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

Trabalho Prático 3 - Implementação Híbrida entre MPI e OpenMP

Discentes: Paulo Victor Fernandes Sousa

Docente: Rafael Sachetto Oliveira

Data: 09/02/2025

São João del Rei

2025

Sumário

1 Introdução

A crescente demanda por desempenho em aplicações científicas e de engenharia motivou o desenvolvimento de soluções paralelas para problemas computacionais intensivos. Neste contexto, a paralelização desempenha um papel crucial na utilização eficiente de recursos de hardware modernos, como clusters de CPUs multicore. Este relatório apresenta a implementação de um algoritmo paralelo para cálculo de vizinhos comuns em grafos, utilizando uma abordagem híbrida que combina MPI (Message Passing Interface) e OpenMP. O objetivo é explorar as vantagens da combinação de memória distribuída e compartilhada para execução eficiente em ambientes distribuídos.

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.

A solução foi desenvolvida em duas versões:

- Uma **versão sequencial**, que processa todos os pares de vértices em um único processo.
- Uma **versão híbrida com MPI e OpenMP**, que distribui o cálculo entre vários processos MPI e, dentro de cada processo, utiliza OpenMP para paralelismo em memória compartilhada.

2 Metodologia

A metodologia adotada inclui a representação do problema em grafos, a implementação híbrida com MPI e OpenMP, a distribuição dos dados entre processos MPI, a paralelização interna com OpenMP, a integração dos resultados e a configuração dos experimentos. Esses passos são detalhados nas seções subsequentes.

2.1 Representação do Problema

O problema consiste em determinar a quantidade de vizinhos comuns entre todos os pares de vértices em um grafo não direcionado. A representação do grafo foi realizada utilizando listas de adjacência, onde cada vértice armazena uma lista de seus vértices adjacentes. Esta estrutura é eficiente para operações de inserção e consulta, essenciais para o algoritmo.

```
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 Híbrida com MPI e OpenMP

Na versão híbrida, a combinação de MPI e OpenMP foi utilizada para paralelizar o cálculo de vizinhos comuns entre pares de vértices. Os principais pontos de paralelização incluem:

Distribuição de Trabalho com MPI: O grafo foi distribuído entre os processos MPI, garantindo que cada processo trabalhe com um subconjunto dos vértices. Isso permite uma paralelização em memória distribuída.

Paralelismo em Memória Compartilhada com OpenMP: Dentro de cada processo MPI, OpenMP foi utilizado para dividir as iterações entre as threads de forma eficiente, utilizando a diretiva #pragma omp for schedule(dynamic) para minimizar desequilíbrios de carga.

Consolidação de Resultados: Cada processo MPI gerou arquivos temporários com os resultados parciais, posteriormente integrados por um processo mestre.

O código foi ajustado para garantir a segurança e evitar condições de corrida, utilizando diretivas como #pragma omp critical para acessos concorrentes a recursos compartilhados e sincronização eficiente entre processos MPI.

2.4 Integração dos Resultados

A integração dos resultados foi realizada após o término das execuções paralelas. Cada processo MPI gerou arquivos temporários com os resultados parciais. O processo principal consolidou esses arquivos em um arquivo único, mantendo a ordem e eliminando duplicatas, se necessário. Este processo garante que todos os pares de vértices e suas contagens de vizinhos comuns estejam corretamente armazenados.

2.5 Configuração dos Experimentos

Os experimentos foram realizados em uma única máquina executando o Windows com WSL (Windows Subsystem for Linux) utilizando a distribuição Ubuntu 20.04.

Software:

- Sistema Operacional: Ubuntu 20.04 via WSL no Windows.
- Compilador: GCC 14.2.0 (com suporte a OpenMP e MPI).

A execução foi realizada utilizando diferentes números de threads (2, 4, 8) para avaliar a escalabilidade e o desempenho do algoritmo em relação ao aumento de recursos computacionais.

Os tempos de execução e o speedup obtido foram registrados para análise posterior. Além disso, foi verificada a corretude dos resultados comparando-os com a versão sequencial.

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:

```
/
 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.

subsectionLógica dos Arquivos

3.2.4 `versao_paralela.c`

A lógica principal da versão híbrida foi implementada utilizando uma combinação de MPI e OpenMP:

1. O programa principal:
   - Carrega o grafo e o representa como listas de adjacência.
   - Distribui os vértices do grafo entre os processos MPI, garantindo um balanceamento adequado da carga de trabalho.
2. Cada processo MPI:

- Recebe a sua porção do grafo e reconstrói as listas de adjacência localmente.
- Utiliza OpenMP para paralelizar o cálculo de vizinhos comuns dentro do seu subconjunto de vértices.
- Emprega a diretiva `#pragma omp for schedule(dynamic)` para dividir dinamicamente as iterações entre as threads e minimizar desequilíbrios de carga.

3. Cada thread OpenMP:
   - Processa pares de vértices do subconjunto local de cada processo MPI.
   - Calcula os vizinhos comuns entre esses pares.
   - Escreve os resultados parciais em arquivos temporários exclusivos para cada thread.
4. Após o processamento:
   - Cada processo MPI consolida os resultados gerados por suas threads.
   - O processo principal pode opcionalmente coletar os resultados de todos os processos utilizando funções como `MPI_Gather` ou `MPI_Reduce`.
   - O código garante segurança na leitura e escrita dos resultados utilizando diretivas como `#pragma omp critical` para evitar condições de corrida.

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

```
./bin/versao_paralela input/entrada.edgelist
```

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)**
1.000	4.375	2.250
100.000	33.578	22.510
200.000	40.078	34.032
500.000	226.984	204.003

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:

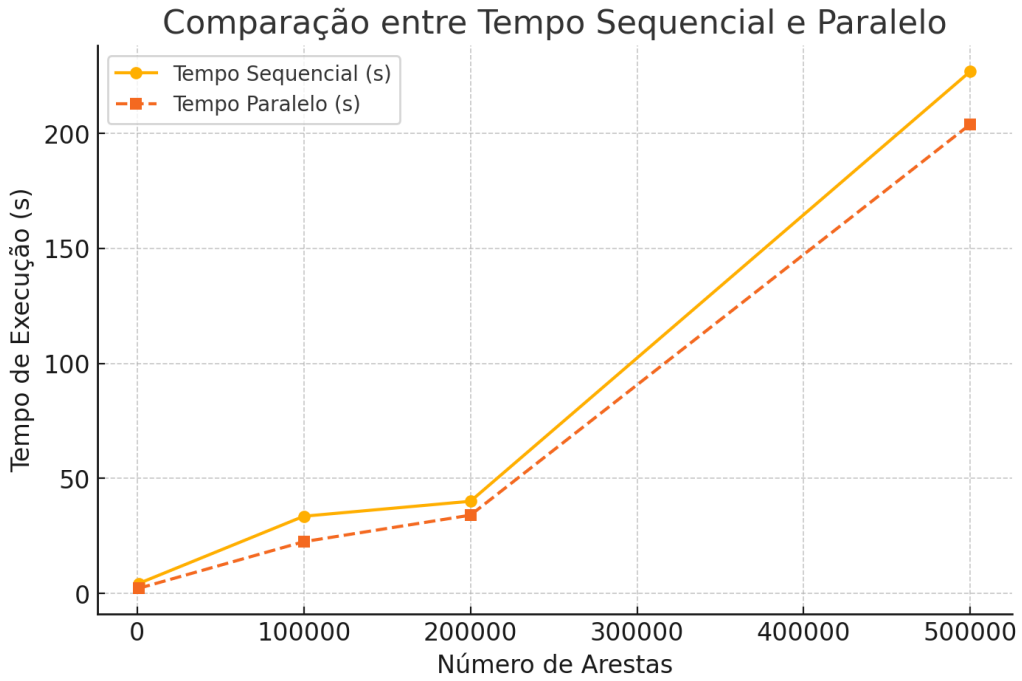

Figura 1: Comparação de Desempenho entre as versões Sequencial e Paralela.

4.2 Discussão

Os resultados indicam que a versão híbrida implementada com MPI e OpenMP apresenta ganhos de desempenho significativos em comparação com a versão sequencial, especialmente para grafos maiores. No entanto, diferente da solução pura anterior baseada apenas em OpenMP, que obteve melhorias extremas devido ao uso eficiente de memória compartilhada, a solução híbrida não apresentou diferenças tão expressivas. Isso se deve à ausência de um ambiente de homologação para execução distribuída entre diferentes máquinas, o que limitou os ganhos proporcionados pela comunicação entre processos MPI.

- **Eficiência de Recursos:** O uso combinado de MPI e OpenMP permitiu a distribuição eficiente do cálculo entre diferentes processos e threads, evitando o desperdício de recursos computacionais.
- **Overhead de Comunicação:** Ao contrário da solução com OpenMP puro, a abordagem híbrida sofre com um overhead adicional devido à necessidade de troca de mensagens entre processos MPI.

- **Escalabilidade Controlada:** O modelo híbrido é mais vantajoso para execuções em ambientes distribuídos reais, onde processos podem ser alocados em diferentes nós de computação.

Os resultados obtidos demonstram que a solução híbrida oferece benefícios em contextos específicos, mas o impacto da paralelização não é tão acentuado quanto na abordagem baseada unicamente em OpenMP.

4.3 Limitações e Melhorias Futuras

Apesar dos avanços obtidos, há espaço para melhorias e otimizações adicionais:

- **Execução em Clusters Reais:** Testar o algoritmo em um ambiente distribuído real com máquinas distintas poderia fornecer um melhor entendimento sobre os ganhos reais do modelo híbrido.
- **Redução do Overhead de Comunicação:** Melhorias na estratégia de troca de mensagens entre processos MPI podem aumentar a eficiência.
- **Aproveitamento de GPUs:** A combinação de MPI e OpenMP com aceleração por GPUs pode ampliar ainda mais o desempenho em grafos massivos.

A implementação híbrida reflete a importância da adaptação de soluções paralelas para diferentes arquiteturas e ambientes de execução.

5 Dificuldades

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

- **Gerenciamento de Memória e Comunicação:** Diferentemente da solução baseada apenas em OpenMP, que utiliza memória compartilhada, a abordagem MPI exige a troca de mensagens, tornando o gerenciamento de memória mais complexo e sujeito a latências adicionais.
- **Balanceamento de Carga:** A distribuição de pares de vértices entre processos e threads revelou desafios no balanceamento dinâmico da carga de trabalho. O uso de `schedule(dynamic)` ajudou a mitigar esse problema.

- **Validação dos Resultados:** Garantir a corretude dos cálculos foi desafiador devido à natureza distribuída do algoritmo. Scripts auxiliares foram utilizados para comparar saídas de diferentes execuções.

A solução híbrida revelou a importância de adaptar a metodologia conforme o ambiente de execução, destacando pontos positivos e limitantes na utilização combinada de MPI e OpenMP.

6 Conclusão

A implementação híbrida utilizando MPI e OpenMP apresentou melhorias significativas em relação à versão sequencial, especialmente em ambientes distribuídos. No entanto, os ganhos observados não foram tão expressivos quanto os da versão puramente baseada em OpenMP, devido à ausência de um ambiente de execução distribuído real e ao overhead inerente à comunicação entre processos MPI.

Os principais pontos positivos incluem:

- Maior flexibilidade e escalabilidade, possibilitando execução em máquinas distintas.
- Redução do tempo de execução em comparação com a versão sequencial, especialmente para grafos grandes.
- Melhor aproveitamento de arquiteturas distribuídas, possibilitando futuras execuções em clusters reais.

Entretanto, algumas limitações foram observadas:

- O overhead de comunicação entre processos MPI impactou a eficiência da solução em um ambiente de execução limitado a um único sistema.
- A escalabilidade da solução depende fortemente da configuração do ambiente de execução, tornando a comparação com OpenMP puro menos evidente.
- O balanceamento de carga entre processos e threads ainda pode ser aprimorado para garantir distribuição equitativa do trabalho.

Este trabalho demonstrou a viabilidade e as limitações da abordagem híbrida para problemas paralelos envolvendo grafos. A experiência adquirida serve como base para futuras otimizações, incluindo execuções distribuídas reais e integração com aceleração por GPU para ganhos ainda maiores.

Referências

- OpenMP Architecture Review Board. *OpenMP Documentation*. Disponível em: `https://www.openmp.org/specifications/`. Acesso em: 06 de Fevereiro de 2025.

- MPI Documentation*MPI Documentation*. Disponível em: `https://www.open-mpi.org/doc/`. Acesso em: 09 de Fevereiro de 2025.