Universidade Federal São João del Rei
Departamento de Ciência da Computação
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Trabalho Prático 1 - MPI

Discentes: Paulo Victor Fernandes Sousa

Docente: Rafael Sachetto Oliveira

Data: 24/11/2024

São João del Rei

2024

Sumário

1 Introdução **2**

2 Metodologia **2**
2.1 Representação do Problema 2
2.2 Versão Sequencial 3
2.3 Versão Paralela com MPI 3
2.4 Integração dos Resultados 4
2.5 Configuração dos Experimentos 4

3 Desenvolvimento **4**
3.1 Organização do Código 5
3.2 Lógica dos Arquivos 5
3.2.1 `grafo.h` e `grafo.c` 5
3.2.2 `util.h` e `util.c` 6
3.2.3 `versao_sequencial.c` 6
3.2.4 `versao_paralela.c` 6
3.3 Instruções para Execução 7
3.3.1 Compilação 7
3.3.2 Execução da Versão Sequencial 7
3.3.3 Execução da Versão Paralela 7

4 Análise de Desempenho **7**
4.1 Resultados Obtidos 7
4.2 Discussão 8
4.3 Limitações e Melhorias Futuras 9

5 Dificuldades **9**

6 Conclusão **10**

1 Introdução

No contexto da computação paralela, a **Message Passing Interface (MPI)** emergiu como uma das ferramentas mais utilizadas para lidar com problemas que demandam alta eficiência. O MPI é um padrão que define funções e rotinas para a comunicação entre processos em sistemas paralelos. Ele possibilita que diferentes processos, localizados em uma ou várias máquinas, colaborem para resolver problemas computacionais grandes, dividindo a carga de trabalho.

Este trabalho aborda o problema de encontrar vizinhos comuns em um grafo não-direcionado $G = (V, E)$, onde V é o conjunto de vértices e E é o conjunto de arestas. Para cada par de vértices u e v, queremos determinar a quantidade de vértices que são vizinhos comuns de ambos.

Definição do Problema:

- Dado um grafo $G = (V, E)$, determinar para cada par de vértices u e v, o conjunto $N(u) \cap N(v)$, onde $N(u)$ representa os vizinhos de u.
- O problema pode ser generalizado para domínios como:
  - **Redes Sociais**: Encontrar amigos em comum entre usuários.
  - **Biologia Computacional**: Identificar interações moleculares compartilhadas entre genes ou proteínas.

A solução foi desenvolvida em duas versões:

- Uma **versão sequencial**, que processa todos os pares de vértices em um único processo.
- Uma **versão paralela**, que utiliza o MPI para dividir a carga entre múltiplos processos, explorando o paralelismo para obter ganhos de desempenho.

2 Metodologia

Este trabalho foi conduzido seguindo as etapas abaixo:

2.1 Representação do Problema

O grafo é representado em formato de lista de arestas (*edgelist*), onde cada linha contém dois inteiros u e v representando uma aresta entre os vértices u e v. Por exemplo, o grafo abaixo com cinco arestas é representado da seguinte forma:

```
0 1
0 2
1 2
1 3
2 3
```

Neste grafo:

- Os vizinhos do vértice 0 são $N(0) = \{1, 2\}$.
- Os vizinhos comuns entre 1 e 3 são $N(1) \cap N(3) = \{2\}$.

2.2 Versão Sequencial

A versão sequencial realiza os seguintes passos:

1. Carrega o grafo em memória utilizando uma lista de adjacência, onde cada vértice armazena uma lista dos seus vizinhos.
2. Para cada par de vértices (u, v), calcula a interseção entre os conjuntos de vizinhos $N(u)$ e $N(v)$.
3. Os pares de vértices e o número de vizinhos comuns são salvos em um arquivo de saída.

Embora eficiente para grafos pequenos, esta abordagem apresenta limitações de desempenho para grafos com centenas de milhares de vértices ou arestas, devido à complexidade combinatória do problema.

2.3 Versão Paralela com MPI

A versão paralela utiliza o MPI para dividir o trabalho entre múltiplos processos. O processo mestre (rank 0) realiza as seguintes etapas:

1. Carrega o grafo e o serializa em um buffer.
2. Distribui o buffer contendo a lista de adjacência para todos os processos utilizando `MPI_Send`.

Cada processo realiza:

1. Recebe o buffer e reconstrói a lista de adjacência localmente.
2. Divide o conjunto de pares (u, v) de forma equilibrada entre os processos.
3. Calcula os vizinhos comuns para os pares atribuídos.
4. Salva os resultados em arquivos parciais.

2.4 Integração dos Resultados

Após o processamento, o processo mestre combina os arquivos gerados por cada processo para produzir o resultado final.

2.5 Configuração dos Experimentos

Os testes foram realizados em um único computador com um processador multicore, simulando um ambiente distribuído. O número de processos foi variado para analisar o impacto do paralelismo no tempo de execução. Diferentes tamanhos de grafos foram utilizados para medir o desempenho das versões sequencial e paralela.

3 Desenvolvimento

O trabalho foi organizado em uma estrutura de diretórios que facilita a manutenção e a modularização do código. A seguir, detalhamos a estrutura e a lógica de cada componente do sistema.

3.1 Organização do Código

A estrutura de diretórios segue o formato abaixo:

```
cmpt_mpi/
 Makefile
 bin/
    versao_paralela
    versao_sequencial
 include/
    grafo.h
    util.h
 input/
    entrada.edgelist
    entrada_1000.edgelist
    entrada_100000.edgelist
    entrada_200000.edgelist
    entrada_500000.edgelist
 output/
    entrada.cng
    saida_parcial_0.cng
    saida_parcial_1.cng
    saida_parcial_2.cng
    saida_parcial_3.cng
 src/
     grafo.c
     grafo.o
     util.c
     util.o
     versao_paralela.c
     versao_paralela.o
     versao_sequencial.c
     versao_sequencial.o
```

3.2 Lógica dos Arquivos

3.2.1 `grafo.h` e `grafo.c`

Estes arquivos contêm as definições e implementações das estruturas de dados para representar o grafo:

- **ListaAdjacencia**: Representa os vértices e seus vizinhos.

- Funções auxiliares como:
  - `inicializar_lista`: Inicializa a estrutura de vizinhos para um vértice.
  - `adicionar_vizinho`: Adiciona um vértice à lista de adjacência.
  - `liberar_grafo`: Libera a memória alocada para o grafo.

3.2.2 `util.h` e `util.c`

Fornecem funções auxiliares para:

- `gerar_nome_saida`: Gera o nome do arquivo de saída baseado no arquivo de entrada.
- Manipulação de buffers e verificação de interseção de conjuntos.

3.2.3 `versao_sequencial.c`

A lógica principal da versão sequencial é implementada neste arquivo:

1. Carregamento do grafo a partir do arquivo *edgelist*.
2. Para cada par de vértices u e v:
   - Calcula a interseção $N(u) \cap N(v)$.
   - Salva o resultado no arquivo de saída.
3. Mede o tempo de execução e exibe o resultado na tela.

3.2.4 `versao_paralela.c`

A lógica principal da versão paralela é implementada utilizando o MPI:

1. O processo mestre (rank 0):
   - Carrega o grafo e serializa os dados em um buffer.
   - Distribui o buffer para os demais processos usando `MPI_Send`.
2. Cada processo:
   - Recebe o buffer e reconstrói a lista de adjacência.
   - Calcula os vizinhos comuns para os pares de vértices atribuídos.
   - Salva os resultados parciais em arquivos separados.
3. Após o processamento, os resultados podem ser combinados manualmente ou programaticamente.

3.3 Instruções para Execução

3.3.1 Compilação

A compilação é feita utilizando o `Makefile`:

```
make all
```

Os binários gerados serão colocados na pasta `bin/`.

3.3.2 Execução da Versão Sequencial

```
./bin/versao_sequencial input/entrada.edgelist
```

O resultado será salvo na pasta `output/` com o nome `entrada.cng`.

3.3.3 Execução da Versão Paralela

```
mpirun -np <n_procs> ./bin/versao_paralela input/entrada.edgelist
```

Cada processo criará um arquivo parcial na pasta `output/`, como `saida_parcial_0.cng`, `saida_parcial_1.cng`, etc.

4 Análise de Desempenho

Nesta seção, comparamos o desempenho das versões sequencial e paralela do programa, medindo o tempo de execução para diferentes tamanhos de grafos. Os testes foram realizados em um único computador com um processador multicore, simulando um ambiente distribuído. Os tempos medidos refletem tanto as vantagens do paralelismo quanto as limitações impostas pela arquitetura do sistema.

4.1 Resultados Obtidos

Os tempos de execução medidos foram os seguintes:

Número de Arestas	**Tempo Sequencial (s)**	**Tempo Paralelo (s)**
10	4.375	1.327
10.000	5.250	1.517
100.000	33.578	22.494
200.000	40.078	35.159
500.000	226.984	180.674

Tabela 1: Tempos de Execução para diferentes tamanhos de grafos.

A seguir, apresentamos um gráfico que ilustra a evolução do tempo de execução com o aumento do tamanho do grafo:

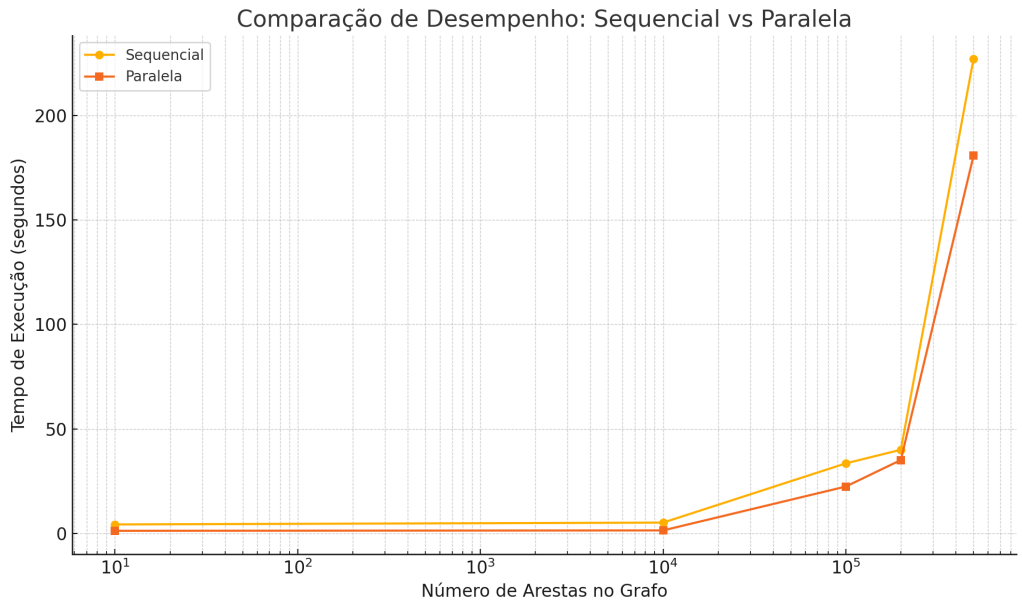

Figura 1: Comparação de Desempenho entre as versões Sequencial e Paralela.

4.2 Discussão

Os resultados indicam que a versão paralela apresenta ganhos de desempenho significativos em comparação com a versão sequencial, especialmente para grafos maiores. Isso é esperado, pois a carga de trabalho é distribuída entre os processos, permitindo que o tempo de execução seja reduzido proporcionalmente à quantidade de processos utilizados.

Entretanto, algumas limitações se tornam evidentes devido à execução em um único computador:

- **Competição por recursos**: Como todos os processos MPI são executados na mesma máquina, eles competem pelo mesmo conjunto de recursos, como CPU, memória e cache. Isso pode limitar o ganho de desempenho.

- **Overhead de comunicação**: Embora o MPI seja projetado para sistemas distribuídos, neste cenário a comunicação ocorre entre processos locais, o que pode adicionar um overhead desnecessário.

- **Escalabilidade limitada**: O número de processos paralelos que podem ser executados está restrito ao número de núcleos físicos disponíveis no processador. Além disso, o desempenho não escala linearmente para um número muito grande de processos devido à sobrecarga de comunicação.

4.3 Limitações e Melhorias Futuras

Para superar as limitações observadas, sugerimos as seguintes melhorias:

- Testar a solução em um cluster real de computadores, onde os processos MPI possam ser distribuídos entre diferentes máquinas, maximizando o paralelismo.
- Implementar técnicas de balanceamento de carga para dividir os pares de vértices de forma mais eficiente entre os processos.
- Utilizar algoritmos mais eficientes para o cálculo da interseção de conjuntos, reduzindo a complexidade do problema.

5 Dificuldades

Durante o desenvolvimento do trabalho, enfrentamos diversos desafios técnicos e conceituais, que destacamos a seguir:

- **Ambiente de Desenvolvimento em WSL (Windows Subsystem for Linux):** A execução do MPI em um ambiente WSL no Windows trouxe dificuldades adicionais, como problemas com as configurações de comunicação entre os processos e restrições no uso do kernel Linux para operações intensivas de memória. Por exemplo, mensagens de aviso relacionadas ao `btl_vader_single_copy_mechanism` indicaram limitações do subsistema em gerenciar cópias de memória de forma eficiente. Além disso, a integração entre ferramentas nativas do Windows e o ambiente Linux nem sempre foi direta, o que demandou ajustes frequentes.
- **Tamanho do Buffer e Falhas de Segmentação:** Durante a serialização do grafo para transmissão entre os processos MPI, enfrentamos problemas relacionados ao tamanho do buffer. Inicialmente, utilizamos um buffer de tamanho fixo, o que resultou em frequentes falhas de segmentação (*segmentation faults*) para grafos maiores. Este problema

foi mitigado com a alocação dinâmica e validações adicionais, mas destacou a importância de um gerenciamento cuidadoso de memória em sistemas paralelos.

- **Divisão de Trabalho e Balanceamento de Carga:** A divisão de pares de vértices entre os processos MPI foi outro ponto crítico. Devido à distribuição uniforme dos índices, alguns processos terminavam suas tarefas muito antes de outros, deixando-os ociosos e reduzindo a eficiência geral. A implementação de estratégias de balanceamento de carga mais inteligentes poderia melhorar significativamente o desempenho.

- **Comparação entre Versões e Validação:** Validar os resultados das versões sequencial e paralela foi um desafio devido à complexidade do problema e ao grande volume de dados gerado. Verificar a precisão dos vizinhos comuns para grafos de tamanhos variados exigiu a implementação de ferramentas de depuração e scripts auxiliares.

6 Conclusão

A implementação paralela utilizando MPI apresentou ganhos de desempenho significativos, especialmente para grafos de grande porte. A comparação entre as versões sequencial e paralela mostrou que a distribuição da carga de trabalho entre os processos pode reduzir substancialmente o tempo de execução.

Entretanto, o trabalho destacou as limitações do uso de MPI em um único computador, onde os processos competem pelos mesmos recursos. Para alcançar melhorias adicionais, algumas sugestões são:

- Executar a solução em um cluster de computadores, permitindo uma distribuição real de processos e um uso mais eficiente dos recursos.

- Implementar estratégias avançadas de balanceamento de carga para dividir os pares de vértices de forma proporcional ao trabalho necessário.

- Utilizar técnicas mais eficientes para comunicação entre os processos MPI, como o uso de `MPI_Scatterv` e `MPI_Gatherv`, em substituição a `MPI_Send` e `MPI_Recv`.

- Explorar algoritmos otimizados para o cálculo de interseções de conjuntos.

Este trabalho demonstrou como o MPI pode ser utilizado para resolver problemas complexos de forma eficiente, mas também evidenciou os desafios que surgem ao lidar com paralelismo em sistemas reais. As lições aprendidas aqui servirão de base para futuras melhorias e implementações em cenários mais amplos.

Referências

- Message Passing Interface Forum. (2015). *MPI: A Message-Passing Interface Standard, Version 3.1.* Disponível em: `https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf`. Acesso em: 18 de Novembro de 2024.

- Open MPI Project. *Open MPI Documentation.* Disponível em: `https://www.open-mpi.org/doc/`. Acesso em: 15 de Novembro de 2024.

- Taneja, H., Aggarwal, S. (2015). *Performance Analysis of Parallel Algorithms using MPI.* International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 5, No. 1. Disponível em: `https://ijarcsse.com`. Acesso em: 21 de Novembro de 2024.