COMPARAÇÃO DE DESEMPENHO DO ALGORITMO DE DEUTSCH-JOZSA NAS LINGUAGENS QUÂNTICAS SILQ E QASM

Mariana Godoy Vazquez Miano[1]
Lucas Gomes Pinheiro[2]
Sthéfanie Costa Amaro[3]
Victor Luis Rodrigues Pereira Ferreira[4]

DOI: 10.47283/244670492023110166

RESUMO
Em virtude da importância atribuída às informações nas últimas décadas, torna-se relevante a vantagem de performance e processamento das informações, algo que pode ser encontrado na computação quântica. O Algoritmo de Deutsch-Jozsa é o primeiro exemplo de algoritmo quântico que oferece uma vantagem exponencial sobre algoritmos clássicos, seja num ambiente local ou em um simulador nuvem. Visando explorar a vantagem da computação quântica, o Algoritmo de Deutsch-Jozsa foi implementado em duas linguagens quânticas. De um lado, a linguagem de alto nível Silq, focada na execução do algoritmo no ambiente local VSCode, a qual oferece sintaxe mais intuitiva e descomputação automática, enquanto a linguagem de baixo nível OpenQASM representará visualmente os circuitos do algoritmo de Deutsch-Jozsa. O objetivo do presente trabalho é evidenciar as diferenças entre linguagens quânticas de alto e baixo nível, bem como incentivar a mudança de paradigma.
PALAVRAS-CHAVE: Programação Quântica. Algoritmo de Deutsch-Jozsa. Desempenho. IBM Quantum. Silq.

ABSTRACT
Due to the importance given to information in the last few decades, a performance and processing advantage of information becomes relevant, something that can be found through quantum computing. The Deutsch-Jozsa Algorithm is the first example of a quantum algorithm that offers an exponential advantage against classical algorithms, whether in a local environment or through cloud simulators. Seeking to explore the advantages of quantum computation, the Deutsch-Jozsa Algorithm was implemented in two quantum programming languages, namely the high level language Silq, focused on the execution of the algorithm on a local environment through VSCode, which offers a cleaner and friendly sintaxe as well as quantum uncomputation, and also in OpenQASM, a low level language meant to interact with quantum circuits, used to better visualize

[1] Docente e pesquisadora da Fatec Americana nos cursos de Tecnologia da Informação. Email: mariana.miano@fatec.sp.gov.br
[2] Aluno do Curso de Tecnologia em Segurança da Informação da Fatec Americana. E-mail: lucas.pinheiro7@fatec.sp.gov.br
[3] Aluna do Curso de Tecnologia em Segurança da Informação da Fatec Americana. E-mail: sthefanie.amaro@fatec.sp.gov.br
[4] Aluno do Curso de Tecnologia em Segurança da Informação da Fatec Americana. E-mail: victor.ferreira31@fatec.sp.gov.br.

the Deutsch-Jozsa Algorithm. This paper aims to make the differences between high and low-level quantum languages clear, as well as incentivize the change to a new paradigm.
KEYWORDS: *Quantum Programming, Deutsch-Jozsa Algorithm. Performance. IBM Quantum. Silq.*

INTRODUÇÃO

A computação quântica é o paradigma de computação multidisciplinar emergente que age como interseção entre campos como a matemática, física quântica e ciência da computação. Computadores clássicos funcionam de maneira binária, podendo ter como entrada e saída de dados apenas zeros e uns, porém devido às propriedades da mecânica quântica, a computação quântica permite fenômenos com a sobreposição (também chamada de superposição) de estados, paralelismo quântico, emaranhamento etc. A partir da exploração desses princípios é possível obter vantagem de performance sobre a sua predecessora, a computação clássica. A origem de computadores quânticos pode ser traçada de volta até os anos 1980 (HASSIJA et. al., 2020, tradução nossa; NIELSEN; CHUANG, 2010, tradução nossa; MIANO, 2020).

Tomando proveito das propriedades da mecânica quântica, um dos pilares para a computação quântica, David Deutsch desenvolveu em 1985 um algoritmo que consegue computar se uma função é constante ou balanceada com apenas uma consulta à função oráculo (algo que na computação clássica levaria no mínimo duas, uma para a constante e outra para a balanceada). Esse algoritmo surgiu como resposta à pergunta "A computação quântica consegue resolver de maneira eficiente problemas sem solução da computação clássica?" (OLIVEIRA et. al., 2021), claramente indicando que sim, pois o princípio da sobreposição permite uma vantagem exponencial de tempo (LACAVA; MIANO, 2018). O Algoritmo de Deutsch-Jozsa é uma junção dos trabalhos de David Deutsch e Richard Josza e se trata de uma generalização do Algoritmo de Deutsch (OLIVEIRA, et. al., 2021).

Para a interação com o hardware quântico são necessárias linguagens de programação quânticas, de maneira análoga à computação clássica. Computadores quânticos são máquinas híbridas, possuindo tanto um computador clássico responsável por mandar instruções, quanto um dispositivo quântico responsável por executá-las (HASSIJA et. al., 2020, tradução nossa). Nos níveis de abstração mais baixos se encontram linguagens como OpenQASM, uma linguagem que representa circuitos quânticos universais e de maneira legível aos seres humanos, possuindo elementos das linguagens C e Assembly (CROSS, et. al., 2017, tradução nossa). Já nas camadas mais altas da máquina quântica residem linguagens como Quipper, Q# e principalmente Silq, a linguagem que será discutida no presente trabalho.

Silq se destaca de suas semelhantes por possuir "descomputação" segura e automática e um código em média 38% menor em relação à Quipper e 48% menor em relação a Q#. Silq também oferece uma sintaxe intuitiva (BICHSEL et. al., 2020, tradução nossa), o que pode ser atrativo para programadores buscando a mudança de paradigma.

O objetivo do presente trabalho é evidenciar as diferenças entre as linguagens de alto nível e baixo nível do paradigma quântico da computação, Silq e QASM respectivamente, a fim de demonstrar o seu ganho de performance em relação à sistemas clássicos, através da execução do Algoritmo de Deutsch-Jozsa no simulador quântico em nuvem da IBM, o *IBM Quantum Experience*, juntamente com o Qiskit, bem como no ambiente de desenvolvimento local VSCode.

O objeto do trabalho é uma comparação de quatro códigos, em pares (função balanceada e função constante) de ambas as linguagens de alto nível Silq e baixo nível OpenQASM, do Algoritmo de Deutsch-Jozsa.

1 COMPUTAÇÃO QUÂNTICA

A computação quântica se baseia no uso e manipulação da mecânica quântica para computar dados. O que torna um computador quântico diferente de um computador denominado "clássico" (baseado no modelo criado por Alan Turing) é sua capacidade de se favorecer de propriedades da mecânica quântica, como a superposição de estados, o emaranhamento quântico etc. (OLIVEIRA et. al., 2021).

A computação quântica teve seu surgimento efetivo na década de 1980, através de um modelo de computador proposto por Paul Benioff, o qual é regido por um sistema quântico microscópico e satisfaz o conceito de uma máquina de Turing (OLIVEIRA et. al., 2021).

1.1 Bit quântico (*qubit*)

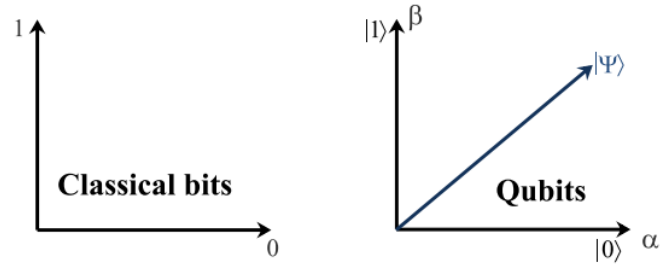
A unidade de medida de uma máquina clássica é denominada *bit* e pode assumir apenas dois valores: 0 ou 1. Entretanto, quando se trata de um computador quântico o processo muda, dado que as leis da física quântica possibilitam fenômenos como sobreposição de estados e emaranhamento quântico. (KOCKUM; NORI, 2019, tradução nossa)

Na computação quântica, a unidade de medida mais simples é denominada *qubit* (*quantum bit*, de maneira abreviada) que possui dois estados principais $|0\rangle$ (estado base) e $|1\rangle$ (estado "animado"), onde $|\ \rangle$ correspondem a notação de Dirac ou *Bra-Ket*. No entanto, em contraste com a computação clássica, existem infinitos estados equivalentes a sobreposições de $|0\rangle$ e $|1\rangle$,

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

onde α e β representam números complexos que satisfazem a equação $|\alpha|^2 + |\beta|^2 = 1$ KOCKUM; NORI, 2019, tradução nossa).

Figura 1 - Representação vetorial de *bits* clássicos e *qubits*



Fonte: (SHARMA, et al, p. 2053, 2021)

É mais simples a visualização gráfica deste esquema através da Esfera de Bloch (KOCKUM; NORI, 2019, tradução nossa).

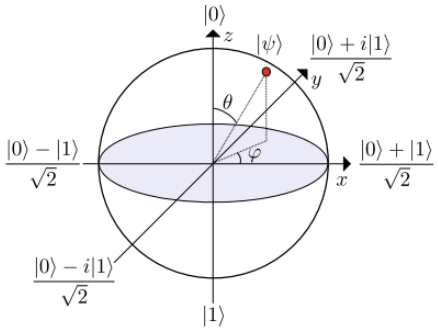
1.2 Sobreposição de estados

A sobreposição de estados é um fenômeno que ocorre enquanto um bit quântico ainda não foi observado. Um estado de sobreposição é um estado intermediário, uma vez que quando um bit quântico é medido sempre estará ou no estado 0 ou 1, enquanto não ocorre a medição o *qubit* pode estar entre 0 e 1 ou ser os dois ao mesmo tempo, com chance de 50% de resultar em 0 e 50% de resultar em 1 (LACAVA; MIANO, 2018).

1.3 Esfera de Bloch

Nielsen e Chuang (2010, p. 15, tradução nossa) discutem como a Esfera de Bloch provê uma ótima visualização dos estados de um *qubit* singular, servindo também como representação geométrica do *qubit* (LACAVA; MIANO, 2018), além de proporcionar uma excelente base de testes para ideias e conceitos sobre computação e informação quântica.

Figura 2 - Representação de um *qubit* na Esfera de Bloch



Fonte: (KOCKUM; NORI, p. 704, 2019)

Segundo (KOCKUM; NORI, p. 704, 2019, tradução nossa):

> "o polo norte da esfera de Bloch mostrada na figura 1 representa o estado base $|0\rangle$ e o polo sul o estado 'animado' $|1\rangle$. Para converter qualquer suposição arbitraria de $|0\rangle$ e $|1\rangle$ para um ponto da esfera, é necessária a parametrização: $\cos\frac{\theta}{2}\ |0\rangle + \ e^{i\varphi} \sin\frac{\theta}{2}\ |1\rangle$ é usada".

1.4 Portas quânticas

De forma similar a computação clássica, os computadores quânticos possuem portas que realizam operações nos circuitos quânticos. Algumas das portas quânticas são análogas às portas lógicas na computação clássica, porém na computação quântica operações de circuito em portas unitárias são reversíveis, o que exige que o número de *qubits* de entrada seja o mesmo que o de saída. As portas quânticas mais utilizadas são os operadores ou matrizes de Pauli, a porta de Hadamard e as portas de fase (LACAVA; MIANO, 2018).

1.4.1 Operadores ou matrizes de Pauli

Os operadores ou matrizes de Pauli são as três principais portas quânticas, denominadas com as notações X, Y e Z (LACAVA; MIANO, 2018).

A porta X funciona de maneira similar em propósito à porta NOT da computação clássica, no sentido de que ela é responsável por inverter o estado do *qubit* (de $|0\rangle$ para $|1\rangle$ e vice-versa). A porta Y além de inverter o estado, também realiza uma inversão de fase, fazendo com que o *qubit* rode o equivalente a π no eixo y da esfera de Bloch. Por fim, a porta Z realiza uma rotação equivalente a π no eixo Z da esfera de Bloch e altera a fase do *qubit* (LACAVA; MIANO, 2018).

Representação das matrizes de Pauli:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad Y = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix} \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

1.4.2 Porta de Hadamard

A porta de Hadamard é responsável por levar um *qubit* ao estado de sobreposição, fazendo com que ela seja uma das mais importantes. Caso a porta de Hadamard seja aplicada a um *qubit* duas vezes, o resultado é o estado inicial do *qubit*. (LACAVA; MIANO, 2018).

Representação da porta de Hadamard:

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

1.4.3 Portas de Fase

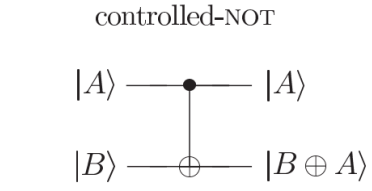
Lacava e Miano (2018) discutem que as portas de fase realizam alterações de fase nos qubits, sendo elas divididas em quatro:

- S, que realiza uma rotação de $\frac{\pi}{2}$ em torno do eixo z;
- S adjunta ou $S^{\dagger}$ que realiza uma rotação de – $\left(\frac{\pi}{2}\right)$ em torno do eixo z;
- T, que realiza uma rotação de $\frac{\pi}{4}$ em torno do eixo z;
- T adjunta ou $T^{\dagger}$ que realiza uma rotação de – $\left(\frac{\pi}{4}\right)$ em torno do eixo z.

1.4.4. Porta CNOT (Controlled-NOT)

NIELSEN e CHUANG (2010, p. 20-21, tradução nossa) descrevem a porta CNOT como a porta prototípica multi-qubit, que também é conhecida como controlled-NOT ou CNOT. Essa porta recebe duas entradas, chamadas de entrada de "dados" e entrada "alvo", respectivamente. A figura 3 ilustra o funcionamento da porta.

Figura 3 – Representação da porta CNOT



$$U_{CN} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Fonte: (NIELSEN; CHUANG, p. 21, 2010)

A linha de cima representa a entrada de dados (ou *qubit* de controle) e a de baixo a entrada alvo (ou *qubit* alvo). A porta performa as seguintes ações: Se o *qubit* de controle estiver como 0, então nada acontece com o *qubit* alvo. Porém se o *qubit* de controle for 1, o *qubit* alvo é invertido. Na forma de equações:

$$|00\rangle \rightarrow |00\rangle ; |01\rangle \rightarrow |01\rangle; |10\rangle \rightarrow |11\rangle; |11\rangle \rightarrow |10\rangle. \quad (2.1)$$

Tudo isso, segundo (NIELSEN; CHUANG, p. 20-21, 2010, tradução nossa).

1.4.5 Porta Identidade (Id ou I)

A porta "Identidade" (também chamada de porta "I", "id" ou porta da ausência) é responsável por garantir que nada ocorrerá com o *qubit* quando ela é aplicada (QISKIT, 2023, tradução nossa).

Isso significa que quando aplicada nos estados $|0\rangle$ e/ou $|1\rangle$ a porta os mantêm.

1.5 Função oráculo

Uma função oráculo é uma operação que é utilizada como entrada de dados em outros algoritmos. Geralmente tais operações são definidas por uma função clássica que segue a estrutura: $f : \{0,1\}^{n} \rightarrow \{1,0\}^{m}$, que recebe $n - bit$ entradas binárias e produz $m - bit$ entradas também binárias (MICROSOFT, 2023a).

1.6 Paralelismo quântico

Nielsen e Chuang (2010, p. 30-31, tradução nossa) descrevem o paralelismo quântico, de maneira geral, como a habilidade de computadores quânticos calcularem uma função *f(x)* para vários valores de *x* simultaneamente.

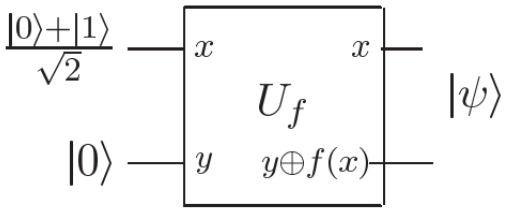
Supondo uma função *f(x)* booleana tal que $f : \{0,1\} \rightarrow \{1,0\}$. Um jeito conveniente de resolver essa função em um computador quântico é considerando um computador de 2 qubits cujos estados começam em $|x, y\rangle$. Utilizando uma sequência adequada de portas lógicas é possível transformar esse estado em $|x, y \oplus f(x)\rangle$, onde $\oplus$ indica a operação de adição em módulo 2. O primeiro registro é chamado de registro de "dados" e o segundo de registro "alvo". É dado o nome U_f à operação unitária definida por

$$|x, y\rangle \rightarrow |x, y \oplus f(x)\rangle \quad (2.2)$$

se *y = 0* então o estado final do segundo qubit é apenas o valor *f(x)*. (NIELSEN; CHUANG, 2010, tradução nossa).

A figura 3 ilustra um circuito quântico que calcula *f(0)* e *f(1)* simultaneamente. U_f transforma entradas como $|x, y\rangle$ para $|x, y \oplus f(x)\rangle$ (NIELSEN; CHUANG, 2010, tradução nossa). Aqui, U_f é considerado uma "caixa preta" (função oráculo) o que significa que não será levado em consideração o que ele faz, apenas que implementa a equação (2.2) no estado $|x\rangle$.

Figura 4 – Circuito U_f



Fonte: (NIELSEN; CHUANG, p. 31, 2010)

No exemplo mostrado acima, U_f é aplicado a uma entrada fora da base computacional. O registro de dados é preparado como sobreposição $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$, que pode ser criada através da atuação da porta de Hadamard sobre $|0\rangle$. O resultado é o seguinte:

$$\frac{|0,\ f(0)\rangle+|1,\ f(1)\rangle}{\sqrt{2}}. \tag{2.3}$$

Segundo Nielsen e Chuang (2010, p. 30-31, tradução nossa), os diferentes termos contem informação sobre *f(0)* e *f(1)*, como se tivessem sido calculados simultaneamente, uma propriedade do paralelismo quântico.

Diferentemente do paralelismo clássico onde vários circuitos precisam ser criados e executados simultaneamente para processar *f(x)*, na computação quântica apenas um circuito contendo *f(x)* é necessário para avaliar a função em múltiplos valores de *x*, se aproveitando da habilidade de um computador quântico poder estar em sobreposições de diferentes estados. Esse procedimento pode ser generalizado facilmente para outras funções com um número arbitrário de bits, utilizando a operação geral conhecida como *Transformada de Hadamard* ou *Transformada de Walsh-Hadamard*. Essa operação se dá aplicando *n* portas de Hadamard em *n* qubits agindo de forma paralela (NIELSEN; CHUANG, 2010, tradução nossa).

A seção 2.2. trata de dois algoritmos quânticos que utilizam paralelismo quântico.

2 ALGORITMOS QUÂNTICOS

Nielsen e Chuang (2010, p. 28-30, tradução nossa) discutem como a computação quântica consegue não apenas simular máquinas clássicas de maneira eficiente, mas também que algoritmos quânticos apresentam vantagem significativa por utilizarem-se de princípios quânticos. Essa vantagem reside no fato de que funções muito mais poderosas podem ser computadas utilizando portas quânticas e *qubits*.

Há duas categorias de algoritmos quânticos voltados para resolver problemas clássicos. Uma delas é baseada na Transformada de Fourier Quântica (QFT) de Shor, enquanto a outra se fundamenta nos Algoritmos de Grover, destinados a realizar buscas quânticas. Os algoritmos que empregam a QFT de Shor têm a finalidade de decompor números grandes em seus fatores primos, um desafio matemático tão intrincado que acabou sendo incorporado à criptografia RSA como uma camada de segurança crucial. Nas suas versões quânticas, o Algoritmo de Shor é capaz de comprometer a segurança da criptografia clássica, uma vez que seu funcionamento envolve a identificação do período de uma função e, em seguida, a dedução dos fatores do valor desejado (MIANO, 2020).

O Algoritmo de Grover consegue localizar um elemento em uma base de dados não ordenada caso a soma de todos os elementos seja conhecida, o que possibilita que a busca de um determinado valor *x* em um banco de dados de tamanho *N* leve menos tempos de maneira

quadrática. Qualquer algoritmo clássico necessitaria a realização de $\Omega(2^n)$ de buscas no pior dos casos, já o Algoritmo de Grover precisa de apenas $O(\sqrt{2^n})$ (QIU, 2022, tradução nossa).

Outros exemplos de algoritmos quânticos são o Algoritmo de Deutsch e o Algoritmo de Deutsch-Jozsa, os mais importantes para o presente estudo e que serão discorridos nas seções seguintes.

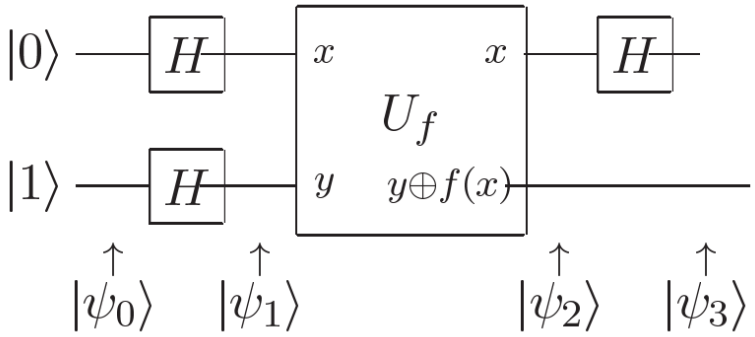
2.1 Algoritmo de Deutsch

O Algoritmo de Deutsch combina paralelismo quântico com uma propriedade da mecânica quântica conhecida como interferência. (NIELSEN; CHUANG, 2010, tradução nossa).

Para exemplificar o Algoritmo de Deutsch, será suposta outra função *f(x)* booleana tal que $f : \{0,1\} \rightarrow \{1,0\}$. Caso a função *f(x)* seja balanceada são possíveis quatro resultados, *f(0)* = 0, *f(0)* = 1, *f(1)* = 0 ou *f(1)* = 1. Caso *f(x)* seja constante, serão possíveis: *f(0)* = *f(1)* = 0 ou *f(0)* = *f(1)* = 1 (OLIVEIRA, et. al., 2021).

Uma simples modificação no circuito da figura 4 é necessário para demonstrar o algoritmo de Deutsch. A porta de Hadamard é usada para preparar o primeiro *qubit* como a sobreposição $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$, porém agora o *qubit y* também será preparado em sobreposição: $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$, aplicando a porta de Hadamard no estado $|1\rangle$ (NIELSEN; CHUANG, 2010, tradução nossa). Portanto, o circuito ficará da seguinte maneira (figura 5):

Figura 5 – Circuito representando o Algoritmo de Deutsch.



Fonte: (NIELSEN; CHUANG, 2010, tradução nossa)

O estado de entrada:

$$|\psi_0\rangle = |01\rangle$$

passa por duas portas de Hadamard para retornar

$$|\psi_1\rangle = \left[\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] \quad (2.3)$$

se for aplicado U_f ao estado $\frac{|x\rangle(|0\rangle-|1\rangle)}{\sqrt{2}}$ então será obtido o estado $(-1)^{f(x)}|x\rangle(|0\rangle - |1\rangle)/\sqrt{2}$. Aplicando U_f em $|\psi_1\rangle$ deixa apenas uma de duas possibilidades:

$$|\psi_2\rangle = \begin{cases} \pm\left[\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] \; se\; f(0) = f(1) \\ \pm\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] \; se\; f(0) \neq f(1) \end{cases} \quad (2.4)$$

A aplicação da porta de Hadamard final portanto retorna:

$$|\psi_3\rangle = \begin{cases} \pm|0\rangle\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] \; se\; f(0) = f(1) \\ \pm|1\rangle\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] \; se\; f(0) \neq f(1) \end{cases} \quad (2.5)$$

Ao perceber que $f(0) \oplus f(1)$ é equivalente a 0 caso f(0) = f(1) e 1 caso contrário, é possível reescrever de maneira concisa o resultado:

$$|\psi_3\rangle = \pm|f(0) \oplus f(1)\rangle\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right], \quad (2.6)$$

O que significa que é necessário determinar a medição do primeiro *qubit*, a fim de obter $f(0) \oplus f(1)$. Nielsen e Chuang (2010, p. 33-34, tradução nossa) concluem que essa é uma propriedade muito interessante, pois através do circuito quântico é possível determinar uma *propriedade global* de *f(x)*, precisamente $f(0) \oplus f(1)$, utilizando apenas uma avaliação de *f(x)*.

Em um computador clássico, a função *f(x)* precisaria ser avaliada para ambos os valores de *x* (0 ou 1) para haver certeza quando determinada se constante ou balanceada. Entretanto devido a propriedade previamente estabelecida do paralelismo quântico, o Algoritmo de Deutsch em um computador quântico consegue avaliar os dois valores de *x* de maneira paralela e simultânea, por meio de uma única medida (OLIVEIRA, et. al., 2021).

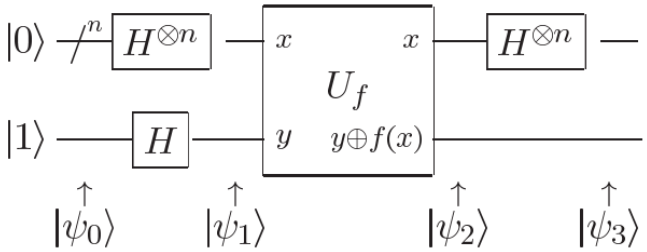
2.3 Algoritmo de Deutsch-Jozsa

O Algoritmo de Deutsch-Jozsa é um algoritmo quântico que determina se uma função f(x) booleana desconhecida de *n bits* é constante ou balanceada. Caso seja constante, todos os resultados serão iguais a 0 ou iguais a 1, caso balanceada, metade dos resultados serão iguais a 0 e a outra metade igual a 1 (LACAVA; MIANO, 2018).

Percebe-se que em princípio, esse algoritmo tem o mesmo objetivo de seu predecessor (Algoritmo de Deutsch), isso porque o Algoritmo de Deutsch-Jozsa nada mais é que uma generalização do Algoritmo de Deutsch, agora permitindo *N* entradas no domínio da função *f(x)*. (OLIVEIRA, et. al., 2021).

A figura 6 ilustra o Algoritmo de Deutsch-Jozsa. Esse é o primeiro exemplo de algoritmo quântico que oferece uma vantagem exponencial sobre algoritmos clássicos (LACAVA; MIANO, 2018).

Figura 6 – Algoritmo de Deutsch-Jozsa



Fonte: (NIELSEN; CHUANG, p. 35, 2010)

2.3. Linguagens de programação Quânticas

Linguagens de programação utilizadas no paradigma quântico.

2.3.1. Silq

Silq é a primeira linguagem quântica a fornecer semântica intuitiva: se um tipo de programa for verificado, sua semântica segue um tipo de "receita" intuitiva que simplesmente

descarta valores temporários. É importante ressaltar que a semântica do Silq é física, ou seja, pode ser realizada em uma máquina quântica de acesso aleatório. No geral, Silq permite expressar algoritmos quânticos de forma mais segura e concisa do que as linguagens de programação quânticas existentes, enquanto normalmente usa apenas metade do número de primitivos quânticos (BICHSEL et. al., 2020, tradução nossa).

De forma similar à programação em linguagens clássicas, linguagens de programação quânticas produzem valores temporários os quais precisam ser eliminados. Contudo, um desafio específico da computação quântica surge, pois remover esses valores introduz medições implícitas que colapsam os estados dos *qubits*, o que por usa vez introduz efeitos colaterais não desejados, principalmente se tratando de *qubits* emaranhados. Então, para remover valores temporários sem induzir medições implícitas, algoritmos quânticos devem "descomputar" esses valores, o que significa mudar seus estados possibilitando ignorá-los sem efeitos colaterais. Ademais, o fato desse processo ser feito manualmente se torna um empecilho em grande parte das linguagens quânticas, dado que linguagens clássicas realizam o descarte de valores temporários automaticamente, o que pode tornar a adoção de linguagens quânticas não tão atrativas para programadores visando a mudança de paradigma (BICHSEL et. al., 2020, tradução nossa).

A linguagem Silq tem como objetivo "construir uma ponte" entre as linguagens quânticas e clássicas, realizando a *descomputação* de maneira automática. Além disso, a sintaxe da linguagem quântica Silq é geralmente menor em relação às suas semelhantes: em média 38% menor que Quipper e 46% menor que Q# (BICHSEL et. al., 2020, tradução nossa).

2.3.2 OpenQASM

Segundo (CROSS, et al, p.3, 2017, tradução nossa): "QASM é uma linguagem de texto simples que descreve algoritmos quânticos".

Ainda de acordo com Cross et al. (2017, p.3, tradução nossa), é ressaltado que a linguagem QASM compartilha elementos com as linguagens clássicas C e Assembly. No início de um código em Open QASM, a primeira linha deve obrigatoriamente conter a sintaxe "OPENQASM M.m;" (sendo "M" maiúsculo para indicar a versão atual e "m" minúsculo para especificar a atualização, por exemplo: 2.0, 2.1, etc.). Essa linha atua como a palavra-chave que define um arquivo OpenQASM, e na versão em questão, está sendo utilizada a 2.0. Quanto à sintaxe, é importante observar que a linguagem desconsidera espaços em branco, e as linhas de código são separadas por ponto e vírgula, além de serem sensíveis a maiúsculas e minúsculas. Para adicionar comentários, é necessário usar um par de barras (//), e eles se estendem até a linha seguinte.

2.4. Qiskit

Conforme QISKIT (2023, nossa tradução) menciona, o Qiskit é uma ferramenta de código aberto destinada à exploração da computação quântica, em níveis como circuitos, pulsos e algoritmos. Além disso, uma variedade de APIs específicas de domínio complementam o núcleo do módulo.

2.5. VSCode

O Visual Studio Code, frequentemente referido como VSCode, é reconhecido como uma eficaz plataforma de edição de códigos. Ele oferece suporte nativo para linguagens como JavaScript, TypeScript e Node.js, porém, sua versatilidade se destaca pela ampla variedade de

extensões disponíveis para suportar outras linguagens, como Java, C#, C++, Python, PHP, entre outras. Além disso, o VSCode é conhecido por ser uma aplicação leve, o que implica que sua operação não exige muitos recursos computacionais ou espaço em disco, sendo capaz de se expandir apenas conforme as necessidades do usuário (MICROSOFT, 2023, tradução nossa).

2.6. IBM Quantum Experience

A IBM é uma empresa multinacional que direciona seus investimentos para o campo da computação quântica, disponibilizando simuladores quânticos em sua plataforma na nuvem. Atualmente, a *Quantum Experience* foi dividida em duas partes: o IBM *Quantum Composer* e o IBM *Quantum Lab*. Esses elementos constituem a plataforma da IBM para computação quântica, que inclui simuladores quânticos na nuvem, além de acesso remoto à máquinas quânticas reais.

3 METODOLOGIA

A metodologia do projeto consiste em coleta de informações quantitativas e qualitativas, pesquisa experimental e revisão bibliográfica. Os materiais consistem em artigos científicos e documentação que abordem assuntos relevantes ao tema principal: algoritmos quânticos (principalmente o Algoritmo de Deutsch-Jozsa), linguagens de programação quântica, diferença entre o alto e baixo nível das linguagens quânticas e o ganho de performance da computação quântica em relação à computação clássica. O desenvolvimento da pesquisa ocorreu na plataforma em nuvem *IBM Quantum Experience* juntamente com a biblioteca Qiskit utilizando a linguagem OpenQASM e no ambiente local VSCode utilizando a linguagem Silq.

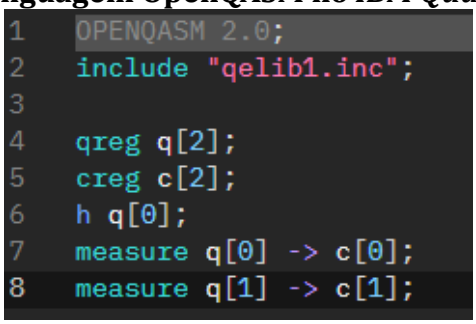
4 RESULTADOS E DISCUSSÕES

A presente seção é dividida em duas partes, na parte inicial são realizados testes simples com o intuito de familiarização com as linguagens quânticas de baixo e alto nível.

4.1. Testes iniciais

Um código gera dois *qubits* (bits quânticos) e é aplicada a porta de Hadamard no primeiro; na sequência é realizada a medição para determinar seus estados. Na linguagem OpenQASM, é necessário estabelecer o número de qubits e bits clássicos, *qreg* e *creg* nas linhas 4 e 5 respectivamente, para depois aplicar portas quânticas e por fim realizar as medições individualmente, como mostra a figura 7 (PINHEIRO et al., 2023).

Figura 7 - Código em linguagem OpenQASM no *IBM Quantum Experience*.



Fonte: Pinheiro, et al. (2023).

Na linguagem Silq, a declaração de variáveis é similar às linguagens clássicas, sendo necessário apenas nomeá-las, declarar seu tipo, aplicar a porta desejada e medir ambos os *qubits* simultaneamente. A figura 8 ilustra parte do código (PINHEIRO, et al., 2023).

Figura 8 - Código em linguagem Silq no *VSCode*.

```
1  def main(){
2      circ1:=0:B;
3      circ2:=1:B;
4      circ1:=H(circ1);
5      return measure(circ1, circ2)
6  }
```

Fonte: Pinheiro, et al. (2023).

4.2. Implementação do Algoritmo de Deutsch-Jozsa

Essa seção do presente trabalho discorrerá sobre a implementação do Algoritmo de Deutsch-Jozsa (abreviado para "ADJ", por conveniência) nas linguagens OpenQASM e Silq. O código será ilustrado e explicado. Vale ressaltar que o objetivo da implementação do ADJ no presente artigo é demonstrar a eficiência de um computador quântico em relação a computadores clássicos, a fim de provar esse princípio, e não avaliar uma função desconhecida como sugere sua aplicação prática.

Serão utilizados dois pares de códigos os quais a diferença é apenas a função oráculo, que determina se a função *f(x)* é constante ou balanceada. A função *f(x)* em sí se dá por: $f : \{0,1\}^3 \rightarrow \{1,0\}$ o que significa que $N = 2^n$ onde n = 3 *qubits*, porém é necessário um *qubit* adicional, o registrador de dados.

Na plataforma IBM *Quantum*, foram realizados 4 testes para a função constante utilizando OpenQASM, através de duas máquinas quânticas reais e um simulador. As máquinas utilizadas foram a "ibm_nairobi" e a "ibm_lagos", no entanto o teste na primeira máquina falhou, logo os testes considerados foram apenas os da segunda máquina. No simulador "simulator_stabilizer" foi realizado 1 teste. Todos os testes tiveram a duração média de uma hora. Enquanto para a função balanceada, ainda em OpenQASM, também foram realizados 4 testes através das 3 máquinas "ibm_nairobi", "ibm_lagos" e "ibm_brisbane" levando em média também uma hora, além do teste no simulador "simulator_stabilizer", levando em média 5 segundos.

É importante ressaltar que a depender da quantidade de pessoas utilizando o simulador ou a máquina, os testes podem levar mais ou menos tempo para serem realizados.

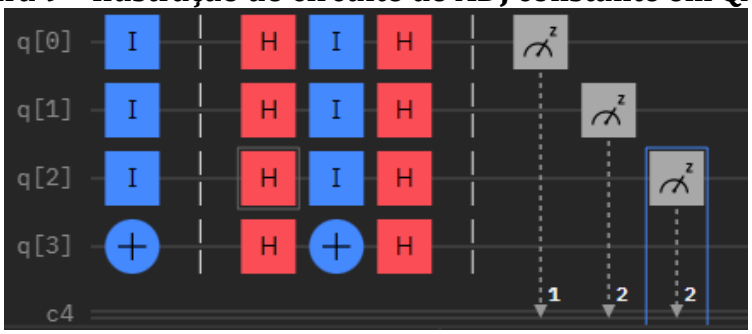
Já no ambiente local VSCode, todos os testes realizados foram relativamente instantâneos, pois por se tratar de um simulador local não há fila de espera nem compartilhamento de hardware. Foram realizados 20 testes para a função constante e 20 para a função balanceada, na linguagem Silq.

No total foram realizados 48 testes.

4.2.1. Linguagem OpenQASM utilizando função constante

A figura 9 mostra a representação gráfica do circuito quântico, já as figuras 10 e 11 os trechos de código que o compõe.

Figura 9 – Ilustração do circuito do ADJ constante em QASM.



Fonte: Elaborado pelos autores através do IBM Q

Primeiramente são atribuídos 4 bits clássicos aos quatro *qubits* através dos comandos *creg* e *qreg* (linhas 5 e 6). Depois, são aplicadas as portas I nos *qubits* 0, 1 e 2 e a porta CNOT no *qubit* 3 (linhas 9, 10, 11 e 12).

O comando "barrier" (linhas 15 e 34) serve apenas para inserir duas linhas verticais separadoras na visualização gráfica do circuito, portanto não possui nenhuma funcionalidade real no circuito, é puramente ilustrativo. Esse comando foi utilizado para organizar cada parte do código e separar a fase de entrada da fase de processamento e saída.

Figura 10 – Início do ADJ constante em QASM.

```
1  OPENQASM 2.0;
2  include "qelib1.inc";
3
4  // Aplicação dos registradores
5  qreg q[4];
6  creg c[4];
7
8  // Aplicação de portas
9  x q[3];
10 id q[2];
11 id q[1];
12 id q[0];
13
14 // Barreira
15 barrier q[0], q[1], q[2], q[3];
16 h q[0];
17 h q[3];
18 h q[2];
19
```

Fonte: Elaborado pelos autores através do IBM Q

Em seguida são aplicadas portas de Hadamard em todos os *qubits* (Linhas 16, 17, 18 e 21) com o intuito de levá-los à sobreposição. Após outra aplicação das portas I nos *qubits* 0, 1 e 2 e da porta CNOT no *qubit* 3 (linhas 22, 23, 24 e 25), o processo de aplicação da porta de Hadamard em todos os *qubits* se repete (linhas 28, 29, 30 e 31).

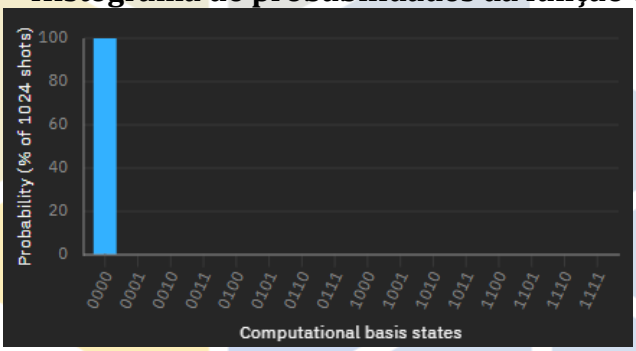
Figura 11 - Continuação do ADJ constante em QASM.

```
20  // Aplicação do oraculo constante
21  h q[1];
22  id q[0];
23  x q[3];
24  id q[2];
25  id q[1];
26
27  // Aplicação de portas
28  h q[0];
29  h q[3];
30  h q[2];
31  h q[1];
32
33  // Barreira
34  barrier q[0], q[1], q[2], q[3];
35
36  measure q[0] -> c[1];
37  measure q[1] -> c[2];
38  measure q[2] -> c[2];
```

Fonte: Elaborado pelos autores através do IBM Q

Após a execução do Algoritmo, são medidos os *qubits* 0, 1 e 2. A figura 12 ilustra o histograma de probabilidades dos resultados de todos os *qubits*, onde a probabilidade de retornar o estado 0 é de 100%. Com isso é possível concluir que a função é constante, pois não há probabilidade de outro resultado.

Figura 12 – Histograma de probabilidades da função constante

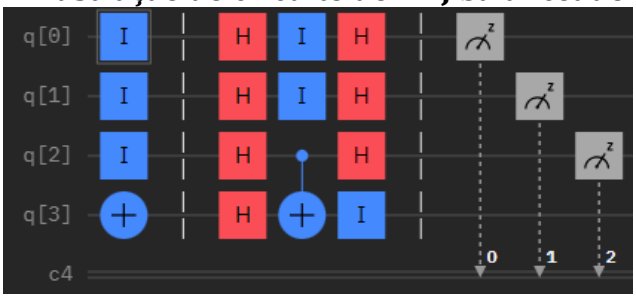


Fonte: Elaborado pelos autores através do IBM Q

4.2.2. Linguagem OpenQASM utilizando função balanceada

A única diferença no código da função balanceada reside na parte de execução da função oráculo do Algoritmo. A figura 13 mostra a representação gráfica do circuito quântico, já as figuras 14 e 15 os trechos de código que o compõe.

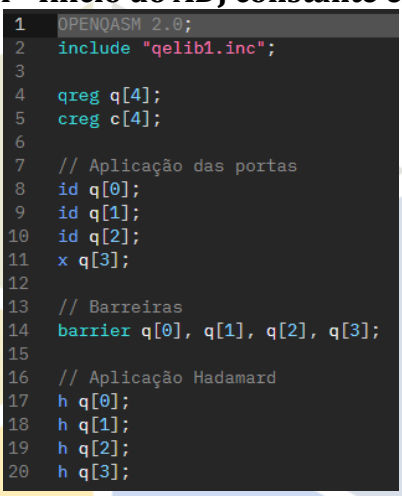
Figura 13 – Ilustração do circuito do ADJ balanceado em QASM



Fonte: Elaborado pelos autores através do IBM Q

É possível observar, tanto pelo código quanto pela representação do circuito que a função oráculo do algoritmo balanceado aplica a porta CNOT em ambos os *qubits* 2 e 3 após a primeira aplicação das portas de Hadamard. Essa mudança na função oráculo permite que a função retorne o resultado "balanceado".

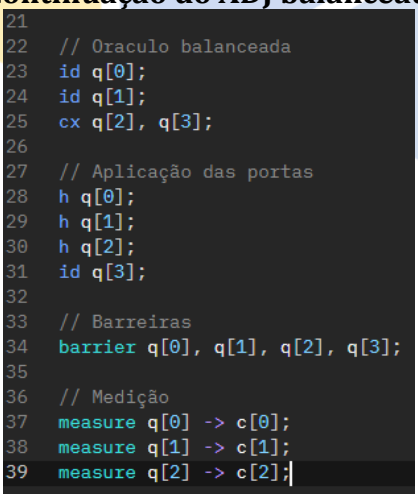
Figura 14 - Início do ADJ constante em QASM.



```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[4];
creg c[4];

// Aplicação das portas
id q[0];
id q[1];
id q[2];
x q[3];

// Barreiras
barrier q[0], q[1], q[2], q[3];

// Aplicação Hadamard
h q[0];
h q[1];
h q[2];
h q[3];
```

Fonte: Elaborado pelos autores através do IBM Q

Figura 15 - Continuação do ADJ balanceado em QASM.

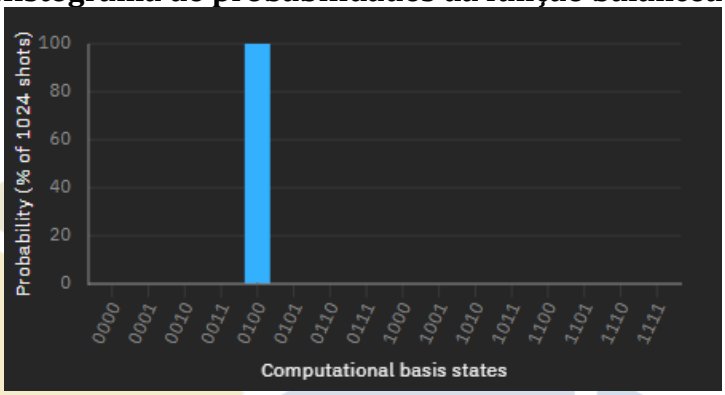


```

// Oraculo balanceada
id q[0];
id q[1];
cx q[2], q[3];

// Aplicação das portas
h q[0];
h q[1];
h q[2];
id q[3];

// Barreiras
barrier q[0], q[1], q[2], q[3];

// Medição
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

Fonte: Elaborado pelos autores através do IBM Q

Como é possível observar no histograma de probabilidades dos resultados de todos os *qubits* (figura 16), o código retorna 1 *qubit* com o resultado 1 e dois com o resultado 0 em 100% dos casos. Como pelo menos um dos *qubits* se encontra em um estado diferente de 0, é possível afirmar que o ADJ funcionou e sua performance foi satisfatória tendo em vista que foi necessária apenas uma única medida.

Figura 16 – Histograma de probabilidades da função balanceada em QASM



Fonte: Elaborado pelos autores através do IBM Q

4.2.3. Linguagem Silq utilizando função constante

Na linguagem de alto nível Silq, o princípio e as operações do ADJ são as mesmas, porém a linguagem permite mais liberdade ao usuário, uma vez que é possível nomear as variáveis de maneira arbitrária, desde que siga a seguinte estrutura: "nome : = estado inicial : tipo da variável ;"

Na figura 17, é possível observar a criação das variáveis (linhas 2, 3, 4 e 5) nomeadas de maneira similar propositalmente, a fim de demonstrar que em uma linguagem quântica de alto nível e devido às funcionalidades de computadores em geral, é possível copiar e colar a maior parte do código, modificando apenas o que é necessário. O intuito é incentivar programadores de linguagens clássicas a adaptar linguagens quânticas, caso estejam em dúvida por se tratar de outro paradigma.

A aplicação das portas segue a mesma estrutura dos códigos anteriores, 3 portas Identidade ("I") nos 3 primeiros *qubits* e uma CNOT (representada por "X") no quarto *qubit* (linhas 7, 8, 9 e 10).

Figura 17 – Início do ADJ constante em Silq

```
def main() {
    circ0:=0:B;
    circ1:=0:B;
    circ2:=0:B;
    circ3:=0:B;

    circ0:=I(circ0);
    circ1:=I(circ1);
    circ2:=I(circ2);
    circ3:=X(circ3);

```

Fonte: Elaborado pelos autores através do VSCode

Depois, conforme a figura 18, se inicia a função oráculo, através da aplicação das portas de Hadamard ("H") em todos os *qubits* (linhas 13, 14, 15 e 16 e posteriormente 23, 24, 25 e 26), aplicação das portas Identidade e CNOT e reaplicação das portas de Hadamard.

Figura 18 - Continuação do ADJ constante em Silq

```
    // Caixa Preta
    circ0:=H(circ0);
    circ1:=H(circ1);
    circ2:=H(circ2);
    circ3:=H(circ3);

    circ0:=I(circ0);
    circ1:=I(circ1);
    circ2:=I(circ2);
    circ3:=X(circ3);

    circ0:=H(circ0);
    circ1:=H(circ1);
    circ2:=H(circ2);
    circ3:=H(circ3);

```

Fonte: Elaborado pelos autores através do VSCode

Por fim, é necessário utilizar a função *measure* para medir os estados dos *qubits* e retorná-los na função *main*, como mostra a figura 19.

Figura 19 – Função *measure* do ADJ constante

```
    // Medição
    return measure(circ0, circ1, circ2, circ3);
}
```

Fonte: Elaborado pelos autores através do VSCode

O resultado do processo são os estados dos 4 *qubits*, que correspondem a 0, o que significa que a função é constante (figura 20).

Figura 20 – Output do código

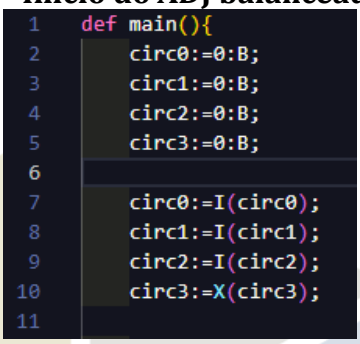
```
(0,0,0,0)
```

Fonte: Elaborado pelos autores através do VSCode

4.2.4. Linguagem Silq utilizando função balanceada

Nas figuras 21, 22, 23 e 24 é possível observar o processo de implementação do Algoritmo de Deutsch-Jozsa balanceado. Novamente, a única alteração se dá pela função oráculo, aplicando a porta CNOT e Identidade à diferentes *qubits*.

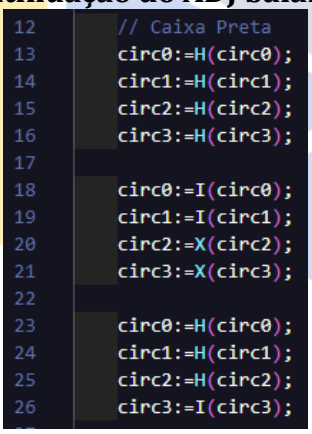
Figura 21 - Início do ADJ balanceado em Silq



Fonte: Elaborado pelos autores através do VSCode

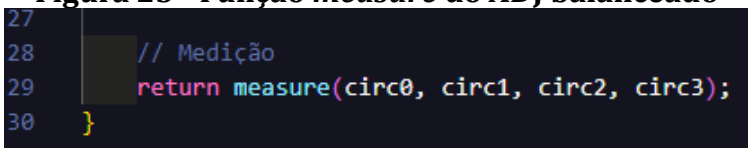
A porta CNOT é aplicada aos *qubits* 2 e 3, tendo o *qubit* 2 como alvo e 3 como o registro de dados. Após a operação o *qubit* 3 passa pela porta Identidade ao invés da de Hadamard.

Figura 22 - Continuação do ADJ balanceado em Silq



Fonte: Elaborado pelos autores através do VSCode

Figura 23 - Função *measure* do ADJ balanceado



Fonte: Elaborado pelos autores através do VSCode

A figura 21 ilustra a saída do código, onde dos 3 *qubits* utilizados para a entrada de dados, apenas 1 retornou resultado 1, assim comprovando o funcionamento do algoritmo.

Figura 24 – Output do código

Fonte: Elaborado pelos autores através do VSCode

Através da realização dos testes nos quatro códigos, foi possível observar o funcionamento do Algoritmo de Deutsch-Jozsa tanto para uma função balanceada quanto uma constante. Nas duas funções a entrada de dados é dada por $|x_0x_1x_2\rangle$ e o resultado, dependendo de qual função se trata, varia entre $|000\rangle$ (constante) ou $|100\rangle$ (balanceada). Utilizando esse algoritmo é possível obter o resultado de uma dada função com apenas uma única consulta à função oráculo. A função oráculo têm relação direta com a função f*(x)* a qual está sendo examinada, dado que *f(x)* é desconhecido, é preciso apenas saber que o oráculo implementa a operação $|x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$.

Em um ambiente clássico, seria necessário que a função oráculo fosse consultada no mínimo duas vezes, pois é necessário "perguntar" se a função é constante ou balanceada. Isso não ocorre na computação quântica devido à propriedade de sobreposição, discutida no tópico 2.1.2.

CONSIDERAÇÕES FINAIS

O presente trabalho teve como objetivo principal demonstrar o ganho de performance da computação quântica em relação a computação clássica, bem como destacar a linguagem de programação de alto nível Silq em relação às suas similares, além de estabelecer uma relação entre linguagens quânticas de alto e baixo nível, a qual já existe no paradigma clássico.

Como discorrido anteriormente, o Algoritmo de Deutsch-Jozsa foi o primeiro a demonstrar objetivamente o ganho de performance do paradigma quântico de computação. Apesar de não possuir muitas aplicações práticas, o intuito de implementar justamente esse algoritmo é voltar a atenção do leitor ao ecossistema quântico como um todo.

A computação quântica oferece vantagens em campos diversos como a Segurança da informação por meio da Criptografia, Machine Learning e Inteligência Artifical (assunto de suma relevância na data de escrita do artigo) e até mesmo em áreas como a biomedicina. A criptografia quântica tem o potencial de mudar os sistemas de encriptação como são conhecidos atualmente, dado que a computação quântica é extremamente eficiente em "quebrar" a criptografia clássica, especialmente o sistema RSA. Além disso, a Distribuição de Chaves Quânticas (DCQ) é a mais madura das tecnologias quânticas e conforme os algoritmos crescem estão sendo criadas mais medidas de segurança (PIRANDOLA et al., 2020, tradução nossa).

Através dos resultados do presente estudo foi possível observar uma prova de princípio. Ao executar o Algoritmo de Deutsch-Jozsa como implementado e detalhado na seção 4, a saída do código é 000 ou 100, o que indica se a função é constante ou balanceada, respectivamente. Computadores clássicos necessitariam de 2 consultas à função oráculo do algoritmo para determinar os aspectos da função f(x) desejada, isso significa que através do Algoritmo de Deutsch-Jozsa é possível provar que existem tarefas na computação quântica que podem ser executadas utilizando tempo exponencialmente menor quando comparado a computação clássica. Esse ganho de performance também pode ser observado no Algoritmo de Shor, desenvolvido por Peter Shor em 1994.

Uma vez que a Lei de Moore chega ao seu fim (NIELSEN; CHUANG, 2010, tradução nossa), algoritmos como o de Deutsch-Jozsa apontam para a computação quântica como sucessora à computação clássica, especialmente no ramo da criptografia.

Uma vez que sistemas quânticos estejam suficientemente desenvolvidos e acessíveis, a sua implementação tende a se tornar iminente. Por exemplo, em setembro de 2023, a instituição de educação SENAI do estado de São Paulo obteve o primeiro computador quântico para fins educacionais.

REFERÊNCIAS

BICHSEL, B.; VECHEV, M.; BAADER, M.; GEHR, T. In: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. 2020. p. 286-300.

CROSS, A. W. et al. Open quantum assembly language. **arXiv preprint arXiv:1707.03429**, 2017.

HASSIJA, V. et al. Present landscape of quantum computing. **IET Quantum Communication**, v. 1, n. 2, p. 42-48, 2020.

KOCKUM, A.F.; NORI, F. Quantum Bits with Josephson Junctions. In: Tafuri, F. (eds) **Fundamentals and Frontiers of the Josephson Effect. Springer Series in Materials Science**, vol 286. Springer, Cham, 2019. Disponível em: https://doi.org/10.1007/978-3-030-20726-7_17. Acesos em: 22 jun. 2023.

LACAVA, L.; MIANO, M. G. V. Implementação do algoritmo quântico Deustch-Josza em linguagem funcional e no simulador IBM Q Experience. **Revista Tecnológica da Fatec Americana**, Americana/SP, v. 6, n. 02, dez. 2018. Disponível em: https://fatec.edu.br/revista/index.php/RTecFatecAM/article/view/186. Acesso em 10 jun. 2023.

MIANO, M. G. V. Aplicação de protocolos quânticos e algoritmo de Shor para a segurança da informação. **Revista Tecnológica da Fatec Americana**, Americana/SP, v. 8, n. 01, ago. 2020. Disponível em: https://fatec.edu.br/revista/index.php/RTecFatecAM/article/view/233. Acesso em: 15 jun. 2023.

MICROSOFT. **Azure Quantum: Work with and define quantum oracles**. 2023a. Disponível em: https://learn.microsoft.com/en-us/azure/quantum/concepts-oracles. Acesso em: 31 out. 2023.

MICROSOFT. **Visual Studio Code: Getting Started**. 2023. Disponível em: https://code.visualstudio.com/docs. Acesso em: 20 mai. 2023.

NIELSEN, M. A; CHUANG, I. L. **Quantum Computation and Quantum Information**. Cambridge: Cambridge University Press, 2010. 698 p.

OLIVEIRA, A. N. et al. **Quantum Algorithms in IBMQ Experience: Deutsch-Jozsa algorithm**. arXiv preprint arXiv:2109.07910, 2021.

PINHEIRO, L. G.; AMARO, S. C.; FERREIRA V. L. R. P.; MIANO, M. G. V. Uso de linguagens quânticas em plataformas híbridas. **Boletim Técnico da Faculdade de Tecnologia de São Paulo**: Resumos do 25º Simpósio de Iniciação Científica e Tecnológica (SICT-2023), São Paulo/SP, v. 56, n. 01, out. 2023. ISSN: 1518-9082. p. 73.

PIRANDOLA, S. et al. Advances in quantum cryptography. **Advances in optics and photonics**, v. 12, n. 4, p. 1012-1236, 2020.

QISKIT, T. D. Qiskit 0.43.0 documentation. 2023. Disponível em: https://qiskit.org/documentation/. Acesso em 07 mai. 2023.

QIU, D.; LUO, L.; XIAO, L. Distributed Grover's algorithm. **arXiv preprint arXiv:2204.10487**, 2022.