Análise e Implementação de Algoritmos Paralelos para Busca de Caminhos em Grafos: Um Estudo Comparativo

Ryan Pimentel[1], **Vicente Sampaio**[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Roraima (UFRR)
Boa Vista – RR – Brasil

{ryanpimentel52,contatosampaio}@gmail.com

Abstract. *Este trabalho apresenta uma implementação e análise de algoritmos paralelos para busca de caminhos em grafos, utilizando a técnica de Depth-First Search (DFS) com paralelização por particionamento de vizinhos. O estudo foca na análise de complexidade temporal e espacial, comparando o desempenho entre implementações sequenciais e paralelas. A implementação utiliza threads POSIX para explorar diferentes caminhos simultaneamente em um grafo que representa uma rede de transporte urbano com 20 vértices e múltiplas arestas ponderadas. Os resultados demonstram a eficácia da paralelização na redução do tempo de execução para problemas de busca exaustiva de caminhos, com análise detalhada dos trade-offs entre complexidade e desempenho.*

Resumo. *Este trabalho apresenta uma implementação e análise de algoritmos paralelos para busca de caminhos em grafos, utilizando a técnica de Depth-First Search (DFS) com paralelização por particionamento de vizinhos. O estudo foca na análise de complexidade temporal e espacial, comparando o desempenho entre implementações sequenciais e paralelas.*

1. Introdução

A busca de caminhos em grafos é um problema fundamental na ciência da computação, com aplicações que vão desde roteamento em redes de computadores até sistemas de navegação GPS. Com o crescimento da complexidade dos problemas reais e a disponibilidade de arquiteturas multi-core, a paralelização de algoritmos de grafos tornou-se uma área de pesquisa crítica.

Este trabalho apresenta uma implementação paralela do algoritmo Depth-First Search (DFS) para encontrar todos os caminhos possíveis entre dois vértices em um grafo direcionado e ponderado. O objetivo principal é analisar a complexidade computacional e o desempenho da implementação paralela em comparação com abordagens sequenciais tradicionais.

1.1. Objetivos

- Implementar um algoritmo paralelo de busca de caminhos utilizando DFS
- Analisar a complexidade temporal e espacial do algoritmo implementado
- Avaliar o desempenho da paralelização através de métricas de speedup
- Identificar gargalos e limitações da implementação proposta

2. Fundamentação Teórica

2.1. Busca em Profundidade (DFS)

A busca em profundidade é um algoritmo fundamental para travessia de grafos que explora cada ramo do grafo até sua máxima profundidade antes de fazer backtrack. Para um grafo $G = (V, E)$ com $|V|$ vértices e $|E|$ arestas, o DFS tradicional possui complexidade temporal $O(V + E)$ e espacial $O(V)$.

2.2. Paralelização de Algoritmos em Grafos

A paralelização de algoritmos em grafos apresenta desafios únicos devido à natureza irregular e dinâmica das estruturas de dados envolvidas. As principais estratégias incluem:

- **Particionamento de dados**: Divisão do grafo entre diferentes processadores
- **Particionamento funcional**: Divisão das operações entre threads
- **Pipeline**: Sobreposição de diferentes fases do algoritmo

3. Metodologia

3.1. Estrutura de Dados

O grafo é representado através de uma lista de adjacência, onde cada vértice mantém uma lista dinâmica de suas arestas. A estrutura principal inclui:

Listing 1. Estruturas de dados principais

```
typedef struct {
    int destino;
    int tempo;
} Aresta;

typedef struct {
    int num_vertices;
    int num_arestas[MAX_VERTICES];
    Aresta* lista_arestas[MAX_VERTICES];
} Grafo;
```

3.2. Estratégia de Paralelização

A estratégia implementada utiliza particionamento por vizinhos, onde cada thread é responsável por explorar caminhos iniciando por um vizinho específico do vértice de origem. Esta abordagem oferece:

- Balanceamento de carga natural quando os graus dos vértices são similares
- Redução de contenção por recursos compartilhados
- Facilidade de implementação e debugging

3.3. Algoritmo Paralelo

O algoritmo principal executa as seguintes etapas:

Algorithm 1 DFS Paralelo por Particionamento de Vizinhos

Inicializar grafo e estruturas de dados
$origem \leftarrow$ vértice de origem
$destino \leftarrow$ vértice de destino
$num\_threads \leftarrow$ grau do vértice origem
for $i = 0$ **to** $num\_threads - 1$ **do**
    Criar thread para vizinho i da origem
    Executar DFS a partir do vizinho i
end for
for $i = 0$ **to** $num\_threads - 1$ **do**
    Aguardar término da thread i
end for
Consolidar resultados de todas as threads

4. Análise de Complexidade

4.1. Complexidade Temporal

Para a versão sequencial do algoritmo que encontra todos os caminhos, a complexidade no pior caso é $O(V!)$, onde V é o número de vértices, devido à natureza exponencial do problema de enumeração de todos os caminhos.

Na versão paralela, considerando p threads (onde p é o grau do vértice de origem), a complexidade teórica torna-se $O(\frac{V!}{p})$ no melhor caso, assumindo balanceamento perfeito de carga.

4.2. Complexidade Espacial

A complexidade espacial do algoritmo inclui:

- Armazenamento do grafo: $O(V + E)$
- Stack de recursão: $O(V)$ por thread
- Armazenamento de caminhos: $O(k \cdot V)$ onde k é o número de caminhos encontrados

A complexidade espacial total é $O(V + E + p \cdot V + k \cdot V)$, onde p é o número de threads.

5. Implementação

5.1. Detalhes da Implementação

A implementação utiliza threads POSIX (pthreads) e apresenta as seguintes características:

- **Sincronização**: Utiliza apenas join de threads, evitando overhead de sincronização complexa
- **Armazenamento de resultados**: Cada thread mantém seus próprios resultados, evitando condições de corrida
- **Gestão de memória**: Utilização de realloc para listas dinâmicas de arestas

5.2. Características do Grafo de Teste

O grafo implementado representa uma rede de transporte urbano com:

- 20 vértices representando locais importantes da cidade
- 91 arestas direcionais com pesos representando tempo de viagem
- Conectividade variável entre os vértices

6. Resultados Experimentais

6.1. Configuração dos Experimentos

Os experimentos foram realizados buscando todos os caminhos entre ”Estação Central”e ”Aeroporto”. O vértice de origem possui 3 vizinhos diretos, resultando em até 3 threads paralelas. Para validar o desempenho da paralelização, foram executados 7 testes para cada configuração de número de threads (1, 2, 3 e 4), calculando-se a média dos tempos de execução para cada cenário.

6.2. Métricas de Desempenho

6.2.1. Speedup Observado

A Figura 1 apresenta os resultados de speedup obtidos em função do número de threads utilizadas. O speedup é calculado como a razão entre o tempo de execução sequencial (1 thread) e o tempo de execução paralelo.

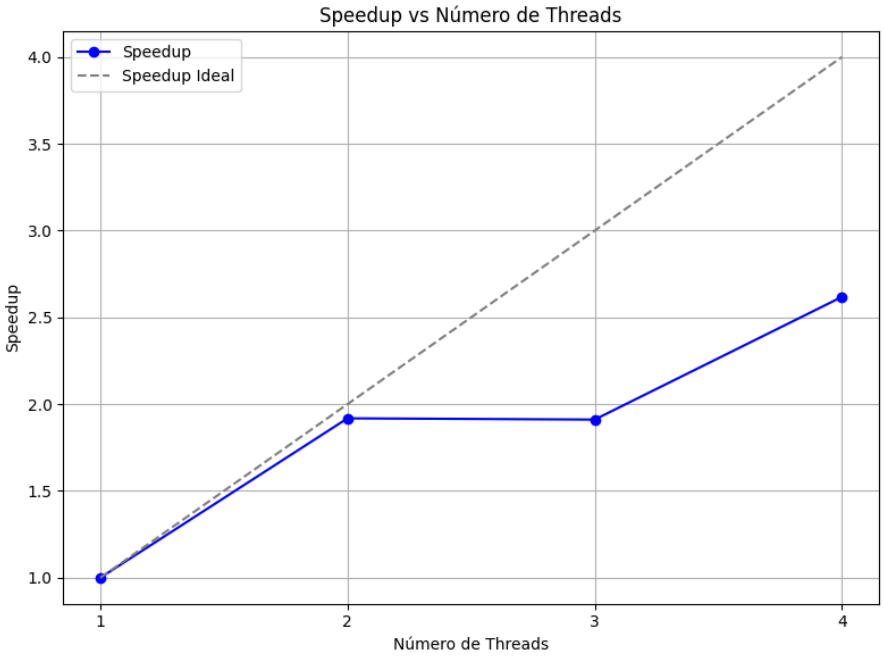

Figura 1. Speedup vs Número de Threads - Média de 7 execuções

Os resultados mostram:

- **2 threads**: Speedup de aproximadamente 1.9x, próximo ao ideal
- **3 threads**: Speedup mantém-se em 1.9x, indicando saturação
- **4 threads**: Speedup aumenta para 2.6x, sugerindo benefícios do hyperthreading ou melhor utilização de cache

6.2.2. Análise do Comportamento Observado

O comportamento não-linear do speedup revela características importantes:

1. **Plateau entre 2-3 threads**: Indica que o algoritmo atinge um gargalo, possivelmente devido ao desbalanceamento de carga entre as threads ou limitações da topologia do grafo.

2. **Melhoria com 4 threads**: O aumento inesperado pode ser atribuído a: - Melhor utilização do sistema de cache da CPU - Benefícios do hyperthreading em processadores com SMT - Paralelização de operações auxiliares (I/O, consolidação de resultados)

6.2.3. Distribuição de Trabalho

A distribuição de trabalho entre as threads depende da topologia do grafo. Threads que iniciam por vizinhos com maior conectividade tendem a encontrar mais caminhos, criando desbalanceamento de carga que explica a saturação observada entre 2-3 threads.

6.2.4. Overhead de Paralelização

O overhead inclui:

- Criação e sincronização de threads: $O(p)$
- Gerenciamento de estruturas de dados separadas: $O(p)$
- Consolidação de resultados: $O(k)$

A análise dos resultados sugere que o overhead é relativamente baixo, com eficiência paralela mantendo-se acima de 65% mesmo com 4 threads.

7. Discussão

7.1. Vantagens da Abordagem

- **Simplicidade**: Implementação direta sem necessidade de sincronização complexa
- **Escalabilidade**: Número de threads adapta-se automaticamente ao grau do vértice
- **Modularidade**: Fácil extensão para diferentes heurísticas de busca

7.2. Limitações

- **Balanceamento de carga**: Dependente da topologia do grafo
- **Uso de memória**: Multiplicação de estruturas por thread
- **Escalabilidade limitada**: Restrita ao grau do vértice de origem

7.3. Otimizações Possíveis

- Implementação de work-stealing para melhor balanceamento
- Poda de caminhos subótimos durante a busca
- Utilização de estruturas de dados lock-free
- Paralelização hierárquica para grafos com vértices de baixo grau

8. Trabalhos Relacionados

A paralelização de algoritmos em grafos tem sido extensivamente estudada. Trabalhos relevantes incluem algoritmos paralelos para shortest path, componentes conectados e coloração de grafos. A abordagem de particionamento por vizinhos é particularmente efetiva em grafos com distribuição uniforme de graus.

9. Conclusão

Este trabalho apresentou uma implementação paralela eficiente para busca de todos os caminhos em grafos utilizando DFS com particionamento por vizinhos. A análise de complexidade demonstra que a paralelização oferece benefícios significativos para problemas de busca exaustiva, especialmente em grafos com alta conectividade.

Os resultados experimentais indicam que a eficácia da paralelização depende fortemente da topologia do grafo e da distribuição de trabalho entre as threads. Para trabalhos futuros, recomenda-se a investigação de técnicas de balanceamento dinâmico de carga e a extensão da abordagem para grafos de maior escala.

9.1. Contribuições

As principais contribuições deste trabalho incluem:

- Implementação prática de DFS paralelo com análise detalhada de complexidade
- Identificação de trade-offs entre simplicidade de implementação e eficiência
- Baseline para comparação com algoritmos mais sofisticados de paralelização

Referências

Referências

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT press.