Busca em profundidade em paralelo

José Victor Rocha de Alencar, Thiago Vieira Camara
Departamento de Ciência da Computação, Universidade Federal de Roraima
Boa Vista, Brasil
thiagocamarasp@gmail.com
victorbvbr@gmail.com

Abstract—Parallel Depth-First Search (DFS) is an advanced optimization of the traditional DFS algorithm, specifically designed to enhance performance when traversing large-scale graphs by leveraging the power of parallel processing. While classical DFS systematically explores a graph by delving as deeply as possible along each branch before backtracking, its inherently sequential nature can create significant bottlenecks, particularly in high-dimensional graphs or those with a vast number of nodes and edges.

In contrast, the parallel version of DFS employs multiple threads or processes to simultaneously explore different parts of the graph. This approach effectively distributes the computational workload across available processing units, leading to a marked reduction in overall traversal time and significantly improving efficiency. However, the implementation of parallel DFS is not without its challenges. Key issues include load balancing among threads to ensure even distribution of work, synchronization to prevent data inconsistencies, and strategies to avoid redundant explorations of the same paths, which can undermine the benefits of parallelization.

Despite these complexities, parallel DFS offers substantial advantages in various applications, particularly within distributed computing environments. Its enhanced search efficiency is particularly valuable in scenarios that require real-time data processing, such as web crawling, network analysis, and AI-driven game simulations. By reducing search times and improving responsiveness, parallel DFS not only enables the handling of larger and more complex datasets but also facilitates the development of more sophisticated algorithms in fields such as artificial intelligence, computational biology, and urban planning. This paper discusses the implementation of parallel DFS, its associated challenges, and its potential applications, demonstrating its significance in advancing the efficiency of graph traversal methodologies in modern computing.

I. INTRODUÇÃO

A busca em profundidade (Depth-First Search - DFS) é um dos algoritmos fundamentais para o percorrimento de estruturas de dados, como árvores e grafos. Sua abordagem sistemática começa explorando um nó inicial e segue por um caminho único até atingir o limite possível, retornando então para investigar outros caminhos ainda não explorados. Essa técnica é amplamente utilizada em diversas áreas, especialmente aquelas que envolvem estruturas interconectadas e grandes espaços de busca, como resolução de quebra-cabeças, busca de soluções em jogos, análise de redes complexas, e até mesmo na navegação de sistemas distribuídos.

O DFS é particularmente eficiente em cenários onde é necessário percorrer profundamente um conjunto de possibilidades antes de retroceder e avaliar alternativas. Ele tem aplicações notáveis em problemas de otimização e em situações que requerem decisões rápidas baseadas em uma exploração exaustiva, sendo essencial em áreas que vão desde a inteligência artificial até o planejamento urbano. No entanto, sua versão sequencial pode se tornar ineficiente ao lidar com grafos muito grandes ou com ambientes dinâmicos que exigem respostas rápidas.

Na era da computação moderna, onde sistemas multinúcleo e arquiteturas paralelas se tornaram comuns, a necessidade de algoritmos capazes de aproveitar esse poder de processamento é cada vez mais evidente. Nesse contexto, o DFS também encontra desafios. O algoritmo, na sua forma clássica, não é otimizado para ambientes paralelos, pois a exploração de um caminho em profundidade não distribui eficientemente a carga de trabalho entre múltiplos núcleos. A falta de balanceamento entre as threads e os conflitos de acesso a recursos compartilhados são alguns dos fatores que limitam sua aplicação em sistemas multicore.

Em resposta a esses desafios, pesquisadores têm se dedicado a aprimorar o DFS por meio de técnicas de paralelismo, buscando explorar o potencial dos processadores modernos. A introdução de multithreading permite que múltiplos caminhos de um grafo sejam percorridos simultaneamente, distribuindo a carga de trabalho de forma mais igual entre os núcleos e, com isso, reduzindo significativamente o tempo de execução do algoritmo. Esse conceito de busca em profundidade paralela se torna especialmente relevante em aplicações que demandam alta performance, como simulações de grandes redes, análises de big data e sistemas de navegação em tempo real.

Neste projeto, exploramos a aplicação da versão paralelizada do DFS para resolver um problema de grande relevância urbana: a busca de rotas alternativas entre dois pontos em uma cidade, levando em consideração fatores como congestionamentos de tráfego, interrupções e mudanças dinâmicas nas condições de mobilidade. A

proposta envolve a utilização de múltiplos núcleos de processamento para acelerar a busca por rotas otimizadas, de forma que o sistema possa responder rapidamente às mudanças nas condições da cidade, proporcionando uma solução eficaz para os desafios de transporte em áreas urbanas complexas e densamente povoadas. Ao aplicar o DFS paralelizado, espera-se não apenas reduzir significativamente o tempo de resposta, mas também melhorar a qualidade das rotas sugeridas, tornando o sistema mais ágil, eficiente e adaptável às condições variáveis do tráfego urbano.

II. DESENVOLVIMENTO DA SOLUÇÃO

Nesta seção, detalharemos o processo de criação do grafo utilizado para mapear e sinalizar os pontos de interesse, além da implementação da paralelização do algoritmo de busca em profundidade (DFS) visando a otimização de rotas.

A. Criação do Grafo

O grafo, que representa a rede de rotas e pontos turísticos de interesse, foi construído utilizando um código personalizado que permite a adição manual de locais específicos. Cada ponto turístico é tratado como um nó no grafo, enquanto as rotas que conectam esses pontos formam as arestas. Para otimizar essas rotas, utilizamos a API Graph Hopper, que se baseia em dados do OpenStreetMap, uma fonte de dados geográficos colaborativa e amplamente utilizada para tarefas de navegação e planejamento de trajetos. A Graph Hopper é capaz de calcular rotas eficientes entre diferentes pontos, considerando variáveis como distâncias e tempos de viagem.

Entretanto, devido a limitações da API, foi possível incluir apenas dez pontos de interesse no modelo final. Essa restrição impôs um desafio na representação da rede completa de pontos turísticos, o que exigiu um planejamento cuidadoso para selecionar os locais mais relevantes dentro da área de estudo. A escolha desses pontos foi feita com base em fatores como popularidade, acessibilidade e proximidade com outros locais de interesse, garantindo que o grafo resultante representasse de maneira adequada o cenário urbano para o qual o sistema foi projetado.

Após a criação do grafo com os dez pontos selecionados, procedemos à geração de duas matrizes de adjacência. Cada matriz foi configurada para representar diferentes meios de transporte – uma para rotas otimizadas para pedestres e outra para veículos. As matrizes de adjacência são fundamentais, pois armazenam as distâncias ou tempos de viagem entre cada par de pontos no grafo. Posteriormente, essas matrizes foram combinadas em um único arquivo, de forma a centralizar os dados necessários para o cálculo das rotas. Esse arquivo consolidado foi então integrado ao código principal do sistema, escrito em C, que realiza o processamento e análise dos trajetos. A escolha da linguagem C se deve ao seu desempenho elevado em operações de baixo nível, algo essencial para o tratamento de grandes volumes de dados em tempo real.

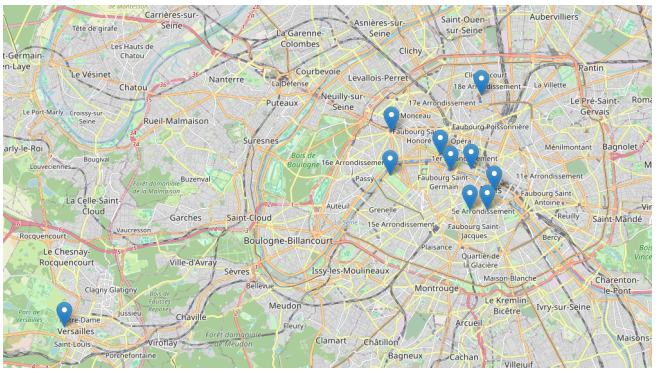

Fig.1 Os dez pontos selecionados para o DFS percorrer e encontrar a melhor rota dependendo do ponto inicial.

B. Paralelização do DFS

A busca em profundidade (DFS) é um algoritmo fundamental na exploração de grafos, amplamente utilizado para encontrar as melhores rotas em diversas aplicações, incluindo sistemas de navegação, jogos e análise de redes. A capacidade da DFS de percorrer grafos de maneira eficiente a torna uma escolha popular para resolver problemas complexos relacionados à conectividade e ao caminho mais curto. No contexto de sistemas de navegação, por exemplo, a DFS pode ser utilizada para identificar rotas alternativas e desvios em tempo real, contribuindo para a otimização de trajetos e a melhoria da experiência do usuário.

Para atender às crescentes demandas de desempenho em sistemas que operam em arquiteturas multicore, a DFS foi otimizada por meio de técnicas de paralelização. O objetivo é maximizar a eficiência da execução do algoritmo em processadores com múltiplos núcleos, que se tornaram comuns em hardware moderno. A biblioteca OpenMP foi escolhida como base para essa paralelização devido à sua comprovada eficiência em programação paralela e sua ampla compatibilidade com linguagens como C e C++. OpenMP fornece um conjunto de diretrizes e ferramentas que facilitam a criação de múltiplas threads, permitindo que diferentes partes do algoritmo sejam executadas simultaneamente. Essa capacidade de processamento paralelo aproveita todo o potencial dos processadores multinúcleo, resultando em um desempenho superior em comparação à execução sequencial.

Na implementação da DFS paralelizada, o algoritmo foi adaptado para gerar várias threads, cada uma encarregada de explorar uma parte específica do grafo. Essa abordagem permite que cada thread percorra um ramo distinto do grafo, processando diferentes combinações de caminhos em paralelo. Ao dividir a carga de trabalho entre múltiplos processadores, o tempo total de execução da busca é significativamente reduzido. O DFS tradicional, quando executado de forma sequencial, limita-se a explorar um único caminho de cada vez, o que pode se tornar um processo lento, especialmente em grafos grandes e

complexos, onde o número de rotas alternativas pode ser elevado. A introdução do multithreading não apenas acelera a descoberta dos melhores caminhos, mas também evita redundâncias na busca, uma vez que diferentes threads podem trabalhar simultaneamente em áreas distintas do grafo.

O uso de OpenMP simplifica significativamente a paralelização da DFS, permitindo que blocos de código sejam facilmente paralelizados com a adição de diretrizes específicas. Por exemplo, ao marcar um loop ou uma função com diretivas OpenMP, o compilador distribui automaticamente o trabalho entre as threads disponíveis, garantindo que cada núcleo da CPU seja utilizado de maneira eficiente. Essa automação no gerenciamento de threads não apenas reduz o esforço do programador, mas também assegura que o DFS paralelizado utilize os recursos de hardware de forma otimizada. Essa abordagem permite que os desenvolvedores se concentrem na lógica do algoritmo, enquanto a biblioteca cuida da complexidade da paralelização.

Além disso, a implementação da DFS paralelizada não só melhora a performance, mas também abre novas possibilidades para aplicações em tempo real, onde a rapidez na resposta é crítica. Em um sistema de navegação, por exemplo, ser capaz de encontrar rapidamente rotas alternativas em resposta a mudanças dinâmicas no tráfego pode ajudar a evitar congestionamentos e oferecer soluções mais eficientes aos usuários. Essa capacidade de adaptação em tempo real é um diferencial importante em um mundo onde a eficiência é frequentemente um fator determinante na satisfação do cliente.

Assim, a combinação da DFS com técnicas de paralelização, como a OpenMP, representa um avanço significativo na exploração de grafos, permitindo que algoritmos complexos sejam executados de maneira mais ágil e eficaz em sistemas modernos. Essa abordagem não apenas proporciona uma melhoria substancial na velocidade de busca, mas também amplia o escopo de aplicações práticas, possibilitando o desenvolvimento de soluções inovadoras em diversos setores, desde transporte até redes sociais e análises de grandes dados. Com a evolução contínua da tecnologia e o aumento da complexidade das redes que precisamos explorar, a utilização de algoritmos como a DFS paralelizada se tornará cada vez mais relevante e necessária.

C. Complexidade

A DFS paralelizada tem uma complexidade de tempo de $O(2^V)$, onde V representa o número de vértices. Essa eficiência a torna uma escolha ideal para a exploração de grafos, independentemente de serem densos ou esparsos. A capacidade de realizar a busca de forma paralela significa que a DFS pode dividir o trabalho entre múltiplos núcleos de processamento, reduzindo significativamente o tempo total necessário para explorar grafos de grande escala.

O artigo "An Efficient Parallel Depth-First Search" (Acar, Blelloch, & Shun, 2004) investiga as técnicas utilizadas para otimizar a DFS em arquiteturas paralelas, abordando questões como a sincronização entre threads e a divisão eficaz de tarefas. Os autores propõem estratégias que permitem que diferentes partes do grafo sejam exploradas simultaneamente, minimizando conflitos e maximizando a utilização de recursos computacionais. Essa abordagem paralelizada não apenas melhora o desempenho em relação à DFS sequencial, mas também é particularmente valiosa em aplicações que exigem processamento em tempo real, como na análise de redes sociais e na exploração de dados em grandes bases de dados.

Além disso, o livro *Introdução aos Algoritmos* destaca outras aplicações da DFS que demonstram sua versatilidade. Uma dessas aplicações é a identificação de componentes fortemente conectados em um grafo dirigido, onde a DFS pode ser utilizada para descobrir subgrafos onde qualquer par de vértices é acessível um ao outro. Isso é fundamental em diversas áreas, como a análise de redes de comunicação e a modelagem de interações em sistemas complexos.

Outro algoritmo importante mencionado é o de Tarjan, que utiliza a DFS para encontrar componentes fortemente conectados em tempo linear. A combinação da DFS com esse algoritmo permite não apenas a identificação eficiente de subgrafos, mas também a aplicação de técnicas de busca para resolver problemas mais complexos em teoria dos grafos. Essas características tornam a DFS e suas variantes ferramentas essenciais em ciência da computação, matemática discreta e em várias aplicações práticas na indústria.

D. Resultados

Durante os experimentos, iniciamos com a implementação de um algoritmo de busca em profundidade (DFS) tradicional para explorar o grafo de pontos turísticos, que serviu como um ponto de partida crucial para medir e comparar o desempenho com a versão paralelizada. Essa versão sequencial do DFS revelou um padrão preocupante nos testes iniciais: um aumento exponencial no tempo de execução à medida que o número de nós no grafo crescia. Esse comportamento era esperado, dado que, em grafos mais complexos, a necessidade de explorar todos os caminhos possíveis leva a um aumento significativo no tempo de processamento. Essa característica da DFS, que verifica cada vértice e aresta, resulta em um tempo de execução que se torna insustentável em cenários práticos, especialmente quando se busca otimizar rotas em uma rede de transporte urbana.

Apesar do crescimento exponencial no tempo de execução, observamos que o uso de memória do algoritmo sequencial

permaneceu relativamente constante. Essa constância se deve ao fato de que, por natureza, a DFS utiliza uma pilha para armazenar apenas o caminho atual de exploração, independentemente do número total de nós no grafo. Portanto, embora o tempo de execução se tornasse um fator limitante, a eficiência em termos de memória do algoritmo sequencial se mostrava adequada para os objetivos iniciais.

Diante desses desafios, decidimos implementar uma versão paralelizada do DFS, utilizando técnicas de multithreading para aprimorar o desempenho. Para isso, o grafo foi subdividido em subgrafos menores, e cada um desses subgrafos foi atribuído a uma thread separada. Essa abordagem permitiu que cada thread executasse a busca DFS em seu subgrafo correspondente de forma independente e simultânea. Ao final do processo, os resultados de todas as threads foram combinados para identificar o caminho final ótimo. Essa técnica não apenas distribui a carga de trabalho de forma mais equilibrada entre os recursos do sistema, mas também aproveita a capacidade de processamento paralelo dos processadores modernos, resultando em ganhos significativos de desempenho.

Os resultados obtidos foram bastante promissores. A versão paralelizada do DFS apresentou uma melhoria significativa no tempo de execução em comparação com a versão sequencial. Em grafos menores, embora o ganho de desempenho fosse visível, não era tão impactante. No entanto, à medida que o tamanho do grafo aumentava, os benefícios da paralelização se tornaram mais evidentes, destacando a eficácia da abordagem adotada. À medida que mais threads eram adicionadas ao processo, o tempo de execução foi consideravelmente reduzido em relação à sua versão sequencial, demonstrando um ganho de velocidade proporcional ao número de threads utilizadas. Esse resultado não apenas valida a eficácia da implementação paralelizada, mas também ressalta o potencial de otimização em sistemas de navegação que necessitam processar rapidamente grandes volumes de dados.

Além disso, essa experiência prática destacou a importância de se considerar a arquitetura do sistema ao desenvolver algoritmos para exploração de grafos. A capacidade de dividir e conquistar, utilizando técnicas de paralelização, não apenas melhora a eficiência do algoritmo, mas também possibilita a exploração de grafos em tempo real, algo crucial em aplicações como sistemas de navegação, onde decisões rápidas são essenciais. Os resultados encorajadores obtidos durante esses experimentos abrem caminho para futuras pesquisas e melhorias na busca em profundidade, incentivando o uso de abordagens paralelizadas em diversas outras aplicações que exigem processamento eficiente e rápido de dados. Assim, o estudo não apenas contribuiu para a compreensão do comportamento da DFS em diferentes contextos, mas também estabeleceu uma base sólida para inovações em algoritmos de exploração de grafos.

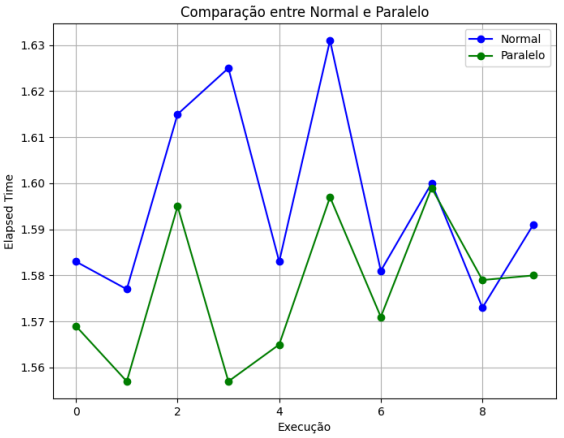

Fig.2 Representa a diferença entre o DFS normal e o paralelizado na questão do tempo

linkgithub: https://github.com/Victor-de-Alencar/Jose_Victor_Thiago_Vieira_FinalProject_AA_RR_2024

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., STEIN, C. *Introdução aos Algoritmos*. 3. ed. Rio de Janeiro: Elsevier, 2009.

Acar, M., Blelloch, G. E., & Shun, M. (2004). An Efficient Parallel Depth-First Search. In *Proceedings of the 2004 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (PPoPP '04).