UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Gabriel da Silva Cardoso

OTIMIZAÇÃO DE CIRCUITOS QUÂNTICOS UTILIZANDO CÁLCULO-ZX NA PLATAFORMA KET

Florianópolis, Santa Catarina – Brasil
2024

Gabriel da Silva Cardoso

OTIMIZAÇÃO DE CIRCUITOS QUÂNTICOS UTILIZANDO CÁLCULO-ZX NA PLATAFORMA KET

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Ciências da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador(a): Evandro Chagas Ribeiro da Rosa, Me.

Florianópolis, Santa Catarina – Brasil
2024

Notas legais:

Não há garantia para qualquer parte do software documentado. Os autores tomaram cuidado na preparação desta tese, mas não fazem nenhuma garantia expressa ou implícita de qualquer tipo e não assumem qualquer responsabilidade por erros ou omissões. Não se assume qualquer responsabilidade por danos incidentais ou consequentes em conexão ou decorrentes do uso das informações ou programas aqui contidos.

Catalogação na fonte pela Biblioteca Universitária da Universidade Federal de Santa Catarina.
Arquivo compilado às 01:46h do dia 8 de julho de 2024.

Gabriel da Silva Cardoso
Otimização de circuitos quânticos utilizando cálculo-zx na Plataforma Ket / Gabriel da Silva Cardoso; Orientador(a), Evandro Chagas Ribeiro da Rosa, Me. – Florianópolis, Santa Catarina – Brasil, julho de 2024.
79 p.

Trabalho de Conclusão de Curso – Universidade Federal de Santa Catarina, INE – Departamento de Informática e Estatística, CTC – Centro Tecnológico, Curso de Graduação em Ciências da Computação.

Inclui referências

1. Otimização de Circuitos Quânticos, 2. Computação Quântica, 3. Cálculo-ZX, I. Evandro Chagas Ribeiro da Rosa, Me. II. Curso de Graduação em Ciências da Computação III. Otimização de circuitos quânticos utilizando cálculo-zx na Plataforma Ket

CDU 02:141:005.7

Gabriel da Silva Cardoso

OTIMIZAÇÃO DE CIRCUITOS QUÂNTICOS UTILIZANDO CÁLCULO-ZX NA PLATAFORMA KET

Este(a) Trabalho de Conclusão de Curso foi julgado adequado(a) para obtenção do Título de Bacharel em Ciências da Computação, e foi aprovado em sua forma final pelo Curso de Graduação em Ciências da Computação do INE – Departamento de Informática e Estatística, CTC – Centro Tecnológico da Universidade Federal de Santa Catarina.

Florianópolis, Santa Catarina – Brasil, julho de 2024.

Profa. Dra. Lúcia Helena Martins Pacheco
Coordenadora do Curso de Graduação em Ciências da Computação

Banca Examinadora:

Evandro Chagas Ribeiro da Rosa, Me.
Orientador(a)
Universidade Federal de Santa Catarina – UFSC

Profa. Dra. Jerusa Marchi
Coorientadora
Universidade Federal de Santa Catarina

Profa. Dra. Juliana Vizzotto
Avaliadora
Universidade Federal de Santa Maria

Prof. Dr. Rafael de Santiago
Avaliador
Universidade Federal de Santa Catarina

Dedico este TCC a todos que se fizeram presentes na minha vida.

AGRADECIMENTOS

Agradeço profundamente a minha familia, por acreditarem em mim e me proporcionarem essa oportunidade. Sou grato aos meus colegas de equipe no LabSEC, pelas enriquecedoras conversas na hora do café, e aos amigos do LIAA, cujos momentos compartilhados guardo para sempre nas minhas memórias, próximos ao meu coração. Minha gratidão se estende à Moara, por estar ao meu lado na minha jornada de autodescoberta. Também agradeço aos *parseros*, que me acompanharam e suportaram minhas exigências nos trabalhos e ao meu amigo de infância Khalil, pelas noites que passamos configurando servidores e pelas conversas técnicas que me moldaram como cientista da computação. Por fim, sou imensamente grato a todos os colegas de computação, que se tornaram grandes amigos e tiveram um enorme impacto na minha vida.

Também gostaria de expressar meus sentimentos pelas minhas amigas que me acolheram quando precisei, em especial Beatriz Fasolo e a Larissa. Sou muito grato pelo suporte que recebi.

Gostaria de agradecer ao Laboratório de Segurança em Computação (LabSEC), por formar o profissional que sou e pelas portas que abriu.

Finalmente, agradeço a Professora Jerusa, a minha banca, e, em especial, ao meu orientador, Evandro Chagas, por ter me introduzido ao vasto horizonte da computação quântica.

RESUMO

A Computação Quântica é uma área relevante de pesquisa porque promete um ganho de desempenho para uma classe de problemas da computação. Alguns desses problemas são a fatoração de números primos, possível pelo algoritmo de Shor e a busca em listas desordenadas, com o algoritmo de Grover. No entanto, há uma série de desafios que precisam ser superados para que seja viável a execução desses algoritmos em hardware quântico. Um deles é o ruído, que em computação quântica refere-se a perturbações indesejadas nos estados quânticos, que comprometem a precisão dos cálculos e a confiabilidade dos resultados. Para contornar esse problema, as perspectivas futuras para computação quântica se encontram na correção de erros usando portas quânticas tolerantes a falhas. A adoção em larga escala desses protocolos encerraria a era quântica atual NISQ (Noisy intermediate-scale quantum era), marcada pela computação ruidosa. No entanto, nestes protocolos algumas portas tem custo de implementação desproporcionalmente maior que outras, em especial a porta T. Visto isso, é responsabilidade dos compiladores de circuitos quânticos otimizar os circuitos de forma a reduzir o uso dessas portas que consomem mais recursos. Neste trabalho, explora-se o cálculo-ZX como forma de representar e otimizar circuitos quânticos e é feito um levantamento do estado da arte de técnicas de otimização na literatura com essa ferramenta. Um algoritmo de otimização é selecionado e implementado na Plataforma de Desenvolvimento Quântico Ket. Para esta implementação ser viável e não comprometer a arquitetura da Plataforma com a representação-ZX, um transpilador é implementado de forma a permitir a comunicação entre código ket e diagramas-ZX via linguagem de assembly quântica OpenQASM 2.0. Os resultados finais são analisados conforme a redução do número de portas T, quantidade total de portas lógicas, caminho crítico do circuito, quantidade de portas de 2-qubits e tempo de execução da otimização.

Palavras-chaves: Otimização de Circuitos Quânticos. Computação Quântica. Cálculo-ZX.

ABSTRACT

Quantum Computing is a relevant research area because it promises performance gains for a class of computational problems. Some of these problems include prime number factorization, made possible by Shor's algorithm, and unstructured list search, with Grover's algorithm. However, several challenges need to be overcome to make the execution of these algorithms on quantum hardware feasible. One of these challenges is noise, which in quantum computing refers to unwanted disturbances in quantum states that compromise the accuracy of calculations and the reliability of results. To address this issue, the future of quantum computing lies in error correction using fault-tolerant quantum gates. The widespread adoption of these protocols would mark the end of the current NISQ (Noisy Intermediate-Scale Quantum) era, characterized by noisy computation. However, within these protocols, some gates have a disproportionately higher implementation cost than others, particularly the T gate. Therefore, it is the responsibility of quantum circuit compilers to optimize circuits to minimize the use of these resource-intensive gates. In this work, ZX-calculus is explored as a way to represent and optimize quantum circuits, and a survey of the state-of-the-art optimization techniques in the literature using this tool is conducted. An optimization algorithm is selected and implemented on the Ket Quantum Development Platform. To make this implementation viable and not compromise the Platform's architecture with the ZX representation, a transpiler is implemented to enable communication between Ket code and ZX diagrams via the OpenQASM 2.0 quantum assembly language. The final results are analyzed in terms of the reduction in the number of T gates, total number of logic gates, critical path of the circuit, number of 2-qubit gates, and optimization execution time.

Keywords: Quantum Circuit Optimization. Quantum Computation. ZX-Calculus.

LISTA DE FIGURAS

LISTA DE CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

Convencionalmente, a representação de algoritmos quânticos é feita mediante circuitos que descrevem as operações lineares a serem feitas. O cálculo-ZX, introduzido por Coecke e Duncan (2011), é uma forma alternativa de representar os mapas lineares dos circuitos quânticos, sendo uma linguagem de diagramação visual que permite a simplificação dos diagramas via regras de reescrita. Esse trabalho aborda otimizações de circuitos que emergem aplicando tais regras, em específico, é abordado o algoritmo apresentado por Duncan *et al.* (2020) e explora sua implementação na plataforma de computação quântica Ket (DA ROSA; DE SANTIAGO, 2021).

A plataforma Ket[1] foi desenvolvida para simplificar o processo de desenvolvimento de aplicações quânticas híbridas, abordando as limitações de interação entre computadores clássicos e quânticos. Ela oferece uma abordagem integrada no ecossistema Python, proporcionando aos desenvolvedores uma ferramenta acessível para explorar os desafios e oportunidades da computação quântica. Essa plataforma possui um simulador de computador quântico e um compilador para gerar código para o mesmo.

Circuitos quânticos que utilizam portas lógicas apenas do conjunto de Clifford, constituído de unitários gerados a partir de composições das portas H, S e CNOT, podem ser eficientemente simulados em computadores clássicos (AARONSON; GOTTESMAN, 2004) e podem ser implementados de maneira eficiente usando códigos de correção de erro (RAUSSENDORF; HARRINGTON, 2007). No entanto, para este conjunto alcançar a universalidade [2], é necessária a adição de uma porta fora do conjunto de Clifford, como a porta T.

A minimização do número de portas T num circuito é de suma importância porque, diferente das portas do conjunto de Clifford, essas portas não podem ser eficientemente simuladas e sua implementação com códigos de correção de erro requer uma ordem de magnitude a mais de recursos para serem implementadas (CAMPBELL, Earl T.; TERHAL, Barbara M.; VUILLOT, 2017b) (GIDNEY; FOWLER, 2019). Esta não é a única métrica existente para a otimização, como será visto na seção 2.2, porém é a mais impactante dada as perspectivas futuras de correções de erro, que encerrariam a era atual de computação quântica, marcada por software ruidoso.

---

[1] Plataforma Ket disponível em: https://gitlab.com/quantum-ket/ket

[2] No contexto da computação quântica, universalidade refere-se à capacidade de realizar qualquer computação fisicamente possível (DEUTSCH; BARENCO; EKERT, 1995)

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Considerando o exposto, o objetivo deste trabalho é adicionar suporte à uma rotina de otimização ao compilador da plataforma Ket para a redução do número de portas T do circuito final, focando na exploração do cálculo-ZX e suas regras de otimização.

1.1.2 Objetivos Específicos

i Revisão do estado da arte para algoritmos de otimização de circuitos quânticos que utilizam o cálculo-ZX.

ii Identificar bibliotecas e ferramentas que utilizam cálculo-ZX para implementação na plataforma Ket.

iii Permitir interoperabilidade da plataforma Ket com bibliotecas de cálculo-ZX.

iv Incorporar suporte para a otimização de circuitos quânticos na plataforma Ket.

v Validar a equivalência de circuitos quânticos otimizados com os originais.

vi Identificar limitações no processo de otimização por cálculo-ZX.

1.2 METODOLOGIA

Iniciamos o trabalho apresentando a problemática da otimização de circuitos quânticos, mostrando sua relevância e suas perspectivas dado o contexto atual da computação quântica. Para isso, realizamos um levantamento do estado da arte de métodos de otimização de circuitos na literatura e escolhemos uma rotina de otimização para implementação na Plataforma Ket.

Em seguida, apresentamos o embasamento teórico para o cálculo-ZX, a linguagem de diagramação utilizada pelo algoritmo para fazer otimizações no circuito. Apresentamos o processo de conversão de circuitos para cálculo-ZX, de otimização e de extração de volta para um circuito quântico.

Finalmente, apresentamos a implementação do software proposto que integra a otimização de circuitos na Plataforma Ket, e em seguida fazemos análise dos resultados obtidos. Comparamos a eficiência dos circuitos otimizados com os originais, demonstrando a eficácia do método utilizado no contexto da computação quântica.

1.3 APRESENTAÇÃO DO TRABALHO

A seguir está descrito como o trabalho estará organizado. O segundo capítulo apresentará diversos conceitos obrigatórios para o entendimento completo do traba-

lho. No terceiro capítulo será apresentada a implementação do software proposto. O capítulo de número quatro apresentará os resultados obtidos. Por quinto capítulo apresentará conclusões mediante os resultados obtidos neste trabalho junto com possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, os principais conceitos para a realização deste trabalho são apresentados. Este trabalho fundamenta-se em princípios de computação quântica, e para uma abordagem mais aprofundada recomendamos os trabalhos apresentados por Nielsen e Chuang (2010) e Pollachini (2018).

Aqui serão apresentados os conceitos principais para a realização deste trabalho, iniciando com uma breve introdução a circuitos quânticos na Seção 2.1 e em seguida a otimização de circuitos quânticos é apresentada a partir de uma revisão da literatura na Seção 2.2, explicando quais métricas são relevantes nesse contexto e qual algoritmo será usado.

Com o algoritmo escolhido, seu embasamento teórico é apresentado numa introdução ao cálculo-ZX é feita na seção 2.3, mostrando como seus diagramas podem representar qualquer circuito quântico e introduzindo conceitos fundamentais como aranhas e regras de reescrita. Com isso feito, o algoritmo de otimização de diagramas-ZX abordado por este trabalho é apresentado na seção 2.4 junto de suas regras de otimização. Essa seção acompanha uma introdução aos conceitos de *flow* e as restrições necessárias para ser possível extrair o circuito otimizado. Detalhamos o algoritmo de extração de um diagrama-ZX com *flow* para um circuito quântico na seção 2.5.

Por fim, apresentamos o QASM e a plataforma Ket nas Seções 2.6 e 2.7, respectivamente, relevantes para a implementação da parte prática deste trabalho. Sendo o primeiro a representação intermediária de circuitos quânticos a ser utilizada para interoperabilidade entre bibliotecas e o segundo a plataforma de desenvolvimento quântico onde a otimização será implementada.

2.1 CIRCUITOS QUÂNTICOS

O circuito quântico é um dos modelos de computação quântica, análogo aos circuitos clássicos na computação tradicional. Eles consistem em uma sequência de portas quânticas, as quais são operações que manipulam qubits — as unidades básicas de informação quântica. As portas quânticas exploram princípios da mecânica quântica, como superposição e emaranhamento. Cada porta aplica uma transformação unitária específica ao estado dos qubits, permitindo a execução de algoritmos quânticos quando organizadas em uma sequência.

Como consequência das portas lógicas serem unitárias, todas as portas lógicas quânticas são reversíveis, ou seja, para todo circuito quântico existe um circuito inverso que retorna as entradas originais do circuito.

As portas lógicas quânticas podem interagir com um ou múltiplos qubits ao mesmo tempo, e algumas das portas lógicas mais comuns estão representadas na Figura 1 com suas respectivas matrizes. Um exemplo de circuito quântico é dado na Figura 2.

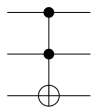
Operator	Gate(s)	Matrix
Pauli-X (X)		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled Z (CZ)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

Figura 1 – Portas lógicas quânticas. Fonte (COMMONS, 2022).

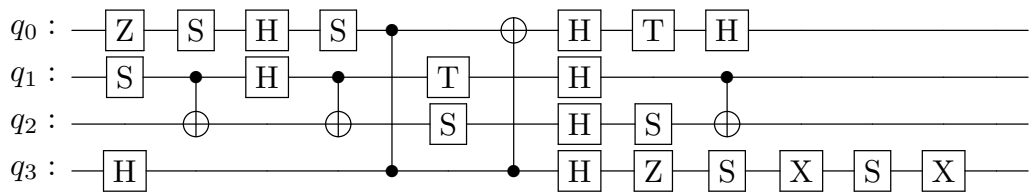


Figura 2 – Exemplo de circuito quântico.

A capacidade de um computador quântico de executar qualquer algoritmo é referida como computação universal, similar à noção de completude de Turing em computação clássica. Para alcançar essa universalidade, é necessário um conjunto de portas lógicas quânticas que seja completo, permitindo a construção de qualquer operação unitária desejada. Um dos conjuntos mais estudados é o conjunto de portas Clifford+T, sendo o conjunto que pode ser gerado pelas portas H, S e CNOT, e a porta T. Enquanto as portas Clifford (H, S, CNOT) são suficientes para correção de erros e algumas operações básicas, a inclusão da porta T é essencial para alcançar

a universalidade, permitindo a realização de cálculos arbitrariamente complexos. Este conjunto é fundamental para a implementação de algoritmos quânticos em hardware quântico e para a otimização de circuitos quânticos.

O conjunto Clifford+T, além de ser universal, é composto por portas lógicas tolerantes a erros. Isso significa que é possível criar códigos de correção de erro para tornar circuitos compostos por esse conjunto tolerantes a erros devido ao ruído de computadores quânticos (NIELSEN; CHUANG, 2010) (AMY; MASLOV; MOSCA, 2014).

Como a computação quântica é limitada por hardware ruidoso, compiladores quânticos tem a responsabilidade de otimizar o código para as limitações da arquitetura. Neste trabalho, focaremos na otimização de portas T, ou seja, na minimização da quantidade de portas lógicas T em um algoritmo quântico. A otimização de portas T é considerado um problema NP-Completo (WETERING; AMY, 2023) e é relevante no contexto de computação quântica tolerante a falhas, que requer esquemas de correção de erros (CALDERBANK; SHOR, 1996; AHARONOV; BEN-OR, 1997) cuja complexidade é dominada por portas lógicas que não pertencem ao conjunto de Clifford (CAMPBELL, Earl T; TERHAL, Barbara M; VUILLOT, 2017a). Por exemplo, o custo de implementar as portas T nesses protocolos é duas ordens de grandeza maior do que os das portas CNOT. (GIDNEY; FOWLER, 2019).

2.2 ESTADO DA ARTE DA OTIMIZAÇÃO DE CIRCUITOS QUÂNTICOS

Dado o impacto das portas T para a correção de erros, a redução do número dessas portas é um dos principais foco da literatura, apesar de haver outras métricas, tais como a quantidade de portas lógicas de 2-qubits, o caminho crítico do circuito e o número total de portas lógicas.

Os principais trabalhos de otimização de número de portas fora do conjunto de Clifford foram feitas por (KISSINGER; WETERING, 2020b) e (DUNCAN *et al.*, 2020). Ambos utilizando o cálculo-ZX como *framework* para reescrever o circuito de maneira otimizada. O primeiro algoritmo utiliza *phase gadgets* para fazer cancelamentos de fases não-Clifford já o segundo utiliza noções de complemento e pivotagem de grafos para fazer reescritas no circuito. O último foi escolhido como foco deste trabalho por apresentar uma redução mais eficiente de portas T, mas ambos poderiam ser usados em conjunto.

Uma das outras métricas de otimização é o caminho crítico do circuito. Dada a característica paralela do hardware quântico, a velocidade de execução é dada pelo caminho crítico do circuito, ou seja, o caminho com maior número de iterações de execução. Na literatura, há otimizações de profundidade de portas lógicas sintetizadas a partir do conjunto universal Clifford+T(NG; WANG, 2018), como a porta de Toffoli/CCX (AMY; MASLOV; MOSCA; ROETTELER, 2013; MATTEO; MOSCA, 2016).

Apesar de menos significativa para protocolos de correção de erros, as portas

lógicas de 2-qubit também contribuem para o ruído do sistema e a sua redução é interessante especialmente em casos onde não há correção de erros, como os dispositivos NISQ (Noisy intermediate-scale quantum era) atuais, que são computadores quânticos contemporâneos suscetíveis a erros. As otimizações feitas para portas T não focam em minimizar o número de portas de 2-qubits e, em vários casos, pode aumentar o número delas, como o algoritmo apresentado por esse trabalho. Existem heurísticas que podem ser utilizadas para diminuir o número de portas de 2-qubits utilizando cálculo-ZX e podem ser utilizadas para melhorar os resultados do algoritmo apresentado(STAUDACHER *et al.*, 2023).

Ainda que o custo computacional do computador quântico seja dada pela profundidade do circuito, o uso de recursos para simulações pode ser diminuído com a redução do total de portas lógicas, simplificando a evolução do sistema quântico. Processos de extração de circuitos a partir de cálculo-ZX podem produzir um aumento no número de portas por não produzirem circuitos ótimos, permitindo otimizações de portas de Hadamard. Para essa otimização são utilizadas permutações e cancelamento de portas em algoritmos que não utilizam o cálculo-ZX(STAUDACHER, 2021).

Por fim, a extração de circuitos de Clifford a partir de diagramas-ZX pode ser feita de maneira assintótica utilizando a técnica de eliminação Gaussiana (PATEL; MARKOV; HAYES, 2004). Este algoritmo separa o circuito em uma ordem fixa de camadas e pode ser utilizado para casos onde após a otimização de algoritmos a quantidade de portas T é igual a zero.

2.3 CÁLCULO-ZX

O cálculo-ZX é uma linguagem gráfica de representação de circuitos quânticos, apresentada inicialmente em (COECKE; DUNCAN, 2011). Este capítulo aborda os conceitos fundamentais do cálculo-ZX necessários para o entendimento do algoritmo de otimização. Para um aprofundamento maior no cálculo-ZX e suas aplicações, ver (WETERING, 2020).

Nos diagramas-ZX, os componentes fundamentais são *aranhas*, *pernas* e *fase*. As aranhas são mapas lineares que tem uma quantidade de pernas de entrada e de saída, elas têm duas variações: aranhas X, representadas em vermelho e aranhas Z, em verde. Ambas estão representadas na Figura 3. A aranha Z é definida pelas autobases da matriz Pauli Z, sendo $|0\rangle$ e $|1\rangle$. Da mesma forma, a aranha X é definida pelas autobases da matriz Pauli X, $|-\rangle$ e $|+\rangle$.

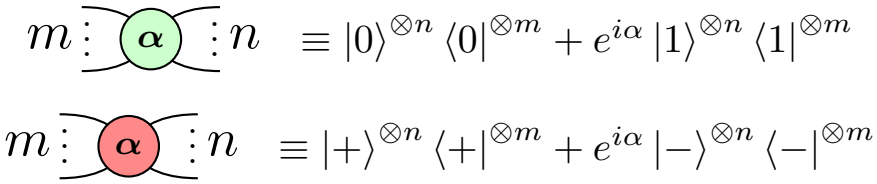


Figura 3 – Definição e notação de aranhas Z e X. M é o número de entradas e N é o número de saídas da aranha.

Dessa forma, as aranhas representam estados, operações unitárias, isometrias e projeções nas bases $|0\rangle, |1\rangle$ e $|+\rangle, |-\rangle$. O tipo gerado é determinado pela quantidade de pernas na aranha, por exemplo, aranhas de estado tem apenas uma perna de saída, aranhas de projeção tem apenas uma perna de entrada, ambas representadas na Figura 4, e as aranhas que representam portas lógicas, as principais desse trabalho, tem exatamente uma entrada e uma saída, como na Figura 5.

Figura 4 – Aranhas de estado e projeção, respectivamente. Aranha X de estado - Para $\alpha = 0$ e $\alpha = \pi$ temos, respectivamente, as bases $|0\rangle, |1\rangle$ não normalizadas. Aranha Z de projeção - Para $\alpha = 0$ ou $\alpha = \pi$ temos uma projeção destrutiva em X para os estados $|+\rangle, |-\rangle$

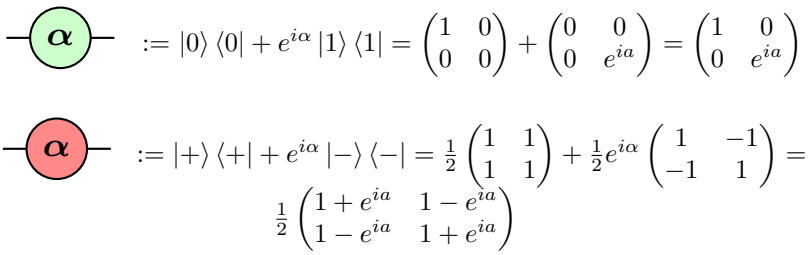


Figura 5 – Representação de portas lógicas por aranhas Z e X com fase α e suas matrizes.

Em casos onde uma aranha Z ou X tenha a fase $\alpha = 0$, podemos descrevê-la como uma aranha sem fase (Figura 6). Para casos onde elas são portas lógicas (apenas uma entrada e uma saída) há uma propriedade extra: elas são equivalentes à matriz de identidade. Essa propriedade significa que é possível remover e colocar livremente aranhas Z e X sem fase no grafo, como na Figura 7.

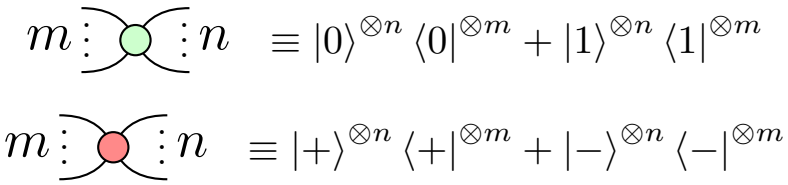


Figura 6 – Aranhas Z e X sem fase e sua notação dirac.

Propriedades como a da matriz identidade são bastante úteis no cálculo-ZX, porque podem ser utilizadas como regra de reescrita. Em um contexto de grafos, regras de reescrita são regras de transformação que especificam como modificar partes de uma estrutura de grafos. Essas regras consistem em padrões que identificam subgrafos específicos a serem substituídos, juntamente com os respectivos modelos de substituição. Isso permite a modificação dinâmica das estruturas de grafos e no caso do cálculo-ZX, permitem simplificar o diagrama obtido.

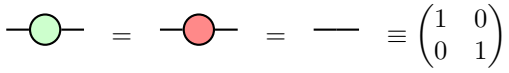


Figura 7 – Regra de reescrita de identidade.

Outra regra de reescrita é a fusão de aranhas, que pode ser feita quando duas aranhas da mesma cor estão adjacentes, como na Figura 8. Todas as pernas, tanto de entrada e saída, das duas aranhas se tornam parte da nova aranha fundida.

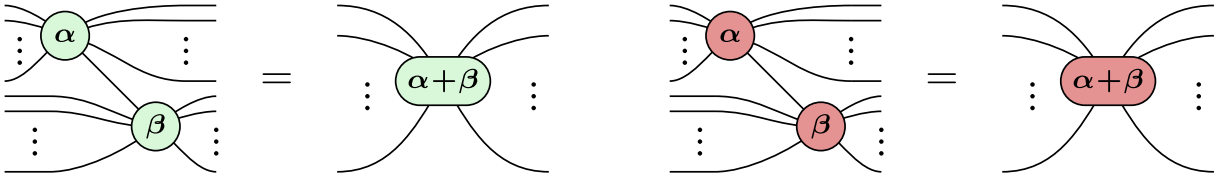


Figura 8 – Regras de reescrita de fusão de aranhas. Fonte: (COMMONS, 2019a) e (COMMONS, 2019b)

As aranhas que representam portas lógicas podem facilmente ser mapeadas para portas de circuitos quânticos reais, quando equivalentes. Na Figura (9) estão os mapeamentos de portas de 1-qubit para uma única aranha. Também podemos unir e separar fases de aranhas da mesma cor para chegar nas portas conhecidas pelo mapeamento. Essas conversões, no entanto, não servem para todas as portas lógicas quânticas, pois não consideram as que operam em mais de uma base e nem portas de mais de um qubit. Para fazer isso são necessárias regras de composição.

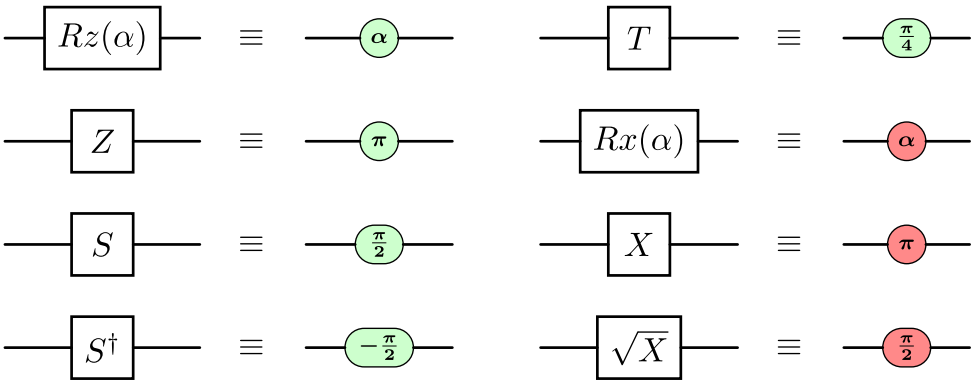


Figura 9 – Mapeamento de portas de 1-qubit para aranhas Z e X.

Diagramas-ZX permitem composições em série e em paralelo. As composições em série correspondem a multiplicação matricial e envolvem a combinação das linhas

da matriz de uma porta lógica com as colunas da outra. Por exemplo, considere duas matrizes 2 por 2, A e B, definidas como:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \tag{1}$$

onde (1) representa as matrizes A e B, respectivamente.

A multiplicação de A por B resulta em:

$$AB = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix} \tag{2}$$

A composição em série permite criar portas lógicas que operam em mais de uma base, como a porta de Hadamard. Por utilizar múltiplas aranhas, essa porta tem uma representação simplificada, vale lembrar que isso não é um elemento novo adicionado ao diagrama-ZX.

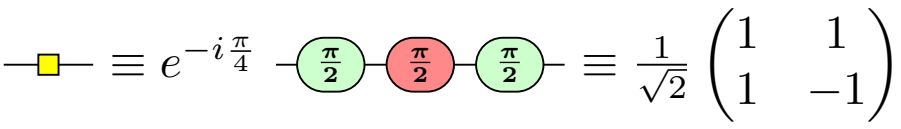


Figura 10 – Definição e composição da porta de Hadamard

Utilizando as matrizes da Figura 5, aplicando as fases do diagrama da Figura 10 e realizando a multiplicação matricial usando $e^{i\frac{\pi}{2}} = i$, temos:

$$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \frac{1}{2} \begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} = \begin{pmatrix} \frac{1}{2}+\frac{i}{2} & \frac{1}{2}-\frac{i}{2} \\ \frac{1}{2}+\frac{i}{2} & -\frac{1}{2}+\frac{i}{2} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} = (\frac{1}{2}+\frac{i}{2}) \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3}$$

O resultado difere da definição de porta de Hadamard. Para que essa definição seja equivalente, precisamos utilizar o escalar $e^{-i\frac{\pi}{4}}$. Escalares são representados no grafo como aranhas sem entradas e saídas e pode ser representado como um número ao lado do diagrama, como a Figura 10. O escalar pode ser ignorado, no cálculo-ZX duas interpretações de um mapa linear são equivalentes se diferirem apenas em um escalar complexo não nulo. Além disso, o escalar pode ser recuperado, visto que a matriz M é proporcional a uma unitária, e, portanto, $MM^\dagger = \lambda I$ para algum $\lambda \in \mathbb{R} > 0$. O escalar correto é então $\frac{1}{\sqrt{\lambda}}$. Esse valor λ pode ser calculado compondo o diagrama-ZX com seu adjunto e simplificando até que ele se reduza à identidade (WETERING, 2020). Multiplicando pelo escalar:

$$e^{-i\frac{\pi}{4}} \left(\tfrac{1}{2}+\tfrac{i}{2}\right) \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = H \tag{4}$$

Para facilitar a notação de conexões entre vértices por Hadamard, a notação da Figura 11 é utilizada. Essa notação é utilizada apenas entre dois vértices de mesma cor, não podendo ser utilizada nas entradas e saídas do circuito.

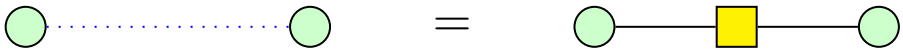


Figura 11 – Notação alternativa para porta de Hadamard usada entre vértices da mesma cor

Uma das propriedades da porta de Hadamard é a conjugação de portas X a partir de portas Z: $HZH = X$. Em diagramas-ZX, a troca de base é uma regra de reescrita e pode ser feita adicionando portas de Hadamard em todas as entradas e saídas da aranha (desde que ela tenha o mesmo número de entradas e saídas), como na Figura 12.

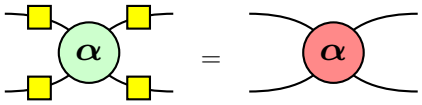


Figura 12 – Aplicação da regra de troca de cor.

Tendo em vista que circuitos quânticos são compostos por portas lógicas unitárias, que são reversíveis, outra regra notável é a reescrita de aplicações de portas consecutivas pela identidade, que não tem efeito no circuito. Um exemplo é a reescrita de portas de Hadamard da Figura 13.

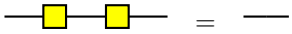


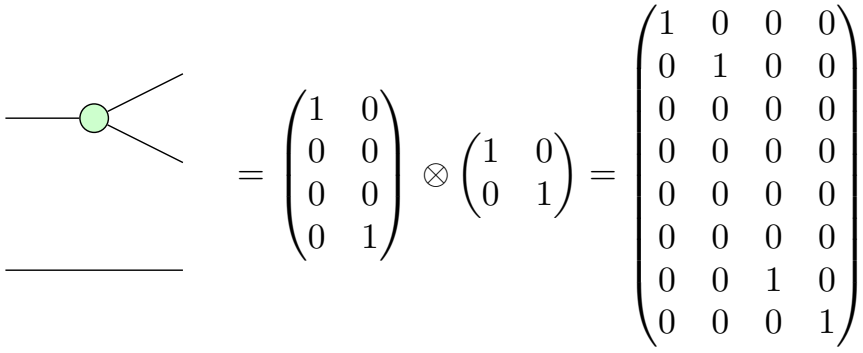
Figura 13 – Regra de reescrita para portas de Hadamard redundantes.

Para portas lógicas de mais de um qubit é possível usar composição paralela. Ela é equivalente a um produto tensorial e envolvem multiplicar todos os elementos da matriz de uma porta lógica pela matriz inteira da outra. Considerando as mesmas matrizes da Equação 1, temos o produto tensorial $A \otimes B$ calculado da seguinte forma:

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{pmatrix} = \begin{pmatrix} a_{11}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} & a_{12}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \\ a_{21}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} & a_{22}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \end{pmatrix} \tag{5}$$

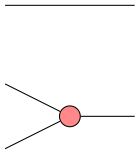
Com a composição paralela podemos construir, por exemplo, portas CNOT sem introduzir elementos novos. Para essa composição, utilizamos uma aranha Z sem fase com uma entrada e duas saídas e uma aranha X com duas saídas e uma entrada. Juntando ambas as aranhas obtemos uma porta CNOT, note que ela pode ser reescrita a realinhando no grafo, porque em cálculo-ZX as deformações no grafo não importam.

Esse processo está ilustrado na Figura 14. Usando esse mesmo processo, é possível obter também a porta CZ. Sua representação está na Figura 15.



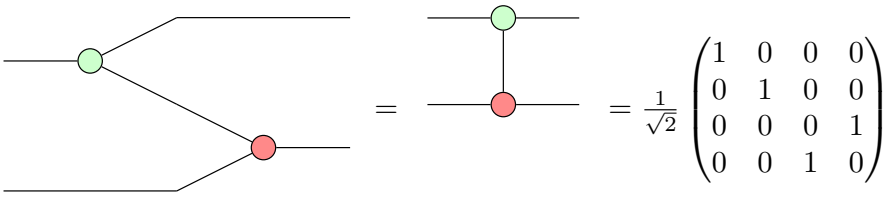
$$= \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(a) Aranha Z sem fase com uma entrada e duas saídas que será utilizada para compor uma porta CNOT.



$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

(b) Aranha X sem fase com duas entradas e uma saída que será utilizada para compor uma porta CNOT.



$$= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

(c) Composição da porta CNOT com escalar de $\sqrt{2}$.

Figura 14 – Composição da porta CNOT no cálculo-ZX.

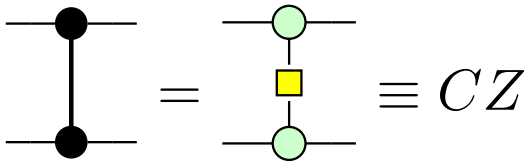


Figura 15 – Representação da porta CZ no cálculo-ZX.

Diagramas-ZX são completos (JEANDEL; PERDRIX; VILMART, 2020) e universais (COECKE; DUNCAN, 2011) e apenas com os dispositivos apresentados já é possível obter computação universal utilizando o conjunto Clifford+T.

Vale ressaltar que nem toda representação é concisa em cálculo-ZX. Portas multi controladas, como a porta Toffoli, por exemplo (Figura 16), são de difícil representação.

Isto acontece porque não há uma forma simples de representar a operação AND em bases computacionais $|x\rangle \otimes |y\rangle \rightarrow |x \cdot y\rangle$. Existem variações do cálculo-ZX que permitem uma melhor representação de portas Toffoli, como o cálculo-ZH (BACKENS; KISSINGER, 2019).

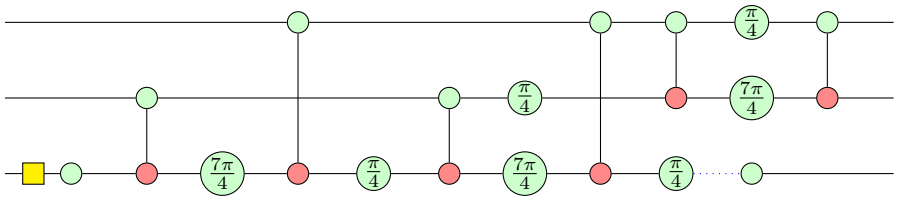


Figura 16 – Diagrama-ZX de uma porta Toffoli por composição.

Por fim, o circuito quântico da Figura 2 pode ser convertido para diagrama-ZX com as regras apresentadas nesta seção. O resultado pode ser observado na Figura 17.

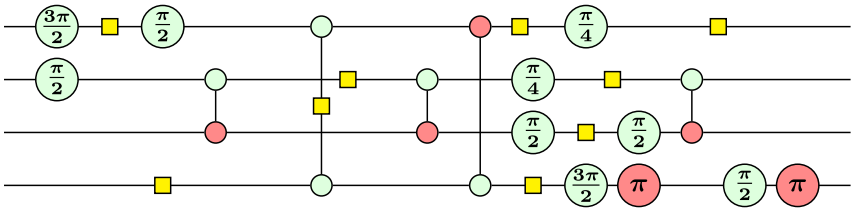


Figura 17 – Circuito da Figura 2 convertido para diagrama-ZX. Fonte: (DUNCAN *et al.*, 2020)

2.4 OTIMIZAÇÃO DE CIRCUITOS QUÂNTICOS

O algoritmo de otimização de portas T abordado neste capítulo é o proposto por (DUNCAN *et al.*, 2020) e consiste em aplicas regras de reescrita baseadas em noções de complemento local e pivotagem de grafos no diagrama-ZX. Para essas operações sejam consideradas válidas para o caso de uso de otimização de circuitos é necessário que o diagrama resultante seja conversível de volta para circuito, essa propriedade se chama extraibilidade.

Para assegurar a extraibilidade, transformaremos o circuito quântico em um grafo de estados, uma notação herdada da computação quântica orientada a medida que tem a propriedade de *focused gflow* (BROWNE *et al.*, 2007), que nesse contexto é utilizado para garantir que existe uma ordem parcial de execução do grafo, implicando na existência de um circuito quântico equivalente para o diagrama-ZX.

2.4.1 General Flow e Focused Gflow

A noção de passagem do tempo em um circuito quântico é dada pela ordem parcial de sua ordenação topológica. Essa propriedade é importante quando lidamos com diagramas-ZX, visto que para o mesmo poder ser extraído em formato de circuito é necessário existir uma ordem parcial de execução.

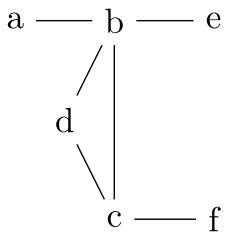
Na Computação Quântica Orientada a Medida, flow é uma propriedade que diz respeito a possibilidade de uma implementação determinística de um circuito quântico. (BACKENS; MILLER-BAKEWELL *et al.*, 2021) apresenta estratégias de extração que utilizam uma especificação dessa propriedade, o *gflow* para garantir a extraibilidade do algoritmo. Para verificar essa propriedade, circuitos são representados em grafos de estados (DANOS; KASHEFI, 2006) e é possível verificar a existência de uma ordem parcial verificando se o grafo tem a propriedade de Generalized Flow (gflow) (BROWNE *et al.*, 2007).

Definição 2.4.1. General Flow: Dado um grafo rotulado $G = (V, E, I, O, \lambda)$, onde V denota um conjunto de vértices, E as arestas, I o conjunto dos vértices que estão na entrada do circuito, O o conjunto de vértices da saída, $\bar{I}$ e $\bar{O}$ o complemento desses conjuntos e λ o eixo de medição (XY, XZ ou YZ), ele tem *generalised flow* caso exista um mapa de correção $g : \bar{O} \to P(\bar{I})$ definindo os conjuntos de correção de vértices, onde $P(\bar{I})$ denota o conjunto de potências de todos os subconjuntos de vértices em $\bar{I}$, dada na ordem na qual as medidas são feitas e uma ordem parcial $\prec$ sobre v de forma que $\forall v \in \bar{O}$:

- Se $w \in g(v)$ e $v \neq w$, então $v \prec w$.
- Se $w \in Odd(g(v))$ e $v \neq w$, então $v \prec w$.
- Se $\lambda(v) = XY$, então $v \notin g(V)$ e $v \in Odd(g(v))$.
- Se $\lambda(v) = XZ$, então $v \in g(V)$ e $v \in Odd(g(v))$.
- Se $\lambda(v) = YZ$, então $v \in g(V)$ e $v \notin Odd(g(v))$.

Onde $g(v)$ é uma função que obtém um conjunto de correção, K é um conjunto de correção, $N_g(u)$ são os vizinhos de um vértice e $Odd(K) = \{u \in V : |N_g(u) \cap K| \equiv 1 \mod 2\}$.

Exemplo 2.4.1. Considere o seguinte grafo:



Onde, $a \in I(entradas)$ e $e, f \in O(saídas)$ e as medidas de cada nó são dadas por: $\lambda(a) = XY$, $\lambda(b) = XY$, $\lambda(c) = XZ$, $\lambda(d) = YZ$. Podemos garantir *gflow* para a seguinte ordem de execução com o seguinte conjunto de correção:

$$a \prec b \prec c \prec d \prec e, f$$

- $g(a) = b$
- $g(b) = c$
- $g(c) = c, d$
- $g(d) = d, e, f$

Obtendo os conjuntos ímpares, temos que:

- $Odd(g(a)) = \{a, c, d, e\}$
- $Odd(g(b)) = \{b, d, f\}$
- $Odd(g(c)) = \{d, f, c\}$
- $Odd(g(d)) = \varnothing$

Para fins didáticos, o conjunto $Odd(g(a))$ é obtido da seguinte forma:

- $Odd(g(a)) = \{a \in V : |\{b\} \cap \{a, c, d, e\}| = 1 \mod 2\}$, condiz, logo $a \in Odd(g(a))$.
- $Odd(g(a)) = \{b \in V : |\varnothing \cap \{a, c, d, e\}| \equiv 1 \mod 2\}$, não condiz, logo $b \notin Odd(g(a))$.
- $Odd(g(a)) = \{c \in V : |\{b, d\} \cap \{a, c, d, e\}| \equiv 1 \mod 2\}$, condiz, logo $c \in Odd(g(a))$.
- $Odd(g(a)) = \{d \in V : |\{b, c\} \cap \{a, c, d, e\}| \equiv 1 \mod 2\}$, condiz, logo $d \in Odd(g(a))$.
- $Odd(g(a)) = \{e \in V : |\{b\} \cap \{a, c, d, e\}| \equiv 1 \mod 2\}$, condiz, logo $e \in Odd(g(a))$.
- $Odd(g(a)) = \{f \in V : |\{c\} \cap \{a, c, d, e\}| \equiv 1 \mod 2\}$, não condiz, logo $f \notin Odd(g(a))$.

No contexto de extração de circuitos quânticos, obter o conjunto de correção é irrelevante. Apenas é necessário garantir a existência de um que satisfaça as condições de *gflow*. Isto é relevante para otimizações, porque para garantir que uma reescrita não remove a extraibilidade do circuito basta provar que será mantida alguma forma de *gflow*, visto que partiremos de um circuito real que já tem uma ordem parcial.

Neste trabalho, é utilizada uma especificação de *gflow*, o *focused gflow*. Isto porque as regras de otimização apresentadas por (DUNCAN *et al.*, 2020) garante a existência dessa especialização após uma reescrita. Em ambos os casos, a estratégia de extração se mantém a mesma.

Definição 2.4.2. Focused Gflow: Dado um grafo rotulado de topologia aberta (G, I, O, λ) com *gflow*, ele será considerado *focused* se $\forall v \in \bar{O}$, $Odd_G = (g(v)) \cap \bar{O} = \{v\}$.

2.4.2 Grafo de Estados

Antes da simplificação do circuito ocorrer, o diagrama-ZX é transformado para um grafo de estados, onde toda aranha está ligada a alguma saída e há fases não nulas, ou seja, permite fases arbitrárias e aranhas internas (aranhas que não estão imediatamente conectadas a uma entrada ou saída). Este estado corresponde aos grafos de estados apresentados por (HEIN, 2005) e é conveniente para desenvolver algoritmos de otimização porque a partir dessa forma é possível aplicar conceitos de ordem de execução de circuito já pesquisados de computação quântica baseada em medida, como no *generalized flow*. Na representação em grafo de estados, cada qubit é representado por um vértice, havendo uma aresta conectando cada par de qubits que interagem, sendo bastante similar ao cálculo-ZX.

Definição 2.4.3. Um diagrama-ZX é um estado de grafos quando:

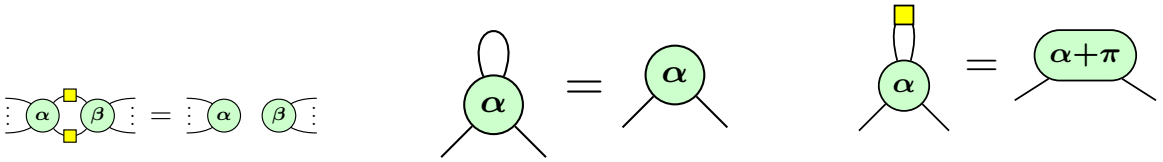
- Todas as aranhas são aranhas Z.
- Aranhas Z só se conectam por portas de Hadamard.
- Não há arestas com portas de Hadamard paralelas ou aranhas com loop em si mesmas.
- Cada entrada ou saída está conectada a uma Z-aranha e cada Z-aranha está conectada, no máximo, a uma entrada ou saída.

Lema 2.4.1. Todo diagrama-ZX pode ser convertido para um grafo de estados.

Demonstração. Para obter o grafo de estados, primeiro aplicamos a regra de troca de cor (Figura 12), em seguida removemos as portas de Hadamard canceláveis pela regra do cancelamento (Figura 13). São removidos os Hadamards em paralelo via regra de reescrita da Figura 18a e autoloops são removidos pelas regras das figuras 18b e 18c.

Após a aplicação dessas regras, as três primeiras condições estão cobertas, faltando apenas a quarta. Que pode ser resolvida aplicando o reverso das reduções 7 e 13, para adicionar aranhas sem fase e Hadamards até que as entradas e saída estejam separadas. Com isso, são satisfeitas as quatro regras apresentadas. ■

Os grafos de estados tem uma propriedade crucial para a etapa de extração de circuitos: todo grafo obtido pela conversão anterior admite um *Focused Gflow*. No entanto, existem operações que podem ser feitas no grafo que não preservam esta propriedade e podem fazer com que o algoritmo de extração nunca pare.



(a) Regra de reescrita para remoção de Hadamards em paralelo.

(b) Regra de reescrita para remoção de auto-loops.

(c) Regra de reescrita para remoção de auto-loops com Hadamards.

Figura 18 – Conjunto de regras de reescrita para otimizações de gráficos.

A transformação para um grafo de estados pode ser observada na Figura 19, que é feito do diagrama-ZX da Figura 17.

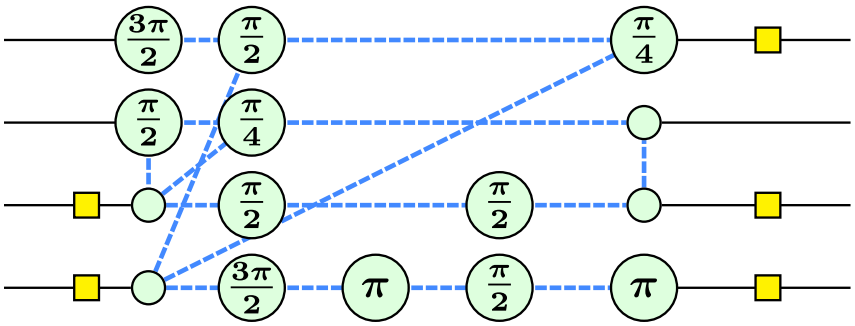


Figura 19 – Circuito transformado em grafo de estados conforme a Figura 18 e com fases somadas conforme a regra da Figura 8. Fonte: (DUNCAN *et al.*, 2020)

2.4.3 Complementação Local de Grafos e Pivotagem

Esta é uma transformação de grafos que altera as conexões entre os vizinhos de um nó específico (ou vértice) no grafo. Ao aplicar a complementação local a um nó, todas as arestas que conectam os vizinhos desse nó são invertidas; isto é, onde havia uma aresta, ela é removida, e onde não havia, uma nova aresta é criada. Esta operação é uma das chaves para simplificar os diagramas ZX, pois pode reduzir a complexidade do grafo ao eliminar conexões redundantes ou desnecessárias.

O pivoteamento é outra transformação de grafos que envolve dois nós conectados (um "pivô") e altera as conexões entre seus vizinhos. É uma operação mais complexa que a complementação local e pode ser vista como uma sequência de complementações locais.

2.4.4 Simplificação de diagramas ZX

A estratégia de otimização utiliza as noções de pivotagem e complemento de grafos previamente estabelecidas para eliminar a maior quantidade possível de aranhas interiores para preservar o *focused gflow* do grafo. Essas regras são extraídas de (DUNCAN *et al.*, 2020).

A Regra 1 de otimização (Figura 20a) prevê que caso exista um elemento no diagrama uma aranha (indicada por um $*$) representando uma variação de fase de $\pm\frac{\pi}{2}$, e esse elemento estiver posicionado internamente, sem ligações diretas com entradas ou saídas, mas conectado somente a outras aranhas, é possível excluí-lo do diagrama. Isso é feito complementando a conectividade na área ao redor dele e somando $\mp\frac{\pi}{2}$.

Já a Regra 2 (Figura 20b) prevê que caso existam aranhas com uma fase de 0 ou π internas ligadas a outras aranhas (indicadas por um $*$), é possível removê-las complementando suas vizinhanças conforme descrito pela regra de pivô e ajustando as fases de acordo com seu tipo de vizinhança (exclusiva da primeira aranha marcada, exclusiva da segunda ou comum de ambas).

Por fim, a Regra 3 (Figura 20c) prevê que caso tenhamos uma aranha de Pauli interior ligada a uma aranha de borda, podemos transformá-la numa aranha sem fase, para permitir que a Regra 2 (20b) seja aplicada.

A rotina de otimização é executada rodando essas três etapas em loop, parando quando não houver mais nenhuma regra a ser aplicada.

1. Aplicar a Regra 1 (Figura 20a) para remover uma aranha Proper Clifford (cuja fase é um múltiplo impar de $\frac{\pi}{2}$, logo não é Pauli).
2. Aplicar a Regra 2 (Figura 20b) para remover pares adjacentes de aranhas Pauli (aranhas cuja fase é múltipla de π) interiores adjacentes.
3. Aplicar a Regra 3 (Figura 20c) em uma aranha Pauli de borda e imediatamente aplicar Regra 2.

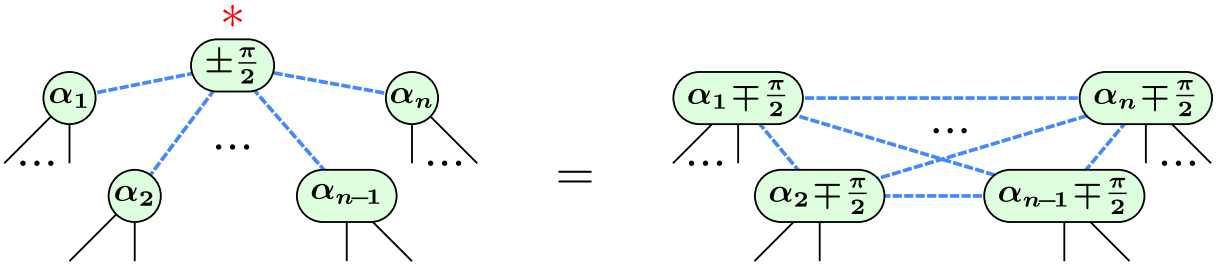
A aplicação das regras de otimização pode ser observadas nas figuras 21a, 21b, 21c, que continuam o processo de otimização a partir da Figura 19.

2.5 EXTRAÇÃO DE CIRCUITOS

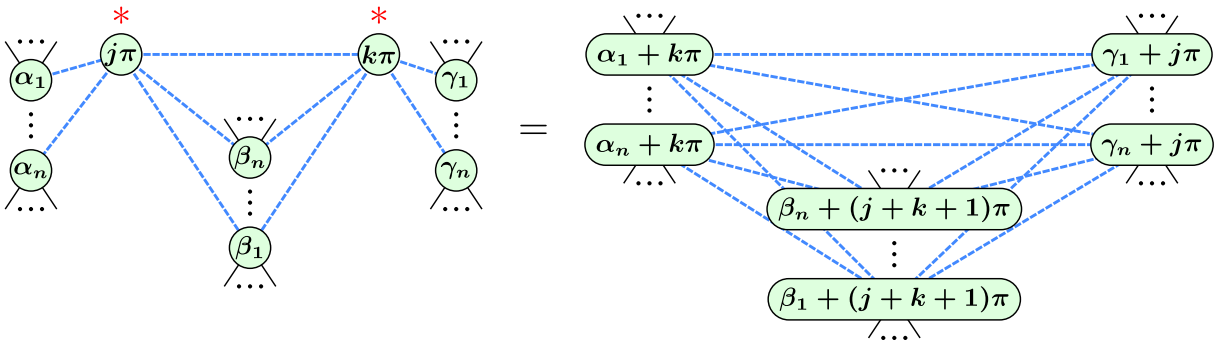
Para extrair um circuito quântico válido a partir do grafo de estados é utilizado uma variante do algoritmo proposto em (BACKENS; MILLER-BAKEWELL *et al.*, 2021) e adaptado em (DUNCAN *et al.*, 2020). Esse algoritmo de extração faz a síntese do circuito da direita para esquerda, onde estratégia é a avançar uma fronteira pelo diagrama, que separa a parte extraída da não-extraída. A garantida de parada na execução do algoritmo é dada pela propriedade de *focused gflow*.

O algoritmo está descrito no pseudocódigo 1 e os passos de extração podem ser observados nas figuras 22, 23, 24, 25, que continuam o processo de otimização a partir da Figura 21c. As principais etapas e execução são:

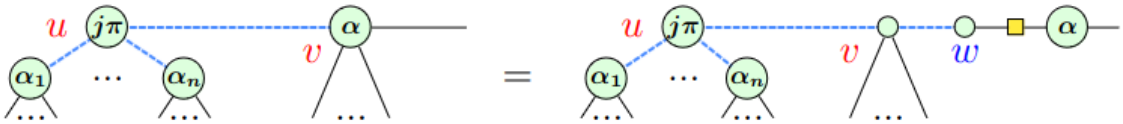
1. Criação de uma fronteira na saída do circuito (Alg. 1, linha 4). A quantidade de elementos na mesma é igual a quantidade de qubits no circuito. Tudo que estiver depois da fronteira representa um circuito quântico extraído.



(a) Regra de reescrita baseada em complemento de grafos.



(b) Regra de reescrita baseada em pivotagem.



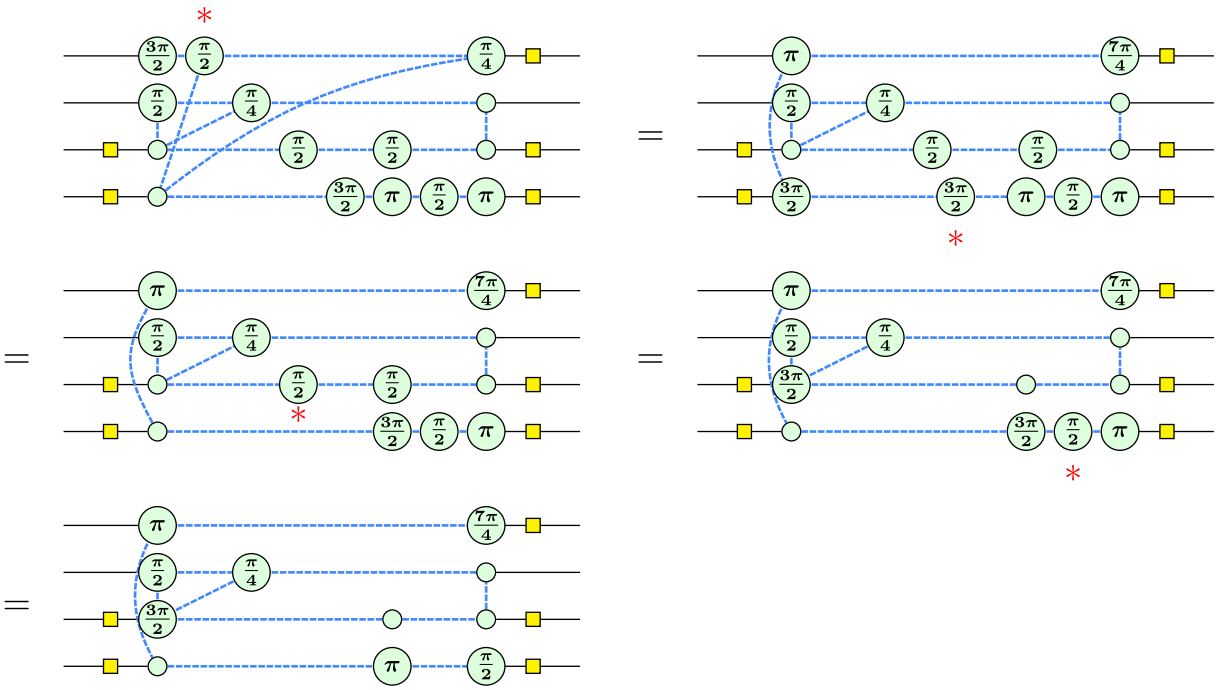
(c) Regra de reescrita para lidar com aranhas de borda com fase.

Figura 20 – Conjunto de regras de reescrita. Fonte: (DUNCAN *et al.*, 2020).

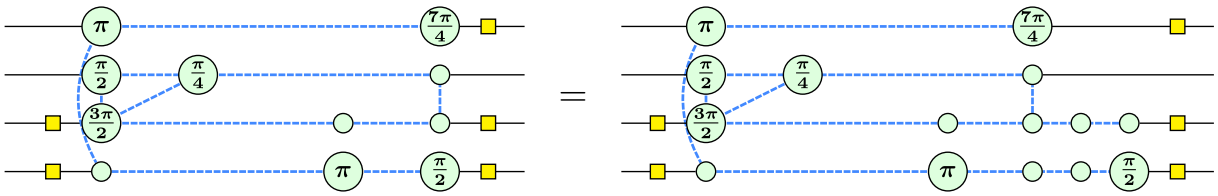
2. Mover a fronteira em direção a entrada do circuito. Toda aranha que fizer contato fará parte da fronteira. Este processo é repetido até alcançar a entrada.

    - Caso uma aranha pertencente a fronteira tenha apenas uma conexão com o grafo não extraído, ela é extraída e movida para a direita da fronteira (Alg. 1, linhas 6-12). A conversão para circuito é possível ser feita pelas conversões da Figura 9 e por composições em série, como na Figura 10.
    - Caso dois vértices da fronteira estejam conectados entre si, a conexão é desfeita por uma porta CZ (Alg. 1, linhas 13-16). O elemento CZ é considerado extraído porque pode ser convertido para circuito como na Figura 15.
    - Caso existam conexões de um vértice da fronteira com outros que não fazem parte da mesma, essas conexões são desfeitas utilizando portas CNOT em etapas utilizando redução gaussiana. (Alg. 1). O elemento CNOT é considerado extraído porque pode ser convertido para circuito como na Figura 14c.

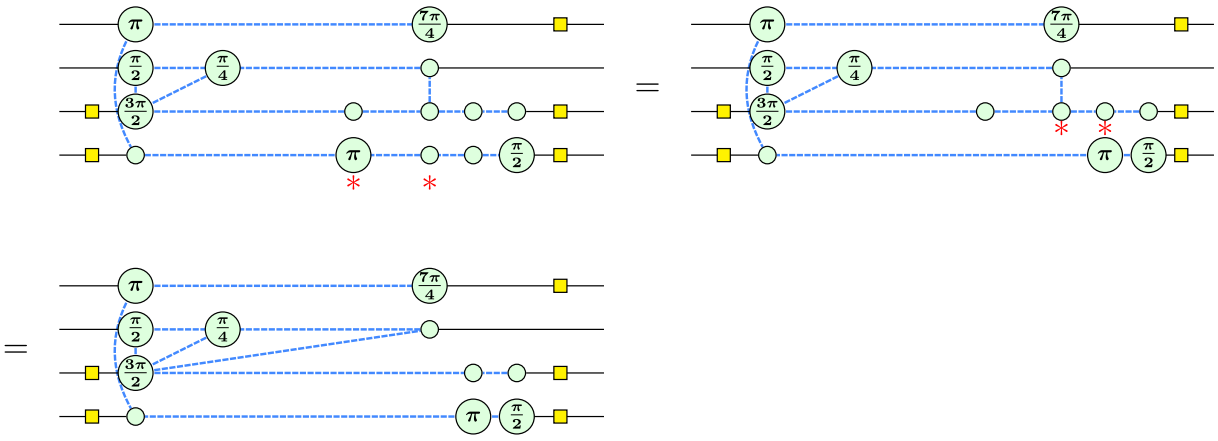
3. Uma vez encerrado o processo de mover a fronteira, os vértices finais presentes



(a) Aplicação da regra de complemento local nos vértices marcados para remoção com asterisco vermelho. Conexões entre vértices vizinhos sãos invertidas (complemento).



(b) Aplicação da regra da borda.



(c) Aplicação da regra de pivotagem nos vértices marcados para remoção com asterisco vermelho.

Figura 21 – Exemplo de aplicação das regras de otimização. Fonte: (DUNCAN *et al.*, 2020).

nela estarão permutados com as entradas. Para garantir equivalência com o circuito original, portas SWAP são utilizadas para fazer as permutações (Alg. 1, linhas 24–26. Portas SWAP tem a mesma representação de circuitos quânticos em cálculo-ZX e estão presentes na Figura 1.

O algoritmo apresentado utiliza uma propriedade da matriz de adjacência dos vértices da fronteira com os seus vizinhos não extraídos apresentada por (DUNCAN *et al.*, 2020), onde a aplicação de uma porta CNOT a duas saídas corresponde a uma operação de linha na matriz de adjacência, que adiciona a linha correspondente ao alvo à linha correspondente ao controle.

Se um *focused gflow* existe no grafo, a parada desse algoritmo é garantida. Isto porque caso essa propriedade não fosse garantida, a etapa de separação por CNOTs ficaria em loop. O focused gflow garante que haverá sempre uma linha com uma única conexão (valor '1') na forma reduzida de escada da matriz durante a eliminação. Dessa forma, podemos apenas fazer rodadas de eliminação de Gauss-Jordan e eventualmente removeremos todas as conexões, por isso a explicação simplificada dessa etapa acima.

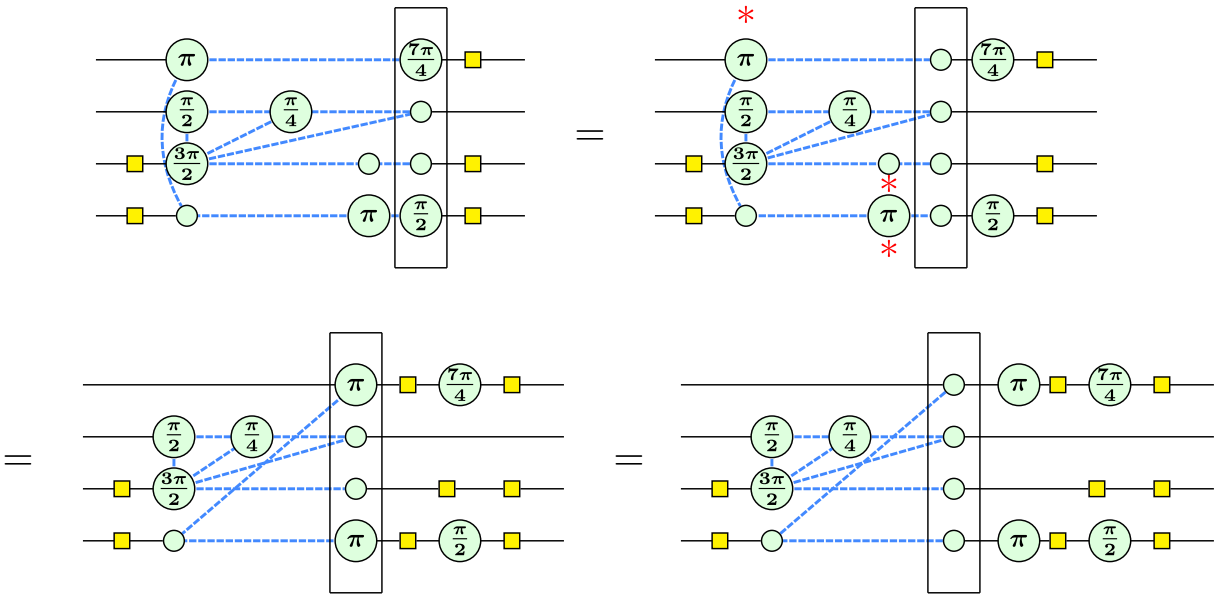


Figura 22 – Inicialização e avanço da fronteira. Fonte: (DUNCAN *et al.*, 2020)

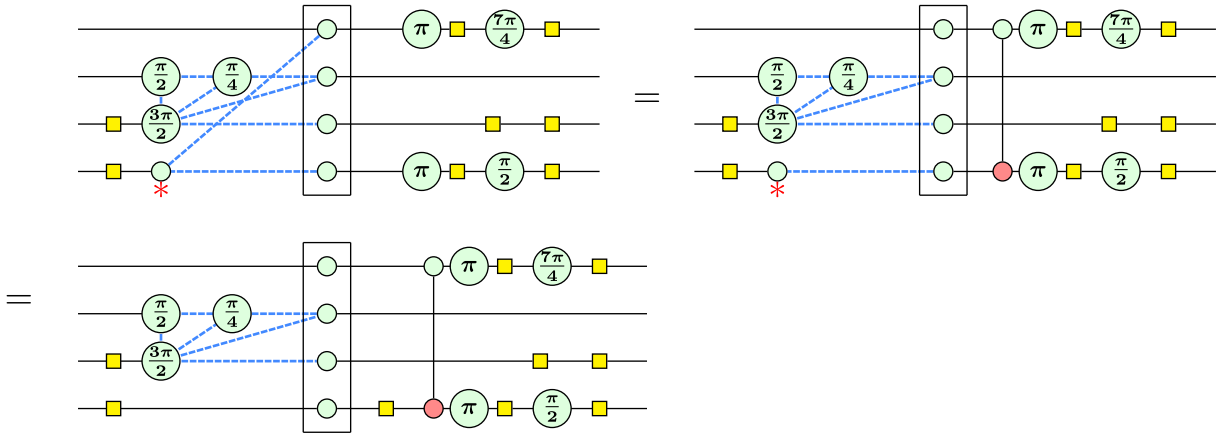


Figura 23 – Diferente do passo anterior, o vértice marcado está com mais de uma conexão com a fronteira e precisa ser decomposto por uma CNOT ou CZ. Fonte: (DUNCAN *et al.*, 2020)

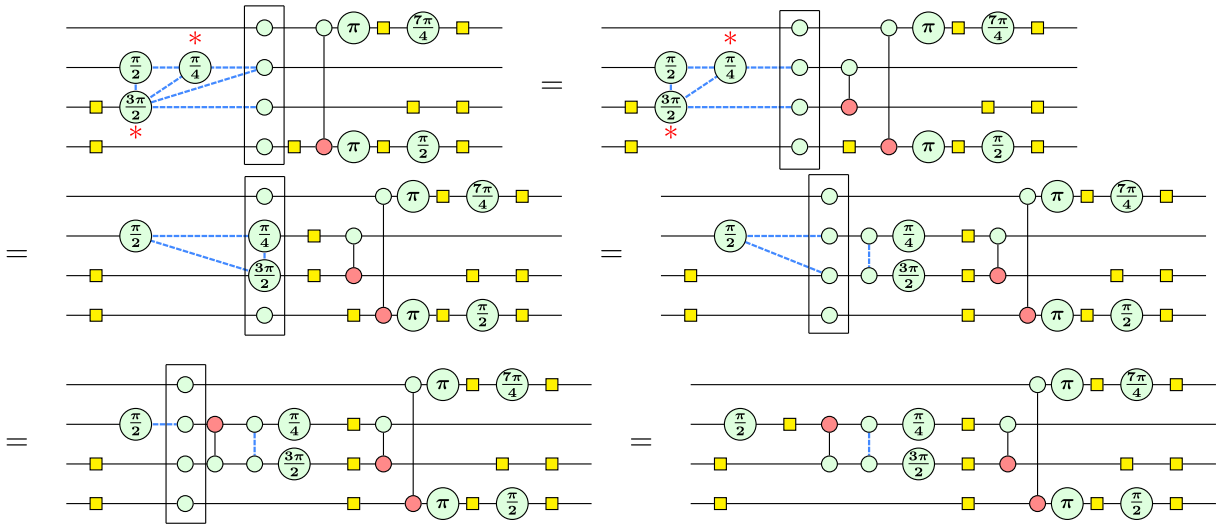


Figura 24 – Passos de extração são executados até a fronteira chegar na entrada. Fonte: (DUNCAN *et al.*, 2020)

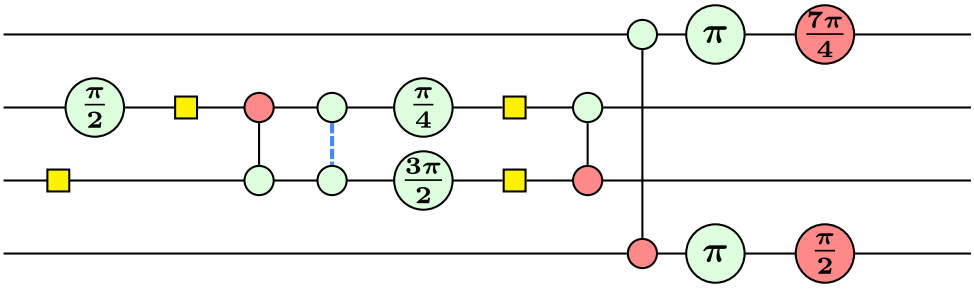


Figura 25 – Resultado passa por simplificações simples (cancelamento de portas e troca de cor). Fonte: (DUNCAN *et al.*, 2020)

Algorithm 1 Algoritmo de extração. Fonte: (DUNCAN *et al.*, 2020)

procedure EXTRAIR(D) ▷ a entrada é o diagrama tipo-grafo D
  Inicializar circuito vazio C
  $G, I, O \leftarrow$ Grafo(D) ▷ obtém o grafo subjacente de D
  $F \leftarrow O$ ▷ inicializa a fronteira
  for $v \in F$ **do**
    if v conectado à saída por Hadamard **then**
      $C \leftarrow$ Hadamard(Qubit(v))
      Remover Hadamard de v para saída
    end if
    if v possui fase não nula **then**
      $C \leftarrow$ Porta de Fase(Fase(v), Qubit(v))
    end if
    for aresta entre v e w em F **do**
      $C \leftarrow$ CZ(Qubit(v), Qubit(w))
      Remover aresta entre v e w
    end for
  end for
  while existir $v \in D \setminus F$ **do** ▷ ainda há vértices a serem processados
    $D, F, C \leftarrow$ AtualizarFronteira(D, F, C)
    for $v \in F$ **do** ▷ os únicos vértices ainda em D estão em F
      if v conectado à entrada via Hadamard **then**
        $C \leftarrow$ Hadamard(Qubit(v))
      end if
      $Perm \leftarrow$ Permutação dos Qubits de F para qubits das entradas
      for swap($q1, q2$) em PermutaçãoComoTrocas($Perm$) **do**
        $C \leftarrow$ Swap($q1, q2$)
      end for
    end for
  end while
end procedure
procedure ATUALIZARFRONTEIRA(D, F, C)
  $N \leftarrow$ Vizinhos(F)
  $M \leftarrow$ Biadjacência(F, N)
  $M_0 \leftarrow$ ReduzirGauss(M)
  Inicializar ws ▷ inicializa conjunto vazio ws
  for linha r em M_0 **do**
    if soma(r) $== 1$ **then** ▷ existe um único 1 na linha r
      Definir w para vértice correspondente à coluna não nula de r
      Adicionar w a ws ▷ w será parte da nova fronteira
    end if
    $M \leftarrow$ Biadjacência(F, ws) ▷ matriz de biadjacência menor
    for $(r1, r2)$ em Operações de Linha Gaussiana(M) **do**
      $C \leftarrow$ CNOT(Qubit($r1$), Qubit($r2$))
      Atualizar D baseado na operação de linha
    end for
    for $w \in ws$ **do** ▷ todos w agora têm um vizinho único em F
      $v \leftarrow$ Vizinho único de w em F
      $C \leftarrow$ Hadamard(Qubit(v))
      $C \leftarrow$ Porta de Fase(Fase(w), Qubit(v))
      Remover fase de w
      Remover v de F
      Adicionar w a F
    end for
    for aresta entre $w1$ e $w2$ de ws **do**
      $C \leftarrow$ CZ(Qubit($w1$), Qubit($w2$))
      Remover aresta entre $w1$ e $w2$
    end for
  end for
  return D, F, C
end procedure

2.6 QUANTUM ASSEMBLY LANGUAGE

Dentre as representações intermediárias de circuitos quânticos, o OpenQASM (Open Quantum Assembly, ou apenas QASM)(CROSS *et al.*, 2017) se destaca pela sua interoperabilidade com diversas bibliotecas quânticas e kits de desenvolvimento de software, como o Qiskit. O QASM serve como uma representação de baixo nível unificada para o IBMQ e outros dispositivos quânticos. No escopo deste trabalho, focamos no OpenQASM 2.0, no entanto, existe versão mais nova, o OpenQASM 3.0, que foca em execução em tempo real, mas não é suportada por grande parte das bibliotecas ainda.

O OpenQASM 2.0 é uma representação intermediária de baixo nível, executada sequencialmente sem *loops*, ramificações ou saltos, o que a torna muito conveniente para manipulação de circuitos. Para obter universalidade, são usadas portas U, que são portas arbitrárias de 1-qubits que utilizam três ângulos e portas CNOT de 2-qubits, Figura 1, todas as outras portas são obtidas a partir dessas e podem ser declaradas separadamente pelo usuário.

$$U(\theta,\phi,\lambda) = \begin{pmatrix} e^{-i(\phi+\lambda)/2}cos(\theta/2) & -e^{-i(\phi-\lambda)/2}sin(\theta/2) \\ e^{i(\phi-\lambda)/2}sin(\theta/2) & e^{i(\phi+\lambda)/2}cos(\theta/2) \end{pmatrix} \tag{6}$$

A gramática QASM permite declarações de qubits e bits, declarações de novas portas lógicas (utilizando as portas U e CNOT ou outras declarações feitas previamente), operações de barreira, que indicam que um ponto de execução que não pode ser otimizado, declarações de portas opacas - cuja implementação não é especificada por outras portas como nas declarações normais, operações de medida e execução condicional e, por fim, operações de reiniciar um qubit. A linguagem fornece um conjunto de portas comumente utilizadas no arquivo cabeçalho padrão *qelib1.inc*, compostas em termos de portas U e CNOT.

2.7 PLATAFORMA DE DESENVOLVIMENTO QUÂNTICO KET

Ket Quantum Programming (DA ROSA; DE SANTIAGO, 2021) é uma plataforma de código embarcada em Python desenvolvida para permitir programação híbrida clássica-quântica, ou seja, interação de código quântico com lógica clássica. A arquitetura da plataforma é separada em três componentes: a Libket, biblioteca de tempo de execução da linguagem (escrita em Rust); o KBW, simulador quântico sem ruído que utiliza a representação *Bitwise* para que o tempo de execução seja ditado pela quantidade de superposição do circuito, invés da quantidade de qubits (ROSA; TAKETANI, 2020) e o Ket, API python que permite o usuário a criar processos quânticos e simulá-los.

O controle clássico se dá por operações de medida, cujos resultados são retornados como objetos manipuláveis pela linguagem Python. Diferente do OpenQASM,

a plataforma não está presa a natureza de computadores quânticos reais devido ao simulador, podendo adicionar instruções que violam as regras da mecânica quântica, como o *dump*, que permite visualizar o estado quântico. Além de medida e *dump*, a plataforma tem suporte para cálculo de valor esperado e *sample*, que mede um qubit. A *libket* suporta portas de Pauli, portas de rotação, fase e Hadamard, apesar de todas serem de 1-qubit, é possível adicionar qubits de controle a elas.

3 DESENVOLVIMENTO DO SOFTWARE PROPOSTO

Neste capítulo é abordada a implementação prática do software projetado para otimização de circuitos quânticos utilizando a plataforma Ket. Primeiramente, o capítulo explora a abordagem na plataforma, detalhando sua arquitetura e localizando onde foram necessárias fazer as alterações para ser possível o suporte da representação ZX. Em seguida, discute-se o uso de uma linguagem intermediária para fazer a comunicação entre a plataforma Ket e a biblioteca otimizadora escolhida, para não ser necessário migrar toda a infraestrutura já existente da plataforma para ZX. Neste contexto, é apresentada a implementação de um transpilador para fazer a tradução do código QASM para a *libket* e vice-versa. Por fim, o capítulo aborda a representação de diagramas ZX na plataforma Ket, a separação do circuito para otimização, e a criação e configuração de processos quânticos otimizados, destacando os desafios e soluções para manter a eficiência durante a execução dos processos quânticos.

3.1 ABORDAGEM NA LINGUAGEM KET

A arquitetura da plataforma Ket, atualmente, é separada em *libket*, biblioteca Rust responsável por fazer operações lógicas em processos quânticos e que coordena o simulador *kbw*, também em Rust, responsável pela execução dos processos e *Ket*, biblioteca Python que interpreta o código quântico alto nível escrito pelo desenvolvedor e os repassa para a *libket*. Nesta arquitetura a comunicação entre bibliotecas ocorre via *bindings* C, permitindo comunicação entre os binários em Rust e a API em Python.

A biblioteca *QuiZX* [1] foi escolhida por apresentar implementações em Rust dos algoritmos apresentados, permitindo uma integração mais fácil com a *libket*, escrita em Rust, e possibilitando uma execução mais rápida que as implementações presentes na outra biblioteca de cálculo-ZX disponível, o *PyZX* (KISSINGER; WETERING, 2020a), escrito em Python.

Para possibilitar que a biblioteca obtenha os circuitos quânticos, transformá-los em diagramas-ZX e fazer as operações necessárias de otimização e extração, é preciso fornecer o circuito mediante uma representação intermediária. Para isso, foi escolhida a linguagem Open Quantum Assembly (Open QASM), feita para fazer a descrição de código quântico executável em computadores quânticos da IBM (CROSS *et al.*, 2017).

Levando isso em conta, o fluxograma da Figura 26 apresenta a proposta de alteração da arquitetura da Plataforma Ket para suportar a otimização de circuito. Nela, é apresentada o processamento de um código quântico na linguagem Ket até a sua otimização. Na Figura, é possível observar a interação entre o front-end da linguagem e a biblioteca *libket* e, em seguida, sua interação com o otimizador por meio

[1] Biblioteca QuiZX está disponível em https://github.com/Quantomatic/quizx

da representação intermediária QASM.

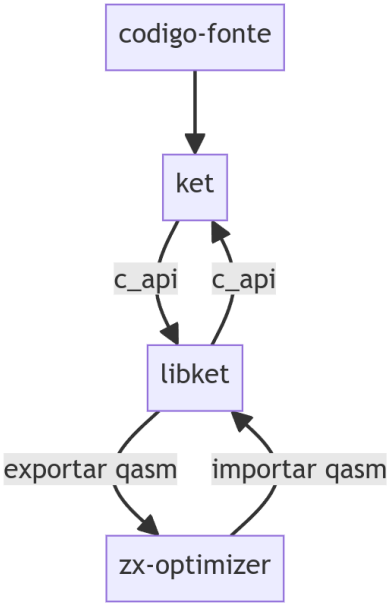


Figura 26 – Fluxograma proposto para o otimizador.

3.2 INTEROPERABILIDADE COM QASM NA LINGUAGEM KET

Como a plataforma Ket não tem suporte nativo para QASM, foi necessário fazer a implementação tanto de um importador e de um exportador. Esta implementação foi feita na *libket* em um módulo dedicado a fazer operações em OpenQASM.

3.2.1 Transpilador de código QASM

O transpilador utiliza a fila de instruções do processo quântico, que contém as portas a serem aplicadas, operações de que devem ser aplicadas e alocações e liberações de recursos quânticos. Essas instruções são traduzidas para o código QASM equivalente, no entanto, há algumas operações que são exclusivas do Ket, como *dump* (que dá informações de *debug* sobre o estado do qubit) e *sample* (que mede um qubit n vezes e apresenta a distribuição dos resultados).

Para estas instruções, são utilizadas instruções opacas, que permite a adição de operações não especificadas pelo QASM. Essa flexibilidade permite então que essas operações sejam preservadas e permitem manter as operações originais declaradas no Ket.

3.2.2 Suporte para código QASM na linguagem Ket

Para adicionar suporte a QASM, foi necessário implementar um importador de código quântico. Este importador necessita ler código QASM e traduzi-lo para instruções na *libket*, para isso, foi utilizada a biblioteca em Rust *openqasm*, sendo um parser

para esse tipo de código. Quando o parser detecta alguma instrução QASM, uma chamada equivalente é feita na *libket* e um processo quântico é construído a partir das instruções.

Inicialmente, é feito um mapeamento das portas suportadas tanto pelo QASM e pela *libket*, no entanto, nem todas são, tornando necessário fazer decomposições. A porta unitária genérica de 1-qubit do QASM não é suportada pela plataforma Ket, no entanto, é possível obter essa porta a partir de decomposição com um conjunto de portas já implementadas na plataforma. Para obter essa decomposição são utilizadas as portas R_Z e R_Y e aplica se a equivalência descrita nas equações abaixo:

$$R_z(\lambda) = \begin{pmatrix} e^{-i(\lambda)/2}cos(\theta/2) & 0 \\ 0 & e^{i(\lambda)/2}cos(\theta/2) \end{pmatrix} \tag{7}$$

$$R_y(\theta) = \begin{pmatrix} cos(\theta/2) & sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{pmatrix} \tag{8}$$

$$U(\theta,\phi,\lambda) := R_z(\phi)R_y(\theta)R_z(\lambda) = \begin{pmatrix} e^{-i(\phi+\lambda)/2}cos(\theta/2) & -e^{-i(\phi-\lambda)/2}sin(\theta/2) \\ e^{i(\phi-\lambda)/2}sin(\theta/2) & e^{i(\phi+\lambda)/2}cos(\theta/2) \end{pmatrix} \tag{9}$$

Outras portas suportadas, como Hadamard, X, Y, etc. são obtidas verificando os parâmetros de U, caso sejam equivalentes a mesma porta ela é utilizada invés da decomposição (CROSS *et al.*, 2017).

3.3 REPRESENTAÇÃO DE DIAGRAMAS-ZX NA PLATAFORMA KET

A biblioteca *QuiZX* permite a transformação de código QASM para diagrama-ZX. Após essa conversão ser feita, o diagrama é transformado num grafo de estados pelo processo apresentado na seção 2.4.2 sendo então simplificado com o algoritmo apresentado na seção 2.4.4, onde as aranhas de Clifford são removidas do circuito. Esse fluxo está descrito na Figura 27, note que todo circuito-quântico está contigo no conjunto de diagramas ZX, mas o contrário não é verdade.

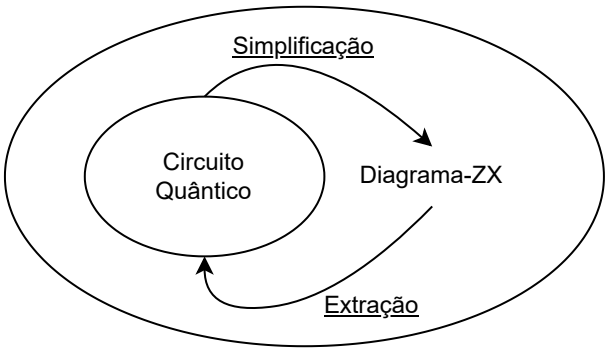


Figura 27 – Fluxo de otimização e extração de diagramas-ZX

Após a simplificação, o algoritmo de extração de circuitos genéricos, seção 2.5, é executado, transformando o diagrama de volta para um circuito quântico, que pode ser convertido de volta para QASM para poder ser importado de volta para a *libket*. A função que implementa esta funcionalidade na *libket* está descrita no Código 1.

Código 1 – Função de otimização utilizando chamadas do QuiZX

```
fn zx_optimize(qasm: &str) -> String {
    let c = Circuit::from_qasm(qasm).unwrap();
    let mut g: Graph = c.to_graph();
    quizx::simplify::clifford_simp(&mut g);
    let c_optimized = g.to_circuit().unwrap();
    c_optimized.to_qasm()
}
```

3.4 SEPARAÇÃO DO CIRCUITO

Algumas precauções precisam ser tomadas para preservar a semântica original do circuito, como preservar as operações de medida, dump e amostragem. Essas operações não são nativamente suportadas pelo otimizador sendo ignoradas durante o processo, ou seja, são perdidas pelo otimizador. É importante preservar a ordem das operações, pois o ponto de execução de uma medida, por exemplo, altera completamente o resultado da computação se estiver em um lugar diferente. Operações como dump, que fornecem informações sobre o estado atual do qubit, precisam ser mantidas onde foram declaradas para que suas informações continuem sendo relevantes.

Para mitigar isso, o código QASM é dividido em seções e otimizado em partes, onde cada uma dessas seções são separadas quando qualquer operação de medição é encontrada. Esses blocos são otimizados e suas operações de medição são restauradas nos seus qubits alvo. Isto pode ser observado no Código 2.

Código 2 – Código a ser otimizado

```
        OPENQASM 2.0;
        include "qelib1.inc";
        opaque dump q1;
        qreg q[2];
        creg c[2];
        h q[0];
        dump q[0];
        x q[0];
        x q[1];
        measure q[0] -> c[0];
        measure q[1] -> c[1];
```

Código 3 – Primeira seção a ser otimizada

```
1  OPENQASM 2.0;
2  include "qelib1.inc";
3  opaque dump q1;
4  qreg q[2];
5  creg c[2];
6  h q[0];
7  dump q[0] -> c[0];
```

Código 4 – Segunda seção a ser otimizada

```
1  OPENQASM 2.0;
2  include "qelib1.inc";
3  qreg q[2];
4  creg c[2];
5  x q[0];
6  x q[1];
7  measure q[0] -> c[0];
8  measure q[1] -> c[1];
```

Vale acrescentar que as declarações de qubits e bits são preservadas para manter o código sintaticamente correto, pois os registradores, tanto clássicos quanto quânticos, precisam estar alocados para que uma declaração de porta seja válida.

Essa separação em seções, apesar de permitir bastante flexibilidade, funciona em detrimento do processo de otimização, porque diminui a quantidade de informação que pode ser utilizada pelo otimizador. Instruções de medida precisam ser postergadas para o final da execução do algoritmo para não precisarem ser separadas em blocos, já instruções como o *dump* são interessantes de serem mantidas na posição desejada pelo desenvolvedor, porque elas têm característica de depuração. A fim de amenizar isso e forçar o usuário a escrever seu circuito em um único bloco, as instruções sensíveis à posição no circuito foram configuradas no simulador *kbw* da seguinte forma:

- Qubits são sempre considerados inválidos após serem medidos. Dessa forma, o usuário da plataforma não fará múltiplos blocos com o mesmo qubit.
- A execução sempre termina após operações de amostra e o cálculo de valores esperados. Dessa forma, haverá um único bloco a ser otimizado.

3.5 CRIAÇÃO DE PROCESSOS QUÂNTICOS OTIMIZADOS

Para alcançar o processo quântico otimizado, é necessário fazer uma série de alterações no processo quântico original, que não pode ser substituído por já ter opções de execução quântica carregadas. Antes de carregar os códigos otimizados, é preciso limpar as informações sobre as portas lógicas não otimizadas, incluindo a fila de instruções e as pilhas auxiliares para portas controladas por múltiplos controles. Após isso, os qubits do processo são desalocados e quaisquer operações de medição são removidas para serem reinseridas posteriormente. A função apresentada no código 5 é responsável por fazer a limpeza das variáveis mencionadas.

Após a otimização desses blocos, cada uma das seções QASM otimizadas é importada para reconstruir um processo quântico otimizado. No entanto, deve-se ter cuidado ao importar as seções, pois as declarações de qubits e bits serão repetidas

Código 5 – Código responsável por reiniciar o estado do processo quântico

```
fn prepare_process(process: &mut Process) {
    process.instructions.clear();
    process.ctrl_stack.clear();
    process.adj_stack.clear();
    process.qubits.clear();
    process.qubit_allocated = 0;
    process.dumps.clear();
    process.measurements.clear();
}
```

em todas as seções. Para contornar esse problema, as alocações são feitas apenas na primeira seção fornecida, porque ela já contém todos os recursos usados na execução.

Uma vez que o processo seja reconstruído, ele está pronto para ser executado pelo simulador quântico e terá uma execução equivalente ao processo original.

3.6 CONFIGURAÇÃO DO PROCESSO QUÂNTICO

Com a otimização implementada, é necessário definir um ponto onde ela será executada. Para isso, a flag de otimização *optimize* foi adicionada à estrutura de dados de configuração do processo, um exemplo de configuração de processo quântico está na linha 1 do Código 6. Quando ativada, ela otimiza o circuito antes da execução. Essa otimização é realizada na etapa de preparação do processo para execução, onde essa flag é verificada.

Vale acrescentar que a plataforma Ket oferece dois tipos de execuções para o simulador: *live* (em tempo real) e *batch* (em lotes). O primeiro não permite otimizações, pois a porta lógica será simulada no momento em que for declarada, não deixando espaço para otimizações. O segundo, no entanto, espera até que a execução seja explicitamente chamada. Assim, uma série de portas pode ser otimizada antes da execução. Portanto, a otimização só é realizada se o modo de simulação for *batch*.

Código 6 – Exemplo de configuração de processo Ket com otimização

```
p = Process(execution="batch", optimize=true)
a, b = p.alloc(2)

CNOT(H(a), b) # Bell state preparation

d = dump(a + b)

p.execute()
```

4 EXPERIMENTO E ANÁLISE DE RESULTADOS

Este capítulo avalia a implementação do algoritmo de otimização na Plataforma Ket. Para analisar a eficácia das otimizações implementadas, utilizamos duas abordagens: a primeira envolve circuitos gerados aleatoriamente com diferentes proporções de portas T; a segunda considera circuitos quânticos relevantes para a computação quântica, como somadores, oráculos de Grover e outros circuitos quânticos gerados aleatoriamente. Esses circuitos foram obtidos do conjunto de testes utilizado para medir o desempenho da biblioteca PyZX (KISSINGER; WETERING, 2020c). Além disso, verificamos a equivalência dos circuitos originais com os otimizados nos dois conjuntos apresentados. Ambos os conjuntos de circuitos estão disponíveis no Github do projeto [1].

O desempenho da implementação do algoritmo na plataforma Ket foi diretamente comparado com a biblioteca Python *PyZX*, por ser a mais completa em implementações relacionadas ao cálculo-ZX. Esta biblioteca abrange o algoritmo de otimização apresentado e rotinas de otimização mencionadas, mas não exploradas neste trabalho, como a teletransportação de fase (KISSINGER; WETERING, 2020b). Com isso em mente, além da comparação direta entre a implementação dos algoritmos (otimização por grafos), foi feita uma comparação com a rotina completa de otimização, que além do algoritmo apresentado, utiliza a teletransportação de fase.

Os *benchmarks* foram executados em uma máquina Linux (6.9.2-arch1-1), com CPU AMD Ryzen 5 3600 (12) @ 3.600GHz e com dois pentes de RAM 8GB DDR4 2666 MT/s. A versão de Python utilizada é a 3.11 e a do compilador Rust é a 1.75.0. Todos os testes foram feitos nas mesmas condições sem tarefas de *background* consumindo quantidades significativas de recursos do sistema.

4.1 VALIDAÇÃO DOS RESULTADOS

Após um processo de otimização, é imperativo verificar que os circuitos são equivalentes. Com esse objetivo, a técnica de validação de circuitos do cálculo-ZX (PEHAM; BURGHOLZER; WILLE, 2022) foi empregada, onde o adjunto do circuito original é adicionado ao circuito otimizado, e reduções são executadas. Para que um circuito otimizado seja equivalente, ele deve resultar na matriz de identidade após sua execução. Tanto o conjunto de algoritmos quanto os circuitos Clifford+T gerados aleatoriamente foram testados, e todos resultaram na identidade, validando a corretude da otimização. Os testes utilizaram a implementação da rotina de validação feita pela biblioteca *PyZX*.

[1] Conjunto de circuitos de testes está disponível em: https://github.com/gabsilvcar/ket/tree/master/benchmarks/circuits.

4.2 BENCHMARKS COM CIRCUITOS QUÂNTICOS GERADOS ALEATORIAMENTE

Este conjunto é formado por circuitos compostos por portas Clifford+T colocadas aleatoriamente. A implementação utilizada é da biblioteca *PyZX* onde a geração do circuito é feita por meio de um sorteio de portas onde todas têm a mesma probabilidade de serem escolhidas em cada inserção. Os parâmetros utilizados para a geração do circuito foram dez qubits e mil portas lógicas. As portas utilizadas foram T, S, HSH ($= e^{\pi/4} Rx(\frac{\pi}{2})$) e CNOT.

Para verificar o comportamento da otimização conforme a redução de portas T, é estabelecida uma proporção dessas portas nos circuitos gerados e os resultados são agrupados conforme sua proporção original. O algoritmo se mostrou eficaz, como ilustrado na Figura 28 e é possível observar também que o resultado está alinhado com a implementação do mesmo algoritmo na biblioteca *PyZX*, evidenciando a corretude da implementação.

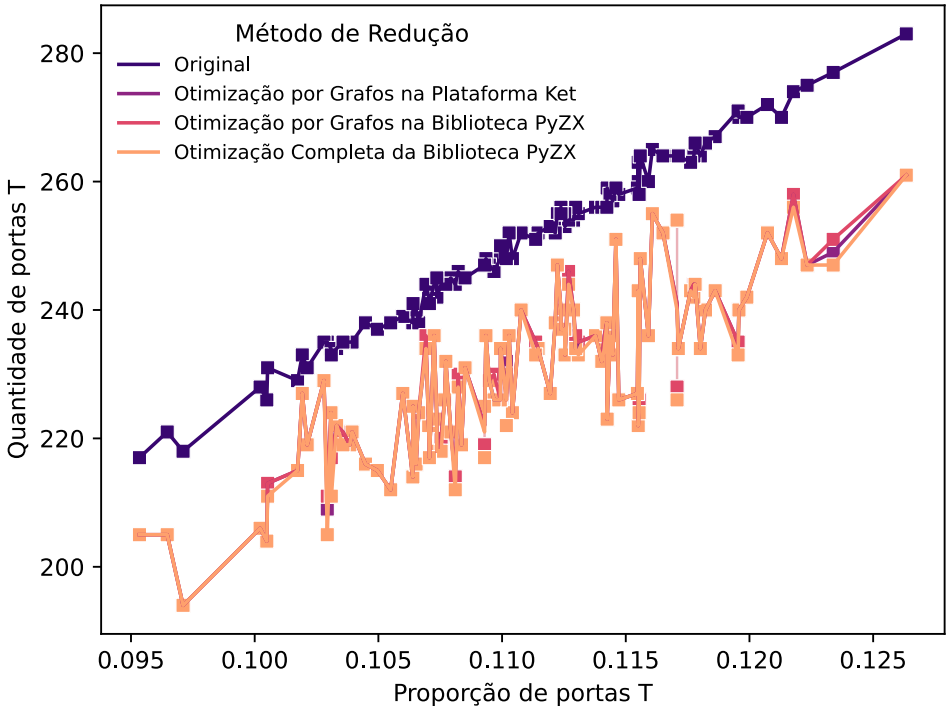


Figura 28 – Comparação de diferentes métodos de otimização de circuito em relação a quantidade de portas T em um conjunto de circuitos gerados aleatoriamente.

O aumento do circuito devido ao processo de extração, particularmente no algoritmo de extração usado pela plataforma Ket, pode ser observado na Figura 29. Para evitar este aumento, o *PyZX* adota uma estratégia que não é adotada pelo Ket, que é separar partes do circuito em circuitos de Clifford e sintetizá-los utilizando o algoritmo de Patel, Markov e Hayes (2004), apenas utilizando o algoritmo de extração de Backens, Miller-Bakewell *et al.* (2021) onde é necessário. Isto causa uma diferença

substancial no número total de portas do circuito, como é possível observar no gráfico apresentado. Além disso, o PyZX incorpora uma série de pequenas otimizações que têm um impacto no circuito final.

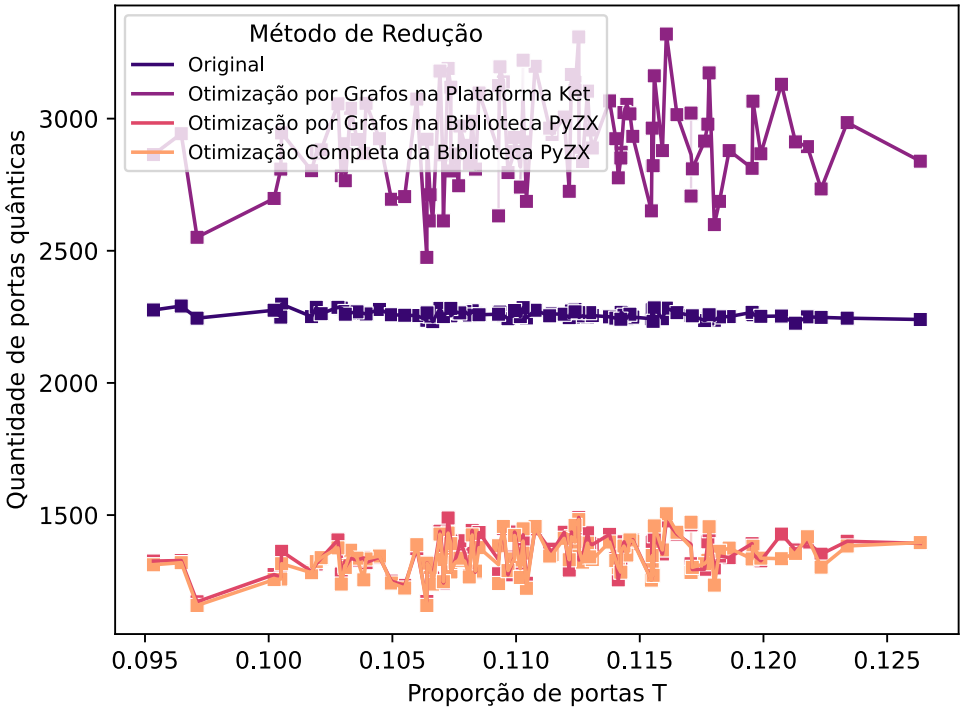


Figura 29 – Comparação de diferentes métodos de otimização de circuito em relação à quantidade de portas num conjunto de circuitos gerados aleatoriamente.

Outra métrica notável é o ganho de desempenho que a plataforma Ket obteve ao usar a implementação em Rust do *quizx*, apresentando uma diferença de ordem de magnitude no tempo de execução dos algoritmos feitos em Python. Esta diferença pode ser observada no gráfico da Figura 30.

4.3 BENCHMARKS COM CIRCUITOS QUÂNTICOS DE ALGORITMOS

Os circuitos analisados nesse *benchmark* podem ser categorizados em três tipos principais. Primeiramente, temos os componentes do algoritmo de fatoração de Shor, que incluem circuitos da Transformada de Fourier Quântica (QFT), circuitos somadores da biblioteca aritmética do Quipper e somadores baseados em QFT, esses circuitos são cruciais para a implementação do algoritmo de Shor. Em segundo lugar, há os circuitos aritméticos e de Toffoli, fundamentais para operações lógicas e de aritmética em computação quântica. Por último, encontram-se os códigos de Hamming, utilizados para a correção de erro. Para que as portas usadas pelos algoritmos sejam consistentes com as da otimização, o circuito teve suas portas expressas em termos de fases X e Z, Hadamards, CNOT e CZ.

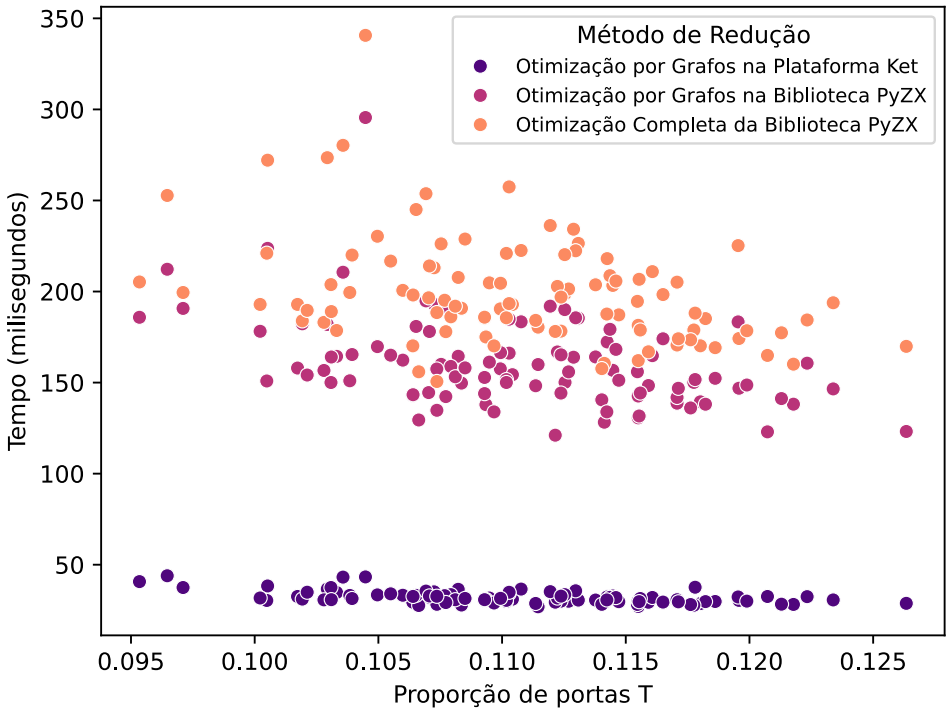


Figura 30 – Comparação de diferentes métodos de otimização de circuito em relação a tempo de otimização e extração num conjunto de circuitos gerados aleatoriamente.

No gráfico da Figura 31, apresentamos os resultados obtidos de otimização de portas T em multiplicadores de campo de Galois, ou apenas GF(2), que são componentes utilizados para fatoração e computação de logaritmos discretos. É possível observar um ganho proporcional ao aumento do circuito, onde é possível reduzir mais portas T devido ao aumento do número de aranhas que podem ser canceladas pela técnica de otimização.

É possível observar também o aumento na quantidade de portas lógicas e profundidade nos circuitos do *benchmark* de portas Toffoli na Figura 32. São apresentadas duas implementações, a de Barenco *et al.* (1995) utilizando qubits-auxiliares em estados arbitrários e de Nielsen e Chuang (2010) utilizando qubits-auxiliares em estados $|0\rangle$. Esse resultado deixa evidente o impacto negativo que o processo de extração tem no circuito.

O resultado do *benchmark* é apresentado de maneira mais detalhada na Figura 33. Nela, é possível observar com mais detalhe informações como a profundidade do circuito, tempo de execução e quantidade de portas total, T e de 2-qubit. Além disso, estão em destaque os resultados obtidos pelos maiores circuitos do *benchmark*, onde é possível observar um aumento severo no tempo de otimização para a biblioteca *PyZX*, enquanto a plataforma Ket manteve um bom desempenho dada a complexidade do circuito.

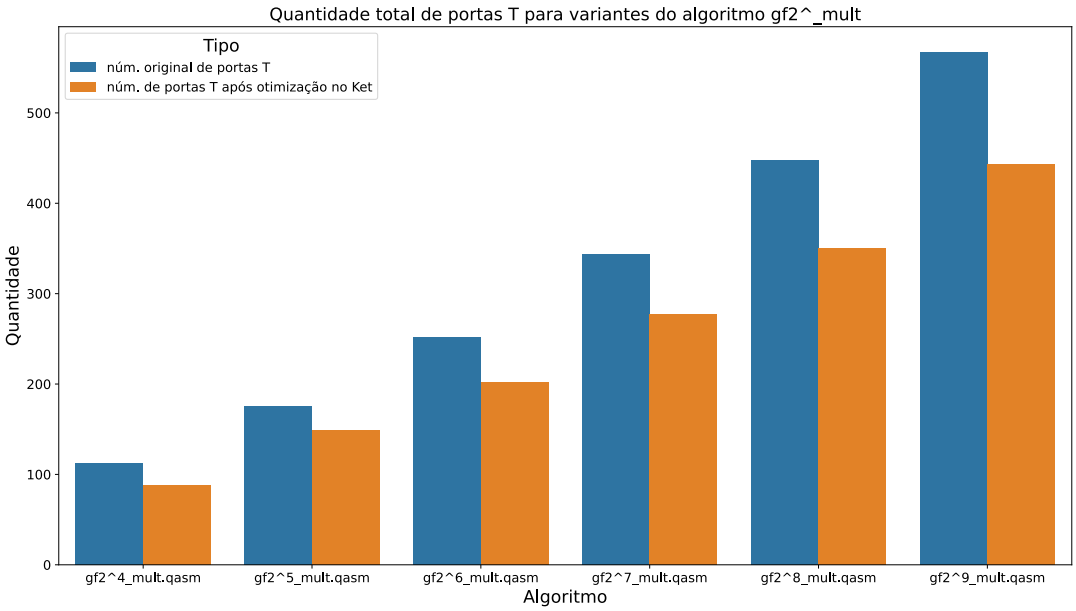
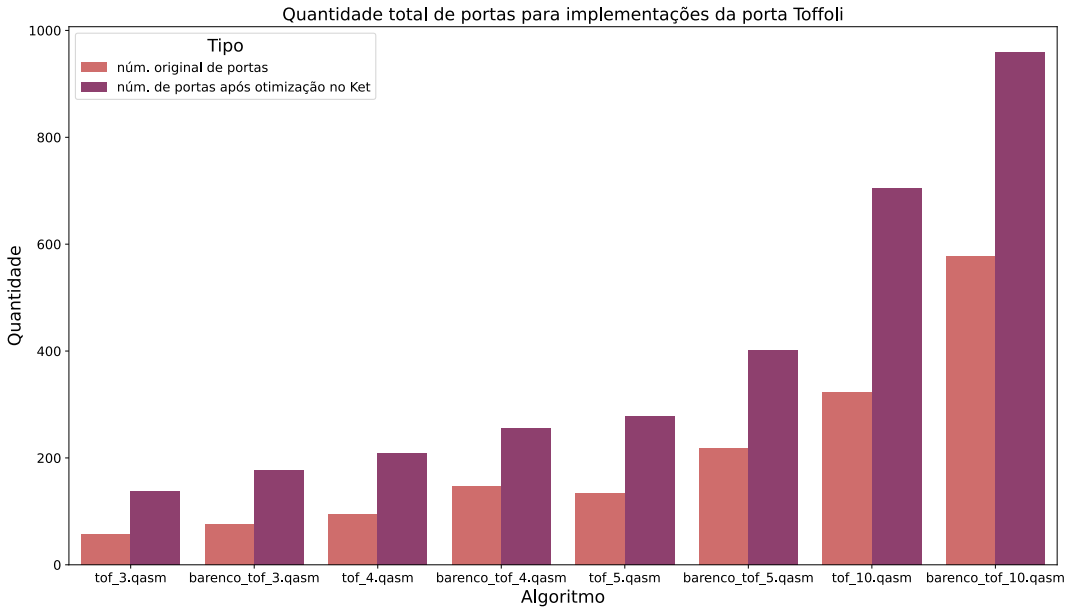
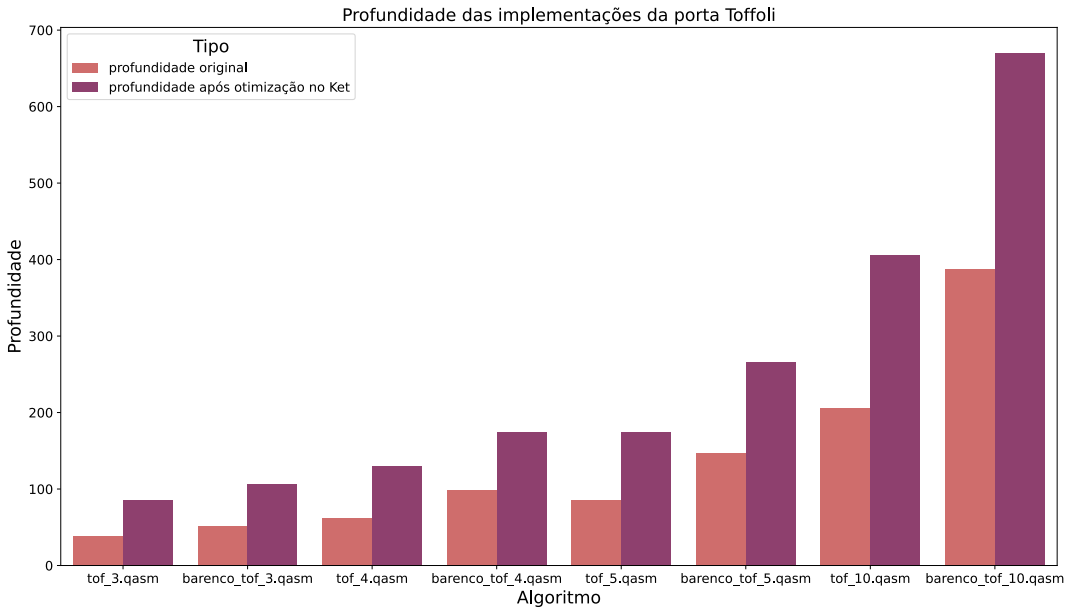


Figura 31 – Resultados obtidos para variantes do circuito GF(2).



(a) Gráfico do número total de portas para implementações da porta Toffoli



(b) Gráfico de profundidade para implementações da porta Toffoli

Figura 32 – Resultados obtidos para diferentes implementações da porta Toffoli.

Circuito	Original				Otimização por Grafos na Plataforma Ket					Otimização por Grafos na Biblioteca PyZX					Otimização Completa na Biblioteca PyZX				
	Σ	T-Σ	Depth	$2Q$	Σ	T-Σ	Depth	$2Q$	Δt	Σ	T-Σ	Depth	$2Q$	Δt	Σ	T-Σ	Depth	$2Q$	Δt
adder_8.qasm	1128	399	282	409	2259	259	1090	670	0.026	1614	355	466	552	0.541	932	173	471	441	0.708
barenco_tof_3.qasm	76	28	51	24	176	24	106	43	0.001	115	24	75	39	0.003	80	16	56	30	0.005
barenco_tof_4.qasm	146	56	99	48	256	32	174	70	0.002	216	48	137	67	0.009	148	28	106	63	0.013
barenco_tof_5.qasm	218	84	147	72	402	58	266	107	0.003	315	72	200	93	0.017	207	40	145	91	0.026
barenco_tof_10.qasm	578	224	387	192	959	130	670	304	0.009	815	192	520	228	0.144	523	100	372	240	0.256
csla_mux_3.qasm	210	70	80	80	724	62	237	193	0.003	296	64	108	112	0.023	344	62	138	178	0.030
csum_mux_9.qasm	532	196	72	168	1783	166	342	409	0.008	881	196	119	221	0.135	619	84	154	341	0.175
gf2^4_*mult.qasm*	289	112	133	99	1317	88	459	364	0.006	436	96	204	136	0.032	531	68	207	324	0.048
gf2^5_*mult.qasm*	447	175	168	154	2471	149	755	683	0.011	663	155	267	198	0.065	782	115	314	458	0.083
gf2^6_*mult.qasm*	639	252	217	221	4239	202	1388	1220	0.020	945	216	356	281	0.121	1304	150	545	838	0.160
gf2^7_*mult.qasm*	865	343	259	300	5454	277	1647	1562	0.029	1263	301	398	360	0.182	1861	217	784	1236	0.259
gf2^8_*mult.qasm*	1139	448	307	405	7530	350	2202	2198	0.045	1643	384	470	485	0.376	2530	264	1084	1744	0.483
gf2^9_*mult.qasm*	1419	567	336	494	9348	443	2331	2703	0.054	2052	495	506	567	0.473	3091	351	1154	2078	0.606
gf2^{10}_*mult.qasm*	1747	700	378	609	14399	526	4421	4305	0.091	2525	600	622	699	0.798	4125	410	1644	2876	0.976
gf2^{16}_*mult.qasm*	4459	1792	643	1581	48161	1486	12463	14652	0.705	6406	1536	1172	1778	5.441	11177	1040	4028	8065	6.815
grover_5.qasm	1023	336	593	288	1372	206	961	399	0.012	1191	296	800	354	0.090	753	166	598	327	0.170
ham15-low.qasm	535	161	317	236	1257	121	720	400	0.011	811	147	506	398	0.110	629	97	402	368	0.234
ham15-med.qasm	1600	574	895	534	2359	316	1531	776	0.066	2017	492	1207	635	0.343	1102	212	735	528	0.706
ham15-high.qasm	6712	2457	3694	2149	9603	1397	7260	3520	0.362	8596	2165	4956	2457	2.193	5132	1019	3922	2415	8.714
hwb6.qasm	319	105	194	116	757	85	470	223	0.005	426	95	302	156	0.023	369	75	225	157	0.042
mod_adder_1024.qasm	5425	1995	3034	1720	9305	1215	6411	3314	0.164	6897	1739	4206	1998	6.759	5470	1011	3821	2777	47.283
mod_mult_55.qasm	147	49	60	48	430	41	216	112	0.003	221	43	114	71	0.009	196	35	96	88	0.016
mod_red_21.qasm	346	119	196	105	641	89	420	187	0.005	465	107	275	150	0.034	370	73	241	171	0.058
mod5_4.qasm	79	28	59	28	188	14	86	46	0.001	113	22	83	40	0.004	57	8	32	25	0.005
qcla_adder_10.qasm	657	238	91	233	1589	190	482	445	0.014	923	212	148	295	0.548	849	162	320	424	0.530
qcla_com_7.qasm	559	203	102	186	1143	123	310	330	0.009	764	169	166	254	0.264	561	95	170	271	0.465
qcla_mod_7.qasm	1120	413	249	382	2645	301	1012	835	0.029	1561	351	456	534	0.697	1426	237	551	763	4.157
qft_4.qasm	187	69	152	46	346	67	237	77	0.003	234	67	176	71	0.011	229	67	174	64	0.013
rc_adder_6.qasm	244	77	129	93	558	65	305	138	0.004	334	65	190	133	0.040	270	47	161	128	0.040
tof_3.qasm	57	21	38	18	138	17	85	33	0.001	87	19	57	29	0.002	72	15	52	33	0.004
tof_4.qasm	95	35	62	30	209	29	130	52	0.002	144	31	94	49	0.005	118	23	70	54	0.008
tof_5.qasm	133	49	86	42	278	33	174	79	0.002	199	43	129	67	0.008	157	31	109	75	0.014
tof_10.qasm	323	119	206	102	704	85	406	193	0.005	474	103	304	157	0.070	361	71	224	166	0.069
vbe_adder_3.qasm	190	70	97	70	433	42	205	125	0.003	274	56	148	101	0.017	173	24	93	89	0.019

Figura 33 – Benchmark de otimização de circuitos. Estão em destaque os circuitos de maior custo de processamento do benchmark. Legenda: Σ: quantidade total de portas, T-Σ: quantidade de portas T, Depth: profundidade do circuito, $2Q$: portas de 2-qubits, Δt: tempo de execução da otimização

5 CONCLUSÕES E TRABALHOS FUTUROS

Uma contribuição desta pesquisa é um *fork* da Plataforma Ket com suporte para cálculo-ZX e o algoritmo de otimização de Duncan *et al.* (2020). O repositório está disponível em https://github.com/gabsilvcar/ket. Todos os resultados obtidos por benchmarks e parâmetros para obtenção dos circuitos gerados aleatoriamente também estão inclusos.

Dadas as perspectivas futuras da computação quântica tolerante a erros, este trabalho cumpriu seu objetivo de identificar o estado da arte para algoritmos e métricas de otimização (Objetivo i). Foi apresentado que as portas T ditam desproporcionalmente a quantidade de recursos gastos pela correção de erros e sua redução foi o foco principal do trabalho, mas outras formas de otimização foram apresentadas.

O trabalho apresentou o cálculo-ZX para fazer otimizações, explorando suas nuances de representação, conversão de circuitos quânticos e, subsequentemente, extração dos diagramas. Sua integração com a Plataforma Ket foi obtida a partir da biblioteca *quizx* de cálculo-ZX (Objetivo iv) que se mostrou adequada para o caso de desenvolvimento de software do Ket, porque não atrela a representação interna da plataforma com a representação-ZX, permitindo que o desenvolvimento futuro e expansão de funcionalidades sejam feitas separadamente (Objetivo ii). A comunicação da mesma com a *libket* (*backend* da plataforma) ocorre mediante uma linguagem de descrição de circuitos intermediária, o QASM (Objetivo iii).

Considerando os resultados obtidos, a otimização de portas T na Plataforma Ket foi um sucesso, apresentando ganhos para circuitos aleatórios e circuitos de algoritmos de aplicações reais da computação quântica. A implementação obteve também um ótimo resultado de desempenho, onde o processo completo de otimização (*parse* do assembly quântico, conversão, otimização, extração) é feito duas ordens de grandeza mais rápido do que implementações em Python. A validade dos resultados foi averiguada utilizando as técnicas de equivalência de circuito proporcionadas pelo cálculo-ZX, cumprindo assim o Objetivo v.

Foram avaliadas também as limitações dessa técnica de otimização (Objetivo vi), em específico os prejuízos que ela pode trazer para a profundidade do circuito, quantidade total de portas e número de portas T. Foi observado uma grande perda nessas métricas, mas elas podem ser amenizadas com outras técnicas de otimizações apresentadas no texto, em especial, a extração pode ser feita de maneira mais eficiente para casos gerais implementando algoritmos de otimização de portas CZ e CNOT, reduzindo o número de portas de 2-qubits, além de cancelamentos simples de portas lógicas para diminuir o número total de portas no circuito.

Para casos nos quais o circuito após a otimização é um circuito de Clifford, ou seja, todas as portas T foram removidas, é possível obter uma extração assintótica

para fazer a síntese desse circuito usando o algoritmo proposto por Patel, Markov e Hayes (2004), podendo ser implementado na Plataforma Ket no futuro. Com isso, seria possível alterar a estratégia de otimização e alinhar ela com o que é feito na biblioteca *PyZX*, onde o algoritmo de extração é usado apenas nas seções do circuito que não são circuitos de Clifford.

É possível também integrar outros algoritmos de otimização de cálculo-ZX, como de Staudacher *et al.* (2023) para redução de portas CZ e o utilizar o conceito de teleporte de fase de Kissinger e Wetering (2020b) para reduzir mais o número de portas T. O segundo algoritmo não necessita de um algoritmo de extração após ser executado, ou seja, não adicionará mais portas no processo de extração.

Em relação ao cálculo-ZX, ainda não existem métodos para utilizarem qubits auxiliares, também chamadas de *ancillas*, para otimizações. O uso de ancillas poderia diminuir o número de portas utilizadas e permitir a sinteze de circuitos quânticos mais eficientes.

No algoritmo de extração apresentado, o processo de síntese de CNOTs não leva em conta a topologia dos qubits restritas adotadas na arquitetura de computadores quânticos. Essas topologias têm limitação na conectividade entre os qubits em um processador quântico. Em vez de qualquer qubit poder interagir diretamente com qualquer outro, as interações são permitidas apenas entre qubits que estão próximos ou conectados de maneira específica. É possível adicionar restrições na etapa de limitação gaussiana usando estratégias de Árvores de Steiner como apresentado por Nash, Gheorghiu e Mosca (2020) e Kissinger e Griend (2019).

REFERÊNCIAS

AARONSON, Scott; GOTTESMAN, Daniel. Improved simulation of stabilizer circuits. **Physical Review A**, American Physical Society (APS), v. 70, n. 5, nov. 2004. ISSN 1094-1622. DOI: 10.1103/physreva.70.052328. Disponível em: <http://dx.doi.org/10.1103/PhysRevA.70.052328>. Citado na p. 20.

AHARONOV, D.; BEN-OR, M. Fault-tolerant quantum computation with constant error. *In:* PROCEEDINGS of the Twenty-Ninth Annual ACM Symposium on Theory of Computing. El Paso, Texas, USA: Association for Computing Machinery, 1997. (STOC '97), p. 176–188. DOI: 10.1145/258533.258579. Disponível em: <https://doi.org/10.1145/258533.258579>. Citado na p. 25.

AMY, Matthew; MASLOV, Dmitri; MOSCA, Michele. Polynomial-Time T-Depth Optimization of Clifford+T Circuits Via Matroid Partitioning. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 33, n. 10, p. 1476–1489, 2014. DOI: 10.1109/TCAD.2014.2341953. Citado na p. 25.

AMY, Matthew; MASLOV, Dmitri; MOSCA, Michele; ROETTELER, Martin. A Meet-in-the-Middle Algorithm for Fast Synthesis of Depth-Optimal Quantum Circuits. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 32, n. 6, p. 818–830, 2013. DOI: 10.1109/TCAD.2013.2244643. Citado na p. 25.

BACKENS, Miriam. The ZX-calculus is complete for stabilizer quantum mechanics. **New Journal of Physics**, IOP Publishing, v. 16, n. 9, p. 093021, set. 2014. DOI: 10.1088/1367-2630/16/9/093021. Disponível em: <https://dx.doi.org/10.1088/1367-2630/16/9/093021>.

BACKENS, Miriam; KISSINGER, Aleks. ZH: A Complete Graphical Calculus for Quantum Computations Involving Classical Non-linearity. **Electronic Proceedings in Theoretical Computer Science**, Open Publishing Association, v. 287, p. 23–42, jan. 2019. ISSN 2075-2180. DOI: 10.4204/eptcs.287.2. Disponível em: <http://dx.doi.org/10.4204/EPTCS.287.2>. Citado na p. 32.

BACKENS, Miriam; MILLER-BAKEWELL, Hector *et al.* There and back again: A circuit extraction tale. **Quantum**, Verein zur Forderung des Open Access Publizierens in den Quantenwissenschaften, v. 5, p. 421, mar. 2021. ISSN 2521-327X. DOI: 10.22331/q-2021-03-25-421. Disponível em: <http://dx.doi.org/10.22331/q-2021-03-25-421>. Citado nas pp. 33, 37, 52.

BACKENS, Miriam; PERDRIX, Simon; WANG, Quanlong. A Simplified Stabilizer ZX-calculus. **Electronic Proceedings in Theoretical Computer Science**, Open Publishing Association, v. 236, p. 1–20, jan. 2017. ISSN 2075-2180. DOI: 10.4204/eptcs.236.1. Disponível em: <http://dx.doi.org/10.4204/EPTCS.236.1>.

BARENCO, Adriano *et al.* Elementary gates for quantum computation. **Physical Review A**, American Physical Society (APS), v. 52, n. 5, p. 3457–3467, nov. 1995. ISSN 1094-1622. DOI: 10.1103/physreva.52.3457. Disponível em: <http://dx.doi.org/10.1103/PhysRevA.52.3457>. Citado na p. 54.

BROWNE, Daniel E *et al.* Generalized flow and determinism in measurement-based quantum computation. **New Journal of Physics**, v. 9, n. 8, p. 250, ago. 2007. DOI: 10.1088/1367-2630/9/8/250. Disponível em: <https://dx.doi.org/10.1088/1367-2630/9/8/250>. Citado nas pp. 32, 33.

CALDERBANK, A. R.; SHOR, Peter W. Good quantum error-correcting codes exist. **Physical Review A**, American Physical Society (APS), v. 54, n. 2, p. 1098–1105, ago. 1996. ISSN 1094-1622. DOI: 10.1103/physreva.54.1098. Disponível em: <http://dx.doi.org/10.1103/PhysRevA.54.1098>. Citado na p. 25.

CAMPBELL, Earl T; TERHAL, Barbara M; VUILLOT, Christophe. Roads towards fault-tolerant universal quantum computation. **Nature**, Nature Publishing Group UK London, v. 549, n. 7671, p. 172–179, 2017. Citado na p. 25.

CAMPBELL, Earl T.; TERHAL, Barbara M.; VUILLOT, Christophe. Roads towards fault-tolerant universal quantum computation. **Nature**, Springer Science e Business Media LLC, v. 549, n. 7671, p. 172–179, set. 2017. ISSN 1476-4687. DOI: 10.1038/nature23460. Disponível em: <http://dx.doi.org/10.1038/nature23460>. Citado na p. 20.

COECKE, Bob; DUNCAN, Ross. Interacting quantum observables: categorical algebra and diagrammatics. **New Journal of Physics**, IOP Publishing, v. 13, n. 4, p. 043016, abr. 2011. ISSN 1367-2630. DOI: 10.1088/1367-2630/13/4/043016. Disponível em: <http://dx.doi.org/10.1088/1367-2630/13/4/043016>. Citado nas pp. 20, 26, 31.

COMMONS, Wikimedia. **File:Quantum Logic Gates.png — Wikimedia Commons, the free media repository**. [*S.l.*: *s.n.*], 2022. [Online; accessed 31-May-2024]. Disponível em: <https://commons.wikimedia.org/w/index.php?title=File:Quantum_Logic_Gates.png&oldid=703298762>. Citado na p. 24.

COMMONS, Wikimedia. **ZX-calculus green spider fusion rule**. 2019. Disponível em: <https://en.wikipedia.org/wiki/ZX-calculus#/media/File:ZX-calculus_green_spider_fusion_rule.svg>. Citado na p. 28.

COMMONS, Wikimedia. **ZX-calculus red spider fusion rule**. 2019. Disponível em: <https://en.wikipedia.org/wiki/ZX-calculus#/media/File:ZX-calculus_red_spider_fusion_rule.svg>. Citado na p. 28.

CROSS, Andrew W. *et al.* **Open Quantum Assembly Language**. [*S.l.*: *s.n.*], 2017. arXiv: 1707.03429 [quant-ph]. Citado nas pp. 43, 45, 47.

DA ROSA, Evandro Chagas Ribeiro; DE SANTIAGO, Rafael. Ket quantum programming. **ACM Journal on Emerging Technologies in Computing Systems (JETC)**, ACM New York, NY, v. 18, n. 1, p. 1–25, 2021. Citado nas pp. 20, 43.

DANOS, Vincent; KASHEFI, Elham. Determinism in the one-way model. **Phys. Rev. A**, American Physical Society, v. 74, p. 052310, 5 nov. 2006. DOI: 10.1103/PhysRevA.74.052310. Disponível em: <https://link.aps.org/doi/10.1103/PhysRevA.74.052310>. Citado na p. 33.

DEUTSCH, David Elieser; BARENCO, Adriano; EKERT, Artur. Universality in quantum computation. **Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences**, The Royal Society London, v. 449, n. 1937, p. 669–677, 1995. Citado na p. 20.

DUNCAN, Ross *et al.* Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. **Quantum**, Verein zur Forderung des Open Access Publizierens in den Quantenwissenschaften, v. 4, p. 279, jun. 2020. DOI: 10.22331/q-2020-06-04-279. Disponível em: <https://doi.org/10.22331%2Fq-2020-06-04-279>. Citado nas pp. 20, 25, 32, 34, 36–42, 58.

GIDNEY, Craig; FOWLER, Austin G. Efficient magic state factories with a catalyzed |CCZ> to 2|T> transformation. **Quantum**, Verein zur Forderung des Open Access Publizierens in den Quantenwissenschaften, v. 3, p. 135, abr. 2019. ISSN 2521-327X. DOI: 10.22331/q-2019-04-30-135. Disponível em: <http://dx.doi.org/10.22331/q-2019-04-30-135>. Citado nas pp. 20, 25.

HEIN, M. Entanglement in graph states and its applications. **Proc. International School of Physics" Enrico Fermi" on" Quantum Computers, Algorithms and Chaos," Varenna, Italy, July 2005**, 2005. Citado na p. 35.

JEANDEL, Emmanuel; PERDRIX, Simon; VILMART, Renaud. Completeness of the ZX-Calculus. **Logical Methods in Computer Science**, jun. 2020. DOI: 10.23638/LMCS-16(2:11)2020. Citado na p. 31.

KISSINGER, Aleks; GRIEND, Arianne Meijer-van de. **CNOT circuit extraction for topologically-constrained quantum memories**. [*S.l.*: *s.n.*], 2019. arXiv: 1904.00633 [quant-ph]. Disponível em: <https://arxiv.org/abs/1904.00633>. Citado na p. 59.

KISSINGER, Aleks; WETERING, John van de. PyZX: Large Scale Automated Diagrammatic Reasoning. **Electronic Proceedings in Theoretical Computer Science**, Open Publishing Association, v. 318, p. 229–241, mai. 2020. ISSN 2075-2180. DOI: 10.4204/eptcs.318.14. Disponível em: <http://dx.doi.org/10.4204/EPTCS.318.14>. Citado na p. 45.

KISSINGER, Aleks; WETERING, John van de. Reducing the number of non-Clifford gates in quantum circuits. **Physical Review A**, American Physical Society (APS), v. 102, n. 2, ago. 2020. ISSN 2469-9934. DOI: 10.1103/physreva.102.022406. Disponível em: <http://dx.doi.org/10.1103/PhysRevA.102.022406>. Citado nas pp. 25, 51, 59.

KISSINGER, Aleks; WETERING, John van de. Reducing the number of non-Clifford gates in quantum circuits. **Physical Review A**, American Physical Society (APS), v. 102, n. 2, ago. 2020. ISSN 2469-9934. DOI: 10.1103/physreva.102.022406. Disponível em: <http://dx.doi.org/10.1103/PhysRevA.102.022406>. Citado na p. 51.

MASLOV, Dmitri. **Reversible Logic Synthesis Benchmarks Page**. [*S.l.*: *s.n.*]. Disponível em: <http://webhome.cs.uvic.ca/~dmaslov/>.

MATTEO, Olivia Di; MOSCA, Michele. Parallelizing quantum circuit synthesis. **Quantum Science and Technology**, IOP Publishing, v. 1, n. 1, p. 015003, out. 2016. DOI: 10.1088/2058-9565/1/1/015003. Disponível em: <https://dx.doi.org/10.1088/2058-9565/1/1/015003>. Citado na p. 25.

NASH, Beatrice; GHEORGHIU, Vlad; MOSCA, Michele. Quantum circuit optimizations for NISQ architectures. **Quantum Science and Technology**, IOP Publishing, v. 5, n. 2, p. 025010, mar. 2020. ISSN 2058-9565. DOI: 10.1088/2058-9565/ab79b1. Disponível em: <http://dx.doi.org/10.1088/2058-9565/ab79b1>. Citado na p. 59.

NG, Kang Feng; WANG, Quanlong. Completeness of the ZX-calculus for Pure Qubit Clifford+T Quantum Mechanics. **arXiv:1801.07993**, jan. 2018. Citado na p. 25.

NIELSEN, Michael A.; CHUANG, Isaac L. **Quantum Computation and Quantum Information**. Cambridge, USA: Cambridge University Press, 2010. DOI: https://doi.org/10.1017/CBO9780511976667. Citado nas pp. 23, 25, 54.

O'GORMAN, Joe; CAMPBELL, Earl T. Quantum computation with realistic magic-state factories. **Physical Review A**, APS, v. 95, n. 3, p. 032338, 2017.

PATEL, Ketan; MARKOV, Igor; HAYES, John. Optimal Synthesis of Linear Reversible Circuits. **Quantum Information and Computation**, v. 8, mai. 2004. DOI: 10.26421/QIC8.3-4-4. Citado nas pp. 26, 52, 59.

PEHAM, Tom; BURGHOLZER, Lukas; WILLE, Robert. Equivalence Checking of Quantum Circuits With the ZX-Calculus. **IEEE Journal on Emerging and Selected Topics in Circuits and Systems**, Institute of Electrical e Electronics Engineers (IEEE), v. 12, n. 3, p. 662–675, set. 2022. ISSN 2156-3365. DOI: 10.1109/jetcas.2022.3202204. Disponível em: <http://dx.doi.org/10.1109/JETCAS.2022.3202204>. Citado na p. 51.

POLLACHINI, Giovani Goraiebe. **Computação Quântica: Uma abordagem para estudantes de graduação em Ciências Exatas**. 2018. Trabalho de Conclusão de Curso de Graduação – UFSC. Disponível em: <http://gcq.ufsc.br/lib/exe/fetch.php?media=wiki:tcc-giovani.pdf>. Citado na p. 23.

RAUSSENDORF, Robert; BRIEGEL, Hans J. A One-Way Quantum Computer. **Phys. Rev. Lett.**, American Physical Society, v. 86, p. 5188–5191, 22 mai. 2001. DOI: 10.1103/PhysRevLett.86.5188. Disponível em: <https://link.aps.org/doi/10.1103/PhysRevLett.86.5188>.

RAUSSENDORF, Robert; HARRINGTON, Jim. Fault-Tolerant Quantum Computation with High Threshold in Two Dimensions. **Physical Review Letters**, American Physical Society (APS), v. 98, n. 19, mai. 2007. ISSN 1079-7114. DOI: 10.1103/physrevlett.98.190504. Disponível em: <http://dx.doi.org/10.1103/PhysRevLett.98.190504>. Citado na p. 20.

ROSA, Evandro Chagas Ribeiro da; TAKETANI, Bruno G. QSystem: bitwise representation for quantum circuit simulations. **arXiv preprint arXiv:2004.03560**, 2020. DOI: https://doi.org/10.48550/arXiv.2004.03560. arXiv: 2004.03560. Citado na p. 43.

STAUDACHER, Korbinian. **Optimization Approaches for Quantum Circuits using ZX-calculus**. 2021. Tese (Doutorado) – Master's thesis, Ludwig-Maximilians-Universität, München. Disponível em: <https://www.mnm-team.org/pub/Diplomarbeiten/stau21/>. Citado na p. 26.

STAUDACHER, Korbinian *et al.* Reducing 2-QuBit Gate Count for ZX-Calculus based Quantum Circuit Optimization. **Electronic Proceedings in Theoretical Computer Science**, Open Publishing Association, v. 394, p. 29–45, nov. 2023. ISSN 2075-2180. DOI: 10.4204/eptcs.394.3. Disponível em: <http://dx.doi.org/10.4204/EPTCS.394.3>. Citado nas pp. 26, 59.

WETERING, John van de. ZX-calculus for the working quantum computer scientist. **arXiv preprint arXiv:2012.13966**, 2020. DOI: https://doi.org/10.48550/arXiv.2012.13966. Citado nas pp. 26, 29.

WETERING, John van de; AMY, Matt. Optimising T-count is NP-hard. **ArXiv**, abs/2310.05958, 2023. Disponível em: <https://api.semanticscholar.org/CorpusID:263829327>. Citado na p. 25.

6 APÊNDICES A - ARTIGO SBC

Otimização de Circuitos Quânticos Utilizando Cálculo-ZX na Plataforma Ket

Gabriel S. Cardoso[1], **Evandro Chagas**[1], **Jerusa Marchi**[1]

[1]Departamento de Informatica e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
CEP: 88040-370 – Florianópolis – SC – Brasil

Abstract. *Quantum Computing promises improvements in problems like prime factorization (Shor's algorithm) and unordered list search (Grover's algorithm). However, noise in quantum states compromises calculation accuracy. Error correction with fault-tolerant gates can overcome this obstacle, but gates like the T gate have high implementation costs. This work explores ZX-calculus to optimize quantum circuits, implementing an algorithm on the Ket Platform. A transpiler is developed to enable communication between ket code and ZX diagrams via OpenQASM 2.0. Results are evaluated in terms of T gate reduction, total gates, critical path, 2-qubit gates, and optimization execution time.*

Resumo. *A Computação Quântica promete ganhos em problemas como a fatoração de números primos (algoritmo de Shor) e a busca em listas desordenadas (algoritmo de Grover). Contudo, o ruído nos estados quânticos compromete a precisão dos cálculos. A correção de erros com portas tolerantes a falhas pode superar esse obstáculo, mas portas como a T têm alto custo de implementação. Este trabalho explora o cálculo-ZX para otimizar circuitos quânticos, implementando um algoritmo na Plataforma Ket. Um transpilador é desenvolvido para comunicação entre código ket e diagramas-ZX via OpenQASM 2.0. Os resultados são avaliados quanto à redução de portas T, total de portas, caminho crítico, portas de 2-qubits e tempo de execução da otimização.*

1. Introdução e Motivação

Convencionalmente, a representação de algoritmos quânticos é feita por circuitos que descrevem as operações lineares. O cálculo-ZX, introduzido por [Coecke and Duncan 2011], é uma alternativa que usa diagramas visuais para representar e simplificar mapas lineares via regras de reescrita. Este trabalho aborda otimizações de circuitos aplicando tais regras, especificamente o algoritmo de [Duncan et al. 2020], e sua implementação na plataforma Ket[1] [Da Rosa and De Santiago 2021], que permite o desenvolvimento de aplicações quânticas híbridas, integrando-se ao ecossistema Python e oferecendo um simulador quântico e um compilador.

Circuitos quânticos com portas do conjunto de Clifford (H, S e CNOT) podem ser eficientemente simulados em computadores clássicos [Aaronson and Gottesman 2004] e implementados com códigos de correção de erro [Raussendorf and Harrington 2007]. Contudo, para alcançar a universalidade[2], é necessária uma porta fora do conjunto de Clifford, como a porta T.

[1]Plataforma Ket disponível em: https://gitlab.com/quantum-ket/ket

[2]No contexto da computação quântica, universalidade refere-se à capacidade de realizar qualquer computação fisicamente possível [Deutsch et al. 1995]

Minimizar o número de portas T é crucial, pois elas não podem ser eficientemente simuladas e sua implementação com códigos de correção de erro requer muito mais recursos [Campbell et al. 2017] [Gidney and Fowler 2019]. Embora existam outras métricas de otimização, a redução das portas T é a mais impactante para algoritmos de correções de erro, sendo a otimização mais impactante dada as perspectivas futuras da computação quântica.

2. Cálculo-ZX

O cálculo-ZX é uma linguagem gráfica de representação de circuitos quânticos, apresentada inicialmente em [Coecke and Duncan 2011]. Nos diagramas-ZX, os componentes fundamentais são *aranhas*, *pernas* e *fase*. As aranhas são mapas lineares que tem uma quantidade de pernas de entrada e de saída, elas têm duas variações: aranhas X, representadas em vermelho e aranhas Z, em verde. Ambas estão representadas na Figura 1. A aranha Z é definida pelas autobases da matriz Pauli Z, sendo $|0\rangle$ e $|1\rangle$. Da mesma forma, a aranha X é definida pelas autobases da matriz Pauli X, $|-\rangle$ e $|+\rangle$.

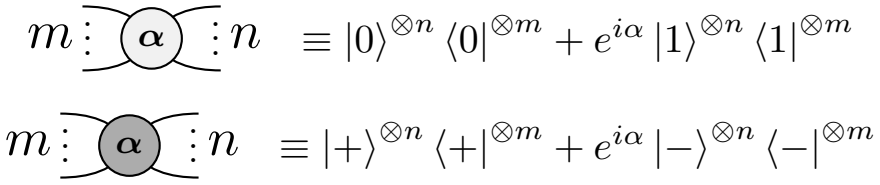


Figure 1. Definição e notação de aranhas Z e X. M é o número de entradas e N é o número de saídas da aranha.

Dessa forma, as aranhas representam estados, operações unitárias, isometrias e projeções nas bases $|0\rangle, |1\rangle$ e $|+\rangle, |-\rangle$. O tipo gerado é determinado pela quantidade de pernas na aranha, por exemplo, aranhas de estado tem apenas uma perna de saída, aranhas de projeção tem apenas uma perna de entrada, ambas representadas, e as aranhas que representam portas lógicas, as principais desse trabalho, tem exatamente uma entrada e uma saída.

As aranhas que representam portas lógicas podem facilmente ser mapeadas para portas de circuitos quânticos reais, quando equivalentes. Na Figura (2) estão os mapeamentos de portas de 1-qubit para uma única aranha. Também podemos unir e separar fases de aranhas da mesma cor para chegar nas portas conhecidas pelo mapeamento. Essas conversões, no entanto, não servem para todas as portas lógicas quânticas, pois não consideram as que operam em mais de uma base e nem portas de mais de um qubit. Para fazer isso são necessárias regras de composição.

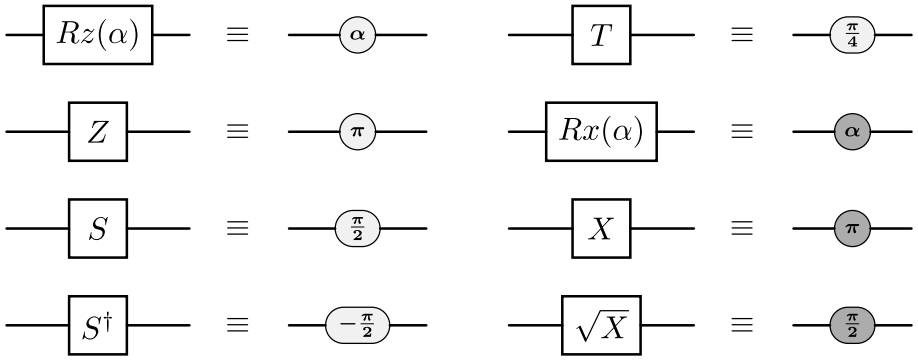


Figure 2. Mapeamento de portas de 1-qubit para aranhas Z e X.

Diagramas-ZX permitem composições em série e em paralelo. As composições em série correspondem a multiplicação matricial e envolvem a combinação das linhas da matriz de uma porta lógica com as colunas da outra. A composição em série permite criar portas lógicas que operam em mais de uma base, como a porta de Hadamard. Para obter portas de 2-qubits são utilizadas composições em paralelo, que são feitas utilizando produtos tensoriais.

O cálculo-ZX permite regras de reescrita, que são úteis para otimizações de circuito. É possível unir e separar aranhas da mesma cor, como na Figura 3 e trocar a cor das aranhas, como na Figura 4.

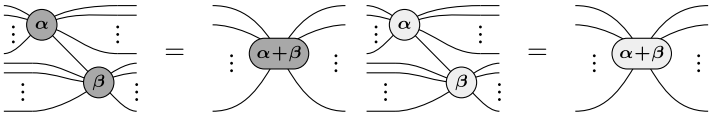


Figure 3. Regras de reescrita de fusão de aranhas. Fonte: [Commons 2019a] e [Commons 2019b]

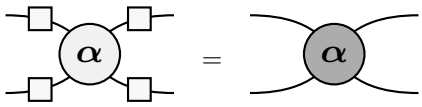


Figure 4. Aplicação da regra de troca de cor.

Vale acrescentar que nem todo Diagrama-ZX tem um circuito quântico equivalente, apesar de todo circuito poder ser representado como Diagrama-ZX, por ter a propriedade de universalidade. A transformação de um diagrama para circuito é chamada de extração. Para garantirmos essa propriedade, utilizamos o *generalised flow* [Browne et al. 2007], porque a existência desse flow garante a extraibilidade no algoritmo utilizado.

3. Otimização

Para fazer a otimização, circuitos são transformados em um grafo de estados, que tem a propriedade de *generalised flow* e, em seguida, são realizadas operações de otimização

que garantem a existência dessa propriedade, mas não necessariamente o preservam (podem criar um *generalised flow* novo no lugar do original). Para fazer a transformação em grafo de estados, todas as aranhas X são transformadas em Z pela regra de troca de cor da Figura 4 e são usadas as regras da Figura 5 para remover loops indesejados do grafo.

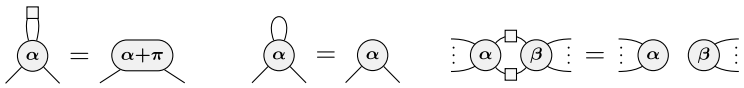


Figure 5. Regras de reescrita usadas para obter o grafo de estados

Com isso feito, são aplicadas em loop três regras de otimização que preservam a propriedade de *generalised flow*, a execução acaba apenas quando nenhuma das três regras pode ser executada. A primeira regra, da Figura 6, remove aranhas cuja fase é um múltiplo impar de $\frac{\pi}{2}$. Para fazer essa remoção, é necessário ajustar as fases dos vizinhos e alterar as suas conexões utilizando complemento de grafos, nele, todos os vizinhos do vértice removido que não tinham conexão entre si passam a ter e os que tinham passam a não ter conexão.

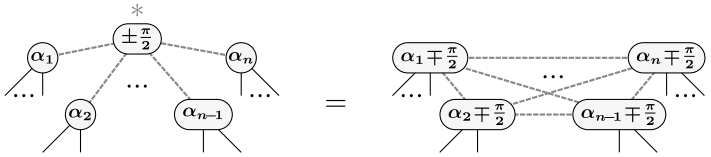


Figure 6. Regras de otimização baseada em complemento.

Para remover pares adjacentes de aranhas cuja fase é múltipla de π, utilizamos a regra da Figura 7. Considerando os vértices demarcados para exclusão como a e j, para aplicar essa remoção, identificamos três grupos de vértices, os vizinhos exclusivos de a, os exclusivos de b e os em comum, ajustamos as fases conforme o grupo a qual o vértice pertence e por fim aplicamos uma operação de pivô entre os dois vértices, feita fazendo o complemento dos vizinhos exclusivos de a com os de b, depois fazendo o complemento dos vizinhos exclusivos de a com os em comum, e repetindo isso para b. Após isso os dois vértices podem ser removidos.

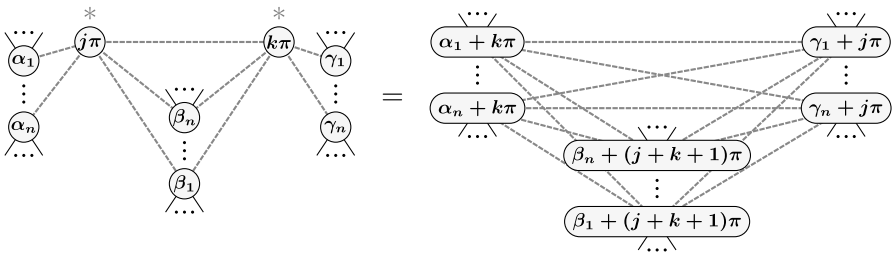


Figure 7. Regras de otimização do pivô.

As regras de otimização não podem agir sobre aranhas marcadas como de entrada ou saída do circuito. No entanto, essas aranhas podem ter outras conexões que podem ser

removidas. Para trazer essas conexões para dentro da área de extração é utilizada a regra da Figura 8

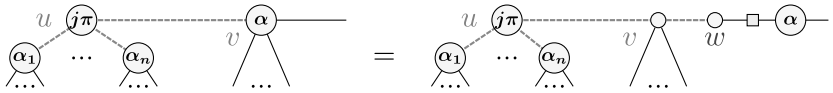


Figure 8. Regras auxiliar para expor conexões fora do alcance de otimização.

4. Extração

Para extrair um circuito quântico válido a partir do grafo de estados é utilizado uma variante do algoritmo proposto em [Backens et al. 2021] e adaptado em [Duncan et al. 2020]. Esse algoritmo de extração faz a síntese do circuito da direita para esquerda, onde estratégia é a avançar uma fronteira pelo diagrama, que separa a parte extraída da não-extraída. A garantida de parada na execução do algoritmo é dada pela propriedade de *generalised flow*.

1. Criação de uma fronteira na saída do circuito. A quantidade de elementos na mesma é igual à quantidade de qubits no circuito. Tudo que estiver depois da fronteira representa um circuito quântico extraído.
2. Mover a fronteira em direção à entrada do circuito. Toda aranha que fizer contato fará parte da fronteira. Este processo é repetido até alcançar a entrada.
    - Caso uma aranha pertencente a fronteira tenha apenas uma conexão com o grafo não extraído, ela é extraída e movida para a direita da fronteira. A conversão para circuito é possível ser feita pelas conversões da Figura 2 e por composições em série.
    - Caso dois vértices da fronteira estejam conectados entre si, a conexão é desfeita por uma porta CZ. O elemento CZ é considerado extraído porque pode ser convertido para circuito por mapeamento.
    - Caso existam conexões de um vértice da fronteira com outros que não fazem parte da mesma, essas conexões são desfeitas utilizando portas CNOT em etapas utilizando redução gaussiana. O elemento CNOT é considerado extraído porque pode ser convertido para circuito por mapeamento.
3. Uma vez encerrado o processo de mover a fronteira, os vértices finais presentes nela estarão permutados com as entradas. Para garantir equivalência com o circuito original, portas SWAP são utilizadas para fazer as permutações. Portas SWAP tem a mesma representação de circuitos quânticos em cálculo-ZX.

5. Implementação na Plataforma Ket

A arquitetura da plataforma Ket, atualmente, é separada em *libket*, biblioteca Rust responsável por fazer operações lógicas em processos quânticos e que coordena o simulador *kbw*, também em Rust, responsável pela execução dos processos e *Ket*, biblioteca Python que interpreta o código quântico alto nível escrito pelo desenvolvedor e os repassa para a *libket*. Nesta arquitetura a comunicação entre bibliotecas ocorre via *bindings* C, permitindo comunicação entre os binários em Rust e a API em Python.

A biblioteca *QuiZX* [3] foi escolhida por apresentar implementações em Rust dos algoritmos apresentados, permitindo uma integração mais fácil com a *libket*, escrita em Rust, e possibilitando uma execução mais rápida que as implementações presentes na outra biblioteca de cálculo-ZX disponível, o *PyZX* [Kissinger and van de Wetering 2020a], escrito em Python.

Para possibilitar que a biblioteca obtenha os circuitos quânticos, transformá-los em diagramas-ZX e fazer as operações necessárias de otimização e extração, é preciso fornecer o circuito mediante uma representação intermediária. Para isso, foi escolhida a linguagem Open Quantum Assembly (Open QASM), feita para fazer a descrição de código quântico executável em computadores quânticos da IBM [Cross et al. 2017].

Levando isso em conta, o fluxograma da Figura 9 apresenta a proposta de alteração da arquitetura da Plataforma Ket para suportar a otimização de circuito. Nela, é apresentada o processamento de um código quântico na linguagem Ket até a sua otimização. Na Figura, é possível observar a interação entre o front-end da linguagem e a biblioteca *libket* e, em seguida, sua interação com o otimizador por meio da representação intermediária QASM.

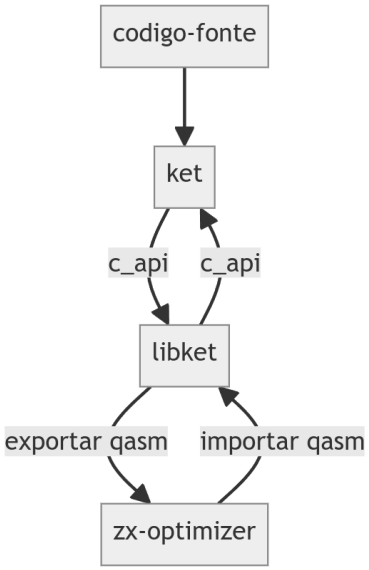


Figure 9. Fluxograma de otimização na plataforma Ket.

5.1. Transpilador QASM

O transpilador utiliza a fila de instruções do processo quântico, que contém as portas a serem aplicadas, operações de que devem ser aplicadas e alocações e liberações de recursos quânticos. Essas instruções são traduzidas para o código QASM equivalente, no entanto, há algumas operações que são exclusivas do Ket, como *dump* (que dá informações de *debug* sobre o estado do qubit) e *sample* (que mede um qubit n vezes e apresenta a distribuição dos resultados).

Para estas instruções, são utilizadas instruções opacas, que permite a adição de operações não especificadas pelo QASM. Essa flexibilidade permite então que essas

[3] Biblioteca QuiZX está disponível em: https://github.com/Quantomatic/quizx

operações sejam preservadas e permitem manter as operações originais declaradas no Ket.

Para adicionar importação QASM, foi necessário implementar um importador de código quântico. Este importador necessita ler código QASM e traduzi-lo para instruções na *libket*, para isso, foi utilizada a biblioteca em Rust *openqasm*, sendo um parser para esse tipo de código. Quando o parser detecta alguma instrução QASM, uma chamada equivalente é feita na *libket* e um processo quântico é construído a partir das instruções.

Inicialmente, é feito um mapeamento das portas suportadas tanto pelo QASM e pela *libket*, no entanto, nem todas são, tornando necessário fazer decomposições. A porta unitária genérica de 1-qubit do QASM não é suportada pela plataforma Ket, no entanto, é possível obter essa porta a partir de decomposição com um conjunto de portas já implementadas na plataforma. Para obter essa decomposição são utilizadas as portas R_Z e R_Y e aplica se a equivalência descrita nas equações abaixo:

$$R_z(\lambda) = \begin{pmatrix} e^{-i(\lambda)/2}cos(\theta/2) & 0 \\ 0 & e^{i(\lambda)/2}cos(\theta/2) \end{pmatrix} \tag{1}$$

$$R_y(\theta) = \begin{pmatrix} cos(\theta/2) & sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{pmatrix} \tag{2}$$

$$U(\theta,\phi,\lambda) := R_z(\phi)R_y(\theta)R_z(\lambda) = \begin{pmatrix} e^{-i(\phi+\lambda)/2}cos(\theta/2) & -e^{-i(\phi-\lambda)/2}sin(\theta/2) \\ e^{i(\phi-\lambda)/2}sin(\theta/2) & e^{i(\phi+\lambda)/2}cos(\theta/2) \end{pmatrix} \tag{3}$$

5.2. Otimização e restrições

Com o suporte a otimização implementada, o usuário final consegue utilizar essas funcionalidades apenas configurando o seu processo quântico. Toda a lógica de otimização é extraída do usuário final, tirando vantagem do Ket ser uma linguagem de alto nível.

Listing 1. Exemplo de configuração de processo Ket com otimização

```
p = Process(execution="batch", optimize=true)
a, b = p.alloc(2)

CNOT(H(a), b) # Bell state preparation

d = dump(a + b)

p.execute()
```

Para preservar operações sensíveis ao local de execução, como uma medida ou operação de debug de estado, o otimizador separa automaticamente o código quântico em blocos menores para otimizar. Isso diminui a eficácia da otimização e para evitar quebras desnecessárias do circuito foram adicionadas restrições no usuário. Essas restrições visam manter a circuito em um único bloco e delegar as medidas para o final do bloco, para não precisar fazer a quebra.

6. Resultados

Para analisar a eficácia do algoritmo de otimização, utilizamos duas abordagens: a primeira envolve circuitos gerados aleatoriamente com diferentes proporções de portas T; a segunda considera circuitos quânticos relevantes para a computação quântica, como somadores, oráculos de Grover e outros circuitos quânticos gerados aleatoriamente. Esses circuitos foram obtidos do conjunto de testes utilizado para medir o desempenho da biblioteca PyZX [Kissinger and van de Wetering 2020b]. Além disso, verificamos a equivalência dos circuitos originais com os otimizados nos dois conjuntos apresentados. Ambos os conjuntos de circuitos estão disponíveis no Github do projeto [4].

O desempenho da implementação do algoritmo na plataforma Ket foi diretamente comparado com a biblioteca Python *PyZX*, por ser a mais completa em implementações relacionadas ao cálculo-ZX. Esta biblioteca abrange o algoritmo de otimização apresentado e rotinas de otimização mencionadas, mas não exploradas neste trabalho, como a teletransportação de fase [Kissinger and van de Wetering 2020c]. Com isso em mente, além da comparação direta entre a implementação dos algoritmos (otimização por grafos), foi feita uma comparação com a rotina completa de otimização, que além do algoritmo apresentado, utiliza a teletransportação de fase.

O conjunto de circuitos gerado aleatoriamente é formado por circuitos compostos por portas Clifford+T colocadas aleatoriamente. A implementação utilizada é da biblioteca *PyZX* onde a geração do circuito é feita por meio de um sorteio de portas onde todas têm a mesma probabilidade de serem escolhidas em cada inserção. Os parâmetros utilizados para a geração do circuito foram dez qubits e mil portas lógicas. As portas utilizadas foram T, S, HSH ($= e^{\pi/4} Rx(\frac{\pi}{2})$) e CNOT.

Já o conjunto de circuitos de aplicações reais da computação quântica analisados nesse *benchmark* podem ser categorizados em três tipos principais. Primeiramente, temos os componentes do algoritmo de fatoração de Shor, que incluem circuitos da Transformada de Fourier Quântica (QFT), circuitos somadores da biblioteca aritmética do Quipper e somadores baseados em QFT, esses circuitos são cruciais para a implementação do algoritmo de Shor. Em segundo lugar, há os circuitos aritméticos e de Toffoli, fundamentais para operações lógicas e de aritmética em computação quântica. Por último, encontram-se os códigos de Hamming, utilizados para a correção de erro. Para que as portas usadas pelos algoritmos sejam consistentes com as da otimização, o circuito teve suas portas expressas em termos de fases X e Z, Hadamards, CNOT e CZ.

Para ambos os conjuntos, foi possível observar uma redução no número de portas T, como é possível observar na Figura 10 e 11 (que são dados obtidos de circuitos aleatórios e do circuito quântico notável GF2, respectivamente). No entanto, a abordagem utilizada causou um aumento significativo no tamanho do circuito, como consequência do processo de extração. É possível observar este aumento na Figura 12 e 13 (circuitos aleatórios e circuito quântico notável Toffoli, respectivamente), além disso, pode-se notar que a implementação da biblioteca *PyZX* não sofre desse problema. Este comportamento se dá porque a biblioteca adota uma estratégia que não é adotada pelo Ket, que é separar partes do circuito em circuitos de Clifford e sintetizá-los utilizando o algoritmo

[4]Conjunto de circuitos de testes está disponível em:
https://github.com/gabsilvcar/ket/tree/master/benchmarks/circuits.

de [Patel et al. 2004], apenas utilizando o algoritmo de extração de [Backens et al. 2021] onde é necessário. Isto causa uma diferença substancial no número total de portas do circuito, como é possível observar no gráfico apresentado. Além disso, o PyZX incorpora uma série de pequenas otimizações de portas CNOT, CZ e H que têm um impacto no circuito final.

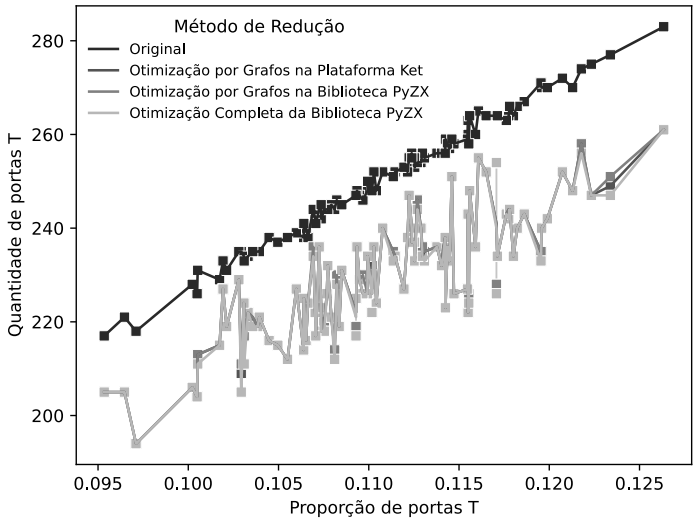


Figure 10. Comparação de diferentes métodos de otimização de circuito em relação a quantidade de portas T em um conjunto de circuitos gerados aleatoriamente.

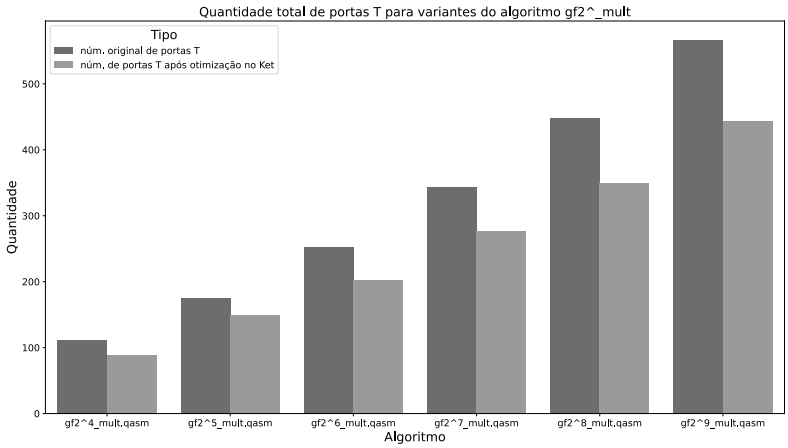


Figure 11. Comparação de número de portas T para circuito o quântico GF2.

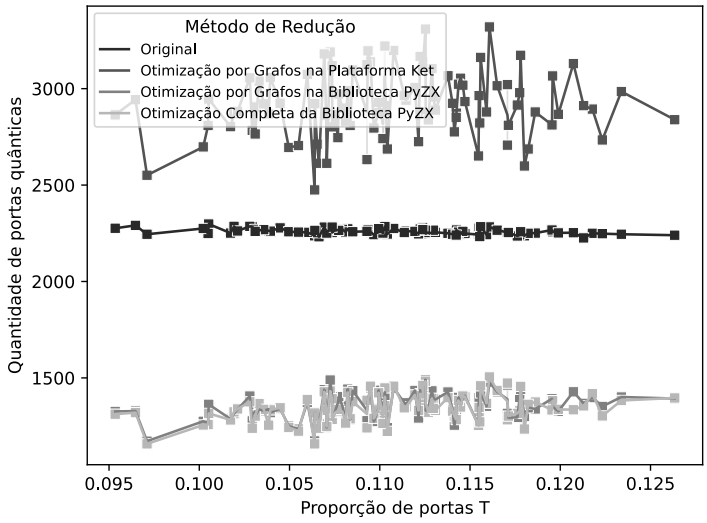


Figure 12. Comparação de diferentes métodos de otimização de circuito em relação à quantidade de portas num conjunto de circuitos gerados aleatoriamente.

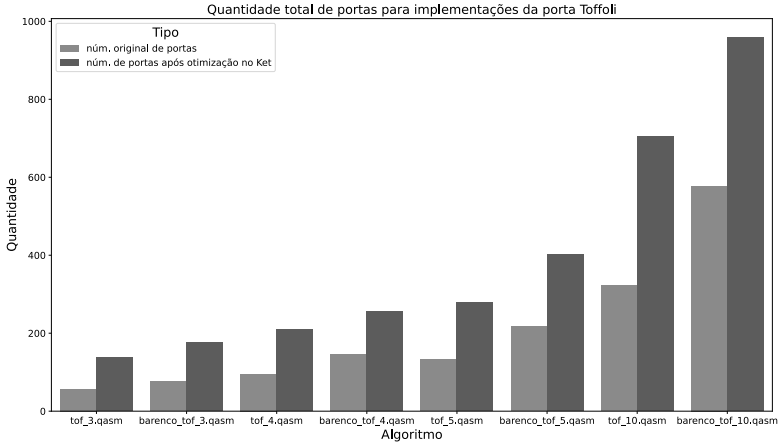


Figure 13. Comparação de número total de portas para diferentes implementações de portas Toffoli.

Outra métrica notável é o ganho de desempenho que a plataforma Ket obteve ao usar a implementação em Rust do *quizx*, apresentando uma diferença de ordem de magnitude no tempo de execução dos algoritmos feitos em Python. Esta diferença pode ser observada no gráfico da Figura 14.

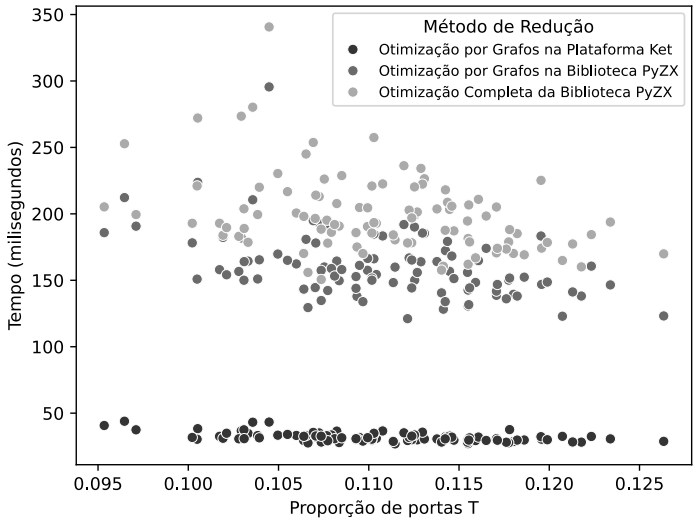


Figure 14. Comparação de diferentes métodos de otimização de circuito em relação a tempo de otimização e extração num conjunto de circuitos gerados aleatoriamente.

7. Conclusões

Uma contribuição desta pesquisa é um *fork* da Plataforma Ket com suporte para cálculo-ZX e o algoritmo de otimização de [Duncan et al. 2020]. O repositório está disponível em https://github.com/gabsilvcar/ket. Todos os resultados obtidos por benchmarks e parâmetros para obtenção dos circuitos gerados aleatoriamente também estão inclusos.

Dadas as perspectivas futuras da computação quântica tolerante a erros, este trabalho cumpriu seu objetivo de identificar o estado da arte para algoritmos e métricas de otimização. Foi apresentado que as portas T ditam desproporcionalmente a quantidade de recursos gastos pela correção de erros e sua redução foi o foco principal do trabalho, mas outras formas de otimização foram apresentadas.

O trabalho apresentou o cálculo-ZX para fazer otimizações, explorando suas nuances de representação, conversão de circuitos quânticos e, subsequentemente, extração dos diagramas. Sua integração com a Plataforma Ket foi obtida a partir da biblioteca *quizx* de cálculo-ZX que se mostrou adequada para o caso de desenvolvimento de software do Ket, porque não atrela a representação interna da plataforma com a representação-ZX, permitindo que o desenvolvimento futuro e expansão de funcionalidades sejam feitas separadamente. A comunicação da mesma com a *libket* (*backend* da plataforma) ocorre mediante uma linguagem de descrição de circuitos intermediária, o QASM.

Considerando os resultados obtidos, a otimização de portas T na Plataforma Ket foi um sucesso, apresentando ganhos para circuitos aleatórios e circuitos de algoritmos de aplicações reais da computação quântica. A implementação obteve também um ótimo resultado de desempenho, onde o processo completo de otimização (*parse* do assembly quântico, conversão, otimização, extração) é feito duas ordens de grandeza mais rápido do que implementações em Python. A validade dos resultados foi averiguada utilizando as técnicas de equivalência de circuito proporcionadas pelo cálculo-ZX.

Foram avaliadas também as limitações dessa técnica de otimização, em específico os prejuízos que ela pode trazer para a profundidade do circuito, quantidade total de portas e número de portas T. Foi observado uma grande perda nessas métricas, mas elas podem ser amenizadas com outras técnicas de otimizações apresentadas no texto, em especial, a extração pode ser feita de maneira mais eficiente para casos gerais implementando algoritmos de otimização de portas CZ e CNOT, reduzindo o número de portas de 2-qubits, além de cancelamentos simples de portas lógicas para diminuir o número total de portas no circuito.

Para casos nos quais o circuito após a otimização é um circuito de Clifford, ou seja, todas as portas T foram removidas, é possível obter uma extração assintótica para fazer a síntese desse circuito usando o algoritmo proposto por [Patel et al. 2004], podendo ser implementado na Plataforma Ket no futuro. Com isso, seria possível alterar a estratégia de otimização e alinhar ela com o que é feito na biblioteca *PyZX*, onde o algoritmo de extração é usado apenas nas seções do circuito que não são circuitos de Clifford.

É possível também integrar outros algoritmos de otimização de cálculo-ZX, como de [Staudacher et al. 2023] para redução de portas CZ e o utilizar o conceito de teleporte de fase de [Kissinger and van de Wetering 2020c] para reduzir mais o número de portas T. O segundo algoritmo não necessita de um algoritmo de extração após ser executado, ou seja, não adicionará mais portas no processo de extração.

Em relação ao cálculo-ZX, ainda não existem métodos para utilizarem qubits auxiliares, também chamadas de *ancillas*, para otimizações. O uso de ancillas poderia diminuir o número de portas utilizadas e permitir a sinteze de circuitos quânticos mais eficientes.

No algoritmo de extração apresentado, o processo de síntese de CNOTs não leva em conta a topologia dos qubits restritas adotadas na arquitetura de computadores quânticos. Essas topologias têm limitação na conectividade entre os qubits em um processador quântico. Em vez de qualquer qubit poder interagir diretamente com qualquer outro, as interações são permitidas apenas entre qubits que estão próximos ou conectados de maneira específica. É possível adicionar restrições na etapa de limitação gaussiana usando estratégias de Árvores de Steiner como apresentado por [Nash et al. 2020] e [Kissinger and van de Griend 2019].

References

Aaronson, S. and Gottesman, D. (2004). Improved simulation of stabilizer circuits. *Physical Review A*, 70(5).

Backens, M., Miller-Bakewell, H., de Felice, G., Lobski, L., and van de Wetering, J. (2021). There and back again: A circuit extraction tale. *Quantum*, 5:421.

Browne, D. E., Kashefi, E., Mhalla, M., and Perdrix, S. (2007). Generalized flow and determinism in measurement-based quantum computation. *New Journal of Physics*, 9(8):250.

Campbell, E. T., Terhal, B. M., and Vuillot, C. (2017). Roads towards fault-tolerant universal quantum computation. *Nature*, 549(7671):172–179.

Coecke, B. and Duncan, R. (2011). Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016.

Commons, W. (2019a). Zx-calculus green spider fusion rule.

Commons, W. (2019b). Zx-calculus red spider fusion rule.

Cross, A. W., Bishop, L. S., Smolin, J. A., and Gambetta, J. M. (2017). Open quantum assembly language.

Da Rosa, E. C. R. and De Santiago, R. (2021). Ket quantum programming. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 18(1):1–25.

Deutsch, D. E., Barenco, A., and Ekert, A. (1995). Universality in quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 449(1937):669–677.

Duncan, R., Kissinger, A., Perdrix, S., and van de Wetering, J. (2020). Graph-theoretic simplification of quantum circuits with the ZX-calculus. *Quantum*, 4:279.

Gidney, C. and Fowler, A. G. (2019). Efficient magic state factories with a catalyzed —ccz¿ to 2—t¿ transformation. *Quantum*, 3:135.

Kissinger, A. and van de Griend, A. M. (2019). Cnot circuit extraction for topologically-constrained quantum memories.

Kissinger, A. and van de Wetering, J. (2020a). Pyzx: Large scale automated diagrammatic reasoning. *Electronic Proceedings in Theoretical Computer Science*, 318:229–241.

Kissinger, A. and van de Wetering, J. (2020b). Reducing the number of non-clifford gates in quantum circuits. *Physical Review A*, 102(2).

Kissinger, A. and van de Wetering, J. (2020c). Reducing the number of non-clifford gates in quantum circuits. *Physical Review A*, 102(2).

Nash, B., Gheorghiu, V., and Mosca, M. (2020). Quantum circuit optimizations for nisq architectures. *Quantum Science and Technology*, 5(2):025010.

Patel, K., Markov, I., and Hayes, J. (2004). Optimal synthesis of linear reversible circuits. *Quantum Information and Computation*, 8.

Raussendorf, R. and Harrington, J. (2007). Fault-tolerant quantum computation with high threshold in two dimensions. *Physical Review Letters*, 98(19).

Staudacher, K., Guggemos, T., Grundner-Culemann, S., and Gehrke, W. (2023). Reducing 2-qubit gate count for zx-calculus based quantum circuit optimization. *Electronic Proceedings in Theoretical Computer Science*, 394:29–45.