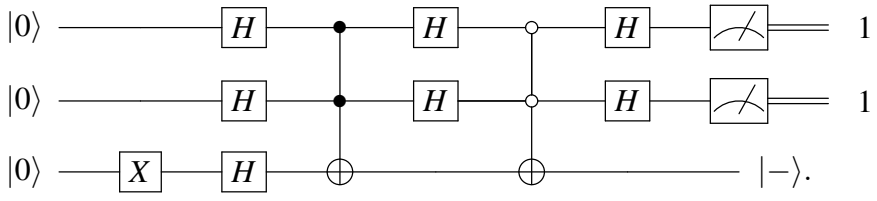


onde a saída i_1, ..., i_n são os bits de x_0, isto é $x_0 = (i_1 \ldots i_n)_2$.

A construção deste circuito nesta seção está seguindo a descrição padrão do algoritmo de Grover, porém este circuito pode ser reduzido, como veremos adiante na descrição econômica.

1.5.5. Implementação no *composer* da IBM

Para implementar o algoritmo de Grover no composer da IBM, vamos usar o caso $N = 4$ e $x_0 = 11$ como exemplo. Particularizando o circuito para este caso, obtemos



Note que a caixa pontilhada na descrição geral do algoritmo não precisa de repetições, pois $\lfloor \pi\sqrt{N}/4 \rfloor = 1$ para $N = 4$. A saída esperada é $i_1 i_2 = 11$.

A figura 1.10 mostra o circuito do algoritmo de Grover para $N = 4$ e $x_0 = 3$ (parte debaixo da figura) junto com a saída do ibmqx2 após 1024 repetições (parte de cima). O

programa *Qasm* deste circuito está no rodapé[11]. Abra um experimento novo no ibmqx2, mude para o editor *Qasm*, copie a nota de rodapé e cole a partir da quarta linha. É necessário também adicionar dois medidores nos qubits q[0] e q[1]. Podemos ver que o resultado com a maior probabilidade foi c[0]c[1]=11 mostrando que o valor de x_0 foi encontrado corretamente usando a matriz F_{x_0} uma única vez. No entanto, quando rodamos este circuito no simulador obtemos a saída c[0]c[1]=11 com 100% de probabilidade. Este é de fato o resultado correto. O resultado mostrado na figura 1.10 está muito degradado.

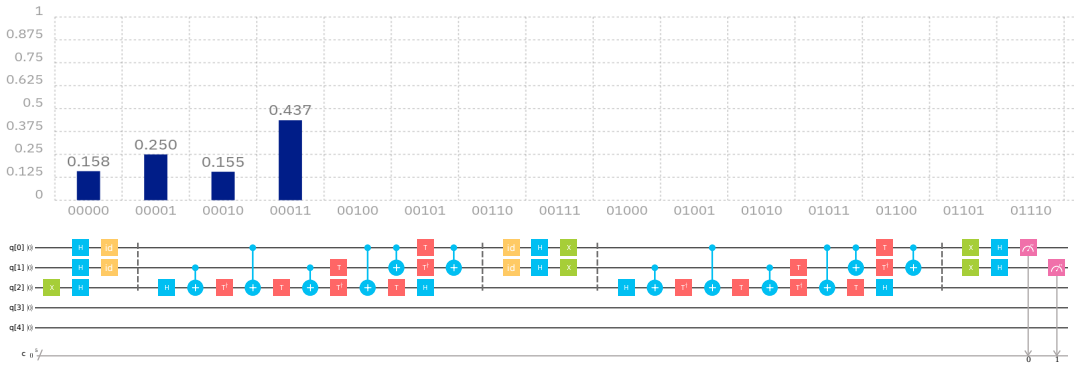


Figura 1.10. Circuito do algoritmo de Grover com $N = 4$ e $x_0 = 3$ no ibmqx2 (Reprint Courtesy of IBM Corporation ©)

Para mudar o valor de x_0 devemos conjugar o controles da primeira porta Toffoli (q[0] e q[1]) por X, isto é, colocar X no lugar das matrizes $\boxed{\text{id}}$ de maneira simétrica. Para $x_0 = 00$, devemos aplicar X tanto em q[0] como em q[1]. Para $x_0 = 01$, devemos aplicar X em q[1]. Para $x_0 = 10$, devemos aplicar X em q[0].

A figura 1.11 mostra a implementação do algoritmo de Grover no *backend* ibmqx4. A saída do algoritmo é dada em c[2]c[1], pois no ibmqx4 temos que inverter o circuito por causa dos CNOTS. O resultado é similar ao resultado do ibmqx2. É possível melhorar a fidelidade destes resultados com o resultado obtido pelo simulador usando uma implementação mais econômica do algoritmo de Grover, como discutido adiante.

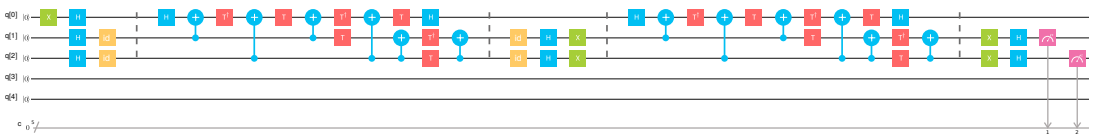


Figura 1.11. Circuito do algoritmo de Grover com $N = 4$ e $x_0 = 3$ no ibmqx4 (Reprint Courtesy of IBM Corporation ©)

1.5.6. Análise do algoritmo

Por que o algoritmo de Grover funciona corretamente? Vamos responder esta pergunta usando uma interpretação geométrica de reflexões de vetores. O objetivo é achar x_0,

[11] x q[2]; h q[0]; h q[1]; h q[2]; id q[0]; id q[1]; barrier q[0],q[1],q[2]; h q[2]; cx q[1],q[2]; tdg q[2]; cx q[0],q[2]; t q[2]; cx q[1],q[2]; t q[1]; tdg q[2]; cx q[0],q[2]; cx q[0],q[1]; t q[2]; t q[0]; tdg q[1]; h q[2]; cx q[0],q[1]; barrier q[0],q[1],q[2]; id q[0]; id q[1]; h q[0]; h q[1]; x q[0]; x q[1]; barrier q[0],q[1],q[2]; h q[2]; cx q[1],q[2]; tdg q[2]; cx q[0],q[2]; t q[2]; cx q[1],q[2]; t q[1]; tdg q[2]; cx q[0],q[2]; cx q[0],q[1]; t q[2]; t q[0]; tdg q[1]; h q[2]; cx q[0],q[1]; barrier q[0],q[1],q[2]; x q[0]; x q[1]; h q[0]; h q[1];

que é uma cadeia de n bits. Na computação quântica, o objetivo é colocar o estado do computador no estado $|x_0\rangle$, pois as medições de cada qubit retornam neste caso os bits de x_0.

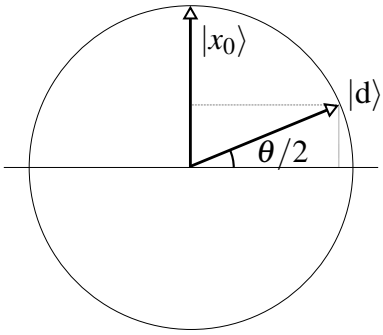


Figura 1.12. Descrição dos vetores $|x_0\rangle$ e $|\mathbf{d}\rangle$

No início do algoritmo, o estado do computador é $|\mathrm{d}\rangle$. Para N grande, $|\mathrm{d}\rangle$ é quase ortogonal a $|x_0\rangle$. Uma medição neste ponto tem poucas chances de retornar x_0. A figura 1.12 mostra a posição destes vetores e o ângulo $\theta/2$ que $|\mathrm{d}\rangle$ faz com o eixo horizontal. Qualquer outra posição dos vetores desde que $|\mathrm{d}\rangle$ seja quase ortogonal a $|x_0\rangle$ serve na análise. O ângulo θ é muito pequeno para N grande, e neste caso, $\theta/2$ é uma aproximação muito boa de $\sin(\theta/2)$. Além disso, o seno de um ângulo é igual ao cosseno do complemento, isto é,

$$\frac{\theta}{2} \approx \sin\frac{\theta}{2} = \cos\left(\frac{\pi}{2} - \frac{\theta}{2}\right).$$

Como $(\pi-\theta)/2$ é o ângulo entre $|x_0\rangle$ e $|\mathrm{d}\rangle$, pela definição do produto interno, $\cos(\pi-\theta)/2$ é o produto interno de $|x_0\rangle$ e $|\mathrm{d}\rangle$, cujo resultado é

$$\frac{\theta}{2} \simeq \sin\frac{\theta}{2} = \cos\left(\frac{\pi}{2} - \frac{\theta}{2}\right) = \langle x_0|\mathrm{d}\rangle = \frac{1}{\sqrt{N}}.$$

Portanto,

$$\theta \approx \frac{2}{\sqrt{N}}.$$

Vamos desconsiderar o estado do segundo registrador, pois ele permanece fixo em $|-\rangle$ durante todo o algoritmo. Por isso, vamos definir a matriz

$$f_{x_0}|x\rangle = \begin{cases} -|x_0\rangle, & \text{se } x = x_0, \\ |x\rangle, & \text{caso contrário,} \end{cases}$$

que é uma versão reduzida de F_{x_0}. Vamos usar f_{x_0} no lugar de F_{x_0} na análise do algoritmo. O primeiro passo é aplicar f_{x_0} em $|\mathrm{d}\rangle$. A ação de f_{x_0} em $|\mathrm{d}\rangle$ (escrito na base computacional) inverte o sinal da amplitude de $|x_0\rangle$ e não altera as outras amplitudes. A amplitude de $|x_0\rangle$ é a linha pontilhada vertical da figura 1.12, que é invertida pela ação de f_{x_0}. Geometricamente, a ação de f_{x_0} é representada por uma reflexão de $|\mathrm{d}\rangle$ em torno do eixo horizontal. O ângulo entre os vetores $|\mathrm{d}\rangle$ e $(f_{x_0}|\mathrm{d}\rangle)$ é θ, conforme mostra a figura 1.13.

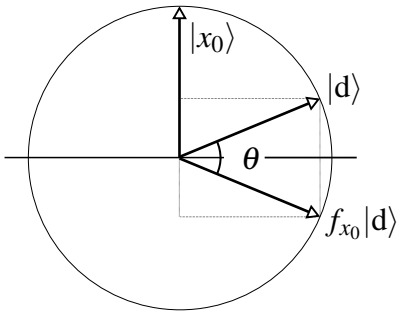


Figura 1.13. O vetor $f_{x_0}|\mathbf{d}\rangle$ é a reflexão de $|\mathbf{d}\rangle$ em relação ao eixo horizontal

O próximo passo é aplicar $\left(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\right)$. Vamos chamar esta matriz de g, pois ela é uma versão reduzida com N dimensões de G, ou seja,

$$g = 2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N.$$

Agora vamos mostrar que a ação da matriz g é uma reflexão em torno do eixo que passa pela origem na direção $|\mathrm{d}\rangle$. A prova disto é feita em duas etapas. Primeiro, mostramos que a ação de g em $|\mathrm{d}\rangle$ não muda a direção de $|\mathrm{d}\rangle$. Segundo, mostramos que a ação de g em $\left|\mathrm{d}^{\perp}\right\rangle$ inverte o sinal de $\left|\mathrm{d}^{\perp}\right\rangle$, onde $\left|\mathrm{d}^{\perp}\right\rangle$ é um vetor ortogonal a $|\mathrm{d}\rangle$. A primeira etapa é consequência da conta

$$g|\mathrm{d}\rangle = \left(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\right)|\mathrm{d}\rangle = 2\,|\mathrm{d}\rangle\left\langle\mathrm{d}\middle|\mathrm{d}\right\rangle - |\mathrm{d}\rangle = |\mathrm{d}\rangle,$$

pois $\left\langle\mathrm{d}\middle|\mathrm{d}\right\rangle = 1$. A segunda etapa é consequência da conta

$$g\left|\mathrm{d}^{\perp}\right\rangle = \left(2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N\right)\left|\mathrm{d}^{\perp}\right\rangle = 2\,|\mathrm{d}\rangle\left\langle\mathrm{d}\middle|\mathrm{d}^{\perp}\right\rangle - \left|\mathrm{d}^{\perp}\right\rangle = -\left|\mathrm{d}^{\perp}\right\rangle,$$

pois $\left\langle\mathrm{d}\middle|\mathrm{d}^{\perp}\right\rangle = 0$.

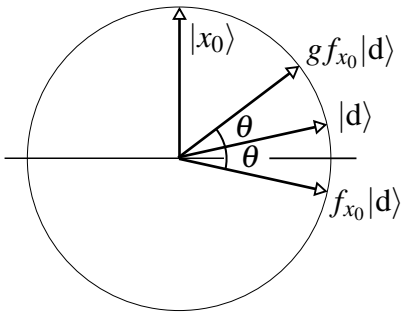


Figura 1.14. O vetor $(g\,f_{x_0}|\mathbf{d}\rangle)$ é a reflexão de $(f_{x_0}|\mathbf{d}\rangle)$ em torno de $|\mathbf{d}\rangle$

A figura 1.14 mostra onde está o vetor $(g \cdot f_{x_0} \cdot |\mathrm{d}\rangle)$. Isto mostra que a ação de $(G \cdot F_{x_0})$ rodou o estado inicial de θ graus em direção ao vetor $|x_0\rangle$. Como θ é um ângulo pequeno, este resultado é modesto, porém promissor. É fácil verificar que a segunda ação de $(G \cdot F_{x_0})$ também roda de novo o estado do computador quântico de θ graus em direção

ao vetor $|x_0\rangle$. Queremos saber quantas iterações r são necessárias tal que $r\theta = \pi/2$. O número de iterações é

$$r = \left\lfloor \frac{\pi}{2\theta} \right\rfloor = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor .$$

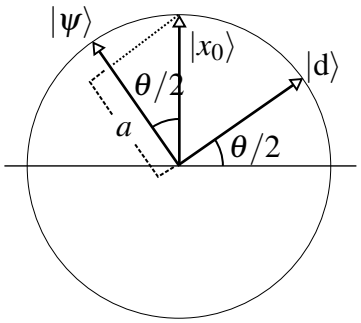


Figura 1.15. O vetor $|\psi\rangle$ é o estado final do computador quântico e a é a projeção de $|x_0\rangle$ no estado final. O ângulo entre $|\psi\rangle$ e $|x_0\rangle$ é menor ou igual a $\theta/2$

O final da análise é o cálculo da probabilidade de sucesso. Após r iterações, o estado do computador quântico sem considerar o segundo registrador é

$$|\psi\rangle = (g\, f_{x_0})^{\left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor} |\mathrm{d}\rangle.$$

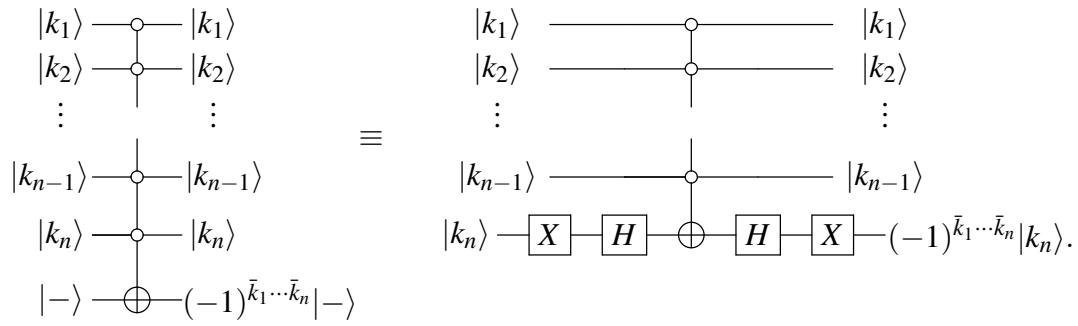
O vetor $|\psi\rangle$ é quase ortogonal a $|\mathrm{d}\rangle$, como descrito na figura 1.15. O ângulo entre $|\psi\rangle$ e $|x_0\rangle$ é menor ou igual a $\theta/2$. A probabilidade de sucesso é maior ou igual do que o módulo ao quadrado da amplitude de $|x_0\rangle$ na decomposição de $|\psi\rangle$ na base computacional. Esta amplitude está mostrada na figura 1.15 através da letra a. A projeção de $|x_0\rangle$ no estado final é no máximo $\cos(\theta/2)$. Portanto, a probabilidade de sucesso $p = |a|^2$ satisfaz

$$p \;\geq\; \cos^2\frac{\theta}{2} \;\geq\; 1 - \sin^2\frac{\theta}{2} \;\geq\; 1 - \frac{1}{N}.$$

O caso $N = 4$ é especial, pois como $\sin(\theta/2) = 1/\sqrt{N}$, segue que $\theta = 60^\circ$. Com uma aplicação de $(g\, f_{x_0})$, o vetor $|\mathrm{d}\rangle$ gira de 60° e coincide com o vetor $|x_0\rangle$. Neste caso, a probabilidade de sucesso é exatamente $p = 1$. Isto mostra que o resultado do ibmqx2 na figura 1.10 está muito degradado enquanto que o resultado do ibmqx4 na figura 1.11 está um pouco melhor.

1.5.7. Implementação usando um circuito econômico

O segundo registrador do algoritmo de Grover pode ser descartado e é possível fazer uma implementação mais econômica fugindo do modelo padrão descritos nos livros e seguindo publicações relacionadas com o Qiskit [24]. Primeiro vamos mostrar a equivalência dos seguintes circuitos:



Note que os circuitos não têm o mesmo número de qubits. Portanto, a equivalência só pode ser mostrada se eliminarmos o segundo registrador (último qubit) do primeiro circuito e antes disto movermos o sinal $(-1)^{\bar{k}_1\cdots\bar{k}_n}$ para o primeiro registrador. Isto é permitido pela seguinte propriedade do produto de Kronecker:

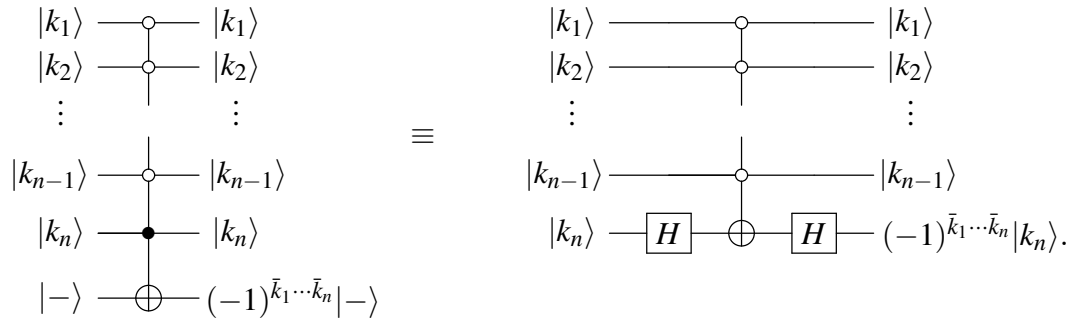
$$|v_1\rangle \otimes (a|v_2\rangle) = (a|v_1\rangle) \otimes |v_2\rangle = a(|v_1\rangle \otimes |v_2\rangle),$$

que é válida para quaisquer vetores $|v_1\rangle$, $|v_2\rangle$ e qualquer constante a. Portanto, a saída do primeiro circuito após a eliminação do segundo registrador é

$$(-1)^{\bar{k}_1\cdots\bar{k}_n}|k_1\rangle \otimes \cdots \otimes |k_n\rangle.$$

No segundo circuito, se $|k_n\rangle = |0\rangle$, a primeira porta X gera o estado $|1\rangle$ e a primeira porta H cria (temporariamente) o estado $|-\rangle$, que inverte o sinal sob ação da porta Toffoli generalizada se os qubits de 0 a $n-1$ forem ativados. Na continuação, a segunda porta H desfaz o estado $|-\rangle$ convertendo para $|1\rangle$ e após a segunda porta X a saída é $(-|0\rangle)$ ou $|0\rangle$ dependendo se os $(n-1)$ primeiros qubits foram ativados ou não. Se $|k_n\rangle = 1$, a primeira porta X gera o estado $|0\rangle$ e a primeira porta H cria o estado $|+\rangle$, que não se altera sob a ação da porta Toffoli generalizada. Na continuação, a segunda porta H converte $|+\rangle$ em $|0\rangle$ e a saída é $|1\rangle$ independentemente se a porta Toffoli generalizada foi ativada ou não. O resultado é o mesmo em comparação com o resultado do primeiro circuito (após a eliminação do segundo registrador). Uma vez que vamos fazer uma medição do primeiro registrado apenas, isto completa a prova de que podemos substituir *sem nenhuma perda* o primeiro circuito pelo segundo no algoritmo de Grover. Esta substituição não é válida em outros algoritmos.

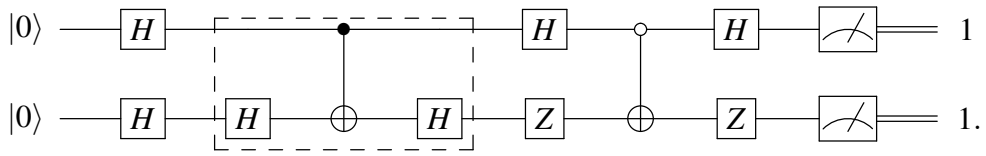
Se o n-ésimo qubit for ativado pelo 1, a equivalência dos circuito fica descrita da seguinte maneira:



A verificação da equivalência é similar ao caso anterior. Portanto, a matriz F_{x_0} também pode ser reduzida de $(n+1)$ para n qubits introduzindo duas portas H de forma conjugada

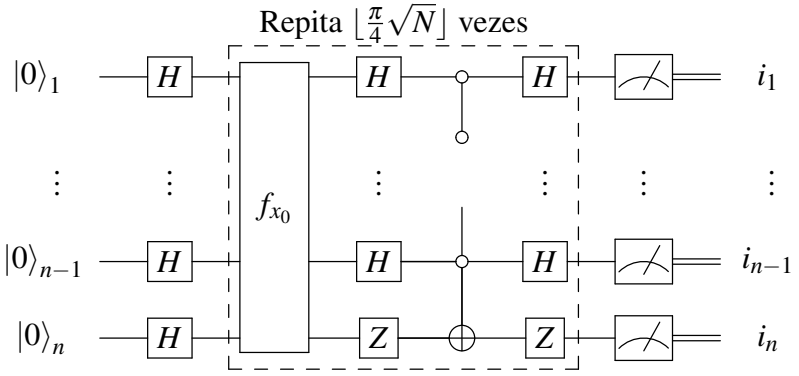
se o n-ésimo qubit for ativado pelo 1 e introduzindo duas portas H e duas portas X de forma conjugada se o n-ésimo qubit for ativado pelo 0. Os $(n-1)$ primeiros qubits podem ser ativados por 0 ou 1 e nada se altera para eles.

O circuito do algoritmo de Grover na forma econômica quando $N = 4$ e $x_0 = 11$ é

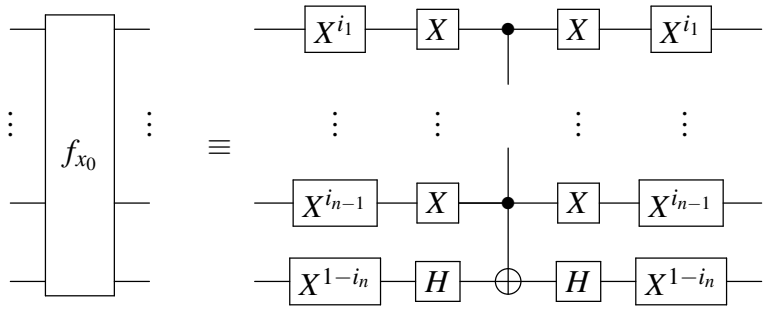


Para simplificar o circuito, substituímos as sequências de portas HXH no segundo qubit do circuito por Z. A parte do circuito dentro da caixa tracejada é escolhida pelo oráculo. O objetivo do algoritmo de Grover é desvendar o valor de x_0 consultando o oráculo, ou seja, usando a caixa tracejada, sem ver os detalhes da implementação. Temos que fingir que a caixa tracejada é uma caixa preta. No caso $N = 4$, existem 4 possíveis caixas tracejadas. Exibimos o caso $x_0 = 11$. É um bom exercício exibir os circuitos que o oráculo usa quando $x_0 = 00$, $x_0 = 01$ e $x_0 = 10$.

Para N genérico, o circuito do algoritmo de Grover na forma econômica com n qubits é



onde a matriz f_{x_0} é a forma reduzida de F_{x_0} e o circuito para f_{x_0} para $x_0 = (i_1 \dots i_n)_2$ genérico é



Na próxima seção, vamos mostrar como implementar o algoritmo de Grover de forma econômica usando o Qiskit.

1.6. Programando os computadores quânticos da IBM

Nesta seção, vamos mostrar como programar os computadores quânticos IBM. Já mostramos exemplos nas seções anteriores de uso dos computadores ibmqx4 de Tenerife e ibmqx2 de Yorktown, ambos de 5 qubits, usando o *composer* da página da IBM Q Experience[12]. O *composer* do IBM Q 5 é muito simples, pois podemos pegar as portas lógicas disponibilizadas e arrastar para o circuito. Porém, o *composer* não é útil para programas grandes. Neste caso, podemos usar a linguagem Qasm (*Quantum Assembly Language*) [10] cujos comandos podem ser editados por meio de um editor de textos usual. Além disto, as outras máquinas disponibilizadas pela IBM, como IBM Q 14 e 20, não possuem o *composer* e exigem a utilização um kit de desenvolvimento de software, chamado Qiskit[13], que é baseado na linguagem Python. Nas próxima seções, vamos descrever como usar e fazer programas através do Qiskit. Os comandos do Qiskit são baseados no modelo de circuitos e são muito semelhantes à linguagem Qasm. Em particular, vamos mostrar como como implementar de forma eficiente o algoritmo de Grover para 2 qubits, isto é, $N = 4$.

1.6.1. Qasm

Qasm é uma linguagem de baixo nível — do tipo *assembly* — desenvolvida pela IBM para representar os circuitos executáveis no IBM Q Experience. Programas em Qasm podem ser escritos de diversas formas. Sempre que um circuito é criado por meio do *composer* um código Qasm é gerado automaticamente. O código gerado pode ser visualizado e editado diretamente no browser, clicando-se em *Switch to Qasm Editor*. Na figura 1.16, temos o mesmo circuito já apresentado na figura 1.4, porém dessa vez visualizando o código Qasm correspondente.

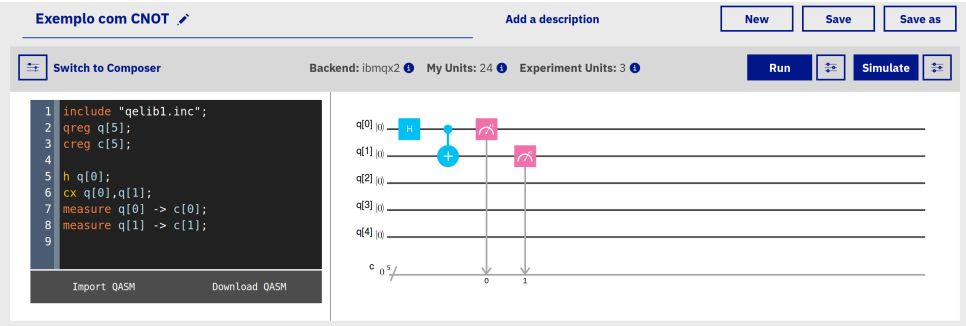


Figura 1.16. Circuito usando as portas H e CNOT, exibindo código-fonte Qasm equivalente ao circuito (Reprint Courtesy of IBM Corporation ©)

Toda linha em Qasm que inicie com duas barras (`//`) são ignoradas e, portanto, servem como comentário. A primeira linha diferente de comentário em qualquer código Qasm deve necessariamente ser o comando `OPENQASM` seguido pela versão do Qasm que está sendo utilizada. Observe que esse comando não aparece na figura 1.16. De fato, essa linha não costuma ser exibida nos editor de Qasm do IBM Q Experience, mas está sempre lá implicitamente — para confirmar, clique em *Download QASM* e em seguida

[12] `https://quantumexperience.ng.bluemix.net/qx/editor`
[13] `https://www.doi.org/10.5281/zenodo.2562110`

abra o arquivo em qualquer editor de textos. Note que todos os comandos na linguagem Qasm devem encerrar com ponto-e-vírgula.

O comando `include`, como o próprio nome já diz, permite incluir código de um arquivo externo. Normalmente incluímos pelos menos o arquivo `qelib1.inc`, pois este contém muitas definições úteis. Depois do `include`, outros comandos que são praticamente obrigatórios nos códigos Qasm são `qreg` e `creg`, utilizados respectivamente para declarar um registrador quântico e um clássico. Portanto, o comando `qreg q[5]` serve para declarar que nosso programa utiliza um registrador quântico de 5 qubits, e este registrador será referenciado no código por meio da variável `q`. Analogamente, o comando `creg c[5]` declara um registrador clássico, importante para armazenar o resultado da medição no final do algoritmo.

Incluir uma porta lógica por meio da linguagem Qasm é bastante intuitivo. Basta digitar o nome da porta e em seguida o qubit sobre o qual ela deve atuar. Por exemplo, o comando `h q[0]` significa que a porta de Hadamard deve ser aplicada no primeiro qubit (`q[0]`). Note que os qubits são numerados a partir de zero! Se o arquivo `qelib1.inc` foi incluído no início, então as principais portas — em particular, todas as portas disponíveis no *composer* — podem ser referenciadas diretamente: `h` é a porta de Hadamard (H), `x` é uma das portas de Pauli (X), e assim sucessivamente. Para incluir a porta $T^\dagger$ utiliza-se o comando `tdg`, e para incluir a porta $S^\dagger$ utiliza-se o comando `sdg`, onde `dg` se refere a *dagger*. No caso da porta CNOT, utiliza-se o comando `cx`, e nesse caso deve-se indicar tanto o qubit de controle como o qubit alvo — por exemplo, `cx q[0], q[1]` representa uma porta CNOT controlada pelo primeiro qubit e com alvo no segundo qubit.

Para efetuar uma medição, utiliza-se o comando `measure` seguido do registrador quântico a ser medido e do registrador clássico onde o resultado deve ser armazenado. Por exemplo, comando `measure q[0] -> c[0]` representa uma medição do primeiro qubit, com o resultado a ser armazenado no primeiro bit clássico.

Em vez de utilizar o *composer*, pode-se também editar um programa Qasm diretamente em outro editor de textos, e depois copiar o código-fonte para a página do IBM Q Experience. Ao clicar em *Switch to Composer*, o circuito correspondente é mostrado, e então pode-se passar a editá-lo do modo visual, arrastando as portas quânticas. O método de editar o programa é bastante conveniente ao escrever circuitos grandes.

Finalmente, o código Qasm pode também ser gerado implicitamente usando linguagens de alto nível. A partir da próxima seção, veremos como o Qiskit pode ser utilizado para escrever programas quânticos usando linguagem Python.

1.6.2. Qiskit

Qiskit é um kit de desenvolvimento de software de código aberto, desenvolvido pela IBM visando facilitar a programação dos computadores quânticos já existentes. Ele pode ser obtido gratuitamente no site oficial do projeto (`https://qiskit.org`) e pode ser usado em Linux, MacOS e Windows. Para instalar o Qiskit, é necessário em primeiro lugar ter o Python instalado na versão 3.5 ou mais recente. O Python normalmente já vem instalado no Linux e no MacOS. Usuários de Windows podem obter gratuitamente o Python no site oficial (`https://www.python.org`), ou por meio

de outras distribuições voltadas para o ambiente científico — um bom exemplo é o Anaconda Python, que pode ser obtido gratuitamente em `https://www.anaconda.com/distribution`.

A forma mais simples de instalar o Qiskit é usando o instalador de pacotes do Python, o pip (`https://pip.pypa.io`). O pip costuma vir instalado nas distribuições mais recentes do Python. Nesse caso, basta usar o comando `pip install qiskit`. Alguns pacotes adicionais de visualização e de algoritmos quânticos avançados só são instalados com o comando mais completo, `pip install qiskit[visualization] qiskit-aqua`. Às vezes, quando há duas versões diferentes do Python instaladas no mesmo computador, é necessário usar o comando pip3 em vez do comando pip. Nesse caso, o comando seria `pip3 install qiskit`.

É recomendável ter instalado também o Jupyter Notebook, em especial durante o aprendizado. O Jupyter Notebook permite programar em Python diretamente no browser, permite a inclusão de textos explicativos com formatação e facilita a visualização dos resultados. Para instalar o Jupyter pode-se também usar o comando `pip install jupyter`. Para abrir o Jupyter, pode-se ir ao terminal de comando e digitar o comando `jupyter notebook`. Com isso, o Jupyter é aberto no browser padrão. Pode-se navegar pela estrutura de pastas do computador e abrir arquivos já existentes com a extensão `ipynb` (chamados de *notebooks*). Pode-se também criar novos notebooks clicando no botão *New* e em seguida escolhendo Python 3. Para mais detalhes, pode-se consultar o site oficial, `https://jupyter.org`.

A proposta dos desenvolvedores do Qiskit é permitir que qualquer pessoa com conhecimentos básicos de computação quântica e da linguagem de programação Python possa programar os computadores quânticos da IBM. Para executar os programas escritos com Qiskit diretamente nos computadores da IBM é necessário utilizar a *API Token* que é obtida no site do IBM Q Experience, clicando em *My account* e depois em *Advanced*. Além de permitir a execução diretamente em computadores quânticos reais, o Qiskit facilita também a execução de experimentos em simuladores clássicos utilizando recursos de computação de alto desempenho.

O Qiskit é composto por quatro elementos — Terra, Aer, Ignis e Aqua —, dos quais utilizaremos os dois primeiros. O elemento Terra contém todos os fundamentos utilizados para escrever programas quânticos segundo o modelo de circuitos, e o elemento Aer possui recursos para simulação por meio de computação (clássica) de alto desempenho. Os elementos Ignis e Aqua contém recursos mais avançados e fogem do escopo deste curso.

1.6.2.1. Escrevendo um programa quântico básico

Para começar a utilizar o Qiskit Terra, vamos abrir a interface do Jupyter Notebook, como exemplificado na Figura 1.17. Devemos incluir as seguintes linhas no início do código para indicar ao Python quais comandos iremos utilizar:

```
from qiskit import QuantumCircuit
```

```
from qiskit import ClassicalRegister
from qiskit import QuantumRegister
```

Exemplos de *notebooks* relacionados com este tutorial podem ser baixados da conta *programaquantica*[14] do GitHub.

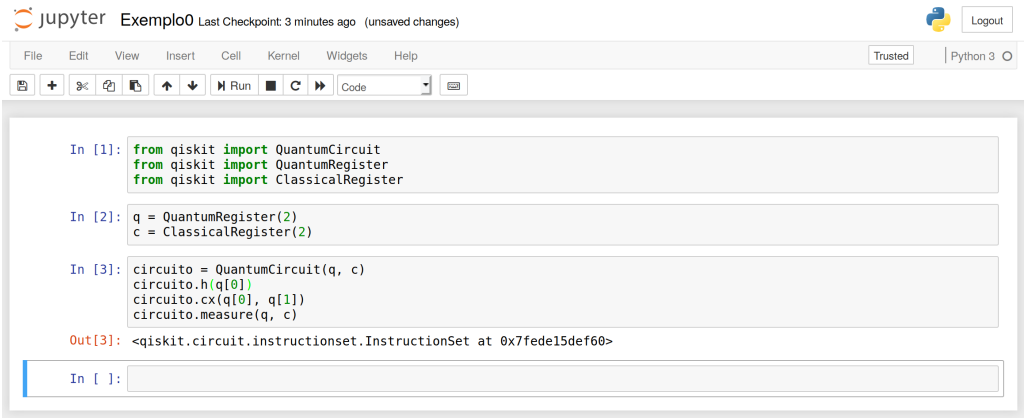


Figura 1.17. Exemplo de utilização do Jupyter Notebook

Os nomes dos módulos importados são bastante sugestivos. Evidentemente, para escrever um programa quântico no modelo de circuitos nosso primeiro passo deve ser definir um circuito! E para definir um circuito quântico, precisamos antes dizer quantos bits quânticos e clássicos ele deve ter. No final, claro, desejamos executar nosso circuito. Vejamos então a sequência de comandos típica para um programa básico no Qiskit. Digamos, por exemplo, que nosso circuito deva ter 2 qubits, como na figura 1.16. Nessa caso, devemos incluir a seguinte linha:

```
q = QuantumRegister(2)
```

Agora, se também vamos precisar de 2 bits clássicos para ler o resultado final do circuito, então devemos incluir a seguinte linha:

```
c = ClassicalRegister(2)
```

Nos exemplos acima, os nomes `q` e `c` são apenas variáveis, e podem ser definidos livremente pelo usuário. Opcionalmente, é possível definir também um apelido para os registradores, para serem utilizados nas visualizações do Qiskit. Por exemplo:

```
q = QuantumRegister(2, 'qubit')
c = ClassicalRegister(2, 'bit')
```

Uma vez que tenhamos nossos registradores, já podemos definir nosso circuito quântico. O comando para isso é o seguinte:

```
circuito = QuantumCircuit(q, c)
```

No exemplo acima, o nome `circuito` é apenas uma variável, e pode ser definido livremente pelo usuário.

[14] `https://github.com/programaquantica`

Nosso circuito quântico ainda está vazio. Precisamos incluir portas, medições etc. Para incluir uma porta quântica de Hadamard que atue no primeiro qubit, por exemplo, usamos o seguinte comando:

```
circuito.h(q[0])      # Hadamard
```

Outras portas de 1 qubits também estão disponíveis por meio de comandos semelhantes. Em particular, todas as portas elementares disponíveis no *composer* estão também disponíveis no Qiskit, com nomes autoexplicativos. Por exemplo, em vez da porta de Hadamard poderíamos aplicar uma das portas abaixo:

```
circuito.id(q[0])     # identidade
circuito.x(q[0])      # Pauli-X
circuito.y(q[0])      # Pauli-Y
circuito.z(q[0])      # Pauli-Z
circuito.s(q[0])      # S
circuito.sdg(q[0])    # transposto conjugado de S
circuito.t(q[0])      # T
circuito.tdg(q[0])    # transposto conjugado de T
```

Não podemos escrever algoritmos quânticos muito interessantes somente com portas atuando em 1 qubit. Precisamos pelo menos de uma porta que atue em 2 qubits, a saber, a porta CNOT. Uma porta CNOT controlada pelo primeiro qubit e com alvo no segundo qubit pode ser incluída no circuito com o seguinte comando do Qiskit:

```
circuito.cx(q[0], q[1])
```

Ao terminarmos de incluir as portas quânticas em nosso circuito, precisamos ainda efetuar uma medição. Foi por esse motivo que ao definirmos o circuito quântico tivemos de estabelecer não somente um registrador quântico mas também um registrador clássico. Agora, podemos indicar que desejamos efetuar uma medição em todos os três qubits do registrador quântico e armazenar os resultados nos três bits do registrador clássico! Para isso, usamos o seguinte comando:

```
circuito.measure(q, c)
```

Se quiséssemos, poderíamos ter medido somente um qubit, deixando os demais intactos. Por exemplo, para medir somente o primeiro qubit do registrador quântico e armazenar o resultado no primeiro bit do registrador clássico, usaríamos o seguinte comando:

```
circuito.measure(q[0], c[0])
```

Nesse ponto, já temos um circuito básico porém completo — com portas quânticas e medições. Para termos certeza de que fizemos tudo corretamente, é interessante visualizar o circuito. Podemos facilmente obter essa visualização no Qiskit usando o seguinte comando[15]:

[15]Caso esteja sendo usado o Jupyter Notebook, recomenda-se incluir antes uma linha com o comando `%matplotlib inline`, para indicar que a visualização deve ser exibida no próprio browser, junto com o código.

```
circuito.draw(output='mpl')
```

O resultado gerado pelo código acima corresponde à Figura 1.18. Em vez de `output='mpl'`, poderíamos ter `output='latex'`. O primeiro significa que a visualização será gerada usando internamente o Matplotlib, um pacote bastante completo para geração de gráficos em Python. Já o segundo significa que a visualização será gerada usando internamente o LaTeX, um sistema de preparação de documentos muito usado em matemática, física, computação e áreas semelhantes. De todo modo, essa escolha faz pouca diferença para o usuário. Não é necessário conhecer comandos de Matplotlib ou de LaTeX.

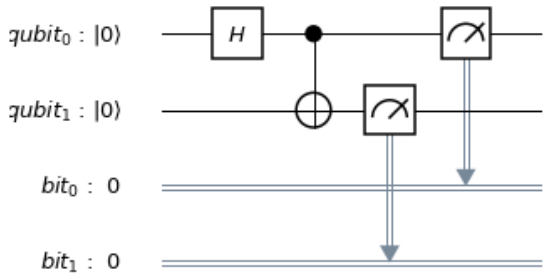


Figura 1.18. Visualização do circuito gerado pelo Qiskit

O Qiskit permite também obter diretamente o código-fonte Qasm para os circuitos quânticos gerados. Para isso, basta executar o seguinte comando:

```
codigo_qasm = circuito.qasm()
print(codigo_qasm)
```

No caso do circuito que estamos considerando em nosso exemplo, o código-fonte Qasm gerado é o seguinte:

```
OPENQASM 2.0;
include "qelib1.inc";
qreg qubit[2];
creg bit[2];
h qubit[0];
cx qubit[0],qubit[1];
measure qubit[0] -> bit[0];
measure qubit[1] -> bit[1];
```

1.6.2.2. Executando o programa quântico

Agora que temos um programa quântico completo, podemos executá-lo em um simulador ou em um computador quântico real. Utilizando somente os comandos que aprendemos até aqui, a forma mais imediata de executar o programa seria gerar o código Qasm no

Qiskit e em seguida copiá-lo no composer do IBM Q Experience. No entanto, veremos que ao utilizarmos o Qiskit, não precisamos mais do composer.

Para simular o circuito quântico, precisamos importar também o `BasicAer` e o `execute`. Também é útil importar uma ferramenta de visualização. Obtemos tudo isso por meio dos seguintes comandos:

```
from qiskit import BasicAer, execute
from qiskit.tools.visualization import *
```

Todos esses comandos import podem ficar agrupados numa única célula no início do notebook, juntamente com os que já havíamos introduzidos na seção anterior. O motivo de os estamos introduzindo gradativamente é puramente didático, para que o leitor compreenda em quais situações cada comando é necessário. Note que o Python também permite utilizar o asterisco para indicar que tudo o que houver em um determinado módulo deve ser importado.

Para simular o circuito, precisamos escolher um dos *backends* disponíveis no BasicAer. Para obter uma lista completa, usamos o comando `BasicAer.backends()`. Atualmente há três backends disponíveis no BasicAer: `qasm_simulator`, para simulação fiel ao funcionamento do computador quântico real; `statevector_simulator`, para simulação visando obter as amplitudes do estado final; e `unitary_simulator`, para obtenção da matriz unitária correspondente ao circuito.

Para executar o simulador no `qasm_simulator`, nosso circuito precisa ter pelo menos uma medição. Afinal, não se pode simular fielmente o funcionamento de um computador quântico se não houver alguma medição que gere resultado clássico. Para simular 1024 repetições do circuito em um computador quântico, por exemplo, usamos os seguintes comandos:

```
backend = BasicAer.get_backend('qasm_simulator')
job = execute(circuito, backend, shots=1024)
```

Agora todas as informações da simulação estão armazenadas no objeto `job`, mas o formato ainda é difícil para um humano ler. Podemos extrair o resultado e realizar uma contagem de todas as medições obtidas:

```
resultado = job.result()
contagem = resultado.get_counts()
```

O objeto `contagem` é um dicionário do Python — mas o leitor não precisa se preocupar com isso. Felizmente, o Qiskit já inclui algumas ferramentas de visualização muito úteis. Por exemplo, o comando `plot_histogram(contagem)` gera um histograma como da figura 1.19.

O backend `statevector_simulator` é utilizado de forma semelhante, por meio dos seguintes comandos:

```
backend = BasicAer.get_backend('statevector_simulator')
job = execute(circuito, backend)
resultado = job.result()
```

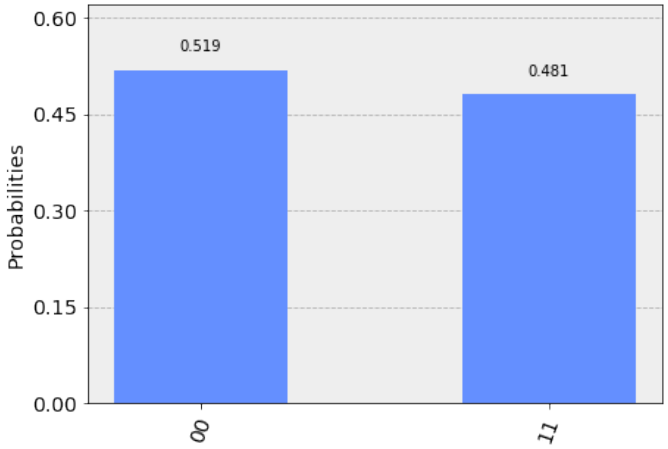


Figura 1.19. Visualização do resultado da simulação usando qasm_simulator

No entanto, ao ler o resultado estamos interessados em um vetor de estado, e não em uma contagem de resultados de medições. O comando que devemos utilizar aqui, portanto, é o seguinte:

```
estado = resultado.get_statevector()
```

Se imprimirmos o estado resultante com `print(resultado)`, veremos um *array* de números complexos. Este array pode ser manipulado de diversas formas dentro do Python. O Qiskit provê algumas visualizações úteis, como por exemplo o comando `plot_state_city(estado)`, que produz gráficos das partes real e imaginária da *matriz densidade* de um sistema, como na figura 1.20. Matrizes densidade são uma forma de representação dos estados de um sistema quântico.

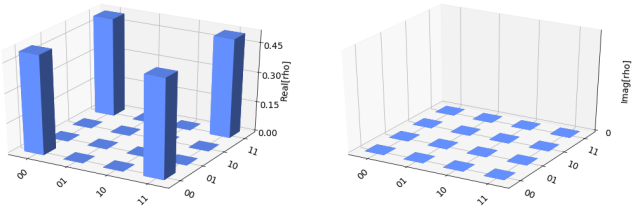


Figura 1.20. Visualização do resultado da simulação usando statevector_simulator

Para executar o simulador no `unitary_simulator`, nosso circuito não pode ter medições. Afinal, a medição não é uma operação unitária. Os comandos a serem executados nesse caso são os seguintes:

```
backend = BasicAer.get_backend('unitary_simulator')
job = execute(circuito, backend)
resultado = job.result()
```

O resultado dessa vez é uma matriz unitária, que pode ser extraída com o seguinte comando:

```
matriz = resultado.get_unitary()
```

A matriz resultante pode ser manipulada no próprio Python, usando pacotes como o *Numpy*.

Além dos backends de simulação mencionados acima, que são executados localmente no computador do usuário, o Qiskit também disponibiliza outros backends que direcionam os jobs para a própria IBM: a maioria deles direciona para computadores quânticos reais, e um direciona para ambiente de simulação de alto desempenho.

Para usar esses backends, é necessário acessar a conta no IBM Q Experience e copiar o API Token. Em seguida, deve-se substituir esse token no código abaixo:

```
from qiskit import IBMQ
IBMQ.save_account('Colar_Token_Aqui')
```

Isso fará com que um arquivo seja salvo no computador contendo informações necessárias para contactar os servidores da IBM. Basta executar uma vez em cada computador onde se queira usar o Qiskit. Depois disso, para acessar a conta IBM Q Experience a partir do Qiskit, será suficiente usar o comando `IBMQ.load_accounts()`. Para obter uma listagem completa dos backends disponíveis, usa-se o comando `IBMQ.backends()`. Atualmente há quatro backends disponíveis no IMBQ: `ibmqx4`, para executar os circuitos diretamente no computador quântico IBM Q 5 Tenerife, de 5 qubits; `ibmqx2`, para executar no IBM Q 5 Yorktown, de 5 qubits; `ibmq_16_melbourne`, para executar no IBM Q 14 Melbourne, de 14 qubits; e `ibmq_qasm_simulator`, para executar remotamente no simulador clássico de alto desempenho, com suporte para até 32 qubits.

Por exemplo, para executar o circuito no IBM Q 5 Yorktown com 1024 repetições, fazemos o seguinte:

```
maquina = IBMQ.get_backend('ibmqx2')
job = execute(circuito, maquina, shots=1024)
```

Com isso, o job entra na fila para ser executado no computador quântico. Note que a espera pode demorar muitas horas, e o notebook precisa permanecer aberto e com conexão à Internet. Para saber a posição do job na fila em tempo real, pode-se usar o seguinte código:

```
from qiskit.tools.monitor import job_monitor
job_monitor(job)
```

Quando o job termina de executar, podemos extrair e visualizar o resultado com comandos muito semelhantes aos que já estamos acostumados:

```
resultado = job.result()
contagem = resultado.get_counts()
plot_histogram(contagem)
```

1.6.3. Implementação do algoritmo de Grover

Agora que já sabemos como implementar circuitos quânticos básicos no Qiskit e como executá-los em um simulador ou em um computador quântico da IBM, podemos estudar um exemplo mais sofisticado. O algoritmo de Grover é ideal para isso. Já estudamos esse importante algoritmo de busca na seção anterior, e já implementamos uma versão dele usando Qasm. No entanto, como os computadores quânticos da IBM não implementam diretamente portas com dois ou mais controles, tivemos de fazer uma decomposição do operador de Grover que foi muito custosa.

Nesta subseção, iremos implementar uma versão melhorada do algoritmo de Grover para uma lista com $N = 4$ elementos. Dessa vez usaremos uma decomposição mais econômica do operador de Grover. Os computadores quânticos atuais ainda acumulam muitos erros quando executam longas sequências de operações, então para conseguir bons resultados é importante reduzirmos tanto quanto possível o número de portas quânticas em nossos circuitos. Para reformular o circuito de Grover, vamos seguir o roteiro da subseção 1.5.7.

Primeiro, temos que definir os registradores e o circuito quântico. Precisamos de dois qubits no registrador quântico e dois no registrador clássico.

```
q = QuantumRegister(2, 'qubit')
c = ClassicalRegister(2, 'bit')
circuito = QuantumCircuit(q, c)
```

Agora, introduzimos no circuito as portas que vão colocar os qubits em superposição uniforme:

```
circuito.h(q[0])
circuito.h(q[1])
```

Para analisar a evolução parcial do algoritmo, podemos executar o circuito no backend `statevector_simulator` já nesse ponto, antes mesmo de ter o circuito completo. Fazemos isso com comandos que já estudamos na seção anterior:

```
backend = BasicAer.get_backend('statevector_simulator')
job = execute(circuito, backend)
estado = job.result().get_statevector()
print(estado)
```

O resultado é o vetor `[0.5+0.j 0.5+0.j 0.5+0.j 0.5+0.j]`, onde j é a notação do Python para a unidade imaginária. Portanto, o vetor corresponde ao estado $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$.

A próxima etapa é a implementação do oráculo. Para facilitar a visualização, sugerimos usar o comando `barrier` entre uma etapa e outra. Esses comandos podem ser removidos antes de executar o circuito no computador real.

Para construir o oráculo, vamos seguir o roteiro da subseção 1.5.7. Se quisermos executar o circuito no IBM Q 5 Yorktown, podemos usar a seguinte sequência de comandos:

```
circuito.h(q[1])
circuito.cx(q[0], q[1])
circuito.h(q[1])
```

Simulando novamente o circuito parcial no `statevector_simulator`, obtemos o vetor `[ 0.5+0.j  0.5+0.j  0.5+0.j -0.5+0.j]`, que corresponde ao estado $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$. Portanto, somente o elemento procurado teve sua fase invertida.

Se preferirmos executar o circuito no IBM Q 5 Tenerife, é recomendável inverter a posição dos qubits para respeitar o mapa de conectividade do computador. Agora vamos começar a construir o operador de Grover. Esse operador pode ser implementado com a seguinte sequência de comandos:

```
circuito.h(q[0])
circuito.z(q[1])

circuito.x(q[0])
circuito.cx(q[0], q[1])
circuito.x(q[0])

circuito.h(q[0])
circuito.z(q[1])
```

Simulando novamente o circuito parcial no `statevector_simulator`, obtemos o vetor `[ 0.+0.j  0.+0.j  0.+0.j -1.+0.j]`, que corresponde ao estado $-|11\rangle$. Portanto, temos exatamente o elemento que estávamos procurando. Para outros valores de N, o resultado não seria exato, mas teria uma amplitude suficientemente alta para o estado procurado.

Finalmente, podemos medir os dois qubits que armazenam o elemento procurado e salvar o resultado no registrador clássico:

```
circuito.measure(q, c)
```

Agora, para visualizar o circuito, usamos os comandos que já aprendemos. O resultado está na figura 1.21.

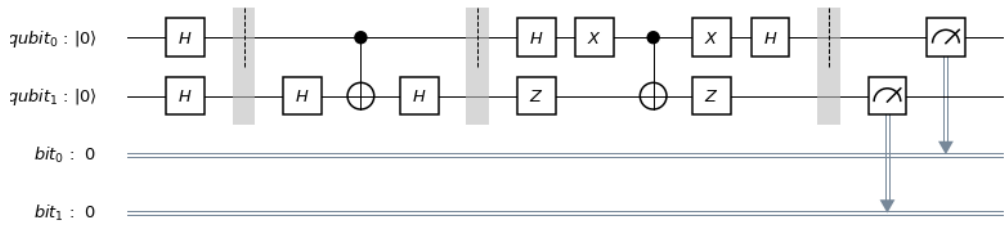


Figura 1.21. Circuito quântico para algoritmo de Grover implementado no Qiskit

O simulador `statevector_simulator` não funciona se houver uma medição, portanto não podemos mais executá-lo. No entanto, podemos executar o simu-

lador `qasm_simulator`, que nos dará 100% de medições resultando no estado $|11\rangle$. Se quisermos executar o circuito no computador quântico da IBM, usamos os seguintes comandos:

```
job = execute(circuito, backend=maquina, shots=1024)
res = job.result()
contagem = res.get_counts()
```

Como o computador quântico real não é perfeito, não devemos esperar que 100% das medições sejam do estado procurado. O resultado obtido em nosso teste foi o seguinte: dentre 1024 rodadas, o estado $|00\rangle$ foi obtido 22 vezes, o estado $|01\rangle$ foi obtido 58 vezes, o estado $|10\rangle$ foi obtido 114 vezes, e o estado $|11\rangle$ foi obtido 830 vezes. Portanto, obtivemos o elemento procurado em 81% das vezes. Na figura 1.22 temos o histograma correspondente.

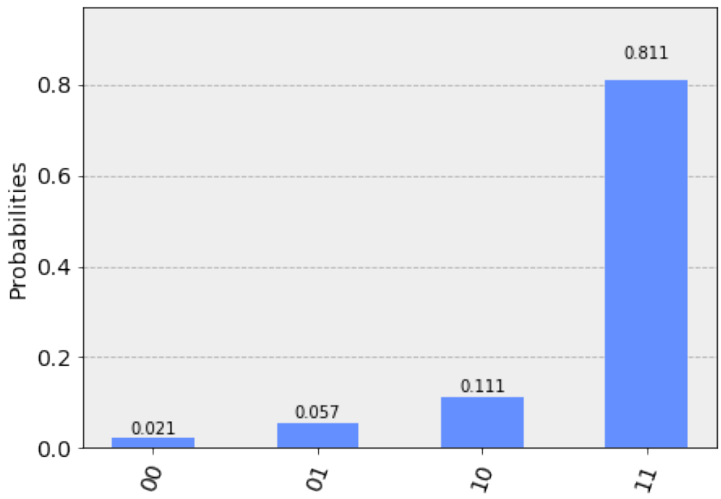


Figura 1.22. Resultado da execução do algoritmo de Grover no IBM Q 5 Yorktown

O leitor deve notar que o circuito dessa seção ficou bastante compacto, principalmente se comparado com a primeira versão do algoritmo de Grover que apresentamos. Isso só foi possível pois no final da seção anterior nos dedicamos a reescrever o circuito da forma mais eficiente possível. Essa busca pelo melhor circuito possível é, atualmente, a parte mais desafiadora na programação de computadores quânticos. Convém ressaltar que o Qiskit permite incluir portas bastante sofisticadas no circuito quântico, e já cuida das decomposições automaticamente. Se incluirmos, por exemplo, uma porta Toffoli, ou um CNOT que não respeite o mapa de conectividade do computador quântico, o Qiskit consegue "compilar" o circuito de modo que ele seja executável no hardware real. Em princípio, isso poderia simplificar bastante nosso trabalho de escrever programas em Qiskit, porém geraria circuitos muito maiores depois de compilados. Como os computadores quânticos atuais ainda possuem baixo tempo de coerência, um circuito maior representa também uma taxa de erros muito maior.

Referências

[1] C. G. Almudever, L. Lao, X. Fu, N. Khammassi, I. Ashraf, D. Iorga, S. Varsamopoulos, C. Eichler, A. Wallraff, L. Geck, A. Kruth, J. Knoch, H. Bluhm, and K. Bertels. The engineering challenges in quantum computing. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 836–845, March 2017.

[2] S. Axler. *Linear Algebra Done Right*. Springer, New York, 1997.

[3] S. Barnett. *Quantum Information*. Oxford University Press, New York, 2009.

[4] G. Benenti, G. Casati, and G. Strini. *Principles of Quantum Computation and Information: Basic Tools and Special Topics*. World Scientific Publishing, River Edge, 2007.

[5] C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev.*, 17(6):525–532, 1973.

[6] J. A. Bergou and M. Hillery. *Introduction to the Theory of Quantum Information Processing*. Springer, 2013.

[7] R. B. Boppana and M. Sipser. Chapter 14 - The complexity of finite functions. In J. V. Leeuwen, editor, *Algorithms and Complexity*, Handbook of Theoretical Computer Science, pages 757 – 804. Elsevier, Amsterdam, 1990.

[8] A. Borodin. On relating time and space to size and depth. *SIAM Journal on Computing*, pages 733–744, 1977.

[9] C. S. Calude and E. Calude. The road to quantum computational supremacy. *ArXiv:1712.01356*, 2019.

[10] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta. Open quantum assembly language. *arXiv e-prints 1707.03429*, pages 1–24, Jul 2017.

[11] M. Davis. *Computability & Unsolvability*. Dover Publications, 1982.

[12] E. Desurvire. *Classical and Quantum Information Theory: An Introduction for the Telecom Scientist*. Cambridge University Press, 2009.

[13] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. Royal Society London Ser. A*, pages 96–117, 1985.

[14] D. E. Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 425(1868):73–90, 1989.

[15] D. Dieks. Communication by EPR devices. *Physics Letters A*, 92(6):271 – 272, 1982.

[16] L. K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79(2):325–328, 1997.

[17] M. Hayashi, S. Ishizaka, A. Kawachi, G. Kimura, and T. Ogawa. *Introduction to Quantum Information Science*. Springer, 2014.

[18] M. Hirvensalo. *Quantum Computing*. Springer, 2010.

[19] P. Kaye, R. Laflamme, and M. Mosca. *An Introduction to Quantum Computing*. Oxford University Press, New York, 2007.

[20] A. Y. Kitaev, A. H. Shen, and M. N. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, Boston, 2002.

[21] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5(3):183–191, July 1961.

[22] S. Lang. *Introduction to Linear Algebra*. Undergraduate Texts in Mathematics. Springer, New York, 1997.

[23] R. J. Lipton and K. W. Regan. *Quantum Algorithms via Linear Algebra: A Primer*. MIT Press, 2014.

[24] A. Mandviwalla, K. Ohshiro, and B. Ji. Implementing Grover's algorithm on the IBM quantum computers. In *2018 IEEE International Conference on Big Data*, pages 2531–2537, 2018.

[25] D. C. Marinescu and G. M. Marinescu. *Approaching Quantum Computing*. Pearson/Prentice Hall, Michigan, 2005.

[26] N. D. Mermin. *Quantum Computer Science: An Introduction*. Cambridge University Press, New York, 2007.

[27] M. A. Nielsen and I. L. Chuang. *Computação Quântica e Informação Quântica*. Editora Bookman, 2005.

[28] J. L. Park. The concept of transition in quantum mechanics. *Foundations of Physics*, 1(1):23–33, Mar 1970.

[29] R. Portugal. *Quantum Walks and Search Algorithms*. Springer, Cham, 2018.

[30] R. Portugal, C. C. Lavor, L. M. Carvalho, and N. Maculan. *Uma Introdução à Computação Quântica*, volume 8 of *Notas em Matemática Aplicada*. SBMAC, São Carlos, 1st edition, 2004.

[31] J. Preskill. Quantum computing in the NISQ era and beyond. *arXiv:1801.00862*, 2018.

[32] E. Rieffel and W. Polak. *Quantum Computing, a Gentle Introduction*. MIT Press, Cambridge, 2011.

[33] C. E. Shannon. A symbolic analysis of relay and switching circuits. *Electrical Engineering*, 57(12):713–723, Dec 1938.

[34] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proc. 35th Annual Symposium onFoundations of Computer Science*, pages 124 –134, 1994.

[35] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2006.

[36] J. Stolze and D. Suter. *Quantum Computing, Revised and Enlarged: A Short Course from Theory to Experiment*. Wiley-VCH, 2008.

[37] G. Strang. *Linear Algebra and Its Applications*. Brooks Cole, 1988.

[38] T. Toffoli. Reversible computing. In J. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming*, pages 632–644. Springer, Berlin, 1980.

[39] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

[40] I. Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons, Inc., New York, NY, USA, 1987.

[41] C. P. Williams. *Explorations in Quantum Computing*. Springer, 2008.

[42] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299:802–803, 1982.

[43] N. S. Yanofsky and M. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, 2008.

Capítulo 2

Como programar aplicações de alto desempenho com produtividade

Álvaro Luiz Fazenda e Denise Stringhini

Abstract

The development of scientific applications that use high performance computing resources historically is considered complex and specialized. At first, the programmer must know the characteristics of the problem in order to perform its computational modeling. Considering the use of parallel hardware, it must still know its architectural characteristics, as well as paradigms, parallel languages and possible frameworks to be used. This chapter presents the basic and intermediate principles of parallel directive-based programming, applicable to problems commonly encountered in scientific applications, such as numerical modeling and computational intelligence, for example. Directive-based programming techniques are presented using OpenMP for multicore computers and OpenACC for machines equipped with GPU-type accelerators.

Resumo

O desenvolvimento de aplicações científicas que se utilizam de recursos de computação de alto desempenho historicamente é considerado complexo e especializado. A princípio, o programador deve conhecer as características do problema de forma a realizar sua modelagem computacional. Considerando o uso de um hardware paralelo, ele ainda deve conhecer suas características arquiteturais, assim como paradigmas, linguagens paralelas e possíveis frameworks a serem empregados. O presente capítulo apresenta os princípios básicos e intermediários da programação paralela baseada em diretivas, aplicáveis a problemas comumente encontrados em aplicações científicas, tais como modelagem numérica e inteligência computacional, por exemplo. Serão apresentadas técnicas de programação por diretivas usando OpenMP para computadores *multicore* e OpenACC para máquinas equipadas com aceleradores do tipo GPU.

2.1. Introdução

Nas últimas décadas tem-se observado uma crescente necessidade no uso plataformas de hardware e software capazes de processar grandes volumes de dados e executar cálculos complexos. O cenário de HPC (*High Performance Computing* ou Processamento de Alto Desempenho - PAD) é composto de uma grande variedade de tipos e de plataformas disponíveis, portanto uma compreensão das características básicas destes sistemas se faz necessária para a tomada de decisão em qualquer tipo de investimento na área.

Sistemas de alto desempenho são capazes de processar grandes volumes de dados e executar cálculos complexos. Por exemplo, pode-se pensar em uma aplicação para previsão de tempo. Este tipo de aplicação, normalmente, divide a atmosfera em uma grande quantidade de volumes tridimensionais e aplica a cada volume uma série de cálculos físicos, baseados em modelagem matemática, utilizando-se de determinados dados de entrada e, repetindo os cálculos de forma iterativa, conseguem determinar o estado de várias propriedades físicas, permitindo assim prever o estado futuro da atmosfera. Quanto menor o volume tridimensional utilizado, mais precisa a previsão tenderá a ser, porém a quantidade de volumes será maior para uma mesma área de interesse e, portanto, mais demanda computacional que implica em mais tempo de processamento, considerando um mesmo recurso computacional. Dessa forma, pode-se imaginar um caso onde se pode obter alta precisão nas informações geradas, porém, finalizada em tempo proibitivo para utilidade prática. Este tipo de aplicação, assim como qualquer outra que necessite de respostas mais rápidas para problemas complexos, exige que sistemas computacionais de alto desempenho sejam empregados.

Sistemas de PAD possuem características bem específicas de hardware e software que os classificam como sistemas que, para um determinado instante no tempo, apresentam desempenho superior ao comparado a computadores usuais, de propósito geral. O objetivo primordial destes tipos de sistema é a obtenção de desempenho, portanto não devem ser confundidos com sistemas puramente distribuídos. Sistemas distribuídos têm características mais abrangentes, não tendo necessariamente um compromisso com a obtenção de desempenho computacional. Possuem um carácter mais funcional voltado ao compartilhamento de recursos.

Para se ter uma ideia do tamanho e capacidade dos maiores sistemas de PAD atualmente, vale a pena consultar a lista Top 500 (TOP500, 2018) que enumera bi-anualmente as 500 máquinas mais rápidas do mundo para uma determinada aplicação padrão. No momento da escrita deste texto (abril de 2019), a máquina mais rápida do mundo é a *Summit*, localizada no *Oak Ridge National Laboratory* nos Estados Unidos. *Summit* é uma máquina fabricada pela IBM que possui 2.397.824 *cores* (núcleos de computação ou processadores) e atingiu 143,5 petaflops ao executar o *benchmark* de referência utilizado pelo Top 500. Além do processador *multicore* da família *Power* da IBM, a Summit é equipada com placas do tipo GPU (*Graphics Processing Unit*) da NVidia, que operam como co-processadores aritméticos permitindo grande aceleração nos cálculos científicos, além de contar com uma rede de interconexão de baixa latência do tipo Infiniband da Mellanox. Embora estas máquinas estejam fora do alcance financeiro de grande parte das empresas e institutos de pesquisa e desenvolvimento,

sobretudo no Brasil, os fabricantes oferecem versões menores com o objetivo de atender a diferentes demandas de mercado.

Considerando-se a complexidade do hardware com seus vários níveis de paralelismo, não é difícil imaginar que a programação destas máquinas exige um bom nível de especialização para que seja possível a obtenção de desempenho. A cada nível de paralelismo (por exemplo: nível de *threads* X nível de processos) diferentes paradigmas de programação paralela podem ser empregados. Mesmo com a utilização de *frameworks* e bibliotecas para desenvolvimento de aplicações paralelas, os softwares de PAD ainda requerem algum nível de especialização e de compreensão do hardware utilizado para que se obtenha um desempenho satisfatório.

O presente capítulo tem o intuito de abordar os paradigmas básicos de programação necessários a utilização de equipamentos mais simples, porém derivados dos atuais supercomputadores. O texto aborda a programação para equipamentos de alto desempenho de uma maneira introdutória, descrevendo paradigmas e técnicas de programação que se aplicam a uma arquitetura paralela de memória compartilhada, bem como a utilização de aceleradores de desempenho do tipo GPU. Neste sentido, são abordados conjuntos de diretivas de programação que podem ser utilizados em conjunto com as linguagens C/C++ para a programação de *multicores* e de GPUs. Para programação de *multicores* é utilizada a programação com diretivas aderentes ao padrão OpenMP (Chapman, 2008), e para programação de equipamentos dotados de GPUs como co-processadores numéricos, será apresentada programação com diretivas do padrão OpenACC (OpenACC, 2019).

Os pré-requisitos esperados para os leitores deste documento é que possuam conhecimentos sobre linguagem de programação C e estruturas de dados. Conhecimentos básicos sobre sistemas operacionais e arquitetura e organização de computadores, bem como arquiteturas do tipo GPU são desejáveis, porém não obrigatórios.

Este documento está dividido da seguinte forma: o segundo subcapítulo aborda a arquitetura de sistemas computacionais de Alto Desempenho de forma a contextualizar os modelos de programação abordados num momento posterior. O terceiro subcapítulo aborda algumas técnicas básicas de projeto e extração de paralelismo considerando o paradigma de memória compartilhada, essencial para que se possa programar através das diretivas paralelas a serem apresentadas posteriormente. O subcapítulo quatro detalha a programação por diretivas OpenMP aplicado a sistemas de memória compartilhada, detalhando um exemplo de aplicação prática. O quinto subcapítulo discorre sobre a programação para aceleradores do tipo GPU, com especial ênfase para a programação com diretivas do padrão OpenACC, mostrando exemplos simples de aplicação, bem como a aceleração do mesmo estudo de caso utilizado no subcapítulo anterior, comparando o desempenho com o programa em OpenMP. O sexto e último subcapítulo apenas ressalta as conclusões.

2.2. Modelo hierárquico de sistemas computacionais de alto desempenho

A arquitetura de sistemas computacionais de alto desempenho compreende diferentes tipos de paralelismo que podem ser vistos a partir de um modelo hierárquico de composição, partindo de múltiplos *cores* até o sistema completo com uma série de *racks*, incluindo componentes adicionais conhecidos atualmente por aceleradores. Este subcapítulo aborda o tema de forma simplificada, enfatizando a arquitetura de um único nó computacional de alto desempenho comum nas atuais arquiteturas de supercomputadores. Vale salientar, no entanto, que o sistema completo de um supercomputador conta com outras possibilidades de componentes não abordados aqui, tais como nós de acesso (*login*) e sistema de arquivos. Um bom exemplo deste tipo de arquitetura hierárquica é a arquitetura Blue Gene/Q da IBM (Gilge, 2014). A seguir, são apresentadas as características básicas de um módulo básico para processador e memória compartilhada.

2.2.1. Módulo básico para processador e memória compartilhada

Em termos de hardware, o módulo básico de um supercomputador basicamente é composto por um ou mais processadores do tipo *multicore* (mais comumente da conhecida família Intel x86) que possui normalmente uma dezena de *cores* (ou núcleos) efetivos e uma memória principal da ordem de centenas de Gbytes. Considerando-se arquiteturas paralelas, este modelo se encaixa entre as chamadas arquiteturas com **memória compartilhada** ou **multiprocessador**.

Nos multiprocessadores, todos os núcleos acessam uma memória compartilhada através de uma rede de interconexão. Nas máquinas mais comuns, a rede de interconexão comumente empregada é o barramento que liga os núcleos à memória. Entretanto, outras interconexões mais sofisticadas, podem ser usadas em máquinas com uma quantidade maior de processadores e memória.

Os diversos níveis da hierarquia de memória (registradores, memórias *cache*, memória principal) afetam diretamente o desempenho das aplicações. Porém, neste texto introdutório as questões relacionadas à hierarquia (Hennessy e Patterson, 2007) são apenas brevemente tratadas. No contexto de arquiteturas paralelas, a memória é comumente abordada a partir de uma classificação básica utilizada para diferenciar as máquinas paralelas: memória compartilhada e memória distribuída (não compartilhada).

A memória compartilhada significa que o espaço de endereçamento é único. Assim, pode-se imaginar que a memória principal, mesmo sendo composta fisicamente por vários bancos, é endereçada pela mesma sequência de endereços. Assim, diferentes processadores ou *cores* podem acessar um mesmo endereço físico de memória, o que permite que compartilhem dados. Na prática, utiliza-se comumente a programação através de *threads* com variáveis compartilhadas.

Antes de continuar, entretanto, vale ressaltar as diferenças básicas entre processos e *threads*. Um processo é uma unidade de programa em execução que inclui a posse de um espaço de memória para guardar a imagem do processo. A imagem do processo inclui o código do programa, os dados globais, os dados locais (pilha), a área

de memória dinâmica (*heap*), além do contador de programa (PC – *program counter*) e demais atributos que definem o processo. Já as *threads* são normalmente unidades de execução dentro do escopo de um processo. Assim, elas compartilham a maior parte do espaço de endereço (memória) de um processo. A sua área própria terá apenas o PC e a pilha (dados locais). As variáveis globais e a área de memória livre são automaticamente compartilhadas entre as *threads* de um mesmo processo, o que possibilita o compartilhamento de variáveis, muito útil na programação paralela deste tipo de arquitetura. Além disso, o fato de possuírem uma quantidade menor de recursos a serem gerenciados, torna as *threads* unidades de processamento concorrente de menor custo de criação e encerramento comparativamente aos processos. Por esta razão, são empregadas no paralelismo de aplicações em arquiteturas do tipo *multicore*.

Este texto trata de modelos de de programação com memória compartilhada, mas vale a pena diferenciá-lo do modelo de memória distribuída. A definição clássica para arquitetura de memória distribuída envolve os itens básicos que todo computador deve possuir: processador e memória. Cada processador possui sua própria memória local, com endereçamento de memória particular e a comunicação acontece por algum meio físico de comunicação (rede de interconexão). Essa representação também é chamada de arquitetura de **multicomputadores**, normalmente conhecidos como *clusters* ou aglomerados de computadores. Ao contrário das arquiteturas com memória compartilhada, as de memória distribuída são altamente escaláveis, permitindo centenas de milhares de *cores* devido principalmente ao desacoplamento de memória. É comum, encontrar-se máquinas híbridas, contendo ambos os paradigmas de memória.

2.2.1.1. Arquitetura *multicore*

Um *multicore* combina dois ou mais núcleos (*cores)*, em uma única peça de silício (ou pastilha – *die*). Normalmente um núcleo consiste de todos os componentes de um processador independente, tais como registradores, ALU (*Arithmetic and Logic Unit* - unidade aritmética e lógica), *pipelines*, unidades de controle, *caches* de dados (de L1 até L3, em alguns casos) e de instruções.

Num nível mais alto de descrição, as principais variáveis em uma organização *multicore* são as seguintes (Stallings, 2009):

- Número de núcleos de processamento no chip.
- Número de níveis de memória *cache*.
- Quantidade de memória *cache* que é compartilhada ou dedicada em cada núcleo.

Para as *caches* não compartilhadas, o hardware inclui a implementação de um protocolo que garante a coerência das *caches* em caso de alteração de seu conteúdo pelo respectivo núcleo. Maiores detalhes sobre protocolos de coerência de *cache* podem ser encontrados em Stallings (2009) e Hennessy e Patterson (2007).

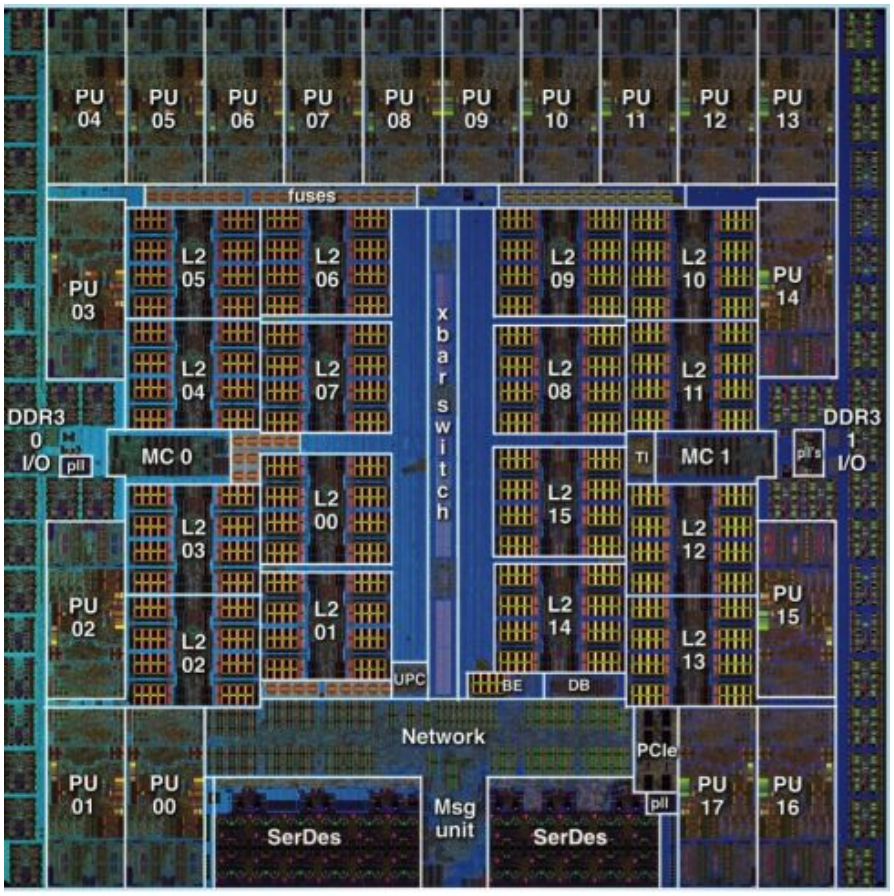


Figura 2.1 - Arquitetura multicore de uma CPU do sistema IBM Blue Gene/Q
Fonte: Stephan, 2015 (*licença CC 4.0*)

A título de exemplo, na figura 2.1 é possível visualizar a arquitetura interna de uma CPU utilizada no sistema IBM Blue Gene/Q. Nesta imagem pode-se identificar os 16 núcleos paralelos por *PUs* (*Processing Units*) e vários bancos de memória *cache* L2, que são compartilhados por todos os núcleos presentes. Além disso é possível verificar a existência de dois módulos de controle de memória (*Memory Controllers - MC 0* e *MC 1*), que permitem a movimentação de dados entre a memória principal externa, a memória *cache* interna e os registradores.

2.2.1.2. Computação heterogênea com GPU como co-processadores

O termo **computação heterogênea** tem sido empregado para designar o uso de diferentes arquiteturas de processamento em um mesmo sistema computacional, utilizados conjuntamente, a fim de se obter melhor desempenho das aplicações. Neste tipo de sistema é comum encontrar-se dispositivos conhecidos por **aceleradores**, os quais, na maioria das vezes, atuam como um co-processadores, ou seja, unidades extra de processamento de código dependente de um processador tradicional. Entre as alternativas existentes destacam-se arquiteturas como as GPUs (*Graphics Processing*

Units). As unidades aceleradoras podem estar fisicamente conectadas a um nó de processamento que conta com um processador tradicional, ou representarem um nó computacional por completo.

As GPUs são compostas de centenas ou até milhares de núcleos (*cores*) simples que executam o mesmo código através de centenas a milhares de *threads* concorrentes. Esta abordagem se opõe ao modelo tradicional de processadores *multicore*, onde algumas unidades de núcleos completos e independentes são capazes de executar *threads* ou processos independentes. As GPUs contam com unidades de controle e de execução mais simples, onde uma unidade de despacho envia apenas uma instrução para um conjunto de núcleos que a executarão em ordem. Este modelo de execução onde várias *threads* são executadas simultaneamente em vários *cores* das GPUs é conhecido como SIMT (*Single Instruction Multiple Threads*), derivado do clássico termo SIMD (*Single Instruction Multiple Data*) (Flynn, 1972). O modelo SIMT impõe um sincronismo na execução das *threads* que executam em grupo nas unidades de multiprocessadas das GPUs. O modelo de *threads* das GPUs será abordado na seção que trata deste modelo de programação. No entanto, a arquitetura das GPUs não será detalhada neste texto por ser altamente especializada, maiores detalhes podem ser encontrados em Kirk e Hwu (2010).

Entre os demais modelos de aceleradores existentes cita-se o uso de FPGAs (Escobar et al. 2016), recentemente impulsionado pela compra da Altera (fabricante de FPGAs) pela Intel. Entretanto, o uso de FPGAs como aceleradores para alto desempenho ainda necessita de modelos de programação com um nível de abstração aceitável pelos desenvolvedores. Neste contexto, a biblioteca OpenCL tem sido utilizada com sucesso para o desenvolvimento de aplicações em FPGA, como por exemplo em Hassan et al. 2018. Este tipo de acelerador, no entanto, está fora do escopo deste texto.

2.3. Paralelização de aplicações

Se a aplicação já existe (código legado), ela poderá ser paralelizada, o que muitas vezes exige uma refatoração para que usufrua do processamento *multicore* ou GPU. Breshears, 2009 sugere quatro passos na paralelização de aplicações: análise, projeto e implementação, testes de correção e análise de desempenho. Os tópicos a seguir abordam estes quatro passos.

2.3.1 Análise

Neste primeiro passo realiza-se a identificação dos possíveis pontos onde se possa implementar concorrência. Esta só poderá ser implementada em pontos de código onde não haja dependência de dados. Além disso, sugere-se escolher os trechos de paralelização a partir da medição de tempo em trechos do código onde haja muito processamento (*hot spots*). Normalmente estes trechos podem ser identificados por laços.

2.3.1.1 Análise de dependências

A implementação de paralelismo implica na execução concorrente (simultânea) de trechos de código. Considerando-se arquiteturas com memória compartilhada é necessário evitar o acesso concorrente a posições de memória (variáveis) que estão sofrendo operações de leitura e escrita de forma intercalada. Este tipo de atualização intercalada de variáveis compartilhadas pode gerar inconsistências na execução do código e são conhecidas como condições de corrida (*race conditions*).

A análise de dependências é o mecanismo formal que permite identificar limitantes à aplicação de concorrência para paralelização em um dado algoritmo originalmente sequencial. A técnica é especialmente útil quando aplicada em um programa que sofrerá uma decomposição de domínio, ou seja, concorrência através da divisão de dados a serem processados por diferentes processadores, como acontece frequentemente em laços que computam sobre *arrays*. A análise de dependências é também muito utilizada por compiladores para fazer paralelização ou vetorização automáticas, além de permitir execução especulativa e fora de ordem de instruções. Entretanto, a solução de um possível problema de dependência pode indicar transformações no código (refatoração) para permitir concorrência.

No algoritmo 2.1, por exemplo, pode-se ver dois laços implementados em linguagem C/C++ onde considera-se as variáveis internas passíveis de compartilhamento. No primeiro (à esquerda), é possível observar que cada iteração pode ser independente uma da outra, haja vista que não há acesso de leitura e escrita intercalada entre as variáveis internas. Deveria-se apenas cuidar para que o índice do laço não fosse compartilhado caso os trechos sejam concorrentes.

No segundo trecho (à direita), existe uma variável interna que pode sofrer operações de leitura e escrita de forma intercalada, pois trata-se de um acumulador (variável `SUM`). Em caso de implementar-se concorrência no código da direita, haveria a necessidade de refatoração para permitir o acesso concorrente ao acumulador sem o perigo de atualizações incorretas. Além disso, a posição dada pelo índice "*i*" do vetor `X` está também sofrendo operações de leitura e escrita durante as iterações do laço, ocasionando dependência de dados entre as duas primeiras linhas. Note-se que a primeira deve ser avaliada antes da segunda para que a posição corrente do *array* X esteja atualizada antes de ser utilizada na linha seguinte. Caso seja os comandos sejam executados de forma concorrente, não será possível garantir esta ordem.

```
for(i=0; i<N; i++) {
    X[i] = Y[i] + Z[i];
    A[i] = Y[i] + delta;

}
```

```
SUM = 0;
for(i=0; i<N; i++) {
    X[i] = Y[i] + Z[i];
    A[i] = X[i] + delta;
    SUM += A[i];
}
```

Algoritmo 2.1 - Exemplo de laços sem e com dependência de dados

Existem vários tipos diferentes de dependências entre instruções de um algoritmo. O fato de haver dependência não significa necessariamente que o fragmento do código não pode ser transformado em um código com concorrência, para tanto deve-se verificar que tipo de dependência se trata. Algumas formas permitem a técnica de vetorização (não abordada neste texto) mas não a paralelização do código, outras o oposto.

De acordo com Dowd e Severance (1998), as dependências podem ser classificadas em três tipos:

a) Dependência de Fluxo - **RAW** (*Read After Write*): a leitura depois da escrita pode ocorrer quando uma variável sofre operação de escrita em uma primeira instrução (atribuição) e subsequentemente é utilizada na próxima instrução (operação de leitura). O problema ocorre caso o inverso aconteça por conta da concorrência.
b) Anti-dependência - **WAR** (*Write After Read*): a escrita depois da leitura pode ocorrer quando uma variável sofre operação de leitura em uma primeira instrução e subsequentemente é atribuída na próxima instrução, o que significa sofrer operação de escrita. Novamente, o problema ocorre caso o inverso aconteça por conta da concorrência.
c) Dependência de saída - **WAW** (*Write After Write*): a escrita depois da escrita pode ocorrer quando uma variável sofre operação de escrita em uma primeira instrução e imediatamente é atribuída, sofrendo nova operação de escrita. Neste caso, o problema ocorre em atualizações fora da ordem sequencial imposta pela sequência de execução do laço.

Ao realizar a análise de dependência, portanto, é preciso observar-se com atenção variáveis compartilhadas que sofrem operações de leitura e escrita dentro dos tracho a serem paralelizados. Busca-se estas variáveis no código e tenta-se realizar a refatoração a fim de tentar eliminá-las ou protegê-las do acesso concorrente. Ao analisar-se novamente os laços apresentados no algoritmo 2.1 verifica-se que o primeiro (à esquerda) não possui nenhuma dependência, pois as duas posições dos vetores `X` e `A` que estão sendo escritas não aparecem em nenhuma operação de leitura (lado direito das atribuições).

No segundo laço, entretanto, posições acessadas do array `X`, assim como a variável `SUM`, aparecem ao lado esquerdo e direito de atribuições, o que significa que estão sofrendo operações de leitura e escrita no trecho de código que se deseja paralelizar. As operações no vetor `X` configuram **RAW**, pois a escrita deve ser executada antes da leitura. Neste caso, o problema na execução poderá ocorrer se uma *thread* fizer a leitura antes de a referida posição do vetor sofrer a atualização necessária (escrita) na linha anterior. Entretanto, este tipo de dependência é facilmente solucionável se cada *thread* concorrente executar uma iteração completa do laço, evitando a concorrência dentro de cada iteração, criando apenas a concorrência entre as diversas iterações do mesmo laço.

Já as operações na variável `SUM` são mais delicadas, pois tem-se operações de leitura e escrita em uma mesma variável compartilhada. Considerando que as *threads* concorrentes executam iterações completas do laço, conforme citado no parágrafo

acima, as operações concorrentes sobre o acumulador podem gerar situações de **WAW** e **WAR** entre *threads* concorrentes. No caso de WAW, tem-se a ocorrência de *threads* concorrentes sobrescrevendo o valor de `SUM` de forma sucessiva e possivelmente fora de ordem. Para o caso de WAR, pode ocorrer o uso desatualizado da variável `SUM` em uma dada iteração *"i"*, caso o valor da mesma variável na iteração *"i-1"*, em execução por outra *thread*, não tenha sido salvo na posição de memória correspondente. Este caso é bem comum em variáveis como acumuladores, visto que várias *threads* podem estar atualizando estas variáveis de forma intercalada, conforme já mencionado. O tratamento para este tipo de operação será visto em várias ocasiões ao longo deste texto.

A utilização de mecanismos de exclusão mútua serve para evitar os problema acima mencionados, conhecidos também como condição de corrida (*race condition*), onde a leitura e escrita em variáveis compartilhadas podem gerar resultados inconsistentes com o esperado. Mecanismos de exclusão mútua (Ben-Ari, 2005) normalmente fazem parte de disciplinas de Sistemas Operacionais e/ou Sistemas Concorrentes e servem para controlar o acesso a posições de memória compartilhada, as chamadas regiões ou seções críticas, tal que apenas um trecho de código concorrente realize o acesso de cada vez. Embora este tipo de mecanismo traga benefícios em termos da correção na execução do código, eles podem limitar o paralelismo, portanto devem ser evitados sempre que possível. Exemplos de tais mecanismos são os *locks*, semáforos e monitores. As linguagens e bibliotecas de programação paralela costumam oferecer opções de controle de acesso a regiões críticas de código, alguns deles serão vistos ao longo do texto.

2.3.1.2 Detecção de *hot spots*

Inicia-se o trabalho de paralelização do código a partir dos trechos mais demorados, os quais representam no jargão popular o "gargalo" no tempo de processamento, também chamados de *hot spots*. Usualmente os laços são bons candidatos, visto que executam tarefas repetitivas sobre um conjunto de dados homogêneos. Os laços são candidatos naturais, portanto, ao tipo de paralelismo conhecido como paralelismo de dados. Neste tipo de paralelismo é possível recorrer à decomposição de domínio, onde o domínio, representado por uma estrutura de dados homogênea, é dividido entre unidades de processamento concorrentes (*threads* ou processos) que executam o mesmo processamento de forma simultânea sobre dados diferentes. Este tipo de paralelismo se opõe ao paralelismo de tarefas, ou decomposição funcional, onde cada unidade de processamento concorrente executa uma tarefa diferente para a resolução de um problema. Este texto tem o foco na decomposição de domínio.

Assim, de uma maneira geral, escolhe-se os laços candidatos e efetua-se a medição de tempo com o objetivo de detectar aqueles que consomem maior tempo de processamento. Usualmente esta medição é realizada capturando-se o valor do relógio antes e depois do trecho a ser medido e efetuando-se a diferença entre as duas medidas. O algoritmo 2.2 mostra um exemplo de código que pode ser utilizado para esta tarefa, onde se introduziu uma função que permite medir um instante de tempo em intervalos distintos (função `gettimeofday`), e depois calcular a diferença entre esses intervalos.

O processo de introduzir medidas relacionadas a desempenho em código-fonte é conhecido por instrumentalização.

```
#include <stdio.h>
#include <sys/time.h>
#define N 1000000

int main(void) {
   int k;
   double p = 1, x;
   struct timeval inicio, final;
   long long tmili;

   gettimeofday(&inicio, NULL);
   x = 1.0 + 1.0/N;
   for(k=0; k<N; k++)
      p = p*x;
   gettimeofday(&final, NULL);

   tmili = (int) (1000*(final.tv_sec - inicio.tv_sec) +
           (final.tv_usec - inicio.tv_usec) / 1000);

   printf("PI aproximado: %g\n", p);
   printf("tempo decorrido: %lld ms\n",tmili);
   return 0; }
```

Algoritmo 2.2 - Exemplo de código-fonte com instrumentação para medida de tempo de execução de um trecho específico.

A detecção de *hot spots* juntamente com análise de dependência servem para identificar a viabilidade e os possíveis benefícios na paralelização de determinados trechos. A paralelização de código deve ser efetuada em casos críticos onde o tempo de processamento é proibitivo e onde o esforço de paralelização seja recompensado através de ganhos de desempenho (*speedup*). Além disso, a análise de dependência deve indicar possíveis empecilhos que possam tornar a paralelização mais difícil, exigindo uma refatoração mais delicada. Assim, a fase de análise deve trazer indícios para a avaliação de custo e benefício da paralelização de um código.

2.3.2 Projeto e implementação

Após a fase de análise, tem-se a fase de desenvolvimento, onde o projeto e a implementação devem ser executados. A fase de análise indica o tipo de decomposição de domínio a ser empregada, assim como quais são os pontos do código onde se poderá ter maior ganho. No projeto, deve-se levar em conta também qual o hardware disponível e consequentemente qual ou quais os paradigmas de programação a serem empregados.

Num primeiro momento, é comum considerar-se o paradigma de memória compartilhada visto que as arquiteturas mais comuns e presentes na primeira escala de paralelismo são as de *multicore*. Seguindo o mesmo raciocínio, pensa-se em seguida nos aceleradores, tais como as placas gráficas, hoje em dia presentes na maioria das

máquinas utilizadas para fins comuns. Embora as arquiteturas sejam diferentes, em ambos os casos pode-se contar com memória compartilhada e paralelismo entre *threads* - a principal diferença e termos de programação é a granularidade do processamento, como será visto mais adiante. Um terceiro passo seria pensar na escalabilidade, ou seja, na execução do código em centenas ou até milhares de *cores*. Neste ponto, entraria o paradigma de memória distribuída e a execução em *clusters*. O presente texto preocupa-se com a utilização de paralelismo em máquinas mais comumente disponíveis aos desenvolvedores, portanto apenas o paradigma de memória compartilhada é abordado.

2.3.2.1 Padrões de programação paralela

Padrões de código são soluções genéricas e reutilizáveis para problemas comuns em determinados contextos. O projeto de código paralelo pode usar algum padrão de código conhecido para este fim, como os apresentados em Mattson *et al* (2004) e McCool *et al* (2012). Particularmente este último apresenta alguns padrões de forma gráfica que podem ser úteis para que se visualize algumas das construções mais comuns em algoritmos paralelos. Foram selecionados quatro deles (figura 2.2) que serão úteis no decorrer deste texto: *fork-join, map, stencil, reduction*.

O padrão *fork-join* (figura 2.2 (a)) é dividido nestas duas operações que podem ser aninhadas. A operação *fork* (representada pelos círculos escuros na figura) permite que o fluxo de controle seja dividido em múltiplos fluxos de execução paralelos (quadrados). Na operação *join* (círculos mais claros) estes fluxos se reunirão novamente no final de suas execuções, quando apenas um deles continuará.

O padrão *map* (figura 2.2 (b)) replica uma mesma operação sobre um conjunto de elementos indexados. O conjunto de índices pode ser abstrato ou estar diretamente relacionado aos elementos de uma coleção. Este padrão se aplica à paralelização de laços, nos casos onde se pode aplicar uma função independente a todo o conjunto de elementos.

O padrão *stencil* (figura 2.2 (c)) é uma generalização do padrão *map*, onde a função é aplicada sobre um conjunto de vizinhos. Os vizinhos são definidos a partir de um conjunto *offset* relativo a cada ponto do conjunto. Todos podem ser calculados em paralelo, porém alguns cuidados de implementação devem ser tomados para evitar o não-determinismo (assim como no padrão *map*). O não-determinismo pode acontecer porque os elementos são lidos e escritos simultaneamente de forma indiscriminada ocasionando diferença de resultados entre as execuções. Alguns pontos podem estar sendo calculados a partir de dados atualizados, enquanto outros podem estar sendo calculados a partir de dados antigos. A cada execução esta situação pode se alterar e gerar resultados diferentes tanto relativamente à execução sequencial quanto entre as execuções paralelas. Entre as técnicas que podem ser utilizadas para manter o determinismo entre as execuções paralelas está o uso de duas matrizes - uma para leitura e outra para a escrita - que se alternam a cada iteração. Esta técnica será vista num dos exemplos mais adiante neste capítulo.

O padrão *reduction* (figura 2.2 (d)) combina todos os elementos de uma coleção em um único elemento a partir de uma função combinadora associativa. Dada a associatividade da operação, muitas ordens de avaliação podem ser implementadas. Uma operação muito comum em aplicações numéricas é realizar um somatório ou encontrar o máximo de um conjunto de elementos.

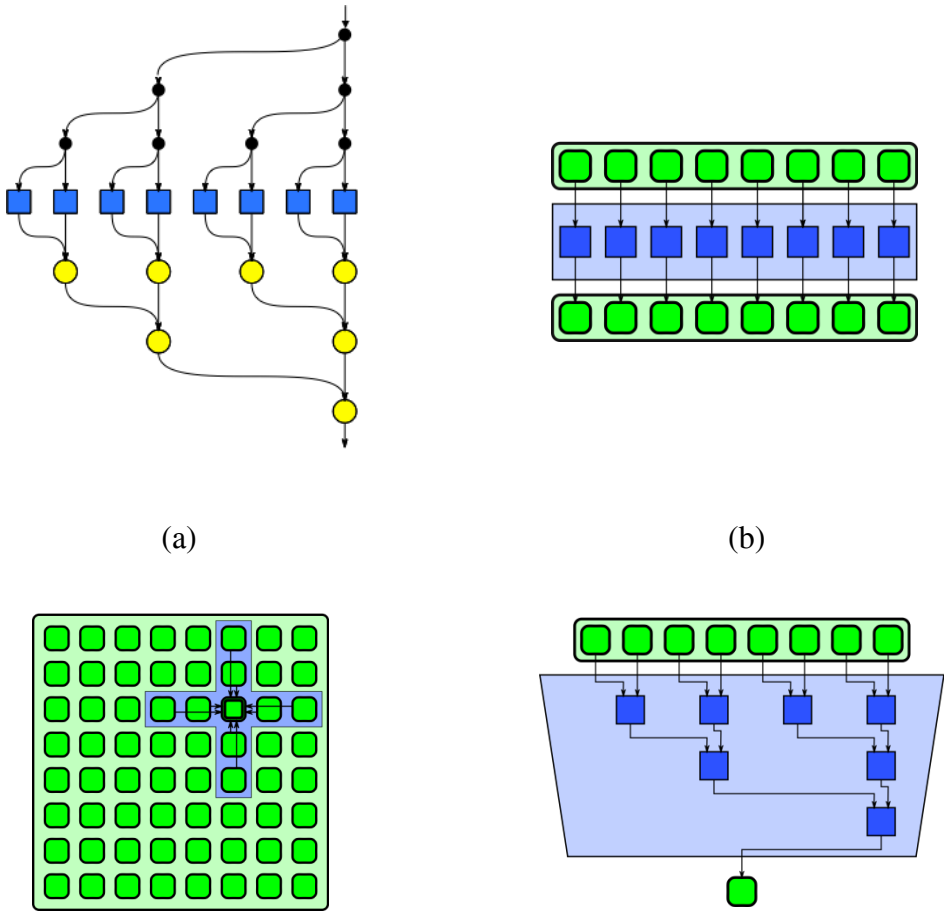


Figura 2.2: Alguns padrões de programação (Fonte: McCool et al, 2012)

Foram descritos alguns padrões básicos de programação paralela que podem ser selecionados na fase de análise. O desenvolvimento do algoritmo paralelo deverá utilizar alguma biblioteca de programação específica e a maioria das bibliotecas possui recursos para a implementação destes e de outros padrões de programação paralela.

Existem algumas opções de programação com *threads* em várias linguagens de programação, como Python, Java e C#. Entretanto, linguagens compiladas como C/C++ ainda apresentam melhor desempenho e são usadas em aplicações onde isto é fundamental. Neste caso, bibliotecas de programação como *Pthreads* (Butenhof, 2006)

e *OpenMP* (Chapman, 2008) são bastante utilizadas. A larga existência de sistemas legados, envolvendo principalmente cálculos numéricos complexos, implica no ainda comum uso da linguagem Fortran em aplicações de alto desempenho.

Este texto aborda duas das bibliotecas de programação mais utilizadas para PAD: OpenMP e OpenACC. A primeira será utilizada para abordar a programação para sistemas paralelos de memória compartilhada (incluindo os *multicores)*, embora algumas funcionalidades voltadas a aceleradores já estejam sendo incorporadas no momento da escrita deste capítulo. Já a segunda possui compilador mais estável para a programação de aceleradores, por isso será utilizada para tal. No caso de se desejar partir para uma implementação voltada à escalabilidade no nível de processos em arquiteturas do tipo *cluster*, a biblioteca de passagem de mensagens MPI - *Message Passing Interface* (Pacheco, 2011) é a mais indicada até o momento, porém foge do escopo deste documento.

OpenMP constitui-se de um padrão que define um conjunto de diretivas de programação juntamente com algumas rotinas de biblioteca (*API - Application Programming Interface* - Interface de Programação de Aplicações) para programação com *threads*. A programação com OpenMP é um dos principais objetivos do curso e será descrita mais adiante, juntamente com a biblioteca OpenACC.

2.3.3 Testes de correção

Este subcapítulo não pretende abordar técnicas e formalismos da área de teste de software, pois fugiria do escopo do texto. A ideia aqui é chamar a atenção para alguns aspectos da execução de programas paralelos que se diferenciam dos sequenciais e que podem ocasionar erros ou inconsistências de execução.

O código *multithreading* é altamente sujeito a erros como condições de corrida (*race conditions*) já abordadas na fase de análise de dependências. Basicamente, estes erros são causados pela alteração concorrente de dados compartilhados, assim como pelo mau-uso dos mecanismos de controle de acesso (por exemplo, *mutex*) e sincronização entre *threads*. Assim, um bom conjunto de testes é necessário, visto que programas concorrentes têm uma tendência a serem não-deterministas, ou seja, uma execução pode ter um resultado diferente de outra ainda que tenham os mesmos dados de entrada.

O método mais comum de teste de correção, embora aparentemente primitivo, ainda é observar a saída dos programas, o que pode ser feito diretamente no console ou num arquivo de saída. Em linguagem C os desenvolvedores de programas paralelos ainda dependem de comandos do tipo *printf* ou *assert* para realizar testes. Um dos principais motivos é que existem poucas ferramentas que permitem algum grau de depuração e a maioria das que existem não são gratuitas.

Assim, caso não haja ferramenta disponível, sugere-se algum teste básico para verificar a coerência da saída da versão sequencial com a da versão paralela. Num primeiro momento, portanto, é recomendável que se tenha uma versão sequencial devidamente testada e com um conjunto ou mais de dados de entrada e saída

conhecidos. Após o processo de paralelização, utiliza-se o mesmo conjunto de entrada para a execução paralela e observa-se se a saída é coerente. Um cuidado extra, porém, deve ser tomado com programas paralelos e sua natureza não-determinística: caso as devidas precauções não sejam tomadas, o resultado pode variar. Assim, existe a possibilidade, por exemplo, de que um primeiro teste seja compatível com o resultado da execução sequencial e os demais não. Assim, é necessário um conjunto de testes bem abrangente para garantir uma coerência constante entre os resultados das execuções, caso contrário dificilmente poderá obter-se certeza da correção da solução.

2.3.4 Análise de desempenho

É absolutamente necessário que o desempenho de uma aplicação paralelizada seja melhor que o de sua versão sequencial, visto que o custo e a complexidade deste processo devem valer à pena. Medidas de tempo e comparações com a versão sequencial do programa são usadas para verificar se houve ganho de desempenho. Caso contrário, o programador deverá verificar pontos de gargalo que estejam atrapalhando o desempenho. Tipicamente, estes pontos são situações onde haja contenção na sincronização de recursos compartilhados (por exemplo, uso sincronizado de variáveis compartilhadas), desbalanceamento de carga de trabalho entre as *threads* e quantidade excessiva de chamadas à biblioteca de *threads*, o que pode causar um *overhead* inexistente na versão sequencial.

Assim, dois dos principais objetivos do projeto de aplicações paralelas consistem em obter-se:

- **Desempenho**: é a capacidade de reduzir o tempo de resolução do problema à medida que os recursos computacionais aumentam;
- **Escalabilidade**: é a capacidade de aumentar ou manter o desempenho à medida que os recursos computacionais aumentam.

A vantagem da escalabilidade está relacionada à possibilidade de se aumentar o *tamanho do problema* e o consequente uso de recursos paralelos para resolvê-lo. O tamanho do problema normalmente corresponde ao tamanho do domínio, ou seja, o volume de dados e de sua estrutura de armazenamento.

Os fatores que limitam o desempenho e a escalabilidade de uma aplicação estão ligados a limites arquiteturais e limites algorítmicos. Entre os limites arquiteturais temos a latência e a largura de banda da camada de interconexão e capacidade de memória da máquina utilizada. Já os limites algorítmicos incluem a própria falta de paralelismo inerente ao algoritmo, usualmente capturado através da análise de dependência de dados. Além disso, tem-se a frequência de comunicação, representada pelo acesso às variáveis compartilhadas ou passagem de mensagens; a frequência de sincronização, normalmente imposta pelo algoritmo, e o escalonamento deficiente, que depende da granularidade das tarefas e do consequente balanceamento de carga a ser efetuado pelo sistema de execução.

A medida básica para a comparação e análise entre versões é o **tempo de execução**. Na fase de análise já utilizou-se medida de tempo semelhante para detectação

dos *hot spots* do programa. Agora, deseja-se saber o quanto o sistema A é mais rápido que o sistema B. Esta medida é dada por: $n = Texec(A) / Texec(B)$.

Baseando-se neste cálculo é possível obter-se o ganho de tempo ou ***speedup*** de um programa paralelo, uma das medidas mais populares para avaliar-se o desempenho de tais aplicações:

Speedup(P) = Texec(1 proc) / Texec(P procs)

- Onde P = número de processadores
- 1 ≤ Speedup ≤ P

Apesar do *Speedup* apresentar os limites teóricos de 1 e P, eventualmente podem-se encontrar valores fora desta faixa, como, por exemplo, no caso de uma otimização mal sucedida, onde a nova versão do código apresenta desempenho inferior a versão serial, obtendo assim um *speedup* menor do que 1. Caso a nova versão favoreça o acesso dos dados em memória *cache* (o que é sempre desejável), devido a um melhor dimensionamento do domínio do problema na versão paralela, o *Speedup* pode ser super-linear, e assim atingir valores superiores à P.

Outra medida importante é a eficiência, que indica o uso dos processadores. A eficiência pode ser calculada da seguinte forma:

Eficiência(P) = Speedup(P) / P

- Onde 0 < Eficiência ≤ 1

Existe uma limitação clássica para o *speedup* conhecida como Lei de Amdhal (Amdhal, 1967). Basicamente, o conhecido autor dividiu o problema em duas categorias de trechos de código: aqueles que podem ser paralelizados (fração paralela) e aqueles que não podem (fração serial). A fração serial é principalmente limitada em função da análise de dependência de dados. Assim, a fração serial sempre será o limite inferior do tempo de execução, não importando o aumento dos recursos computacionais utilizados pela parte paralela.

Vale salientar que a Lei de Amdhal considera apenas a escalabilidade dos recursos computacionais, sendo o tamanho do problema fixo. Neste caso, efetivamente temos limitação do *speedup* que uma aplicação paralela poderá obter. Em contra partida, temos a Lei de Gustafson-Barsis (Gustafson, 1988) que parte da premissa que toda aplicação tem uma fração inerentemente serial e que para se obter escalabilidade é necessário aumentar o tamanho da aplicação ou do domínio a medida que se aumentam o número de recursos computacionais.

Assim, pode-se definir dois tipos de escalabilidade:

- **Escalabilidade forte**: mantém-se o tamanho do problema e escala-se o número de processadores; é a capacidade de executar aplicações *n* vezes mais rápidas, onde *n* é a quantidade de processadores utilizados (*speedup*).
- **Escalabilidade fraca**: escala-se o tamanho do problema juntamente com o número de processadores; é a capacidade de aumentar a carga de trabalho e a quantidade de processadores por um fator de *n* e manter o tempo de computação.

As próximas subseções apresentarão o paralelismo através de primitivas, além de estudos de caso onde estes conceitos teóricos abordados serão aplicados.

2.4. Programação OpenMP

OpenMP constitui-se de um padrão, composto por um conjunto de diretivas de programação, acrescido de um pequeno conjunto de funções de biblioteca e variáveis de ambiente, que usam como base as linguagens C/C++ e Fortran. Por se tratar de um padrão, várias implementações estão disponíveis. É comum que compiladores já conhecidos, como o popular *gcc (GNU Compiler Collection,* versão *open-source* de compilador C/C++*)*, possuam opções de compilação para OpenMP.

As diretivas em C/C++ para OpenMP estão contidas em diretivas do tipo **`#pragma`**, as quais permitem definir instruções que não existem previamente na linguagem C padrão, mais a palavra chave **`omp`** e o nome da diretiva. O conjunto de diretivas e sua cláusulas é relativamente grande, mas com um pequeno conjunto delas já é possível gerar código paralelo.

A principal diretiva do OpenMP é a **`parallel`** que automaticamente cria as *threads* que devem executar concorrentemente um bloco de instruções que a segue. Vale lembrar que um bloco de instruções com mais de uma linha em linguagem C é definido pelos símbolos `{` e `}` (abre e fecha chaves). Assim, para criar um bloco paralelo basta usar a seguinte combinação de palavras-chave: **`#pragma omp parallel.`** As diretivas podem ser complementadas por cláusulas que têm a função de especificar o modo de execução das diretivas, entre outros atributos.

Um ponto importante da programação paralela em OpenMP é a definição da quantidade de *threads* que será criada, também conhecida como "time de *threads*". A figura 2.3 mostra o modelo de execução do OpenMP, onde times de *threads* podem ser criados em qualquer ponto de um código em linguagem C. No contexto de Sistemas Operacionais, este modelo de execução é conhecido como *fork-join* (a operação *fork* cria processos enquanto a operação *join* realiza a sincronização ao final da execução concorrente). O modelo *fork-join* também foi apresentado entre os padrões de programação apresentados no subcapítulo correspondente.

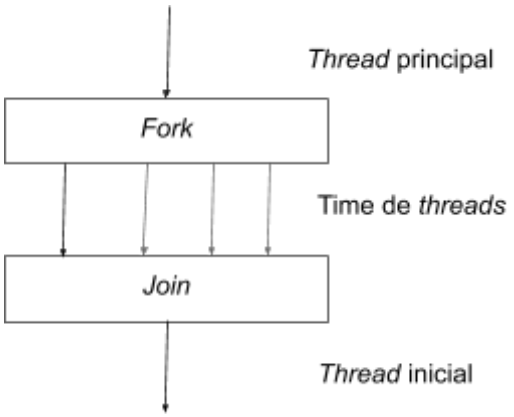


Figura 2.3: Modelo de programação *fork-join* suportado pelo OpenMP

Quanto à quantidade de *threads* criadas em um time existem diferentes métodos para sua definição. Caso não seja especificada de forma explícita no código, o sistema utiliza uma variável de ambiente (que pode ser alterada através de comandos do sistema operacional) chamada **`OMP_NUM_THREADS`**. Outra maneira de alterar a quantidade de *threads* é através de uma função da biblioteca do OpenMP chamada **`omp_set_num_threads()`**. Finalmente, a terceira alternativa é o uso da cláusula **`num_threads`**, juntamente com a diretiva **`parallel`**.

O OpenMP apresenta uma série de diretivas relacionadas ao compartilhamento de trabalho entre as *threads*. A principal delas é a diretiva **`for`**, que automaticamente divide entre as *threads* a execução do laço que a segue. Considerando que normalmente os *hot spots* de programas a serem paralelizados se encontram em laços, a diretiva `for` constitui-se na maneira mais acessível de se obter um programa paralelo com ganhos de desempenho. Esta diretiva, assim como suas principais cláusulas, serão abordadas com mais detalhe no estudo de caso a ser apresentado a seguir.

O padrão OpenMP é amplamente utilizado para paralelização de tarefas visando a obtenção de desempenho em máquinas com memória compartilhada. Também foi o responsável pela popularização da programação paralela por diretivas, a qual tem sido empregada também na programação de aceleradores, através de novas versões do padrão OpenMP e do padrão OpenACC, abordado mais adiante. O estudo de caso a seguir demonstra como aplicar a metodologia de paralelização vista até agora, assim como empregar as principais diretivas do OpenMP para obter-se uma versão paralela e com melhor desempenho da aplicação escolhida.

2.4.1. Estudo de caso em OpenMP: Jogo da Vida

Para exemplificar, vamos tomar um estudo de caso para servir de exemplo de paralelização: O Jogo da Vida (do inglês: *Game of Life* - GOL, Adamatzky 2010), criado por volta de 1960 pelo matemático e professor de Princeton John Conway. O jogo se passa em tabuleiro, similar ao encontrado em tabuleiros de xadrez ou damas, no qual formas geométricas bidimensionais, formada por um conjunto de pontos dispostos no tabuleiro (ou na grade) que replicam-se autonomamente e modificam sua forma a cada evolução (ou iteração do jogo), de acordo com um conjunto de regras pré-definido. O jogo da vida é objeto de estudos para Físicos, Biólogos, Economistas, Matemáticos, Filósofos, entre outros, pois permite observar como padrões complexos podem emergir a partir de implementações de regras muito simples, tal como ocorre em um autômato celular.

As regras de evolução dos elementos que compõem o jogo são muito simples, dado um tabuleiro bidimensional finito ou de bordas infinitas, células desta grade podem ter apenas dois estados, vivo ou morto, sendo que:

a. Cada célula viva com menos de dois vizinhos morre de solidão;
b. Cada célula viva com quatro ou mais vizinhos morre por superpopulação.

c. Cada célula viva com dois ou três vizinhos deve permanecer viva para a próxima geração;
d. Cada célula morta com exatamente 3 vizinhos deve se tornar viva.

A figura 2.4 ilustra um exemplo de configuração para o tabuleiro do jogo, em um determinado instante ou geração.

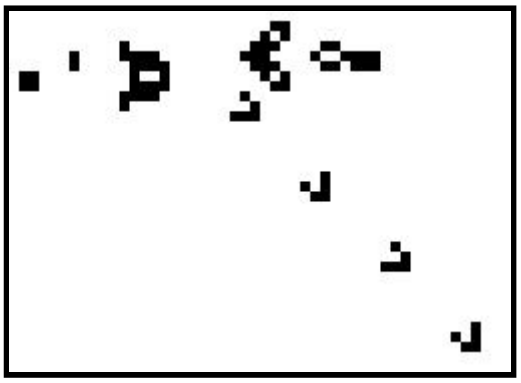


Figura 2.4 - Exemplo de tabuleiro de GOL

O trecho de código-fonte serial que descreve a possível mudança de estado de uma determinada célula no tabuleiro para uma próxima geração, considerando um tabuleiro de bordas periódicas, foi baseado no código-fonte disponível no repositório GitHub sob o endereço: https://github.com/olcf/game_of_life_tutorials, o qual foi desenvolvido pela divisão OLCF/ORNL (*Oak Ridge Leadership Computing Facility no Oak Ridge National Laboratory - TN*), podendo ser conferido no algoritmo 2.3.

```
1. // loop principal para evolucao de geracoes
2. for (iter = 0; iter<maxIter; iter++) {
3.       // Colunas esquerda e direita
4.       for (i = 1; i<=dim; i++) {
5.             // copiar ultima coluna real para coluna
6.             // esquerda de borda (left ghost column)
7.             grid[i][0] = grid[i][dim];
8.             // copiar primeira coluna real para coluna
9.             // direita de borda (right ghost column)
10.            grid[i][dim+1] = grid[i][1];
11.      }
12.      // Linhas superior e inferior
13.      for (j = 0; j<=dim+1; j++) {
14.            grid[0][j] = grid[dim][j];
15.            grid[dim+1][j] = grid[1][j];
16.            }
17.      // Loop sobre as celulas para nova geracao
18.      for (i = 1; i<=dim; i++) {
19.            for (j = 1; j<=dim; j++) {
20.                  // calcula numero de vizinhos
21.                  int numNeighbors = getNeighbors(grid, i, j);
22.
```

```
23.                   // aplicacao das regras do GOL
24.                   if (grid[i][j]==1 && numNeighbors<2)
25.                         newGrid[i][j] = 0;
26.                   else if (grid[i][j]==1 &&
27.                            (numNeighbors==2 || numNeighbors==3))
28.                         newGrid[i][j] = 1;
29.                   else if (grid[i][j]==1 && numNeighbors>3)
30.                         newGrid[i][j] = 0;
31.                   else if (grid[i][j]==0 && numNeighbors == 3)
32.                         newGrid[i][j] = 1;
33.                   else
34.                         newGrid[i][j] = grid[i][j];
35.                   }
36.             }
37.
38.       // troca de arrays para proxima geracao
39.       int **tmpGrid = grid;
40.       grid = newGrid;
41.       newGrid = tmpGrid;
42.       }// Fim do laco principal
```

Algoritmo 2.3 - Principal laço do programa Jogo da Vida (Fonte: OLCF/ORNL)

Este citado programa disponibilizado pela OLCF/ORNL foi avaliado pelos autores do presente texto em um equipamento com a seguinte configuração: *dual* Intel Xeon E5-2660v4 @ 2.00GHz, onde cada CPU conta com 14 núcleos (*cores*). Desta forma a máquina utilizada conta com um total de 28 núcleos homogêneos de processamento, distribuídos em duas CPUs que compartilham o mesmo endereçamento de memória principal de 128 Gb.

O tempo de processamento da versão serial foi avaliado em um tabuleiro bidimensional de **dimensões 2048x2048**, ou seja, com **4.194.304 células,** sem contar as camadas de contorno, necessárias à implementação de uma condição de contorno periódica, de forma que a face esquerda deve ser considerada ligada a face oposta a direita, e a face superior ligada a face inferior (configuração conhecida como malha do tipo *torus*). Executando-se **10.000 iterações**, ou seja, processando 10 mil novas gerações sucessivas do tabuleiro, o tempo de execução medido foi de pouco mais de 5 minutos e 30 segundos (**330,474 segundos**). Vale lembrar ainda que o programa foi compilado com um compilador de linguagem C *PGI community edition* versão 18.10. Este compilador foi escolhido por permitir comparações de desempenho tanto em OpenMP quanto em OpenACC, o qual será mostrado em capítulos seguintes.

Esta versão inicial poderá ser otimizada em seu desempenho computacional ao se utilizar de programação *multithread* por OpenMP. O primeiro passo para introdução das diretivas de OpenMP consiste em analisar o código atual, identificando trechos onde se pode dividir o processamento em sub-tarefas a serem executadas concorrentemente (ou paralelamente). Este passo corresponde à análise de dependência de dados, mencionada anteriormente no texto. Assim, é possível perceber que o laço mais externo (algoritmo 2.3, linha 2), que itera sobre as gerações, não pode ser transformado em uma

região paralela OpenMP, pois há uma dependência entre as operações de cada passo de iteração do laço, ou seja, a próxima geração somente pode ser calculada quando a geração anterior estiver totalmente pronta.

Entretanto, os laços internos, correspondentes às linhas 4, 13 e 19/20 do algoritmo 2.3, realizam tarefas completamente independentes entre as iterações e podem ser paralelizados facilmente. O leitor pode observar que as posições que estão sendo escritas (esquerda das atribuições) não se repetem do lado direito, onde ocorrem as operações de leitura. Isto elimina as dependências do tipo RAW e WAR abordadas anteriormente. Já a dependência do tipo WAW e as que podem ocorrer entre iterações diferentes pode ser eliminada utilizando-se o recurso de duas matrizes: uma para leitura e outra para escrita, explicada mais adiante.

Após a identificação dos citados trechos sem dependência de dados, foi realizada uma tomada de tempo para checar a relevância destes trechos no tempo total de execução, o que permite verificar por *hot spots*. Os três laços citados correspondem por demandas relativas de tempo de computação correspondentes à: 0,19%, 0,02% e 99,75%, respectivamente. Desta forma, percebe-se que há claramente apenas um *hot spot*, que corresponde ao terceiro laço analisado, responsável pelo cálculo das novas gerações do tabuleiro, e que demanda mais de 99% do tempo total de execução (as diferenças de tempo de processamento nos dois primeiros laços, que são muito semelhantes, se devem a questões relacionadas a arquitetura de memória). Entretanto, por questões didáticas neste estudo de caso, todos os laços analisados receberão diretivas OpenMP para geração de laços paralelos, ficando na forma mostrada nos algoritmos 2.4, 2.5 e 2.6.

```
4.       // Colunas esquerda e direita
5.       #pragma omp parallel for
6.       for (i = 1; i<=dim; i++) {
7.             ...
8.             }
```

Algoritmo 2.4 - Laço paralelo para preenchimento de fronteiras esquerda e direita

```
12.      // Linhas superior e inferior
13.      #pragma omp parallel for
14.      for (j = 0; j<=dim+1; j++) {
15.            ...
16.            }
```

Algoritmo 2.5 - Laço paralelo para preenchimento de fronteiras superior e inferior

```
18.       // Laco sobre as celulas para nova geracao
19.       #pragma omp parallel for
20.       for (i = 1; i<=dim; i++) {
21.             for (j = 1; j<=dim; j++) {
22.                   ...
23.                   }
24.             }
```

Algoritmo 2.6 - Laço para cálculo de nova geração do tabuleiro

Cabe ainda citar que, após o laço que percorre as novas gerações do tabuleiro, outro trecho do código poderá ser paralelizado entre as *threads*, ainda que sua relevância em termos de demanda computacional seja de apenas 0,01% do tempo total de execução. Ocorre que ao final de todas as gerações, o número total de células vivas deverá ser exibido como resultado final do programa. Este trecho deverá simplesmente realizar uma contagem (somatório) dos valores de cada célula disposta no tabuleiro, somando o valor um (1) para cada célula viva em uma determinada variável.

Operações como somatórios, produtórios, busca por máximos e mínimos, entre outras, são conhecidas como operações de redução (*reduction*). Por serem muito comuns em códigos numérico-científicos, o OpenMP contempla uma cláusula específica para esta situação, permitindo que o laço paralelo calcule, da maneira eficaz, o resultado de uma operação de redução, conforme já demonstrado anteriormente. A operação paralela de redução foi abordada na seção sobre padrões de programação paralela, onde a figura 2.2(d) demonstra esquematicamente este padrão, o qual combina todos os elementos de uma coleção em um único elemento a partir de uma função combinadora associativa. O trecho que realiza a contagem (somatório) está demonstrado pelo algoritmo 2.7.

```
43.// Somatorio das celulas do tabuleiro
44.int total = 0;
45.   #pragma omp parallel for reduction(+:total)
46.for (i = 1; i<=dim; i++) {
47.       for (j = 1; j<=dim; j++) {
48.             total += grid[i][j];
49.       }
50.}
```

Algoritmo 2.7 - Laço paralelo cálculo de somatória dos valores do tabuleiro por redução

O desempenho da primeira versão OpenMP do código correspondente ao Jogo da Vida foi avaliado com as mesmas configurações e no mesmo equipamento para qual a versão serial gastou 330,474 segundos. Os resultados em relação a tempo de execução, bem como valores de *Speedup* e eficiência paralela podem ser conferidos na tabela 2.1 a seguir, onde, cada *thread* executa em um núcleo da máquina utilizada.

Tabela 2.1 - Desempenho da versão 1 em OpenMP

Nº threads	Tempo(s)	*Speedup*	Eficiência
Serial	330,474	1	100%
2	256,37	~1,289	~64,453%
4	132,531	~2,494	~62,339%
8	70,377	~4,696	~58,697%
16	39.945	~8,273	~51,708%
28 (max)	27.365	~12,076	~43,130%

Conforme se observa na tabela 2.1, a implementação já obteve ganhos de desempenho (*speedup*), apesar deste ganho ter ficado abaixo do *speedup* linear (ideal). Já a eficiência da versão paralela inicia-se com pouco mais de 64%, e reduz-se significativamente até chegar em aproximadamente 43,1%, utilizando-se de todos os 28 núcleos de processamento disponíveis.

As possíveis causas de ineficiência em sistemas paralelos podem ser várias, dentre elas destacam-se:

1. tempo despendido em operações sequenciais (Lei de Amdhal - subcapítulo 3),
2. tempo gasto com comunicações e tarefas de sincronização e,
3. desbalanceamento de carga entre as *threads*/processadores.

Dos itens acima, pode-se descartar o item 3 como possível causa da queda de eficiência observada, pois as tarefas concorrentes presentes nos laços são homogêneas e, portanto, com cargas computacionais idênticas. Para o primeiro item, cabe destacar que a fração serial do código em questão corresponde a toda alocação e iniciação de memória referente ao tabuleiro e seu estado inicial, além do processamento e controle do laço principal entre as gerações de tabuleiro. Esse trecho sequencial corresponde à apenas 0,02% do tempo total de execução da versão serial. Desta forma, na configuração atual, não pode ser considerado um severo fator limitante ao *speedup*. Além deste fato, cabe lembrar que, para a versão paralela, o tempo gasto com abertura e fechamento de *threads* correspondentes às três regiões paralelas implementadas através da diretiva *parallel*, ocorrem 10.000*3 vezes. Assim, este fator limitante corresponde ao item 2, tempo gasto com comunicação e sincronização de processos paralelos. Sua

medida de tempo exata é difícil de ser estimada, pois implica medir tarefas implementadas de forma automática por operações do OpenMP.

Uma possível otimização neste programa poderia ser a paralelização do procedimento que inicia a população do primeiro tabuleiro, entretanto, deve-se lembrar que este trecho apresenta pouca significância no tempo total de computação, conforme citado no parágrafo anterior. Além disso, uma vez que o tabuleiro é iniciado de forma pseudo-aleatória, a partir da geração de números aleatórios de uma semente fixa, a subdivisão desta tarefa em outras subtarefas concorrentes poderia gerar problemas de falta de compatibilidade binária entre a versão serial de referência e as versões paralelas em desenvolvimento, pois, desta forma, o tabuleiro inicial poderia ter seu estado alterado em função da quantidade de *threads*, o que impediria a checagem do resultado final esperado. Assim, durante o desenvolvimento, resolveu-se não alterar a forma da geração do estado inicial, permitindo verificar se o tabuleiro final da versão paralela gera o mesmo resultado da versão serial.

Outra possível otimização consiste em minimizar a quantidade de *threads* criadas e destruídas dentro do laço que controla as gerações. Assim, poderia-se abrir uma determinada quantidade de *threads* paralelas apenas uma única vez. A implementação desta funcionalidade exige, porém, uma maior quantidade de modificações no código serial, ou seja, exigirá do programador mais do que a simples colocação de uma diretiva `parallel for`. A ideia desta refatoração é justamente separar as diretivas `parallel` e `for` e utilizar cláusulas do OpenMP para controlar o escopo das variáveis (compartilhadas ou locais).

Desta forma, as *threads* serão abertas antes do laço que controla as gerações, o qual continuará a ser executado serialmente, porém, por todas as *threads* existentes. Esta mudança exigirá a definição formal do escopo das principais variáveis e *arrays* a serem utilizados na nova macro região paralela a ser criada. Assim, deve-se definir explicitamente quais variáveis/*arrays* terão seus valores compartilhados pelas *threads*, definidas pela cláusula `shared`, e quais serão criadas para serem variáveis locais ou privadas (cláusula `private`) para as *threads* existentes. Os *arrays* `grid` e `newgrid`, além das variáveis apenas de leitura, `maxIter` e `dim`, deverão ser consideradas compartilhadas pelas *threads*, enquanto que as variáveis contadoras de laços `iter`, `i` e `j`, deverão ser privadas, onde em cada *thread* um dado independente será alocado e utilizado.

Outro destaque fica por conta do trecho de código que faz a troca de ponteiros entre a grade antiga e nova, preparando a uma nova geração de população. Esta operação deverá ser efetuada por apenas uma única *thread*, uma vez que a mesma deverá ser atômica. O código resultante, que define a seção paralela com o escopo das variáveis e a operação atômica de troca de ponteiros, pode ser conferido no algoritmo 2.8.

```
1.  // loop principal para evolucao de geracoes
2.  #pragma omp parallel \
3.              shared(grid,newGrid,maxIter,dim) \
4.              private(iter,i,j)
5.  { // principal regiao paralela
```

```
6.  for (iter = 0; iter<maxIter; iter++) {
7.      // Colunas esquerda e direita
8.      #pragma omp for
9.      for (i = 1; i<=dim; i++) {
10.         // copiar ultima coluna real para coluna
11.         // esquerda de borda (left ghost column)
12.         grid[i][0] = grid[i][dim];
13.         // copiar primeira coluna real para coluna
14.         // direita de borda (right ghost column)
15.         grid[i][dim+1] = grid[i][1];
16.     }
17.     // Linhas superior e inferior
18.     #pragma omp for
19.     for (j = 0; j<=dim+1; j++) {
20.         grid[0][j] = grid[dim][j];
21.         grid[dim+1][j] = grid[1][j];
22.         }
23.
24.     // Loop sobre as celulas para nova geracao
25.     #pragma omp for
26.     for (i = 1; i<=dim; i++) {
27.         for (j = 1; j<=dim; j++) {
28.             // calcula numero de vizinhos
29.             int numNeighbors = getNeighbors(grid, i, j);
30.             // aplicacao das regras do GOL
31.             if (grid[i][j]==1 && numNeighbors<2)
32.                newGrid[i][j] = 0;
33.             else if (grid[i][j]==1 &&
34.                      (numNeighbors==2 || numNeighbors==3))
35.                      newGrid[i][j] = 1;
36.             else if (grid[i][j]==1 && numNeighbors>3)
37.                      newGrid[i][j] = 0;
38.             else if (grid[i][j]==0 && numNeighbors == 3)
39.                      newGrid[i][j] = 1;
40.             else
41.                      newGrid[i][j] = grid[i][j];
42.         }
43.     }
44.
45.     // troca de arrays para proxima geracao
46.     #pragma omp single
47.          { // regiao atomica para troca de ponteiros
48.            int **tmpGrid = grid;
49.            grid = newGrid;
50.            newGrid = tmpGrid;
51.          } // fim da regiao atomica
52.     } // Fim do laco principal
53.} // Fim da regiao paralela principal
```

Algoritmo 2.8 - Segunda versão de código OpenMP com otimização na criação de *threads*

Nesta última versão do código com OpenMP percebe-se que os laços paralelos apenas recebem anotações para terem suas iterações subdivididas entre as *threads*, não sendo mais necessário criar a própria *thread*, uma vez que as mesmas já foram criadas anteriormente. Além disso, o trecho que faz a troca de ponteiros recebeu uma cláusula **`single`** indicando que apenas uma *thread* deverá executar o trecho demarcado, e as demais *threads* deverão se abster de executá-lo.

O padrão OpenMP não define explicitamente a forma com que as iterações do laço serão divididas entre as *threads* concorrentes. Divisões em padrões de blocos ou divisões cíclicas, além de combinações de ambas, são igualmente aplicáveis e podem ser explicitamente definidas pelo programador se desejado. No estudo de caso em questão, a divisão de tarefas para os laços paralelos entre as *threads* é deixada a cargo da forma padrão (*default*) do OpenMP. Usualmente, a forma padrão divide as tarefas em blocos contíguos entre *threads* paralelas, de forma a manter uma divisão mais homogênea de carga possível.

Os resultados, comparados com a versão anterior, foram melhores, conforme se pode checar nas figuras 2.5 e 2.6. Percebe-se que a segunda versão consegue atingir, para o número máximo de *threads* (28 *threads*), um ***speedup*** de, aproximadamente, **21,4** em contraste com 12,1 da primeira versão. Já a **eficiência** paralela passou de 43,13% para **76,33%**, atestando a validade da transformação aplicada ao código. Cabe ainda citar que o trecho relativo ao cálculo da somatória dos valores no tabuleiro por redução, previamente mostrado, permanece inalterado nas duas versões. O *speedup* paralelo linear, exibido na figura 2.5, refere-se ao *speedup* exatamente igual a quantidade de *threads* em processamento, e serve como um parâmetro de comparação para estimar a distância do desempenho atingido/medido com o desempenho ótimo teórico, caso a paralelização não enfrentasse nenhum limitante, o que não ocorre na prática, na grande maioria das vezes.

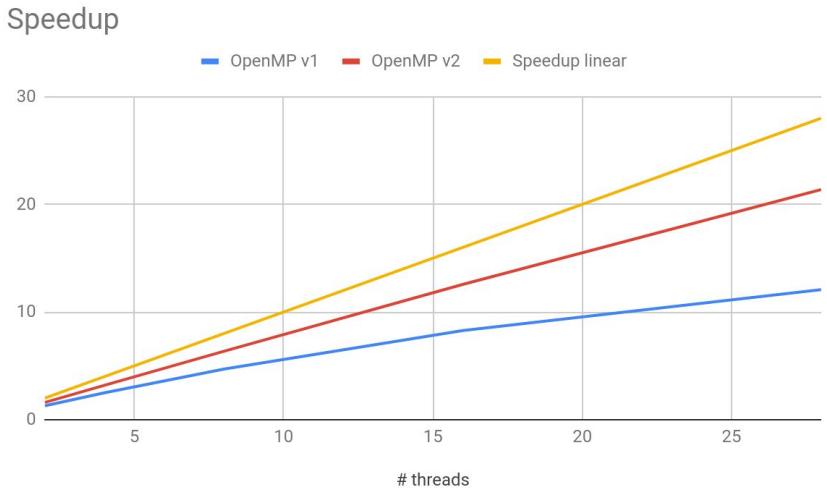


Figura 2.5 - *Speedup* para as versões OpenMP desenvolvidas

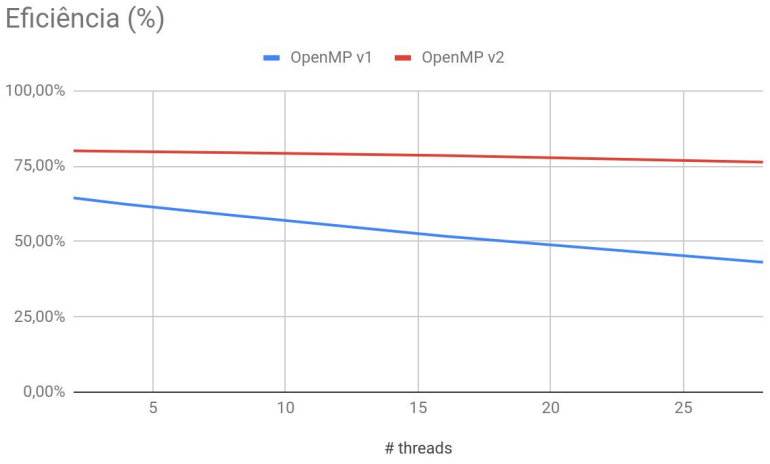


Figura 2.6 - Eficiência paralela para as versões OpenMP desenvolvidas

2.5. Programação para GPUs

As GPUs (*Graphics Processing Units*) estão entre os dispositivos conhecidos por **aceleradores**, os quais, na maioria das vezes, atuam como co-processadores, ou seja, unidades extra de processamento de código dependente de um processador tradicional. As unidades aceleradoras podem estar fisicamente conectadas à um nó de processamento que conta com um processador tradicional, ou representarem um nó computacional por completo.

No momento da escrita deste capítulo, cinco entre as dez primeiras máquinas da lista Top 500 (TOP500, 2018) possuem placas aceleradoras do tipo GPU da NVidia. Além de serem aceleradores eficientes, as GPUs têm um baixo consumo de energia comparados aos processadores *multicores*, por isso também são opções adotadas em "computação verde" (*green computing*) .

Este subcapítulo tem por finalidade apresentar os principais conceitos por parte dos aceleradores do tipo GPU e demonstrar as principais técnicas para programação do mesmo, com detalhamento do padrão OpenACC

As GPUs são compostas por centenas ou até milhares de núcleos (*cores*) simples que, normalmente, executam o mesmo código através de centenas a milhares de *threads* concorrentes. Tal característica se opõe ao modelo tradicional de processadores *multicore*, onde algumas unidades de núcleos complexos são capazes de executar *threads* ou processos independentes. Assim, ao se considerar o processamento paralelo em GPUs, será necessário introduzir o conceito para um modelo de computação conhecido por SIMT (*Single Instruction Multiple Threads*), derivado do clássico termo SIMD (*Single Instruction Multiple Data*). A arquitetura das GPUs não será detalhada neste capítulo por ser altamente especializada. Maiores detalhes das mesmas poderão ser encontrados em Kirk (2010).

Esta seção faz uma breve introdução do modelo de abstração mais conhecido para este tipo de arquitetura, conhecido por CUDA (*Compute Unified Device*

Architecture) (NVIDIA, 2018), a qual ajuda a padronizar a programação e o uso de GPUs ao definir um modelo de programação comum a todos os diferentes tipos de placas que a fabricante disponibiliza. Entretanto, a programação CUDA não será detalhada, servindo apenas de base introdutória para a programação seguindo o padrão OpenACC.

2.5.1 Programação CUDA

CUDA é uma plataforma de computação paralela, livremente distribuído pela empresa NVIDIA, aplicada a computação de aspectos gerais em GPUs. Constitui-se de uma série de extensões para a linguagem C/C++, juntamente com uma API (*Application Programming Interface*) que define algumas funções para manipulação destes dispositivos. O modelo de programação assume que o sistema é composto de um *host* (CPU) e de um dispositivo (*device* ou GPU), que funcionará com o um co-processador.

A programação consiste em definir o código de uma ou mais funções que executarão no dispositivo (conhecidos por *kernels*) e de uma ou mais funções que executarão no *host* (a função principal de um programa C, *main()*, por exemplo). Quando um *kernel* é invocado, centenas ou até milhares de *threads* são iniciadas no dispositivo, executando simultaneamente o código descrito no *kernel*. Os dados utilizados devem estar na memória do dispositivo e CUDA oferece funções para realizar esta transferência.

O Algoritmo 2.9 apresenta um exemplo de código em CUDA que implementa a soma de matrizes no dispositivo. O comando de invocação do *kernel* define a quantidade de *threads* dimensionadas em um bloco e a dimensão de uma grade de blocos. Resumidamente, as *threads* são organizadas em blocos de até três dimensões, e estes blocos compõem uma grade. Este mapeamento por vezes é considerado complexo e necessita de uma atenção maior do programador. O exemplo não aborda características um pouco mais complexas da programação CUDA, tais como o gerenciamento de memória (os dados devem ser transferido para o dispositivo antes da execução e trazidos de volta para a memória principal após o processamento) nem o uso de blocos maiores. Recomenda-se o próprio manual CUDA *Toolkit* disponível em (NVIDIA, 2018).

A partir do exemplo de código apresentado, no entanto, é possível observar algumas das principais características da programação CUDA. São elas:

- uso da palavra-chave `__global__` que indica que a função é um *kernel* e que só poderá ser invocada a partir do código executado no *host*, criando uma grade de *threads* que executarão no dispositivo (linha 02);
- uso das variáveis pré-definidas `threadIdx.x` e `threadIdx.y` que identificam a *thread* dentro do bloco através de suas coordenadas (linhas 6 e 7);
- uso do tipo pré-definido `dim3` (linha 16) para definir um bloco de *threads* com mais de uma dimensão;

- uso dos símbolos `<<<...>>>` (linha 17) para invocar várias instâncias do *kernel* no dispositivo de acordo com a quantidade de blocos e de *threads* por bloco indicada entre eles.

```
01 ...
02 // Kernel definition
03 __global__ void MatAdd(float A[N][N], float B[N][N],
04                         float C[N][N]) {
05     int i = threadIdx.x;
06     int j = threadIdx.y;
07     C[i][j] = A[i][j] + B[i][j];
08 }
09
10 int main() {
11     ...
12  // Kernel invocation with one block of N*N*1 threads
13     int numBlocks = 1;
14     dim3 threadsPerBlock(N, N);
15     MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
16     ...
   }
```

Algoritmo 2.9 - Trecho de código em CUDA somar duas matrizes.
Fonte: (NVIDIA, 2018)

2.5.2. Programação para aceleradores seguindo o padrão OpenACC

Embora o uso de CUDA seja a forma mais direta de programar GPUs, diversas opções têm surgido com o objetivo de tornar a programação mais acessível. Entre elas se destacam o uso de diretivas suportadas pelo OpenMP, na sua versão 4.0 e 4.5 (Van der Pas, 2017) e OpenACC (OPENACC, 2019).

O uso do padrão OpenACC consiste em um modelo para se expressar paralelismo baseado em diretivas aplicadas a linguagem C/C++ ou Fortran. Este modo de se expressar paralelismo reduz consideravelmente a complexidade do código-fonte empregado em comparação com CUDA, por permitir automatização de diversas tarefas comumente executadas em GPUs, como, por exemplo, o gerenciamento de memória. O padrão OpenACC emergiu de iniciativas isoladas de diversas empresas tal como a empresa *Portland Group* (PGI), a qual pode ser encontrada na pesquisa de Lee et al (2009), consistindo de uma proposta de um tradutor automático de código paralelo escrito com diretivas, como no OpenMP, para a linguagem CUDA, bem como nos projetos denominados CUDA-Lite (Ueng *et al*, 2008) e hiCUDA (Han e Abdelrahman, 2009).

Este subcapítulo introduz o modelo de programação OpenACC através de exemplos em linguagem C, incluindo uma aplicação completa conhecida por Jogo da Vida, já apresentada no subcapítulo dedicado ao OpenMP. Destacam-se as principais

vantagens na programação de aceleradores através deste padrão, bem como suas possíveis limitações e as formas de se mitigá-las.

Atualmente, alguns compiladores são disponíveis para uso, alguns com licença proprietária, como o compilador distribuído exclusivamente pela fabricante de supercomputadores Cray, de uso exclusivo dos equipamentos da marca, e outros permitindo o livre uso para finalidades não comerciais, como é o caso do compilador *Portland Group* em sua versão conhecida por *community edition*, o qual será utilizado nos experimentos práticos de programação deste curso.

Convém ainda citar que o padrão OpenACC foi criado com propósitos gerais, não sendo considerado uma ferramenta exclusiva para uso com GPUs. Assim, é possível utilizar um código aderente a este padrão para acelerar programas executando na própria CPU, de certa forma, rivalizando com o padrão OpenMP descrito anteriormente. Entretanto, por entender que o OpenMP já supre as necessidade de programação para sistemas de memória compartilhadas, além de ser um padrão muito bem conhecido na área, este texto limitar-se-á a aplicar códigos em OpenACC para execução em GPUs.

O próprio padrão OpenMP, em suas recentes versões 4, 4.5 e 5, prevê a aceleração de trechos de código para GPUs e demais possíveis aceleradores através do uso de diretivas. Porém, não existe, até o momento da escrita deste capítulo, um compilador que implemente todas as funcionalidades previstas nas últimas versões do padrão OpenMP para execução conhecida por *offloading*, desta forma, essa funcionalidade não será abordada neste capítulo.

2.5.2.1. Criação de regiões paralelas aceleradas com OpenACC

A criação de regiões aceleradas por diretivas do padrão OpenACC em um código-fonte escrito em linguagem C é simples e ocorre de forma muito similar ao bem conhecido padrão OpenMP. Duas diretivas principais podem ser utilizadas para esta finalidade, são elas:

`#pragma acc kernels`	`#pragma acc parallel`

Ambas permitem definir um trecho de código para ser executado em paralelo na GPU, porém, a primeira diretiva `kernels` é usada de forma mais conservativa, demandando análise automática do compilador do trecho delimitado procurando por possíveis dependências de dados, decidindo autonomamente se o trecho deve ou não ser paralelizado; enquanto que a diretiva `parallel` define uma região paralela de forma definitiva, onde a garantia da não existência de dependências deve ser dada pelo programador.

Para os trechos de códigos exibidos a seguir (Algoritmo 2.10), os quais realizam, ambos, uma simples operação de multiplicação seguida de soma (a bem conhecida operação *multiply and add*), temos duas possíveis opções de paralelização por OpenACC.

```
void saxpy(int n, float a,
        float *x,
        float *restrict y) {
     #pragma acc kernels
     #pragma acc loop
     for (int i = 0; i < n; ++i)
         y[i] = a * x[i] + y[i];
     }
```

```
void saxpy(int n, float a,
        float *x,
        float *restrict y) {
     #pragma acc parallel
     #pragma acc loop
     for (int i = 0; i < n; ++i)
         y[i] = a * x[i] + y[i];
     }
```

Algoritmo 2.10 - Duas versões de criação de região paralela acelerada

Nas duas versões do código exibido no algoritmo 2.10, o argumento `float *restrict y` define que o endereço de memória apontado por `*y` será exclusivo e, portanto, não apontará para outras áreas de memória referenciadas por outros ponteiros presentes nos argumentos da função `saxpy`. Assim, o programador garante que não haverá dependência entre os arrays `*x` e `*y`, pois ambos não apontam para uma mesma área de memória, o que poderia impedir a paralelização do laço.

Ambas as versões deverão produzir desempenho semelhante, uma vez que o código utilizando a diretiva `kernels` não deverá identificar dependências que impeçam a paralelização. Assim, o laço será transformado em um código binário a ser executado em GPU, utilizando-se de múltiplas *threads*, em múltiplos blocos, para executar de forma maciçamente paralela.

A cláusula `loop` indica ao compilador para paralelizar um laço imediatamente abaixo da mesma. Pode ser usada também de forma contraída e integrada na mesma linha da cláusula que indica a região acelerada, como em: `#pragma acc parallel loop`. Após a diretiva `loop` pode-se especificar, opcionalmente, outras cláusulas para: i) determinar escopo de variáveis que estão presentes no laço, e a forma de transferência destas para o dispositivo; ii) definir variáveis que irão sofrer processo de redução (`reduction`); iii) colapsar laços aninhados para paralelismo (`collapse`); iv) especificar a configuração lógica de quantidade de *threads* paralelas (`vector`) e de blocos de *threads* (`gang`) em GPUs; e v) indicar que o processamento interno ao laço deve dar-se na forma SIMD (*Single Instruction, Multiple Data*), ou seja, aplicar o mesmo conjunto de instruções para cada *thread* paralela, porém atuando em conjuntos de dados diferentes. Algumas destas características serão vistas a seguir.

A compilação de um código em linguagem C com diretivas no padrão OpenACC, utilizando-se do compilador PGI com chaves usuais, deve ser feito de forma semelhante a linha de compilação do bem conhecido compilador *gcc*, como no exemplo a seguir:

```
pgcc -acc [-Minfo=accel] [-ta=nvidia] [-o exemplo.x] exemplo.c
```

Onde a chave de compilação `-acc` habilita o compilador a reconhecer o padrão OpenACC, enquanto que as chaves `-Minfo` e `-ta` são opcionais, e definem, respectivamente, a exibição de informações sobre o procedimento de aceleração realizado e a especificação do dispositivo para o qual o programa binário deverá ser gerado. Além das chaves opcionais aqui descritas, existem muitas outras que podem ser consultadas no manual ou páginas de ajuda do compilador.

Um procedimento comum em programação paralela consiste em calcular reduções em operações de somatórios, produtórios, extração de máximo valor, etc, o qual também foi previsto em OpenACC, de forma similar ao visto em OpenMP, e que pode ser verificado no algoritmo 2.11.

```
float saxpy_sum(int n, float a, float *x, float *restrict y) {
     float total = 0.;
     #pragma acc parallel loop reduction (+:total)
     for (int i = 0; i < n; ++i) {
          y[i] = a * x[i] + y[i];
          total += y[i]; }
     return total;
     }
```

Algoritmo 2.11 - Exemplo de operação de redução com OpenACC

O padrão OpenACC permite ainda definir nas regiões paralelas o local em que as transferências entre *host* e *device* serão executadas, bem como o escopo dos dados transferidos, ou seja, se são dados apenas utilizados para leitura dentro do dispositivo, se são de leitura e escrita, se são criados no dispositivo e transferidos de volta ao *host* ao final da região, ou ainda se são dados temporários, usados de forma auxiliar pelo algoritmo que executa na GPU, mas que não necessitam ser sincronizados com o *host*. O exemplo no algoritmo 2.12 ilustra o uso de diretivas de transferência de dados, onde a diretiva `copyin` especifica que a transferência do *array* "`x`" será feita no local indicado apenas no sentido CPU→ GPU, pois os dados serão utilizados apenas no modo de leitura pelo *kernel* executando no dispositivo. Entretanto, as informações do *array* "`y`", contidas na diretiva `copy`, devem ser transferidas do *host* para *device* antes da execução do *kernel*, como dado de entrada, e transferida de volta, no sentido GPU→ CPU, ao término da região paralela, como um dado de saída.

Um programa com diretivas OpenACC realiza, por definição, as transferências de dados de forma automática, analisando, no momento da compilação, a região acelerada do código e verificando quais dados são de entrada, saída ou temporários. Entretanto, o compilador toma decisões que podem ser consideradas conservadoras, assim, em muitas oportunidades, o programador deve intervir, alterando a forma de transferência dos dados de maneira a buscar a melhoria no desempenho computacional, uma vez que o tempo despendido neste processo de transferência não pode ser desprezado, podendo afetar negativamente no desempenho do programa. Outro exemplo envolvendo o mesmo assunto será tratado na seção seguinte, para melhorar o entendimento.

```
float saxpy_sum(int n, float a, float *x, float *restrict y) {
     float total = 0.;
     #pragma acc parallel copyin(x[0:n]) copy(y[0:n])
          {
          #pragma acc loop reduction (+:total)
          for (int i = 0; i < n; ++i) {
               y[i] = a * x[i] + y[i];
               total += y[i]; }
          } // fim da região paralela com transf. de dados
     return total;
     }
```

Algoritmo 2.12 - Exemplo de operação de redução com OpenACC

As demais possíveis cláusulas relacionadas a dados em construções paralelas, para códigos em OpenACC, são as seguintes:

- `copy(lista de variáveis e arrays)` - Aloca área de memória para os dados especificados na lista no acelerador, e copia-os do *host* para o *device* (CPU→ GPU) ao entrar na região, e do *device* para o *host* ao sair da região delimitada (GPU→CPU).
- `copyin(lista de variáveis e arrays)` - Aloca área de memória para os dados especificados na lista no acelerador, e copia-os do *host* para o *device* (CPU→GPU) ao entrar na região.
- `copyout(lista de variáveis e arrays)`- Aloca área de memória para os dados especificados na lista no acelerador, e copia-os do *device* para o *host* (CPU→GPU) ao sair da região.
- `create(lista de variáveis e arrays)`- Aloca área de memória para os dados especificados na lista no acelerador, e não realiza nenhuma transferência de dados entre *host* e *device*. Normalmente aplicável a dados temporários.
- `present(lista de variáveis e arrays)` - Indica que os dados citados na lista já estão presentes (ou alocados) no acelerador e devem ser usados. Também não realiza nenhuma transferência de dados entre *host* e *device*.
- `present_or_copy(lista de variáveis e arrays)` - Indica que, caso os dados citados na lista já estejam presentes (ou alocados) no acelerador devem ser usados e nenhuma transferência será realizada. Entretanto, caso os dados da lista não estejam presentes, procede da mesma forma que a cláusula `copy`.
- `present_or_copyin(lista de variáveis e arrays)` - Indica que, caso os dados citados na lista já estejam presentes (ou alocados) no acelerador devem ser usados e nenhuma transferência será realizada. Entretanto, caso os dados da lista não estejam presentes, procede da mesma forma que a cláusula `copyin`.

- `present_or_copyout(lista de variáveis e arrays)` - o mesmo que o anterior, mas para a cláusula `copyout`.
- `present_or_create(lista de variáveis e arrays)` - o mesmo que o anterior, mas para a cláusula `create`.
- `deviceptr(lista de variáveis e arrays)` - A lista deve especificar ponteiros para endereços de memória de dados já alocados no acelerador (tal como usado na função `acc_malloc`).

2.5.2.2. Estudo de caso com o programa Jogo da Vida

Da mesma forma como descrito na seção 2.4.1, será utilizado o programa Jogo da Vida como um estudo de caso para paralelização do código, seguindo o padrão OpenACC, a ser utilizado em um acelerador de hardware do tipo GPU. A versão sequencial a ser tomada como base será a mesma descrita na seção mencionada.

A análise de desempenho foi anteriormente efetuada para OpenMP, e identificou quatro regiões com potencial paralelismo:

a) Laço que atualiza valores de bordas periódicas a esquerda e direita do tabuleiro;
b) Laço que atualiza valores de bordas periódicas superior e inferior;
c) Laço que processa uma nova geração do tabuleiro (com maior relevância de carga computacional);
d) Laço que calcula o valor da somatória dos valores do tabuleiro.

As três primeiras regiões potenciais (ou trechos) estão aninhadas em outro laço, o qual itera sobre as gerações sucessivas do tabuleiro, enquanto o laço descrito pelo item (d) está isolado dos demais, conforme já citado na seção 2.4.1. Para a paralelização por OpenACC as mesmas regiões serão consideradas. Assim, uma primeira versão do algoritmo paralelo em OpenACC consiste em utilizar a diretiva **`parallel`** para paralelizar todos as regiões previamente descritas, com o devido cuidado com a última região que exigirá uma operação de redução, tal qual também ocorreu em OpenMP. Assim, a primeira versão que paraleliza os quatro laços a seguir pode ser conferida nos algoritmos 2.13, 2.14, 2.15 e 2.16.

```
4.       // Colunas esquerda e direita
5.       #pragma acc parallel
6.       #pragma loop
7.       for (i = 1; i<=dim; i++) {
8.             ...
9.             }
```

Algoritmo 2.13 - Laço paralelo em OpenACC para preenchimento de fronteiras esquerda e direita

```
12.       // Linhas superior e inferior
13.       #pragma acc parallel
14.       #pragma loop
15.       for (j = 0; j<=dim+1; j++) {
16.             ... }
```

Algoritmo 2.14 - Laço paralelo em OpenACC para preenchimento de fronteiras superior e inferior

```
18.       // Loop sobre as celulas para nova geracao
19.       #pragma acc parallel
20.       #pragma acc loop
21.       for (i = 1; i<=dim; i++) {
22.             for (j = 1; j<=dim; j++) {
23.                   ...
24.                   }
25.             }
```

Algoritmo 2.15 - Laço paralelo em OpenACC para cálculo de nova geração do tabuleiro

Cabe citar que tanto a diretiva `parallel` quanto a diretiva `kernels` poderiam ser igualmente aplicadas aos laços citados. Deve-se lembrar porém, que a diretiva `kernels` é utilizada em uma abordagem conservadora, deixando para o compilador a tarefa de analisar o trecho de código e decidir ou não pela geração de versão paralela acelerada, enquanto que a diretiva `parallel` força o compilador a paralelizar o trecho demarcado, deixando a responsabilidade para o programador definir sobre a aceleração da região. Nos exemplos aqui utilizados serão utilizados a diretiva `parallel` por conveniência.

O trecho relativo a operação de redução, para cálculo da somatória dos valores do tabuleiro, será paralelizado em OpenACC de acordo com o algoritmo 2.16.

```
43.   // Somatorio das celulas do tabuleiro
44.   int total = 0;
45.   #pragma acc parallel loop reduction(+:total)
46.   for (i = 1; i<=dim; i++) {
47.      for (j = 1; j<=dim; j++) {
48.            total += grid[i][j];
49.            }
50.      }
```

Algoritmo 2.16 - Laço paralelo em OpenACC para cálculo de somatória dos valores do tabuleiro por redução

Entretanto, ao compilar esta primeira versão e analisar as informações exibidas na tela pelo compilador PGI, percebeu-se algo fora do esperado, relacionado a transferência de dados entre CPU e GPU. O compilador implicitamente fez estas transferências para todas as quatro regiões paralelas, porém, na terceira região paralela, a transferência de dados não contemplou todos os elementos das matrizes **`grid`** e **`newGrid`** exigidas, conforme se vê abaixo:

```
80, Accelerator kernel generated
      Generating Tesla code
      82, #pragma acc loop gang /* blockIdx.x */
      83, #pragma acc loop vector(128) /* threadIdx.x */
  80, Generating implicit copyin(grid[1:2048][1:2048])
      Generating implicit copy(newGrid[1:2048][1:2048])
  85, getNeighbors inlined, size=16, file GOL-openacc2-v1.c (8)
       83, Loop is parallelizable
```

No texto acima referente a saída do compilador, a informação `Generating implicit copyin(grid[1:2048][1:2048])` bem como a informação `Generating implicit copy(newGrid[1:2048][1:2048])` significam que o *array* `grid` sofrerá uma transferência, da CPU para a GPU, para os dados que abrangem as células na faixa entre as linhas e colunas que vão de 1 até 2048. O array `newGrid` sofreu processo semelhante, porém, após o processamento no dispositivo, ocorrerá uma transferência no sentido oposto, ou seja, da GPU para CPU, abrangendo a mesma faixa de valores já citada. Ocorre que a aplicação das regras do Jogo da Vida exigirão que se façam leituras nos valores nas posições correspondentes às bordas das matrizes, ou seja, em valores nas linhas 0 e 2050, bem como nas colunas de mesma numeração. Entretanto, o compilador não detectou a necessidade de transferir valores contidos nesta faixa (nas bordas do tabuleiro), desta forma, não foram transferidos para a GPU, implicando em um *array* de tamanho menor a ser alocado no acelerador.

O resultado desta transferência incompleta de dados, que não contempla todos os valores necessários, acabou gerando um erro de execução. Assim, foi necessário corrigir o problema indicando explicitamente para o compilador a faixa de dados a ser utilizada nas transferências. O código que corrige este problema, relativo exclusivamente ao terceiro laço, pode ser conferido no algoritmo 2.17. As cláusulas `copyin` e `copy` especificam a exata faixa de valores das matrizes `grid` e `newgrid`, respectivamente, conforme exigido, iniciando no elemento de índice 0 até o último elemento dos *arrays,* em ambas as dimensões da matriz.

```
18.       // Loop sobre as celulas para nova geracao
19.       #pragma acc parallel \
20.             copyin(grid[0:fullSize][0:fullSize]) \
```

```
21.             copy(newGrid[0:fullSize][0:fullSize])
22.       #pragma acc loop
23.       for (i = 1; i<=dim; i++) {
24.             for (j = 1; j<=dim; j++) {
25.                   ...
26.                   }
27.             }
```

Algoritmo 2.17 - Laço paralelo corrigido em OpenACC para cálculo de nova geração do tabuleiro, contemplando toda a matriz

O desempenho desta primeira versão em OpenACC em comparação com a versão serial e com a melhor versão paralela em OpenMP com 28 threads foi muito abaixo do esperado, e pode ser conferido na tabela 2.2. A execução da versão em GPU por OpenACC se deu em uma GPU NVIDIA *TitanBlack*, a qual conta com 2880 núcleos (chamados de *cuda cores*) executando à 889 MHz, e memória de 6GB.

Tabela 2.2 - Desempenho da primeira versão paralela em GPU por OpenACC

Versão	Tempo (s)	Speedup
Serial	330,474	1
OpenMP v2 - 28 threads	15,463	21,37
OpenACC v1	1208,54	0,27

O desempenho considerado inesperado e pífio, obtido pela primeira versão em OpenACC, pode ser explicado devido ao tempo despendido com transferências de dados entre a CPU e a GPU, pois, uma vez que as três primeiras regiões paralelizadas estão dentro de um laço que se repete 10.000 vezes, essas transferências ocorrem 70.000 vezes durante o tempo de execução do código, sendo 20.000 correspondente ao primeiro laço, devido a 10.000 transferência CPU→ GPU antes do início e 10.000 transferências GPU→ CPU ao término do laço; mais 20.000 vezes correspondente ao segundo laço de forma semelhante, ambos devido ao array `grid`, e 30.000 para o terceiro laço, já que ocorre a transferência CPU→ GPU de `grid` e `newGrid` antes do início do mesmo, e apenas de `newGrid`, no sentido GPU→ CPU, ao término do laço.

Uma segunda versão deste código deverá minimizar a enorme quantidade de transferências de dados que ocorrem entre CPU e GPU. Desta forma, similarmente à técnica aplicada para a segunda versão em OpenMP, o laço principal, que percorre as sucessivas gerações do tabuleiro, receberá uma diretiva em OpenACC que indicará a transferência inicial de dados da CPU para a GPU, e a transferência no sentido oposto (GPU→ CPU) ao término desta seção. Esta nova versão deverá, portanto, delimitar uma região de dados, que compreende todo o laço principal, especificando que o *array* `grid` deverá ser transferido tanto na entrada quanto na saída, nos sentidos CPU→ GPU e

GPU→ CPU: `copy(grid[0:fullSize][0:fullSize])`, enquanto o *array* `newGrid` deverá ser utilizado apenas internamente, portanto podendo ser apenas criado por lá no dispositivo: `create(newGrid[0:fullSize][0:fullSize])`, pois não necessita ser retirado, já que após o seu cálculo, todo o seu conteúdo deverá ser copiado para o *array* `grid`, preparando-se para uma nova possível geração.

Um ressalva importante para esta nova versão do programa em OpenACC refere-se ao trecho de código responsável pela troca de ponteiros entre `grid` e `newGrid`, a qual terá de ser substituída por uma simples cópia entre os dois citados *arrays*, visto que não será possível efetuar a troca de ponteiros do código em um ambiente massivamente paralelo como a GPU, onde os ponteiros estão distribuídos pelos diversos módulos de processamento. Desta forma, o código final resultante pode ser conferido no algoritmo 2.18.

```
1. // loop principal para evolucao de geracoes
2. #pragma acc data copy(grid[0:fullSize][0:fullSize]) \
3.                  create(newGrid[0:fullSize][0:fullSize])
4. { // principal regiao paralela acelerada
5.       for (iter = 0; iter<maxIter; iter++) {
6.             // Colunas esquerda e direita
7.             #pragma acc parallel loop
8.             for (i = 1; i<=dim; i++) {
9.                   // copiar ultima coluna real para coluna
10.                  // esquerda de borda (left ghost column)
11.                  grid[i][0] = grid[i][dim];
12.                  // copiar primeira coluna real para coluna
13.                  // direita de borda (right ghost column)
14.                  grid[i][dim+1] = grid[i][1];
15.                  }
16.            // Linhas superior e inferior
17.            #pragma acc parallel loop
18.            for (j = 0; j<=dim+1; j++) {
19.                  grid[0][j] = grid[dim][j];
20.                  grid[dim+1][j] = grid[1][j];
21.                  }
22.
23.            // Loop sobre as celulas para nova geracao
24.            #pragma acc parallel loop
25.            for (i = 1; i<=dim; i++) {
26.                  for (j = 1; j<=dim; j++) {
27.                    // calcula numero de vizinhos
28.              int numNeighbors = getNeighbors(grid,i, j);
29.                    // aplicacao das regras do GOL
30.                    if (grid[i][j]==1 && numNeighbors<2)
31.                          newGrid[i][j] = 0;
32.                    else if (grid[i][j]==1 &&
33.                         (numNeighbors==2||numNeighbors==3))
34.                          newGrid[i][j] = 1;
35.                    else if (grid[i][j]==1 && numNeighbors>3)
36.                          newGrid[i][j] = 0;
37.                    else if (grid[i][j]==0 && numNeighbors==3)
38.                          newGrid[i][j] = 1;
39.                    else
```

```
40.                         newGrid[i][j] = grid[i][j];
41.                     }
42.             }
43.
44.             // troca de arrays para proxima geracao
45.             #pragma acc parallel loop
46.             for(i = 1; i <= dim; i++) {
47.                   for(j = 1; j <= dim; j++) {
48.                         grid[i][j] = newGrid[i][j];
49.                         }
50.                   } // copia de dados entre grid e newGrid
51.    } // Fim do laco principal
52.} // Fim da regiao paralela principal
```

Algoritmo 2.18 - Segunda versão de código OpenACC com otimização na transferência de dados entre CPU e GPU

O desempenho da segunda versão OpenACC pode ser conferido na tabela 2.3, onde se demonstra um *speedup* de pouco mais de 50 vezes no tempo de execução em relação a versão serial e de 2,37 vezes a melhor versão OpenMP executada.

Tabela 2.3 - Desempenho da primeira versão paralela em GPU por OpenACC

Versão	Tempo (s)	*Speedup*
Serial	330,474	1
OpenMP v2 - 28 threads	15,463	21,37
OpenACC v1	1208,54	0,27
OpenACC v2	6,50	50,84 (2,37 em relação a versão OpenMP v2)

2.6. Conclusão

O capítulo aqui apresentado abordou, de forma prática, através de exemplos em linguagem C, a programação por diretivas para processadores *multicores* e GPUs, utilizando-se de OpenMP e OpenACC, respectivamente. Espera-se que com o texto tutorial e os exemplos aqui demonstrados os leitores tenham capacidade de empregar corretamente as diretivas considerando as características específicas das arquiteturas paralelas de memória compartilhada sugeridas.

Referências bibliográficas

Adamatzky, A. (2010). “Game of Life Cellular Automata”. Springer-Verlag London Limited, 2010.

Amdahl, G. M. (1967). “Validity of the single processor approach to achieving large scale computing capabilities”, AFIPS Spring Joint Computer Conference, 1967.

Ben-Ari, M. (2005). “Principles of Concurrent and Distributed Programming”, Pearson, 2 edition, 2005.

Bernstein, A. J. Analysis of programs for parallel processing. IEEE Trans. on El. Computers, EC-15:757–762, 1966.

Breshears, C. (2009). “The art of concurrency: a thread monkey´s guide to writing parallel applications”. O´Reilly, 2009.

Butenhof, D. R. (2006) “Programming with POSIX Threads”, Addison-Wesley, 2006.

Chapman, B.; Jost, G.; Van der Pas, R. (2008) “Using OpenMP – Portable shared memory parallel programming”. The MIT Press, 2008.

Dowd, K., Severance, C. “High Performance Computing”. 2nd Edition, O'Reilly Media, 1998.

Escobar, Fernando A ; Xin Chang ; Valderrama, Carlos (2016) Suitability Analysis of FPGAs for Heterogeneous Platforms in HPC. IEEE Transactions on Parallel and Distributed Systems, Vol.27(2), pp.600-612. Fevereiro, 2016.

Flynn, Michael J. (1972). "Some Computer Organizations and Their Effectiveness". IEEE Transactions on Computers. C-21 (9): 948–960.doi:10.1109/TC.1972.5009071.

Gilge, M. (2014) IBM System Blue Gene Solution Blue Gene/Q: Application Development, IBM redbooks series, ISBN: 9780738438238, 2014.

Gustafson, J.L. Reevaluating Amdahl's law, Communications of the ACM, Vol. 31, Issue 5, May 1988.

Han, T.D.; Abdelrahman, T.S. hiCUDA: a high-level directive-based language for GPU programming. Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, Washington, D.C., p. 52-61, 2009.

Hassan, Mohamed W.; Helal, Ahmed E.; Athanas, Peter M.; Feng, Wu-chun, Hanaf;, Yasser Y. (2018) Exploring FPGA-specific Optimizations for Irregular OpenCL Applications. Em Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig), Cancun, Mexico, Dezembro de 2018.

Hennessy, D; Patterson, D. (2007) “Computer architecture”. 4ª. Ed, Elsevier, 2007.

Kirk, D. B., Hwu, W.W. (2010) “Programming massively parallel processors: a hands-on approach”. Morgan Kaufman, 2010.

Mattson, T.G., Sanders, B., Massingill, B. “Patterns for Parallel Programming”, Addison-Wesley Professional, 2004.

McCool, M., Reinders, J. and Robinson, A. (2012) Structured Parallel Programming: Patterns for Efficient Computation, Morgan Kaufmann, 1st edition.

NVIDIA (2018) "CUDA Toolkit Documentation", NVIDIA. Version 10.0.130, Outubro 2018. Disponível em https://docs.nvidia.com/cuda. Acessado em janeiro de 2019.

OPENACC (2019) OpenACC website disponível em https://www.openacc.org. Acessado em janeiro de 2019.

Pacheco, P.S. "An Introduction to Parallel Programming", Morgan Kaufmann, 2011.

Stallings, W. (2009) "Computer organization and architecture: designing for perfomance", 8ª. Ed., Prentice Hall, 2009.

TOP500 (2018) disponível em http://www.top500.org, acessado em janeiro de 2019.

Ueng, s.; Lathara, M.; Baghsorkhi, S.S.; Hwu, W.; CUDA-lite: Reducing GPU Programming Complexity. Languages and Compilers for Parallel Computing: 21th International Workshop, LCPC 2008, Edmonton, Canada, July 31 - August 2, 2008.

Van der Pas, Ruud; Stotzer, Eric; Terboven, Christian. (2017) Using OpenMP—The Next Step: Affinity, Accelerators, Tasking, and SIMD. The MIT press, Scientific and Engineering Computation series. Outubro, 2017

Capítulo 3

Correntes de Blocos: Algoritmos de Consenso e Implementação na Plataforma Hyperledger Fabric

Gabriel Antonio F. Rebello[1], Gustavo F. Camilo[1], Leonardo G. C. Silva[1], Lucas Airam C. de Souza[1], Lucas C. B. Guimarães[1], Eduardo A. P. Alchieri[3], Fabíola Greve[2] e Otto Carlos M. B. Duarte[1]

[1] Universidade Federal do Rio de Janeiro - GTA/PEE/COPPE
[2] Universidade Federal da Bahia - GAUDI/DCC
[3] Universidade de Brasília - COMNET/CIC

Resumo

A corrente de blocos (blockchain) é uma tecnologia disruptiva que deve revolucionar o nosso modo de viver, trabalhar e negociar. A corrente de blocos é considerada a tecnologia que vai revolucionar a Internet, provendo uma camada de confiança distribuída. Assim como a Internet permite hoje a transferência de arquivos, a tecnologia de corrente de blocos permitirá a Internet de Valores, na qual é possível a transferência sem intermediários de ativos, tais como dinheiro, ações, propriedade intelectual, votos, etc. A corrente de blocos em sua essência é uma simples estrutura de dados imutável que armazena registros de transações e que é replicada em todos os participantes da rede. Os algoritmos de consenso que determinam a forma como se tomam as decisões coletivas e como são obtidas as finalizações das tarefas são a parte fundamental e mais complexa de corrente de blocos. Existem dezenas de algoritmos de consenso com diferentes características, dentre elas gasto energético, desempenho, tolerância a falha de equipamentos, robustez a ataques de conluio, escalabilidade, etc. Assim, este capítulo foca em algoritmos de consenso para ajudar o projetista de corrente de blocos a escolher o consenso apropriado para suas aplicações. Diversos algoritmos de consenso são apresentados especificando suas características, suas vantagens, suas desvantagens e finalidades. Outro foco deste minicurso é uma parte prática que mostra a construção de uma aplicação de corrente de blocos usando a plataforma Hypeledger Fabric, que é uma plataforma de corrente de blocos privada, modular e de código aberto, além de ser a mais utilizada pelas empresas. A aplicação provê garantia de análise forense em um ambiente de funções virtualizadas de rede.

Este trabalho foi realizado com recursos do CNPq, CAPES, FAPERJ e FAPESP (2015/24514-9, 2015/24485-9 e 2014/50937-1).

3.1. Introdução

As cidades inteligentes buscam o uso de novas tecnologias para prover facilidades que melhorem a qualidade de vida do cidadão. No entanto, há várias brechas de segurança em relação a privacidade, integridade e confidencialidade dos dados que precisam ser tratadas. Um exemplo são provedores de serviços centralizados, como Facebook e Google, que proveem pouca informação aos cidadãos sobre o uso, armazenamento e análise de dados pessoais [Xie et al. 2019]. Um arcabouço usando corrente de blocos é uma potencial solução para prover uma sistema de comunicação seguro [Biswas and Muthukkumarasamy 2016, Bano et al. 2017].

As tecnologias de corrente de blocos de primeira geração, baseadas no Bitcoin [Nakamoto 2008], e de segunda geração, baseadas nos contratos inteligentes do Ethereum [Wood 2014] já revolucionaram o mundo atual ao criar uma camada de confiança para transferências seguras de ativos e execução segura de contratos. No entanto, para a Internet do Futuro os desafios de escalabilidade são ainda maiores, pois prevê-se que em 2025 o uso de dispositivos de Internet das coisas (*Internet of Things* - IoT) gere até 3 trilhões de dólares em valor de mercado [Machina Research 2016] e atinja a marca de 75 bilhões de dispositivos conectados [Statista 2018], que atenderão a diversas aplicações, desde redes veiculares tolerantes a atraso até serviços críticos como saúde eletrônica (*e-Health*) [Zhang et al. 2018, Esposito et al. 2018], cidades inteligentes (*smart cities*) [Xie et al. 2019, Rahman et al. 2019, Mora et al. 2018] e redes elétricas inteligentes (*smart grids*) [Pieroni et al. 2018].

Este capítulo procura focar em algoritmos de consenso e se baseia em trabalhos anteriores dos autores tais como artigos publicados em eventos nacionais e internacionais e, principalmente, os minicursos apresentados no Simpósio Brasileiro de Redes de Computadores "Segurança na Internet do Futuro: Provendo Confiança Distribuída através de Correntes de Blocos na Virtualização de Funções de Rede" [Rebello et al. 2019b] e "Blockchain e a Revolução do Consenso sob Demanda" [Greve et al. 2018].

Além disso, este capítulo objetiva motivar o leitor quanto à importância da tecnologia de corrente de blocos para fornecer segurança às telecomunicações e aos ambientes distribuídos. Assim, esse capítulo dedica uma parte significativa à realização de uma atividade prática que mostra o desenvolvimento de duas correntes de blocos baseadas que executam contratos inteligentes na plataforma Hyperledger Fabric. As correntes de blocos mitigam ataques em uma arquitetura de fatiamento de redes que provê o encadeamento de funções de rede para construir serviços fim-a-fim sob demanda. Os participantes poderão criar, verificar e discutir o funcionamento de cada componente de uma corrente de blocos baseada no Hyperledger Fabric e aprender a controlar as atividades através de contratos inteligentes.

O capítulo é organizado em seis partes principais: i) os fundamentos da tecnologia de corrente de blocos; ii) o consenso em corrente de blocos iii) requisitos básicos de consenso e iv) consensos baseados em prova e tolerantes a falhas de paradas e bizantinas, v) a elaboração prática de uma aplicação que se serve de uma corrente de blocos programada no Hyperledger Fabric [2] e vi) as perspectivas futuras e problemas em aberto.

[2]Disponível em https://github.com/hyperledger/fabric

3.2. A Tecnologia de Corrente de Blocos[3]

Esta seção aborda a tecnologia de corrente de blocos, iniciando com o problema do gasto duplo resolvido em 2008 por Satoshi Nakamoto com a proposta da moeda virtual Bitcoin [Nakamoto 2008]. Aborda-se também propriedades e categorias de correntes de bloco, modelos de rede, assim como tipos de consenso e classes de protocolos de consenso.

3.2.1. As Moedas Digitais e o Problema do Gasto Duplo

Durante a maior parte do século XX, engenheiros, desenvolvedores, criptógrafos e profissionais de tecnologia da informação procuraram resolver o problema de transferência de ativos para permitir a criação de uma moeda digital descentralizada. A transferência de arquivos na Internet é realizada facilmente copiando-se o arquivo original e enviando-o para o destino. No entanto, a transferência de ativos (dinheiro, ações etc.) é bem mais complexa e requer um intermediário para prover confiança porque ao transferir um ativo da origem para o destino deve-se garantir a subtração do valor do ativo a ser transferido da origem e garantir que o valor do ativo a ser transferido será acrescentado no destino. A dificuldade reside no problema do gasto duplo (*double spending problem*), que ocorre quando um mesmo ativo é utilizado mais de uma vez em diferentes transações de um sistema, pois os ativos digitais podem ser facilmente replicados. Ainda que comumente postulado para o caso de moedas digitais, o problema estende-se a qualquer tipo de recurso digital único, como endereços IP, domínios, registros de identidade, propriedade intelectual, recursos computacionais, serviços, etc. A Figura 3.1 ilustra o problema do gasto duplo. No exemplo, suponha que Alice deseja transferir moedas para Bob. Se Alice e Bob utilizam dinheiro em espécie, Alice deve ceder suas moedas a Bob e não possuirá mais as moedas após a transação. Porém, se uma moeda digital for utilizada, é necessária uma maneira de garantir que Alice gastou as moedas, pois moedas digitais são representações em bits que podem ser facilmente duplicados. Neste caso, Alice pode enviar um arquivo digital com o valor desejado a Bob enquanto mantém uma cópia local do arquivo. Se Alice ainda mantiver uma cópia, ela pode enviá-la a uma terceira pessoa, Carol, realizando um gasto duplo.

Tradicionalmente, a prevenção do gasto duplo é realizada através da intermediação centralizada em uma terceira entidade com poderes de autoridade, que utiliza regras de negócio para autorizar as transações e verificar se as moedas envolvidas foram gastas. Essa abordagem é normalmente utilizada em bancos, companhias de crédito, plataformas de vendas em linha (*online*) e, no caso de ativos na Internet, através de organizações como a *Internet Assigned Numbers Authority* (IANA)[4]. Porém, a centralização implica que todos os participantes do sistema possuam total confiança na autoridade central, que pode cobrar taxas, limitar o volume de transações e controlar a rede de forma arbitrária. Além disso, uma falha na autoridade central pode interromper todas as transações do sistema por tempo indeterminado. O modelo centralizado, portanto, cria um ponto único de falha na rede, impactando tanto a disponibilidade quanto a confiança no sistema. No caso de ativos financeiros, os bancos cobram taxas exorbitantes e introduzem enormes

[3]Esta seção é baseada no projeto de fim de curso de Gabriel Antonio Fontes Rebello [Rebello 2019].
[4]Disponível em https://www.iana.org/

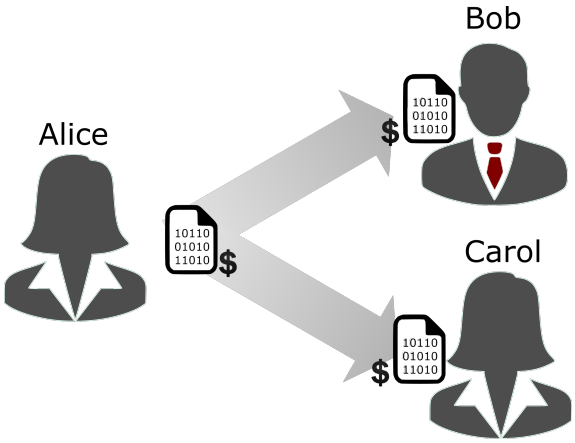


Figura 3.1: Exemplo do problema do gasto duplo no qual a mesma representação digital de uma moeda é replicada e gasta duas vezes.

atrasos para a transferência.

O conceito de moeda digital descentralizada, doravante referida como criptomoeda, é o principal propulsor de aplicações envolvendo trocas de ativos. Durante décadas, procurou-se um mecanismo para viabilizar uma criptomoeda totalmente funcional. Em 1983, David Chaumian introduziu o primeiro protocolo anônimo para criação de criptomoedas utilizando o conceito de assinatura cega (*blind signature*) [Chaum 1983]. A assinatura cega provê privacidade às transferências de moedas, mas depende fortemente de entidades centralizadas para realizar as assinaturas. Em 1998, Wei Dai propôs *b-money*, uma criptomoeda que introduz a ideia de criar valor monetário através da resolução de desafios computacionais e com consenso descentralizado [Dai 1998]. No entanto, a proposta possuía poucos detalhes sobre a implementação do consenso, dificultando sua adoção. Em 2002, Adam Back propôs formalmente *Hashcash*, um algoritmo de prova de trabalho (*Proof of Work* - PoW) baseado em funções resumo (*hash*) criptográficas[5] para prevenção de *spam* de e-mails e ataques de negação de serviço [Back 2002]. Em 2005, Hal Finney desenvolveu o conceito de provas de trabalho reutilizáveis (*Reusable Proof of Work* - RPoW) em um sistema que reúne os fundamentos do *b-money* e os desafios computacionalmente custosos do *Hashcash* [Finney 2005]. No entanto, a proposta também dependia de entidades confiáveis para ser implementada.

Satoshi Nakamoto revolucionou a área de consenso distribuído ao propor em 2008 um consenso probabilístico baseado em prova de trabalho (*Proof of Work* - PoW) [Nakamoto 2008]. O artigo "Bitcoin: A peer-to-peer electronic cash system"foi publicado em uma lista de correio eletrônicos de criptografia. No algoritmo de consenso por prova de trabalho um nó que propõe um bloco, denominado nó minerador[6], deve apresentar uma prova de que pode liderar o consenso gastando recursos computacionais para resolver independentemente um desafio matemático computacionalmente custoso[7]. O Bitcoin é a primeira criptomoeda totalmente descentralizada, combinando a prova de trabalho com

[5]Os fundamentos de funções resumo criptográficas e criptografia de chaves assimétricas são apresentados no Apêndice A.

[6]O nome minerador vem da da dificuldade e do enorme trabalho necessário para vencer o desafio.

[7]O desafio é calcular um *hash* com um certo número de zeros.

uma nova e revolucionária estrutura de dados organizada em blocos de transações: a corrente de blocos (*blockchain*). A proposta de solução do problema do gasto duplo utiliza tecnologias já conhecidas e dominadas como criptografia, consenso, redes de comunicação e incentivos.

O desafio matemático usado como prova de trabalho depende apenas do estado mais recente da corrente de blocos e, portanto, é totalmente paralelizável. A dificuldade do desafio[8] é ajustada periodicamente de acordo com a capacidade de mineração da rede para garantir uma taxa constante de mineração de blocos. Ao resolver o desafio, o minerador vencedor submete o bloco e a prova de trabalho para todos os seus vizinhos. Qualquer minerador pode tentar gerar um novo bloco a qualquer momento. O nó que vence o desafio recebe incentivos em troca do trabalho computacional realizado. O incentivo recebido em resolver o desafio deve ser maior e compensar o gasto computacional realizado para resolver o desafio. Assim, os nós maliciosos tendem a seguir a ordem instituída pelos nós honestos, pois os ganhos em agir honestamente compensam ações maliciosas [Nakamoto 2008]. A probabilidade de um minerador minerar um bloco é, portanto, proporcional à sua capacidade computacional.

O projeto Ethereum, em 2014, idealizado por Vitalik Buterin [Wood 2014, Buterin 2014], usando o consenso com prova de trabalho adota o conceito de contratos inteligentes, que permite a execução de uma máquina de Turing completa. Os contratos inteligentes expressam uma lógica de transações mais sofisticada, possibilitando a implementação de aplicações descentralizadas e autônomas, em diversos níveis e escopos. O Ethereum é considerado uma grande evolução em relação ao Bitcoin, que incorpora uma máquina de estados bem simplificada, com elementos voltados essencialmente para transações com a moeda bitcoin.

A corrente de blocos provê uma camada de confiança e, portanto, permite a transferência de ativos (valores) de uma pessoa para a outra. A Internet do Futuro está sendo chamada Internet de Valor [Truong et al. 2018], pois ativo é um valor como ações, votos, pontos de programa de fidelidade, propriedades intelectuais, músicas, descobertas científicas, entre outras. Com o sucesso da corrente de blocos começam a surgir aplicações em diversas áreas que vão revolucionar o nosso modo de fazer negócios. Na evolução de projeto da corrente de blocos *blockchain*, são então destacadas três fases [Bashir 2018]: a Blockchain 1.0 envolve o lançamento do Bitcoin em 2008, com as primeiras implementações das criptomoedas, e um ecossistema de aplicações e pagamentos com o ativo digital. A Blockchain 2.0 inicia-se com a proposta inovadora dos contratos inteligentes em 2013, e toda gama de aplicações financeiras possíveis. A Blockchain 3.0 caracteriza a adoção da tecnologia corrente de blocos no benefício de aplicações em diversas áreas, para além da financeira: governo, comércio, artes, saúde, cidades digitais, etc.

[8]O número de zeros aumenta se a capacidade de computação aumentar e o tempo para vencer o desafio diminuir. Da mesma forma, se a capacidade de processamento usada para vencer o desafio diminuir, o número de zeros diminui.

3.2.1.1. A Corrente de Blocos

A corrente de blocos, conhecida por suas aplicações em criptomoedas como o Bitcoin [Nakamoto 2008], Ethereum [Wood 2014] e outras tantas, é uma tecnologia disruptiva que deve revolucionar não somente a tecnologia da informação como também a economia e a sociedade, permitindo uma maior descentralização do mundo atual. O principal diferencial de uma corrente de blocos é ser uma estrutura de dados distribuída que garante confiabilidade sem necessitar de uma autoridade central comum a todas as entidades. Pela primeira vez, duas partes que não confiam uma na outra ou mesmo que não se conhecem podem trocar ativos sem um intermediário. A corrente de blocos substitui o modelo centralizado por um modelo colaborativo no qual a maior parte da rede deve concordar sobre todas as decisões. A corrente de blocos ao permitir decisões coletivas, eliminando a entidade centralizadora, potencialmente torna desnecessário governos, bancos, agências de crédito etc. para transferência de ativos, fornecendo poder à população. A corrente de blocos é aplicável em todo ambiente distribuído no qual os participantes não possuem confiança mútua e também não confiam ou são incapazes de entrar em acordo sobre uma autoridade centralizada que governe todos os procedimentos sensíveis da rede [Wüst and Gervais 2018]. Suas propriedades, discutidas com maior profundidade na Seção 3.2.1.3, garantem a auditabilidade, a imutabilidade e o não repúdio de todas as transações realizadas por participantes da rede, e são chave para a criação de um ambiente seguro e escalável.

A corrente de blocos é uma tecnologia de livro-razão distribuído (*Distributed Ledger Technology* - DLT) que utiliza máquinas independentes, denominadas de nós mineradores[9], para gravar, compartilhar e sincronizar transações em um sistema de controle descentralizado sobre uma rede par a par (*peer-to-peer* - P2P). Cada nó minerador possui uma cópia local da corrente de blocos na qual pode verificar qualquer transação emitida na rede desde sua criação. A adição de blocos é realizada de maneira descentralizada através de um protocolo de consenso comum aos nós mineradores. Devido à natureza descentralizada e à existência de cópias da corrente de blocos em todos os mineradores, um atacante, ou grupo de atacantes em conluio, que deseja realizar um gasto duplo precisa controlar uma parcela grande o suficiente da rede para propor um bloco que oculte uma de suas transações e seja aceito pelo protocolo de consenso [Eyal and Sirer 2018, Nakamoto 2008]. O protocolo de consenso garante que as cópias das correntes de blocos são a mesma em qualquer nó minerador e, portanto, a propriedade de não repúdio de transações é garantida. Todas as transações são assinadas por seus emissores através de criptografia de chaves assimétricas e, na maior parte das implementações, utiliza-se a chave pública como identificador do nó na rede.

A Figura 3.2 ilustra o funcionamento de uma corrente de blocos como um livro-razão distribuído através de um protocolo de consenso genérico. Um nó minerador da rede que deseja inserir um novo bloco, alterando o estado atual S da corrente de bloco, inicia uma rodada de consenso reunindo as transações válidas recebidas de usuários em um novo bloco a ser proposto. A seguir, o bloco proposto é validado localmente de acordo com políticas pré-definidas e é difundido na rede. Dependendo do protocolo de consenso

[9]Os mineradores usualmente recebem uma recompensa para executarem a mineração e, em modelos de consenso sem recompensa, são chamados de participantes do consenso ou simplesmente de nós.

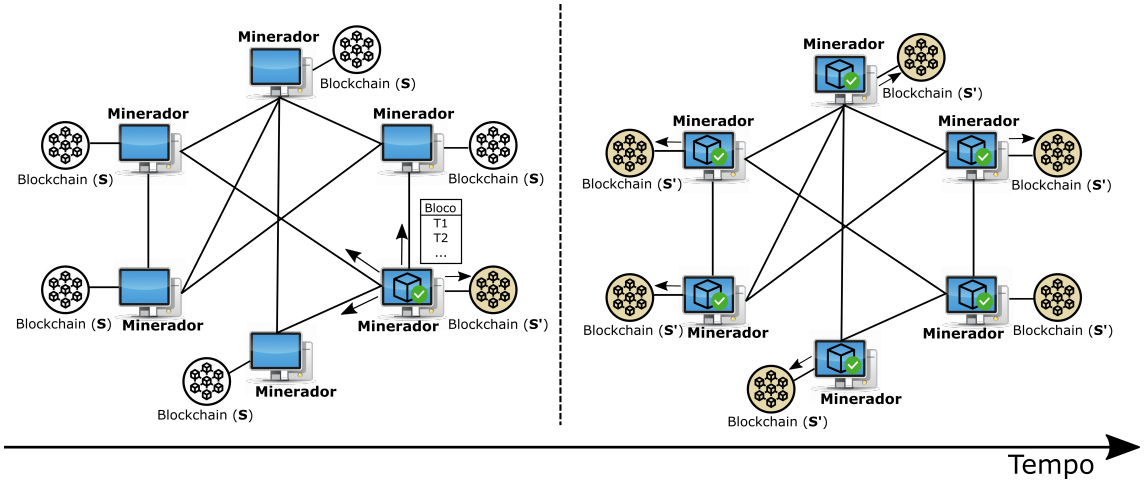


Figura 3.2: Atualização de uma corrente de blocos (*blockchain*) através de um protocolo de consenso genérico. Um nó minerador propõe um novo bloco em difusão e os demais nós mineradores verificam e adicionam independentemente o bloco proposto à corrente de blocos, incrementando o estado global S de maneira consistente.

adotado, o bloco proposto pode ser adicionado imediatamente, de forma assíncrona, à corrente de blocos do nó minerador que propôs o bloco, ou de forma síncrona em todos os nós mineradores ao fim da rodada. A figura mostra o caso de um protocolo no qual o estado S é alterado para S' no nó minerador que propôs o bloco antes da difusão. Ao

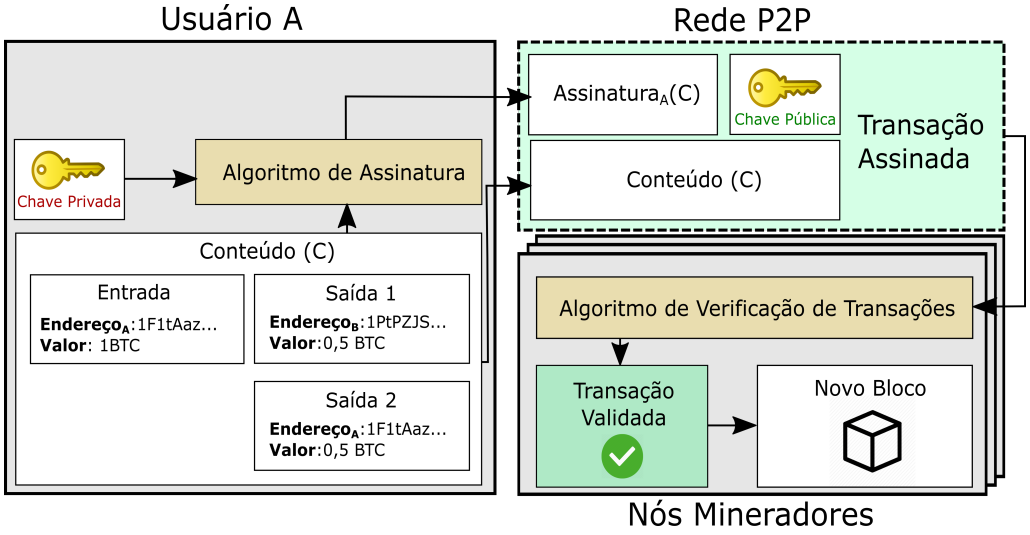


Figura 3.3: Etapas de uma transação em uma corrente de blocos. Para transferir 0,5 BTC ao usuário B, o usuário A cria uma transação contendo: i) uma entrada com seu endereço e o total de BTC que possui; ii) uma saída com o endereço de Bob e o valor que deseja transferir e iii) uma saída com seu próprio endereço e o valor que irá possuir após a transferência do valor desejado. A seguir, o usuário A utiliza um algoritmo criptográfico para assinar a transação e a envia com sua chave pública em difusão para a rede par a par (*peer-to-peer* - P2P) na qual estão os nós mineradores.

receber o bloco proposto, cada nó minerador valida o bloco independentemente e, caso aprove o bloco, o adiciona à corrente de blocos e chega ao estado S'. O algoritmo de validação de um bloco e de cada transação deve ser acordado previamente por todos os nós mineradores. Todo protocolo de consenso possui um mecanismo de atualização para obter o estado mais recente e corrigir discrepâncias de estado resultantes de blocos não aprovados, garantindo a consistência da corrente de blocos em toda a rede. Os protocolos de consenso e suas premissas são melhor discutidos na Seção 3.3.

A Figura 3.3 mostra as etapas de uma transação em um sistema de troca de ativos digitais através de uma transação simplificada de Bitcoin (BTC) [Nakamoto 2008]. Para enviar ou receber ativos, um usuário deve utilizar um par de chaves assimétricas para assinar transações. Após a difusão da transação assinada, qualquer participante do consenso pode aplicar o algoritmo de validação da rede para verificar criptograficamente a autenticidade da transação e se a transação conflita com outra já registrada na corrente de blocos. Transações inválidas são descartadas imediatamente. Caso a validação ocorra com sucesso, o participante do consenso adiciona a transação a um novo bloco que é proposto na próxima rodada do protocolo de consenso. A maior parte dos sistemas de troca de ativos provê programas que gerenciam chaves, armazenam endereços e emitem transações assinadas de forma transparente ao usuário. Estes programas são amplamente conhecidos como "carteiras".

3.2.1.2. As Estruturas de Dados da Corrente de Blocos

A estrutura de dados da corrente de blocos proposta por Nakamoto como parte da criptomoeda Bitcoin é uma lista encadeada de estruturas de dados menores, conhecidas como blocos. Cada bloco está conectado a seu antecessor através de uma função resumo (*hash*) criptográfica, criando uma corrente de blocos conforme a mostra a Figura 3.4. A corrente de blocos funciona como um livro-razão (*ledger*), ou registro permanente (*log*), de blocos ordenados no tempo. Cada bloco na corrente possui um cabeçalho contendo o valor resultante de uma função resumo (*hash*) criptográfica do bloco antecessor, e uma seção de conteúdo que armazena dados relevantes a uma aplicação. A única exceção a esta regra é o bloco inicial, o gênesis, que por definição não possui bloco anterior. A utilização de uma sequência de funções resumo criptográficas, cujas saídas diferem drasticamente após a alteração de qualquer bit na entrada[10], implica que todos os blocos incluídos a partir de uma possível alteração devem ser reconstruídos. Assim, a adição de blocos torna cada vez mais custoso alterar a corrente e, como a corrente de blocos é replicada em cada nó minerador da rede, um atacante deve invadir todos os mineradores e recriar cada corrente de blocos para modificar uma transação passada. A replicação da corrente de blocos em todos os nós mineradores e o encadeamento através de funções *hash* criptográficas garante, portanto, a imutabilidade de um bloco com muitos sucessores.

As principais características da corrente de blocos são descritas em [Greve et al. 2018, Mello et al. 2017, Chicarino et al. 2017, Mattos et al. 2018]. A literatura de corrente de blocos apresenta propostas específicas que utilizam a seção de conteúdo do bloco

[10]Os fundamentos de funções resumo criptográficas e criptografia de chaves assimétricas são apresentados no Apêndice A.

para diferentes propósitos. Nakamoto propõe originalmente registrar transações bancárias para criar uma criptomoeda [Nakamoto 2008]. Wood *et al.*, Frantz *et al.* e Androulaki *et al.* propõem utilizar correntes de blocos para armazenar e executar contratos inteligentes através de uma máquina virtual distribuída [Wood 2014, Frantz and Nowostawski 2016, Androulaki et al. 2018]. Wilkinson *et al.* propõem um sistema baseado em correntes de blocos para prover armazenamento descentralizado em nuvem com privacidade [Wilkinson et al. 2014]. Ali *et al.* propõem o uso de correntes de blocos para registrar nomes de domínio [Ali et al. 2016]. Azaria *et al.* propõem registrar informações de fichas médicas [Azaria et al. 2016]. Lee *et al.* propõem registrar votos para criar um sistema de votação descentralizado [Lee et al. 2016]. Christidis e Hun *et al.* propõem rastrear dispositivos de Internet das Coisas (*Internet of Things* - IoT) através do armazenamento de informações na corrente de blocos [Christidis and Devetsikiotis 2016, Huh et al. 2017]. Bozic *et al.* e Alvarenga *et al.* propõem registrar informações de máquinas virtuais e funções virtualizadas de rede na corrente de blocos [Bozic et al. 2017, Alvarenga et al. 2018].

O principal elemento de um sistema baseado em corrente de blocos são as transações. Uma transação representa uma ação atômica a ser armazenada na corrente de blocos que transfere um ativo entre duas entidades. A transação possui quatro componentes principais: i) um remetente, que possui um ativo que deseja transferir e que emite a transação; ii) um destinatário que receberá o ativo que se deseja transferir; iii) o ativo que será transferido e iv) uma assinatura do remetente da função resumo de toda a transação, que corresponde a uma função *hash* de todos os campos anteriores pelo remetente. Quando a imutabilidade da corrente de blocos é utilizada com transações assinadas, cria-se um sistema que funciona como um livro-razão distribuído. As transações no sistema são ordenadas dentro de cada bloco e podem ser verificadas por qualquer nó participante da rede, desde o bloco inicial da corrente de blocos, denominado bloco gênesis. Pela propriedade de chaves criptográficas assimétricas, a assinatura do *hash* da transação, usando a chave privada do remetente provê autenticidade, não repúdio e integridade à transação, fornecendo maior segurança à rede. Ainda, é possível obter pseudo-anonimidade utilizando chaves públicas ou endereços únicos, que não revelam as identidades pessoais, para identificar remetentes e destinatários. A privacidade de transações, desejada em casos onde apenas o emissor e destinatário devem poder consultar a transação [Androulaki et al. 2018], pode ser garantida criptografando a transação com a chave pública do destinatário.

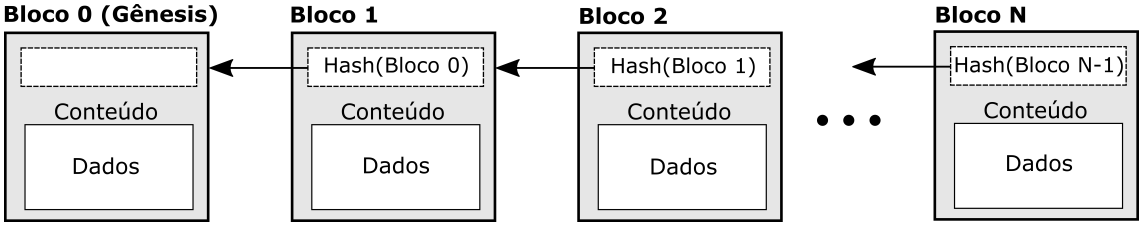


Figura 3.4: Estrutura de uma corrente de blocos na qual cada bloco é associado ao bloco seguinte e a integridade das transações de um bloco é garantida por uma uma função resumo (*hash*) criptográfica.

3.2.1.3. Propriedades de Corrente de Blocos

A corrente de blocos possui propriedades que proveem benefícios às aplicações e sistemas baseados nesta tecnologia. As principais propriedades da corrente de blocos são [Zheng et al. 2018, Xu et al. 2017, Wüst and Gervais 2018]:

- **Descentralização**. As correntes de blocos executam de maneira distribuída, através do estabelecimento de consenso entre todos os participantes da rede. Não há uma entidade centralizadora;
- **Desintermediação**. A corrente de blocos elimina a necessidade de um intermediário confiável para a troca de ativos. Os ativos podem ser trocados diretamente pela rede e a confiança é estabelecida através de consenso;
- **Imutabilidade**. Os dados armazenados em uma corrente de blocos são imutáveis. Não é possível modificar ou recriar qualquer dado incluído na corrente de blocos de forma retroativa. Toda atualização na corrente de blocos é realizada de forma incremental;
- **Irrefutabilidade**. Os dados são armazenados na corrente de blocos em forma de transações assinadas, que não podem ser alteradas devido à propriedade de imutabilidade da corrente de blocos. Portanto, o emissor de uma transação jamais pode negar sua existência;
- **Transparência**. Todos os dados armazenados na corrente de bloco são acessíveis por todos os participante da rede. Em correntes de blocos públicas, como o Bitcoin e o Ethereum, as transações são abertas para qualquer usuário com acesso à Internet;
- **Auditabilidade**. Como consequência da transparência, todo participante pode verificar, auditar e rastrear os dados inseridos na corrente de blocos para encontrar possíveis erros ou comportamentos maliciosos. No caso de correntes de blocos federadas, o processo de auditagem pode responsabilizar um malfeitor na rede;
- **Disponibilidade**. As correntes de blocos são estruturas replicadas em cada participante da rede e, portanto, a disponibilidade do sistema é garantida mesmo sob falhas, devido à redundância de informações;
- **Anonimidade**. Os usuários e nós mineradores de uma corrente de blocos são identificados por chaves públicas ou identificadores únicos que preservam suas identidades. Ainda, é possível utilizar uma chave pública em cada transação, evitando a rastreabilidade do usuário e conferindo um grau a mais de anonimidade.

Além das propriedades citadas, a corrente de blocos também pode prover privacidade, ao custo da transparência e auditabilidade. Em algumas implementações de corrente de blocos, os dados inseridos na corrente de blocos são criptografados com uma chave pública, evitando que todos os participantes possam ver o conteúdo da transação [Smolensk 2018, Androulaki et al. 2018]. Nesses casos, a transparência e a auditabilidade limitam-se a um ou mais participantes que detêm a chave privada correspondente e podem descriptografar a transação criptografada.

3.3. O Consenso em Correntes de Blocos

Um protocolo de consenso é o principal componente utilizado para manter a consistência de uma corrente de blocos. No modelo de corrente de blocos públicas usando prova de trabalho (*Proof-of-Work* – PoW), qualquer nó pode participar do protocolo de consenso e procurar acrescentar blocos, estendendo a corrente de blocos. Para isso, geralmente é necessário resolver um enigma (desafio) criptográfico, que é chamado de mineração pela dificuldade computacional do processo. O procedimento de mineração é usado para provocar um custo, que na prova de trabalho é um custo computacional que tem o grande inconveniente de consumir muita energia. Portanto, estes protocolos escalam no número de participantes na rede, mas apresentam um desempenho muito baixo em número de transações por segundo, pois múltiplas propostas de inserção de novos blocos podem ser feitas concorrentemente e gerar bifurcações (*forks*) na corrente, sendo necessário aplicar uma regra onde o ramo da bifurcação mais longo ganha. A possibilidade de bifurcações e regras para decidir qual dos ramos deve continuar são feitas de forma probabilística, resultando em um consenso probabilístico.

Diferentemente do modelo de consenso probabilístico, existe o modelo consenso determinístico que é tolerantes a falhas de parada (*crash*) ou bizantinas. No modelo de corrente de blocos privadas com tolerância a falhas, apenas nós autorizados, denominados validadores, executam um protocolo de consenso para acrescentar blocos e estender a corrente de blocos, que apresenta a mesma evolução em todos os participantes do consenso, pois não ocorrem bifurcações (*forks*). Apesar de apresentar um bom desempenho, o modelo de consenso tolerante a falhas não escala no número de participantes na rede[11].

Nota-se que a convergência e funcionamento correto de um sistema baseado em corrente de blocos depende do consenso entre todos os nós participantes, que devem adotar um protocolo comum para tal. Esta seção apresenta a fundamentação teórica necessária para a compreensão de protocolos de consenso em sistemas distribuídos, que é o principal desafio de implementação de correntes de blocos. A escolha do protocolo de consenso deve levar em consideração características arquiteturais da rede de comunicação que interconecta os participantes, requisitos de desempenho como a vazão em transações por segundo, a escalabilidade em número de nós, a tolerância a possíveis falhas e ataques, dentre outros. Esta seção procura fazer uma análise destas características.

3.3.1. O Sistema de Comunicação da Corrente de Blocos

Um sistema distribuído envolve um conjunto de diferentes processos[12], por exemplo computadores participantes, que trocam mensagens uns com os outros e se coordenam para obter um objetivo comum. Portanto, todo sistema distribuído se baseia em um sistema de comunicação que interconeta os participantes e que é usado pelos protocolos de consenso. Os protocolos de consenso assumem diversas hipóteses sobre o sistema de comunicação. Praticamente todos os protocolos de consenso assumem um sistema

[11]Deve ser ressaltado que o modelo de consenso por prova de trabalho (PoW) escala, mas não apresenta uma alta vazão de transações. Por outro lado, o consenso tolerante a falhas apresenta alta vazão, mas não escala, existem propostas de modelos híbridos, usando o PoW para definir comitês de participantes que executam o consenso tolerante a falhas bizantinas tradicional.

[12]Este capítulo usa os termos nós (*node*), par (*peer*), participante do consenso, computador ou componente como sinônimos de processo (*process*).

de comunicação par-a-par (*peer-to-peer*) completamente conectado, em que cada par de participantes consegue se comunicar diretamente através de canais bidirecionais. Além disso, várias outras suposições são realizadas para definir o nível de sincronia fornecido pelo sistema de comunicação, o tipo de falha que o sistema consegue suportar, e a disponibilidade de primitivas criptográficas.

Modelos de Sincronia do Sistema de Comunicação - Os modelos de sincronia definem os aspectos relacionados com o tempo no sistema relativos ao processamento nos participantes e para entrega correta de mensagem pelo sistema de comunicação. Existem vários modelos de sincronia [Attiya and Welch 2004, Lynch 1996], a seguir são descritos os três mais importantes e práticos:

- Assíncrono [Fischer et al. 1985, Lynch 1996] - Este é o modelo mais fraco de sincronia para um sistema distribuído. Neste modelo, não existe um limite para o tempo de processamento local em um participante (processo) e para a entrega/comunicação de mensagens. Consequentemente, neste modelo não existe a noção de tempo e seria o modelo ideal de ser considerado em sistemas de comunicação. No entanto, a impossibilidade de FLP[13] (*FLP Impossibility*) afirma que não possível se atingir consenso em sistemas de comunicação assíncronos quando pelo menos um participante pode sofrer falha de parada;

- Síncrono [Lynch 1996] - Este é o modelo mais forte de sincronia, no qual existe um limite de tempo conhecido para os processamentos locais nos participantes e também para a comunicação/entrega das mensagens. Este modelo de sistema de comunicação é praticamente irreal de se obter;

- Parcialmente ou Eventualmente Síncrono [Dwork et al. 1988]. A ideia por trás destes modelos é de que o sistema trabalha de forma assíncrona (não respeitando nenhum limite de tempo) a maior parte do tempo. Porém, durante períodos de estabilidade, o tempo para a entrega entrega (*delivery*) de mensagens é limitado. Deve ser ressaltado que este modelo engloba o comportamento de redes de melhor esforço, como a Internet, que passam por tempos de estabilidade e tempos de perturbação. Este é modelo de sincronia mais fraco no qual o consenso possui solução determinística [Fischer et al. 1985]. Este é um modelo intermediário mais realístico, que geralmente é considerado na implementação de aplicações distribuídas tais como protocolos de consenso.

Modelos de Falhas do Participante do Consenso - Normalmente dois tipos de falhas são consideradas:

- Falhas por Parada ou Desastrosas (*crash*) - Neste modelo, um nó participante em falha não responde e não executa novas operações durante a execução do sistema.

[13]FLP vem de F-Fisher, L-Linch e P-Paterson.

- Falhas Bizantinas - Neste modelo, um nó participante em falha pode exibir um comportamento arbitrário, desviando do protocolo especificado e executando qualquer ação. A falha bizantina é muito mais complexa de se tolerar que a falha desastrosa, uma vez que um participante pode ser um agente malicioso que responde uma coisa para um participante e responder o contrário para outro participante.

Modelo de Mensagens com Autenticação Criptográfica – No modelo de falhas bizantinas, os protocolos geralmente necessitam de mensagens autenticadas, que podem ser implementados através de criptografia assimétrica para gerar assinatura digitais para as mensagens, que requer um sistema de infraestrutura de chaves públicas, ou através de criptografia simétrica e uso de código de autenticação de mensagens (*Message Authentication Codes* –MAC), que requer o compartilhamento de segredos [Bishop 2002].

3.3.2. Consenso em Corrente de blocos

Consenso é o processo pelo qual a partir de um grupo de participantes independentes, todos os participantes corretos atingem a mesma decisão coletiva de aceitar o novo bloco a ser inserido na corrente de blocos. A proposta de um novo bloco foi feita previamente por um dos participantes do consenso. Formalmente, este problema é definido em termos de duas primitivas [Hadzilacos and Toueg 1994]:

- *propose(P,b)*: o novo bloco, *b*, é proposto ao conjunto de participantes do consenso *P*;
- *decide(b)*: executado pelo protocolo de consenso para notificar ao(s) interessado(s), geralmente alguma aplicação, que *b* é o novo bloco a ser acrescido na corrente de blocos.

Para essas primitivas satisfazerem a correção (*safety*) e a vivacidade (*liveness*), elas devem verificar as seguintes propriedades [Aspnes 2003, Attiya and Welch 2004]:

- **Acordo**: Se um processo correto decide *v*, então todos os processos corretos terminam por decidir *v*.
- **Validade**: Um processo correto decide *v* somente se *v* foi previamente proposto por algum processo.
- **Terminação**: Todos os processos corretos terminam por decidir.

A propriedade de acordo garante que todos os processos corretos decidem o mesmo valor. A validade relaciona o valor decidido com os valores propostos e sua alteração dá origem a outros tipos de consensos. As propriedades de acordo e validade definem os requisitos de correção (*safety*) do consenso, já a propriedade de terminação define o requisito de vivacidade (*liveness*) deste problema.

Fisher, Lynch e Paterson (FLP) publicaram em 1985 um dos trabalhos mais importantes envolvendo o problema do consenso [Fischer et al. 1985], conhecido como a

impossibilidade de FLP. Eles provam que não é possível obter-se consenso em um sistema distribuído completamente assíncrono no qual pelo menos um processo pode falhar (qualquer tipo de falha). Em vista disso, a maioria das propostas de consenso encontradas na literatura consideram algum comportamento temporal do sistema, no qual componentes (processos e/ou canais) obedecem a requisitos temporais. Estes requisitos geralmente estão relacionados com temporizadores (*timeouts*) encapsulados em detectores de falhas [Chandra and Toueg 1996] ou são atendidos a partir da ocorrência de possíveis comportamentos síncronos do sistema subjacente.

3.3.3. O Teorema CAP

Os principais desafios de projeto de corrente de blocos com alta disponibilidade são a tolerância a falhas e a coordenação entre os nós mineradores. O teorema CAP (*Consistency*, *Availability*, *Partition*) prova que todo sistema distribuído apresenta um compromisso entre três propriedades [Brewer 2000]:

- **Consistência (*consistency*)**: todos os nós do sistema distribuído possuem os dados mais atuais da rede;
- **Disponibilidade (*availability*)**: o sistema está operante, acessível e é capaz de responder corretamente a novas requisições;
- **Tolerância à partição (*partition tolerance*)**: o sistema distribuído opera corretamente mesmo que um grupo de nós falhe. Se for possível haver comportamento malicioso dos nós, o sistema continua funcionando corretamente com alguns nós honestos, mesmo na presença de um grupo de nós maliciosos.

O teorema prova que é impossível para um sistema distribuído atingir plenamente todas as três propriedades [Brewer 2000]: consistência, disponibilidade e telerância a partição. Por isto, na prática, sistemas distribuídos privilegiam uma ou duas propriedades, sacrificando as demais. A replicação contribui para a tolerância a falhas. A consistência é obtida com o uso de algoritmos de consenso que garantem que todos os nós mineradores possuem os mesmos dados. No Bitcoin e em diversas outras aplicações [Nakamoto 2008, Wood 2014, Ali et al. 2016], a consistência é sacrificada em nome da disponibilidade e da tolerância a partições. Isto significa que a consistência não é obtida simultaneamente com as outras duas propriedades, mas sim gradativamente, com o tempo, à medida que os blocos são validados pelos nós da rede. Correntes de blocos baseadas em protocolos tolerantes a falhas bizantinas privilegiam fortemente a consistência, em detrimento da disponibilidade [Schwartz et al. 2014, Androulaki et al. 2018, Buchman 2016, Miller et al. 2016, Sousa et al. 2018].

3.3.4. Consenso Probabilístico: Protocolos Baseados em Provas

Apesar da inovação, a prova de trabalho do Bitcoin e do Ethereum apresentam limitações evidentes de desempenho. O Bitcoin possui uma latência de consenso de aproximadamente dez minutos, e uma vazão de transações, de apenas 7 transações por segundo. O Ethereum possui uma latência de 15 segundos e uma vazão de 15 transações por segundo. Este baixo desempenho consiste em um desafio para atender às necessidades dos sistemas atuais [Popov 2017, Eyal et al. 2016].

Outra importante limitação do consenso por prova de trabalho é o excessivo gasto energético de 57,7 TWh por ano[14]. Para se ter uma ideia, esse valor é três vezes maior que a energia gerada pelas usinas nucleares de Angra. O gasto energético do consenso por prova de trabalho é ecologicamente inaceitável. Outros algoritmos baseados em prova procuram mitigar o excesso de gasto energético e as limitações de desempenho presentes na prova de trabalho [Vasin 2014, Schwartz et al. 2014, Kiayias et al. 2017]. Uma breve explicação dos algoritmos mais conhecidos é mostrada a seguir:

- Prova de Posse[15] (*Proof of Stake* - PoS) - Em vez de gastar recursos computacionais, um nó minerador deve apostar parte de seus ativos para receber uma chance de minerar um bloco. Sua chance é proporcional à quantia de ativos apostados. Nos últimos anos, a prova de participação têm se destacado devido ao desejo do Ethereum de abandonar a prova de trabalho para implementar PoS [Tikhomirov 2018];

- Prova de Posse Delegada (*Delegated Proof of Stake* - DPoS) - Os nós mineradores utilizam seus ativos para eleger delegados em um quórum que define o bloco a ser adicionado. A quantidade de votos de um minerador é proporcional aos seus ativos [Kiayias et al. 2017];

- Prova de Queima ou Destruição (*Proof of Burn* - PoB) - Como o PoS, porém os ativos são imediatamente destruídos [Paul et al. 2014];

- Prova de Autoridade (*Proof of Authority* - PoA) - Similar ao DPoS, porém o conjunto de delegados (autoridades) é pré-determinado em acordo e suas identidades são públicas e verificáveis por qualquer participante da rede [Angelis et al. 2018];

- Prova de Capacidade de Armazenamento (*Proof of Capacity* - PoC) - A probabilidade de propor um bloco é proporcional ao espaço de armazenamento cedido à rede por um nó minerador. Quanto maior a capacidade de armazenamento em disco, maior o domínio sobre o consenso [Zheng et al. 2018];

- Prova de Tempo Decorrido (*Proof of Elapsed Time* - PoET) - Cada nó minerador recebe um temporizador aleatório decrescente e o nó cujo temporizador terminar primeiro propõe o próximo bloco [Olson et al. 2018]. Este protocolo de consenso funciona exclusivamente em *hardware* que suportam a tecnologia *Intel software guard extensions* (SGX). O Intel SGX garante a distribuição aleatória de temporizadores e que nenhuma entidade tem acesso a mais de um nó minerador.

Um exemplo de protocolo de prova de posse é o Ouroboros que é detalhado a seguir.

O Protocolo Ouroboros [Kiayias et al. 2017] baseado em prova de posse determina um comitê com partes interessadas (*stakeholders*) aleatoriamente selecionadas para a eleição de um líder, que propõe o próximo bloco da corrente. A probabilidade de um

[14]https://digiconomist.net/bitcoin-energy-consumptio

[15]*Proof of Stake* é também traduzido por prova de participação ou prova de montante.

nó ser selecionado como líder ou de participar do comitê para eleição de líder é diretamente proporcional à participação do nó na rede. O Ouroboros é utilizado pela plataforma Cardano[16], responsável por executar a corrente de blocos da criptomoeda ADA.

O protocolo determina dois tipos de intervalo de tempo: períodos (*slots*) e épocas (*epochs*). A Figura 3.5 ilustra a divisão do tempo no protocolo Ouroboros. Os períodos são intervalos de tempo curtos dentro de uma época em que, a cada período, um líder previamente eleito propõe o bloco a ser adicionado à corrente. As épocas são intervalos de tempo em que um comitê formado por partes interessadas aleatoriamente selecionadas executam uma rodada de consenso. A saída da execução define os participantes do comitê da próxima época e os líderes de cada período da próxima época.

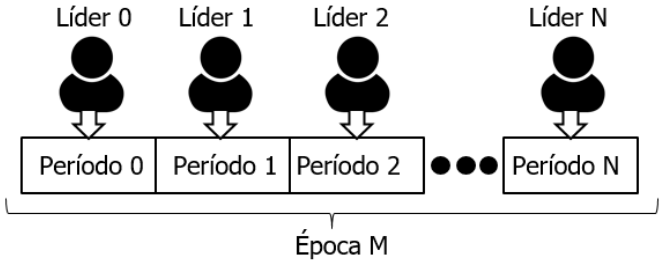


Figura 3.5: Divisão do tempo no protocolo Ouroboros.

O procedimento para a eleição do líder é feito através de uma computação segura multipartidária (*Secure Multi-party Computation* - MPC). A computação segura multipartidária permite que múltiplos participantes com entradas individuais obtenham a mesma saída como resultado [Rabin and Ben-Or 1989]. O processo de eleição é separada em três fases: efetivação (*commitment*), revelação (*reveal*) e recuperação (*recovery*). A Figura 3.6 ilustra as etapas para eleição do líder e escolha dos participantes do consenso no protocolo Ouroboros. Na fase de efetivação, cada participante do comitê gera um valor aleatório e difunde uma mensagem de efetivação. A mensagem de efetivação contém o valor gerado criptografado. Na fase de revelação, cada participante difunde uma mensagem de abertura, revelando o segredo enviado anteriormente. Na fase de recuperação, os participantes formam uma semente (*seed*) com os segredos revelados e usam a semente no algoritmo Siga-o-Satoshi (*Follow-the-Satoshi* - FTS) que seleciona uma moeda da rede para escolher o líder e os participantes do comitê da próxima época.

Durante a execução da fase de revelação, um membro do comitê pode decidir por abandonar o protocolo e não revelar o segredo ao observar os valores recebidos. O Ouroboros usa o esquema de compartilhamento de segredo verificável (*Verifiable Secret Sharing* - VSS), que divide o valor gerado em parcelas. As parcelas são criptografadas e enviadas aos membros do comitê na fase de efetivação, de maneira que os participantes do consenso acumulem mensagens de efetivação dos outros participantes. O esquema permite a recuperação dos segredos não-revelados durante a fase de recuperação, em que os membros honestos postam as parcelas recebidas dos membros que abandonaram o protocolo. Este procedimento permite a reconstrução do segredo e a conclusão do protocolo mesmo que alguns participantes do consenso sejam adversários.

[16]https://www.cardano.org/en/home/

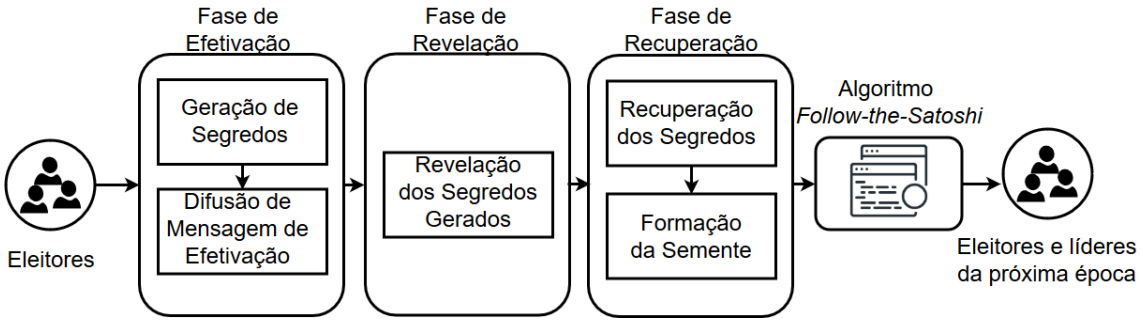


Figura 3.6: Visão geral das fases para formação do comitê responsável por eleger os líderes de períodos no protocolo Ouroboros

3.3.5. Consenso Determinístico: Protocolos Tolerantes à Falhas

Em redes bem-definidas, como redes permissionadas síncronas ou eventualmente síncronas, é possível obter consenso determinístico. Nesse tipo de consenso, todos os nós mineradores devem votar pela aprovação ou rejeição de um bloco e atingir um consenso antes de adicionar o novo bloco à corrente. Como consequência, todos os nós mineradores[17] devem ser conhecidos e identificados. O protocolo de consenso garante que a adição de um bloco à corrente ocorre de forma sincronizada para todas as réplicas, i.e., as réplicas adicionam a mesma sequência de blocos na corrente. Portanto, a consistência do sistema é garantida em qualquer momento e não existem bifurcações (*forks*) na corrente de blocos.

Os protocolos tolerantes a falhas são divididos em tolerantes a falhas de paradas (*Crash Tolerant Fault* - CFT) e tolerantes a falhas bizantinas. A classe de protocolos de consenso para as correntes de blocos tolerantes a falhas bizantinas (*Byzantine Fault Tolerant* - BFT) é particularmente interessante, pois que garantem a consistência da corrente de blocos sob comportamento malicioso. A ideia principal dos protocolos BFT é eleger um líder que inicia e comanda uma rodada de consenso distribuindo o novo bloco proposto a todos os participantes do consenso. Protocolos tolerantes a falhas bizantinas representam um meio termo entre um ambiente totalmente sem confiança e um ambiente totalmente confiável. Os protocolos BFT promovem uma camada de confiança a mais em comparação a protocolos tolerantes apenas a falhas por parada pois toleram comportamento malicioso na rede. No entanto, geralmente necessitam de mais réplicas, quando comparado com os protocolos que toleram falhas apenas por parada, para tolerar a mesma quantidade de falhas.

Existem muitos protocolos de consenso propostos na literatura, a seguir são apresentados um protocolo tolerante a falha de parada e quatro protocolos que foram projetados para tolerar falhas bizantinas (BFT) em um sistema parcialmente síncrono: PBFT [Castro and Liskov 2002] – o primeiro protocolo BFT prático, que utiliza vetores de MAC ao invés de assinaturas digitais para conseguir um desempenho similar aos protocolos que toleram apenas falhas por parada; MinBFT [Veronese et al. 2013] – que utiliza componentes seguros para diminuir tanto a quantidade de réplicas quanto os pas-

[17]É comum usar o termo nós validadores em vez de mineradores uma vez que eles votam no consenso sem executar um desafio ou sem apostar.

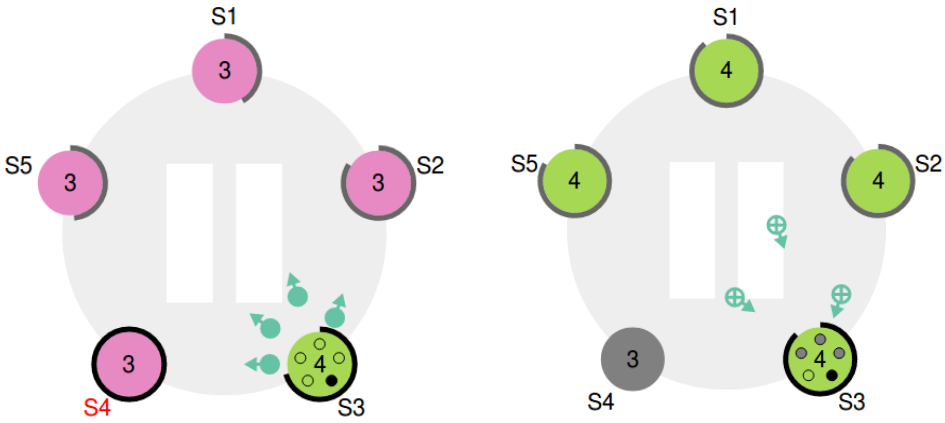
sos de comunicação necessários para finalizar o protocolo; Zyzzyva [Kotla et al. 2009] – que utiliza especulação com o objetivo de melhorar o desempenho; e BFT-CUP [Alchieri et al. 2018] – que foi projetado para ambientes em que os processos são, a priori, desconhecidos.

O Protocolo Raft é um protocolo de consenso tolerante a falhas de parada (*crash*) proposto por Ongaro em 2014 [Ongaro and Ousterhout 2014]. Desenvolvido como uma alternativa de fácil compreensão ao Paxos [Lamport 2001], o Raft possui eficiência e resultados equivalentes ao Paxos, além de providenciar uma fundamentação mais adequada para construção de sistemas práticos. A simplicidade na compreensão é consequência da divisão do consenso em 3 fases: a eleição do líder que ocorre para definir o primeiro líder ou caso o atual falhe; a replicação do registo nas máquinas de estado de todos os nós; e a segurança do registro, garantido que nenhum registro é apagado ou duplicado. O protocolo é baseado nas fortes premissas de que todos os nós são conhecidos, não existe comportamento bizantino na rede e os nós operam em um ambiente assíncrono com relógios defeituosos [Howard 2014]. O Raft utiliza a replicação de registros utilizando a replicação de máquina de estados (RME) [Schneider 1990] que, em conjunto com o protocolo de consenso, garante a existência dos mesmos comandos na mesma ordem em todos as máquinas de estados dos nós. Dessa forma, o Raft garante a integridade dos registros e o funcionamento do consenso com até f participantes em estado de falha de parada dos $n = 2f + 1$ participantes da rede.

O Raft divide o tempo em mandatos de duração indefinida e numerados com inteiros consecutivos. A qualquer momento, cada nó está em um dos três estados: líder, seguidor ou candidato. O líder aceita as entradas do cliente, replica as entradas nos outros nós e informa aos nós quando é seguro aplicar as entradas às máquinas de estado. O seguidor apenas responde a pedidos do líder e do candidato. O candidato é utilizado para eleger um novo líder caso o atual falhe, assim começando um novo mandato. A falha do líder é detectada utilizando o mecanismo de pulsação periódica (*heartbeat*) através de chamadas remotas de procedimento (*Remote Procedure Call* - RPC).

Para detectar falhas, cada nó possui um temporizador de duração escolhida aleatoriamente de um intervalo de 150 a 300 ms para marcar o tempo limite de eleição. Caso o tempo limite de eleição do seguidor seja atingido sem que haja sinal de comunicação do líder pelo mecanismo de pulsação, o seguidor assume que não existe um líder ou que líder está em estado de falha e começa a eleição de um novo líder ao incrementar o mandato atual e se tornar um candidato. Em seguida, o candidato vota em si mesmo e envia pedidos de voto para o restante dos nós por RPC. A fase de eleição está representada na Figura 3.7. O tempo limite de eleição de cada seguidor é redefinido no início de toda eleição. O candidato permanece nesse estado até que consiga $f + 1$ votos, outro candidato se estabeleça como líder ou o tempo limite de eleição seja atingido. A escolha aleatória do tempo limite de eleição garante que a existência de múltiplos candidatos é rara.

Uma vez que o líder é eleito, ele começa a atender as solicitações dos clientes. Cada solicitação de cliente consiste em um comando para a máquina de estados replicada executar. Essas solicitações são guardadas em forma de registros contendo o comando a ser executado, o número do mandato em que o líder recebeu a solicitação e um identificador que define a sua ordem no livro-razão. O líder registra a solicitação no seu



(a) Início da eleição do líder. (b) Término da eleição do líder.

Figura 3.7: A Figura 3.7(a) representa o início da eleição de um novo líder, onde o tempo limite de eleição do seguidor S3 esgotou-se. O seguidor S3 vira um candidato, incrementa o número de mandato de 3 para 4, representado pela cor rosa e verde respectivamente, e envia pedidos de voto aos demais seguidores. Logo em seguida, os seguidores respondem o pedido de voto e incrementam seus números de mandato, assim elegendo o candidato S3 como o novo líder, conforme representado na Figura 3.7(b). Como o antigo líder S4 não respondeu as chamadas do novo líder S3, detectou-se uma falha em S4, representada pela cor cinza.

livro-razão e informa, por RPC em paralelo, os seguidores para replicar o novo registro em seus livros-razão. Após receber $f+1$ respostas positivas dos seguidores, o líder aplica o comando do cliente em sua máquina de estados, envia uma ordem por RPC em paralelo para todos os seguidores aplicarem a entrada em suas máquinas de estados e informarem a saída de suas máquinas de estados para o líder. A partir desse ponto, a solicitação do cliente está efetivada.

O Protocolo Prático de Consenso Tolerante a Falhas Bizantinas - *Practical Byzantine Fault Tolerance* (PBFT) [Castro and Liskov 2002] foi o primeiro protocolo de consenso BFT que considera aspectos práticos, alcançando um desempenho semelhante aos protocolos para falhas por parada e resolve o problema dos generais bizantinos [Lamport et al. 1982]. Neste protocolo, a autenticidade das mensagens é garantida através de vetores de MACs (*Message Authentication Codes*) ao invés de assinaturas digitais, aumentando o seu desempenho ao usar criptografia simétrica em vez de criptografia assimétrica. O PBFT executa em rodadas (*rounds*), sendo que em cada rodada (*round*) um participante do consenso é escolhido líder e tentará fazer da sua proposta de novo bloco para a corrente de blocos. O PBFT necessita de $n = 3f+1$ participantes do consenso, com réplicas da corrente de blocos, para tolerar até f falhas bizantinas e utiliza quóruns de $2f+1$ participantes de consenso em cada passo do protocolo. Também é necessário um modelo de sistema de comunicação parcialmente síncrono para garantir terminação. A Figura 3.8 ilustra os passos do protocolo em uma execução normal, i.e., quando o líder é correto e o sistema está em um período de sincronia. Os passos do protocolo são os seguintes:

- o líder envia uma mensagem de PRE-PREPARE para todos os participantes com a sua proposta de novo bloco, p;
- os participantes do consenso fazem algumas validações para definir se a proposta de novo bloco, p, enviada pelo líder é válida. Em caso afirmativo aceitam esta proposta e enviam uma mensagem de PREPARE contendo a proposta de novo bloco, p, para todos os outros participantes do consenso;
- quando um participante do consenso receber $\lceil \frac{n+f}{2} \rceil$ mensagens PREPARE para uma mesma proposta de novo bloco, p, ele envia uma mensagem de COMMIT contendo p para todos os outros participantes do consenso;
- quando um participante do consenso receber $\lceil \frac{n+f}{2} \rceil$ mensagens de COMMIT para uma mesma proposta de novo bloco, p, então a proposta p, é decidida.

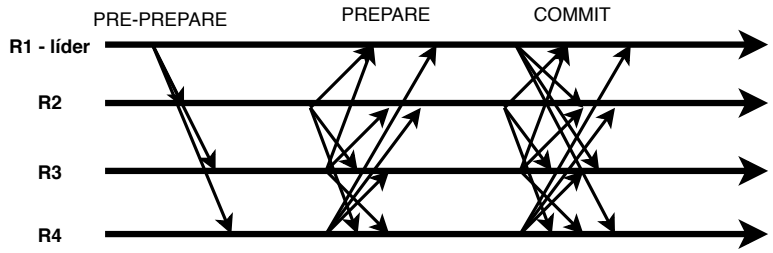


Figura 3.8: Execução normal do PBFT.

Quando o líder for malicioso e fizer propostas divergentes, um quórum de mensagens PREPARE ou COMMIT para uma mesma proposta não é obtido, o mesmo ocorre em um período de perturbação do sistema em que as mensagens não recebidas até um limite de tempo estipulado. Nestes casos, um protocolo de troca de visão é executado e uma nova rodada (*round*) é iniciada com um novo líder.

O Protocolo MinBFT [Veronese et al. 2013] requer a mesma quantidade de réplicas e passos de comunicação dos protocolos que toleram apenas falhas por parada, como por exemplo, o Paxos [Lamport 1998]. Para isso, faz uso de componentes seguros nos participantes do consesno para restringir as ações de participantes maliciosos. Os componentes seguros propostos para auxiliar no desenvolvimento de protocolos que tolerem falhas maliciosas possuem diferenças consideráveis tanto em aspectos de implementação quanto de desempenho. Quanto à implementação, estes componentes podem ser: i) implementados de forma distribuída, como o *Trusted Timely Computing Base* (TTCB) [Correia et al. 2004]; ii) locais baseados em memória, como o *Attested Append-Only Memory* (A2M) [Chun et al. 2007]; e iii) locais baseados em um simples contador, como o *Unique Sequential Identifier Generator* (USIG) [Veronese et al. 2013].

Basicamente, um atacante não consegue acessar estes componentes, mesmo que consiga invadir um servidor. Com isso, é possível projetar protocolos que restringem as ações que um processo malicioso (invadido) pode executar sem ser descoberto. O MinBFT utiliza o componente seguro USIG, que é muito simples e de fácil implementação e utilização no sistema. Este componente é local em cada replica e é usado para

atribuir um identificador único para uma mensagem, além de assiná-la. Para cada replica, os identificadores atribuídos às mensagens são únicos (um mesmo identificador nunca é atribuído a duas ou mais mensagens), monotônicos (o identificar atribuído a uma mensagem nunca é menor do que o anterior) e sequenciais (o identificador atribuído a uma mensagem sempre é o sucessor do anterior).

A interface definida para acessar este serviço é a seguinte [Veronese et al. 2013]:

- `createUI(m)`: retorna um certificado que contém um identificador único `UI` e a prova de que este identificador foi criado por este componente e atribuído a `m`. O identificador é basicamente um contador monotonicamente crescente que é incrementado a cada chamada desta função. Já a prova envolve criptografia e pode ser baseada em um *hash* (*Hash-based Message Authentication Code* – HMAC) ou em criptografia assimétrica (assinaturas digitais).
- `verifyUI(PK,UI,m)`: verifica se o identificador único `UI` é válido para `m`.

Com o uso deste componente, uma replica não consegue enviar duas mensagens diferentes para processos diferentes com um mesmo identificador. Assim, basta que cada replica mantenha armazenado o identificador da última mensagem recebida de uma replica para saber qual é o próximo identificador esperado, e assim reduz-se as ações de uma replica maliciosa: ou envia a mesma mensagem (que pode conter qualquer valor) para todos as outras replicas ou não envia nada.

Como o PBFT, o MinBFT também executa em *rounds*. Porém, o MinBFT necessita de apenas $n = 2f + 1$ replicas para tolerar até f falhas bizantinas e utiliza quóruns de $f + 1$ replicas em cada passo do protocolo. Também é necessário um modelo de sistema parcialmente síncrono para garantir terminação. A Figura 3.9 ilustra os passos do protocolo em uma execução normal, i.e., quando o líder é correto e o sistema está em um período de sincronia. Os passos do protocolo são os seguintes:

- O líder envia uma mensagem de PREPARE para todas as réplicas com a sua proposta p. Neste passo, o USIG é utilizado para evitar que o líder envie propostas diferentes para as replicas, i.e., um UI (identificar único é associado a esta mensagem).
- As replicas fazem algumas verificações para definir se a proposta p é válida (por exemplo, verificam se o UI é válido), e em caso afirmativo enviam uma mensagem de COMMIT contendo p para todas as outras replicas. Neste passo, o USIG também é utilizado para evitar que uma replica maliciosa envie mensagens diferentes para diferentes replicas.
- Quando uma replica receber $f + 1$ mensagens de COMMIT válidas (UI válido) para uma mesma proposta p, então p é decidido.

Como no PBFT, pode ser que nem todas as replicas consigam decidir em um *round*, neste caso um protocolo de troca de visão também é executado para inicializar um novo *round*, até que todas as replicas consigam decidir pelo mesmo valor.

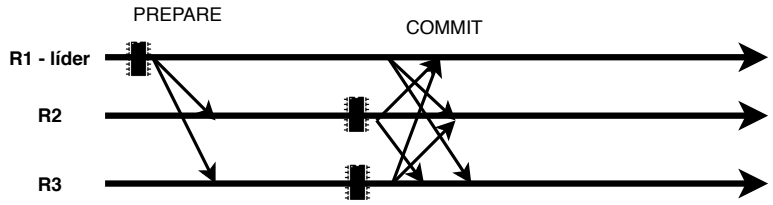


Figura 3.9: Execução normal do MinBFT.

O Protocolo Zyzzyva - é utilizado para implementar uma camada de ordenação em aplicações distribuídas. Por exemplo, podem implementar a camada de ordenação de uma Replicação Máquina de Estados ou a camada de ordenação que mantém a consistência de uma corrente de blocos. No contexto de uma corrente de blocos, podemos pensar como um minerador que envia um bloco para ser adicionado na corrente, e o protocolo de consenso define a ordem em que os blocos são adicionados na mesma. Neste cenário, o Zyzzyva [Kotla et al. 2009] utiliza execução especulativa para melhorar o desempenho destes protocolos, i,e., o bloco é adicionado na corrente antes da ordenação e o minerador verificará que não ocorreu desacordo. Usando este padrão de comunicação, o Zyzzyva requer menos passos de comunicação e melhora o desempenho do sistema. O Zyzzyva utiliza as mesmas suposições do PBFT em relação ao número de replicas e modelo de sistema. A Figura 3.10 apresenta o padrão de comunicação para o caso normal de execução, que é como segue:

- Primeiramente, o minerador (cliente) envia o bloco (requisição) a ser adicionado (executado) na corrente de blocos para a replica líder.
- A réplica líder define uma ordem e envia este bloco para todas as outras réplicas.
- As réplicas especulativamente incluem este bloco na corrente de blocos e enviam uma resposta para o minerador, que aguarda por todas as respostas $(3f + 1)$ e verifica se ocorreu desacordo. Note que isso somente ocorre quando o líder for malicioso e envia ordens diferentes de inclusão de blocos na corrente para diferentes replicas. Também é utilizado um *timeout* no minerador para verificar se seu bloco foi incluído na corrente. Caso ocorra um *timeout* antes de receber as respostas ou o minerador defina que ocorreu um desacordo, protocolos adicionais são utilizados e o minerador envia uma mensagem para as replicas com as informações recebidas. Neste caso as replicas podem executar uma troca de visão e inicializar um novo *round* (caso ocorreu desacordo) ou apenas executar o *commit* daquela operação (caso ocorreu o *timeout*, mas o minerador tinha recebido pelo menos $2f + 1$ respostas iguais).

Vale destacar que o processo a enviar o bloco para inclusão na corrente poderia ser o próprio líder (como exemplificado nas figuras anteriores para o PBFT e MinBFT), neste caso fica claro a redução na quantidade de passos de comunicação necessárias para incluir um bloco na corrente. De qquer forma, após a inclusão ainda existe um passo adicional para verificar se ocorreu desacordo. De forma resumida, caso tenha ocorrido desacordo, um protocolo é utilizado para reiniciar um novo *round* com um novo lider.

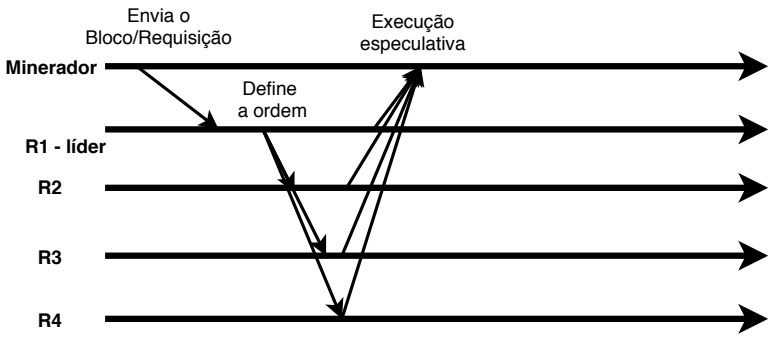


Figura 3.10: Execução normal do Zyzzyva.

O Protocolo de Consenso BFT entre Participantes Desconhecidos (*BFT Consensus with Unknown Participants* BFT-CUP) [Alchieri et al. 2008, Alchieri et al. 2018, Cavin et al. 2004, Cavin et al. 2005, Greve and Tixeuil 2007, Greve and Tixeuil 2010] foi proposto para a solução do problema do consenso em ambientes dinâmicos e auto-organizáveis. Nestes protocolos, cada participante inicialmente conhece apenas um subconjunto dos participantes da rede e, baseado neste conhecimento inicial, expandem este conhecimento a respeito dos outros participantes do sistema para então estabelecer um consenso.

O conhecimento inicial de cada participante é encapsulado em um módulo local chamado de detectores de participação. A união do conhecimento inicial obtido por cada participante da computação distribuída pode ser representado como um grafo direcionado de conectividade por conhecimento, no qual os vértices representam participantes e cada aresta representa o conhecimento entre dois participantes. É importante não confundir este grafo de conhecimento com o grafo de conectividade da rede. Por exemplo, na Figura 3.11, o participante 4 conhece o participante 9, enquanto que os participantes 5 e 6 se conhecem um ao outro. Também é importante perceber que este grafo não é conhecido pelos participantes do sistema, pois cada um apenas tem a informação sobre o seu conhecimento inicial e mesmo que posteriormente este conhecimento seja expandido, cada participante consegue obter apenas uma visão parcial do sistema.

Dado este conhecimento inicial, os protocolos de consenso primeiramente tentam expandir este conhecimento através de uma busca no grafo. Depois disso, é definido o componente poço do grafo, i.e., um subgrafo composto pelos participantes que se conhecem e compartilham a mesma visão do sistema (veja a Figura 3.11). Após a definição desta componente, os participantes no poço executam um protocolo de consenso tradicional (por exemplo, o PBFT [Castro and Liskov 2002]) e enviam o valor da decisão para os outros participantes do sistema fora do poço. Os trabalhos existentes nesta área estabeleceram o conhecimento inicial mínimo que os participantes devem obter para resolver o consenso considerando diferentes modelos de sistemas: síncrono sem falhas [Cavin et al. 2004], síncrono com falhas por parada [Cavin et al. 2005], parcialmente síncrono com falhas por parada [Greve and Tixeuil 2007], e parcialmente síncrono com falhas Bizantinas [Alchieri et al. 2008, Alchieri et al. 2018].

O BFT-SMaRt [Bessani et al. 2014] é uma implementação Java robusta para Replicação Máquina de Estados [Schneider 1990] que utiliza um protocolo consenso para

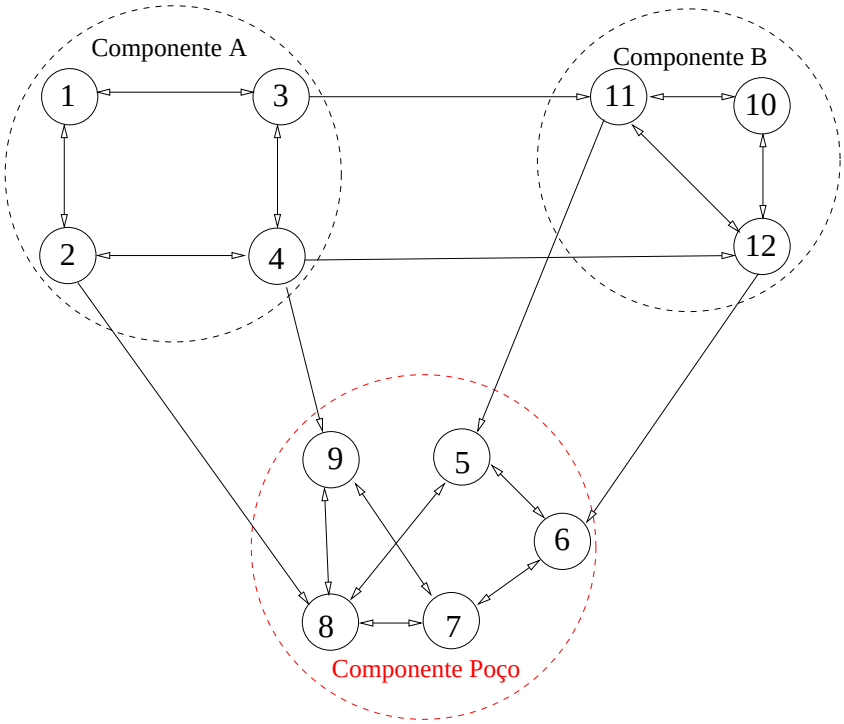


Figura 3.11: Grafo de conectividade por conhecimento.

ordenar requisições. O BFT-SMaRt pode ser configurado para tolerar tanto falhas por parada, e neste caso utiliza um protocolo similar ao Paxos [Lamport 2001], quanto bizantinas, e neste caso utiliza um protocolo similar ao PBFT [Castro and Liskov 2002]. Esta biblioteca de replicação foi desenvolvida buscando-se um alto desempenho em execuções livres de falhas e corretude nos casos em que replicas faltosas estarem presentes no sistema. Além disso, o BFT-SMaRt é a primeira implementação de Replicação Máquina de Estados que permite reconfigurações no conjunto de replicas [Lamport et al. 2010] e fornece um suporte eficiente e transparente para serviços duráveis [Bessani et al. 2013].

Apesar de não ser um protocolo de consenso, o BFT-SMaRt pode ser utilizado para ordenar os pedidos de inserção de blocos em uma cadeia de blocos, pois todas as requisições são entregues na mesma ordem em todas as replicas do sistema. Note que para isso é utilizado um protocolo de consenso subjacente, conforme já comentado. O BFT-SMaRt foi incorporado como a camada de ordenação no Hyperledger Fabric v1.3 [Sousa et al. 2018].

3.3.6. Protocolos de Consenso Híbridos

Os protocolos híbridos procuram combinar dois ou mais modelos de consenso com o objetivo de herdar as vantagens dos tipos de consenso usados. Um exemplo desta classe de protocolos de consenso é o protocolo Tendermint que é detalhado a seguir.

O Protocolo Tendermint [Kwon 2014, Buchman 2016] é um protocolo de consenso tolerante a falhas bizantinas, que não requer mineração por prova de trabalho e oferece proteção contra gasto duplo, simultaneamente apresentando uma taxa de milhares de tran-

sações por segundo e segurança baseada em fatores intrínsecos. O protocolo atua de modo similar a prova de participação (*proof of stake*), com os nós responsáveis por adicionar novos blocos a rede oferecendo parte de seus ativos como um modo de atuarem no processo de efetivação. Diferentemente da prova de participação, o Tendermint foi desenvolvido tendo ataques de nada a perder (*nothing at stake*) em mente, implementando medidas de segurança para proteger a rede destes ataques. Como o protocolo é tolerante a falhas bizantinas, garante-se a segurança da corrente contra até 1/3 de participantes bizantinos.

Cada bloco adicionado a corrente é constituído por 3 partes: o cabeçalho, os validadores do bloco anterior e as transações inclusas nesse bloco. Para a determinação do *hash* do bloco, primeiramente utiliza-se uma função resumo em cada uma dessas partes; a seguir, esta função é utilizada novamente sobre os hashs obtidos. Este hash do bloco é utilizado em uma Árvore de Merkle (Merkle Tree) de modo a permitir a verificação de qualquer bloco na corrente, garantindo assim sua auditabilidade.

Validadores são contas que possuem moedas em um depósito de títulos (*bond deposit*), com essas moedas determinando o poder de votação do validador. Para adicionar moedas ao depósito, o usuário deve transmitir uma transação de título (*bond transaction*), indicando à rede quantas moedas serão utilizadas com este propósito. Esses validadores participam do consenso difundindo assinaturas criptográficas, ou votos, para concordar na adição de um bloco à corrente. Essas moedas podem ser transferidas e gastas, desde que não estejam armazenadas no depósito de títulos; para removê-las desse depósito, deve-se transmitir uma transação de desvínculo (*unbond transation*) com este objetivo, e aguardar por um período pré-determinado denominado período de desvínculo (*unbonding period*). O funcionamento das transações de título e desvínculo é apresentado na Figura 3.12. O poder de votação da rede é igual a soma do poder de votação de todos os participantes.

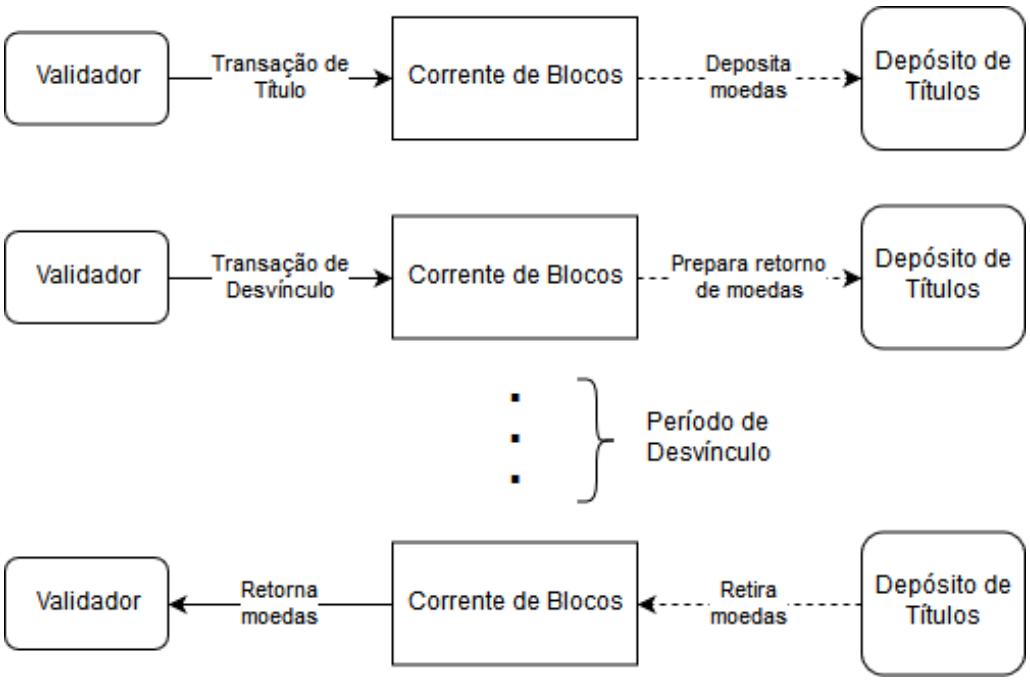


Figura 3.12: Transações de título e desvínculo do protocolo Tendermint, utilizadas para modificar o poder de votação de cada validador.

O processo de adição de novos blocos a corrente se dá através de rodadas. Cada rodada é composta por 3 passos: proposta (*propose*), pré-voto (*prevote*) e pré-efetivação (*precommit*), com os validadores enviando votos em cada passo da rodada. Existem 3 tipos de votos: o pré-voto (*prevote*), a pré-efetivação (*precommit*) e a efetivação (*commit*). Um bloco é dito efetivado pela rede quando assinado e difundido por mais de 2/3 do poder de votação dos validadores.

As etapas que constituem uma rodada estão apresentadas na Figura 3.13. Cada rodada é feita de modo que um proponente (*proposer*) seja escolhido entre os validadores, onde os que possuem maior poder de votação são escolhidos com maior frequência. Inicialmente o proponente escolhido anuncia o bloco através de uma proposta, com os outros nós confirmando sua validade, e que este foi recebido a tempo, através de um pré-voto. Quando mais de 2/3 do poder de votação da rede realizar o pré-voto para este bloco, realiza-se então a pré-efetivação antes que se inicie o processo de efetivação do bloco. Esta também ocorre quando mais de 2/3 do poder de votação da rede concordar na pré-efetivação do bloco.

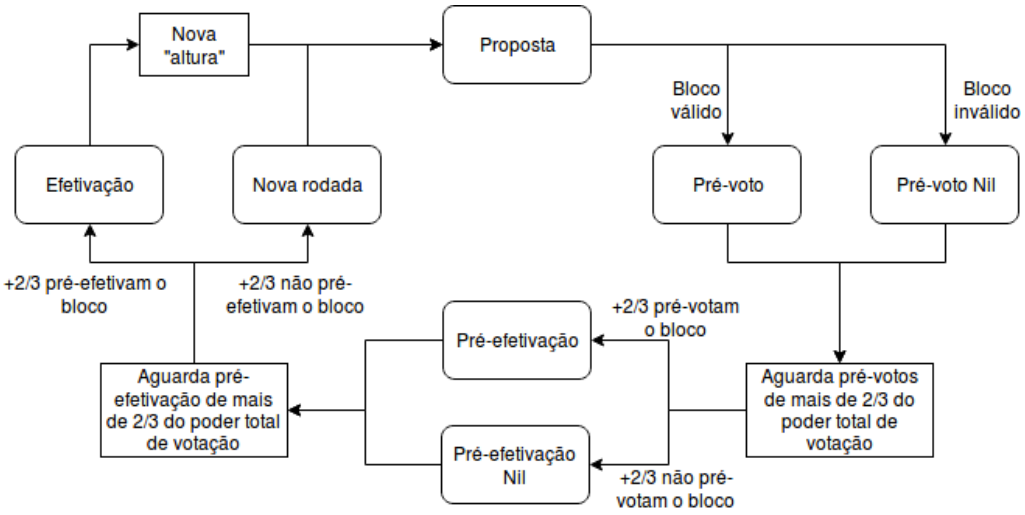


Figura 3.13: Etapas de uma rodada do protocolo Tendermint.

Um bloco é considerado efetivado quando pelo menos 2/3 do poder total de votação assina votos de efetivação para esse bloco. Uma bifurcação (*fork*) ocorre quando dois ou mais blocos na mesma "altura" são assinados por uma maioria de 2/3 do poder de votação, indicando que pelo menos 1/3 dos validadores votaram com duplicidade. Quando isto ocorre, uma transação de evidência (*evidence transaction*) é gerada, destruindo as moedas de título dos validadores culpados. O período de desvínculo garante que validadores maliciosos ainda possuirão suas moedas no depósito de título quando a transação de evidência for gerada, desse modo assegurando que a punição pelo ataque será devidamente realizada. Essa medida evita que validadores votem para efetivar diferentes blocos em uma única rodada, devido ao risco de ocorrer prejuízo financeiro, desencorajando assim ataques de nada a perder.

3.3.7. Protocolo baseado em Grafo Acíclico Direcionado - IOTA Tangle

O IOTA [Popov 2017] é uma criptomoeda construída para atender a micropagamentos máquina a máquina (*machine to machine* - M2M) característicos de um ambiente de Internet das Coisas. Seus mecanismos de pagamento e protocolo de consenso, formalizados por Popov em 2016 [Popov 2017], baseiam-se em uma estrutura de dados inovadora chamada *Tangle*. A principal inovação do *Tangle* é a estrutura de livro-razão distribuído, que reúne as transações da rede em um grafo acíclico direcionado (*directed acyclic graph* - DAG) em vez de utilizar uma corrente de blocos. Além disso, o Tangle elimina a distinção entre clientes e mineradores: todos os usuários do sistema devem realizar trabalho para emitir uma nova transação. Uma característica notável do *Tangle* em comparação ao consenso em corrente de blocos é que diferentes participantes na rede podem ter as diferentes visões das transações. Esta característica contrasta fortemente com a visão global da corrente de blocos, na qual todas as transações são idênticas em qualquer participante.

A Figura 3.14 mostra um exemplo da estrutura de dados do *Tangle*. Cada vértice do grafo representa uma transação e cada aresta representa o resultado da validação de uma transação. O usuário deve confirmar ao menos duas transações não-confirmadas para adicionar sua transação à *Tangle*[18]. As transações não-confirmadas são chamadas de "pontas"(*tips*) do *Tangle*. Para adicionar uma transação à rede, o usuário adiciona os resumos das duas pontas escolhidas à sua transação, resolve um desafio baseado em prova de trabalho, e difunde o resultado na rede. A prova de trabalho, neste caso, tem dificuldade bem menor que a do Bitcoin e serve apenas como um mecanismo para prevenir *spam* de transações. O procedimento de adição de uma transação cria duas novas arestas direcionadas no grafo que confirmam as transações anteriores e funcionam como uma versão generalizada da sequência de funções resumo da corrente de blocos. O IOTA Tangle não recompensa financeiramente os validadores de transações, pois o incentivo para validar transações é adicionar sua própria transação à rede. Todas as moedas trocadas na rede são criadas na primeira transação do sistema, chamada de transação gênesis.

Um dos conceitos fundamentais no IOTA é o peso de transações: o peso individual e peso acumulado. O peso individual de uma transação é um peso proporcional aos recursos gastos por um usuário para emitir a transação. A ideia dos pesos individuais é diferenciar transações mais importantes, que possuem maior peso individual, de transações com menor importância, que possuem menor peso individual. O peso acumulado de uma transação é definido pelo seu peso individual mais a soma de todos os pesos individuais das transações que aprovam, direta ou indiretamente, a transação. Uma transação aprova diretamente outra se há uma aresta conectando as duas, e indiretamente se há pelo menos um caminho composto de mais de uma aresta entre as duas transações.

O consenso do IOTA Tangle não é realizado *a priori* não precisam obter consenso sobre quais transações podem ser adicionadas ao grafo. Todas as transações são adicionadas, bastando que o usuário resolva o desafio computacional necessário para encadear as transações. No entanto, quando há pontas conflitantes, cada usuário precisa decidir quais transações serão consideradas válidas para escolher duas pontas que serão aprova-

[18]Na implementação do IOTA, o número de confirmações necessárias para adicionar uma transação à rede é exatamente dois.

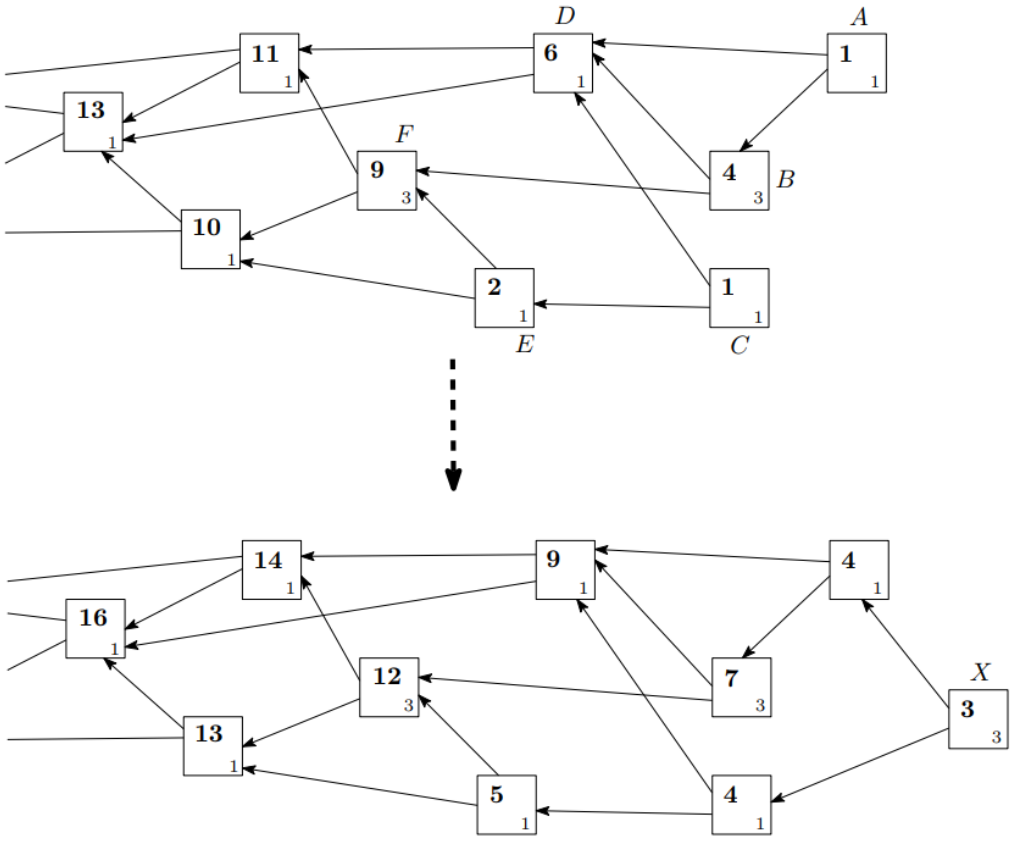


Figura 3.14: A adição de uma ponta, X, na estrutura de dados do *Tangle*. O peso individual de cada transação é representado no canto direito inferior e o peso acumulado é representado pelo número em negrito.

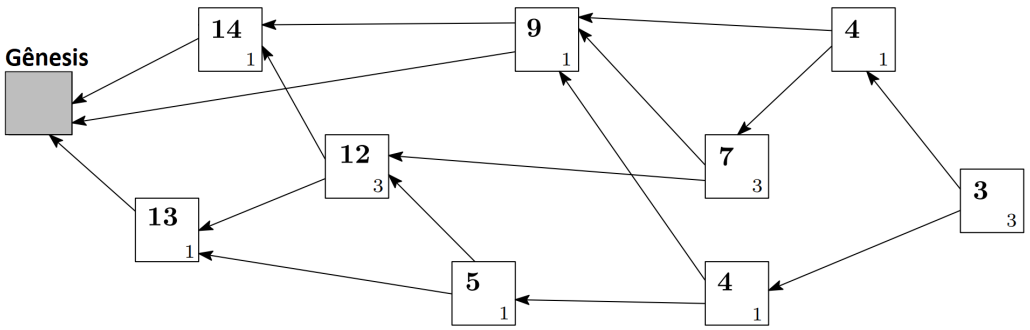


Figura 3.15: Um exemplo de estrutura de dados baseada em grafos acíclicos direcionados (DAG) do *Tangle*. Cada vértice do grafo representa uma transação e cada aresta representa o resultado da validação de uma transação. O número no canto inferior direito de cada transação representa o peso individual da transação, e o número ao centro, em negrito, representa o peso acumulado da transação.

das por sua nova transação. As pontas inválidas são ignoradas pelo usuário. O principal mecanismo para identificar pontas válidas é realizar múltiplas rodadas do algoritmo de seleção de pontas (*tip selection algorithm*) e verificar qual das duas pontas conflitantes tem a maior probabilidade de ser escolhida. Por exemplo, se uma ponta foi escolhida 95 vezes

em 100 rodadas do algoritmo de seleção, a ponta tem confança de 95%. O IOTA Tangle atualmente utiliza um algoritmo de seleção baseado em passeios aleatórios (*random walks*) e cadeias de Markov e Monte Carlo (*Markov-chain and Monte Carlo* - MCMC) que prioriza transações de maior peso acumulado.

Quando existem mais de duas pontas válidas disponíveis para seleção, o usuário escolhe as duas pontas de acordo com o peso acumulado de cada ponta.

3.4. Atividade Prática (*Hands-On*): Desenvolvimento de uma Corrente de Blocos para Telecomunicações

O objetivo da atividade prática é evidenciar a segurança proporcionada pelo uso de corrente de blocos em um ambiente de funções virtualizadas de rede (*Virtualized Network Function* - VNF), um cenário comum de Internet do Futuro em telecomunicações. A corrente de blocos desta prática registra operações de criação e configuração de VNFs, garantindo confiabilidade, integridade e auditabilidade das informações armazenadas. A atividade prática utiliza, como estudo de caso, a arquitetura de fatiamento de rede (*network slicing*) apresentada na Figura 3.16 com dois tipos de corrente de blocos [Rebello et al. 2019c]. A arquitetura apresentada fornece a capacidade de auditoria de criação e gerenciamento de fatias, registrando todas as operações de orquestração da VNF em uma corrente de blocos de gerenciamento. Além disso, as propriedades de automação e transparência dos contratos inteligentes são apropriadas para criar fatias de rede fim-a-fim seguras que envolvem VNFs em vários domínios concorrentes, pois garantem que todos os nós da rede obedeçam ao mesmo conjunto de regras e que o código executado seja visível para qualquer nó participante. A demonstração usa a plataforma *Hyperledger Fabric* para a criação de uma corrente de blocos permissionada. O *Hyperledger Fabric* é uma plataforma de código aberto para o desenvolvimento de correntes de blocos permissionadas em um ambiente sem confiança entre os participantes.

3.4.1. A Arquitetura da Plataforma Hyperledger Fabric

O Hyperledger[19] é uma iniciativa colaborativa e de código aberto da Linux Foundation que objetiva desenvolver tecnologias baseadas em corrente de blocos para a indústria. A colaboração global inclui empresas de tecnologia da informação, mercado financeiro, Internet das Coisas, cadeias de suprimentos, saúde e financiamento.

Um dos diversos projetos inclusos na iniciativa Hyperledger é o *Hyperledger Fabric* [Androulaki et al. 2018], uma plataforma proposta e desenvolvida pela IBM para a implementação de redes permissionadas baseadas em corrente de blocos. O Hyperledger Fabric é o projeto mais maduro da iniciativa Hyperledger e o mais utilizado por empresas que desenvolvem aplicações privadas baseadas em corrente de blocos. A arquitetura modular e plugável do Fabric permite o uso de diferentes tipos de protocolos de consenso, diferentes linguagens de programação para a criação de contratos inteligentes e diferentes bancos de dados para armazenar a corrente de blocos e seu estado atual. Outra proposta do Hyperledger Fabric é criar redes nas quais podem coexistir diversas correntes de blocos isoladas, independentes, e com diferentes configurações, contratos inteligentes e participantes. As entidades que participam de uma rede baseada em correntes de blocos

[19]Disponível em https://www.hyperledger.org/

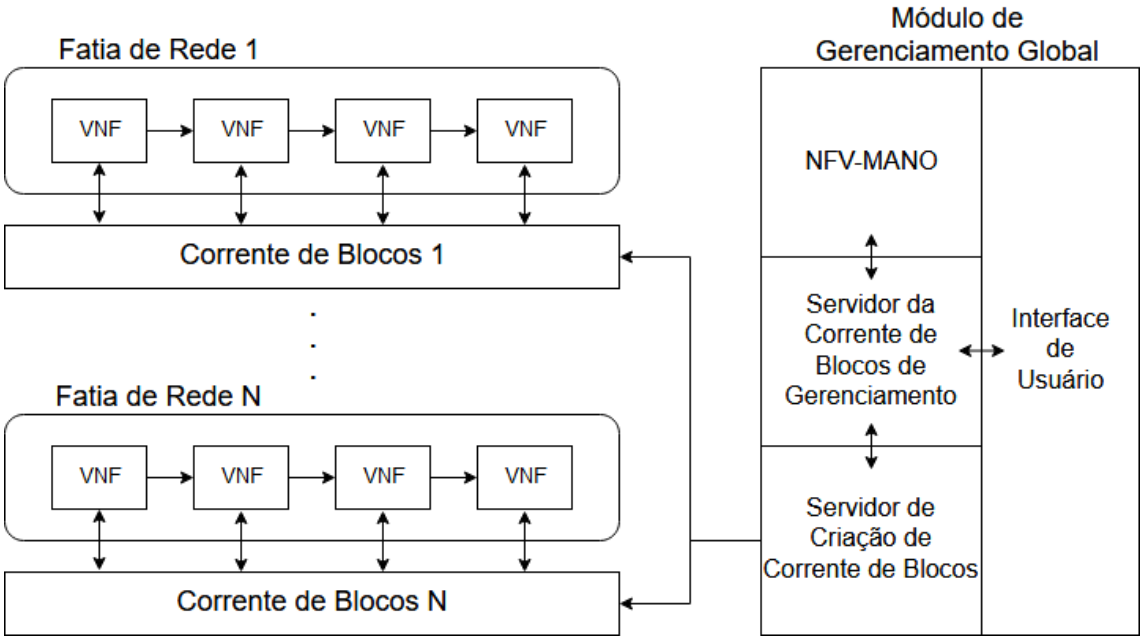


Figura 3.16: A arquitetura proposta por Rebello *et al.* baseada em corrente de blocos para fatiamento de rede [Rebello et al. 2019c]. O usuário interage com o módulo de gerenciamento global para criar fatias de rede seguras. Cada VNF em uma fatia de rede é conectada a uma corrente de blocos responsável por registrar solicitações de configuração e informações relevantes, conforme especificado pelo usuário.

do Fabric podem ter diferentes funções e permissões de acesso à corrente de blocos. As entidades com poder administrativo podem modificar as permissões de escrita e leitura das demais entidades em cada corrente de blocos através de transações de configuração, enquanto entidades comuns têm permissão para emitir transações que não sejam sensíveis ao funcionamento da rede, como uma transferência de recursos entre duas entidades. Há a possibilidade de definir políticas específicas e adaptadas para a validação de transações que exigem a assinatura por múltiplos validadores. Os desenvolvedores das correntes de blocos no Fabric podem configurar permissões de leitura e escrita para criar redes permissionadas, nas quais todos os nós da rede se conhecem. Isso é apropriado para ambientes empresariais ou de consórcio, tipicamente constituídos por até dezenas de nós. O Hyperledger Fabric fornece um serviço de identidade e associação de membros que gerencia os identificadores dos usuários, e autentica todos os participantes na rede automaticamente. Além disso, podem existir nós internos em cada organização/empresa. Todas essas configurações disponíveis no Hyperledger Fabric são adequadas e facilitam a implementação de correntes de blocos adaptadas a cada aplicação, com necessidades e características diferentes, em um ambiente empresarial.

Nós e canais são os conceitos-chave mais importantes de uma rede baseada em corrente de blocos permissionada do Hyperledger Fabric. Os nós representam as entidades que participam do processamento de uma transação ou mantêm uma cópia da corrente de blocos. O Hyperledger Fabric provê três tipos de nós: clientes, pares (*peers*) e ordenadores. Um cliente representa um usuário que envia transações aos pares para validação e assinatura, e transmite propostas de transação assinadas para o serviço de ordenação.

Os pares são um elemento fundamental da rede porque executam propostas de transação, validam transações e mantém os registros na corrente de blocos. Os pares também instanciam contratos inteligentes e armazenam o estado global, uma representação sucinta do estado mais recente da corrente de blocos. Os nós ordenadores formam coletivamente o serviço de ordenação, que é responsável por estabelecer a ordem total e o empacotamento de todas as transações em um bloco usando um protocolo de consenso. Os ordenadores não participam da execução da transação nem validam transações. O desacoplamento das funcionalidades de ordenação e validação aumenta a eficiência da rede, pois permite o processamento paralelo de cada fase. A Figura 3.17 descreve um exemplo de uma corrente de blocos permissionada com quatro organizações no Hyperledger Fabric. Cada organização recebe transações de clientes e as retransmite para os ordenadores após a validação pelos pares. Cada organização possui um único ordenador, garantindo a justiça no protocolo de consenso.

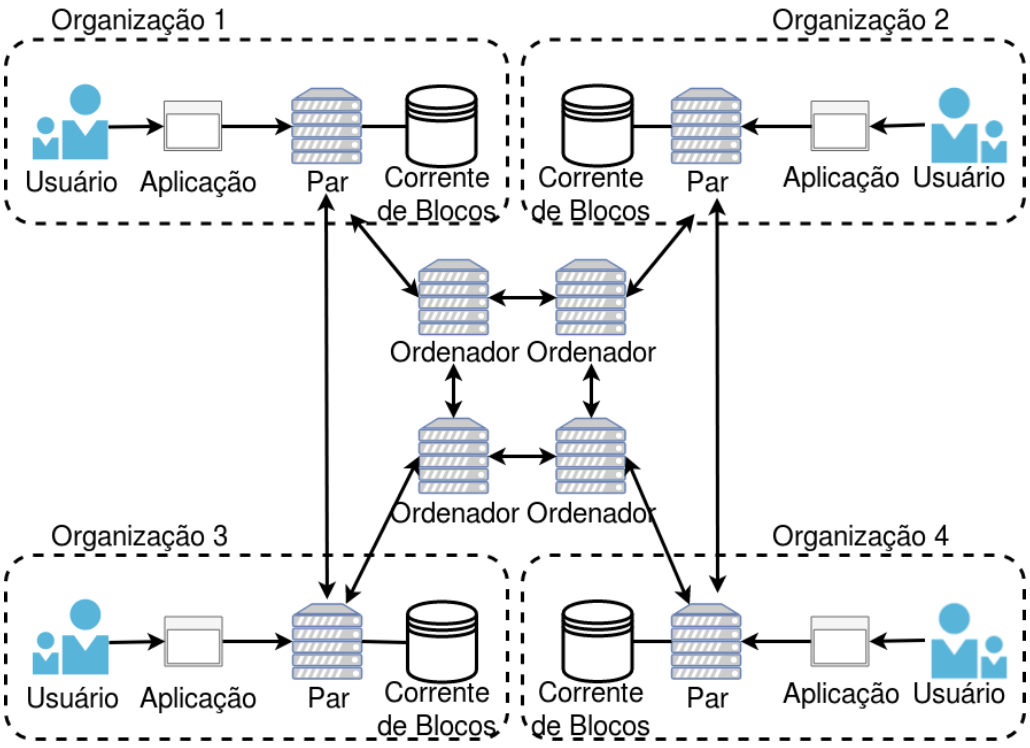


Figura 3.17: Um exemplo de corrente de blocos permissionada do Hyperledger Fabric. Usuários de cada organização usam aplicações para emitir transações assinadas e as submetem para validação nos pares. Os pares validam a transação e respondem à aplicação, que retransmite a transação validada para os ordenadores. Os ordenadores trocam mensagens para definir a ordenação global das transações recebidas em um intervalo de tempo e as adicionam em um novo bloco. Depois de um bloco ser proposto e aceito pelos ordenadores através do consenso, os pares atualizam a corrente de blocos e o estado global da rede.

Caminhos de mensagens diferentes, chamados canais, isolam as correntes de blocos. Um canal Hyperledger Fabric é uma sub-rede de comunicação privada e isolada entre um subconjunto de nós da rede específicos para fornecer privacidade e confidencialidade

às transações. Todos os dados transmitidos em um canal, incluindo transações, contratos inteligentes, configurações de associação e informações de canal, são invisíveis e inacessíveis a qualquer entidade externa a um canal. As mensagens trocadas em um canal são criptografadas. A funcionalidade do canal é ideal para a proposta de oferecer correntes de blocos personalizadas para diferentes serviços de rede, pois permite que os administradores dos canais estabeleçam diferentes formatos de bloco e transação, além do protocolo de consenso, para cada canal. Portanto, é possível usar canais para oferecer fatias de rede protegidas por correntes de blocos configuradas de forma específica. Formatos de transação são definidos em contratos inteligentes, chamados de *chaincode* no Hyperledger Fabric, escritos em Go, Node.js ou linguagem Java.

3.4.2. Configuração do Ambiente e Experimentação

Para realizar esta atividade, é necessário uma máquina com acesso à Internet que possua pelo menos 4 GB de memória RAM e processador Intel Core i3 ou equivalente. A arquitetura do sistema deve ser de 64 bits e recomenda-se o uso de um sistema operacional baseado em Linux por experiência de uso, maior facilidade para programação e por serem baseados em código aberto e gratuito. São disponibilizados dispositivos USB removíveis (*pen drives*) com imagens para criação de uma máquina virtual com sistema operacional *Debian 9.0 (Stretch)* que inclui todos os pré-requisitos necessários para utilização do Hyperledger Fabric já instalados. Vale ressaltar que o computador utilizado pelo participante deve possuir uma ferramenta de virtualização como o VirtualBox[20] ou VMWare Workstation Player[21] para utilizar a máquina virtual fornecida pelos autores.

Para dar maior dinamicidade à aula prática, os participantes que possuem acesso à Internet também podem baixar e instalar os pacotes necessários manualmente enquanto os demais participantes obtêm a imagem dos dispositivos removíveis. Os pré-requisitos estão listados na documentação do Hyperledger Fabric[22] e consistem dos seguintes itens:

- Instalar o cURL[23]. Para Debian, executar o comando `apt-get install curl`.
- Instalar o Docker[24] na versão 17.06.2-ce ou superior. Para Debian, executar os comandos na seguinte ordem:

  i) `apt-get install apt-transport-https ca-certificates gnupg2 software-properties-common`

  ii) `curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add -`

  iii) `add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release -cs) stable"`

[20]Disponível em https://www.virtualbox.org/
[21]Disponível em https://www.vmware.com/br/products/workstation-player.html
[22]Disponível em https://hyperledger-fabric.readthedocs.io/en/release-1.4/prereqs.html
[23]Disponível em https://curl.haxx.se/download.html
[24]Disponível em https://www.docker.com/get-started

iv) `apt-get update && apt-get install docker-ce docker-ce-cli containerd.io`

- Instalar Docker Compose[25] na versão 1.14.0 ou superior. Para Debian, executar os comandos na seguinte ordem:

  i) `curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose`

  ii) `chmod +x /usr/local/bin/docker-compose`

  iii) `ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose`

- Instalar a linguagem Go[26] na versão 1.11.x. Para Debian, executar os comandos na seguinte ordem:

  i) `wget https://dl.google.com/go/go1.11.5.linux-amd64.tar.gz`

  ii) `tar -C /usr/local -xzf go1.11.5.linux-amd64.tar.gz`

  iii) `export PATH=$PATH:/usr/local/go/bin && export GOPATH=$HOME/go && export PATH=$PATH:$GOPATH/bin`

  iv) `source $HOME/.profile`

- Python na versão 2.7. Verifique a versão com o comando `python --version`

- Instalar os executáveis e as imagens de Docker do Hyperledger Fabric versão 1.4.1. Para Debian, executar os comandos na seguinte ordem:

  i) `cd /root && git clone https://github.com/hyperledger/fabric-samples.git`

  ii) `cd fabric-samples`

  iii) `curl -sSL http://bit.ly/2ysbOFE | bash -s -- 1.4.1 1.4.1 0.4.15`

3.4.3. Instalação de uma Corrente de Blocos

Esta aula prática desenvolve um protótipo de aplicação que usa a plataforma Hyperledger Fabric [Androulaki et al. 2018] para implementar correntes de blocos entre organizações em ambientes sem confiança. Cada organização mantém uma réplica da corrente de blocos e pode acrescentar blocos através de um protocolo de consenso. O protótipo implementado segue a arquitetura de fatiamento de rede apresentada na Seção 3.4. O protótipo implementa cada nó da rede do Hyperledger Fabric como um contêiner em um único computador e envia transações simultaneamente. A aula prática implementa dois contratos inteligentes[27] escritos em Go, que são executados em todos os nós da rede [Alvarenga et al. 2018, Rebello et al. 2019a].

[25] Disponível em https://docs.docker.com/compose/install
[26] Disponível em https://golang.org/dl/
[27] O código completo está disponível em https://github.com/gta-ufrj/hpsr-smart-contracts

```
struct instructionTransaction
{
    command                 string
    transactionType         string
    transactionName         string
    issuer                  string
}
initialize queue
initInstruction (instruction <command,name,issuer>)
{
    if instruction is not unique or instruction is not well-formatted:
        return error
    putState (instruction.name, instruction)
    put (transactionID, queue)
    notify orchestrator
    return success
}
```

Lista 3.1: Parte do pseudocódigo do contrato inteligente que emite transações de instrução. O campo de comando contém a operação de orquestração a ser executada por um orquestrador. O contrato estabelece uma fila de instruções pendentes a serem processadas pelo orquestrador.

O primeiro contrato inteligente, parcialmente descrito na Lista 3.1, gerencia autonomamente o gerenciamento e a orquestração de VNF através de transações de instrução e resposta. Quando um cliente solicita uma fatia, o servidor da corrente de blocos de gerenciamento emite uma transação de instrução com o comando de instrução. O contrato coloca a transação de instrução em uma fila de transações pendentes de instrução. O código notifica o módulo NFV-MANO, que executa a instrução pendente e envia a saída do comando para o servidor da corrente de blocos de gerenciamento. O módulo NFV-MANO emite uma transação de resposta que inclui um campo contendo o identificador da transação de instrução correspondente. Isso fornece a rastreabilidade de cada transação executada na corrente de blocos e, portanto, possibilita a responsabilização de entidades maliciosas.

O segundo contrato inteligente, descrito na Lista 3.2, define e atualiza a configuração de uma VNF. Um cliente emite uma transação para a corrente de blocos conectada a cada VNF em uma fatia de rede. A transação contém um texto descritivo com a configuração associada no campo de descrição, assim como os dados de configuração no campo de configuração.

O protótipo usa os certificados de autoridade (*Certificate Authorities* - CA) do Hyperledger Fabric para criar e gerenciar certificados digitais em cada nó da rede de corrente de blocos. Certificados digitais garantem auditabilidade e que somente nós certificados e autorizados podem participar da rede de corrente de blocos.

Este parágrafo inicia um roteiro a ser executado pelos participantes desta aula prática realizarem a atividade prática descrita. A atividade prática foi desenvolvida utilizando a versão 1.4.1 do Hyperledger Fabric. Os comandos de versões anteriores ou posteriores podem ser incompatíveis com a versão utilizada nesta atividade. O roteiro

```
struct configurationTransaction
{
    configurationIdentifier string
    versionIdentifier       string
    description             string
    configuration           string
    transactionType         string
    transactionName         string
    issuer                  string
}
initConfiguration (configuration <description,configuration,name,issuer
    >){
    if configuration is not unique or configuration is not well-formed:
        return error
    putState (configuration.name, configuration)
    return success
}
```

Lista 3.2: Pseudocódigo parcial para emitir transações de configuração. O campo configurationIdentifier contem um identificador único para a configuração.

deve ser seguido utilizando o usuário *root*. Primeiramente, o participante deve criar o diretório `contract` dentro do diretório `fabric-samples/chaincode` e baixar o contrato inteligente que será utilizado ao longo desta atividade. De dentro do diretório `fabric-samples`, o participante deve executar:

```
cd chaincode && mkdir contract && cd contract

wget https://github.com/gta-ufrj/hpsr-smart-contracts/raw/
master/contract.go
```

O próximo passo é gerar todo o material criptográfico a partir do arquivo `crypto-config.yaml` e da ferramenta `cryptogen`. O arquivo `crypto-config.yaml` contém as informações referentes aos participantes iniciais da rede. As configurações presentes nesse arquivo informam os endereços de cada nó da rede, o número de ordenadores, as organizações participantes da rede e a quantidade de participantes por organização. Esses dados são utilizados pela ferramenta `cryptogen` para gerar pares de chaves assimétricas e certificados para os nós, que serão utilizados no processo de validação, controle de acesso e não-repúdio às transações realizadas por um membro da rede. De dentro do diretório `fabric-samples/first-network`, o usuário deve gerar os certificados que serão usados na rede utilizando a ferramenta `cryptogen` que cria um diretório chamado `crypto-config` contendo os certificados e chaves da rede:

```
../bin/cryptogen generate --config=<caminho do arquivo de
configuração>
```

- caminho do arquivo de configuração: ./crypto-config.yaml

Em seguida, o usuário deve configurar a rede do Hyperledger Fabric utilizando o arquivo `configtx.yaml` e a ferramenta `configtxgen`. Nesse arquivo estão contidas

informações sensíveis à rede como o tipo de consenso utilizado, tempo máximo para a criação de um novo bloco, tamanho máximo das transações contidas em um bloco, tamanho máximo em *bytes* do bloco, políticas para validação de transações, e perfil de cada nó. A ferramenta `configtxgen` utiliza o arquivo `configtx.yaml` para criar o bloco gênesis que contém as informações de configuração da rede. Portanto para configurar a rede, o usuário deve indicar o diretório do arquivo de configuração `configtx.yaml` para a ferramenta `configtxgen`. Feito isso, utilize a ferramenta `configtxgen` para gerar o bloco gênesis no diretório `channel-artifacts`. Esses dois passos são realizados da seguinte forma:

```
export FABRIC_CFG_PATH=$PWD
```

```
../bin/configtxgen -profile <perfil do canal>
-channelID <nome do canal> -outputBlock
./channel-artifacts/genesis.block
```

- perfil do canal: SampleDevModeKafka
- nome do canal: byfn-sys-channel

Para a criação do canal, o seguinte comando deve ser executado:

```
export CHANNEL_NAME=<nome do canal> &&
../bin/configtxgen -profile <perfil do canal>
-outputCreateChannelTx ./channel-artifacts/channel.tx
-channelID $CHANNEL_NAME
```

- perfil do canal: TwoOrgsChannel
- nome do canal: jaichannel[28]

O comando cria um arquivo chamado `channel.tx` contendo as configurações do canal dentro do diretório `channel-artifacts`.

Após a criação do arquivo com as configurações do canal, o usuário deve definir os pares-âncora (*anchor peers*). Pares-âncora conectam uma organização a outra. Pares de uma organização comunicam com os pares-âncora para descobrir pares de outras organizações. Cada organização possui ao menos um par-âncora. O comando para a definição do par-âncora para as organizações configuradas deve ser executado para cada organização listadas nos arquivos de configuração:

```
../bin/configtxgen -profile <perfil do canal>
-outputAnchorPeersUpdate ./channel-artifacts/<nome da
organização>anchors.tx -channelID $CHANNEL_NAME -asOrg
<nome da organização>
```

- perfil do canal: TwoOrgsChannel

[28]O nome do canal deve conter apenas letras minúsculas.

- nome da organização: Org1MSP

Como o exemplo desta atividade usa duas organizações:

```
../bin/configtxgen -profile <perfil do canal>
-outputAnchorPeersUpdate ./channel-artifacts/<nome da
organização>anchors.tx -channelID $CHANNEL_NAME -asOrg
<nome da organização>
```

- perfil do canal: TwoOrgsChannel
- nome da organização: Org2MSP

A rede de corrente de blocos do Hyperledger Fabric usa contêineres como nós. O usuário deve usar um arquivo de configuração em conjunto com a ferramenta *Docker Compose* para criar a rede e seus participantes. A plataforma Fabric disponibiliza os arquivos de configuração `docker-compose-cli.yaml` e `docker-compose-kafka.yaml` para facilitar a configuração e instanciação de contêineres. Para instanciar os contêineres, execute o seguinte comando:

```
docker-compose -f docker-compose-cli.yaml -f
docker-compose-kafka.yaml up -d
```

Para acessar o contêiner cliente:

```
docker exec -it cli bash
```

Agora, o usuário deve passar as configurações do canal criada anteriormente para criar o canal do Hyperledger Fabric:

```
export CHANNEL_NAME=jaichannel
```

```
peer channel create -o <nome do ordenador>:<porta do
ordenador> -c $CHANNEL_NAME -f ./channel-artifacts/
channel.tx --tls --cafile <caminho do certificado>
```

- nome do ordenador: orderer0.example.com
- porta do ordenador: 7050
- caminho do certificado: /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto /ordererOrganizations/example.com/orderers/orderer0.example.com /msp/tlscacerts/tlsca.example.com-cert.pem

Para inicializar um par no canal criado, os usuários devem executar os seguintes comando:

```
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/
hyperledger/fabric/peer/crypto/peerOrganizations/
org1.example.com/users/Admin@org1.example.com/msp
```

```
export CORE_PEER_ADDRESS=peer0.org1.example.com:7051
```

```
export CORE_PEER_LOCALMSPID="Org1MSP"

export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/
github.com/hyperledger/fabric/peer/crypto/peerOrganizations
/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

peer channel join -b jaichannel.block
```

Os participantes devem repetir os cinco comandos acima mudando o par e a organização. Como o exemplo desta atividade usa quatro pares, os comandos devem ser modificados para atender aos peer0 e peer1 da organização Org1 e aos peer0 e peer1 da organização Org2.

Com todos os pares adicionados, é necessário atualizar o canal para escolher os pares-âncoras de cada organização. Uma atualização no canal do Hyperledger Fabric adiciona uma informação de configuração do canal. Neste exemplo, os pares-âncoras são o peer0 da organização Org1 e peer0 da organização Org2.

```
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/
hyperledger/fabric/peer/crypto/peerOrganizations/
org1.example.com/users/Admin@org1.example.com/msp

export CORE_PEER_ADDRESS=peer0.org1.example.com:7051

export CORE_PEER_LOCALMSPID="Org1MSP"

export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/
github.com/hyperledger/fabric/peer/crypto/peerOrganizations
/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

peer channel update -o <ordenador> -c
$CHANNEL_NAME -f ./channel-artifacts/<org name>anchors.tx
--tls --cafile <caminho do certificado>
```

- ordenador: orderer0.example.com:7050
- caminho do certificado: /opt/gopath/src/github.com/hyperledger/fabric/peer /crypto/ordererOrganizations/example.com/orderers/orderer0.example.com /msp/tlscacerts/tlsca.example.com-cert.pem

Os participantes devem repetir os cinco comandos acima para atender ao peer0 da organização Org2.

Agora, é necessário instalar o contrato em cada par que executa e apoia as transações.

```
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/
hyperledger/fabric/peer/crypto/peerOrganizations/
org1.example.com/users/Admin@org1.example.com/msp

export CORE_PEER_ADDRESS=peer0.org1.example.com:7051

export CORE_PEER_LOCALMSPID="Org1MSP"
```

```
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/
github.com/hyperledger/fabric/peer/crypto/peerOrganizations
/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

go get -u github.com/golang-collections/go-datastructures
/queue && peer chaincode install -n <nome do chaincode>
-v 1.0 -p <diretório do chaincode>
```

- nome do chaincode: mycc
- diretório raiz do chaincode: github.com/chaincode/contract

Os participantes devem repetir os cinco comandos acima mudando o par e a organização. Como o exemplo desta atividade usa quatro pares, os comandos devem ser modificados para atender aos peer0 e peer1 da organização Org1 e aos peer0 e peer1 da organização Org2.

Em seguida, o participante deve instanciar o *chaincode* no canal e também definir a política de endosso (*endorsement*) do canal:

```
peer chaincode instantiate -o <ordenador>
--tls --cafile <caminho do certificado> -C $CHANNEL_NAME
-n mycc -v 1.0 -c <mensagem em JSON para o chaincode>
-P <política do canal>
```

- ordenador: orderer0.example.com:7050
- caminho do certificado: /opt/gopath/src/github.com/hyperledger/fabric/peer /crypto/ordererOrganizations/example.com/orderers/orderer0.example.com /msp/tlscacerts/tlsca.example.com-cert.pem
- mensagem em JSON para o chaincode: ‘{"Args":["init"]}’
- política do canal: "AND (‘Org1MSP.peer’,‘Org2MSP.peer’)"

Neste caso, a política descrita acima define que uma transação efetuada no canal deve ser endossada por um par pertencente à organização Org1 e por um par pertencente à organização Org2.

Para gerar uma transação, o Fabric disponibiliza o comando `invoke`, que chama uma função do contrato instanciado no canal. O comando recebe como argumento o endereço e o caminho dos certificados dos pares, o serviço de ordenação, o nome do canal, o nome do contrato instanciado e a mensagem para o contrato em formato JSON. Como exemplo, o comando abaixo emite uma transação na rede.

```
peer chaincode invoke -o <ordenador>
--tls --cafile <caminho do certificado> -C $CHANNEL_NAME
-n mycc --peerAddresses <endereço do par de org1>
--tlsRootCertFiles <caminho do certificado do par>
--peerAddresses <endereço do par de org2>
```

```
--tlsRootCertFiles <caminho do certificado do par>
-c <mensagem em JSON para o chaincode>
```

- ordenador: orderer0.example.com:7050
- caminho do certificado: /opt/gopath/src/github.com/hyperledger/fabric/peer /crypto/ordererOrganizations/example.com/orderers/orderer0.example.com /msp/tlscacerts/tlsca.example.com-cert.pem
- endereço do par de org1: peer0.org1.example.com:7051
- caminho do certificado do par: /opt/gopath/src/github.com/hyperledger/fabric/peer /crypto/peerOrganizations/org1.example.com/peers /peer0.org1.example.com/tls/ca.crt
- endereço do par de org2: peer0.org2.example.com:7051
- caminho do certificado do par: /opt/gopath/src/github.com/hyperledger/fabric /peer/crypto/peerOrganizations/org2.example.com/peers /peer0.org2.example.com/tls/ca.crt
- mensagem em JSON para o chaincode: ‘{"Args":["initInstructionTransaction", "transaction","issuer","instruction"]}’

O Fabric permite serviços de busca e pesquisa de histórico de transações através do comando `peer chaincode query`. O comando recebe como argumento o nome do canal, o nome do contrato instanciado e a mensagem para o contrato em formato JSON. Um exemplo:

```
peer chaincode query -C $CHANNEL_NAME -n mycc -c
<chaincode JSON message>
```

- mensagem em JSON para o chaincode: ‘{"Args":["getHistoryForTransaction" , "transaction"]}’

3.5. Considerações Finais, Perspectivas e Problemas em Aberto

A tecnologia de corrente de blocos é muito recente, pois tem apenas dez anos. O fato desta tecnologia prover uma camada de confiança distribuída faz com que ela seja considerada disruptiva e também a tecnologia mais importante depois da Internet. As criptomoedas já são um sucesso e existem predições que afirmam que continuará a crescer fortemente o volume de "dinheiro virtual"registrado em correntes de blocos. Esta seção aborda os principais desafios e problemas em aberto das correntes de blocos assim como as oportunidades de pesquisa na área.

3.5.1. Oportunidades e Aplicações da Tecnologia de Corrente de Blocos

Além da transferência de ativos utilizando criptomoedas, a tecnologia de corrente de blocos pode ser aplicada em diversas áreas. Praticamente em todas aplicações que requerem intermediários a corrente de blocos pode eliminar este intermediário provendo a camada de confiança distribuída. Também para fazer registros permanentes e imutáveis.

Papel da Corrente de Blocos em uma Economia Compartilhada - A economia do compartilhamento, ou compartilhada, é um modo de consumo onde bens e serviços não são de posse de um único usuário. Estes bens e serviços são temporariamente disponibilizados por outros indivíduos (terceiros), que recebem um incentivo financeiro para oferecer o serviço, através de uma plataforma que atua como intermediário entre o provedor do serviço e o consumidor. Os principais exemplos de empresas que utilizam este modelo são a Uber e a Airbnb, com outros exemplos incluindo: Cohealo, BlaBlaCar, JustPark, Skillshare, RelayRides e Landshare. Estas empresas conseguem seu lucro tomando uma parte dos ganhos dos usuários que provêm o serviço, ao fornecer uma plataforma capaz de conectar pessoas com interesses coincidentes.

A tecnologia de corrente de blocos permite prover confiança entre o provedor de serviço e o usuário do serviço sem necessidade de uma empresa intermediária. As empresas centralizadoras são assim eliminadas [Hawlitschek et al. 2018]. Além disso, contratos inteligentes permitem definir e impor regras para um acordo entre duas partes, possibilitando a fácil responsabilização de qualquer partícipe caso ocorra uma violação desse contrato. Plataformas de economia compartilhada implementadas com corrente de blocos são desse modo administradas coletivamente por todos seus usuários. Como é do interesse dos membros da plataforma que esta continue funcionando, há um incentivo para que os participantes atuem com honestidade.

Um exemplo de aplicação da corrente de blocos na economia compartilhada é a plataforma descentralizada de carona solidária Arcade City, que propõe um serviço similar ao Uber. Todo o processamento necessário para tratar do compartilhamento de caronas e outras funções é executado pela corrente de blocos. Como não existem intermediários, os custos das caronas podem ser reduzidos significativamente. Segurança é garantida usando um sistema de classificação no próprio aplicativo, garantindo que avaliações feitas por usuários encontrem-se sempre disponíveis e inalteradas graças a características da corrente de blocos.

Corrente de Blocos para Cidades Inteligentes - O uso de corrente de blocos em sistemas de Cidades Inteligentes possui restrições de segurança e privacidade que ainda são desafios [Khatoun and Zeadally 2016]. As correntes de dados públicas, como o Bitcoin, efetuam transações anônimas identificadas por chaves públicas. No entanto, a anonimidade não é absoluta, pois todas as transações são visíveis para todos os participantes da corrente de blocos e, assim, as atividades do usuário podem ser rastreadas. Combinando as informações dessas transações com alguns dados externos permite revelar a identidade real do usuário. Uma vez que a identidade do usuário é descoberta, todas suas ações serão rastreadas e dados confidencias, como padrões de compra e venda, serão vazados. Portanto, estes sistemas não garantem a privacidade dos dados dos usuários o que viola um dos princípios da segurança da informação [Talari et al. 2017].

O Uso de Corrente de Blocos na Área de Saúde - A alta produção e transferência de dados na área da saúde podem ser beneficiadas pelo uso de corrente de blocos. Sistemas baseados em corrente de blocos permite que diferentes serviços de saúde possam compartilhar e armazenar dados e registros médicos em uma rede permissionada [Azaria et al. 2016]. Enquanto o uso de contratos inteligentes permite que pacientes controlem o acesso e a transferência de seus dados privados, a cópia da corrente de blocos em diferentes locais e a auditabilidade atendem à demanda de acesso de diferentes interessados ao mesmo documento.

A corrente de blocos também possui aplicações na rastreabilidade de dados na área da saúde. Sistemas baseados em corrente de blocos permitem o estabelecimento de uma rede privada e segura entre serviços de saúde para armazenar prescrições de remédios [Zhang et al. 2018]. As mudanças no estado ou posição do medicamento, assim como a transferência de propriedade são registradas no livro-razão distribuído, formando um registro histórico de dados do remédio desde a origem. A auditabilidade permite o acesso das entidades credenciadas às informações, facilitando a identificação de medicamentos falsos ou desviados.

3.5.2. Atividades em Organismos de Normalização

A aplicabilidade da tecnologia de corrente de blocos tem se demonstrado enorme. Assim, é essencial o pronto estabelecimento de normas internacionais para prover diretrizes relacionadas à uma nova tecnologia que propiciem um maior envolvimento e mais investimentos das indústrias. Em relação aos tópicos abordados neste capítulo, diferentes organismos de normalização criaram grupos de trabalho com objetivos variados.

A organização internacional de normalização (*International Organization for Standartization* - ISO) criou um comitê técnico que visa normalizar a tecnologia de corrente de blocos e a tecnologia de livro-razão distribuído [ISO/TC 307 2016]. O comitê possui projetos para formalizar os riscos de segurança, ameaças e vulnerabilidades da tecnologia, além de normalizar a arquitetura de referência, taxonomia, contratos inteligentes e a proteção de privacidade e de informações pessoais. A *Internet Research Task Force* (IRTF) criou o grupo de pesquisa da infraestrutura descentralizada da Internet (*Decentralized Internet Infrastructure Research Group* - DINRG) que estuda os serviços de infraestrutura beneficiados pela descentralização [IRTF 2017]. A *World Wide Web Consortium* (W3C) criou o grupo da comunidade de corrente de blocos (*Blockchain Community Group*) que tem como objetivo normalizar os formatos das mensagens em sistemas baseados em corrente de blocos [W3C 2016]. A comunidade usa o formato de mensagem definido na ISO 20022 como base para as normas. O grupo também busca definir diretrizes para o uso de armazenamento. A união internacional de telecomunicações (*International Telecommunications Union* - ITU), através do grupo de foco na aplicação da tecnologia de livro-razão distribuído (*Focus Group on Applications of Distributed Ledger Technology* - ITU-T FG DLT), pretende normalizar os serviços interoperáveis baseados na tecnologia de livro-razão distribuído [ITU-T FG DLT 2017]. O grupo de trabalho de corrente de blocos da Europa (*Europe Blockchain Working Group*) da associação internacional de segurança para comunicação institucional do comércio (*International Securities Association Trade Communication* - ISITC) discute a adoção da tecnologia de livro-razão distribuído pela indústria de serviços financeiros [Radford 2016].

3.5.3. Desafios e Problemas em Aberto

Os desafios em corrente de blocos são enormes: escalabilidade, vazão de transações, segurança e *software*, algoritmos criptográficos eficientes, entre outros.

Escalabilidade e Vazão de Transações - Todos os protocolos de consenso e tipos de corrente de blocos apresentados possuem limitações em termos de taxa de transferência, latência de transação ou quantidade de nós [Popov 2017]. Os sistemas baseados em correntes de blocos ainda são incapazes de atingir o desempenho dos atuais sistemas centralizados de transferência de ativos. Enquanto as plataformas do PayPal e Visa processam aproximadamente 2000 transações por segundo em média e até 56.000 transações em pico [Visa 2019, PayPal 2019] com tempos de resposta da ordem de segundos [BitcoinWiki 2019], o Ethereum processa um máximo de 20 transações por segundo e o Bitcoin atinge uma vazão de apenas 7 transações por segundo. Ainda, o processo de mineração da prova de trabalho no Bitcoin gera gastos insustentáveis de energia sem retorno proporcional, atingindo em média 50 TW consumidos por ano [Digiconomist 2019]. Algumas correntes de blocos permissionadas atingem taxas da ordem de milhares de transações por segundo, porém reduzem o número de participantes possíveis [Schwartz et al. 2014, Buchman 2016, Kiayias et al. 2017].

Outro desafio de correntes de blocos é a eficiência no armazenamento e na busca de transações, pois, para prover segurança descentralizada, é necessário que todos os participantes do consenso processem todas as transações da rede e armazenem todo o histórico de transações. A verificação de transações é fundamental para garantir segurança em um ambiente sem confiança entre os pares como o de corrente de blocos. No entanto, verificar a existência ou corretude de uma transação pode envolver uma busca custosa em uma lista crescente de todas as transações presentes em todos os blocos da rede. Além disso, a quantidade de dados armazenados em cada participante da corrente de blocos cresce constantemente, uma vez que nenhuma transação ou bloco jamais é descartado. As principais implementações de correntes de blocos implementam estruturas auxiliares chamadas de árvores de Merkle para contornar o problema[29]. O trilema entre descentralização, segurança e escalabilidade dificulta que os sistemas baseados em correntes de blocos atinjam o desempenho de sistemas atuais e que atendam às necessidades do futuro.

Revogação de Chaves criptográficas - Um dos principais desafios de sistemas baseados em correntes de blocos é a perda de chaves privadas, pois todas as transferências de ativos na rede são realizadas através de transações assinadas. A ausência de uma autoridade central confiável que armazene as identidades dos participantes da rede dificulta a revogação de chaves perdidas ou roubadas e consiste em um risco constante de perda de ativos. Em especial, é impossível revogar uma chave perdida em correntes de blocos públicas, nas quais o único identificador de um usuário é um endereço baseado na sua chave pública correspondente. Estima-se que mais de 30% dos *bitcoins* minerados estão inutilizados devido à perda de chaves, totalizando uma perda média de mais de 1000 *bitcoins* ou 8 milhões de dólares por dia [Chainalysis 2019].

Gerenciadores de chaves de correntes de blocos públicas implementam mecanismos de múltiplas assinaturas (*multisignature* ou *multisig*) para prevenir a perda de ativos

[29] A estrutura e fundamentos de árvores de Merkle são apresentados no Apêndice B.

decorrente da perda de uma chave privada. O mecanismo *multisignature* cria 3 pares de chaves assimétricas e exige a assinatura de pelo menos duas para emitir uma transação. As chaves podem ser armazenadas em locais diferentes e, possivelmente, controladas por entidades diferentes. Caso uma chave seja perdida, é possível assinar com as duas restantes. Nenhuma chave pode emitir uma transação individualmente. O mecanismo permite tolerar perdas de até uma chave ou repartir a posse de um ativo, pois, caso as chaves pertençam a entidades diferentes, pelo menos duas entidades devem concordar para emitir uma transação. Caso uma chave seja perdida no mecanismo de assinaturas múltiplas, basta criar um novo endereço e emitir uma transação assinada transferindo o ativo com as duas chaves restantes. Sistemas baseados em correntes de blocos permissionadas podem implementar mecanismos baseados em identidades pessoais para criar listas locais de revogação de chaves que devem ser alteradas através de consenso [Androulaki et al. 2018].

Segurança da Corrente de Blocos - Falhas de segurança em *software* de corrente de blocos podem causar um desastre. Existe uma enorme carência de profissionais que programem de forma segura e de ferramentas que consigam analisar, validar e testar a segurança de um *software* de corrente de blocos. Existem mais de 700 criptomoedas, no entanto, não existe nenhuma garantia que os softwares usando nas criptomoedas é seguro. Até mesmo, a maioria das implementações dos algoritmos de consenso não passaram por algum tipo de prova. Uma falha em um contrato inteligente pode ser catastrófica. Um exemplo foi a falha da chamada recursiva (*recursive call bug*) no acesso ao objeto (*data access object* - DAO) que causou o roubo de mais de 50 milhões de dólares no Ethereum[30].

No consenso com prova de trabalho (*Proof of Work* – PoW) usado no Bitcoin e Ethereum existe o conhecido ataque de 51% ou ataque de maioria, que se refere quando um atacante ou grupo de atacantes possui mais de 50% do poder computacional da rede, porque, neste caso, os atacantes podem fazer gasto duplo. Embora um ataque de 51% nunca tenha sido executado com sucesso no Bitcoin[31], no entanto, ele foi demonstrado que funciona em moedas alternativas do bitcoin (*altcoin*)[32] [33].

Ainda em relação ao mecanismo de consenso por prova de trabalho, Eyal and Sirer da Universidade de Cornell apresentaram em 2013 o ataque de mineração egoísta (*selfish mining*) [Eyal and Sirer 2013]. Este ataque se aplica ao consenso de prova de trabalho quando mineradores em conluio procuram aumentar seus lucros para receberem mais incentivos. A ideia chave deste ataque é que um conjunto de mineradores em conluio achem o resultado do desafio para formar um bloco, mas mantém a o resultado em segredo, forçando uma bifurcação de forma intencional. O conjunto de mineradores em conluio continuam minerando o ramo da bifurcação que eles forçaram sem divulgar novos blocos, enquanto mineradores honestos vão minerar o outro ramo. Caso os mineradores

[30]https://medium.com/@MyPaoG/explaining-the-dao-exploit-for-beginners-in-solidity-80ee84f0d470

[31]O conjunto de mineradores "Ghash.io"ultrapassou 50% do poder computacional da rede bitcoin em julho de 2014. Este conjunto de mineradores se comprometeu a reduzir o poder computacional para um valor sempre menor que 40%.

[32]A moeda Bitcoin Gold, na época a 26^a maior moeda, sofreu um ataque de 51% em maio de 2018. Os atacantes efetuaram gasto duplo por diversos dias e roubaram mais de US$18 milhões em Bitcoin Gold.

[33]As correntes de blocos Krypton e Shift, baseadas em Ethereum, sofreram ataques de 51% em agosto de 2016.

em conluio, mais de 25% da capacidade total, resolvam mais desafios, e não divulguem o resultado, eles passam a ter mais vantagem para conseguir obter o ramo mais longo. Quando o ramo minerado de forma pública pelos mineradores honestos se aproxima em tamanho do ramo minerado pelos malfeitores em conluio, os mineradores egoístas divulgam os blocos. Os mineradores honestos perderão dinheiro porque o ramo que eles mineraram não vai vingar.

Assiste-se hoje uma colaboração das indústrias sem precedentes mesmo entre empresas que eram concorrentes. O interesse na tecnologia de corrente de blocos cresce exponencialmente e o profissionais especialistas desta área estão muito requisitados e estão entre os mais bem pagos. Muitos desafios ainda persistem e muitos avanços são esperados para os próximos anos. Com certeza é uma área com enormes perspectivas e oportunidades.

A corrente de blocos pode resolver problemas em diversos setores da sociedade como economia, saúde, transporte, educação, entre outros. A tecnologia de corrente de blocos é uma poderosa ferramenta que oferece muitas possibilidades de atacar problemas de injustiça, desigualdade, carência, entre outras. No entanto, as soluções para problemas humanos na maioria das vezes não é técnica, mas sim humanas. A corrente de blocos não corrige seres humanos.

A. Criptografia Assimétrica, Assinatura e Função *Hash*

Este apêndice apresenta alguns conceitos básicos de criptografia que são usados pelas correntes de blocos.

Criptografia Assimétrica - A criptografia assimétrica consiste em um sistema criptográfico baseado em um par de chaves assimétricas: uma chave pública e uma chave privada. Enquanto a chave pública pode ser amplamente disseminada, a chave privada deve ser mantida em segredo. Assim, a chave pública é usada para criptografar uma mensagem que apenas quem possui a chave privada, par desta chave pública, é capaz de descriptografar. A criptografia RSA (Rivest-Shamir-Adleman) e a criptografia de curvas elípticas são as principais tecnologias de chaves assimétricas. As duas tecnologias podem prover a segurança necessária a qualquer sistema. No entanto, para um mesmo nível de segurança, mas as curvas elípticas requerem chaves de menor tamanho.

Assinatura Digital - A operação de assinatura criptográfica é quando a originador de uma mensagem a encripta com sua chave privada. Apenas a chave pública, par desta chave privada, consegue decriptar a mensagem. Assim, ao decriptar uma mensagem tem-se a garantia da autenticidade da origem da mensagem uma vez que a chave privada pertence apenas a originador da mensagem. O mecanismo de assinatura digital deve prover as propriedades básicas de autenticidade, não repúdio e integridade. A propriedade de autenticidade deve permitir a confirmação da autenticidade de uma mensagem, ou seja, que somente o signatário deve ser capaz de gerar sua assinatura digital para aquela mensagem. O não repúdio deve garantir que o signatário de uma mensagem assinada digitalmente não possa negar a autoria da assinatura. Por fim, é necessário a garantia de integridade: para sustentar as características anteriores de autenticidade e de não repúdio, a mensagem não pode ser adulterada.

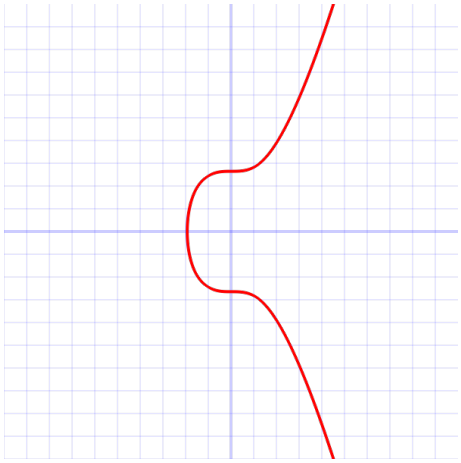


Figura 3.18: Curva Elíptica secp256k1 usada na moeda eletrônica Bitcoin.

A utilização de curvas elípticas em criptografia assimétrica foi sugerida por Neal Koblitz e Victor S. Miller em 1985. Uma curva elíptica é o locus dos pontos do plano cujas coordenadas satisfazem a equação cúbica $y^2 = x^3 + ax + b$ junto com um ponto na infinidade O. A Figura 3.18 ilustra a curva elíptica "secp256k1", usada na criptografia assimétrica do Bitcoin e definida nas normas para criptografia eficiente (*Standards for Efficient.00Cryptography* - SEC)[34].

O algoritmo de assinatura digital por curvas elípticas (*Elliptic Curve Digital Signature Algorithm - ECDSA* é uma variante do tradicional algoritmo de assinatura digital (*Digital Signature Algorithm* - DSA). O algoritmo baseado em curvas elípticas implementa uma modificação da norma de assinatura digital (*Digital Signature Standard* - DSS) que executa operações sobre pontos de curvas elípticas, ao invés das operações de exponenciação usadas no DSS. A maior vantagem do ECDSA sobre o DSA é que o ECDSA requer tamanhos bem menores de chave para propiciar a mesma segurança[35].

Função Resumo (*Hash*) - Uma função resumo, ou função *hash*, é uma função que mapeia dados de comprimento variável em dados de comprimento fixo. Uma função *hash* criptográfica é uma transformação matemática que, a partir de uma mensagem de comprimento arbitrário ou uma sequência de bits de tamanho arbitrário, obtém uma sequência curta e com tamanho fixo de bits. A função *hash* é descrita por $H(m) = h$, onde m é a mensagem em claro, $H()$ é a função *hash* e h é o valor *hash* resultante. A função *hash* deve ter as seguintes propriedades:

- a função *hash*, $H(m)$, pode ser aplicada a uma mensagem, m, de qualquer tamanho;
- a função *hash*, $H(m)$, gera uma saída h de tamanho fixo (por exemplo, o tamanho é 256 bits se $H(m)$ for o algoritmo SHA256);
- a função *hash* é simples, ou seja, é computacionalmente eficiente calcular $H(m)$

[34]Certicom Research, http://www.secg.org/sec2-v2.pdf

[35]Uma chave de criptografia de curvas elípticas (*Eliptic Curve Criptography* -ECC) de 163 bits corresponde à mesma garantia de segurança de uma chave RSA (*Rivest-Shamir-Adleman*) de 3.072 bits e uma chave de criptografia simétrica AES (*Advanced Encription Standard*) de 128 bits.

para qualquer m (a computação de $H(m)$ utiliza operações lógicas e evita as multiplicações e exponenciações usadas em criptografia assimétrica);

- a função *hash* é unidirecional ou não-invertível, isto é, para um h qualquer é computacionalmente impossível achar m tal que $H(m) = h$;

- a função *hash* é livre de colisões, ou seja, para uma mensagem, m, é computacionalmente impossível achar uma mensagem diferente, m', tal que $H(m) = H(m')$. Deve ser ressaltado que embora possam existir muitas mensagens que geram o mesmo *hash*, a função *hash* criptográfica deve ser projetada para dificultar ao máximo a colisão de *hashes*;

- é computacionalmente impossível achar um par de mensagens (m, m') tal que $H(m) = H(m')$.

A aplicação principal da função *hash* é a garantia de integridade de uma mensagem. Em geral, o emissor de uma mensagem calcula o seu *hash* e o insere no final da mensagem. O receptor recalcula o *hash* da mensagem recebida e compara o resultado obtido com o *hash* recebido. Se os *hashes* forem diferentes, há garantia de que a mensagem recebida é diferente da mensagem enviada. Se os *hashes* forem iguais, há uma probabilidade muito alta da mensagem recebida ser igual a mensagem enviada. Assim, qualquer alteração na mensagem pode ser detectada pelo recálculo e comparação do *hash*, o que garante a integridade da mensagem.

Existem diferentes funções *hash*, mas o Bitcoin e maioria das moedas criptográficas usam a função *hash* SHA256, que resulta em 256 bits. O algoritmo seguro de *hash* (*Secure Hash Algorithm* - SHA) foi desenvolvido pela Agência de Segurança Nacional (*National Security Agency* - NSA) dos EUA.

Assinatura da Mensagem e Assinatura do *Hash* da Mensagem - Uma mensagem criptografada por uma chave privada só pode ser descriptografada pela chave pública correspondente à chave privada que criptografou a mensagem. Esta operação garante o sigilo da mensagem, uma vez que apenas o destino pode descriptografar seu conteúdo, e a autenticidade do emissor, pois só ele possui a chave privada que foi usada na criptografia, e qualquer um que possua sua chave pública correspondente pode verificar a mensagem. Porém, a mensagem pode ter sido adulterada no caminho. Por outro lado, se o emissor computa o *hash* da mensagem, criptografa o *hash* com sua chave privada e envia o *hash* criptografado para o destinatário junto da mensagem, esta operação garante tanto a integridade da mensagem quanto autenticidade do emissor. A mensagem vai em claro e, portanto, não garante o sigilo, mas o destinatário garante ao mesmo tempo que a mensagem chegou íntegra e que o emissor é autêntico. Para isso, basta recalcular o *hash* a partir da mensagem, descriptografar o resultado com a chave pública do emissor e compará-lo com o *hash* assinado contido na mensagem.Este procedimento é muito usado quando não se requer o sigilo, pois o tempo de processamento para encriptar o *hash* é muito menor do que o necessário para encriptar uma mensagem inteira.

B. Árvores de Merkle e Verificação Simples de Pagamento

As árvores de Merkle são a principal estrutura auxiliar utilizada em correntes de blocos para verificar a integridade e não-repúdio de uma transação de maneira eficiente [Nakamoto 2008]. Nomeadas em homenagem a Ralph Merkle, que patenteou o conceito em 1979, as árvores de Merkle são estruturas de dados em forma de árvore na qual cada nó não-folha, denominado de "nó-galho,"é o resultado de um *hash* de seus respectivos nós filhos. Cada nó-folha da árvore é o resultado de um *hash* de um conjunto de dados que, no caso da corrente de blocos, representam as transações da rede. A Figura 3.19 ilustra a estrutura de uma corrente de blocos que utiliza uma árvore de Merkle binária para armazenar transações. A estrutura da árvore pode ser construída calculando o *hash* de cada transação T para criar o nível mais baixo da árvore, e, posteriormente, utilizando os *hashes* de cada nível para construir o próximo nível, até chegar à raiz da árvore. A complexidade de construção de uma árvore de Merkle de um bloco é $O(n.log_2(n)$ onde n é o número de transações contidas no bloco. A árvore binária é o tipo de árvore de Merkle mais utilizado em correntes de blocos, porém algumas implementações utilizam tipos diferentes de árvore [Wood 2014].

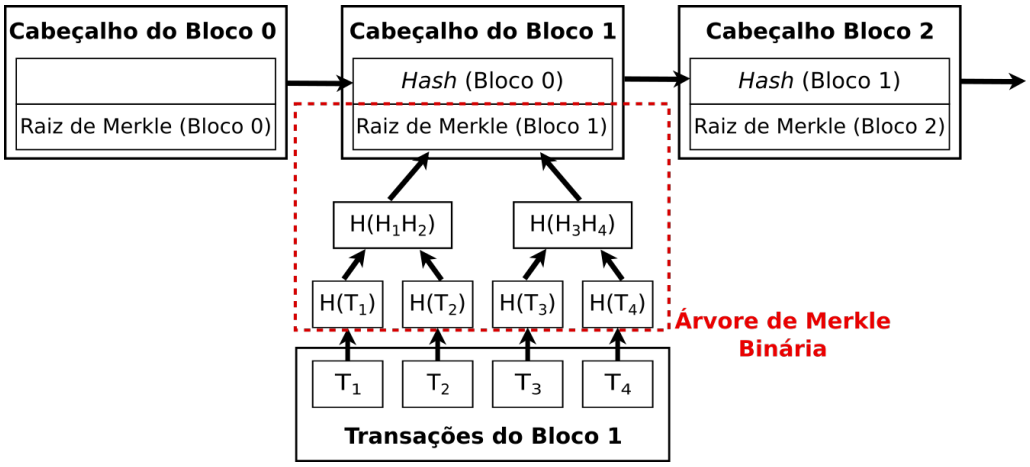


Figura 3.19: Estrutura de uma árvore de Merkle binária utilizada em correntes de blocos. O uso de árvores de Merkle permite armazenar apenas a raiz da árvore sem prejuízo à verificação das transações.

O uso de árvores Merkle permite obter uma prova de Merkle (*Merkle proof*), que verifica a presença e a corretude de uma transação em um bloco de maneira eficiente. Para verificar uma transação, basta fornecer os *hashes* necessários para reconstruir o caminho da árvore correspondente à transação verificada. A reconstrução de um caminho e, logo, a verificação de uma transação, tem complexidade $O(n)$. Assim, é possível verificar transações rapidamente mesmo em meio a milhares de transações, e não é necessário armazenar o bloco inteiro para verificar uma transação. Além disso, o uso de *hashes* diminui o tráfego na rede, pois não é necessário transferir a transação ou bloco completos. Isto permite criar "nós leves"e um mecanismo simplificado de verificação, pois, para verificar uma transação, basta armazenar o cabeçalho do bloco e solicitar a nós que possuem todas as transações os *hashes* do caminho até a transação. As provas de Merkle são a base do mecanismo de verificação simples de pagamento (*Simple Payment Verification* - SPV)

proposto por Nakamoto e utilizado na maioria das implementações de corrente de blocos atuais [Nakamoto 2008, Wood 2014].

Referências

[Alchieri et al. 2008] Alchieri, E. A., Bessani, A. N., Silva Fraga, J., and Greve, F. (2008). Byzantine consensus with unknown participants. In *OPODIS '08: Proceedings of the 12th International Conference on Principles of Distributed Systems*, pages 22–40, Berlin, Heidelberg. Springer-Verlag.

[Alchieri et al. 2018] Alchieri, E. A. P., Bessani, A., Greve, F., and d. S. Fraga, J. (2018). Knowledge connectivity requirements for solving byzantine consensus with unknown participants. *IEEE Transactions on Dependable and Secure Computing*, 15(2):246–259.

[Ali et al. 2016] Ali, M., Nelson, J. C., Shea, R., and Freedman, M. J. (2016). Blockstack: A global naming and storage system secured by blockchains. In *USENIX Annual Technical Conference*, pages 181–194.

[Alvarenga et al. 2018] Alvarenga, I. D., Rebello, G. A. F., and Duarte, O. C. M. B. (2018). Securing Management, Configuration, and Migration of Virtual Network Functions Using Blockchain. In *IEEE/IFIP NOMS*.

[Androulaki et al. 2018] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. (2018). Hyperledger Fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM.

[Angelis et al. 2018] Angelis, S. D., Aniello, L., Baldoni, R., Lombardi, F., Margheri, A., and Sassone, V. (2018). PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain. In *Italian Conference on Cyber Security (06/02/18)*.

[Aspnes 2003] Aspnes, J. (2003). Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175.

[Attiya and Welch 2004] Attiya, H. and Welch, J. (2004). *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. Wiley Series on Parallel and Distributed Computing. Wiley-Interscience, 2nd edition.

[Azaria et al. 2016] Azaria, A., Ekblaw, A., Vieira, T., and Lippman, A. (2016). Medrec: Using blockchain for medical data access and permission management. In *Open and Big Data (OBD), International Conference on*, pages 25–30. IEEE.

[Back 2002] Back, A. (2002). Hashcash - a denial of service counter-measure. `http://www.hashcash.org/papers/hashcash.pdf`. Acessado em 12 de maio de 2019.

[Bano et al. 2017] Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., and Danezis, G. (2017). SoK: Consensus in the age of blockchains. Technical report, University College London, United Kingdom. https://arxiv.org/pdf/1711.03936.pdf.

[Bashir 2018] Bashir, I. (2018). *Mastering Blockchain: Distributed ledger technology, decentralization, and smart contracts explained*. Packt Publishing Ltd.

[Bessani et al. 2013] Bessani, A., Santos, M., Felix, J., Neves, N., and Correia, M. (2013). On the efficiency of durable state machine replication. In *Proc. of USENIX ATC 2013*.

[Bessani et al. 2014] Bessani, A., Sousa, J., and Alchieri, E. (2014). State machine replication for the masses with BFT-SMaRt. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.

[Bishop 2002] Bishop, M. (2002). *Computer Security: Art and Science*. Addison-Wesley.

[Biswas and Muthukkumarasamy 2016] Biswas, K. and Muthukkumarasamy, V. (2016). Securing smart cities using blockchain technology. In *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1392–1393.

[BitcoinWiki 2019] BitcoinWiki (2019). Bitcoin Scalability. `https://en.bitcoin.it/wiki/Scalability`. Acessado em 12 de maio de 2019.

[Bozic et al. 2017] Bozic, N., Pujolle, G., and Secci, S. (2017). Securing virtual machine orchestration with blockchains. In *1st Cyber Security in Networking Conference*.

[Brewer 2000] Brewer, E. A. (2000). Towards Robust Distributed Systems. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, New York, NY, USA. ACM.

[Buchman 2016] Buchman, E. (2016). *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph.

[Buterin 2014] Buterin, V. (2014). A Next Generation Smart Contract & Decentralized Application Platform. Technical report, white paper.

[Castro and Liskov 2002] Castro, M. and Liskov, B. (2002). Practical Byzantine Fault-Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems*, 20(4):398–461.

[Cavin et al. 2005] Cavin, D., Sasson, Y., and Schiper, A. (2005). Reaching agreement with unknown participants in mobile self-organized networks in spite of process crashes. Technical Report IC/2005/026, EPFL - LSR.

[Cavin et al. 2004] Cavin, D., Sasson, Y., and Schiper, A. (2004). Consensus with unknown participants or fundamental self-organization. In *Proc. of the 3rd Int. Conf. on Ad hoc Networks and Wireless - ADHOC-NOW 2004*, pages 135–148.

[Chainalysis 2019] Chainalysis (2019). Crypto Crime Report: Decoding increasingly sophisticated hacks, darknet markets, and scams. Technical report, Chainalysis Inc.

[Chandra and Toueg 1996] Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267.

[Chaum 1983] Chaum, D. (1983). Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, Hong-Ning.

[Chicarino et al. 2017] Chicarino, V. R. L., Jesus, E. F., de Albuquerque, C. V. N., and de A. Rocha, A. A. (2017). Uso de blockchain para privacidade e segurança em internet das coisas. In *Minicursos do SBSeg'2017*, pages 149–200.

[Christidis and Devetsikiotis 2016] Christidis, K. and Devetsikiotis, M. (2016). Blockchains and smart contracts for the Internet of Things. *IEEE Access*, 4:2292–2303.

[Chun et al. 2007] Chun, B.-G., Maniatis, P., Shenker, S., and Kubiatowicz, J. (2007). Attested append-only memory: Making adversaries stick to their word. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 189–204, New York, NY, USA. ACM.

[Correia et al. 2004] Correia, M., Neves, N. F., and Veríssimo, P. (2004). How to tolerate half less one Byzantine nodes in practical distributed systems. In *Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems*, pages 174–183.

[Dai 1998] Dai, W. (1998). B-money. `http://www.weidai.com/bmoney.txt`. Acessado em 12 de maio de 2019.

[Digiconomist 2019] Digiconomist (2019). Bitcoin Energy Consumption Index. `https://digiconomist.net/bitcoin-energy-consumption`. Acessado em 12 de maio de 2019.

[Dwork et al. 1988] Dwork, C., Lynch, N. A., and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of ACM*, 35(2):288–322.

[Esposito et al. 2018] Esposito, C., De Santis, A., Tortora, G., Chang, H., and Choo, K. R. (2018). Blockchain: A panacea for healthcare cloud-based data security and privacy? *IEEE Cloud Computing*, 5(1):31–37.

[Eyal et al. 2016] Eyal, I., Gencer, A. E., Sirer, E. G., and Van Renesse, R. (2016). Bitcoin-ng: A scalable blockchain protocol. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 45–59.

[Eyal and Sirer 2013] Eyal, I. and Sirer, E. G. (2013). Majority is not Enough: Bitcoin Mining is Vulnerable. Technical report, Department of Computer Science, Cornell University. https://arxiv.org/pdf/1311.0243.pdf.

[Eyal and Sirer 2018] Eyal, I. and Sirer, E. G. (2018). Majority is Not Enough: Bitcoin Mining is Vulnerable. *Commun. ACM*, 61(7):95–102.

[Finney 2005] Finney, H. (2005). RPOW: Reusable Proofs of Work. `http://nakamotoinstitute.org/finney/rpow/theory.html`. Acessado em 12 de maio de 2019.

[Fischer et al. 1985] Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382.

[Frantz and Nowostawski 2016] Frantz, C. K. and Nowostawski, M. (2016). From institutions to code: Towards automated generation of smart contracts. In *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 210–215. IEEE.

[Greve et al. 2018] Greve, F., Sampaio, L., Abijaude, J., Coutinho, A. A. R., Brito, I. V. S., and Queiroz, S. (2018). Blockchain e a Revolução do Consenso sob Demanda. In *Minicursos do SBRC'2018*, volume 36.

[Greve and Tixeuil 2010] Greve, F. and Tixeuil, S. (2010). Conditions for the Solvability of Fault-Tolerant Consensus in Asynchronous Unknown Networks: Invited Paper. In *III ACM SIGACT-SIGOPS International Workshop on Reliability, Availability, and Security. Co-located with the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*.

[Greve and Tixeuil 2007] Greve, F. G. P. and Tixeuil, S. (2007). Knowledge Connectivity vs. Synchrony Requirements for Fault-Tolerant Agreement in Unknown Networks. In *Proc. of the Int. Conf. on Dependable Systems and Networks - DSN 2007*, pages 82–91.

[Hadzilacos and Toueg 1994] Hadzilacos, V. and Toueg, S. (1994). A Modular Approach to the Specification and Implementation of Fault-Tolerant Broadcasts. Technical report, Department of Computer Science, Cornell University, New York - USA.

[Hawlitschek et al. 2018] Hawlitschek, F., Notheisen, B., and Teubner, T. (2018). The limits of trust-free systems: A literature review on blockchain technology and trust in the sharing economy. *Electronic commerce research and applications*, 29:50–63.

[Howard 2014] Howard, H. (2014). ARC: Analysis of Raft Consensus. Technical Report UCAM-CL-TR-857, University of Cambridge, Computer Laboratory.

[Huh et al. 2017] Huh, S., Cho, S., and Kim, S. (2017). Managing IoT Devices Using Blockchain Platform. In *Advanced Communication Technology (ICACT), 2017 19th International Conference on*, pages 464–467. IEEE.

[IRTF 2017] IRTF (2017). Decentralized Internet Infrastructure Research Group. `https://trac.ietf.org/trac/irtf/wiki/blockchain-federation`. Acessado em 9 abril de 2019.

[ISO/TC 307 2016] ISO/TC 307 (2016). Blockchain and distributed ledger technologies. `https://www.iso.org/committee/6266604.html`. Acessado em 12 de maio de 2019.

[ITU-T FG DLT 2017] ITU-T FG DLT (2017). Focus Group on Application of Distributed Ledger Technology. `https://itu.int/en/ITU-T/focusgroups/dlt/Pages/default.aspx`. Acessado em 12 de maio de 2019.

[Khatoun and Zeadally 2016] Khatoun, R. and Zeadally, S. (2016). Smart Cities: Concepts, Architectures, Research Opportunities. *Commun. ACM*, 59(8):46–57.

[Kiayias et al. 2017] Kiayias, A., Russell, A., David, B., and Oliynykov, R. (2017). Ouroboros: A provably secure proof-of-stake blockchain protocol. In Katz, J. and Shacham, H., editors, *Advances in Cryptology – CRYPTO 2017*, pages 357–388, Cham. Springer International Publishing.

[Kotla et al. 2009] Kotla, R., Alvisi, L., Dahlin, M., Clement, A., and Wong, E. (2009). Zyzzyva: Speculative Byzantine fault tolerance. *ACM Transactions on Computer Systems*, 27(4):7:1–7:39.

[Kwon 2014] Kwon, J. (2014). Tendermint: Consensus without mining. *Draft v. 0.6, fall*.

[Lamport 1998] Lamport, L. (1998). The Part-Time Parliament. *ACM Transactions Computer Systems*, 16(2):133–169.

[Lamport 2001] Lamport, L. (2001). Paxos Made Simple. *ACM SIGACT News*, 32(4):18–25.

[Lamport et al. 2010] Lamport, L., Malkhi, D., and Zhou, L. (2010). Reconfiguring a state machine. *SIGACT News*, 41:63–73.

[Lamport et al. 1982] Lamport, L., Shostak, R., and Pease, M. (1982). The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401.

[Lee et al. 2016] Lee, K., James, J. I., Ejeta, T. G., and Kim, H. J. (2016). Electronic voting service using block-chain. *Journal of Digital Forensics, Security and Law*, 11(2):8.

[Lynch 1996] Lynch, N. A. (1996). *Distributed Algorithms*. Morgan Kauffman.

[Machina Research 2016] Machina Research (2016). IoT global forecast anaysis 2015-2025. Technical report. `https://machinaresearch.com/login/?next=/report/iot-global-forecast-analysis-2015-25/`. Acessado em 31 de outubro de 2018.

[Mattos et al. 2018] Mattos, D. M. F. et al. (2018). Blockchain para Segurança em Redes Elétricas Inteligentes: Aplicações, Tendências e Desafios. In *Minicursos do SBSeg'2018*, pages 140–194.

[Mello et al. 2017] Mello, A. M., Marino, F. C. H., and Santos, R. R. (2017). Segurança de Aplicações Blockchain Além das Criptomoedas. In *Minicursos do SBSeg'2017*, pages 99–148.

[Miller et al. 2016] Miller, A., Xia, Y., Croman, K., Shi, E., and Song, D. (2016). The Honey Badger of BFT Protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 31–42, New York, NY, USA. ACM.

[Mora et al. 2018] Mora, O. B., Rivera, R., Larios, V. M., Beltrán-Ramírez, J. R., Maciel, R., and Ochoa, A. (2018). A Use Case in Cybersecurity based in Blockchain to deal with the security and privacy of citizens and Smart Cities Cyberinfrastructures. In *2018 IEEE International Smart Cities Conference (ISC2)*, pages 1–4.

[Nakamoto 2008] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

[Olson et al. 2018] Olson, K., Bowman, M., Mitchell, J., Amundson, S., Middleton, D., and Montgomery, C. (2018). Sawtooth: An Introduction. *The Linux Foundation, Jan*.

[Ongaro and Ousterhout 2014] Ongaro, D. and Ousterhout, J. (2014). In Search of an Understandable Consensus Algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA. USENIX Association.

[Paul et al. 2014] Paul, G., Sarkar, P., and Mukherjee, S. (2014). Towards a More Democratic Mining in Bitcoins. In *International Conference on Information Systems Security*, pages 185–203. Springer International Publishing.

[PayPal 2019] PayPal (2019). Paypal Holdings, Inc. (PYPL) SEC Filing 10-K Annual report for the fiscal year ending Monday, December 31, 2018. `https://filings.last10k.com/sec-filings/1633917/000163391719000043/pypl201810-k.htm.pdf`. Acessado em 12 de maio de 2019.

[Pieroni et al. 2018] Pieroni, A., Scarpato, N., Di Nunzio, L., Fallucchi, F., and Raso, M. (2018). Smarter City: Smart Energy Grid Based on Blockchain Technology. *International Journal on Advanced Science, Engineering and Information Technology*, 8(1):298–306.

[Popov 2017] Popov, S. (2017). The Tangle. *cit. on*, page 131. `http://www.descryptions.com/Iota.pdf`. Acessado em 31 de outubro de 2018.

[Rabin and Ben-Or 1989] Rabin, T. and Ben-Or, M. (1989). Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, pages 73–85, New York, NY, USA. ACM.

[Radford 2016] Radford, D. (2016). Europe Blockchain Working Group. Standard, The International Securities Association for Institutional Trade Communication. `https://isitc-europe.com/isitc-europe-blockchain-working-group`. Acessado em 12 de maio de 2019.

[Rahman et al. 2019] Rahman, M. A., Rashid, M. M., Hossain, M. S., Hassanain, E., Alhamid, M. F., and Guizani, M. (2019). Blockchain and IoT-Based Cognitive Edge Framework for Sharing Economy Services in a Smart City. *IEEE Access*, 7:18611–18621.

[Rebello 2019] Rebello, G. A. F. (2019). Encadeamento Seguro de Funções Virtuais de Rede Baseado em Correntes de Blocos. Projeto Final de Graduação, Universidade Federal do Rio de Janeiro, Brasil.

[Rebello et al. 2019a] Rebello, G. A. F., Alvarenga, I. D., Sanz, I. J., and Duarte, O. C. M. B. (2019a). BSec-NFVO: A Blockchain-based Security for Network Function Virtualization Orchestration. In *IEEE International Conference on Communications (ICC)*. A ser publicado.

[Rebello et al. 2019b] Rebello, G. A. F., Camilo, G. F., Silva, L. G. C., de Souza, L. A. C., Guimarães, L. C. B., and Duarte, O. C. M. B. (2019b). Segurança na Internet do Futuro: Provendo Confiança Distribuída através de Correntes de Blocos na Virtualização de Funções de Rede. In *Minicursos do SBRC'2019*.

[Rebello et al. 2019c] Rebello, G. A. F., Camilo, G. F., Silva, L. G. C., Guimarães, L. C. B., de Souza, L. A. C., Alvarenga, I. D., and Duarte, O. C. M. B. (2019c). Provendo uma Infraestrutura de Software Fatiada, Isolada e Segura de Funções Virtuais através da Tecnologia de Corrente de Blocos. Technical report, Grupo de Teleinformática e Automação (GTA/COPPE/UFRJ).

[Schneider 1990] Schneider, F. B. (1990). Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comput. Surv.*, 22(4):299–319.

[Schwartz et al. 2014] Schwartz, D., Youngs, N., and Britto, A. (2014). The Ripple Protocol Consensus Algorithm. *Ripple Labs Inc White Paper*, 5.

[Smolensk 2018] Smolensk, M. (2018). Lightstreams White Paper. `https://s3.amazonaws.com/lightstreams/lightstreams_whitepaper.pdf.Acessadoem12demaiode2019.`

[Sousa et al. 2018] Sousa, J., Bessani, A., and Vukolic, M. (2018). A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 51–58.

[Statista 2018] Statista (2018). Internet of things - number of connected devices worldwide. Technical report. `https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`. Acessado em 31 de outubro de 2018.

[Talari et al. 2017] Talari, S., Shafie-khah, M., Siano, P., Loia, V., Tommasetti, A., and Catalão, J. P. S. (2017). A Review of Smart Cities Based on the Internet of Things Concept. *Energies*, 10(4).

[Tikhomirov 2018] Tikhomirov, S. (2018). Ethereum: State of Knowledge and Research Perspectives. In *Foundations and Practice of Security*, pages 206–221. Springer International Publishing.

[Truong et al. 2018] Truong, N. B., Um, T., Zhou, B., and Lee, G. M. (2018). Strengthening the Blockchain-Based Internet of Value with Trust. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7.

[Vasin 2014] Vasin, P. (2014). Blackcoin's Proof-of-Stake Protocol v2. *URL: https://blackcoin. co/blackcoin-pos-protocol-v2-whitepaper. pdf*, 71.

[Veronese et al. 2013] Veronese, G., Correia, M., Bessani, A., Chung, L., and Verissimo, P. (2013). Efficient Byzantine fault tolerance. *IEEE Transactions on Computers*, 62(1):16–30.

[Visa 2019] Visa (2019). About VisaNet. `https://usa.visa.com/about-visa/visanet.html`. Acessado em 12 de maio de 2019.

[W3C 2016] W3C (2016). Blockchain Community Group. `https://www.w3.org/community/blockchain`. Acessado em 12 de maio de 2019.

[Wilkinson et al. 2014] Wilkinson, S., Boshevski, T., Brandoff, J., and Buterin, V. (2014). Storj: a Peer-to-Peer Cloud Storage Network. `https://storj.io/storj2014.pdf`. Acessado em 12 de maio de 2019.

[Wood 2014] Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger.

[Wüst and Gervais 2018] Wüst, K. and Gervais, A. (2018). Do you need a blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54. IEEE.

[Xie et al. 2019] Xie, J., Tang, H., Huang, T., Yu, F. R., Xie, R., Liu, J., and Liu, Y. (2019). A Survey of Blockchain Technology Applied to Smart Cities: Research Issues and Challenges. *IEEE Communications Surveys Tutorials*.

[Xu et al. 2017] Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., Pautasso, C., and Rimba, P. (2017). A Taxonomy of Blockchain-Based Systems for Architecture Design. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 243–252.

[Zhang et al. 2018] Zhang, P., Schmidt, D. C., White, J., and Lenz, G. (2018). Chapter One - Blockchain Technology Use Cases in Healthcare. In Raj, P. and Deka, G. C., editors, *Blockchain Technology: Platforms, Tools and Use Cases*, volume 111 of *Advances in Computers*, pages 1 – 41. Elsevier.

[Zheng et al. 2018] Zheng, Z., Xie, S., Dai, H.-N., Chen, X., and Wang, H. (2018). Blockchain Challenges and Opportunities: A Survey. *International Journal of Web and Grid Services*, 14(4):352–375.

Capítulo 4

Autenticação usando Sinais Biométricos: Fundamentos, Aplicações e Desafios

Eduardo Cerqueira, Paulo Resque, Iago Medeiros, Lucas Bastos,
Alex Santos, Thais Tavares, Denis Rosário, Aldri Santos e Michele Nogueira

Abstract

Our systems and data (e.g., websites, smartphones, safes, cars, banks, airports) are protected by traditional authentication methods. However, the growing concern about information security and the deployment of smart cities are demanding efforts to find solutions that prevent theft, loss, unauthorized copy, or the forgery of keys, tokens, and passwords. The Internet of Things (IoT) enabled a large increase of personal data collected and published in the Internet. In this context, biometric authentication has earned more and more place as a security solution because they demand, in general, physical presence, vitality, and they are hard to falsify. Among the biometric signals used in academic or commercial products, we mention: iris, face, the palm of hand, fingerprints, walking, voice, electrocardiogram (ECG), electroencephalogram (EEG), and photoplethysmogram (PPG). This chapter brings the state-of-the-art about the use of several biometric signals in an authentication system, contextualizing the applications already developed and the challenges they faced when coexisting with Smart Cities and the Internet of Things. Each type of biometric signal has its own challenges for data acquisition, cost, feature selection, and method to implement the classification. In addition, this chapter presents a review of the machine learning techniques used in biometric systems. The proper choice of the identification technique and classification directly influences results, costs and also the required amount of input data, and the quality of the captured data.

Resumo

Nossos sistemas e dados (ex. websites, smartphones, cofres, carros, bancos, aeroportos) ainda são protegidos por métodos tradicionais de autenticação. Porém, a crescente preocupação com a segurança da informação e a implantação de cidades inteligentes vêm exigindo esforços para encontrar soluções que evitem o roubo, perda, cópia ou falsificação da chaves, tokens ou senhas. A IoT possibilitou um grande aumento na quantidade de dados pessoais coletados e publicáveis na Internet. Nesse contexto, os sistemas biométricos têm ganhado cada vez mais espaço como solução de segurança por exigir a presença física, vitalidade, ser de difícil falsificação. Dentre os sinais biométricos utilizados na academia ou em produtos comerciais citamos: a íris, a face, a palma da mão, as digitais, o padrão no modo de andar, a voz, ECG, EEG e PPG.

Este capítulo traz o estado da arte sobre a utilização de diversos sinais biométricos em sistemas de autenticação, contextualizando as aplicações já desenvolvidas e os desafios que enfrentaram ao coexistirem com Cidades Inteligentes e a Internet das Coisas (IoT). Cada tipo de sinal biométrico tem seus desafios quanto a aquisição de dados, o custo, a seleção de características e o método de implementar a classificação. Além disso, este capítulo também apresenta uma revisão das técnicas de aprendizado de máquina utilizadas em sistemas de biometria. A escolha adequada da técnica de identificação de padrões e classificação influencia diretamente os resultados obtidos, os custos e também os requisitos da quantidade de dados de entrada e qualidade dos dados capturados.

4.1. Introdução: a IoT e a expansão da biometria

Atualmente, a Internet das Coisas (IoT) está presente na vida cotidiana da maioria das pessoas nas grandes cidades. Quase todos os lugares já possuem dispositivos inteligentes. Por exemplo, sensores em edifícios e em veículos e alguns itens embarcados em sistemas eletrônicos com conexões entre si ou com a Internet, permitindo, assim, a coleta e o compartilhamento de dados [Dhanvijay and Patil 2019]. A IoT permite que mais objetos sejam controlados e os dados sejam sensoreados remotamente através de uma infraestrutura de rede preexistente. Isto permite que haja maior interação de sistemas computacionais com o mundo real, aumentando, assim, a eficiência em serviços nas cidades inteligentes. Hoje em dia esses serviços estão geralmente ligados ao transporte, à segurança e ao lazer da população, proporcionando o uso mais eficiente de recursos públicos e melhorando a qualidade de vida da população.

A globalização e a redução no custo de produção de dispositivos eletrônicos contribuíram para a expansão da IoT, a qual passou a oferecer tecnologia para países em desenvolvimento na Ásia e na África, e o acesso à rede mundial de computadores (Internet). Diferentes instituições têm feito previsões para o lançamento e a implantação de produtos e serviços de IoT [Columbus 2018]. Os gastos com estes produtos atingirão a marca de 1 trilhão de dólares em 2022 e está em forte expansão, principalmente em países em desenvolvimento e com grande população, como a China e a Índia. Para a [Ericsson 2018], a estimativa é de que haja 3,5 bilhões de dispositivos com rede celular em 2023, como observado na Figura 4.1. Ou seja, uma taxa de crescimento de 27% por ano. A *DBS Asian Insights* acredita que o setor atinja apenas 20% do seu potencial de aplicações em 2019, abrindo caminho para a Inteligência Artificial (IA) e para a Realidade Aumentada.

Contudo, as pesquisas mostram que existem barreiras para a implantação de serviços de IoT. [Bosche et al. 2018] realizaram um *survey* em 2016, e o refizeram em 2018, buscando avaliar a percepção de diversos *players* do mercado em relação às implantações em IoT. Através desses levantamentos, os pesquisadores constataram que as novas tecnologias de virtualização como *Microsoft Azure* e AWS têm possibilitado a expansão de serviços em IoT. Porém, as preocupações com os aspectos de segurança ainda estão entre as maiores barreiras para a adoção de soluções, como ilustra a Figura 4.2. Diferentes aplicações possuem requisitos específicos de segurança computacional. Os sistemas biométricos surgem como uma excelente solução para quando for necessário que haja acesso físico direto, podendo oferecer uma solução escalável com a IoT, protegendo-a de acesso sem autorização, de troca de identidades ou evitando a checagem manual de credenciais, por exemplo. Os sistemas biométricos podem reconhecer indivíduos com base

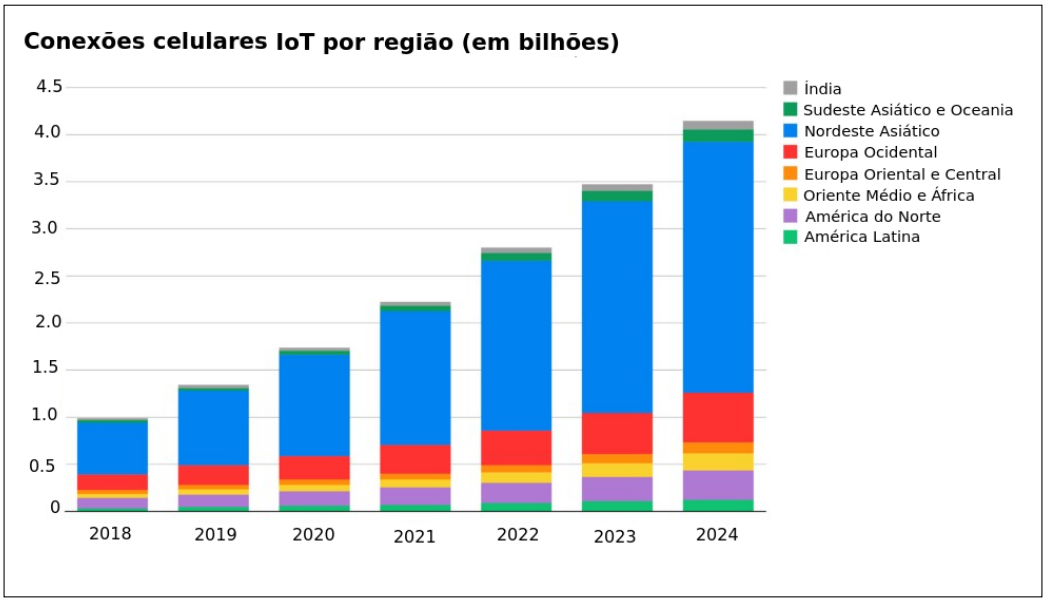


Figura 4.1: Projeção da quantidade de dispositivos IoT conectados (Adaptado de [Ericsson 2018])

em seu comportamento ou suas características biológicas. No *survey*, os consumidores afirmam que comprariam mais dispositivos de IoT e estariam dispostos a pagar um valor até 22% mais caro caso entendessem que as vulnerabilidades de segurança foram tratadas. O potencial para a IoT é imenso e está presente em diversos domínios como *healthcare*, sistemas de transporte, monitoramento ambiental, cidades inteligentes, controle industrial e outros, ainda mais quando a associamos com o aumento de vazão e redução de latência esperados com as tecnologias de 5G.

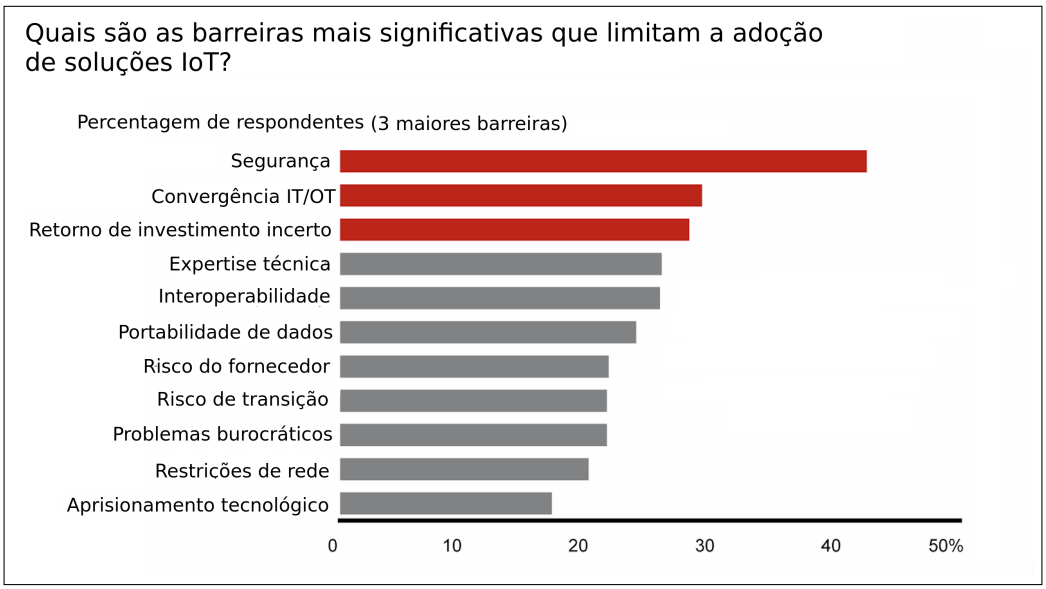


Figura 4.2: Barreiras para a adoção de IoT (Adaptado de [Bosche et al. 2018])

Esses fatores têm motivado pesquisadores a buscar por soluções para fornecer

segurança dos dados ao transmitir informações de saúde através de múltiplos sensores. Um dos desafios consiste em como restringir o acesso aos dados e serviços apenas a usuários autorizados, visto que este controle passa pela autenticação da identidade do usuário. Na sociedade moderna, garantir a correta identificação de um indivíduo tornou-se um requisito essencial para aplicações de tempo real. As aplicações vão desde investigação forense, controle de imigração, transações financeiras e segurança computacional. Nesse contexto, os dispositivos móveis possuem um papel importante na vida cotidiana, não somente pela comunicação mas também pelo entretenimento e pelas relações sociais. É preciso proteger dados bancários, *emails*, fotografias, vídeos e diversos outros dados confidenciais.

Com o aumento de dispositivos conectados à rede, o escopo por potenciais ataques *hackers* ou outros crimes cibernéticos também aumentou. Há o risco de utilização de dispositivos controlados remotamente para ataques de *botnet*, onde milhares de dispositivos podem ser utilizados em conjunto para um ataque em rede. Os ataques clássicos de *Man-in-the-middle*, no qual o invasor faz de forma transparente o intermédio na comunicação entre dois dispositivos, podem enviar informações falsas ou coletar informações sensíveis. Houve um aumento de ocorrência no roubo de dados e de identidades, quando o usuário utiliza de modo descuidado os dispositivos, como telefones celulares, *smartwatches* e outros. O acúmulo de muitas informações em dispositivos pessoais deixa as pessoas vulneráveis a grandes impactos caso esses dados sejam violados. Assim, garantir a segurança desses dados somente com o uso de senhas não tem sido mais suficiente frente aos ataques possíveis. Essa necessidade crescente por proteção dos dados sensíveis fez com que a busca por soluções mais seguras aumentasse, com destaque maior para a biometria. Os sistemas de segurança biométrica podem substituir métodos tradicionais que utilizam senhas ou PINs gestuais na tela do dispositivo. Os métodos embarcados em *smartphones* são reconhecimento por digitais, por face, assinatura, de voz e por íris.

O escaneamento de digitais começou a se popularizar em 2013 com a chegada do *iPhone 5S*. Inicialmente, os usuários podiam registrar suas digitais para desbloquear o *iPhone*, depois passaram a utilizar a digital para autenticar o processamento de compras através do sistema de pagamento *Apple Pay*. Três anos depois, grande parte dos *smartphones* modernos já possuíam sensores de digitais como o *iPhone* 8 e 8 *Plus*, *Samsung Galaxy* S8 e *Galaxy Note* 8, LG G6 e V30, *Huawei Mate* 10, *Google Pixel* 2 e *Pixel* 2 XL, *OnePlus* 5 e 5T, além de muitos outros. Em poucos anos os aparelhos celulares incorporaram a biometria, o que até então parecia um recurso de filmes de ficção científica.

Atualmente, os dispositivos vestíveis podem medir sinais biométricos vitais e não vitais, tais como temperatura corporal, frequência cardíaca, pressão arterial, eletromiograma (EMG), eletrocardiograma (ECG), fotopletismograma (PPG), frequência respiratória, dentre outros sinais. Nesse contexto, a biometria refere-se a tecnologias usadas para medir características físicas ou comportamentais humanas, tais como as fornecidas pela íris, face, impressões digitais, retina, geometria da mão, voz ou assinaturas para detectar e reconhecer indivíduos. Por exemplo, os dispositivos vestíveis no antebraço podem obter sinais de ECG/PPG, processá-lo para extrair as características que identifiquem o usuário e utilizar tal informação tanto para identificação quanto para autenticação.

De um modo geral, estes diferentes sinais vitais ou não vitais são processados em um sistema biométrico em duas etapas: a primeira ligada a captura dos dados e outra ligada

ao reconhecimento. Na etapa de captura, o sistema biométrico coleta o traço biométrico e extrai um conjunto de *features* (características) relevantes e armazena o modelo desses dados extraídos em um *database* (banco de dados). Na etapa de reconhecimento, o sistema captura novamente o traço biométrico de um indivíduo, extrai as características desse sinal e compara esse conjunto de características com os padrões armazenados no banco de dados de modo que possa afirmar de quem é a identidade.

O objetivo deste capítulo que documenta o conteúdo dessa Jornada de Atualização em Informática (JAI) é possibilitar uma revisão do estado arte da utilização de diversos sinais biométricos em sistema de autenticação e suas aplicações. São enfatizadas as particularidades de cada sinal biométrico em termos de acurácia e características utilizadas para identificação de indivíduos. Além do mais, será apresentado ao leitor o conhecimento das diversas técnicas de aprendizado de máquina que usualmente são usadas em sistemas biométricos. Com o conhecimento das características de cada sinal e das técnicas de inteligência computacional existente, o leitor pode aplicar esse conhecimento em base de dados de repositórios públicos disponíveis para estudo.

O restante deste capítulo do JAI está organizado da seguinte forma. A Seção 4.2 apresenta a evolução histórica do uso de biometria e as aplicações de segurança já utilizadas. A Seção 4.3 trata dos aspectos técnicos para a coleta das características únicas de um sinal biométrico. Essa extração de características é essencial para um sistema eficiente de biometria. A Seção 4.4 apresenta as tecnologias de aprendizado de máquina utilizadas comumente em sistemas biométricos. Na Seção 4.5, apresentamos uma breve parte prática com a indicação de bases de dados públicas e avaliações de técnicas de aprendizado de máquina para sistemas biométricos. Na Seção 4.6, são apresentadas as conclusões e oportunidades de pesquisa para o tema.

4.2. Sinais Vitais e Segurança: Evolução Histórica

Diversas evidências mostram que a biometria já vem sendo utilizada desde o mundo antigo por diversas sociedades. Na Babilônia em 1900 A.C. [Daluz 2014], as digitais eram utilizadas em contratos para dar validade aos mesmos. Porém, foram os chineses que aproveitaram melhor o potencial da biometria, utilizando as digitais dos dedos para uma variedade de funcionalidades, incluindo desde registro da população e em cenas de crimes até para validar documentos importantes utilizando as digitais de um lado e um selo oficial do outro lado em documentos para casamento, divórcio, registros do exército e outros. Isso ocorria pois o domínio da escrita era restrita a uma parcela bem pequena da população. Até hoje, a digital é utilizada em documentos em casos que a pessoa não sabe ou não pode assinar. Apesar de ter surgido há tanto tempo, somente em 1901, foi fundado o primeiro escritório de investigação de digitais, a polícia da Inglaterra, conhecida como *Scotland Yard* que foi o primeiro órgão de investigação oficial a ter um departamento dedicado para digitais. Depois disso, a utilização de digitais começou a se espalhar pelas polícias do mundo todo. Nessa época, o reconhecimento através de digitais não era automático, pois era um trabalho complicado, manual e demorado. Além dessas dificuldades, a quantidade de registros ainda era pequena.

A primeira publicação científica sobre o reconhecimento automático biométrico foi realizada por Mitchell Trauring na revista científica *Nature* em 1963 sobre a corres-

pondência entre digitais do dedo [Trauring 1963]. O desenvolvimento de sistemas automatizados baseados em outros traços como voz [Pruzansky 1963], rosto [Bledsoe 1966] e assinatura [Mauceri 1965] também começaram por volta dos anos 60. Somente depois surgiram os sistemas baseados em geometria da mão e íris, assim, a biometria vem evoluindo nos últimos 50 anos. Há uma citação interessante publicada em [Wayman 2007], onde foi feita uma análise dos avanços da biometria entre 1960 e os anos 1990: *"A quick overview of biometric history shows that much of what we consider to be "new" in biometrics was really considered decades ago."* Novas soluções biométricas, que chegaram recentemente a produtos cotidianos, já vinham sendo estudados há muitos anos.

Saindo no mundo forense e judicial, o uso de biometria de digitais chegou ao dia a dia da população através dos celulares e se expandiu por outros setores como segurança predial e serviços médicos, de modo a prover maior segurança e praticidade na identificação dos usuários. Outros tipos, como o reconhecimento facial, só começaram a se popularizar recentemente. O primeiro dispositivo foi lançado em 2011 pela empresa *Google*, o *Galaxy Nexus*, porém naquela época o sistema não se mostrou muito eficaz, ainda necessitando de melhorias. O sistema melhorou a partir do *Galaxy* S8 e teve novo salto com o lançamento do *Iphone* X, no qual o usuário precisava apenas registrar o rosto no celular e depois conseguia acesso instantâneo à tela inicial ou outros serviços que antes eram realizados através de senhas. Ainda hoje, há a necessidade de evolução dos sistemas baseados na face utilizados por muitos celulares devido à qualidade dos sensores ou ao processamento utilizado. Ou seja, ainda é um campo aberto para pesquisa assim como toda a biometria.

Devido a diferentes níveis de segurança ao usar senhas, digitais e o rosto, vários aparelhos celulares utilizam mais de uma opção de biometria. Vinculando o reconhecimento facial para desbloquear o dispositivo e a digital para autorizar uma compra de algum produto/serviço. Além do *Iphone* X, outros dispositivos já utilizam o reconhecimento facial, *Galaxy* S8, *Galaxy Note* 8, LG V30, *OnePlus* 5T, HTC U11, *Huawei* P10, *Moto* G5, *Xiaomi Mi* 6 e *Xiaomi Mi* MIX 2 [Insider 2019]. O escaneamento de íris está mais presente em dispositivos lançados na Ásia, como o *Fujitsu* NX F-04G e *Microsoft Lumia* 950, ambos lançados em 2015. A *Samsung* inclui a funcionalidade no *Galaxy* S8 e *Galaxy Note* 8 lançado em 2017. Assim como o reconhecimento facial, o reconhecimento de íris não atingiu ainda um grau de confiabilidade alto suficiente para ser utilizado em meios de pagamento, por exemplo, sendo utilizada geralmente como um recurso para desbloqueio de tela e diversificação para o usuário. Nesses cenários, o usuário sempre precisa manter uma senha ou desenho padrão como medida de segurança de *backup* em caso de falha de uma das opções biométricas. O reconhecimento de voz ainda não apareceu como uma opção disponível em *smartphones*, apesar das grande marcas como *Apple*, *Google*, *Amazon* e *Microsoft* estarem investindo bastante em soluções com assistentes inteligentes de processamento de voz. Por outro lado, a biometria entrou no imaginário da população através dos filmes de ficção científica, apesar de alguns exageros e previsões que não se concretizaram, grande parte das projeções cinematográficas foram criadas baseadas de trabalhos em andamento pela comunidade científica da época. A Figura 4.3 traz uma linha do tempo do desenvolvimento da biometria até 2014.

Atualmente, os dispositivos vestíveis podem medir sinais biométricos vitais e não vitais, tais como temperatura corporal, frequência cardíaca, pressão arterial, ECG, EMG, frequência respiratória, dentre outras informações. Nesse contexto, a biometria refere-se

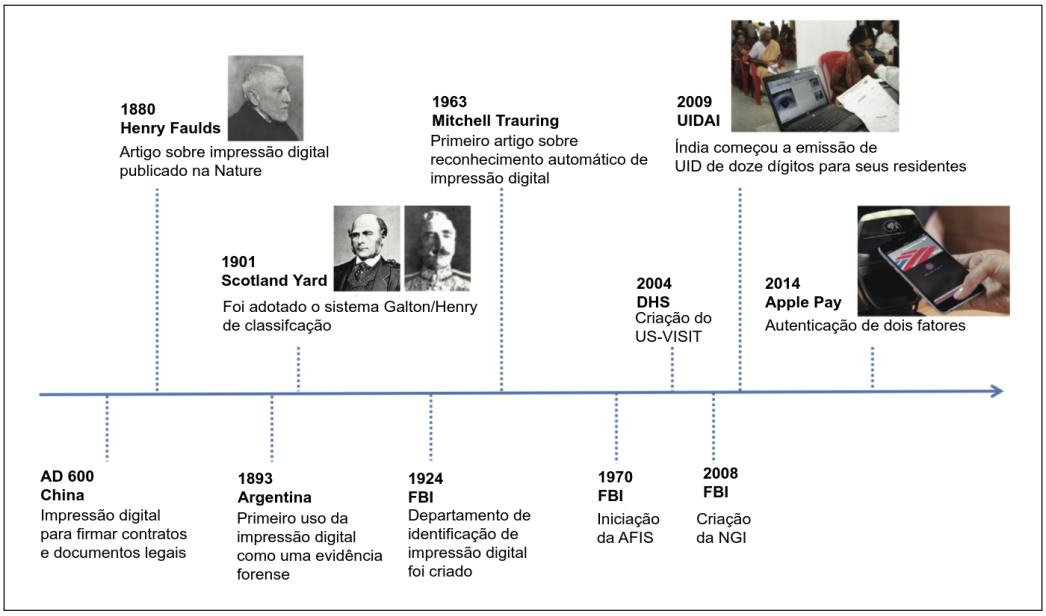


Figura 4.3: Linha do tempo das aplicações com biometria

às tecnologias usadas para medir características físicas ou comportamentais humanas, tais como as fornecidas pela íris, face, impressões digitais, retina, geometria da mão, voz ou assinaturas para detectar e reconhecer indivíduos. A biometria fornece não apenas uma alternativa para IDs ou números PIN (esquemas baseados em conhecimento) para autenticar alguém em um sistema, mas também um método de autenticação contínua. Além disso, ela possui vantagens relacionadas à clonagem, perda de dispositivos e adivinhação de senhas. Embora a biometria possa reduzir limitações de segurança associadas a senhas, os sistemas biométricos também são vulneráveis a ataques de falsificação, ataques de vinculação equivocada de usuários (ou seja, quando um impostor tenta se passar por outro usuário), além de possivelmente aumentar os custos com *hardware* e *software* comparado com o uso de senha ou *token* [Jain et al. 2016]. Mesmo a identificação facial, a qual passou a ser utilizada recentemente em dispositivos móveis, possui riscos associados a falta de confidencialidade, com o uso de imagens publicadas na Internet em sistemas que usam as imagens da face, além das limitações na distinção de gêmeos legítimos. Por esses motivos, julga-se importante a avaliação dos diversos tipos de biometrias existentes, onde a combinação entre elas e o uso adicional de senhas tradicionais seja a solução adotada pela maioria dos sistemas no futuro.

Para entender o papel da biometria e do uso de sinais vitais para aplicações de segurança, é necessário visualizar como a estrutura típica de um sistema de segurança e/ou autenticação baseado em biometria. A Figura 4.4 apresenta aspectos interessantes de um sistema típico de autenticação. Um sistema típico de biometria possui dois estágios de operação, um estágio ligado a captura dos dados e outro ligado ao reconhecimento. Na etapa de captura, o sistema biométrico coleta o traço biométrico e extrai um conjunto de *features* (características) relevantes, armazena um modelo desses dados extraídos em um banco de dados (normalmente é referido como *template*/padrões), e assim consegue associar esses dados coletados a um indivíduo. Na etapa de reconhecimento, o sistema

captura novamente o traço biométrico de um indivíduo, extrai as características desse sinal e compara esse conjunto de características com os padrões armazenados no *database* (banco de dados) de modo que possa afirmar se a identidade é aquela reivindicada ou não.

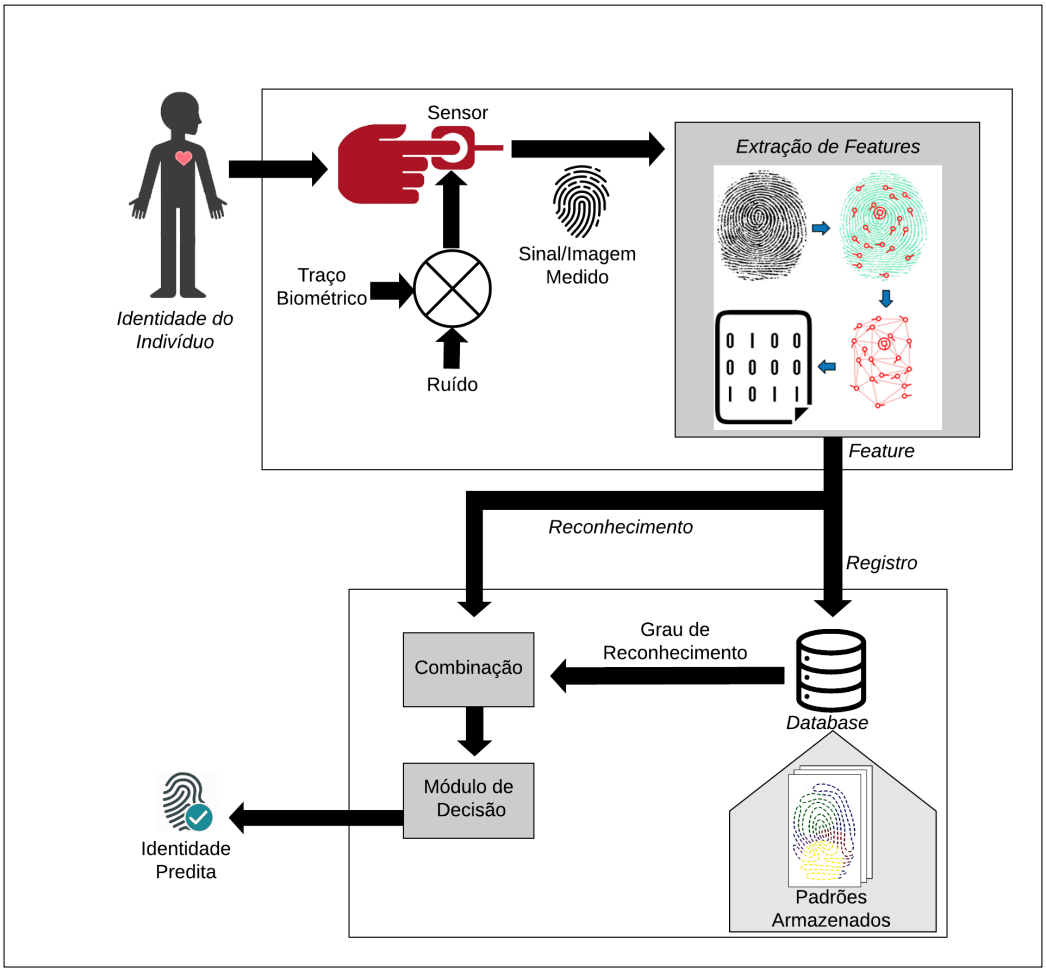


Figura 4.4: Um típico sistema de biometria

4.3. Extração de *features* dos sinais biométricos

Os sinais biométricos, tais como EMG, ECG, PPG e outros, desempenham um papel crucial na autenticação, fornecendo características individuais como uma forma de controle de acesso e identificação de seres humanos por suas características, tais como sinais vitais. Portanto, é necessário considerar técnicas exclusivas de extração de características para sinais biométricos adquiridos de dispositivos vestíveis. Por exemplo, os dispositivos vestíveis obtêm sinais de ECG/PPG, processam os para extrair as características que identifiquem o usuário e utilizam tal informação para identificação e autenticação.

Um sistema típico de biometria possui duas fases de operação: (i) a inscrição ou registro do indivíduo e (ii) o reconhecimento. A primeira fase coleta os sinais biométricos dos indivíduos. Assim, nessa fase, é extraído um conjunto de características relevantes e armazenadas em um banco de dados como um *template*, de modo a associar àquelas características a um usuário ou indivíduo. Na segunda fase, o sistema captura a carac-

terística novamente do indivíduo e a compara aos dados armazenados; através do uso de diversas técnicas de aprendizado de máquina o sistema responde se os dados coletados são do indivíduo alvo ou não.

Teoricamente, qualquer sinal anatômico, comportamental ou fisiológico de um indivíduo pode ser usado como um marcador biométrico. Contudo, a escolha de qual marcador utilizar depende dos requisitos da aplicação e do grau de algumas propriedades a serem satisfeitas: (i) exclusividade ou distinção, (ii) durabilidade, (iii) universalidade, (iv) coleta, (v) desempenho, (vi) aceitação do usuário, (vii) invulnerabilidade e $(viii)$ integração. A Figura 4.5 apresenta as principais características humanas usadas em identificação biométrica. Nas próximas subseções, serão descritos os aspectos técnicos da extração de *features* de sinais biométricos.

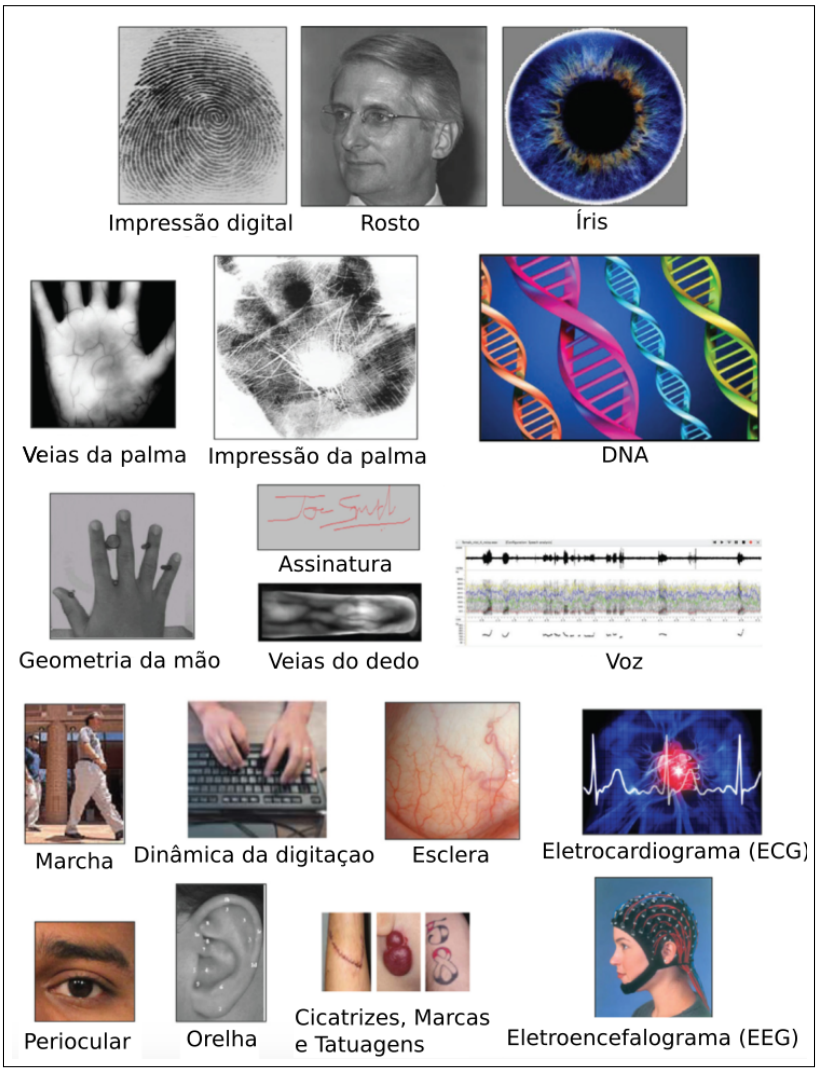


Figura 4.5: Características mais utilizadas em sistemas de identificação biométrica

4.3.1. *Features* consideradas no reconhecimento de mãos

A impressão digital é uma informação humana usada como biometria. Porém, há vários atributos, além das impressões digitais, que foram identificados e testados, tais como

palmprint (impressão da palma), geometria da mão, impressão das juntas dos dedos, região abaixo das unhas e o padrão das veias da mão (Figura 4.6). No entanto, os atributos baseados em mão são uma extensão da tecnologia de impressão digital [Unar et al. 2014].

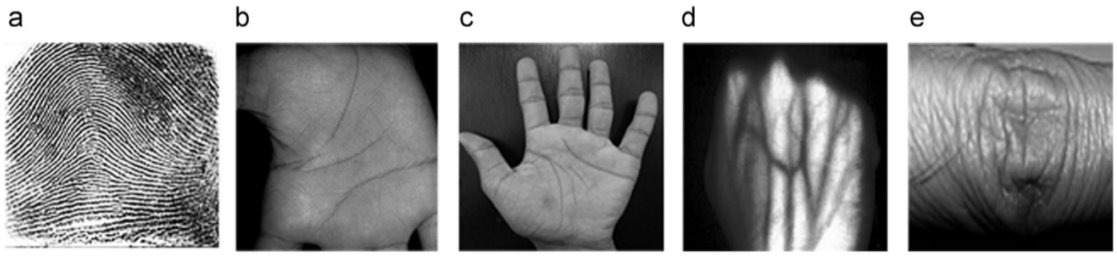


Figura 4.6: Modalidades das regiões da mão: (a) impressão digital (b) impressão da palma, (c) geometria da mão, (d) padrão das veias da mão, (e) articulação dos dedos (Adaptado de [Unar et al. 2014])

O sistema de reconhecimento de impressão digital caracteriza as texturas de cumes e sulcos (linhas) presentes nas pontas dos dedos. As linhas são quase paralelas, com exceção de alterações do padrão, chamadas de minúcias. Existem categorias para estes pontos característicos (minúcias), como arco, presilha interna, presilha externa e verticilo [Liu 2010]. Individualmente, os pontos de minúcia executam a tarefa de reconhecimento. Como todos os outros sistemas biométricos, um módulo de aquisição captura as pontas dos dedos, de preferência, a partir de imagens de alta resolução e o sistema extrai os sulcos, alguns pontos singulares e pontos de minúcia. A Figura 4.7 mostra o processo de extração de *features* para a impressão digital, onde são detalhados a coleta, o mapeamento e o registro das informações, como dados a serem inseridos no *database* para reconhecimento. Outros sistemas, como os baseados na geometria da mão, conseguem trabalhar com imagens de baixa qualidade, capturando as linhas principais, rugas e textura.

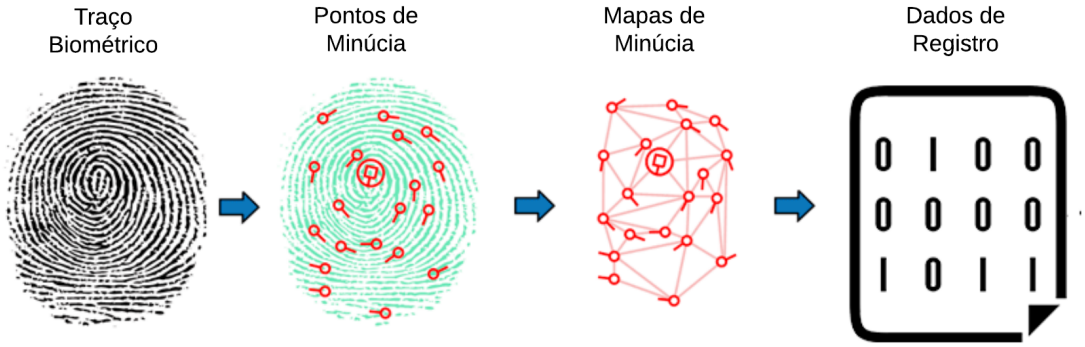


Figura 4.7: Identificando pontos de minúcia

Estes métodos de aquisição de imagens utilizados nas modalidades baseadas das mãos usam dados de sensores ópticos, térmicos, de silício ou imagens de ultrassom.

Devido à redução do custo dos sensores de imagem, assim como o pequeno tamanho para guardar o *template*, torna os atributos da região da mão como uma boa opção para muitos tipos de aplicações, em comparação com assinaturas biométricas mais complexas. No entanto, a biometria manual tem desafios. As imagens distorcidas, o contato físico com o dispositivo de imagem, a necessidade de cooperação do usuário, doenças da mão (artrite), contaminantes naturais para a imagem (como células mortas, cicatrizes, cortes, pele úmida e/ou seca), e sensores de imagens com superfícies sujas ou oleosas que comprometem a eficácia do sistema. Outros fatores também influenciam a precisão da mão.

A presença de usuários não treinados e não cooperativos colocando de forma imprópria o polegar no sensor, como apontado por [Jain and Duta 1999], [Ross et al. 1999], pode prejudicar o reconhecimento [Kukula and Elliott 2006]. Dedos pequenos também são mais complicados para reconhecimento. É importante notar que a mão humana é um objeto flexível e sua silhueta pode sofrer deformações não lineares que dificultam o processo de captura e reconhecimento de mão. Como exemplo, temos o dedo do polegar que se deforma mais do que outros quatro dedos. Assim, excluir o dedo polegar do cálculo do vetor de recursos resolve esse problema [Duta 2009]. Mesmo assim, [Chen et al. 2005] mostrou que os sistemas de formas manuais são vulneráveis a ataques de falsificação.

Alguns trabalhos encontrados na literatura abordam técnicas de *Deep Learning* para realizar o reconhecimento de mãos, principalmente envolvendo impressões digitais e impressão de veias da palma. [Sajjad et al. 2018] propuseram uma nova técnica híbrida, composta por reconhecimento das mãos e o reconhecimento facial, que garante a autenticidade do usuário ao sistema. Uma das duas etapas do esquema proposto testa os dados biométricos coletados em modelos baseados em Redes Neurais Convolucionais (*Convolutional Neural Networks* - CNN) para detectar um possível *spoofing* (falsificação) desses dados. Caso não seja atestado fraude, o sistema realiza a autenticação do usuário. Para impressões digitais coletadas com baixa qualidade, [Wang et al. 2016] propuseram um algoritmo de reconhecimento dessas impressões, melhorando sua qualidade a partir de pontos de *features*. Este algoritmo também é baseado em CNN e sua taxa de reconhecimento é comparada com a taxa de reconhecimento baseada na Análise de Componentes Principais e *k-Nearest Neighbor* (k-NN). Outra abordagem de *Deep Learning* para melhorar o desempenho de sistemas de identificação por impressão digital foi proposta por [Su et al. 2017]. Essa abordagem consiste na detecção de poros na pele através da capacidade de classificação e aprendizado de *features* das CNNs. A fim de aumentar a robustez contra os materiais falsificados, [Zhang et al. 2016] propuseram um novo método de detecção de impressão digital 2D, principalmente para *smartphones*, combinando CNNs com dois descritores locais (Padrão Binário Local e Quantização de Fase Local). Por fim, [Jung and Heo 2018] introduziram uma nova arquitetura de CNNs para o problema de detecção de vivacidade das impressões digitais, a qual pode fornecer uma estrutura mais robusta para treinamento e detecção de redes do que os métodos anteriores. A proposta permite controlar o nível aceitável de falsos positivos ou falsos negativos das impressões digitais coletadas.

4.3.2. *Features* consideradas no reconhecimento ocular

Para o reconhecimento ocular, três modalidades foram as mais utilizadas ao longo da história e estão ilustradas na Figura 4.8. Modalidades de região ocular como retina, íris

e padrão das veia da esclera, ganharam considerável atenção de pesquisadores devido a região ocular possuir sinais mais precisos, altamente confiáveis, bem protegidos, estável e quase impossíveis de forjar assinaturas biométricas [Oinonen et al. 2010]. Um sistema de identificação da retina leva conta a estrutura única e invariante de veias sanguíneas presentes na retina humana para estabelecer a identidade. O processo de escaneamento captura algumas *features* relacionadas às marcas (como posição e bifurcações dos vasos sanguíneos) ou medidas da área de referência (fóvea ou o disco ótico).

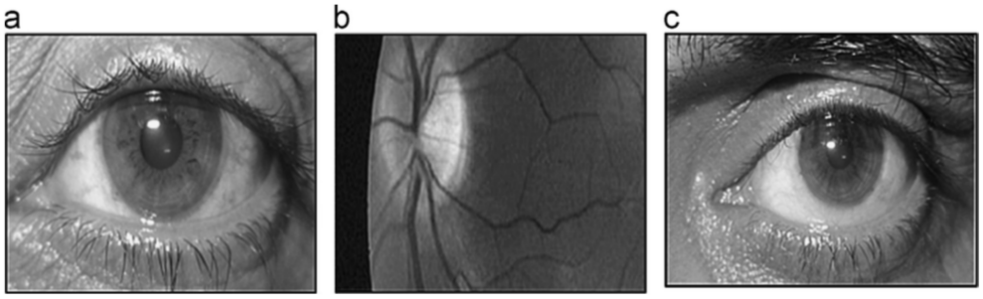


Figura 4.8: Modalidades da região ocular: (a) íris, (b) retina, (c) esclera (Adaptado de [Unar et al. 2014])

Já um sistema de identificação baseado em escleras considera o padrão vascular dos vasos sanguíneos presentes na região da esclera do olho humano [Lumini and Nanni 2017]. O sistema baseado em íris cria um modelo usando os padrões únicos presentes na íris como as criptas, linhas radiais, área da pupila, área ciliar e anel limite [Daugman 2010]. Foram propostos modelos onde tanto as cores quanto as formas eram utilizadas para distinguir as pessoas. Apesar das primeiras propostas de utilização da íris terem iniciado em 1936 com Frank Burch, o primeiro sistema completo foi implementado somente no início dos anos 1990 com uma câmera para capturar a imagem da íris, algoritmos para processar as imagens e retirar a região da íris e código de representação da íris capaz de convertê-la em um código binário compacto. Ou seja, cada íris de um indivíduo era convertido para uma espécie de *hash* e utilizado a distribuição de distâncias *hamming* para classificá-las.

Recentemente, as técnicas de *Deep Learning*, especialmente utilizando CNNs, têm mostrado grande potencial para a classificação de imagens. O primeiro trabalho aplicando *Deep Learning* ao reconhecimento de íris foi o *framework* DeepIris, proposto por [Liu et al. 2016], o qual reconhece íris heterogêneas, das quais as imagens foram obtidas com diferentes tipos de sensores, e estabelece a similaridade entre um par de imagens de íris usando CNNs. [Gangwar and Joshi 2016] também desenvolveram um aplicativo para o reconhecimento da íris em imagens obtidas de diferentes sensores, chamado DeepIrisNet, através de CNN. Além disso, duas arquiteturas da CNN foram apresentadas, a saber, DeepIrisNet-A e DeepIrisNet- B, sendo a primeira baseada em camadas convolucionais padrão, e a segunda baseada em camadas de iniciação [Szegedy et al. 2015].

Outras abordagens propostas como [Al-Waisy et al. 2018] utilizaram um sistema multi-biométrico, usando as íris esquerda e direita de uma pessoa. Experimentos foram realizados em bancos de dados de imagens NIR (próximo ao infravermelho) obtidos em ambientes controlados. O processo tem cinco etapas: detecção de íris, normalização de íris, extração de *features*, correspondência com *Deep Learning* e, finalmente, a fusão de escores

correspondentes de cada íris. Durante a fase de treinamento, os autores aplicaram diferentes configurações e arquiteturas da CNN e escolheram a melhor com base nos resultados do conjunto de validação. [Nguyen et al. 2017] demonstraram que os descritores genéricos usando *Deep Learning* são capazes de representar as características da íris. Uma descrição desse sistema pode ser visualizada na Figura 4.9.

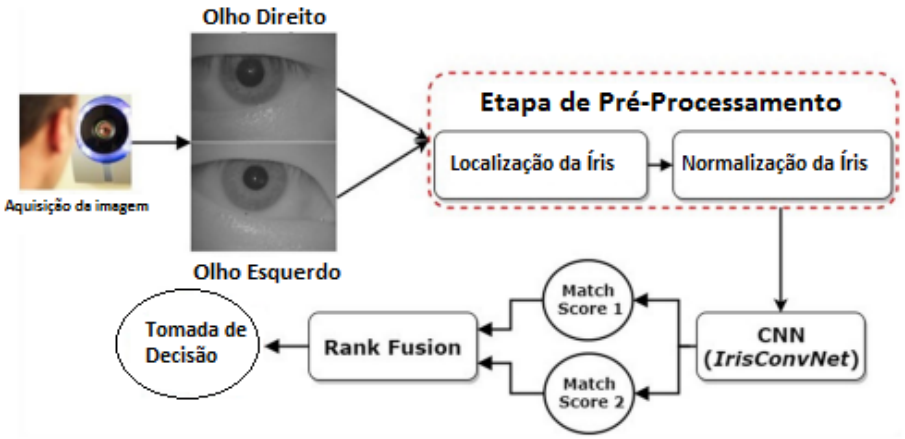


Figura 4.9: Esquema de Identificação utilizando CNN e Íris dos olhos Direito e Esquerdo (Adaptado de [Al-Waisy et al. 2018])

De um modo geral, os trabalhos de reconhecimento de íris utilizam uma imagem de entrada que passa por um processo de normalização, localização da íris e segmentação antes de ser utilizada em uma técnica de inteligência computacional. Todas essas abordagens têm limitações práticas. Por exemplo, o padrão vascular da retina só pode ser visto expondo o olho humano a uma luz infravermelha, enquanto a textura da íris pode ser adquirida através da iluminação do olho humano com uma luz perto da infravermelha ou luz de comprimento de onda invisível [Daugman 2010]. A íris se torna uma abordagem promissora em assinaturas biométricas oculares devido a sua aquisição de imagens menos invasiva. As modalidades da região ocular exigem um alto grau de cooperação por parte dos sensores médicos/químicos, o alto custo de sensores de imagem, e reflexões de fontes de luz ambiente. Esses requisitos eventualmente limitam a sua utilização em ambientes industriais ou a sua utilização em dispositivos com limitações de recursos.

4.3.3. *Features* consideradas no reconhecimento facial

O reconhecimento facial humano é uma técnica bem conhecida, já que o rosto humano é o mais natural traço biométrico usado para reconhecer indivíduos por séculos. Um sistema de reconhecimento facial leva em conta algumas características/*features*, como distância entre os olhos, boca, lado do nariz, imagem da face inteira, pontos de canto, contornos, pelos faciais, redondeza de face, etc [Unar et al. 2014]. Mesmo com muitos recursos disponíveis, esses sistemas não garantem uma identificação confiável na presença de alguns artefatos, como o uso de cirurgias plásticas, sendo necessários alguns novos algoritmos para mapear essas possíveis alterações. Por exemplo, o sistema deve estar preparado para reduzir o impacto das mudanças pessoais ao longo do tempo sobre a

precisão de tais sistemas. A comunidade de pesquisa propôs a ideia de reconhecimento humano baseado em termografia facial com objetivo de fornecer um sistema robusto de reconhecimento de face. A Figura 4.10 mostra as modalidades faciais e de termografia facial usadas para identificar os indivíduos.

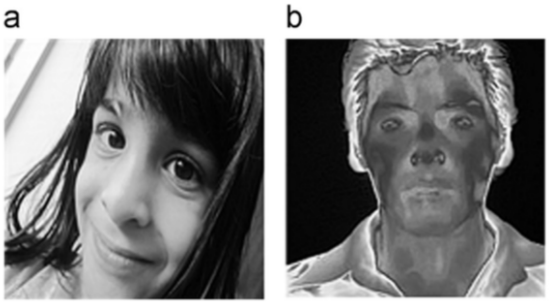


Figura 4.10: Modalidades na região facial: (a) face, (b) termografia facial (Adaptado de [Unar et al. 2014])

A estrutura não linear da face humana faz dela um sofisticado problema de reconhecimento de padrões, bem como uma área ativa de pesquisa em aplicações de visão computacional [Abate et al. 2007]. Em relação às medidas de segurança, é necessária a utilização de *features* 3D ao invés de *features* 2D não apenas para melhorar a acurácia e reduzir erros de reconhecimento, mas também para reduzir a possibilidade de falsificação, quando um invasor tenta ser outra pessoa.

Nos últimos anos, as técnicas de rede neural, como *Deep Learning* e CNN, alcançaram bom desempenho em várias tarefas de visão computacional, desde classificação de imagem até a detecção de objetos e segmentação semântica. Ao contrário das abordagens de visão computacional tradicionais, os métodos de *Deep Learning* demonstraram resultados satisfatórios no desafio visual do reconhecimento da grande escala de imagens (ILSVRC) [Krizhevsky et al. 2012]. Juntamente com a popularidade de *Deep Learning* em visão computacional, mais pesquisas estão surgindo para explorar esta técnica na resolução de tarefas de detecção de rostos. Em geral, o reconhecimento facial pode ser considerado como uma tarefa de detecção de objeto de especialidade em visão computacional. [Sun et al. 2018] explora tais técnicas para a detecção facial.

Tendo em vista o crescimento no uso de técnicas de inteligência artificial em reconhecimento facial, várias empresas estão lançando serviços que se beneficiam dessa tecnologia. A China é um país líder nesse setor em várias frentes: seja permitir sacar dinheiro em caixas eletrônicos de bancos sem uso de cartão, fazer compras em lojas de conveniência [Zuo 2019], viajar sem passagem ou identidade em um aeroporto [Kinetz 2019] e outros. Entretanto, alguns atos podem ser considerados duvidosos por partes das autoridades, como cobrar multas para as pessoas que não atravessam na faixa de pedestre e exibir tais pessoas em um mural da vergonha [Baynes 2019], ou guardas usarem *smart glasses* para avaliar quem o sistema considera como infrator [Russel 2019], podendo ser pessoas que realmente cometeram crimes graves como assassinato ou roubo, pessoas que simplesmente atravessaram a rua fora da faixa, ou até mesmo pessoas que foram identificadas erroneamente [Dodds 2019]. Em outros locais, como em Nova Déli, as autoridades conseguiram encontrar crianças desaparecidas ao usar esta tecnologia [Times 2019].

Ainda haverá muitos debates sobre o tema e aplicação consciente do uso de reconhecimento facial, sendo que alguns lugares são mais favoráveis (como em Londres [Wright 2019]) e outros já entraram em processo de banimento (como em San Francisco [Kate Conger 2019]). O uso de reconhecimento facial é uma das técnicas biométricas mais famosas atualmente por estas aplicações. Apesar de polêmico, cabe ao governo, empresas legais e sociedade verem formas de aplicar seu uso de forma benéfica para a população, sem tentar infringir questões de privacidade.

4.3.4. *Features* consideradas em sinais vitais

A Figura 4.11 ilustra os passos para a extração de *features* de sinais vitais encontrados na literatura. A maioria dos sistemas pode ser dividida em cinco etapas principais [Bonissi et al. 2013]: aquisição, pré-processamento do sinal, extração de *features*, correspondência e classificação. Geralmente, os sinais vitais são capturados por um sensor e pré-processados para remoção de ruídos. Em seguida, a detecção de picos do sinal vital é realizada para dividir o sinal em diferentes segmentos (batidas). Depois da segmentação e normalização, aplica-se a extração de *features*. As *features* resultantes são processadas para formar um *template*, o qual é comparado com o *template* de usuários autorizados. Finalmente, a classificação é aplicada para distinguir os dados de sinais vitais genuínos dos dados de sinais vitais impostores [Karimian et al. 2017, Sancho et al. 2018]. Por conta disso, a extração das *features* é a etapa mais importante, pois é quando as características do usuário são extraídas do sinal vital para que o processo de autenticação seja realizado. Na sequência, a extração das *features* de sinais ECG e PPG é descrita em detalhes.

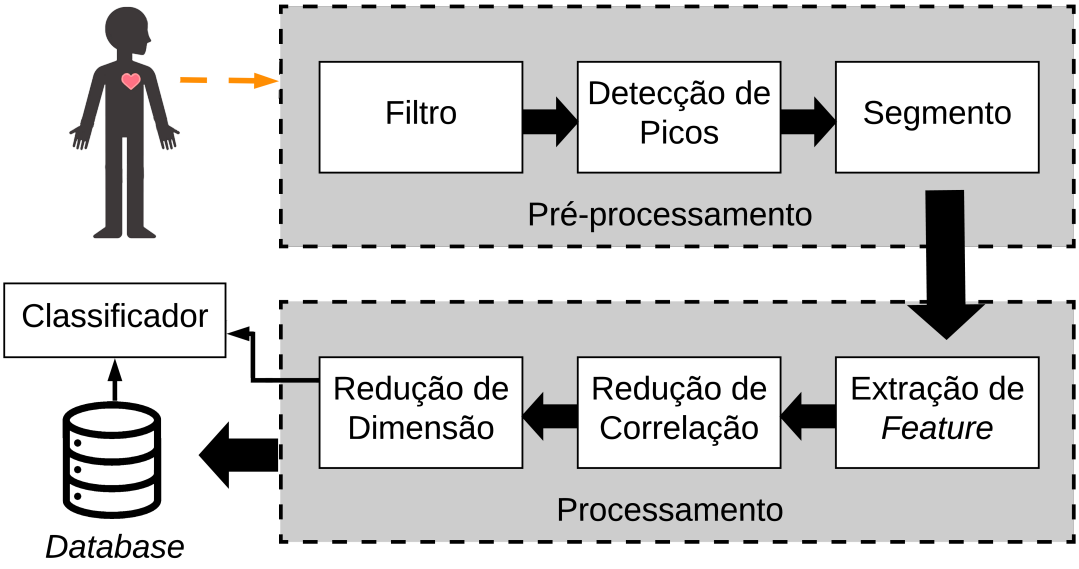


Figura 4.11: Esquema para extração de *features* de sinais vitais (Adaptado de [Karimian et al. 2017])

4.3.4.1. *Features* consideradas para sinais PPG

A fotopletismografia (PPG) é um método eletro-óptico, não invasivo, que mede o volume sanguíneo que flui através da parte do corpo humano em análise (exemplo: pulso, dedo, lóbulos auriculares, etc). Os sinais PPG refletem as ações pulsativas das artérias através da interação entre a hemoglobina oxigenada e os fótons. Acredita-se que cada pessoa tenha uma hemodinâmica e um sistema cardiovascular únicos [Yadav et al. 2018]. Por conta disso, os sinais PPG podem ser utilizados para autenticação biométrica.

Os sinais PPG são registrados através de uma combinação de LED, que emite luminosidade em uma parte do corpo, e Foto-Diodo (PD), que mede a luz absorvida pelos tecidos epiteliais. Esta combinação proporciona maior flexibilidade para o projeto de sistemas de autenticação. As medições indicam as mudanças no volume sanguíneo. Como o registro do PPG requer apenas LED e PD, ele é muito econômico, comparado aos outros traços biométricos. No contexto da biometria médica, o registro do PPG não requer nenhum tipo de gel, estímulo externo ou vários eletrodos e pode ser convenientemente registrado de praticamente qualquer parte do corpo [Yadav et al. 2018, Nakayama et al. 2019].

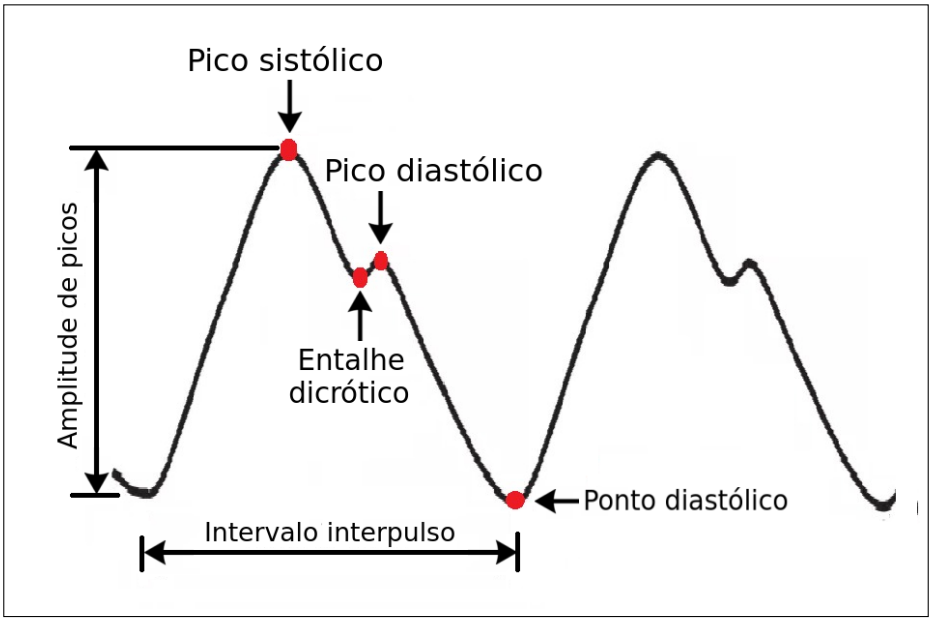


Figura 4.12: Pontos fiduciais de uma amostra de sinal PPG

Muitos trabalhos têm sido desenvolvidos para investigar a viabilidade de se aplicar sinais PPG como identificadores biológicos. Os sinais PPG apresentam pontos fiduciais como características de suas formas de onda. Esses pontos são pico sistólico, pico diastólico, entalhe dicrótico, intervalo interpulso, amplitude de picos, entre outras, como ilustrado na Figura 4.12. Os picos sistólicos representam a pressão sistólica, que é a pressão sanguínea mais alta durante a contração dos ventrículos, quando o coração está bombeando o sangue [Allen 2007]. A pressão sistólica aumenta durante a fase anacrótica do batimento cardíaco[Sarkar et al. 2016]. Os picos diastólicos representam a pressão diastólica, que é a menor pressão registrada pouco antes da próxima contração do coração, quando este está relaxado [Clark and Kruse 1990, Allen 2007]. A pressão diastólica é registrada durante a

fase catacrótica do batimento cardíaco [Sarkar et al. 2016]. Existe mais um ponto fiducial na forma de onda do sinal PPG que é o entalhe dicrótico. Ele consiste em uma ascendência secundária correspondente ao aumento transiente da pressão sanguínea quando a válvula aórtica se fecha [Politi et al. 2016].

As *features* de um sinal PPG podem ser utilizadas para identificar diferentes indivíduos, ao mesmo tempo, similares o suficiente para reconhecer uma mesma pessoa. Os sinais PPG têm diversas vantagens para autenticação de usuários quando comparados com outras abordagens biométricas. Eles possuem baixo custo de desenvolvimento e são acessíveis a várias partes do corpo humano (dedo, lóbulo da orelha, pulso ou braço) [Gu et al. 2003]. Por conta disso, muitos trabalhos concentram-se em pesquisar sobre o uso dos sinais PPG como identificadores biométricos. Muitos deles desenvolveram esquemas para sistemas de autenticação biométrica, como ilustrado na Figura 4.13. Esses esquemas geralmente apresentam as mesmas etapas, que em sua maioria são a aquisição e filtragem do sinal, tratamento de ruídos, extração de *features*, aplicação de alguma técnica de aprendizado de máquina ou estatística e a identificação propriamente dita dos indivíduos. A seguir serão descritos alguns trabalhos encontrados na literatura que abordam extração de *features* de sinais PPG.

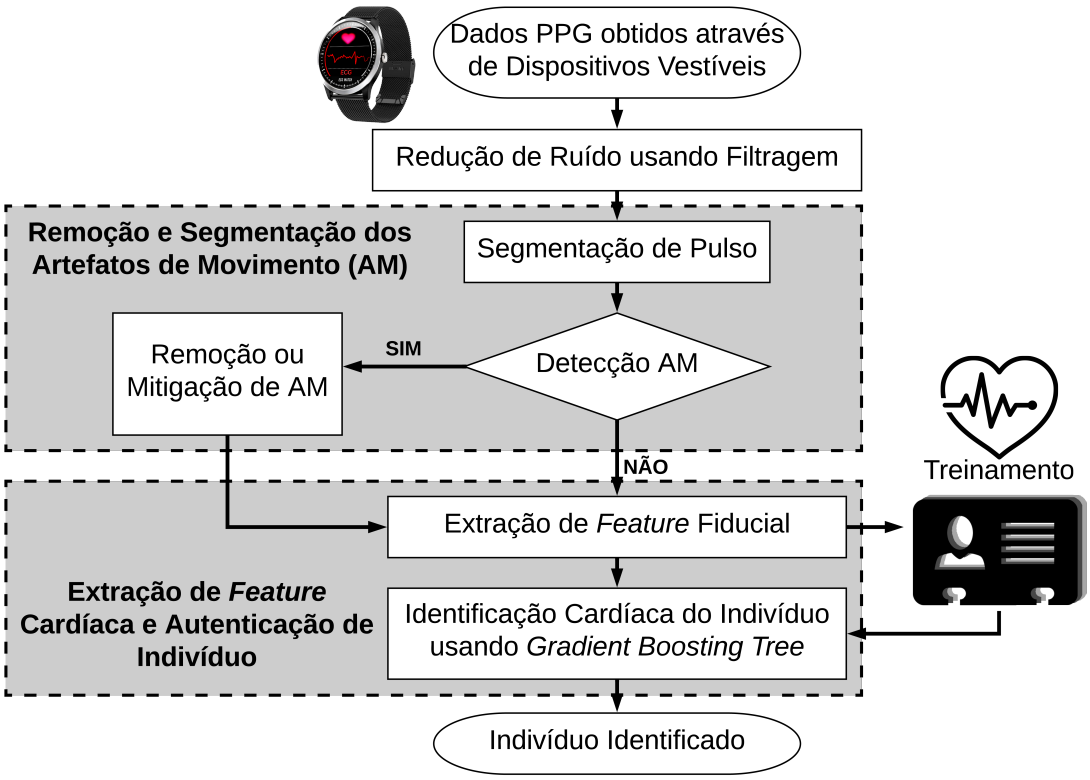


Figura 4.13: Esquema de sistemas de autenticação biométrica (Adaptado de [Zhao et al. 2018])

[Sancho et al. 2018] propuseram oferecer uma excelente discriminação entre indivíduos através da análise de um sinal PPG normal e de outro contaminado por artefato

de movimento. Eles consideraram como característica única o período de tempo dos ciclos completos do sinal PPG. [Salanke et al. 2013] também consideraram os ciclos PPG para fins de autenticação biométrica. Os ciclos foram normalizados dividindo-se a amplitude do ciclo pela amplitude do pico sistólico e, em seguida, eles foram alinhados alocando seus picos sistólicos no mesmo ponto. [Bonissi et al. 2013] adotaram como características únicas um número variável de pulsações distintas da amostra de sinal. Os autores consideraram os valores máximos de correlação cruzada entre cada batimento cardíaco e o batimento cardíaco médio. Se o valor de correlação de um batimento cardíaco for menor que um limite empiricamente estimado, o sinal relacionado é removido do *template* de *features*.

Dada a variabilidade na forma PPG em diferentes estados, a detecção fiducial no sinal PPG bruto pode ser malsucedida ou incorreta. Consequentemente, muitos pesquisadores estenderam a ideia para a primeira derivada (FD) e a segunda derivada (SD) de sinais PPG brutos e usaram pontos semelhantes em FD e SD como recursos para autenticação. Por exemplo, [Orjuela-Cañón et al. 2013] usou padrões de ataque e de pico nos sinais PPG. [Sarkar et al. 2016] demonstraram que os sinais PPG também têm o potencial de serem usados para autenticação biométrica, mostrando um exemplo da morfologia do sinal para um único batimento que compreende duas fases principais: a fase anacrótica, que significa o surgimento de a pressão sistólica e a borda ascendente do sinal, e a fase catacrótica, que reflete a diástole e a reflexão da onda a partir da periferia. O pico diastólico e o entalhe dicrótico aparecem nesta fase. A base da pulsação mostra o ponto mais baixo da diástole e o ponto inicial da sístole.

Muitos dos métodos anteriores focaram em abordagem fiducial, mas a detecção de pontos fiduciais em qualquer condição é propensa a erros [Yadav et al. 2018]. Consequentemente, muitos outros trabalhos preferiram se concentrar na abordagem não-fiducial. [Kavsaoğlu et al. 2014] analisaram o mais completo conjunto de características únicas dos sinais PPG. O conjunto é composto por quarenta características envolvendo pico sistólico, pico diastólico, entalhe dicrótico, intervalo de pulso, pico a pico, índice de aumento, índice de aumento alternativo, tempo de pico sistólico, tempo de entalhe dicrótico, tempo de pico diastólico, tempo entre os picos sistólico e diastólico, largura de pulso com semi-altura do pico sistólico, razão de área de ponto de inflexão, curva de saída de pico sistólico, curva descendente de pico diastólico, entre outros.

[Orjuela-Cañón et al. 2013] usaram uma base de dados composta por sinais de sete voluntários, sendo quatro homens e três mulheres, que ficaram em repouso por cinco minutos na posição supina. Os sinais ECG e PPG foram coletados simultaneamente através de um canal para o ECG e dois canais para o PPG. Os autores apresentaram uma proposta baseada no reconhecimento de padrões, utilizando um *Multilayer Perceptron* (MLP) para aprender as informações temporais sobre o início e o pico de pulsos. Esses pulsos estão localizados no meio do segmento. Em seguida, as janelas temporais são extraídas para treinar a rede neural. Os MLPs possuem apenas conexões *feed forward* e são treinados de forma supervisionada. Para validação dos modelos, implementou-se o método de validação cruzada *Leave One Out* (LOO). Neste caso, seis dos sete sinais foram utilizados no treinamento. Em seguida, calculou-se a validação usando apenas o sinal não incluído no treinamento. Os resultados da classificação alcançaram 98,1 % no pior dos casos.

[Kavsaoğlu et al. 2014] analisaram uma base de dados composta por 30 indivíduos

saudáveis, sendo 17 homens e 13 mulheres, que estavam sentados durante a coleta de dados. Um sinal de 15 períodos foi coletado de cada indivíduo em dois intervalos de tempo diferentes. O primeiro conjunto de dados com *features* dos primeiros sinais recebidos dos indivíduos, o segundo conjunto com *features* de sinais recebidos em um horário logo após o primeiro e o terceiro conjunto de dados sendo a combinação dos dois conjuntos anteriores. A fórmula de diferenciação fornece as três *features* dos sinais digitais unidimensionais: tanto a primeira quanto a segunda derivada devem ser zero onde a função é constante. A primeira derivada deve ser constante e a segunda derivada deve ser zero para os feixes crescentes e decrescentes. Foram obtidas identificações de 90,44%, 94,44% e 87,22% para o primeiro, o segundo e o terceiro conjunto de dados, respectivamente.

[Karimian et al. 2017] registraram sinais PPG brutos de 42 indivíduos saudáveis, os quais estavam com respiração espontânea ou controlada. Um filtro de banda passante *Butterworth* de terceira ordem com frequência de corte de 1Hz-5Hz foi empregado para reduzir o efeito de ruído. Os autores criaram segmentos PPG identificando os picos sistólicos de cada batimento cardíaco através de um algoritmo *Pan Tompkins* modificado. Os autores aplicaram *support vector machine* para aprendizado de máquina supervisionado (AMS), e *k-nearest neighbours* e *self-organization map*, para aprendizado de máquina não supervisionado (AMNS). Eles avaliaram abordagens não-fiduciais e fiduciais para extração de *features*. Os resultados demonstraram que o desempenho do AMNS é melhor, especialmente no caso de *features* fiduciais, comparado ao aprendizado de máquina não supervisionado (AMS). A abordagem não fiducial teve melhor desempenho em termos de acurácia e *equal error rate* (EER), de modo que obteve 99,5% de acurácia para AMS e 99,84% de acurácia para AMNS com EER igual a 1,31%.

Dentre as técnicas mencionadas, algoritmos baseados em PPG (para dispositivos vestíveis) mostram um grande potencial, precisando considerar as dificuldades associadas ao uso desses dispositivos. Em primeiro lugar, as gravações de PPG desses dispositivos podem frequentemente ter ruídos devido aos movimentos contínuos dos usuários. Em segundo lugar, a ocorrência de eventos nas artérias podem registrar ruídos. Assim, existe a necessidade de desenvolver algoritmos que respondam aos fatores acima de detecção de ruídos [Shashikumar et al. 2017]. Nos últimos anos, *Deep Learning* foi utilizado com sucesso no campo do processamento de sinais biométricos. Pesquisas na área de processamento de sinais vitais envolvem o estudo de sinais como ECG, eletroencefalograma (EEG) e PPG para prever ampla gama de eventos fisiológicos no corpo humano. Algumas das aplicações incluem reconhecimento de emoções [Jirayucharoensak et al. 2014], detecção de crises [Turner et al. 2014] e detecção do estágio do sono [Längkvist et al. 2012].

Deep Learning também tem sido usada para melhorar a robustez nos procedimentos atuais de monitoramento de sinais PPG em ambientes clínicos, de *e-health* e *fitness*, fazendo uso de biossensores vestíveis. Através desses dispositivos, [Jindal et al. 2016] apresentaram uma nova técnica de identificação biométrica utilizando sinais PPG por meio de *Deep Belief Networks* e *Restricted Boltzman Machines*. De acordo com [Miotto et al. 2017], as abordagens de *Deep Learning* podem ser o veículo para traduzir grandes dados biométricos em saúde humana. Entretanto, [Miotto et al. 2017] também notaram as limitações e as necessidades para desenvolvimento de métodos melhorados, especialmente em termos de facilidade de entendimento para especialistas. Portanto, estes autores sugerem o desenvolvimento de arquiteturas holísticas e significativas para unir modelos de *Deep Learning* e

interpretabilidade humana.

Alguns dos trabalhos que analisaram sinais PPG reportados neste JAI utilizaram bases de dados criadas por seus próprios autores. A maioria deles informa a quantidade e a idade dos indivíduos que participaram no processo de coleta de dados, assim como também o equipamento utilizado. Entretanto, alguns trabalhos também utilizaram base de dados públicas, disponíveis na Internet [Hatzinakos and Yadav 2019]. Sobre a extração de *features* de sinais PPG, pode-se concluir que a maioria dos trabalhos realizou duas fases: filtragem e identificação. Há várias fontes de artefatos que interferem na aquisição de sinais PPG, incluindo mudança de linha de base, artefato de movimento e respiração [Karimian et al. 2017]. Portanto, a fase de filtragem é essencial para remover/minimizar o ruído dos sinais PPG coletados. Depois da remoção de ruído, os autores utilizaram técnicas para identificar e correlacionar os usuários. As principais técnicas para identificação foram técnicas de aprendizagem de máquina (supervisionadas ou não supervisionadas), redes neurais ou até mesmo simplesmente a primeira e a segunda derivada do sinal PPG.

4.3.4.2. *Features* consideradas para sinais ECG

As batidas do coração geram ondas de polarização e despolarização nas fibras musculares. O eletrocardiograma (ECG) é feito de uma forma não evasiva, o qual representa simplesmente o registro da atividade elétrica cardíaca baseada nas diferenças de potencial resultantes [Biel et al. 2001]. Sua amostra está associada ao ciclo cardíaco, conforme ilustrado na Figura 4.14. O ECG é bastante útil para várias aplicações biomédicas, tais como a medição da taxa de frequência cardíaca, exame de ritmo das batidas do coração em busca de arritmias, diagnóstico de anormalidades do coração, reconhecimento de emoção e mais recentemente identificação biométrica.

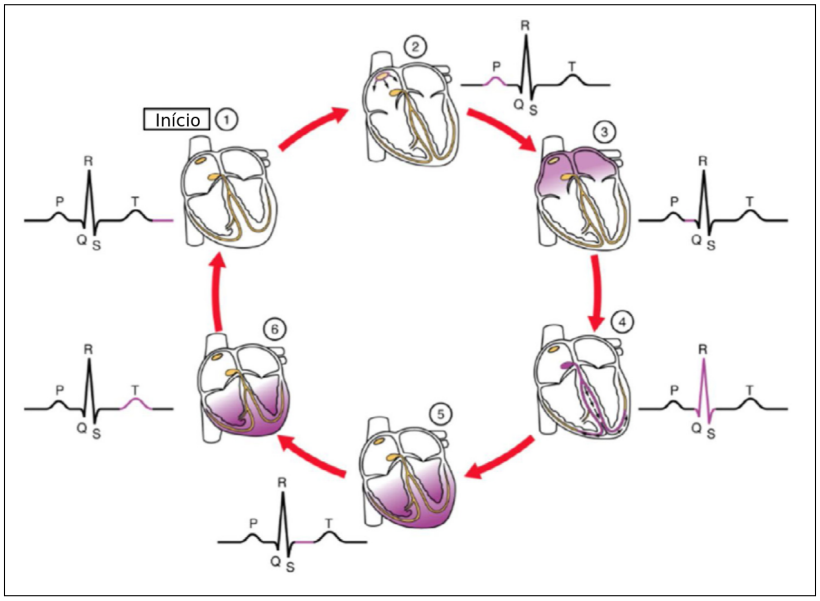


Figura 4.14: Etapas de funcionamento do coração e forma de onda captada pelo ECG

Para a extração de *features* do sinal de ECG é necessário que o sinal elétrico

coletado seja tratado previamente, passando por um processo de filtragem para retirada de ruídos, normalização e amostragem. Como explicado por [Odinaka et al. 2012], o ECG apresenta três componentes predominantes: onda P, complexo QRS e onda T como representadas na Figura 4.15. Primeiro, a despolarização do átrio gera um pulso registrado como onda P. A série de pulsos seguinte à onda P é o complexo QRS e está associada com a atividade ventricular. Finalmente, a onda T está associada à repolarização ventricular [Rezgui and Lachiri 2016]. Este complexo P-QRS-T é o mais utilizado para identificação de pessoas. Ele basicamente corresponde às localizações, durações, amplitudes e formas de onda do sinal coletado do coração (ou seja, ECG). Tipicamente, um sinal ECG possui um total de cinco deflexões principais, as ondas P, Q, R, S e T, mais uma deflexão menor, chamada de onda I, conforme descrito a Figura 4.15. A literatura apresenta três tipos de *features*: fiduciais, não fiduciais e híbridas. As *features* fiduciais extraem características no domínio do tempo das formas de onda ECG, as *features* não fiduciais aplicam uma função de transformação aos pontos característicos, e as *features* híbridas são a combinação das *features* fiduciais com as *features* não fiduciais.

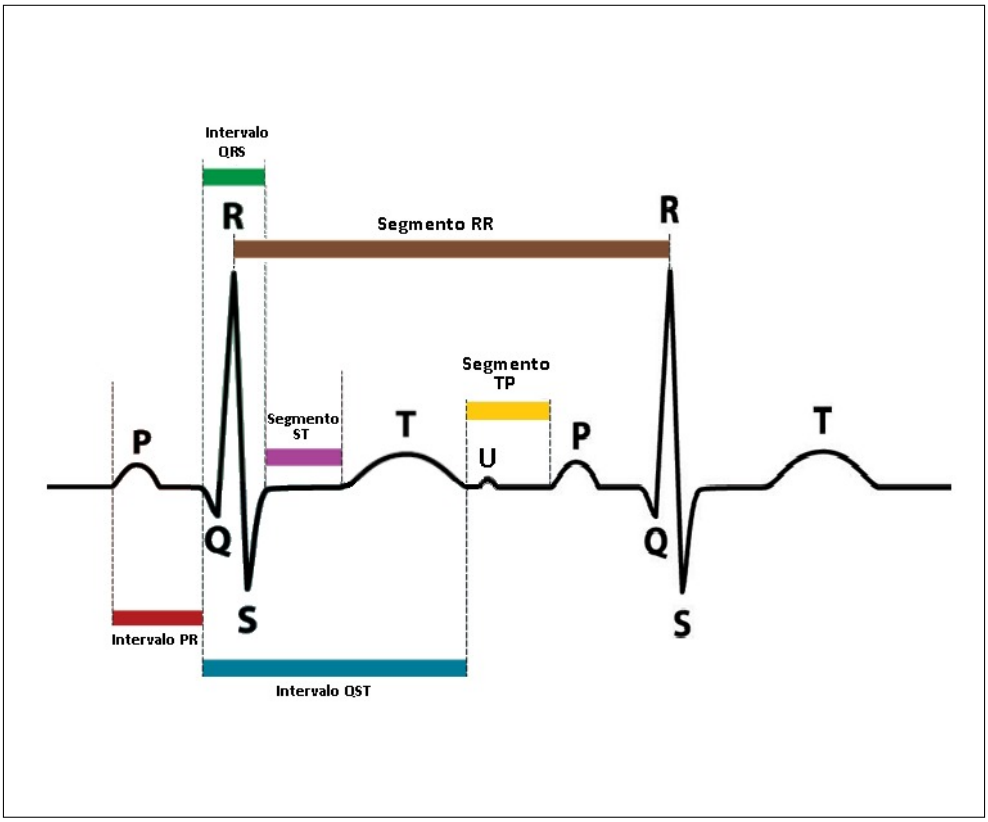


Figura 4.15: Pontos fiduciais utilizados como *features* no ECG

Segundo [Odinaka et al. 2012], há pesquisas que usam ECG como identificador biométrico: [Biel et al. 2001], [Irvine et al. 2001] e [Kyoso and Uchiyama 2001]. Estes trabalhos consideraram a hipótese de que o ECG possui informações suficientes para aplicabilidade no reconhecimento humano. Especificamente, [Odinaka et al. 2012] pesquisou cinquenta estudos dedicados à identificação humana, onde 66% dos artigos pesquisados empregaram características não-fiduciais, 26% aplicaram características fiduciais, e 8%

dos trabalhos de pesquisa usaram a abordagem híbrida. Quanto ao método de classificação, 44% dos trabalhos de pesquisa selecionaram os algoritmos *k-Nearest Neighbour* (k-NN) ou *Nearest Center* (NC), 16% implementaram Redes Neurais (ANN) e 16% utilizaram *Linear Discriminant Analysis* (LDA). Finalmente, 12% das pesquisas atingiram uma acurácia superior a 99% e 20% dos artigos pesquisados atingiram 100% de acurácia. Segundo [Odinaka et al. 2012], as características fiduciais, não-fiduciais ou híbridas não influenciam diretamente na acurácia.

[Camara et al. 2018] consideraram um cenário de uma torre de controle de tráfego aéreo. Nesse conceito, os pesquisadores assumiram que os controladores exigiam monitoramento permanente para evitar incidentes de segurança, como um intruso tentando usar o sistema de controle, um controlador assumindo a posição de um colega e a alta variação da frequência cardíaca do controlador devido a uma situação estressante. Os autores consideraram um dispositivo médico implantável para capturar o sinal de ECG e uma etapa de filtragem para limpar a corrente contínua (CC). Por simplicidade, os pesquisadores optaram por trabalhar com características não-fiduciais do ECG. Nesse sentido, o módulo de extração de recursos analisa a janela de ECG e a envia para uma Transformada *Walsh-Hadamard* (HT). [Camara et al. 2018] consideraram o HT menos computacionalmente complexo que as transformadas de *Fourier* ou *Wavelet*. O modelo de aprendizado de máquina implementou um algoritmo k-NN, usando o conjunto de dados *MIT-BIH Normal Sinus Rhythm* [Goldberger et al. 2000a]. Para a estratégia de monitoramento contínuo, o sistema atingiu 96% de acurácia.

[Zhang and Wu 2016] e [Tomlinson et al. 2019] consideraram um cenário de autenticação usando o ECG. O primeiro trabalho coleta ECG de eletrodos de dois dedos associados a um aplicativo de *smartphone*. Nos dois casos, a aplicação do sistema considera três etapas: etapa de filtragem de pré-processamento, etapa de extração de *features* e etapa de classificação. A filtragem elimina o ruído associado à linha de energia, interferência, movimento muscular e ruído de alta frequência. O módulo de detecção funciona como uma pré-etapa para a extração de *features*. Os autores em [Zhang and Wu 2016] consideraram o complexo de computação de extração de *features* não fiduciais e, assim, decidiram implementar o modelo usando formas de onda divididas por fiduciais (WF). A etapa de classificação refere-se a dois métodos biométricos: autenticação e identificação. Durante a autenticação, o sistema compara a biometria do usuário com um modelo armazenado que calcula a distância euclidiana como uma métrica de comparação. Para identificação, o sistema realiza uma classificação usando *Support Vector Machine* (SVM) e Redes Neurais (ANN). Para ambos os casos, um mecanismo de votação exigiu que mais da metade dos eleitores validassem o assunto do teste. Em comparação com outros trabalhos relacionados, a pesquisa de [Zhang and Wu 2016] atingiu 97,55% de acurácia e executou a autenticação em quatro segundos.

[Zhang et al. 2018] propuseram um sistema que combina *features* fiduciais e não fiduciais (abordagem híbrida) para aumentar a acurácia ao autenticar um grande número de indivíduos. Os autores consideraram os picos do PQRST como as principais *features*. Especificamente, os segmentos PQ, QR, RS e duração ST, as amplitudes PQ, PT e SQ determinadas por transformada *wavelet*. Para *features* não fiduciais, os autores definiram o sinal do ECG como uma matriz X, obtiveram a matriz Gramiana multiplicando $X^T X$ e finalmente obtiveram as características dos autovalores e autovetores da matriz Gramiana.

A pesquisa considerou um segundo conjunto de *features* não fiduciais como o espectro do sinal ECG gerado pela Transformada Rápida de *Fourier*. O reconhecimento do padrão ECG considerou o treinamento incremental de um algoritmo *Linear Discriminant Analysis* (LDA), usando as *features* fiduciais e não fiduciais expostas. [Zhang et al. 2018] consideraram o conjunto de dados MIT-BH que incluiu 100 amostras contendo 200 arquivos de sinal ECG por arquivo de amostra, ou seja, 20.000 sinais ECG. A pesquisa concluiu que a identificação baseada em características fiduciais e Transformada Rápida de *Fourier* apresentou baixa acurácia, na ordem de 70%-75%. Por outro lado, a implementação de uma abordagem híbrida alcançou 99%. Finalmente, os autores mostraram que o aumento do número de *features* leva a um aumento no esforço computacional, e o esquema proposto melhora a eficiência.

A seleção de *features* e a extração combinada com o aprendizado de máquina apresentam um papel relevante no reconhecimento de indivíduos. Conforme discutido por [Biel et al. 2001], as *features* do ECG contêm uma quantidade significativa de informações, mas algumas características são altamente correlacionadas. Entretanto, como discutido por [Odinaka et al. 2012], o tipo de *feature* (fiducial ou não-fiducial) não afeta a acurácia do modelo. No entanto, esta conclusão entra em conflito com a conclusão de [Zhang et al. 2018], que claramente influencia o modelo para a aplicação de *features* híbridas supostamente devido ao aprimoramento da acurácia. Portanto, é preciso conduzir a presente pesquisa por meio de simulações para concluir a melhor combinação entre o algoritmo de aprendizado de máquina *versus* o tipo de *features* no contexto de dispositivos vestíveis. Além disso, a identificação correta de um número significativo de indivíduos é uma condição *sine qua non* para a avaliação do modelo.

[Camara et al. 2018] consideraram os avanços da Internet das Coisas (IoT) e dos dispositivos médicos implantáveis da próxima geração. Nesse sentido, os autores observaram que a mineração de fluxo de dados é uma área de pesquisa promissora para lidar com a autenticação contínua. Especificamente, o ECG muda com o tempo e, portanto, o sistema precisa de atualizações. Nesse contexto, a pesquisa considera ampliar os conceitos para investigar outros sinais fisiológicos, como PPG e EEG (eletroencefalograma).

[Zhang and Wu 2016] provaram o conceito de usar um *smartphone* para reconhecimento de identidade baseado em um sinal ECG. No entanto, o teste considerou três conjuntos de dados ECG abertos e, portanto, os resultados práticos podem ser diferentes. Considerando que os dispositivos portáteis, como o *Apple Watch* [Apple 2019], já fornecem ECG, outras empresas seguirão a tendência em breve. Nesse sentido, o uso do ECG como método de autenticação se tornará frequente. Assim, o conceito de [Zhang and Wu 2016] precisa de mais investigações.

[Rezgui and Lachiri 2016] realizaram pesquisas sobre a aplicação do ECG para identificação biométrica. Na metodologia proposta, os autores consideraram um detector QRS chamado ECGPUWAVE como extrator de *features*. Este *software* implementa o algoritmo proposto por [Pan and Tompkins 1985] com melhorias propostas por [Laguna 1990]. Por outro lado, [Venkatesh and Jayaraman 2010] usaram *Dynamic Time Warping* (DTW), *Fisher Linear Discriminant Analysis* (FLDA) e *K-nearest neighbour* (k-NN). A extração de *features* considerou um sinal ECG filtrado com um minuto de duração. Primeiramente, o método identificou o complexo QRS, o intervalo R-R, a frequência cardíaca, o desvio

padrão e o número de picos na amostra considerada. Em uma segunda análise, o método identificou as ondas P, T, Q e S. Finalmente, após a redução de dimensão, o vetor de *features* considerou os intervalos de onda P, de T, de ST, o de PR interno, de QRS e de QT.

[Belgacem et al. 2012] consideraram Transformada *Wavelet* Discreta (TWD) para extração de *features* e *Random Forest* (RF) como método de classificação para identificar indivíduos com base em sinais ECG. *Wavelets* são usadas para representar sinais e outras funções de maneira normalizada, isto é, com média zero. Primeiramente, o procedimento de extração de *features* considerou um ciclo cardíaco médio obtido pela sobreposição do ciclo de cada indivíduo a partir da amostra de dados. Finalmente, as *features* são obtidas como os coeficientes do TWD dos batimentos médios. Um método de extração de *features* similar, *i.e.*, Transformada *Wavelet* Discreta, foi usado por [Dar et al. 2015]. Entretanto, os métodos mais recentes diferem em propor uma estratégia de redução de *features* aplicando *Greedy Best First Search* para subconjuntos de um subconjunto de características altamente correlacionadas.

[Karpagachelvi et al. 2010] avaliaram 17 artigos sobre técnicas de extração de *features* de ECG. Neste contexto, a transformada de *wavelet* e suas variações, tais como *wavelets* ortogonais e bi-ortogonais, *wavelets* discretas e *wavelets* quadráticas, receberam atenção de vários trabalhos. Além disso, métodos propondo algoritmos inovadores para detecção de onda P, complexo QRS, onda T e a detecção do intervalo R-R estavam em evidência no *survey* de [Karpagachelvi et al. 2010]. Métodos estatísticos, filtros combinados, coeficientes de *cepstrum* e teoria do caos eram menos frequentes.

Para [Page et al. 2015], o ECG, juntamente com um marcador biométrico secundário (ex.: impressão digital), desempenham um papel fundamental na segurança de dispositivos vestíveis. Portanto, estes autores, através de técnicas de *Deep Learning*, implementaram um sistema de reconhecimento de padrões de ECG, para fins de autenticação biométrica, confiável, robusto e rápido, utilizando redes neurais para identificar segmentos QRS complexos do sinal de ECG e, em seguida, executar a autenticação do usuário nesses segmentos. [Wieclaw et al. 2017] também aplicaram redes neurais profundas (DNN) para identificação humana com base no sinal de ECG bruto. Os resultados do estudo apontaram que o número de usuários identificados, bem como o número de neurônios e camadas ocultas, têm um impacto significativo na precisão da identificação em comparação a outros fatores. A precisão da identificação pode ser potencialmente melhorada usando outra solução, por exemplo, outras arquiteturas DNN. Isso é reforçado por [Labati et al. 2018], que afirmam que os sistemas biométricos baseados em ECG são geralmente menos precisos do que os baseados em outras características fisiológicas. Para [Labati et al. 2018], métodos como as CNNs podem extrair automaticamente características distintas de sinais ECG e demonstrar sua eficácia para outros sistemas biométricos. Para tal fim, estes autores apresentaram Deep-ECG, uma abordagem biométrica baseada em CNN para sinais de ECG, sendo o primeiro estudo na literatura que utiliza uma CNN para biometria de ECG. O Deep-ECG extrai *features* significativas de uma ou mais derivações usando uma CNN profunda, obtendo uma precisão notável para identificação, verificação e nova autenticação periódica.

Com base na análise de tais trabalhos, cujo objetivo é analisar o uso de sinais ECG como identificadores biométricos, geralmente buscam detectar a onda P, o complexo QRS,

a onda T e o intervalo R-R, através de técnicas de aprendizado de máquina e transformada *wavelet*. Também são utilizados métodos estatísticos e filtros combinados, porém estes com menor frequência, pois alguns trabalhos têm buscado elaborar algoritmos inovadores para detecção e extração das *features* dos sinais ECG.

4.3.5. Discussão

O monitoramento de sinais ECG pode ser usado como uma ferramenta essencial para monitorar condições de saúde de pacientes, assim como para identificação de indivíduos. As principais limitações dos sistemas biométricos estão relacionadas com: *i*) condições ambientais variáveis (*i.e.*, ruído, mudanças na iluminação, posicionamento da impressão digital ou da face em relação ao sensor), as quais afetam fortemente a acurácia do sistema, especialmente quando a aquisição não é realizada em condições restritas; *ii*) grandes variações intra-classe causadas pela aquisição em diferentes condições ou efeitos de envelhecimento; *iii*) não-universalidade de alguma credencial biométrica, devido a doença ou deficiência, *iv*) ataques de fraudes que são realizados falsificando um traço biométrico e, em seguida, apresentando essas informações falsificadas ao sistema biométrico [Lumini and Nanni 2017].

Nesse contexto, interferências afetam fortemente a performance dos sistemas biométricos existentes [Nakayama et al. 2019]. Por exemplo, um sistema pode rejeitar as amostras fornecidas devido à baixa qualidade ou imagens ruidosas [Lumini and Nanni 2017]. Desempenho de reconhecimento é uma medida de o quanto o sistema está apto para combinar corretamente as informações biométricas de uma mesma pessoa. Em alguns casos, o desempenho de uma simples modalidade biométrica é insuficiente e, por isso, combinar biomarcadores tem se tornado um interesse acadêmico. Essa combinação é chamada *"fusão biométrica"*, a qual pode ser classificada em dois grupos: sistemas biométricos unimodais e sistemas biométricos multimodais.

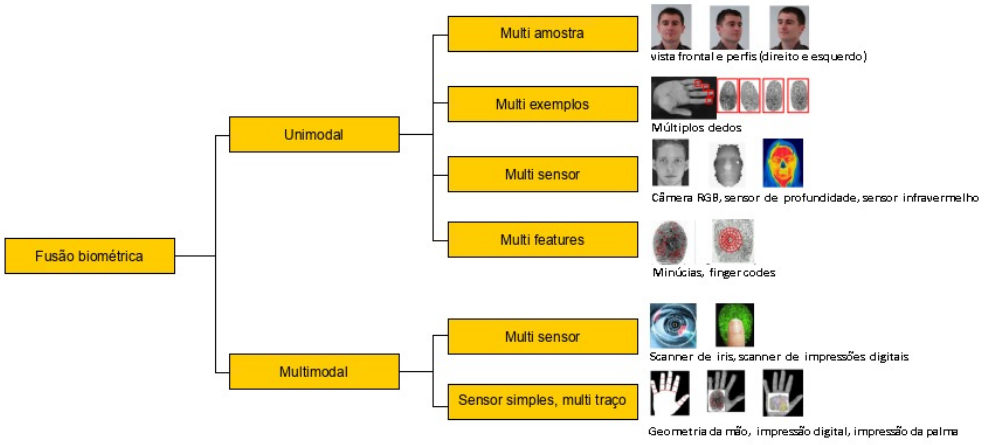


Figura 4.16: Fontes possíveis de informação em um sistema com fusão biométrica (Adaptado de [Lumini and Nanni 2017])

Os sistemas biométricos multimodais adquirem e usam vários traços biométricos para autenticação de pessoas. Algumas possíveis amostras de fontes de informação, em um sistema unimodal e multimodal, são apresentadas por [Lumini and Nanni 2017]. A

multimodalidade tem diversas vantagens sobre a unimodalidade, devido à sua capacidade de reduzir a taxa de falha de registro (do inglês *Failure-to-Enroll Rate* (FTER)) e a taxa de falha de captura (do inglês *Failure-to-Capture Rate* (FTCR)) e garantir uma cobertura populacional suficiente. Por exemplo, este processo estima que 2% da população pode não estar apta para fornecer uma impressão digital, devido a condições médicas/genéticas/acidentais/temporárias. Os sistemas multimodais são também resistentes a ataques fraudulentos porque é mais difícil para o atacante fraudar múltiplas fontes biométricas simultaneamente. O futuro dos sistemas biométricos está relacionado aos sistemas multimodais. A Figura 4.16 representa algumas possíveis fontes de informação em um sistema de fusão biométrica.

Tabela 4.1: Comparação entre sinais biométricos

Sistema biométrico	**Prós**	**Contras**
Mão	O pequeno *template* torna os atributos da região da mão uma boa opção para muitos tipos de aplicações, comparados com assinaturas biométricas mais complexas.	Pode comprometer facilmente a eficácia do sistema.
Ocular	É mais preciso, altamente confiável, bem protegido, estável e quase impossível de forjar as assinaturas biométricas.	Alto custo de sensores de imagem; Requer cuidados com fontes de luz ambiente.
Facial	Aquisição de imagem não intrusiva e sem contato, resultando em maior aceitação pública.	Esses sistemas não podem garantir identificação confiável na presença de ruídos.
ECG	*Features* ECG não podem ser copiadas ou manipuladas.	É necessário o uso de vários eletrodos durante a aquisição dos sinais ECG, causando desconforto no indivíduo. Muda de acordo com a frequência cardíaca (repouso ou exercício)
PPG	Sua coleta requer apenas LED e Foto-Diodo, ou seja, não requer nenhum tipo de gel, estímulo externo ou vários eletrodos; pode ser coletado de qualquer parte do corpo; é econômico, comparado aos outros traços biométricos.	Geralmente é prejudicado por ruídos durante a aquisição (artefatos de movimento, movimentos de sensor, respiração, contração ventricular prematura e luz ambiente); muda junto com a frequência cardíaca (repouso ou exercício); as emoções influenciam o funcionamento do sistema nervoso autônomo e do coração.

Uma outra discussão pertinente é em relação para as etapas de separação de *features* e classificação. Alguns trabalhos separam tais etapas e realizam de uma forma desacoplada, como vários artigos que usaram técnicas mais tradicionais de aprendizado de máquina

supervisionadas. Enquanto outros trabalhos já tratam ambas as etapas e executam de uma vez só, de forma integrada, como vistos nas redes neurais como *Convolutional Neural Network* e *Deep Learning*. Ambas as estratégias são válidas, no entanto separar as etapas permite ter uma maior discussão e entendimento sobre o próprio motivo de tal algoritmo classificar um sinal biométrico. Quando é usada uma integração de etapas por aplicação de redes neurais ou *Deep Learning*, nota-se que estes algoritmos trazem uma resposta, mas que continuam sendo caixas pretas, o que inviabiliza um pouco algumas afirmações como: Qual foi o processo para chegar a resposta? Ou quais *features* são mais úteis para identificar alguém?

A Tabela 4.1 mostra uma comparação entre os sistemas biométricos, destacando os prós e contras do uso de cada um deles. O sistema biométrico que faz uso da mão e o que faz uso facial, apesar de serem não intrusivos, não passam muita segurança quanto a probabilidade de serem forjados. Já o sistema que considera os olhos como identificador biométrico, apesar de ser altamente confiável e protegido, acaba sendo custoso devido aos sensores de imagem exigidos. Entre os sinais ECG e PPG, apesar de o PPG ser mais fácil e mais confortável de ser coletado, ele é mais prejudicado por ruídos durante a aquisição que o ECG. Sinais ECG são também mais fáceis de serem interpretados, de acordo com a natureza da sua forma de onda. Por este fato, este sinal será o foco principal deste JAI.

4.4. Biometria: Aspectos Técnicos de Aprendizado de Máquina

Nessa seção serão abordadas as diferentes técnicas de aprendizado de máquina utilizadas em sistemas de biometria. O objetivo é mostrar que essa é uma área de pesquisa com muitas oportunidades e uma grande variedade de técnicas que, quando dominadas, podem ser utilizadas em outras aplicações além da biometria.

4.4.1. *Artificial Neural Networks* (ANN)

As redes neurais artificiais (ANN) são algoritmos de aprendizado de máquina capazes de classificar dados lineares e não lineares. A arquitetura típica é inspirada nas redes neurais biológicas e apresenta um determinado número conectado de neurônios dispostos em diferentes camadas. A estrutura de dados representa um neurônio artificial, geralmente configurado com ponteiros conectados a outros neurônios e um valor de peso (número real) para ponderar cada conexão.

Multilayer Perceptron (MLP) refere-se a uma rede neural configurada com um número variável de neurônios dispostos em uma camada de entrada, uma ou mais camadas ocultas, e uma camada de saída, como vistos na Figura 4.17. Especificamente, o MLP propaga um estímulo injetado nos neurônios na camada de entrada, através dos neurônios conectados nas camadas ocultas, e reflete na camada de saída. *Back propagation* é uma técnica de treinamento comum para o MLP, o qual é executado em duas fases:

- **Processamento Direto:** Inicialmente, cada neurônio recebe um valor de peso fixo. O algoritmo transfere o sinal injetado da camada de entrada para a camada de saída usando os pesos para ponderar o resultado.
- **Processamento Reverso (*Back Propagation*):** O algoritmo calcula o erro entre as saídas obtidas e as saídas desejadas, e retro-propaga o erro na rede ajustando os

pesos do neurônio no caminho de volta.

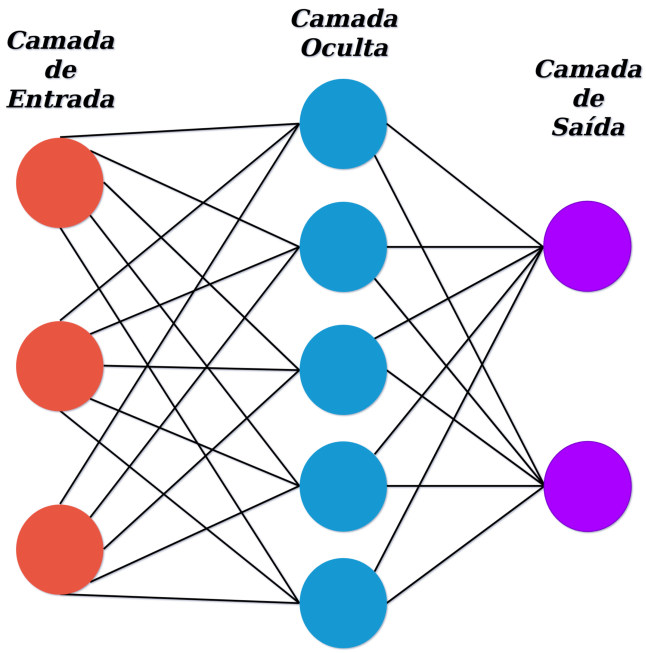


Figura 4.17: Exemplo de rede neural

O processo de treinamento repete as fases de processamento direto e reverso por um determinado número de iterações. Cada iteração se concentra em minimizar o erro entre a saída real e a saída desejada. O processo de treinamento termina quando o erro cai abaixo de um determinado limite ou quando o algoritmo atinge o número máximo de iterações estabelecidas.

Convolutional Neural Network (CNN) é um tipo de rede neural artificial (ANN) que apresenta algumas características distintas. CNN apresenta uma ou mais camadas de unidades de convolução, que reduz as unidades em mapeamentos muitos-para-um. Essa redução de parâmetros torna o modelo menos complexo, e tais unidades de convolução ao juntar algumas das unidades permite a criação de pequenas vizinhanças que compartilham informações. Devido a sua menor complexidade, CNN acaba precisando de menos amostras, o que torna o tempo de treinamento menor e deixa o modelo mais rápido.

Deep Learning é uma técnica de rede neural que consiste em transformar um dado de entrada em várias camadas de representações abstratas. Quanto mais camadas tiver na rede, maior será o número de *features* que vai aprender. A camada de saída reunirá todas estas e *features* e fará a previsão. Um dos impedimentos para se usar *deep learning* é que dependendo do número de neurônios, aumenta também o número de *features* e consequentemente a necessidade de aumentar a base de dados de treinamento.

4.4.2. *k-Nearest Neighbor* (k-NN)

k-NN classifica os vetores de características de acordo com os rótulos das amostras de treinamento mais recentes no espaço de recursos. Para um vetor de característica desconhecida, as distâncias deste vetor para todos os vetores no conjunto de treinamento

são calculadas usando uma medida de distância, *e.g.*, distância euclidiana, entre um ponto de dados particular e seus *k*-vizinhos. Em seguida, um vetor de recurso desconhecido é atribuído à classe na qual as amostras *k* mais próximas pertencem. Assim, um tipo de abordagem por maioria de votos é aplicado. O valor de *k* é um inteiro positivo e é conhecido por ser um fator que influencia fortemente a precisão da classificação.

k-NN tem a vantagem de não fazer uma suposição inicial sobre o conjunto de dados, pois apenas agrupa os pontos de dados baseados em uma vizinhança. No entanto, o algoritmo armazena todos os dados de treinamento e só libera-os quando todos os dados são classificados. Normalmente, mas nem sempre, o algoritmo consegue uma melhor precisão com *k* valores mais altos. O algoritmo k-NN herda formas de tratar características de aplicações, como previsão meteorológica [Yesilbudak et al. 2017], detecção de quedas de idosos [Tsinganos 2017], detecção de crime [Tayal et al. 2015], além de um uso amplo na maioria dos problemas de reconhecimento de padrões, como também é empregado em alguns estudos recentes de classificação de ECG ou de detecção de convulsões epilépticas [Shanir et al. 2017].

4.4.3. *Support Vector Machine* (SVM)

SVM é uma ferramenta amplamente usada para resolver problemas de classificação binária devido ao seu excelente desempenho de generalização. A ideia principal do SVM é encontrar uma margem máxima entre os dados de treinamento e o limite de decisão. Os vetores de suporte, que são as amostras de treinamento mais próximas do limite de decisão que são usados para a maximização da margem. O SVM pode ser considerado como um classificador linear ou não linear de acordo com o tipo de sua função *kernel*. Enquanto uma função de *kernel* linear torna o SVM um classificador linear, outras funções do *kernel*, como base radial de *gaussian*, polinômio e sigmoide, fazem dele um classificador não linear. O SVM é utilizado na maioria dos estudos de classificação do ECG.

4.4.4. *Naïve Bayes* (NB)

NB é um método de classificação probabilístico e estatístico baseado nas regras do Teorema de *Bayes*. A definição do teorema deriva-se da definição probabilística condicional [Goodfellow et al. 2016], que estabelece a probabilidade de um evento *A* de ocorrer baseado em uma ocorrência prévia de um evento *B*, como mostrado na Eq. 1. Esta teoria é uma abordagem estatística fundamental na qual a ideia por trás é que, se a classe for conhecida, os valores dos outros recursos podem ser previstos. No caso em que a classe não é conhecida, a regra de *Bayes* pode ser usada para prever o rótulo da classe de acordo com os valores de recurso fornecidos.

$$P(A|B) = \frac{P(A) * P(B|A)}{P(B)} \tag{1}$$

Onde,

- $P(A|B)$, é a probabilidade condicional de B ocorrer sabendo que A já ocorreu.
- $P(A)$, é a probabilidade do evento A de acontecer.

- $P(B|A)$, é a probabilidade condicional de A ocorrer sabendo que B já ocorreu.
- $P(B)$, é a probabilidade de evento B de ocorrer. pode ser tanto descoberto a priori ou calculado por $P(B) = \sum_A P(B|A) * P(A)$.

O algoritmo inicializa as probabilidades para as variáveis de resultado e as ajusta em cada interação baseada no que aconteceu com as outras variáveis do conjunto de dados. Em classificadores *bayesianos*, os modelos probabilísticos dos recursos são criados para prever o rótulo de classe de uma nova amostra. Eles são um dos métodos mais utilizados para problemas de reconhecimento de padrões.

4.4.5. *Bagging*

Também chamado de *Bootstrap Aggregating*, é um algoritmo de *ensemble* metaheurístico. Como modelo de *ensemble*, ele tenta diminuir a variância das diferentes classes, ajudando assim a fugir de problema do *overfitting* (sobre-ajuste). Para os métodos de *ensemble* como o *bagging*, quanto maior a diversidade do conjunto de dados, melhor será seu desempenho. Apesar de ter sido criado para evitar *overfitting*, há outros algoritmos bem mais aleatórios, como o RF, mais eficazes nesse processo.

4.4.6. *K-means clustering*

K-Means é um clusterizador que divide n-instâncias do conjunto de dados em *k-clusters*. A associação de cada dado é feita de acordo com a distância mais próxima do centro de cada *cluster*. Portanto, cada *cluster* agrega os conjuntos de dados mais próximos. No final, o *K-Means* divide todo o conjunto de dados usados em um diagrama de Voronoi. É importante ressaltar que *K-Means* é uma técnica não supervisionada, diferente de todos os outros modelos de aprendizado de máquina apresentados aqui. Por sua natureza não supervisionada, ele não precisa de pontos rotulados. Portanto, *K-Means* é ótimo para ser usado em cenários onde não sabemos muito a respeito do conjunto de dados processado, sendo assim muito útil para utilizá-lo inicialmente em grandes *datasets* como demografia da população, tendências nas redes sociais, detecção de anomalia, entre outros.

4.4.7. *Decision Tree (DT)*

A DT tornou-se um popular método de classificação de aprendizado de máquina devido à sua versatilidade para aplicações em muitos problemas, desde a identificação de objetos até diagnósticos médicos como análise de digitais, íris ou até ECG. O conceito é usar a estrutura em árvore para dividir as *features* em classes diferentes com base em critérios probabilísticos e em limites numéricos. *features* ou atributos definem a classe. As estruturas de DT são chamadas de árvores de classificação ou regressão. Enquanto as folhas das árvores de classificação representam rótulos de classe, as folhas das árvores de regressão representam valores contínuos. Existem muitos algoritmos para desenvolver um DT, como ID3 (*Iterative Dichotomiser 3*), C4.5 (alternativa para ID3), *Cart* (Árvore de Classificação e Regressão) e *Chaid* (*Chi-Squared Automatic Interaction Detector*).

DT tem a vantagem de ser fácil de implementar e interpretar quando comparado a outros métodos de classificação. Entretanto, dependendo das *features* escolhidas e da forma como os dados são divididos na árvore, o modelo pode perder a capacidade de

generalização devido ao *overfitting*. Há duas maneiras comuns de lidar com o *overfitting*: limitar o número de divisões ou permitir a divisão apenas se houver um número mínimo de pontos de dados na ramificação da árvore.

Um aspecto essencial da configuração do DT é como definir a importância de cada *feature* para maximizar os resultados da classificação. O algoritmo pega a característica que melhor representa a classe e a posiciona na raiz. Existem alguns índices propostos na literatura para identificar as características mais relevantes: Coeficiente de *Gini* e Índice de Entropia. o Coeficiente de *Gini* é uma medida da importância da variável para o conjunto de dados e o índice de entropia é uma medida de incerteza associada com a variável. A plataforma *Knime* usou a metodologia discutida por [Shafer et al. 1996] para implementar DT usando o Índice de *Gini*, calculado de acordo com a Eq. 2.

$$Gini(s) = 1 - \sum_{j} p_j^2 \tag{2}$$

onde,

- s representa o conjunto de dados
- p_j representa a frequência relativa da classe j no conjunto de dados s

Além das abordagens de árvores de decisão comuns, existem algumas estruturas de árvores de decisão mais específicas que são usadas frequentemente para classificação de ECG. Uma abordagem mais complexa é a utilização da floresta aleatória, onde várias árvores de decisão são treinadas com subconjuntos de dados e será explicada a seguir.

4.4.8. *Random Forest* (RF)

Um conjunto de árvores de decisão (DT) pode receber a nomeação de RF, o qual treina cada DT com dados selecionados aleatoriamente. Essa metodologia garante que cada árvore seja ligeiramente diferente uma da outra. Assim, cada árvore pode retornar um resultado distinto para um conjunto de dados. O algoritmo de RF classifica os dados com base em um sistema de votação envolvendo os resultados de árvores individuais, como visto na Figura 4.18. O sistema de votação pode calcular o voto direto ou ponderado. Especificamente, o voto direto conta quantas árvores classificaram uma determinada *feature* sob uma classe específica. A votação ponderada retorna a proporção de elementos pertencentes a uma determinada classe.

O RF executa melhor que DT em dois aspectos críticos: detecção de anomalia e *overfitting*. Devido ao processo de treinamento, os *outliers* estarão presentes em algumas das árvores, mas não em todas elas. Assim, o sistema de votação garante que os resultados anômalos sejam isolados. O sistema de votação também minimiza o efeito do *overfitting* em relação ao DT individual. No entanto, tanto RF quanto DT tem problemas para extrapolar dados. Especificamente, os valores de atributo no conjunto de validação devem estar dentro dos limites de valor do conjunto de treinamento. Os atributos não treinados ou fora do limite levam a resultados imprevisíveis quando incluídos no conjunto de validação.

O algoritmo de RF aceita que o número de árvores cresça como um parâmetro configurável. Não há um melhor valor e o limite deve ser a capacidade de armazenamento

para salvar a DT. No entanto, um número maior de árvores de decisão não reflete necessariamente nos resultados da classificação. Uma abordagem é começar com poucas árvores e aumentar gradualmente seu número até que os benefícios não valham os aumentos.

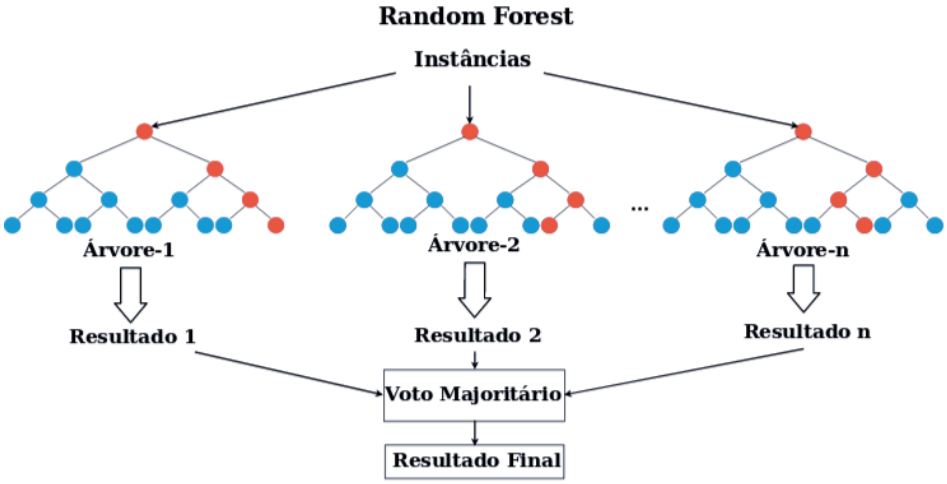


Figura 4.18: *Random Forest*

4.5. Um Estudo de Caso: na prática

Como forma de validar a utilização de sinais biométricos para autenticação, será utilizado um dos vários bancos de dados públicos disponíveis na base do site *Physionet*[1] [Goldberger et al. 2000b]. A maioria das bases de dados foi coletadas com objetivos clínicos, avaliando pessoas com problemas cardíacos e pessoas saudáveis. Para a parte prática deste JAI, será considerado o seguinte roteiro de atividades:

1. **Escolha da Base de Dados:** O participante analisará as bases públicas disponíveis no *Physionet*. Serão observados os desafios que o pesquisador enfrentar ao trabalhar com dados biométricos. Para a validação dos modelos desenvolvidos é preciso escolher uma base de dados com uma quantidade suficiente de amostras. Bases com poucas instâncias podem nos levar a conclusões divergentes daquelas encontradas em aplicações reais, com milhares ou milhões de amostras. No caso de uma base com muitas amostras, ainda será necessário avaliar algumas questões como: qual é a qualidade dessas amostras? Como o sinal foi capturado? Qual é a amostragem do sinal ou sua resolução (casas decimais)? Há ruído no sinal capturado?

2. **Seleção das *features* e aplicação de técnica de aprendizado de máquina:** Serão revisados de modo prático as vantagens e desvantagens das tecnologias de aprendizado disponíveis. Aspectos como quantidade dos dados e características das *features* utilizadas podem influenciar em um melhor ou pior resultado com uma determinada técnica. Será realizado comparativos entre resultados obtidos através de diferentes técnicas. Há ferramentas abertas como o *knime* que permitem a execução de diversas técnicas de aprendizado de máquina sobre a mesma base de dados, possibilitando

[1] Banco de dados de sinais biométricos, https://physionet.org

a definição de qual técnica pode fornecer o melhor resultado para um determinado *dataset*.

3. **Avaliação dos resultados:** Após a escolha do *dataset*, das *features* que serão analisadas e da execução do algoritmo de aprendizado de máquina, serão estimuladas avaliações dos resultados encontrados. Variações na quantidade de *features* ou utilização de *datasets* diferentes podem nos levar a resultados diferentes. É necessário entender o que motiva esse tipo de resultado. Uma análise estatística e de vários cenários são essenciais para a validação do sistema. Figura 4.19 resume este roteiro de atividades.

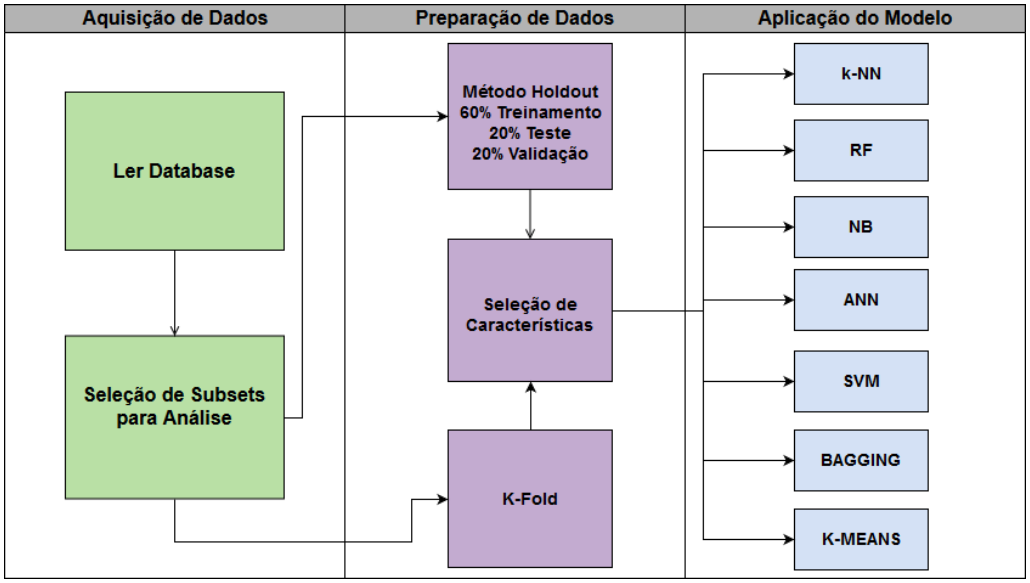


Figura 4.19: Roteiro de atividades

4.5.1. Características das Base de dados

O conjunto de dados usado foi proveniente de dois *datasets*, provenientes do Physionet: *Stress Recognition in Automobile Drivers*[2] (Reconhecimento de Stress em Motoristas de Automóveis) e *ECG-ID Database*[3] (Base de Dados ECG-ID). Ambos *datasets* serão usados na etapa de avaliação. Mais especificamente:

1. ***Dataset I - Stress Recognition in Automobile Drivers*:** O Reconhecimento de Stress em motoristas de automóveis [Healey and Picard 2005] é um *dataset* usado para tentar identificar o grau de stress de motoristas dirigindo. A coleta foi feita com 24 motoristas saudáveis, por pelo menos 50 minutos cada. O *dataset* apresenta uma coleção de sinais biométricos distintos disponíveis, como o ECG, EMG, respiração e resposta galvânica da pele (medidos nas mãos e pés). A coleta foi realizada enquanto os motoristas seguiam uma rota pré-determinada de rodovias na região

[2] *Dataset I - Stress Recognition in Automobile Drivers*, https://physionet.org/pn3/drivedb/
[3] *Dataset II* - ECG-ID, https://physionet.org/pn3/ecgiddb/

Tabela 4.2: Todas as 30 features mapeadas pelo Siemens Megacart

N.	*Features*/Características
1	início da onda P
2	duração da onda P (ms)
3	início da onda QRS
4	duração da onda QRS (ms)
5	duração da onda Q (ms)
6	duração da onda R (ms)
7	duração da onda S (ms)
8	duração da onda R' (ms)
9	duração da onda S' (ms)
10	duração da onda P+ (ms)
11	deflação da onda QRS (ms)
12	amplitude da onda P+ (μV)
13	amplitude da onda P- (μV)
14	amplitude de pico a pico da onda QRS (μV)
15	amplitude da onda Q (μV)
16	amplitude da onda R (μV)
17	amplitude da onda S (μV)
18	amplitude da onda R' (μV)
19	amplitude da onda S' (μV)
20	amplitude do segmento ST (μV)
21	amplitude do segmento 2/8 ST (μV)
22	amplitude do segmento 3/8 ST (μV)
23	amplitude da onda T+ (μV)
24	amplitude da onda T- (μV)
25	área da onda QRS (μV * ms)
26	morfologia da onda T [-2,2]
27	existência de corte da onda R
28	grau de confiança da onda Delta [0,100]%
29	inclinação do segmento ST [-90,90] graus
30	início da onda T

metropolitana de Boston. O principal objetivo do estudo foi investigar a viabilidade de reconhecimento automatizado de estresse baseado nos sinais registrados. O resultado demonstrou que a maioria dos motoristas estudados apresentou uma boa correlação entre o nível do registro galvânico e métricas do batimento cardíaco com o nível de stress das pessoas. O *dataset* está dividido nos 2 experimentos realizados por [Healey and Picard 2005].

2. ***Dataset II* - ECG-ID** O outro *dataset* foi uma investigação sobre a possibilidade de identificar pessoas a partir do sinal biométrico do ECG. O estudo envolveu 90 voluntários e apresenta 310 gravações distintas de voluntários com idades variando entre 13 e 75 anos. Devido a natureza ruidosa da coleta do sinal ECG, o autor do *dataset* disponibilizou 2 tipos de sinais: sinal original e sinal filtrado para análise.

4.5.2. Selecionando *features*

Tabela 4.2 reúne as principais *features* presentes nos sinais ECG, ambos disponíveis nos dois *datasets* que serão trabalhados.

4.5.3. Métricas de desempenho

A escolha das métricas foi feita de acordo com os resultados apresentados em uma matriz de confusão para avaliar o desempenho dos algoritmos. A matriz de confusão tabula os resultados previstos em relação às observações dos resultados reais, como mostra a Tabela 4.3. Com base na matriz de confusão, tem-se 2 linhas e 2 colunas. A primeira coluna apresenta os valores negativos (sejam os verdadeiros e os falsos) e a segunda coluna apresenta os valores positivos (sejam os verdadeiros e os negativos). A primeira linha apresenta os valores reais observados negativos (indicados como verdadeiros negativos e falsos positivos). A segunda linha apresenta os valores, de fato, positivos (indicados como falsos negativos e verdadeiros positivos).

Tabela 4.3: Exemplo de uma matriz de confusão

		Valores Preditos	
		A	B
Valores Reais	A	verdadeiro negativo (TN)	falso positivo (FP)
	B	falso negativo (FN)	verdadeiro positivo (TP)

A escolha de cada métrica depende de acordo com o que quer avaliar. A matriz de confusão oferece uma forma visual para enxergar como deriva cada métrica de desempenho que será usada na seção prática. Todos os algoritmos serão avaliados. As métricas de desempenho são acurácia, *recall*, precisão e *F1 Score*. Todas estas métricas avaliam quantitativamente o quanto cada classificador atingiu durante os testes e valores maiores representam melhores desempenhos.

A acurácia é aplicada para medir o desempenho de cada classificador, o qual tem sido usada na avaliação dos algoritmos de aprendizagem de máquina. A acurácia calcula a porcentagem de acertos que o classificador alcançou durante identificação do sinal ECG, sendo calculada pela Eq. (3). O TP refere-se aos verdadeiros positivos, TN aos verdadeiros negativos, FP aos falsos positivos, e FN aos falsos negativos. Uma outra interpretação para esta equação seria a razão entre o número de predições corretas sobre o número total de predições realizadas. Em outras palavras, acurácia pode ser também definido como a fração das predições que o modelo de aprendizado de máquina acertou prever.

$$Acurácia = \frac{(TP+TN)}{(TP+FP+TN+FN)} \tag{3}$$

A métrica de precisão tenta responder o seguinte questionamento: qual é a proporção de identificações positivas que estão realmente corretas? A precisão pode ser calculada de acordo com a Eq. (4), onde TP refere-se aos verdadeiros positivos e FP aos falsos positivos. Uma outra forma de especificar a precisão é que esta seja a proporção de verdadeiros positivos pelo número total de valores positivos preditos.

$$Precisao = \frac{(TP)}{(TP+FP)} \tag{4}$$

O *Recall* tenta responder o seguinte questionamento: qual é a proporção de atuais positivos foram identificados corretamente? *Recall* pode ser calculado de acordo com a Eq. (5), onde TP se refere aos verdadeiros positivos e FN aos falsos negativos. Em outros termos, *recall* pode ser interpretado como a forma que os modelos podem calcular os atuais positivos e marcar como positivos (TP).

$$Recall = \frac{(TP)}{(TP+FN)} \tag{5}$$

F1 Score é uma outra métrica um pouco mais distinta. A Eq. (6) apresenta a forma de calculá-la e depende tanto dos valores de precisão quanto de *recall*. *F1 Score* é empregada quando há valores bem diferentes entre precisão e *recall*, tentando achar um certo equilíbrio ao utilizar a média harmônica. Dentre as quatro métricas de desempenho usadas, o *F1 Score* consegue ser a mais apropriada para conjuntos de dados desbalanceados que apresentam distribuições de classes bem distintas.

$$F1\ Score = 2 \times \frac{\text{Precisao} \times \text{Recall}}{\text{Precisao} + \text{Recall}} \tag{6}$$

4.5.4. Algoritmos de Aprendizagem de Máquina

A análise faz uso de vários algoritmos de aprendizado de máquina supervisionados e não supervisionados. No total, empregou-se sete algoritmos de classificação e um *clusterizador*, todos bem diversificados e conhecidos na literatura. Especificamente, considerou-se *Naïve Bayes*, *Random Forest*, *Bagging*, *k-Nearest Neighbor*, *Support Vector Machine*, *K-Means*, *Artificial Neural Network*.

4.5.5. Sobre o ambiente de trabalho e avaliação

Durante as análises, essas características foram combinadas conforme resultados de correlação obtidos na ferramenta Weka[4]. Weka é um conjunto de ferramentas que possuem diversos algoritmos de aprendizado de máquina para as mais distintas funções de mineração de dados, seja estas de classificação, preparação, regressão, *clusterização*, entre outros.

Para a identificação através de algoritmos de aprendizado de máquinas, baseamos a avaliação prática nos métodos *holdout* e *k-fold*. Ambos os métodos são usados para realizar o particionamento dos dados. O objetivo com estes métodos é dividir os dados em conjuntos mutualmente exclusivos e depois utilizar alguns destes subconjuntos para estimação dos parâmetros, ou seja, fazer uma previsão que estima o quão acurado é um determinado modelo (técnica de aprendizado de máquina) na prática. É importante que estes resultados sejam bem avaliados e que consigam determinar com precisão sobre o que tentam classificar, sem que estes dados estejam enviesados por *overfitting* (sobre-ajuste).

[4]Weka 3, https://www.cs.waikato.ac.nz/ ml/weka/. Último acesso em Abril/2019.

Ou seja, é importante que os modelos tenham bons desempenhos para o conjunto de dados e que façam boas estimações sem estarem enviesados/viciados.

Para evitar o *overfitting*, tentamos usar dois métodos recomendados de particionamento. No método *holdout* dividimos os dados em treinamento (60 %), teste (20 %) e validação (20 %). *k-fold* usa apenas uma amostra de dados (dados de treinamento) e tem um único parâmetro chamado *k* [Pacheco et al. 2018]. O parâmetro refere-se ao número de grupos que uma determinada amostra de dados deve ser dividida. Nós executamos a validação cruzada de *k-fold* e *holdout* nas sessões práticas.

O objetivo de toda a avaliação é mostrar de forma prática como pode ser realizada a identificação de pessoas com alguma técnica de aprendizado de máquina, usando sinais vitais como o ECG, registrando assim uma demonstração de tudo o que foi explicado neste JAI. Para maiores detalhes das etapas práticas e procedimentos a serem adotados na apresentação prática, estará disponibilizado um outro material que permitirá o acompanhamento da seção prática, através do *link*[5].

4.6. Considerações Finais e Direções Futuras

Esta seção está organizada de acordo com as conclusões tiradas e desafios/*open issues* feitos acerca do tema. Ao longo deste capítulo realizamos uma revisão da evolução tecnológica que a linha de pesquisa biométrica, como técnica de segurança, sofreu nas últimas décadas. Independente do sinal biométrico utilizado (digital, mão, rosto, íris, ECG, etc), foi possível observar que a acurácia e qualidade do sistema dependem de diversos fatores como a qualidade do sensor utilizado para a aquisição dos dados, técnica de pré-processamento do sinal, escolha das *features*, técnicas de aprendizado de máquina aplicadas e outros. Para a simulação e estudos científicos, há dificuldades em relação às base de dados públicas disponíveis que possam refletir um cenário realístico no número de indivíduos para testes.

Para traços biométricos estabelecidos e utilizados ao longo de décadas, como as digitais, existem bases de dados disponíveis coletadas através de situações reais. Para o sinal biométrico ECG, o qual tem sido bastante utilizado em novas pesquisas, apesar de trabalhos como o [Merone et al. 2017], onde é feito um mapeamento sistemático das bases de dados mais utilizadas, percebemos que ainda falta uma padronização que permita testes mais robustos, com milhares de indivíduos para teste, algo possível para digitais, por exemplo. Assim, novos sinais biométricos como o ECG, EEG, PPG e EMG ainda precisam ser validados com bases de testes maiores e mais robustas. Esta é uma das grandes dificuldades: validar esses modelos com uma quantidade maior de indivíduos e com uma padronização em relação ao *hardware* utilizado para permitir uma comparação e avaliação de desempenho fidedigna.

Além disso, entende-se que, devido à grande quantidade de *features* disponíveis para cada sinal e à variação de técnicas de aprendizado de máquina que podem ser utilizadas em conjunto com essas *features*, é difícil que ocorra uma padronização e resposta exata para qual a melhor característica a ser utilizada de um sinal ou qual a melhor técnica de aprendizado de máquina. Muitos aspectos precisam ser levados em consideração,

[5]Material complementar do JAI, https://github.com/TheHealthsenseProject/JAI. Último acesso em Maio/2019.

tais como a sensibilidade do algoritmo de classificação em relação ao ruído no sinal capturado, expectativa no tempo de processamento, domínio do sinal e outros. As métricas de desempenho também devem ser escolhidas criteriosamente, por representarem aspectos diferentes da qualidade de um sistema de autenticação.

Pode-se observar que vários classificadores são utilizados, como o ANN, k-NN, SVM, RF. De um modo geral, as variações de redes neurais (ANN) têm sido mais utilizadas, assim como o SVM e o k-NN. Os algoritmos ligados à árvores de decisão, como o *Random Forest*, são muito utilizados também devido à sua menor complexidade e maior facilidade de compreensão para quem inicia nessa linha de pesquisa. Sendo assim, entende-se que o estudo de técnicas de biometria proporciona para o estudante de computação uma visão transversal e prática de diversas disciplinas, como eletrônica (sensores para aquisição de dados), processamento de Sinais (tratamento dos ruídos e extração/Seleção de *features*), redes (transmissão de dados e processamento distribuído) e inteligência Computacional (utilização de aprendizado de máquina para identificação de padrões).

Acreditamos que ainda há grande oportunidade para avançarmos na utilização de novos sinais biométricos, principalmente em relação ao ECG, PPG, e EEG. Ainda precisam ser endereçados aspectos na aquisição dos dados, aprendizado de máquina, utilização multimodal de sinais, publicidade, falsificação e segurança dos dados. Inicialmente, no caso do ECG, os dados eram capturados através de dispositivos médicos e em um ambiente controlado dentro de uma clínica ou hospital. Como a finalidade dos dados era prioritariamente a identificação de doenças cardíacas, essa metodologia era razoável. Com o avanço da Eletrônica, a aquisição dos sinais passou a ser distribuída, através de dispositivos vestíveis e *gadgets* tecnológicos, aumentando significativamente o conforto e aceitabilidade pelo usuário. Esse tipo de avanço possibilitou um maior interesse das pessoas por monitorar seus sinais cardíacos, obtendo ganhos de saúde ao monitorar o ritmo do coração ao longo do dia.

As pesquisas devem continuar a explorar melhores técnicas de aquisição de sinais, melhorando a aceitabilidade dos novos sistemas, como o sinal de ECG por exemplo. Além disso, também será foco o endereçamento para o uso de múltiplos sinais, conforme a Figura 4.16, onde é visualizada a comparação entre multimodal e unimodal. A utilização do rosto/face como método de autenticação estava apenas em laboratório até poucos anos atrás, mas hoje já está em uso por vários fabricantes de celulares, por exemplo. Espera-se que, em breve, outros sinais passem a ser utilizados, como o PPG e o ECG.

Aspectos ligados a proteção dos dados também devem ser levados em consideração, pois acreditamos que a preocupação com a privacidade fará com que tanto a indústria como a academia desloque mais esforços para aumentar o controle dos dados sensíveis produzidos e seu armazenamento. Técnicas de criptografia passarão a ser itens obrigatórios no armazenamento de dados em dispositivos IoT, além da utilização de biometria em serviços que, até então, não existia a preocupação com o acesso malicioso.

Um dos primeiros desafios que precisam ser endereçados, principalmente ao levar em conta a iminente utilização da biometria em dispositivos IoT, com limitações de *hardware*, é o aumento da qualidade nos sensores. Possibilitando a supressão do efeito de várias fontes de ruídos sem degradar as informações contidas no sinal biométrico que possibilitam identificar um indivíduo. Os sistemas biométricos se beneficiarão dos avanços

nos sensores, principalmente com novas tecnologias ópticas para coleta de sinais, as quais já vem melhorando a coleta de sinais PPG, por exemplo. Assim, outros esforços vêm sendo direcionados a alguns desafios.

Existe ainda uma dificuldade de encontrar a melhor representação de *features* de um traço biométrico. Por exemplo, para o ECG, como vimos na Figura 4.5 e na Tabela 4.2, há um número grande de *features* que podem ser utilizadas. O ideal é a escolha do conjunto de discriminantes mais representativos. Por intuição, isso seria relativamente fácil de se atingir ao aumentar o número de *features*, mas não há garantia que isso vá aumentar a acurácia, exceto se, de fato, existir. Portanto, é necessário uma avaliação profunda sobre quais *features* devem ser selecionadas.

Uma outra dificuldade se apresenta em escolher o melhor algoritmo para ser usado como classificador. É importante mencionar que não há um classificador que possa ser aplicado universalmente para todos os traços biométricos. Portanto, há a necessidade de analisar cuidadosamente o algoritmo de aprendizado de máquina com o conjunto de dados (tipos de sinais biométricos) a ser trabalhado.

Apesar de todas as dificuldades, acreditamos na necessidade e importância de se estudar com mais profundidade sobre a autenticação usando sinais biométricos, que tende a se tornar cada vez mais popular na sociedade. Neste capítulo e na apresentação durante a Jornada de Atualização em Informática (JAI), demos um enfoque maior ao ECG como estudo de caso prático, por entendermos que o ECG, dentre os novos sinais biométricos, é o que possui maior potencial para ser utilizado brevemente na indústria, tornando-se assim uma alternativa viável e com grande valor agregado por possibilitar o monitoramento dos sinais do coração e prover segurança ao mesmo tempo.

Agradecimentos

Este trabalho foi desenvolvido através do projeto denominado *HealthSense: Assessing and Protecting Privacy in Wireless Wearable Sensor-generated Medical Data*, através do apoio da Rede Nacional de Ensino e Pesquisa (Brasil) e da *National Science Foundation* (EUA). No Brasil, o projeto está sendo executado pela Universidade Federal do Paraná (UFPR) e pela Universidade Federal do Pará (UFPA).

Referências

[Abate et al. 2007] Abate, A. F., Nappi, M., Riccio, D., and Sabatino, G. (2007). 2d and 3d face recognition: A survey. *Pattern Recognition Letters*, 28(14):1885 – 1906. Image: Information and Control.

[Al-Waisy et al. 2018] Al-Waisy, A. S., Qahwaji, R., Ipson, S., Al-Fahdawi, S., and Nagem, T. A. (2018). A multi-biometric iris recognition system based on a deep learning approach. *Pattern Analysis and Applications*, 21(3):783–802.

[Allen 2007] Allen, J. (2007). Photoplethysmography and its application in clinical physiological measurement. *Physiological measurement*, 28(3):R1.

[Apple 2019] Apple (2019). Apple watch. `https://www.apple.com/watch`. Acessado em: Janeiro de 2019.

[Baynes 2019] Baynes, C. (2019). Chinese police to use facial recognition technology to send jaywalkers instant fines by text. https://www.independent.co.uk/news/world/asia/china-police-facial-recognition-technology-ai-jaywalkers-fines-text-wechat-weibo-cctv-a8279531.html. Acessado em: Maio de 2019.

[Belgacem et al. 2012] Belgacem, N., Nait-Ali, A., Fournier, R., and Bereksi-Reguig, F. (2012). Ecg based human authentication using wavelets and random forests. *Int. J. Cryptogr. Inf. Secur*, 2(3):1–11.

[Biel et al. 2001] Biel, L., Pettersson, O., Philipson, L., and Wide, P. (2001). Ecg analysis: a new approach in human identification. *IEEE Transactions on Instrumentation and Measurement*, 50(3):808–812.

[Bledsoe 1966] Bledsoe, W. (1966). Man-machine facial recognition. *Technical Report, PRI 22 - Panoramic Research, Inc.*, 73.

[Bonissi et al. 2013] Bonissi, A., Labati, R. D., Perico, L., Sassi, R., Scotti, F., and Sparagino, L. (2013). A preliminary study on continuous authentication methods for photoplethysmographic biometrics. In *Biometric Measurements and Systems for Security and Medical Applications (BIOMS), 2013 IEEE Workshop on*, pages 28–33. IEEE.

[Bosche et al. 2018] Bosche, A., Crawford, D., Jackson, D., Schallehn, M., and Schorling, C. (2018). 2018 roundup of internet of things forecasts and market estimates. https://www.bain.com/insights/unlocking-opportunities-in-the-internet-of-things/. Acessado em: Abril de 2019.

[Camara et al. 2018] Camara, C., Peris-Lopez, P., Gonzalez-Manzano, L., and Tapiador, J. (2018). Real-time electrocardiogram streams for continuous authentication. *Applied Soft Computing Journal*.

[Chen et al. 2005] Chen, H., Valizadegan, H., Jackson, C., Soltysiak, S., and Jain, A. K. (2005). Fake hands: spoofing hand geometry systems. *Biometric Consortium*.

[Clark and Kruse 1990] Clark, V. L. and Kruse, J. A. (1990). Clinical methods: the history, physical, and laboratory examinations. *Jama*, 264(21):2808–2809.

[Columbus 2018] Columbus, L. (2018). 2018 roundup of internet of things forecasts and market estimates. https://www.forbes.com/sites/louiscolumbus/2018/12/13/2018-roundup-of-internet-of-things-forecasts-and-market-estimates/#51e137ea7d83. Acessado em: Abril de 2019.

[Daluz 2014] Daluz, H. (2014). *Fundamentals of Fingerprint Analysis*. Taylor & Francis.

[Dar et al. 2015] Dar, M. N., Akram, M. U., Usman, A., and Khan, S. A. (2015). Ecg biometric identification for general population using multiresolution analysis of dwt based features. In *Information Security and Cyber Forensics (InfoSec), 2015 Second International Conference on*, pages 5–10. IEEE.

[Daugman 2010] Daugman, J. (2010). How iris recognition works. *IEEE Trans. Circuits Syst. Video Technol. 14*, pages 21–30.

[Dhanvijay and Patil 2019] Dhanvijay, M. M. and Patil, S. C. (2019). Internet of things: A survey of enabling technologies in healthcare and its applications. *Computer Networks*, 153:113 – 131.

[Dodds 2019] Dodds, L. (2019). Chinese businesswoman accused of jaywalking after AI camera spots her face on an advert. https://www.telegraph.co.uk/technology/2018/11/25/chinese-businesswoman-accused-jaywalking-ai-camera-spots-face/. Acessado em: Maio de 2019.

[Duta 2009] Duta, N. (2009). A survey of biometric technology based on hand shape. *Pattern Recognition*, 42(11):2797–2806.

[Ericsson 2018] Ericsson (2018). Iot connections outlook. https://www.ericsson.com/en/mobility-report/reports/november-2018/iot-connections-outlook. Acessado em: Abril de 2019.

[Gangwar and Joshi 2016] Gangwar, A. and Joshi, A. (2016). Deepirisnet: Deep iris representation with applications in iris recognition and cross-sensor iris recognition. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 2301–2305. IEEE.

[Goldberger et al. 2000a] Goldberger, A. L., Amaral, L. A., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K., and Stanley, H. E. (2000a). Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiological signals. *circulation*, 10(23):e215–e220.

[Goldberger et al. 2000b] Goldberger, A. L., Amaral, L. A. N., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K., Stanley, H. E., and et al. (2000b). Physiobank, physiotoolkit, and physionet. *Circulation*, 101(23).

[Goodfellow et al. 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[Gu et al. 2003] Gu, Y., Zhang, Y., and Zhang, Y. (2003). A novel biometric approach in human verification by photoplethysmographic signals. In *Information Technology Applications in Biomedicine, 2003. 4th International IEEE EMBS Special Topic Conference on*, pages 13–14. IEEE.

[Hatzinakos and Yadav 2019] Hatzinakos, D. and Yadav, U. (2019). Photoplethysmograph (PPG) based Biometric Recognition. https://www.comm.utoronto.ca/~biometrics/PPG_Dataset/index.html. Acessado em: Janeiro de 2019.

[Healey and Picard 2005] Healey, J. A. and Picard, R. W. (2005). Detecting stress during real-world driving tasks using physiological sensors. *IEEE Transactions on Intelligent Transportation Systems*, 6(2):156–166.

[Insider 2019] Insider, B. (2019). Password-free smartphones are no longer the stuff of science fiction — they're everywhere. https://www.businessinsider.com/smartphone-biometrics-are-no-longer-the-stuff-of-science-fiction-2017-12. Acessado em: Abril de 2019.

[Irvine et al. 2001] Irvine, J., Wiederhold, B., Gavshon, L., Israel, S., McGehee, S., Meyer, R., and Wiederhold, M. (2001). Heart rate variability: a new biometric for human identification. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI'01)*, pages 1106–1111.

[Jain and Duta 1999] Jain, A. K. and Duta, N. (1999). Deformable matching of hand shapes for user verification. In *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on*, volume 2, pages 857–861. IEEE.

[Jain et al. 2016] Jain, A. K., Nandakumar, K., and Ross, A. (2016). 50 years of biometric research: Accomplishments, challenges, and opportunities. *Pattern Recognition Letters*, 79:80 – 105.

[Jindal et al. 2016] Jindal, V., Birjandtalab, J., Pouyan, M. B., and Nourani, M. (2016). An adaptive deep learning approach for ppg-based identification. In *2016 38th Annual international conference of the IEEE engineering in medicine and biology society (EMBC)*, pages 6401–6404. IEEE.

[Jirayucharoensak et al. 2014] Jirayucharoensak, S., Pan-Ngum, S., and Israsena, P. (2014). Eeg-based emotion recognition using deep learning network with principal component based covariate shift adaptation. *The Scientific World Journal*, 2014.

[Jung and Heo 2018] Jung, H. and Heo, Y. (2018). Fingerprint liveness map construction using convolutional neural network. *Electronics Letters*, 54(9):564–566.

[Karimian et al. 2017] Karimian, N., Tehranipoor, M., and Forte, D. (2017). Non-fiducial ppg-based authentication for healthcare application. In *Biomedical & Health Informatics (BHI), 2017 IEEE EMBS International Conference on*, pages 429–432. IEEE.

[Karpagachelvi et al. 2010] Karpagachelvi, S., Arthanari, M., and Sivakumar, M. (2010). ECG feature extraction techniques - A survey approach. *CoRR*, abs/1005.0957.

[Kate Conger 2019] Kate Conger, Richard Fausset, S. F. K. (2019). San Francisco Bans Facial Recognition Technology. https://www.nytimes.com/2019/05/14/us/facial-recognition-ban-san-francisco.html. Acessado em: Maio de 2019.

[Kavsaoğlu et al. 2014] Kavsaoğlu, A. R., Polat, K., and Bozkurt, M. R. (2014). A novel feature ranking algorithm for biometric recognition with ppg signals. *Computers in biology and medicine*, 49:1–14.

[Kinetz 2019] Kinetz, E. (2019). China deploys fully automated airport check-ins using facial recognition. https://www.pressherald.com/2018/10/16/fully-automated-airport-check-ins-using-facial-recognition-arrive-in-china/. Acessado em: Maio de 2019.

[Krizhevsky et al. 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Kukula and Elliott 2006] Kukula, E. and Elliott, S. (2006). Implementation of hand geometry: an analysis of user perspectives and system performance. *IEEE Aerospace and Electronic Systems Magazine*, 21(3):3–9.

[Kyoso and Uchiyama 2001] Kyoso, M. and Uchiyama, A. (2001). Development of an ecg identification system. In *Engineering in medicine and biology society, 2001. Proceedings of the 23rd annual international conference of the IEEE*, volume 4, pages 3721–3723. IEEE.

[Labati et al. 2018] Labati, R. D., Muñoz, E., Piuri, V., Sassi, R., and Scotti, F. (2018). Deep-ecg: Convolutional neural networks for ecg biometric recognition. *Pattern Recognition Letters*.

[Laguna 1990] Laguna, P. (1990). New electrocardiographic signal processing techniques: Application to long-term records. *PhD. Disertation, Science Faculty, University of Zaragoza*.

[Längkvist et al. 2012] Längkvist, M., Karlsson, L., and Loutfi, A. (2012). Sleep stage classification using unsupervised feature learning. *Advances in Artificial Neural Systems*, 2012:5.

[Liu 2010] Liu, M. (2010). Fingerprint classification based on adaboost learning from singularity features. *Pattern Recogn.*, 43(3):1062–1070.

[Liu et al. 2016] Liu, N., Zhang, M., Li, H., Sun, Z., and Tan, T. (2016). Deepiris: Learning pairwise filter bank for heterogeneous iris verification. *Pattern Recognition Letters*, 82:154–161.

[Lumini and Nanni 2017] Lumini, A. and Nanni, L. (2017). Overview of the combination of biometric matchers. *Information Fusion*, 33:71 – 85.

[Mauceri 1965] Mauceri, A. (1965). Feasibility study of personal identification by signature verification. *Technical Report SID65-24 North American Aviation*.

[Merone et al. 2017] Merone, M., Soda, P., Sansone, M., and Sansone, C. (2017). Ecg databases for biometric systems: A systematic review. *Expert Systems with Applications*, 67:189 – 202.

[Miotto et al. 2017] Miotto, R., Wang, F., Wang, S., Jiang, X., and Dudley, J. T. (2017). Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics*, 19(6):1236–1246.

[Nakayama et al. 2019] Nakayama, F., Lenz, P., Cremonezi, B., dos Santos, A., Nogueira, M., Chowdhoury, K., Banou, S., Rosario, D., and Cerqueira, E. (2019). Autenticação contínua e segura baseada em sinais ppg e comunicação galvânica. In *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*.

[Nguyen et al. 2017] Nguyen, K., Fookes, C., Ross, A., and Sridharan, S. (2017). Iris recognition with off-the-shelf cnn features: A deep learning perspective. *IEEE Access*, 6:18848–18855.

[Odinaka et al. 2012] Odinaka, I., Lai, P.-H., Kaplan, A. D., O'Sullivan, J. A., Sirevaag, E. J., and Rohrbaugh, J. W. (2012). Ecg biometric recognition: A comparative analysis. *IEEE Transactions on Information Forensics and Security*, 7(6):1812–1824.

[Oinonen et al. 2010] Oinonen, H., Forsvik, H., Ruusuvuori, P., Yli-Harja, O., Voipio, V., and Huttunen, H. (2010). Identity verification based on vessel matching from fundus images. In *Proceedings of the 17th IEEE International Conference on Image Processing ICIP, Hong Kong, September 26-29, 2010*, pages 4089–4092. Contribution: organisation=sgn,FACT1=1.

[Orjuela-Cañón et al. 2013] Orjuela-Cañón, A. D., Delisle-Rodríguez, D., López-Delis, A., de la Vara-Prieto, R. F., and Cuadra-Sanz, M. B. (2013). Onset and peak pattern recognition on photoplethysmographic signals using neural networks. In *Iberoamerican Congress on Pattern Recognition*, pages 543–550. Springer.

[Pacheco et al. 2018] Pacheco, F., Exposito, E., Gineste, M., Baudoin, C., and Aguilar, J. (2018). Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Com. Surveys & Tuts,*.

[Page et al. 2015] Page, A., Kulkarni, A., and Mohsenin, T. (2015). Utilizing deep neural nets for an embedded ecg-based biometric authentication system. In *2015 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4. IEEE.

[Pan and Tompkins 1985] Pan, J. and Tompkins, W. J. (1985). A real-time qrs detection algorithm. *IEEE Trans. Biomed. Eng*, 32(3):230–236.

[Politi et al. 2016] Politi, M. T., Ghigo, A., Fernández, J. M., Khelifa, I., Gaudric, J., Fullana, J. M., and Lagrée, P.-Y. (2016). The dicrotic notch analyzed by a numerical model. *Computers in biology and medicine*, 72:54–64.

[Pruzansky 1963] Pruzansky, S. (1963). Pattern-matching procedure for automatic talker recognition. *J. Acoust. Soc. Am. 35*, 73.

[Rezgui and Lachiri 2016] Rezgui, D. and Lachiri, Z. (2016). Ecg biometric recognition using svm-based approach. *IEEJ Transactions on Electrical and Electronic Engineering*, 11:S94–S100.

[Ross et al. 1999] Ross, A., Jain, A., and Pankati, S. (1999). A prototype hand geometry-based verification system. In *Proceedings of 2nd conference on audio and video based biometric person authentication*, pages 166–171.

[Russel 2019] Russel, J. (2019). Chinese police are using smart glasses to identify potential suspects. `https://techcrunch.com/2018/02/08/chinese-police-are-getting-smart-glasses/`. Acessado em: Maio de 2019.

[Sajjad et al. 2018] Sajjad, M., Khan, S., Hussain, T., Muhammad, K., Sangaiah, A. K., Castiglione, A., Esposito, C., and Baik, S. W. (2018). Cnn-based anti-spoofing two-tier multi-factor authentication system. *Pattern Recognition Letters*.

[Salanke et al. 2013] Salanke, N. G. R., Maheswari, N., Samraj, A., and Sadhasivam, S. (2013). Enhancement in the design of biometric identification system based on photoplethysmography data. In *Green High Performance Computing (ICGHPC), 2013 IEEE International Conference on*, pages 1–6. IEEE.

[Sancho et al. 2018] Sancho, J., Alesanco, Á., and García, J. (2018). Biometric authentication using the ppg: A long-term feasibility study. *Sensors*, 18(5):1525.

[Sarkar et al. 2016] Sarkar, A., Abbott, A. L., and Doerzaph, Z. (2016). Biometric authentication using photoplethysmography signals. In *Biometrics Theory, Applications and Systems (BTAS), 2016 IEEE 8th International Conference on*, pages 1–7. IEEE.

[Shafer et al. 1996] Shafer, J. C., Agrawal, R., and Mehta, M. (1996). Sprint: A scalable parallel classifier for data mining. In *In Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 544–555. Morgan Kaufmann Publishers Inc.

[Shanir et al. 2017] Shanir, P. P. M., Khan, K. A., Khan, Y. U., Farooq, O., and Adeli, H. (2017). Automatic seizure detection based on morphological features using one-dimensional local binary pattern on long-term eeg. *Clinical EEG and Neuroscience*.

[Shashikumar et al. 2017] Shashikumar, S. P., Shah, A. J., Li, Q., Clifford, G. D., and Nemati, S. (2017). A deep learning approach to monitoring and detecting atrial fibrillation using wearable technology. In *2017 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pages 141–144. IEEE.

[Su et al. 2017] Su, H.-R., Chen, K.-Y., Wong, W. J., and Lai, S.-H. (2017). A deep learning approach towards pore extraction for high-resolution fingerprint recognition. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2057–2061. IEEE.

[Sun et al. 2018] Sun, X., Wu, P., and Hoi, S. C. (2018). Face detection using deep learning: An improved faster rcnn approach. *Neurocomputing*, 299:42–50.

[Szegedy et al. 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[Tayal et al. 2015] Tayal, D. K., Jain, A., Arora, S., Agarwal, S., Gupta, T., and Tyagi, N. (2015). Crime detection and criminal identification in India using data mining techniques. *AI and Society*, 30:117–127.

[Times 2019] Times (2019). Delhi: Facial recognition system helps trace 3,000 missing children in 4 days. `https://timesofindia.indiatimes.com/city/delhi/delhi-facial-recognition-system-helps-trace-`

3000-missing-children-in-4-days/articleshow/63870129.cms. Acessado em: Maio de 2019.

[Tomlinson et al. 2019] Tomlinson, W. J., Banou, S., Yu, C., Chowdhury, K. R., and Nogueira, M. (2019). Secure on-skin biometric signal transmission using galvanic coupling. In *IEEE INFOCOM*.

[Trauring 1963] Trauring, M. (1963). Automatic comparison of finger-ridge patterns. *Nature*, 79.

[Tsinganos 2017] Tsinganos, Panagiotis; Skodras, A. (2017). A smartphone-based fall detection system for the elderly. In *In Proceedings of the 10th International Symposium on Image and Signal Processing and Analysis*. IEEE.

[Turner et al. 2014] Turner, J., Page, A., Mohsenin, T., and Oates, T. (2014). Deep belief networks used on high resolution multichannel electroencephalography data for seizure detection. In *2014 AAAI Spring Symposium Series*.

[Unar et al. 2014] Unar, J., Seng, W. C., and Abbasi, A. (2014). A review of biometric technology along with trends and prospects. *Pattern Recognition*, 47(8):2673 – 2688.

[Venkatesh and Jayaraman 2010] Venkatesh, N. and Jayaraman, S. (2010). Human electrocardiogram for biometrics using dtw and flda. In *Pattern recognition (icpr), 2010 20th international conference on*, pages 3838–3841. IEEE.

[Wang et al. 2016] Wang, Y., Wu, Z., and Zhang, J. (2016). Damaged fingerprint classification by deep learning with fuzzy feature points. In *2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 280–285. IEEE.

[Wayman 2007] Wayman, J. (2007). The scientific development of biometrics over the last 40 years. *The History of Information Security: A Comprehensive Handbook, Elsevier, Amsterdam.*, 79.

[Wieclaw et al. 2017] Wieclaw, L., Khoma, Y., Fałat, P., Sabodashko, D., and Herasymenko, V. (2017). Biometric identification from raw ecg signal using deep learning techniques. In *2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 1, pages 129–133. IEEE.

[Wright 2019] Wright, M. (2019). Metropolitan Police trial facial recognition technology in central London for first time. https://www.telegraph.co.uk/news/2018/12/17/metropolitan-police-trial-facial-recognition-technology-central/. Acessado em: Maio de 2019.

[Yadav et al. 2018] Yadav, U., Abbas, S. N., and Hatzinakos, D. (2018). Evaluation of ppg biometrics for authentication in different states. In *2018 International Conference on Biometrics (ICB)*, pages 277–282. IEEE.

[Yesilbudak et al. 2017] Yesilbudak, M., Sagiroglu, S., and Colak, I. (2017). A novel implementation of knn classifier based on multi-tupled meteorological input data for wind power prediction. *Energy Conversion and Management*, 13:434–444.

[Zhang et al. 2018] Zhang, Y., Gravina, R., Lu, H., Villari, M., and Fortino, G. (2018). Pea: Parallel electrocardiogram-based authentication for smart healthcare systems. *Journal of Network and Computer Applications*, 117:10 – 16.

[Zhang and Wu 2016] Zhang, Y. and Wu, J. (2016). Practical human authentication method based on piecewise corrected electrocardiogram. In *Software Engineering and Service Science (ICSESS), 2016 7th IEEE International Conference on*, pages 300–303. IEEE.

[Zhang et al. 2016] Zhang, Y., Zhou, B., Wu, H., and Wen, C. (2016). 2d fake fingerprint detection based on improved cnn and local descriptors for smart phone. In *Chinese Conference on Biometric Recognition*, pages 655–662. Springer.

[Zhao et al. 2018] Zhao, T., Wang, Y., Liu, J., and Chen, Y. (2018). Your heart won't lie: Ppg-based continuous authentication on wrist-worn wearable devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 783–785. ACM.

[Zuo 2019] Zuo, M. (2019). China's high-tech snack shops face a sizzling problem. `https://www.businessinsider.com/china-bingo-box-convenience-store-shanghai-melting-heat-2017-7`. Acessado em: Maio de 2019.