Capítulo 5

Introdução à Computação Quântica com IBM/Qiskit

Calebe P. Bianchini, Giancarlo P. Gamberi, Ryan M. A. Santos

Abstract

This chapter demonstrate how to develop algorithms for the architecture of a Quantum Computer using the IBM/Qiskit development kit. Consequently, we aim to solve classical computing problems using this new architecture. We understand that writing algorithms and programs in this new paradigm presents a challenge, and therefore, key topics in Quantum Computing will be defined and presented, such as its architecture, entanglement, logic gates, the circuits used, and how they behave differently from those in a traditional architecture. Through demonstrations, we will show how this emerging technology offers an interesting degree of parallelism and significant computational acceleration compared to classical architectures, helping the reader better prepare to program on a quantum computer.

Resumo

O objetivo deste capítulo é mostrar como desenvolver algoritmos para a arquitetura de um Computador Quântico usando o kit de desenvolvimento IBM/Qiskit. Consequentemente, pretendemos resolver problemas clássicos da computação tradicional nessa nova arquitetura. Sabemos que escrever algoritmos e programas nesse novo paradigma é um desafio e, por isso, serão definidos e apresentados assuntos importantes da Computação Quântica como sua arquitetura, o emaranhamento, as portas lógicas, os circuitos utilizados e como eles se comportam de maneira diferente em relação a uma arquitetura tradicional. Por meio de demonstrações, será apresentado como essa tecnologia emergente fornece um grau interessante de paralelismo e aceleração computacional significativos em relação às arquiteturas clássicas, permitindo que o leitor se prepare melhor para programar em um computador quântico.

5.1. Introdução

As primeiras propostas de Computadores Quânticos, operantes de acordo com a mecânica quântica, tiveram motivações guiadas pelo estudo dos sistemas físicos e simulações quânticas [Feynman, 1982]. Rapidamente, então, houve diversos estudos na literatura da Ciência da Computação que buscam entender e projetar Computadores Quânticos, o que se mostrou efetivo ao longo do tempo pela popularidade alcançada por estes sistemas: [Deutsch, 1985] provou que estes seriam Turing-completo, isto é, dotados da capacidade de computação universal; [Shor, 1994] desenvolveu um algoritmo puramente quântico para fatoração de números compostos; [Grover, 1996] apresentou uma busca eficiente em uma base de dados desordenada de complexidade $O(\sqrt{N})$; para citar alguns dos grandes avanços na área.

Todos estes exemplos utilizam das vantagens da informação quântica, que é a informação em estado de processamento usando sistemas da mecânica quântica [Nielsen and Chuang, 2010]. Obviamente, elas se diferenciam fundamentalmente da computação clássica tradicional. A unidade da informação assume as leis da mecânica quântica, como emaranhamento e superposição quântica [Shor, 1994, Preskill, 2012]. Sistemas quânticos então fazem uso de técnicas estocásticas que encontram atalhos em meio a todos os caminhos que um algoritmo pode seguir. Por conta disso, sistemas construídos a partir dos *qubits* já são considerados mais poderosos para a solução de certas tarefas do que computadores clássicos, como busca, simulações físicas complexas, modelagem de moléculas e materiais, dentre diversos outros problemas da ciência [Silva, 2018a, Preskill, 2012].

Outro fator que corrobora para a popularização da computação quântica são os desenvolvimentos no campo da engenharia de computadores quânticos, os quais estão cada vez mais acessíveis. Ferramentas como o SDK *Qiskit* da IBM, *pyQuil* da Google, Q# da Microsoft e *myQLM* da Eviden permitem que sejam feitos estudos sobre os comportamentos de um circuito e computador quântico sem um investimento na tecnologia. O que se comprova pela quantidade de trabalhos realizados com a tecnologia, com 33.230 no tópico de '*quantum computing*' no *Web of Science* [Valdez and Melin, 2022], e a ideia de que este paradigma superará Moore [Arute et al., 2019].

Os algoritmos quânticos se caracterizam por operações unitárias sobre os *qubits*, de forma a manipular os estados de superposição [Nielsen and Chuang, 2010]. Estas operações são empregadas nos circuitos por portas lógicas que agem sobre os *qubits* desejados, similarmente às portas lógicas da computação clássica [Silva, 2018a]. Contudo, a saída de um algoritmo normalmente se difere de acordo com a entrada e, por isso, há a implementação de oráculos, também referidos como "*caixas-preta*". Eles adaptam o circuito à entrada recebida, de forma a adquirir a saída esperada por um algoritmo [Silva, 2018a].

A proposta deste deste minicurso é apresentar os princípios mais básicos desse novo modelo de computação, descrevendo as principais características de um Computador Quântico, bem como descrevendo como construir algoritmos quânticos usando o *kit* de desenvolvimento IBM/Qiskit [Javadi-Abhari et al., 2024, Gamberi and Bianchini, 2023]. Ao longo do minicurso, resolveremos problemas clássicos da computação tradicional nesse nova arquitetura.

5.2. Introdução a Mecânica Quântica

A Mecânica Quântica é uma teoria fundamental da física que descreve o comportamento da matéria e da energia em níveis atômicos e subatômicos [Yanofsky and Mannucci, 2008]. Diferente da física clássica, que trata do mundo macroscópico, a mecânica quântica introduz o conceito de *quantização*. Isso significa que propriedades como energia, momento e momento angular existem em pacotes discretos chamados *quanta*. Por exemplo, um átomo só absorve ou emite energia em quantidades específicas, quantizadas, resultando nos níveis de energia discretos observados em espectros atômicos. Essa quantização contrasta com a física clássica, onde essas propriedades variam continuamente [Jorio and Frossard, 2024].

Na mecânica clássica, o estado de um sistema, como a posição e o momento de uma partícula, pode ser determinado com precisão. No entanto, a mecânica quântica descreve o estado de um sistema usando a *função de onda*, denotada por ψ. Essa função matemática codifica as probabilidades de vários resultados possíveis após uma medição, refletindo a incerteza inerente aos sistemas quânticos. Essa abordagem probabilística é um afastamento da visão determinista da física clássica, onde a evolução futura de um sistema poderia ser prevista com certeza. No final, esta função de onda não especifica a localização exata da partícula, mas fornece uma distribuição de probabilidade para encontrar a partícula em várias posições (quando medida). O quadrado da magnitude da função de onda, $|\psi(\mathbf{r},t)|^2$, descreve a probabilidade de encontrar o sistema na posição $\mathbf{r}$ e no tempo t.

A evolução de um sistema quântico ao longo do tempo é regida pela *equação de Schrödinger*. Essa equação, um dos pilares da mecânica quântica, descreve como a função de onda muda sob diferentes potenciais. Ao resolver a equação de Schrödinger, é possível determinar os níveis de energia permitidos e as funções de onda de um sistema quântico. Em sistemas com condições estáveis, costuma-se usar a equação de Schrödinger independente do tempo para identificar os estados estacionários do sistema.

A mecânica quântica também é conhecida devido a três conceitos que se destacam [Shafique et al., 2024]:

- Superposição: um sistema quântico pode existir em uma superposição de múltiplos estados simultaneamente, ao contrário dos sistemas clássicos, que só podem estar em um estado de cada vez.
- Emaranhamento: ele descreve uma forte correlação entre dois ou mais sistemas quânticos, mesmo quando separados por grandes distâncias.
- Decoerência: é um processo pelo qual um sistema quântico perde sua superposição e outras propriedades quânticas devido a interações com seu ambiente.

5.2.1. Superposição Quântica

A *superposição quântica* [Yanofsky and Mannucci, 2008, Microsoft Quantum, ndb] é outro princípio fundamental da mecânica quântica, permitindo que sistemas quânticos existam em múltiplos estados ao mesmo tempo. Enquanto um sistema clássico, como uma

moeda, está em um estado definido (cara ou coroa), um *qubit* pode existir em uma combinação de ambos os estados simultaneamente [Jorio and Frossard, 2024].

O estado geral de um *qubit* é expresso na Equação 1.

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

Nesta equação, α e β são números complexos chamados *amplitudes de probabilidade*, e os quadrados dos módulos dessas amplitudes representam as probabilidades de encontrar o *qubit* em cada estado após a medição. A soma dessas probabilidades deve ser igual a 1, ou seja, $|\alpha|^2 + |\beta|^2 = 1$.

O conceito de superposição é crucial na computação quântica. Computadores clássicos processam uma computação de cada vez, pois *bits* clássicos só podem estar em um dos dois estados possíveis, 0 ou 1. No entanto, computadores quânticos, aproveitando a superposição, podem explorar várias possibilidades simultaneamente ao processar todos os possíveis estados dos *qubits* de uma vez, permitindo uma forma de processamento paralelo conhecida como paralelismo quântico.

O equilíbrio delicado da superposição, no entanto, é interrompido quando há uma medição ou observação. Semelhante ao exemplo de uma moeda que eventualmente cai de um lado ou do outro, o ato de observação força o *qubit* a "colapsar" em um de seus estados base (0 ou 1), de acordo com uma probabilidade determinada pela sua superposição. Esse colapso da superposição é uma diferença fundamental entre sistemas quânticos e clássicos.

A superposição não é apenas um conceito teórico; é uma característica fundamental dos sistemas quânticos. Sua capacidade de permitir o paralelismo quântico forma a base do potencial da computação quântica e de outras tecnologias quânticas. Ela sustenta muitos fenômenos e aplicações quânticas, incluindo algoritmos quânticos, simulações quânticas de comportamento molecular e de materiais, criptografia quântica para comunicações seguras e sensoriamento quântico para medições ultra-sensíveis.

Além disso, a superposição não se limita a *qubits* individuais, mas também pode se estender a sistemas com múltiplos *qubits*. Ao emaranhar vários *qubits* e manipulá-los em superposição, os computadores quânticos podem realizar operações em vários estados possíveis simultaneamente. Essa capacidade é fundamental para o potencial poder computacional dos computadores quânticos.

5.2.2. Emaranhamento Quântico

O *emaranhamento quântico* [Yanofsky and Mannucci, 2008, Microsoft Quantum, nda] é um dos fenômenos mais intrigantes da mecânica quântica, onde duas ou mais partículas tornam-se correlacionadas de tal maneira que seus estados estão entrelaçados, mesmo quando separadas por grandes distâncias. Uma vez emaranhadas, a medição do estado de uma partícula determina instantaneamente o estado da outra, independentemente da distância entre elas. Isso desafia a física clássica e representa uma conexão não local entre as partículas.

Para entender o emaranhamento, é crucial lembrar como esses sistemas são des-

critos. Diferente dos sistemas clássicos, que possuem propriedades definidas, os sistemas quânticos são descritos por uma função de onda. Essa função de onda não determina um único resultado para uma medição, mas fornece uma distribuição de probabilidade sobre todos os resultados possíveis.

Quando dois sistemas quânticos tornam-se entrelaçados, suas funções de onda individuais não podem mais ser escritas separadamente. Em vez disso, eles são descritos por uma única função de onda combinada que abrange ambos os sistemas. É essa função de onda compartilhada que resulta na forte correlação entre os sistemas entrelaçados.

Considere, por exemplo, dois *qubits* inicialmente preparados em um estado emaranhado representado pelo *estado de Bell*, conforme mostra a Equação 2.

$$\left|\Phi^{+}\right\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2}$$

Essa equação ilustra que os dois *qubits* existem em uma superposição em que ambos estão simultaneamente no estado $|0\rangle$ e ambos estão no estado $|1\rangle$. Se uma medição em um *qubit* revelar que ele está no estado $|0\rangle$, o estado do outro *qubit* colapsa instantaneamente para $|0\rangle$ também, apesar da separação física entre eles. Da mesma forma, se o primeiro *qubit* for medido no estado $|1\rangle$, o estado do segundo *qubit* colapsa instantaneamente para $|1\rangle$. Isso destaca a natureza não local do emaranhamento, onde a correlação entre sistemas emaranhados transcende as distâncias espaciais.

5.2.3. Ruído Quântico

O *ruído quântico* [Yanofsky and Mannucci, 2008, Shafique et al., 2024] representa a incerteza e as flutuações inerentes presentes nos sistemas quânticos, devido à natureza probabilística fundamental da mecânica quântica. Diferentemente do ruído clássico, frequentemente atribuído a distúrbios externos, o ruído quântico é intrínseco e persiste mesmo ambiente extremamente controlados. Ele apresenta desafios significativos para a computação quântica, especialmente ao afetar os resultados dos circuitos quânticos. Uma das principais formas pelas quais o ruído quântico se manifesta é por meio do ruído da própria medição.

No domínio quântico, o ato de medir um sistema quântico inevitavelmente perturba o sistema. Quando uma medição é realizada em um *qubit* em superposição, a função de onda colapsa, forçando o *qubit* a assumir um estado definido, seja 0 ou 1. Esse colapso introduz incerteza, pois o resultado específico da medição é probabilístico e determinado pelas amplitudes de probabilidade associadas a cada estado. O próprio processo de obtenção de informações sobre um sistema quântico por meio da medição introduz ruído.

A decoerência, outra consequência significativa do ruído quântico, surge da interação inevitável entre um sistema quântico e seu ambiente. Essa interação leva ao emaranhamento entre os *qubits* do sistema e o ambiente externo, resultando na perda de propriedades quânticas como superposição e emaranhamento. A decoerência perturba o equilíbrio delicado necessário para a computação quântica, causando erros e afetando a precisão dos algoritmos quânticos.

Para ilustrar, imagine um registrador quântico composto por elétrons, com seus

estados de *spin* representando *qubits*. Quando isolados, os elétrons podem ser cuidadosamente inicializados em um estado puro, como o *spin* para cima, representando um valor específico de *qubit*. No entanto, quando esse registrador interage com o ambiente, os elétrons se entrelaçam com inúmeras outras partículas, cada uma em um estado desconhecido. Esse entrelaçamento destrói a pureza do estado do registrador, transformando-o em um estado misto, onde as relações de fase definidas entre os *qubits*, cruciais para a interferência quântica, são perdidas. Só é possível recuperar essa informação de fase perdida rastreando e medindo o estado de cada partícula ambiental que interagiu com o registrador – uma tarefa praticamente impossível.

A decoerência representa um obstáculo fundamental para a construção de computadores quânticos práticos. O desafio reside em encontrar o equilíbrio delicado entre isolar o sistema quântico para minimizar a decoerência, e ainda assim ser capaz de interagir com ele para realizar operações e medições. Superar a decoerência requer abordagens inovadoras, incluindo o desenvolvimento de operações de portas quânticas mais rápidas para superar a decoerência e a implementação de técnicas de computação quântica tolerante a falhas, como códigos de correção de erros quânticos, para mitigar o impacto dos erros.

5.3. Computação Quântica

A computação quântica representa uma mudança fundamental na forma como a informação é processada e os cálculos são realizados [Yanofsky and Mannucci, 2008]. Diferentemente da computação clássica, que utiliza *bits* como a unidade fundamental de informação, a computação quântica aproveita os princípios da mecânica quântica para processar informações de uma maneira essencialmente diferente [Nayak et al., 2024]. No centro desse processamento está o *qubit*, o análogo quântico do *bit*. Enquanto um *bit* clássico pode existir em apenas um de dois estados, 0 ou 1, um *qubit* pode existir em uma superposição, representando simultaneamente tanto 0 quanto 1 com certas probabilidades. Essa capacidade de ocupar múltiplos estados simultaneamente oferece aos computadores quânticos o potencial de explorar vastos espaços computacionais, permitindo que eles resolvam problemas que são intratáveis até mesmo para os mais poderosos computadores clássicos.

O poder da computação quântica une a superposição e o emaranhamento. Assim, a capacidade probabilística de um sistema quântico e o entrelaçamento entre suas partes (ou subsistemas) desbloqueia o potencial para o processamento paralelo em uma escala sem precedentes. À medida que múltiplos *qubits* são manipulados dentro de um computador quântico, eles exploram efetivamente inúmeras possibilidades computacionais simultaneamente, acelerando significativamente certos tipos de cálculos. Esse paralelismo forma a base para os algoritmos quânticos, permitindo que eles superem os algoritmos clássicos em problemas específicos.

Algoritmos quânticos, projetados especificamente para aproveitar a superposição e o emaranhamento, oferecem o potencial de acelerações exponenciais em certas tarefas computacionais. O algoritmo de Shor, por exemplo, pode fatorar grandes números exponencialmente mais rápido do que qualquer algoritmo clássico conhecido, representando um desafio significativo para a criptografia moderna, que se baseia na dificuldade de fa-

torar grandes números. Da mesma forma, o algoritmo de Grover oferece uma aceleração quadrática para a busca em bases de dados não ordenadas, demonstrando o potencial da computação quântica para revolucionar áreas como análise de dados e otimização.

A implementação de computadores quânticos apresenta desafios significativos, principalmente devido à fragilidade dos estados quânticos e à sua suscetibilidade ao ruído. Como apresentado anteriormente, o ruído quântico, particularmente a decoerência resultante das interações entre o sistema quântico e seu ambiente, perturba a delicada superposição e o emaranhamento, levando a erros computacionais. A busca pela computação quântica tolerante a falhas, que visa mitigar o impacto do ruído por meio de técnicas como a correção de erros quânticos, continua sendo um foco central no campo. Superar esses obstáculos é essencial para escalar os computadores quânticos de forma eficaz para lidar com problemas do mundo real.

Apesar desses desafios, o campo da computação quântica continua avançando rapidamente. O surgimento dos computadores quânticos ruidosos de escala intermediária (*Noisy Intermediate-Scale Quantum*, ou NISQ), embora ainda limitados em tamanho e sujeitos a erros, abriu novas possibilidades para explorar o potencial da computação quântica. Esses dispositivos quânticos de estágio inicial demonstraram a capacidade de realizar cálculos que desafiam as capacidades dos computadores clássicos. À medida que a pesquisa avança, abordar as limitações dos dispositivos NISQ por meio de técnicas de correção de erros e tempos de coerência de *qubit* aprimorados abre caminho para computadores quânticos mais poderosos e confiáveis. O impacto potencial da computação quântica abrange diversos campos, desde criptografia e ciência dos materiais até descoberta de medicamentos e inteligência artificial. Embora a jornada para realizar todo o potencial da computação quântica esteja em andamento, o poder transformador que ela detém promete remodelar nosso cenário tecnológico nos próximos anos.

5.3.1. Qubit vs. Bit

O *qubit* [Yanofsky and Mannucci, 2008, Shafique et al., 2024] é o bloco fundamental da computação quântica, análogo ao bit clássico, mas com a capacidade de existir em uma superposição de dois estados simultaneamente. Essa superposição é uma das propriedades mais intrigantes do *qubit* e permite que ele não esteja limitado apenas aos estados binários fixos, como acontece na computação clássica.

Uma forma importante de representar o estado de um *qubit* é através da notação *bra-ket*. Essa notação é amplamente utilizada para descrever estados quânticos e suas interações.

Um *ket*, denotado como $|\psi\rangle$, representa um vetor coluna que descreve um estado quântico. Esse estado pode corresponder a um estado específico de um *qubit*, a uma superposição de estados ou até mesmo a um estado emaranhado envolvendo múltiplos *qubits*. A notação $|\psi\rangle$ encapsula todas as informações necessárias sobre o estado do sistema.

O *bra*, por sua vez, é denotado como $\langle\psi|$ e representa a transposta conjugada do *ket* $|\psi\rangle$. Matematicamente, isso significa tomar a transposta do vetor (convertendo-o de coluna para linha) e, em seguida, aplicar o conjugado complexo a cada elemento.

Por exemplo, se o *ket* $|\psi\rangle$ for representado como $[3, 1-2i]^T$, o *bra* correspondente seria $\langle\psi| = [3, 1+2i]$. Dessa forma, o *bra* pode ser interpretado como um vetor linha.

Quando um *bra* e um *ket* são combinados, formando uma expressão do tipo $\langle\psi|\phi\rangle$, eles representam o produto interno entre os dois estados quânticos $|\psi\rangle$ e $|\phi\rangle$. Em espaços vetoriais de dimensão finita, o produto interno é calculado como o produto escalar entre os vetores *bra* e *ket*, resultando em um valor escalar. Esse valor, tipicamente um número complexo, é conhecido como a amplitude de transição entre os dois estados $|\psi\rangle$ e $|\phi\rangle$. A combinação "*bra-ket*" muitas vezes é referida de forma abreviada como "*braket*".

Além de servir para descrever estados, a notação *bra-ket* é amplamente utilizada em algoritmos quânticos, que consistem em operações sequenciais sobre *qubits*. Essas operações, chamadas de portas lógicas quânticas, modificam o estado dos *qubits* de maneira controlada e são a base de qualquer processamento em um computador quântico.

Os estados $|0\rangle$ e $|1\rangle$ são definidos como os estados de base computacional para um único *qubit*. Eles são análogos aos estados 0 e 1 de um bit clássico. O estado $|0\rangle$, frequentemente chamado de "*ket 0*", indica um *qubit* no estado "desligado" ou "*spin-down*". Ele é representado pela matriz coluna, conforme descreve a Equação 3. O valor no topo da matriz representa a amplitude de probabilidade de o *qubit* estar no estado *desligado*, enquanto o valor na parte inferior representa a amplitude de probabilidade de o *qubit* estar no estado "ligado" ou "*spin-up*".

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{3}$$

Já o estado $|1\rangle$, conhecido como "*ket 1*", representa um *qubit* no estado "ligado" ou "*spin-up*". Sua representação matricial pode ser vista na Equação 4. De forma similar à representação de $|0\rangle$, a entrada superior corresponde à amplitude de probabilidade do estado *desligado*, enquanto a entrada inferior corresponde à amplitude de probabilidade do estado *ligado*.

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{4}$$

Esses estados de base são fundamentais para representar e manipular *qubits* em cálculos quânticos. Uma das formas de utilização é na construção de *qubits* arbitrários. Conforme descrito na Equação 1, qualquer *qubit* único pode ser expresso como uma combinação linear de $|0\rangle$ e $|1\rangle$.

Por exemplo, o estado do *qubit* representado na Equação 5 pode ser expresso como uma combinação linear de $|0\rangle$ e $|1\rangle$, conforme mostra a Equação 6.

$$V = \begin{bmatrix} 3+2i \\ 4-2i \end{bmatrix} \tag{5}$$

$$V = (3+2i)\,|0\rangle + (4-2i)\,|1\rangle \tag{6}$$

Outra aplicação importante é na formação de estados de múltiplos *qubits*, que utilizam os estados de base para construir estados de sistemas quânticos utilizando o produto tensorial. Por exemplo, um sistema de dois *qubits* pode existir em quatro estados possíveis, derivados do produto tensorial dos estados de *qubits* individuais, conforme apresenta a Equação 7.

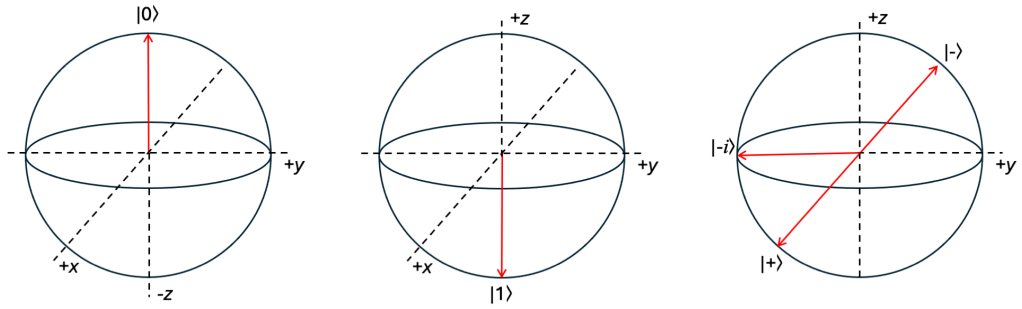
$$|00\rangle = |0\rangle \otimes |0\rangle, \quad |01\rangle = |0\rangle \otimes |1\rangle, \quad |10\rangle = |1\rangle \otimes |0\rangle, \quad |11\rangle = |1\rangle \otimes |1\rangle \tag{7}$$

Esses estados de base formam o alicerce para representar estados mais complexos e emaranhados em sistemas de múltiplos qubits.

Por fim, os estados $|0\rangle$ e $|1\rangle$ são cruciais para as operações de portas quânticas. As portas quânticas, representadas como matrizes, atuam sobre esses estados de *qubit* para realizar transformações, da mesma forma que as portas lógicas operam sobre *bits* clássicos.

Outra forma importante de compreender os estados de um qubit é através da *Esfera de Bloch*, uma representação geométrica que permite visualizar o estado quântico de um único *qubit* [Yanofsky and Mannucci, 2008, ShareTechNote, nd]. Todos os estados puros de um *qubit* podem ser representados como pontos na superfície dessa esfera.

Na Esfera de Bloch, os estados base $|0\rangle$ e $|1\rangle$ são visualizados nos polos norte e sul da esfera, respectivamente, conforme pode ser observado na Figura 5.1a e na Figura 5.1b. Todos os outros pontos na superfície da esfera correspondem a superposições desses dois estados.



(a) *Qubit* básico $|0\rangle$ (b) *Qubit* básico $|1\rangle$ (c) *Qubits* diversos

Figura 5.1: Estados de base computacional quânticos

Por exemplo, o estado $|+\rangle$, que é uma superposição equitativa de $|0\rangle$ e $|1\rangle$, pode ser expresso conforme mostra a Equação 8.

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{8}$$

Este estado é representado como um ponto no equador da esfera, indicando uma probabilidade de 50% de o *qubit* colapsar para o estado $|0\rangle$ e 50% para o estado $|1\rangle$.

A Esfera de Bloch também é uma ferramenta valiosa para visualizar operações sobre *qubits*, como rotações e mudanças de fase. Por exemplo, uma rotação do estado quântico ao redor do eixo z da esfera altera a fase do *qubit*, sem mudar a probabilidade de medir o *qubit* nos estados $|0\rangle$ ou $|1\rangle$. Estados ortogonais, como $|0\rangle$ e $|1\rangle$, são representados por pontos opostos na esfera.

Por exemplo, o estado $|+\rangle$, que, como mencionado, está localizado no equador da Esfera de Bloch ao longo do eixo x positivo, tem como estado ortogonal outro estado relevante, que é o estado $|-\rangle$, conforme expresso na Equação 9.

$$|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{9}$$

Este outro estado também se encontra no equador, mas ao longo do eixo x negativo. Estes dois estados, $|+\rangle$ e $|-\rangle$ podem ser vistos na Figura 5.1c.

Considerando o estado estado $|+\rangle$, e se aplicarmos uma porta quântica que rotacione o *qubit* em 90 graus em torno do eixo y, ele seria transformado no estado $|1\rangle$, ilustrando a mudança geométrica na superfície da esfera. Imagine a mudança de estado gerado por esta porta a partir da Figura 5.1c.

Por fim, os estados que estão localizados no equador ao longo do eixo z também podem ser deduzidos a partir da Figura 5.1c, como o estado $|-i\rangle$.

Embora a Esfera de Bloch seja extremamente útil para visualizar os estados de um *qubit* único, ela apresenta limitações para sistemas quânticos mais complexos, como os estados emaranhados, que envolvem interações entre múltiplos *qubits*. A Esfera de Bloch é eficaz para representar estados de *qubits* individuais, mas a complexidade de sistemas envolvendo múltiplos *qubits* e suas interdependências requer representações mais sofisticadas.

5.3.2. Operações Quânticas e Portas Lógicas

As operações quânticas estão no centro da computação quântica, servindo como transformações matemáticas que atuam sobre estados quânticos [Yanofsky and Mannucci, 2008, Shafique et al., 2024]. Essas operações são inerentemente reversíveis devido à natureza da mecânica quântica e são representadas por matrizes unitárias. A reversibilidade garante que, dado o resultado de uma operação, a entrada possa sempre ser determinada, destacando a natureza determinística da evolução quântica fora dos processos de medição.

As portas quânticas, os blocos de construção fundamentais dos circuitos quânticos, desempenham um papel crucial na manipulação de *qubits* e na transformação da informação quântica. Assim como as portas lógicas clássicas na computação tradicional, as portas quânticas também operam por meio de matrizes unitárias. Vários tipos de portas quânticas servem a funções distintas.

As portas quânticas são instâncias específicas de operações quânticas e são as ferramentas práticas usadas para implementar as transformações teóricas descritas por essas operações. Um circuito quântico é, essencialmente, uma sequência de portas quânticas

aplicadas a *qubits*, representando uma série controlada de operações quânticas que transformam estados quânticos de maneira precisa e estruturada. Essa manipulação de *qubits* forma a base dos algoritmos quânticos e possibilita a computação quântica.

Um dos conceitos mais significativos na computação quântica é a ideia de universalidade. Assim como conjuntos de portas lógicas clássicas podem simular qualquer computação clássica, existem conjuntos universais de portas quânticas que podem aproximar qualquer porta quântica arbitrária com alta precisão. Apesar dessa universalidade, a computação quântica enfrenta algumas limitações inerentes. Todas as operações devem ser reversíveis, e o Teorema da Não-Clonagem proíbe a cópia exata de um estado quântico arbitrário [Jorio and Frossard, 2024].

Um porta quântica importante é a porta de Identidade (I). Ela é um exemplo de porta quântica de um único *qubit*. Sua função é preservar o estado do *qubit* sobre o qual atua, sem alteração. Embora aparentemente trivial, a porta de Identidade possui uma importância significativa em estruturas teóricas e implementações práticas de algoritmos quânticos.

Esta porta é representada por uma matriz identidade 2×2 caracterizada por 1's ao longo da diagonal principal e 0's nos outros lugares, garantindo a manutenção do estado do *qubit* de entrada, conforme mostra a Equação 10.

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{10}$$

Considere, por exemplo, um *qubit* em um estado de superposição arbitrária $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, onde α e β são amplitudes de probabilidade. A aplicação da porta Identidade mantém o estado deste *qubit*, deixando o estado de superposição inalterado, conforme pode ser visto na Equação 11.

$$I|\psi\rangle = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \alpha|0\rangle + \beta|1\rangle = |\psi\rangle \tag{11}$$

Embora a porta de Identidade não induza mudanças observáveis no estado de um *qubit*, ela atua como o elemento identidade para a multiplicação de portas, uma propriedade que é crucial para descrever e analisar circuitos quânticos complexos. Embora não manipule diretamente os estados dos *qubits*, a porta de Identidade facilita a construção de circuitos ao fornecer um espaço reservado para potenciais operações futuras ou manter alinhamentos específicos de portas. Além disso, manter a coerência do qubit é um desafio significativo na computação quântica prática. A porta de Identidade contribui também para esquemas de correção de erros ao preservar o estado do qubit, ajudando a mitigar o impacto do ruído e da decoerência.

Outras portas quânticas que operam sobre um único *qubit* são as portas Pauli, representadas pelas matrizes X, Y e Z. Semelhantes a porta NOT clássica, elas realizam rotações específicas nos *qubits*, e que podem ser visualizados na esfera de Bloch.

A porta Pauli-X, muitas vezes referido como porta de inversão de *bit*, inverte o estado de um *qubit*, transformando $|0\rangle$ em $|1\rangle$ e vice-versa. Sua representação matricial

pode ser vista na Equação 12.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{12}$$

A ação sobre os estados os estados base $|0\rangle$ e $|1\rangle$ pode ser vista nas Equação 13 e na Equação 14, respectivamente.

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \tag{13}$$

$$X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \tag{14}$$

Considerando a esfera de Bloch, a porta Pauli-X realiza uma rotação de 180 graus do *qubit* em torno do eixo X.

Uma outra porta, a Pauli-Y, aplica uma operação combinada de inversão e de inversão de fase a um *qubit*. Ela realiza uma rotação de 180 graus em torno do eixo Y do *qubit* na esfera de Bloch. Sua representação matricial pode ser vista na Equação 15, onde também se encontra sua aplicação sobre os estados base $|0\rangle$ e $|1\rangle$, respectivamente.

$$\begin{aligned} Y &= \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \\ Y|0\rangle &= i|1\rangle \\ Y|1\rangle &= -i|0\rangle \end{aligned} \tag{15}$$

A última porta Pauli, chamada Pauli-Z, é conhecido como o portão de inversão de fase. Ela altera a fase de um *qubit* sem afetar sua amplitude de probabilidade. Na esfera de Bloch, a porta Pauli-Z rotaciona o estado do *qubit* em 180 graus em torno do eixo Z. Sua representação matricial pode ser vista na Equação 16, onde também se encontra sua aplicação sobre os estados base $|0\rangle$ e $|1\rangle$, respectivamente.

$$\begin{aligned} Z &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ Z|0\rangle &= |0\rangle \\ Z|1\rangle &= -|1\rangle \end{aligned} \tag{16}$$

Dentre diversas características importantes, as portas Pauli desempenham um papel crucial nas técnicas de correção de erros quânticos. Ao entender como esses portões afetam os estados dos *qubits*, podemos projetar esquemas para detectar e corrigir erros que surgem devido ao ruído e à decoerência, possibilitando o desenvolvimento de circuitos mais robustos.

Uma outra porta que atua em um único *qubit* é chamada de Hadamard. Ela incorporando a essência da mecânica quântica através de sua capacidade de gerar superposições.

A porta Hadamard (H) transforma o estado de um *qubit* através de uma matriz unitária 2×2, conforme Equação 17.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{17}$$

Aplicar uma porta Hadamard a qualquer estado base $|0\rangle$ ou $|1\rangle$ produz uma superposição, mostrando efetivamente o fenômeno quântico onde o *qubit* existe em uma combinação linear de ambos os estados simultaneamente. A Equação 18 mostra a ação desta porta sobre os estados $|0\rangle$ e $|1\rangle$, respectivamente.

$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ H|1\rangle &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{aligned} \tag{18}$$

A interpretação geométrica da ação da porta Hadamard pode ser visualizada usando a esfera de Bloch, que representa os estados de um único *qubit*. Aplicar o portão de Hadamard a um *qubit* significa rotacionar seu estado 180° ao redor de um eixo que está a 45° entre os eixos X e Z. Este eixo, muitas vezes denominado de "eixo de Hadamard", destaca a transformação única do portão, distinta das rotações induzidas pelos portões Pauli.

Além disso, a capacidade da porta Hadamard de gerar superposições o torna indispensável em vários algoritmos quânticos [Yanofsky and Mannucci, 2008] [Jorio and Frossard, 2024, Shafique et al., 2024]:

- aplicar o portão de Hadamard a um qubit inicializado no estado $|0\rangle$ produz uma superposição igual, efetivamente criando um "lançamento de moeda"quântico. Medir este qubit resulta em $|0\rangle$ ou $|1\rangle$ com igual probabilidade, fornecendo uma fonte de verdadeira aleatoriedade inalcançável em sistemas clássicos.
- muitos algoritmos quânticos dependem fortemente da porta Hadamard. Ao criar superposições de estados de entrada, esses algoritmos podem explorar múltiplos caminhos computacionais simultaneamente, levando a acelerações exponenciais em relação aos seus equivalentes clássicos.
- a porta Hadamard desempenha um papel crucial no protocolo para teletransporte quântico. Sua capacidade de transformar entre a base computacional e a base de Bell (um conjunto de estados maximizadamente emaranhados) possibilita a transferência de informações quânticas entre partes distantes.

A porta Hadamard, apesar de sua forma matemática aparentemente simples, incorpora a *estranheza* e o *poder* da mecânica quântica. Sua capacidade de gerar superposições e realizar rotações específicas na esfera de Bloch o destaca na computação quântica. Ao entender suas propriedades e aplicações, desbloqueamos um conjunto de novas possibilidades computacionais a serem aplicadas em diversos cenários, que vão desde criptografia e ciência dos materiais até descoberta de medicamentos e inteligência artificial.

As portas Identidade (I), Pauli (X, Y, Z) e Hadamard (H) discutidas formam os blocos de construção para criar circuitos quânticos mais complexos que manipulam os estados de múltiplos *qubits*. Esses circuitos, visualizados como sequências de portas aplicadas aos *qubits*, representam algoritmos quânticos. Esses circuitos quânticos utilizam essas portas quânticas para explorar os princípios de superposição, emaranhamento e ruído para resolver problemas computacionais. Por exemplo, múltiplas portas de Hadamard aplicadas a vários *qubits* geram um estado de superposição que abrange todas as possíveis combinações dos estados individuais dos *qubits*. A aplicação subsequente de outras portas quânticas, como as portas de Pauli, manipula as amplitudes e fases desses estados de superposição, codificando informações e realizando computações de maneiras inimagináveis no mundo clássico. Os circuitos quânticos básicos das portas I, Pauli-X, Pauli-Y, Pauli-Z e H são apresentados na Tabela 5.1.

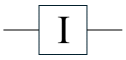
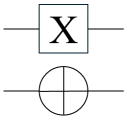
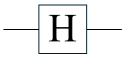
Porta Quântica	Circuito Quântico
Identidade (I)	
Pauli-X (X)	
Pauli-Y (Y)	
Pauli-Z (Z)	
Hadamard (H)	

Tabela 5.1: Relação entre as portas e os circuitos quânticos.

5.4. Modelo de Programação do IBM/Qiskit

O *Qiskit* é uma biblioteca *python* de propriedade da IBM publicada em 2017 e tendo o lançamento de sua versão estável em 2024 [Silva, 2018b, Javadi-Abhari et al., 2024]. Dentre suas diversas características, é possível utilizar essa biblioteca para:

- o desenvolvimento e experimentação de circuitos quânticos, com acesso a recursos como criação de circuitos com diferentes quantidades de *qubits*.

- integração com provedores para execução dos circuitos em computadores quânticos como os da IBM, Azure, Amazon, entre outros provedores.
- utilização de transpiladores, responsáveis por adaptar e traduzir o circuito para execução em computadores quânticos reais ou simulados.
- simular a execução dos circuitos em sistemas clássicos.
- visualizar os circuitos e os *qubits* em forma de esferas de Bloch, e dos resultados observados.
- criação de portas quânticas personalizadas.

Neste capítulo, usamos a versão 1.2.2 do *Qiskit*. A instalação pode ser realizada usando o comando `pip install qiskit` no seu ambiente. Recomendamos também a instalação do módulo de visualização, através do comando `pip install qiskit[visualization]`, além das bibliotecas: *matplotlib*, como complemento na visualização dos resultados; *numpy*, para auxiliar com a manipulação dos dados; e a biblioteca *math*, para uso de algumas funções matemáticas importantes para o desenvolvimento de um algoritmo quântico.

5.4.1. Circuitos no Qiskit

Para a criação de um circuito, a preparação de um ambiente de execução e a análise dos resultados é necessário usar um conjunto de classes e funções disponíveis no Qiskit.

Conforme apresentado no Código 5.1, a representação de um circuito é feita por meio da classe *QuantumCircuit*. É necessário também usar a função *transpile* que adapta as portas e os circuitos para a execução.

Já a classe *BasicProvider* é o provedor escolhido para os experimentos, neste caso simulando o circuito em um ambiente clássico de computação quântica. A classe *Statevector* é usada para retornar o vetor de estados do circuito; no nosso caso foi usado também para visualização das esferas de Bloch.

Por fim, foram usadas as funções *plot_histogram* e *plot_bloch_multivector* para a renderização de histograma e esferas de Bloch, respectivamente.

```
from qiskit import QuantumCircuit, transpile
from qiskit.providers.basic_provider import BasicProvider
from qiskit.quantum_info import Statevector
from qiskit.visualization import plot_histogram,plot_bloch_multivector
```

Código 5.1: Importações necessárias para o desenvolvimento do circuito teste proposto

Ao instanciar um objeto a partir da classe *QuantumCircuit*, é possível definir no construtor quantos *qubits* serão utilizados em um circuito. No Código 5.2, o circuito *qc* foi criado com 3 *qubits*.

```
qc = QuantumCircuit(3)
qc.h(0)
qc.h(1)
qc.h(2)
```

Código 5.2: Criação e inicialização de um circuito

Deste ponto em diante, são definidas as portas quânticas de forma declarativas diretamente no circuito. Para cada porta, o Qiskit possui diferentes comandos e parâmetros para configurará-las. Por exemplo, a porta Hadamard é alocada através do método *h* no objeto do circuito e pede apenas um parâmetro: quais os *qubits* são entradas para esta porta. No Código 5.2, os três *qubits* do circuito passarão pela porta Hadamard conforme declarado nas linhas 3 a 5.

Caso medíssemos este circuito neste exato momento, observaríamos que todos os estados possuem a mesma probabilidade de serem observados. Isto se deve em função das portas Hadamard realizar uma superposição simples, conforme apresentado na Seção 5.3.2. Isso também pode ser observado nas esferas de Bloch apresentadas na Figura 5.2.

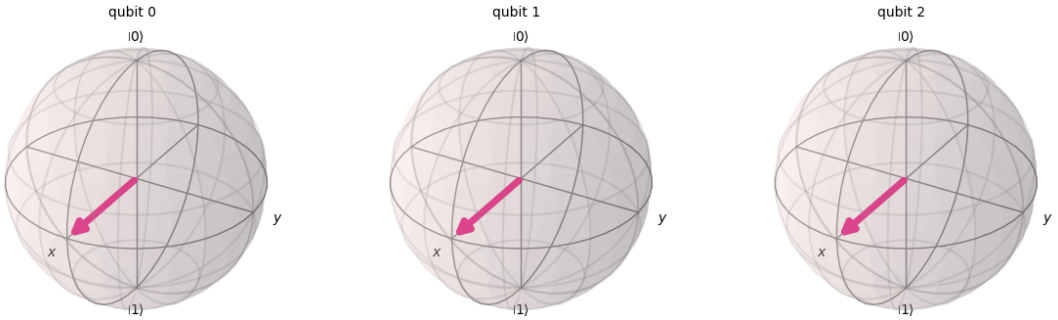


Figura 5.2: Circuito com três *qubits* em superposição inicial renderizado com o Qiskit

Para entender melhor como adicionar novas portas neste circuito e, consequentemente entender melhor as rotações que podem ser feitas nos *qubits*, vamos aumentar nosso sistema quântico, como pode ser visto no Código 5.3. Ao primeiro *qubit*, adicionamos uma rotação Z de 135°. Tal rotação é realizada pelo método *ry*, passando como parâmetro os graus em radianos, $3\frac{\pi}{4}$, seguido do índice do *qubit*, 0.

```
qc.rz(3*pi/4,0)
qc.ry(pi/4,1)
qc.ry(-pi/4,2)
qc.rx(pi/4,2)
qc.measure_all()
qc.draw('mpl')
```

Código 5.3: Aplicação das rotações nos 3 *qubits* seguida da medição do circuito

Ao segundo *qubit*, adicionamos uma rotação Y de 45°, representada pelo método *ry*, passando como parâmetro π radianos seguido d índice do *qubit*, 1. Ao terceiro qubit, adicionamos uma rotação Y de $-45°$ (ou 315°) seguida de uma rotação X de 45°, executada pelos métodos *ry* e *rx*, com os parâmetros $-\frac{\pi}{4}$ e $\frac{\pi}{4}$, respectivamente, com o índice do *qubit*, 2.

Após a execução dessas movimentações no circuito, é possível medi-lo. Para isso, utilizamos o método *measure* indicado quais *qubits* que desejamos observar. No caso do Código 5.3, foram medidos todos os *qubits* com o método *measure_all*. Ao final, pode ser interessante solicitar para o Qiskit renderizar o circuito construído usando o método *draw('mpl')*. O resultado do desenho deste circuito pode ser visto na Figura 5.3. Vale mencionar que essa visualização pode ser feita em qualquer lugar do seu código, durante a montagem do circuito.

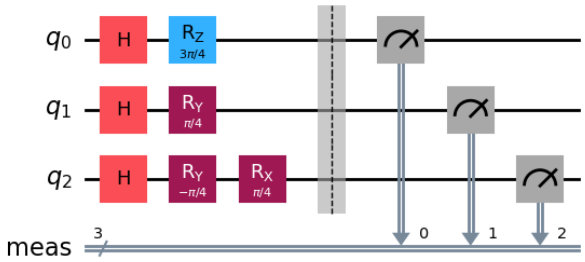


Figura 5.3: Desenho automático do circuito com inicialização de sobreposição, rotações, e medição dos *qubits*

Após a descrição do circuito, podemos executá-lo. Para isso, é necessário escolher um provedor disponível. No caso de uma simulação simples pode ser utilizado o provedor *BasicProvider* e seu *backend*, conforme mostra o Código 5.4. Esse *backend* simula o circuito, conseguindo obter resultados muito próximos de um cenário real.

Em seguida, é necessário *transpilar* o circuito para poder executá-lo. *Transpilar* um circuito é um processo onde as portas são adaptadas para execução no provedor selecionado. Cada provedor pode ter particularidades especificas que remetem a forma como o *qubit* é manipulado. Ao usar o Qskit, o simulador é baseado no funcionamento dos computadores quânticos da IBM, e recebem a tratativa para isso. O processo de transpilação é feito pela função *transpile*, indicando o circuito e o *backend* como parâmetros. O retorno é um circuito transpilado, o qual será utilizado para a execução.

```
provider = BasicProvider()
backend = provider.get_backend("basic_simulator")
new_circuit = transpile(qc, backend)
job = backend.run(new_circuit, shots=1024)
result = job.result()
```

Código 5.4: Exemplo de como simular um circuito quântico e obter seu resultado

A execução do (novo) circuito é feita no método *run* do *backend* escolhido, indicando o circuito transpilado. Nesta etapa, podemos controlar a quantidade de *shots* a serem executados, isto é, quantas medições do mesmo circuito serão executadas. Para que possamos calcular a probabilidade que cada estado obteve nas medições, o número que utilizaremos será um padrão de 1024. Caso estivéssemos utilizando um provedor de um computador físico (real), precisaríamos aguardar a execução do nosso circuito (que pode estar em uma fila) para podermos obter os resultados. Como não é o caso, podemos recuperar os resultados logo em seguida através do método *result* do *job* criado anteriormente.

Os resultados dessa execução podem ser visto usando a função `plot_histogram`, que recebe como parâmetro as contagens da execução dos resultados medidos (obtidas através do método `get_counts`, conforme pode ser visto no Código 5.5. A Figura 5.4 é um exemplo de execução do circuito apresenta na Figura 5.3. Como computadores quânticos possuem tanto o fator aleatório, quanto o fator ruido (que, neste caso, também é simulado), a cada execução, obtemos resultados diferentes, ainda que semelhantes, devido a probabilidade do nosso circuito e ao número de medições.

```
counts = result.get_counts(qc)
plot_histogram(counts)
```

Código 5.5: Exemplo para visualização do histograma dos resultados

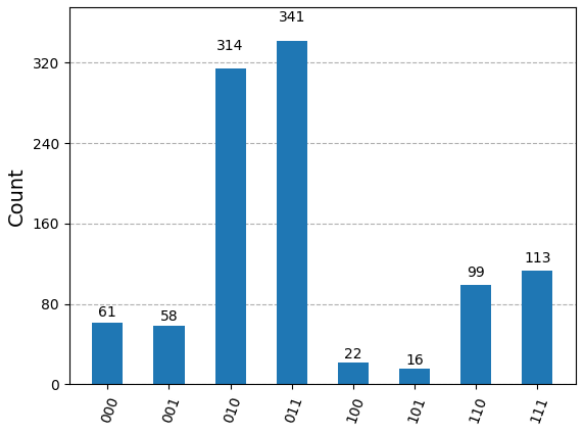


Figura 5.4: Histograma de uma execução do circuito da Figura 5.3

Podemos observar nesta Figura 5.4 alguns resultados das rotações previamente aplicadas: o terceiro *qubit* foi mais observado no estado $|0\rangle$ (ele é o primeiro digito dos números do eixo x do histograma); o segundo *qubit* foi mais observado no estado $|1\rangle$; enquanto o primeiro *qubit* teve observações semelhantes para ambos os estados. Tais resultados eram esperados, uma vez que o segundo e o terceiro *qubits* foram rotacionados de tal forma a possuírem uma maior probabilidade nos respectivos estados observados. Se observamos a Figura 5.5, podemos notar que o terceiro *qubit* ficou em uma posição próxima do estado $|0\rangle$, da mesma forma que o segundo se aproximou do estado $|1\rangle$, enquanto o primeiro *qubit*, apesar de ser rotacionado, não favoreceu sua medição em qualquer estado específico – portanto manteve-se com 50% para ambos os estados.

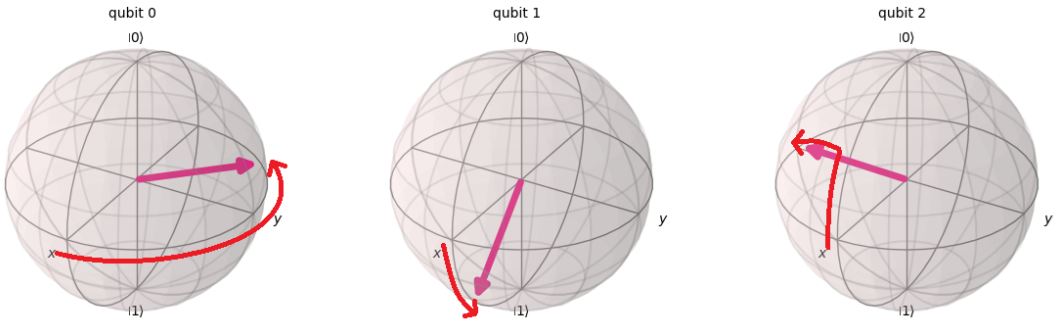


Figura 5.5: Esferas de Bloch para o circuito da Figura 5.3 após aplicarmos a rotações, destacando seu movimento em vermelho

Por fim, a partir dos trechos de códigos apresentados, temos os conceitos básicos de como construir um circuito e executá-lo por meio da biblioteca IBM/Qiskit.

5.5. Aplicações usando IBM/Qiskit

Neste seção apresentamos algumas aplicações amplamente conhecidas que utilizam computação quântica. As soluções são apresentados usando a biblioteca Qiskit, bem como seu circuito e o resultado de suas execuções.

5.5.1. Algoritmo de Grover

Imagine uma lista Telefônica contendo N nomes, arranjados em ordem aleatória. Para se encontrar o telefone de alguém com probabilidade de $1/2$, qualquer algoritmo clássico (seja ele determinístico ou probabilístico) vai necessitar observar no mínimo $N/2$ nomes [Grover, 1996, Shafique et al., 2024].

Grover propôs em 1996 um algoritmo quântico que resolve o problema da busca desordenada com complexidade $O(\sqrt{N})$, onde N é a quantidade de *qubits* sendo utilizados, superando portanto a complexidade de qualquer algoritmo clássico que resolva o problema de busca. Mesmo em cenários ordenados, a complexidade mínima necessária é de $O(n \log n)$.

O algoritmo de Grover originalmente consiste em três passos:

1. Inicializar o circuito em uma superposição
2. Repetir as seguintes etapas $\sqrt{N}$ vezes:
   - Rotacionar a fase buscada em π radianos
   - Aplicar a matriz de difusão D no circuito, definida por $D = HRH$, onde H é a porta $Hadamard$ e R é uma matriz diagonal cujo primeiro elemento é 1, e o restante é -1
3. Observar o circuito

O primeiro passo é trivial para a operação de circuitos quânticos, e não necessariamente é obrigatório para o funcionamento do algoritmo de Grover. O que é necessário

no entanto, é que o sistema esteja em superposição quando aplicarmos o passo seguinte. Por conta de tal comportamento, este primeiro passo pode ser substituído por um oráculo que opere como uma forma de *algoritmo inicial*, que poderia processar diferentes dados, e que usasse o algoritmo de Grover para buscar o resultado desejado.

O segundo passo podemos chamar de principal componente do algoritmo, pois é nele que a *busca* é aplicada e operada. Explicando de forma bem simplificada, este passo essencialmente *"marca"* o estado a ser buscado e então e através de um operador matriz, *"acentua"* tal estado, tornando-o mais provável de ser observado. De forma mais intrínseca, essa *marcação* é realizada através de uma rotação Y de π radianos no estado desejado, seguida por uma matriz de difusão D, que se trata de uma porta definida por Grover como a sequência das portas HRH. A porta H é simplesmente a aplicação da porta Hadamard, enquanto R é um operador matriz, mais especificamente uma matriz diagonal, com a diagonal definida pelo primeiro elemento como -1, e os demais 1. O segundo passo precisa ser repetido $\sqrt{N}$ vezes, pois, quanto maior o número de *qubits*, menos efetiva a acentuação da fase, cenário contornado portanto pela múltipla execução da matriz de difusão.

O terceiro e ultimo passo também é trivial, e se trata da medição do circuito. Vale mencionar, no entanto, que não necessariamente o circuito precisaria finalizar com o algoritmo de Grover, uma vez que o estado encontrado poderia passar por mais uma camada de processamento. O resultado da acentuação da fase pode ser observado por uma chance de *pelo menos* 50% de medição de tal estado. Outro detalhe que podemos mencionar é que podemos buscar mais de uma fase ao mesmo tempo, porém a probabilidade de medição seria dividida entre tais estados. Portanto, quanto mais elementos buscado, menos efetiva é a acentuação de fase, sendo um ponto de atenção ao trabalharmos com o algoritmo de Grover.

Logicamente, podemos reproduzir o algoritmo no Qiskit sem grandes adaptações. Além das importações discutidas na Seção anterior, no Código 5.1, é necessário algumas outras bibliotecas, conforme visto no Código 5.6. A classe *DiagonalGate* é utilizado para aplicação de matrizes diagonais sem a necessidade de criar toda a matriz – basta usar um vetor que represente tal diagonal. Já as demais bilbiotecas são conhecidas: *numpy*, para criação e manipulação de vetores; e *sqrt* e *floor*, utilizada para calcular raízes quadradas e para arredondar para baixo, respectivamente.

```
from qiskit.circuit.library import DiagonalGate
import numpy as np
from math import sqrt,floor
```

Código 5.6: Importações auxiliares para o algoritmo de Grover

O algoritmo completo de Grover pode ser observado no Código 5.7. Na linha 1 estamos declarando a quantidade de *qubits N* que, neste caso, é 3. Na segunda linha há a variável $faseBuscada$ que define o estado que será buscado utilizando o algoritmo de Grover que, neste caso o estado 011 (vale mencionar que podemos utilizar diversas maneiras para identificar tal fase). Para demontrar a explicação anterior, por hora, vamsos para a linha 23 que é onde o circuito *Grover* é criado com *N qubits*. Todos estes *qubits* são

iniciador com a porta Hadamard na linha seguinte. Dessa forma, concluímos o primeiro passo do algoritmo.

```
1  N=3
2  faseBuscada='011'
3
4  #preparação da matriz de rotaço
5  diagonalMatrizRotacao = -np.ones(2**N,dtype=int)
6  diagonalMatrizRotacao[0] = 1
7  matrizRotacao = DiagonalGate(diagonalMatrizRotacao)
8  matrizRotacao.name = 'R'
9
10 #preparação da matriz de difusão
11 matrizDifusao = QuantumCircuit(N, name='D')
12 matrizDifusao.h(range(N))
13 matrizDifusao.append(matrizRotacao,range(N))
14 matrizDifusao.h(range(N))
15
16 #preparação do oráculo
17 rotacaoFase = np.ones(2**N,dtype=int)
18 rotacaoFase[int(faseBuscada,2)] = -1
19 oraculo = DiagonalGate(rotacaoFase)
20 oraculo.name='oraculo'
21
22 #Passo 1
23 Grover = QuantumCircuit(N)
24 Grover.h(range(N))
25
26 #passo 2
27 for i in range(floor(sqrt(N))):
28     #etapa I
29     Grover.append(oraculo,range(N))
30     #etapa II
31     Grover.append(matrizDifusao,range(N))
32
33 #passo 3
34 Grover.measure_all()
35
36 Grover.draw('mpl')
```

Código 5.7: Algoritmo de Grover usando o Qiskit

Voltando para as linhas 4 a 8 do Código 5.7, criamos um pequeno circuito adicional para ser utilizado como uma porta matriz de rotação R utilizada pela matriz de difusão. Na linha 5 temos um vetor $diagonalMatrizRotacao$ composto pelos números inteiros -1, de tamanho 2^N, criado automaticamente pela função do *numpy*: *ones*. Em seguida, alocamos o valor 1, concluindo assim a diagonal de rotação. Na linha 7 utilizamos o comando *DiagonalGate* que, dada a diagonal de rotação, retorna um circuito com a aplicação desta matriz digonal. Este circuito posteriormente pode ser aplicado a outro circuito no formato de porta, explicitando nossa intenção. A linha 8 altera o rótulo desse circulo, por meio da propriedade *name*, para R.

As linhas 10 a 14 do Código 5.7 preparam um circuito que representa a matriz de difusão D. Na linha 11 criamos este circuito $matrizDifusao$, de tamanho N e com o rótulo como D. Nas linhas 12 e 14 aplicamos as portas Hadamard em todos os *qubits*, conforme previsto na definição da matriz de difusão. Na linha 13, entre as duas Hadamard, aplicamos a matriz de rotação, através do comando *append*, passando o circuito a ser utilizado como porta.

Nas linhas 16 a 20, é preparado um oráculo básico que rotaciona em π radianos o estado procurado. Tal rotação pode ser obtida através da aplicação de uma matriz diagonal, composta por elementos 1, e com as fases a serem marcadas com valor -1. Tal diagonal, nomeada de *rotacaoFase* é criada na linha 17. Na linha 18 aplicamos o valor negativo ao respectivo estado buscado. Na linha 19 criamos o circuito *oraculo* que servirá como porta utilizando novamente a função *DiagonalGate*, e por fim renomeamos o circuito para *oraculo* na linha 20.

Agora que finalizamos de configurar os circuitos e as portas, passamos para o segundo passo do algoritmo. Na linha 27 do Código 5.7 iniciamos um *loop* para repetir $\sqrt{N}$ vezes o processo do passo 2. Dentro desta repetição aplicamos a rotação do elemento buscado e a matriz de difusão, ambos aplicados pelo comando *append* usando, como parâmetros, o circuito (ou porta) *oraculo* e $matrizDifusao$, em todos os *qubits*, respectivamente.

Por fim, no terceiro e último passo do algoritmo, realizaremos a medida do circuito para observar os resultados, conforme descrito na linha 34. Na linha 36 há o desenho do circuito, conforme podemos observar na Figura 5.6.

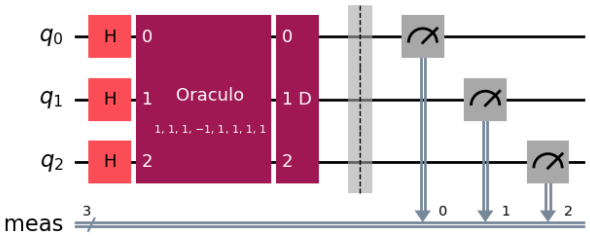


Figura 5.6: Exemplo de um circuito do algoritmo de Grover com um oráculo

Para executar esse código utilizaremos o mesmo princípio apresentado no Código 5.4. Além de usar o novo circuito *Grover*, mudamos a quantidade de *shots* para 10000, mas mantendo a simulação. O histograma resultado da execução pode ser observado na Figura 5.7. Notamos facilmente a medição do estado 011, que se destaca pelas 7800 medições, ou 78% delas, demonstrando a efetividade do algoritmo de Grover para a procura de um elemento. Na Figura 5.8 podemos ver a execução do algoritmo com 4 *qubits*, buscando os estados 0011 e 1010, evidenciando a capacidade de execução do algoritmo para busca de mais de um elemento.

Note como o algoritmo de Grover foi capaz de operar com todos os estados ao mesmo tempo, em sobreposição, utilizando rotações para diferenciá-los, ainda que inicialmente estivessem todos no mesmo estado. O algoritmo de Grover é por muitas vezes

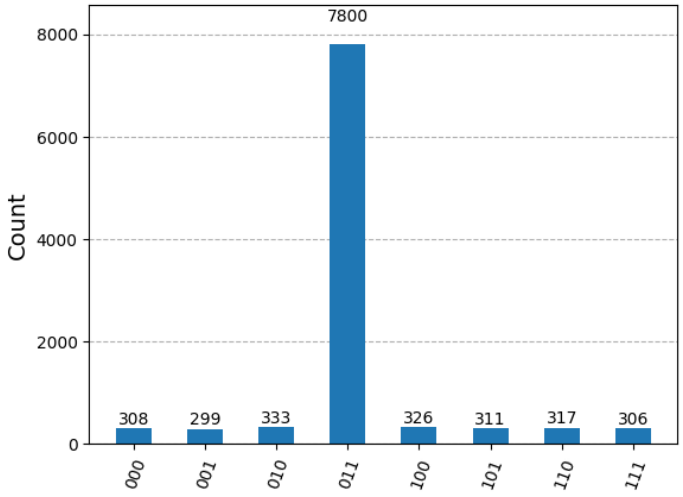


Figura 5.7: Contagem das medições da execução do algoritmo de Grover, com 3 qubits, buscando o estado 011

utilizado em conjunto com outros algoritmos. É também utilizado para fins didáticos pela sua simplicidade e eficácia de execução, demonstrando o potencial da computação quântica para diferentes aplicações.

5.5.2. Resolvendo um Autômato Celular

Autômatos celulares são um modelo de computação úteis na representação de sistemas dinâmicos e complexos por apresentarem um comportamento local (em cada célula) simples, cujo sistema completo detém um comportamento complexo e caótico com o passar do tempo. Cada célula é uma unidade de computação que age de acordo com regras de transição em passos discretos de tempo, de modo que podemos representar tais sistemas de forma genérica por uma função, conforme mostra a Equação 19.

$$\sigma_i^{T+1} = \phi(\{\sigma_j^{(T)}\}, j \in N_i) = \phi(\sigma_{i-1}^{(T)}, \sigma_i^{(T)}, \sigma_{i+1}^{(T)}), \tag{19}$$

Nesta equação, σ_i^{T+1} é uma célula i na próxima geração, e ϕ é a regra de transição do autômato, sendo N_i o conjunto da vizinhança de σ_i [McIntosh, 2009]. Em outras palavras, o estado de uma célula na próxima geração do sistema é definida por regras que consideram os vizinhos mais próximos daquela célula, assim como ela mesma, no momento atual do tempo.

A Equação 19 considera autômatos celulares de apenas uma dimensão, isto é, o sistema pode ser representado por um vetor de células, lado à lado. Portanto esta regra também considera que cada célula possua apenas 2 vizinhos, configurando uma vizinhança de 3 células ao todo; logo, este é o autômato celular mais simples possível, também chamado de elementar.

Por fim, a notação utilizada para representar as regras em um autômato celular elementar de uma dimensão surge a partir do seu valor binário. Há 8 possibilidades de

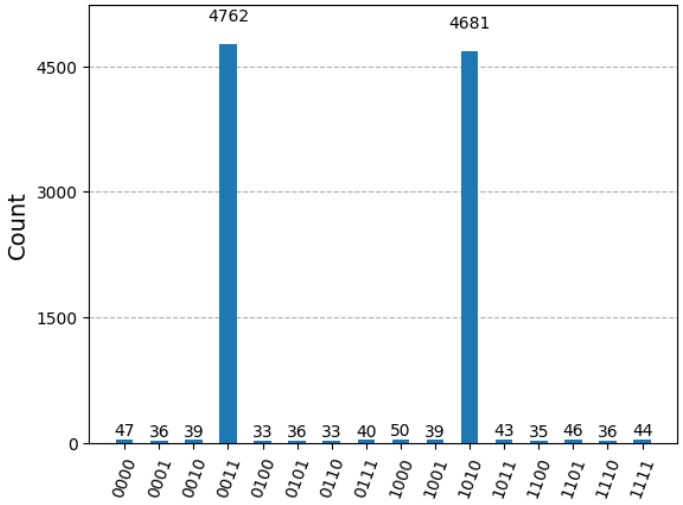


Figura 5.8: Contagem das medições da execução do algoritmo de Grover, com 4 qubits, buscando os estados 0011 e 1010

configurações dentro de uma vizinhança de tamanho 3 e, ao se pensar em cada uma dessas possibilidades como um *bit* em uma *string* de *bits*, podemos adquirir um valor inteiro que informa com qual regra se está trabalhando e quais as configurações que fazem uma célula viver ou morrer [Wolfram, 2002]. A Equação 20 descreve de forma genéria a transcrição das configurações e seu arranjo na *string* de *bits*, onde são apresentadas todas as configurações possíveis em uma vizinhança de um autômato celular elementar em relação ao próximo estado da célula ao centro, sendo β o estado da célula i na próxima geração.

$$(\sigma_{i-1}^{(T)}, \sigma_{i}^{(T)}, \sigma_{i+1}^{(T)}) : \frac{111}{\beta_1}, \frac{110}{\beta_2}, \frac{101}{\beta_3}, \frac{100}{\beta_4}, \frac{011}{\beta_5}, \frac{010}{\beta_6}, \frac{001}{\beta_7}, \frac{000}{\beta_8}. \tag{20}$$

A *Regra 90* é utilizada neste capítulo como demonstração em função da didática e simplicidade, o que torna a derivação dela direta, tanto da fórmula *booleana* quanto do circuito. Na Equação 21 há a apresentação por extenso da *Regra 90*, sendo os valores dos numeradores as configurações possíveis para uma determinada vizinhança e, denominadores, o valor da célula central no próximo momento no tempo. Em seguida, na Equação 22, é apresentada sua forma reduzida.

$$R_{90} : \frac{111}{0}, \frac{110}{1}, \frac{101}{0}, \frac{100}{1}, \frac{011}{1}, \frac{010}{0}, \frac{001}{1}, \frac{000}{0}. \tag{21}$$

$$f_{90}(l,m,r) = XOR[l,r] \tag{22}$$

, onde l denota a célula da esquerda, m a do meio e r a da direita.

A Equação 22 pode ser obtida ao observar o comportamento da regra de acordo com a interação que ocorre na vizinhança. A célula central passa a assumir o valor 1 se e somente se as células à esquerda e à direita forem diferentes, isto é, elas não podem ser tanto 1 quanto 0 ao mesmo tempo. Isto pode ser transcrito para uma fórmula *booleana* que descreve este comportamento, simplesmente definida pela porta XOR. Da mesma

forma, outras regras definidas para este tipo de autômato também podem ser transcritas para suas versões *booleanas* [Wolfram, 2002].

Para o algoritmo quântico, propomos uma espécie de *framework* de trabalho para criação de oráculos [Zhao, 2022]. Primeiro, encontre a fórmula *booleana* que descreva o comportamento desejado. Em seguida, construa o circuito que a descreve. Vale observar, por exemplo que, neste caso, a fórmula é facilmente encontrada ao observar a vizinhança e como a célula central se altera de acordo com seus companheiros. Porém para situações mais complexas, é recomendável ter calma e construir uma tabela verdade do sistema.

A partir deste ponto, é possível transpor a regra de transição para um circuito quântico que realiza esta operação utilizando uma das portas quânticas que melhor se encaixa ao automato desejado. Uma outra opção é encontrar a porta unitária que descreve as transformações necessárias aos *qubits* e trabalhar a partir dele para derivar quais outras portas podem fazer tais transformações. Neste capítulo, usamos a primeira abordagem, tanto pela sua simplicidade quanto pela praticidade.

A operação clássica XOR possui como análogo na computação quântica a porta $CNOT$ (NOT controlado ou X-controlado) em um sistema de dois *qubits*, O comportamento da porta $CNOT$ pode ser observado na Tabela 5.2. Esta porta age de acordo com o estado de um *qubit* de controle e aplica uma rotação *pi* (180°) no eixo X do *qubit* alvo, ou seja, altera seu estado de 0 para 1 e vice-versa. Esta rotação tem um efeito semelhante ao de uma porta NOT em álgebra booleana. Assim, se o *qubit* de controle está no estado 1, o *qubit* alvo terá seu estado revertido.

ψ_1	ψ_2
00	00
01	11
10	10
11	01

Tabela 5.2: Configuração de um sistema com uma porta CNOT onde o *bit* de controle é o mais à direita, e o *bit* alvo o mais à esquerda

Desta lógica, podemos utilizar o segundo *qubit* (alvo) como indicativo de ambos estarem no mesmo estado ou não (com valores 0 e 1, respectivamente). Portanto, o *qubit* alvo determina se uma célula estará viva ou morta na próxima geração com a necessidade de codificar somente os valores de seus vizinhos para o circuito. O resultado desta operação em um circuito de 2 qubits pode ser observado na Figura 5.9.

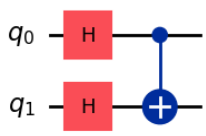


Figura 5.9: Configuração de um XOR em um circuito quântico

Com esta abordagem, considera-se que cada célula é um sistema quântico local com circuito próprio, ou seja, um autômato celular quântico é um vetor n-dimensional de sistemas quântico n-dimensionais [Arrighi, 2019].

O circuito ilustrado na Figura 5.9 forma a base para computar um avanço temporal em uma célula. Embora ele já realize a verificação da igualdade entre valores, é crucial indicar ao circuito se algum dos *qubits* tende a possuir o valor inicial 1. Esta tarefa é atribuída ao oráculo do autômato e pode ser realizada através de rotações nos *qubits*, utilizando, por exemplo, as portas T, que introduz uma fase no *qubit*, aumentando sua amplitude quando este assume o valor 1, e Z, que inverte a fase quando o *qubit* está no estado 1. Essas duas portas estão descritas na Equação 23. Esses operadores unitários agem apenas quando o *qubit* está no estado 1, permitindo-nos "*sinalizar*" ao circuito a configuração da vizinhança. Após tais modificações, aplicamos uma porta Hadamard adicional em ambas as células, consolidando as alterações nas amplitudes resultantes das rotações.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{23}$$

Antes de iniciarmos o processo de superposição, precisamos rotacionar o segundo *qubit* para o estado $|1\rangle$, pois, por padrão, o Qiskit inicializa todos os *qubits* do circuito no estado $|0\rangle$. Tal configuração indica um vizinho morto, enquanto a ausência dela indica um vizinho vivo. O resultado final do circuito que representa a *Regra 90* pode ser observado na Figura 5.10. O código completo desse circuito pode ser visto no Código 5.8.

Neste código definimos a operação local de cada célula do sistema ao passarmos como parâmetro os estados das células à direita e à esquerda em relação à central. Nele, computamos a igualdade dos valores dos vizinhos e medimos o resultado ao fim do circuito, o qual pode ser usado futuramente para alterar o valor da célula central. Com isso, temos o programa que traduz esta regra de transição para um sistema quântico.

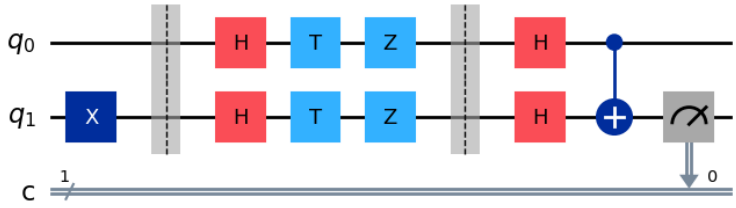


Figura 5.10: Circuito que considera uma configuração específica de uma vizinhança, neste caso $f(1,m,0)$

Com este circuito, podemos simular e observar se ele se comporta da maneira desejada. Conforme observado, de maneira geral, as simulações são feitas em alguns passos:

- Instancia-se o simulador desejado

- Otimiza o circuito para o tipo de hardware no qual ele será executado
- Executa o circuito para adquirir os resultados

```python
from qiskit import QuantumCircuit
from qiskit.circuit.library import HGate, CXGate, TGate, ZGate

def step(left: int, right: int) -> QuantumCircuit():
  # Instancia um circuito
  c = QuantumCircuit(2, 1, name="step")

  # Indica onde deve ser aplicada porta NOT
  if left == 0:
    c.x(0)
  if right == 0:
    c.x(1)
  c.barrier()

  # Preparação inicial - superposição e rotações
  c.append(HGate(), [0])
  c.append(HGate(), [1])
  c.append(TGate(), [0])
  c.append(TGate(), [1])
  c.append(ZGate(), [0])
  c.append(ZGate(), [1])
  c.barrier()

  # Alterar a superposição
  c.append(HGate(), [0])
  c.append(HGate(), [1])

  # Realizar a verificação se os estados são semelhantes
  c.append(CXGate(), [0,1])

  # Adquirir o resultado
  #   - Se 1, as células possuem estados diferentes
  #   - Se 0, as células possuem estados iguais
  c.measure(1,0)
  return c
```

Código 5.8: Código que traduz a regra de transição 90 para um circuito quântico

Essa descrição de simulação pode ser vista no Código 5.9. Apesar dessa simulação ter seus passos bem definidos, este código foi escrito para a versão atual do Qiskit, podendo sofre alterações, conforme seu fabricante achar importante.

Com os resultados da simulação, podemos analisar como o circuito se comportou e verificar se ele realmente implementa a regra de transição para cada célula.

Assim, ao executar o circuito definido com um simulador ideal, isto é, sem ruído, nota-se que conseguimos realizar a transição de uma célula de acordo com os estados de seus vizinhos, conforme mostra a Figura 5.11.

```python
from qiskit_aer import AerSimulator
from qiskit.transpiler.preset_passmanagers import
    ↪ generate_preset_pass_manager

def simulate_ideal(circuit, shots=1024):
    # Instancia o simulador
    sim_ideal = AerSimulator()

    # Otimiza o circuito para o tipo de hardware
    pm = generate_preset_pass_manager(backend=sim_ideal,
    ↪ optimization_level=3)
    isa_qc = pm.run(circuit)

    # Roda a simulação
    result = sim_ideal.run(circuit)

    return result
```

Código 5.9: Função para simulação ideal de um circuito quântico

A partir do retorno do circuito, podemos alterar o estado da célula em questão. Ao aplicar este circuito para cada célula de um conjunto de células é possível resolver um autômato celular onde cada célula constitui um sistema quântico. Neste caso, se seu retorno for uma medição 0, significa que aquela célula estará morta na próxima geração e, se uma medição for 1, ela estará viva. A localidade e simplicidade características de autômatos celulares é mantida e a complexidade na evolução do sistema também.

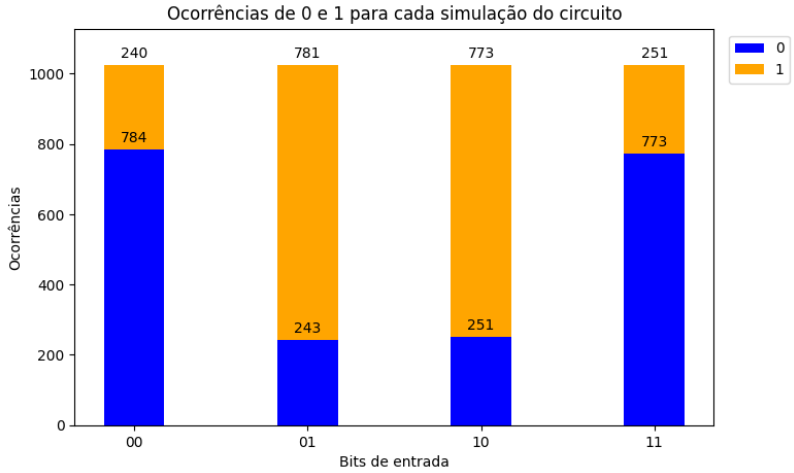


Figura 5.11: Contagem das medições de 1s e 0s de acordo com a configuração dos vizinhos de uma determinada célula

Os passos apresentados nessa seção podem ser usados para criação de oráculos com diversos propósitos em circuitos quânticos, em especial na tradução das regras de transição para autômatos celulares. Resumidamente, primeiro, adquire-se a função *booleana* que representa o comportamento desejado e esperado do sistema, a qual servirá

de orientação para a implementação do oráculo. Após a análise desta fórmula, busca-se encontrar um circuito que replique o seu comportamento. Para tal, podemos utilizar de tabelas verdades, árvores de decisão ou outros análogos na computação clássica como o feito nesta seção. E, por fim, constrói-se o circuito que deve ser modificado para diferentes entradas.

5.6. Conclusão

A computação quântica, um paradigma revolucionário que aproveita os princípios da mecânica quântica, tem imenso potencial para remodelar o cenário da computação e revolucionar diversos campos da ciência e da sociedade. Diferente da computação clássica, que depende de bits representando 0 ou 1, a computação quântica utiliza *qubits*. Eles podem existir em superposição, representando simultaneamente 0 e 1, o que permite que os computadores quânticos realizem cálculos exponencialmente mais rápidos para conjuntos específicos de problemas. Esse paralelismo inerente constitui a base do poder da computação quântica. O entrelaçamento quântico, outra característica única, estabelece uma forte correlação entre os *qubits*, ampliando ainda mais as capacidades computacionais.

Os fundamentos teóricos da computação quântica foram inicialmente estabelecidos neste capítulo, com os circuitos quânticos fornecendo modelos robustos para entender e projetar algoritmos quânticos. Outros aspectos desta fundamentação teórico ainda precisam ser exploradas, como, por exemplo, a máquina de Turing quântica [Yanofsky and Mannucci, 2008].

Além dos circuitos, pode-se perceber também a aplicação prática de algoritmos quânticos para resolver problemas clássicos da computação. Por exempo, o conceito de autômatos celulares quânticos apresenta um modelo promissor para a computação quântica. Os autômatos celulares quânticos operam em uma grade de *qubits*, evoluindo através de regras locais para realizar cálculos, oferecendo uma possível via para o paralelismo e a escalabilidade. Outro importante algoritmo quântico, o algoritmo de Grover, fornece uma aceleração quadrática para problemas de busca não estruturada. O algoritmo de Grover demonstra a ampla aplicabilidade da computação quântica para uma variedade maior de tarefas, incluindo busca em banco de dados e otimização.

Apesar de sua promessa, a computação quântica enfrenta desafios significativos. A decoerência, que é a perda das propriedades quânticas devido a interações com o ambiente, representa um grande obstáculo na construção de computadores quânticos confiáveis. O ruído quântico agrava ainda mais esse problema, exigindo técnicas robustas de correção de erros para preservar a integridade dos cálculos quânticos. Abordar esses desafios é crucial para escalar os computadores quânticos e seus respectivos algoritmos de forma eficaz para lidar com problemas do mundo real.

A busca pela computação quântica continua a cativar pesquisadores e líderes da indústria, talvez, motivado pelos seus grandes desafios [Parmar, 2024]. O campo testemunhou marcos notáveis, incluindo a demonstração da supremacia quântica pelo Google em 2019, ao realizar uma tarefa computacional impossível para computadores clássicos em um período de tempo aceitável. Embora a verdadeira supremacia quântica ainda seja um tema de pesquisa contínua, esses feitos destacam o progresso tangível em direção à realização do pleno potencial da computação quântica [Baczyk, 2024].

Por fim, a computação quântica representa uma mudança de paradigma na computação, prometendo revolucionar a pesquisa científica, a inovação tecnológica e diversos aspectos de nossas vidas. Embora desafios significativos estejam à frente, a busca incessante por essa tecnologia transformadora continua a inspirar, prometendo um futuro onde o enigmático mundo da mecânica quântica nos capacita a resolver problemas antes considerados impossíveis.

Todos os códigos utilizados neste capítulo estão disponíveis em um repositório no GitHub em `https://github.com/hpc-fci-mackenzie/SSCAD24-QuantumComputing`.

Agradecimentos

Os autores agradecem ao MackCloud[1], Laboratório Multidisciplinar de Computação Científica e Nuvem e ao projeto SPRACE - Processo nº 2018/25225-9, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

Referências

[Arrighi, 2019] Arrighi, P. (2019). An overview of quantum cellular automata.

[Arute et al., 2019] Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., Biswas, R., Boixo, S., Brandao, F. G. S. L., Buell, D. A., Burkett, B., Chen, Y., Chen, Z., Chiaro, B., Collins, R., Courtney, W., Dunsworth, A., Farhi, E., Foxen, B., Fowler, A., Gidney, C., Giustina, M., Graff, R., Guerin, K., Habegger, S., Harrigan, M. P., Hartmann, M. J., Ho, A., Hoffmann, M., Huang, T., Humble, T. S., Isakov, S. V., Jeffrey, E., Jiang, Z., Kafri, D., Kechedzhi, K., Kelly, J., Klimov, P. V., Knysh, S., Korotkov, A., Kostritsa, F., Landhuis, D., Lindmark, M., Lucero, E., Lyakh, D., Mandrà, S., McClean, J. R., McEwen, M., Megrant, A., Mi, X., Michielsen, K., Mohseni, M., Mutus, J., Naaman, O., Neeley, M., Neill, C., Niu, M. Y., Ostby, E., Petukhov, A., Platt, J. C., Quintana, C., Rieffel, E. G., Roushan, P., Rubin, N. C., Sank, D., Satzinger, K. J., Smelyanskiy, V., Sung, K. J., Trevithick, M. D., Vainsencher, A., Villalonga, B., White, T., Yao, Z. J., Yeh, P., Zalcman, A., Neven, H., and Martinis, J. M. (2019). Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510.

[Baczyk, 2024] Baczyk, M. (2024). Shall you buy a quantum computer today? - analysis of qc on-premise deployments. *Quantum Computing Report*. Accessed: 2024-08-08.

[Deutsch, 1985] Deutsch, D. (1985). Quantum theory, the Church–Turing principle and the universal quantum computer. *Proc. R. Soc. Lond.*, 400(1818):97–117.

[Feynman, 1982] Feynman, R. P. (1982). Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488.

[Gamberi and Bianchini, 2023] Gamberi, G. P. and Bianchini, C. P. (2023). Study of quantum algorithms and their implementations. In *2023 International Conference on Electrical, Communication and Computer Engineering (ICECCE)*, pages 1–6.

[1] `https://mackcloud.mackenzie.br`

[Grover, 1996] Grover, L. K. (1996). A fast quantum mechanical algorithm for database search.

[Javadi-Abhari et al., 2024] Javadi-Abhari, A., Treinish, M., Krsulich, K., Wood, C. J., Lishman, J., Gacon, J., Martiel, S., Nation, P. D., Bishop, L. S., Cross, A. W., Johnson, B. R., and Gambetta, J. M. (2024). Quantum computing with qiskit.

[Jorio and Frossard, 2024] Jorio, A. and Frossard, J. V. (2024). *Material de Estudos para Mecânica Quântica*. Programa de Pós-Graduação em Física, UFMG, 2nd edition.

[McIntosh, 2009] McIntosh, H. V. (2009). *One Dimensional Cellular Automata*. Luniver Press.

[Microsoft Quantum, nda] Microsoft Quantum (n.d.a). Quantum computing concepts: Entanglement. `https://quantum.microsoft.com/en-us/insights/education/concepts/entanglement`. Accessed: 2024-09-05.

[Microsoft Quantum, ndb] Microsoft Quantum (n.d.b). Quantum computing concepts: Superposition. `https://quantum.microsoft.com/en-us/insights/education/concepts/superposition`. Accessed: 2024-09-05.

[Nayak et al., 2024] Nayak, P., Rathod, S., Surabhi, and Sukanya (2024). Quantum computing: Circuits, algorithms and application. *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, 4(1).

[Nielsen and Chuang, 2010] Nielsen, M. A. and Chuang, I. L. (2010). *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press.

[Parmar, 2024] Parmar, D. (2024). Patent landscape for quantum computing: A survey of patenting activities on different physical realization methods. *IPWatchdog*. Accessed: 2024-08-08.

[Preskill, 2012] Preskill, J. (2012). Quantum computing and the entanglement frontier.

[Shafique et al., 2024] Shafique, M. A., Munir, A., and Latif, I. (2024). Quantum computing: Circuits, algorithms, and applications. *IEEE Access*, 12:22296–22314.

[ShareTechNote, nd] ShareTechNote (n.d.). Quantum computing - bloch sphere. `https://www.sharetechnote.com/html/QC/QuantumComputing_BlochSphere.html`. Accessed: 2024-09-05.

[Shor, 1994] Shor, P. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134.

[Silva, 2018a] Silva, V. (2018a). *Practical Quantum Computing for Developers: Programming Quantum Rigs in the Cloud using Python, Quantum Assembly Language and IBM QExperience*. Apress.

[Silva, 2018b] Silva, V. (2018b). *Practical Quantum Computing for Developers: Programming Quantum Rigs in the Cloud Using Python, Quantum Assembly Language and IBM QExperience*. Apress L.P., New York.

[Valdez and Melin, 2022] Valdez, F. and Melin, P. (2022). A review on quantum computing and deep learning algorithms and their applications. *Soft Computing*.

[Wolfram, 2002] Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media.

[Yanofsky and Mannucci, 2008] Yanofsky, N. S. and Mannucci, M. A. (2008). *Quantum computing for computer scientists*. Cambridge University Press.

[Zhao, 2022] Zhao, J. (2022). Possible implementations of oracles in quantum algorithms. *J. Phys. Conf. Ser.*, 2386(1):012010.