
Universidade Federal
de São João del-Rei

Departamento de Ciência da Computação

**Leonardo de Oliveira Tedeschi**
**Geovanna Vittória de Jesus Assis**

# Computação Paralela: Trabalho prático 1

São João del-Rei, novembro de 2024

# Sumário

# 1 Introdução

Neste trabalho, implementamos um algoritmo de processamento de grafos utilizando a técnica de computação paralela. A tarefa consistia em comparar a versão sequencial do algoritmo com sua versão paralelizada utilizando a biblioteca MPI (Message Passing Interface), executando os testes em grafos de diferentes tamanhos. O objetivo foi avaliar a eficiência da paralelização e comparar os tempos de execução, além de identificar possíveis melhorias.

# 2 Descrição da Aplicação

A aplicação foi desenvolvida para realizar operações em grafos representados por listas de adjacência. O algoritmo foi implementado inicialmente de forma sequencial, realizando o cálculo dos vizinhos comuns entre pares de vértices. Posteriormente, a versão paralela foi desenvolvida, onde as operações de cálculo de vizinhos comuns entre os vértices foram distribuídas entre múltiplos processos, visando melhorar o desempenho em grafos grandes.

# 3 Listagem das Rotinas

Nesta seção, são detalhadas as rotinas utilizadas no programa durante sua execução, incluindo instruções sobre como executar os programas e os parâmetros necessários.

## 3.1 Funções da Aplicação

As funções principais da aplicação são:

- `adicionar_aresta(u, v)`: Adiciona uma aresta entre os vértices $u$ e $v$ no grafo.
- `vizinhos_comuns(u, v)`: Conta o número de vizinhos comuns entre os vértices $u$ e $v$.
- `substituir_extensao(nome_arquivo, nova_extensao)`: Substitui a extensão de um nome de arquivo para gerar o arquivo de saída com a extensão adequada.

## 3.2 Como Executar os Programas

Para executar o programa, o usuário deve compilar o código-fonte com um compilador C, como o **GCC**, e depois executar o arquivo gerado, passando os parâmetros apropriados.

### 3.2.1 Parâmetros para o Programa Sequencial

O programa sequencial deve ser executado com os seguintes parâmetros:

```
./sequencial <arquivo_entrada>
```

Onde:

- `<arquivo_entrada>` é o arquivo contendo os dados do grafo (número de vértices e arestas).

Exemplo de execução:

```
./sequencial grafo_entrada.txt
```

### 3.2.2 Parâmetros para o Programa Paralelo

O programa paralelo deve ser executado com o MPI, com os seguintes parâmetros:

```
mpirun -np 6 ./paralelo <arquivo_entrada>
```

Onde:

- `<arquivo_entrada>` é o arquivo contendo os dados do grafo.

- `-np 6` especifica que o programa será executado com 6 processos, que foi o número utilizado para todos os testes.

Exemplo de execução:

```
mpirun -np 6 ./paralelo grafo_entrada.txt
```

# 4 Implementação do Algoritmo

O algoritmo foi desenvolvido em duas versões: sequencial e paralela. A implementação teve como foco a eficiência no processamento de grandes grafos e a correta divisão de trabalho na versão paralela.

## 4.1 Versão Sequencial

A versão sequencial foi implementada para processar a estrutura do grafo armazenado como uma lista de adjacência. O algoritmo realiza as seguintes etapas:

1. **Leitura do Arquivo de Entrada:** O grafo é carregado a partir de um arquivo de lista de arestas (`.edgelist`). Cada linha contém dois números inteiros representando uma aresta entre dois vértices.
2. **Construção da Estrutura de Dados:** Para cada aresta lida, os vértices adjacentes são armazenados em uma lista dinâmica. Caso a lista de adjacência de um vértice exceda sua capacidade inicial, a memória é expandida dinamicamente.
3. **Cálculo dos Vizinhos Comuns:** O algoritmo percorre todas as combinações de pares de vértices ($u$, $v$) e verifica quantos vizinhos eles têm em comum. Para cada par com pelo menos um vizinho comum, o resultado é salvo em um arquivo de saída (`.cng`).
4. **Medição do Tempo de Execução:** A função `clock()` foi utilizada para medir o tempo total gasto na execução do programa.

A seguir, apresentamos os tempos de execução para diferentes tamanhos de grafo:

| Número de Vértices | Tempo de Execução (segundos) |
|---|---|
| 10 | 0.000150 |
| 5000 | 0.421304 |
| 10000 | 1.690862 |
| 20000 | 1.681394 |

## 4.2 Versão Paralela

A versão paralela foi desenvolvida utilizando a biblioteca MPI (*Message Passing Interface*) e segue os passos abaixo:

1. **Leitura do Arquivo de Entrada:** O processo principal (`rank 0`) lê o arquivo de entrada e distribui as informações do grafo para os demais processos.
2. **Divisão de Tarefas:** O grafo é dividido em partes, e cada processo é responsável por calcular os vizinhos comuns de uma porção específica dos pares de vértices. A divisão é feita com base no índice dos vértices, garantindo que a carga de trabalho seja aproximadamente igual entre os processos.
3. **Comunicação entre Processos:** Utilizamos funções como `MPI_Send` e `MPI_Recv` para a troca de dados entre processos. Isso inclui enviar a estrutura do grafo e consolidar os resultados de cada processo.
4. **Cálculo dos Vizinhos Comuns:** Cada processo calcula os vizinhos comuns para os pares de vértices de sua responsabilidade, de forma independente.

5. **Agregação dos Resultados:** O processo principal coleta os resultados de todos os processos utilizando `MPI_Gather` e salva os dados no arquivo de saída (`.cng`).

6. **Medição do Tempo de Execução:** A função `MPI_Wtime()` foi utilizada para medir o tempo total gasto na execução do programa paralelo.

A seguir, os tempos de execução para a versão paralela:

| Número de Vértices | Tempo de Execução (segundos) |
|---|---|
| 10 | 0.000871 |
| 5000 | 44.526581 |
| 10000 | 353.927371 |
| 20000 | 2786.076121 |

### 4.3 Considerações sobre as Implementações

Na versão sequencial, o cálculo de vizinhos comuns utiliza uma abordagem de força bruta, que percorre todas as combinações possíveis de pares de vértices. Isso funciona bem para grafos pequenos, mas apresenta um crescimento exponencial no tempo de execução conforme o número de vértices aumenta.

Já na versão paralela, a divisão do trabalho entre os processos reduziu significativamente o tempo de execução para tarefas maiores. No entanto, o overhead causado pela comunicação entre processos e a consolidação dos resultados impactou a performance, especialmente em grafos menores.

Essas observações são detalhadas no relatório, com gráficos e tabelas que ilustram as diferenças de desempenho entre as versões.

## 5 Relatório

### 5.1 Estratégia de Paralelização

A estratégia adotada para paralelizar o problema envolveu a decomposição do grafo e a execução de suas operações de forma independente entre múltiplos processos. O algoritmo sequencial foi primeiramente implementado e, em seguida, adaptado para execução paralela utilizando a biblioteca **MPI** (Message Passing Interface), que é amplamente utilizada para paralelização em sistemas de computação distribuída. O problema foi paralelizado no nível de operações entre pares de vértices, de modo que cada processo ficou responsável por uma parte do grafo e a computação dos vizinhos em comum entre os vértices. A decomposição do grafo foi feita com base na divisão do número de vértices, garantindo que cada processo tratasse uma quantidade aproximada de dados de maneira balanceada.

Durante a execução paralela, a comunicação entre os processos foi minimizada, e os dados foram distribuídos de forma a evitar redundância e sobrecarga nas trocas de mensagens. A comunicação entre os processos foi realizada utilizando o envio de dados entre os diferentes processos de forma eficiente.

A paralelização foi realizada utilizando 6 processos fixos para todos os testes paralelos. Esse número foi escolhido após análise preliminar, levando em consideração a arquitetura do hardware utilizado e a complexidade do problema.

### 5.2 Análise de Desempenho

A comparação entre a versão sequencial e paralela foi feita com base no tempo de execução para grafos de diferentes tamanhos (10, 5000, 10000 e 20000 vértices). A seguir, apresentamos uma análise detalhada de cada versão do algoritmo:

#### 5.2.1 Versão Sequencial

Os tempos de execução para a versão sequencial são apresentados na tabela anterior.

#### 5.2.2 Versão Paralela

Os tempos de execução para a versão paralela também foram apresentados na tabela anterior.

### 5.2.3 Comparação Detalhada

A figura a seguir apresenta a comparação entre os tempos de execução da versão sequencial e paralela para diferentes tamanhos de grafo. O gráfico foi gerado a partir dos dados do arquivo CSV contendo os tempos de execução para ambos os casos.

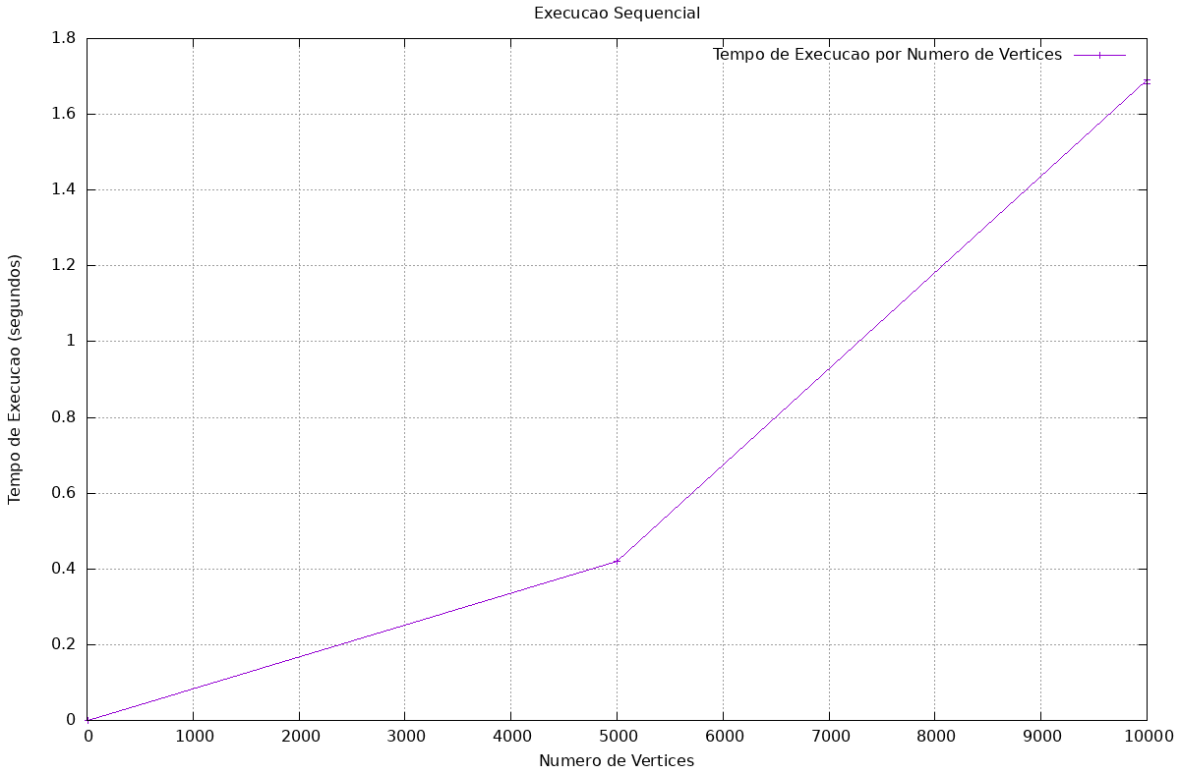


Figura 1: Gráfico de tempo de execução da versão sequencial

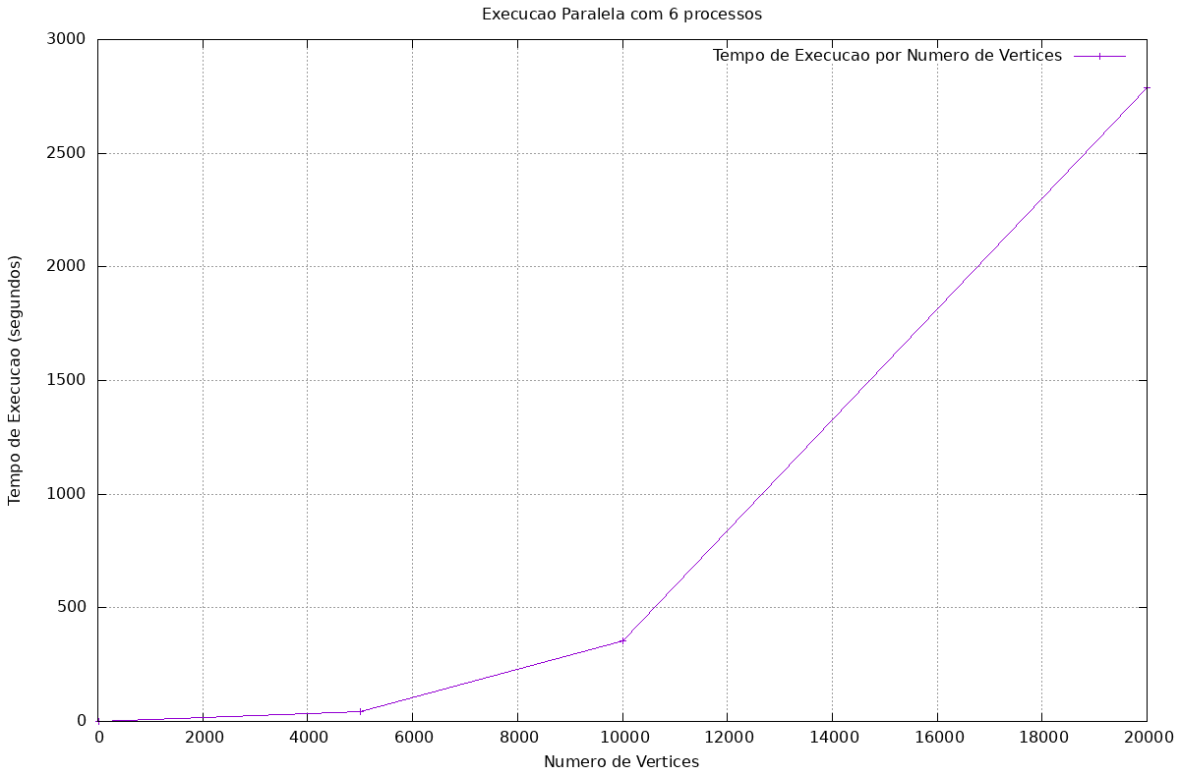


Figura 2: Gráfico de tempo de execução da versão paralela

## 5.3 Melhorias e Otimizações

Embora o algoritmo paralelizado tenha mostrado benefícios, alguns pontos podem ser otimizados:

- **Ajuste na distribuição de tarefas**: Para grafos maiores, a distribuição das tarefas entre os processos poderia ser refinada, garantindo que os processos mais lentos não sejam sobrecarregados com mais dados do que podem processar rapidamente.
- **Otimização da comunicação entre processos**: A comunicação excessiva entre processos pode ser um fator limitante na versão paralela. Uma redução nesse tráfego de dados pode resultar em ganhos significativos de desempenho.
- **Uso de mais processos**: Para grafos maiores, aumentar o número de processos pode resultar em uma redução mais pronunciada no tempo de execução, mas isso também depende da capacidade do hardware disponível.

## 5.4 Dificuldades Encontradas

Durante o desenvolvimento da solução paralela, algumas dificuldades surgiram, como:

- **Balanceamento de carga**: Garantir que todos os processos fossem usados de maneira equilibrada e eficiente foi um desafio, especialmente à medida que o tamanho do grafo aumentava.
- **Gerenciamento da memória**: À medida que o número de processos aumentava e o grafo se expandia, o gerenciamento de memória se tornou mais complexo, exigindo técnicas de alocação dinâmica de memória para garantir que os dados fossem manipulados corretamente.
- **Overhead da comunicação**: A comunicação entre os processos, embora eficiente, ainda apresentou overhead, o que impactou o desempenho para grafos maiores.

Essas dificuldades, no entanto, foram oportunidades valiosas de aprendizado, permitindo compreender melhor os desafios da computação paralela e os trade-offs entre aumento do número de processos e a eficiência da execução.

# 6 Considerações Finais

O trabalho mostrou a importância da computação paralela para a redução do tempo de execução de algoritmos em grafos de grande porte. No entanto, a eficiência da paralelização depende de vários fatores, como a distribuição das tarefas e a comunicação entre processos. A comparação entre as versões sequenciais e paralelas revelou que, embora a versão paralela tenha se mostrado eficiente para grafos menores, há ainda espaço para melhorias, principalmente na otimização da comunicação e na utilização de mais processos.