Universidade Estadual de Campinas - UNICAMP
Faculdade de Tecnologia - FT

LABORATÓRIO III - MPI

GRUPO: **OPTIMUS TECH**

Integrantes:
Carolina da Silva Sancho 214376
Gabriella Carlos do Nascimento 260587

LIMEIRA, SÃO PAULO
21 DE DEZEMBRO DE 2020

SUMÁRIO COM HYPERLINKS

1. Introdução

Este laboratório tem como objetivo resolver um problema utilizando biblioteca para programação de alto desempenho (PAD), onde neste trabalho será utilizado a biblioteca "mpi.h", cujo código foi realizado obrigatoriamente na Linguagem C e executado no MobaXterm (para Windows).

O problema a ser resolvido neste laboratório, e nos demais, consiste em calcular a multiplicação entre matrizes e a redução pela soma dos elementos da matriz resultante dessa multiplicação. Como requisitos importantes para a realização da multiplicação das matrizes temos:

- As matrizes precisam ser alocadas dinamicamente (com o comando malloc ou equivalente).
- Todas as matrizes necessárias para os programas precisam ser alocadas em uma única etapa.

2. Compilação e Execução

1) Para compilar e executar os programas em C **sem o script** no MobaXterm ou em outro terminal siga os seguintes passos:

PASSO 1: Para compilar e executar o código em "criaArquivos.c" , onde serão gerados os arquivos com os dados que irão compor as matrizes do código em "matrizMPI.c", e que serão usadas para o cálculo da multiplicação das matrizes e da redução pela soma da matrizD, digite o seguinte comando:

```
gcc criaArquivos.c -o criaArquivos
./criaArquivos y w v arqA.dat arqB.dat arqC.dat
```

As variáveis "y w v" representam o tamanho das matrizes (matrizA[y][w], matrizB[w][v] e matrizC[v][1]), e "arqA.dat arqB.dat arqC.dat" são os arquivos que foram criados no código "criaArquivos.c". Para os valores de "y w v", utilizaremos: y=10 w=10 v=10, y=100 w=100 v=100 e y=1000 w=1000 v=1000, então basta substituir esses valores para gerar novos arquivos.

PASSO 2: Compilar e executar o código em "matrizMPI.c", para realizar o cálculo da multiplicação das matrizes e da redução pela soma da matrizD, para isso, digite os seguintes comandos:

```
gcc criaArquivos.c -o criaArquivos
./criaArquivos y w v arqA.dat arqB.dat arqC.dat
```

```
mpicc -o matrizMPI matrizMPI.c
mpirun -n 4 ./matrizMPI y w v arqA.dat arqB.dat arqC.dat arqD.dat
```

Em que, "matrizMPI.c" é o nome do arquivo em C, "mpicc" e "mpirun" são os comandos para executar a biblioteca "mpi.h", "y w v" são as variáveis do tamanho das matrizes (matrizA[y][w], matrizB[w][v] e matrizC[v][1]), e "arqA.dat arqB.dat arqC.dat" são os arquivos que foram criados no código "criaArquivos.c" e o "arqD.dat" que foi criado no arquivo "matrizMPI.c", todos com valores aleatórios, conforme quantidade determinada pelas variáveis digitadas, cujos valores serão utilizados para realizar a multiplicação das matrizes no código "matrizMPI.c". Abaixo, estão os exemplos dos comando com os valores das variáveis determinadas pelas especificações dos laboratório:

Para y=10, w=10 e y=10, temos:

```
gcc criaArquivos.c -o criaArquivos
./criaArquivos 10 10 10 arqA.dat arqB.dat arqC.dat
```

```
mpicc -o matrizMPI matrizMPI.c
mpirun -n 4 ./matrizMPI 10 10 10 arqA.dat arqB.dat arqC.dat
arqD.dat
```

Para y=100, w=100 e y=100, temos:

```
gcc criaArquivos.c -o criaArquivos
./criaArquivos 100 100 100 arqA.dat arqB.dat arqC.dat
```

```
mpicc -o matrizMPI matrizMPI.c
mpirun -n 4 ./matrizMPI 100 100 100 arqA.dat arqB.dat arqC.dat
arqD.dat
```

Para y=1000, w=1000 e y=1000, temos:

```
gcc criaArquivos.c -o criaArquivos
./criaArquivos 1000 1000 1000 arqA.dat arqB.dat arqC.dat
```

```
mpicc -o matrizMPI matrizMPI.c
mpirun -n 4 ./matrizMPI 1000 1000 1000 arqA.dat arqB.dat arqC.dat
arqD.dat
```

2) Para compilar e executar o programa em C **com o script** digite os seguintes comandos (em que novamente "y w v" são as variáveis do tamanho das matrizes):

PASSO 1: Compilar e executar o código em "`criaArquivo.c`", seguindo os passos disponíveis acima em **PASSO 1** onde não é utilizado o script.

PASSO 2: Escrever os seguintes comandos:

```
chmod +x matriz.sh
./matriz.sh
mpirun -n 4 ./matrizMPI y w v arqA.dat arqB.dat arqC.dat arqD.dat
```

Exemplo:

```
chmod +x matriz.sh
./matriz.sh
mpirun -n 4 ./matrizMPI 10 10 10 arqA.dat arqB.dat arqC.dat
arqD.dat
```

3. Análise e Gráficos

A partir da execução dos códigos em linguagem C disponíveis em: https://github.com/carolsancho/Laboratorios-PAD, os tempos de processamento da multiplicação e do cálculo da redução por soma da matriz D, descontadas as operações de entrada e saída, para os valores das seguintes variáveis: y = 10, w = 10, v = 10 **(1)**; y =100, w = 100, v = 100 **(2)**; y =1000, w = 1000, v = 1000 **(3)**, em que **não foi utilizado a biblioteca "`mpi.h`"** tiveram os seguintes resultados:

VARIÁVEIS	TEMPO DE PROCESSAMENTO (ms)
(1)	0.03
(2)	5.01
(3)	6320.56

Agora, os tempos de processamento para as mesmas variáveis e **utilizando a biblioteca "`mpi.h`"** foram obtidos os seguintes resultados:

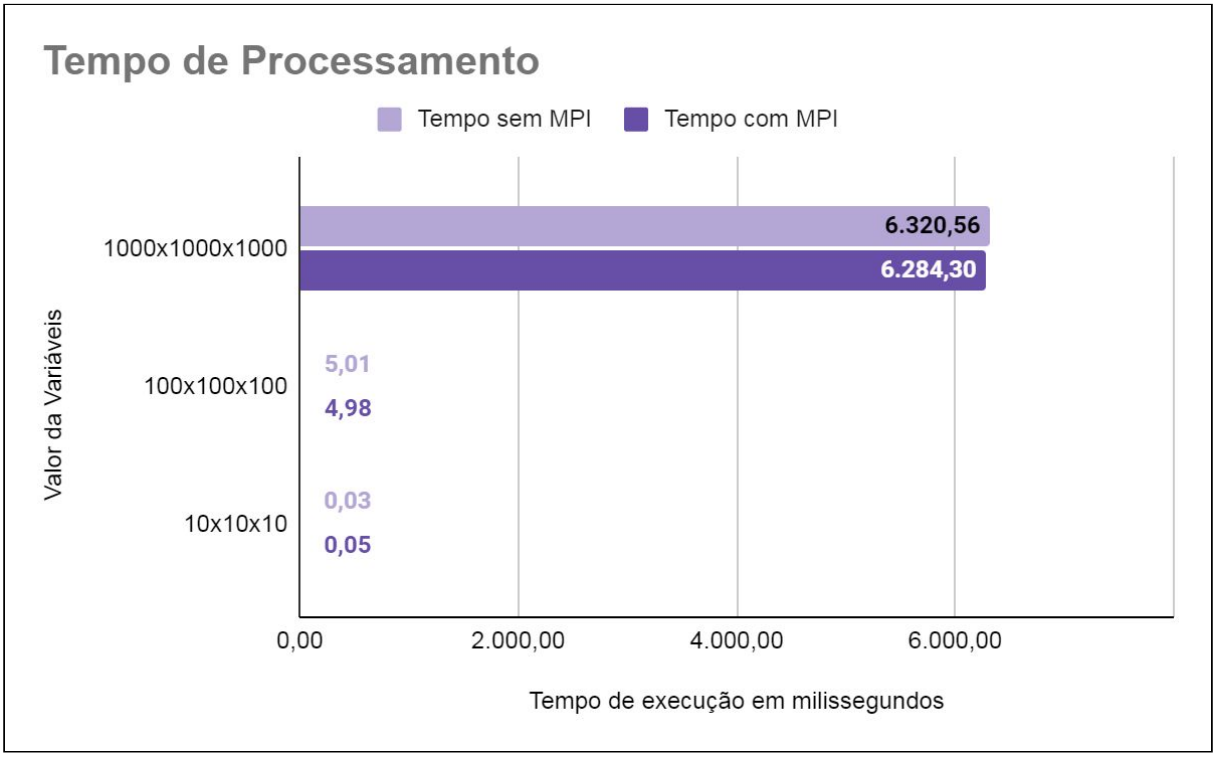
VARIÁVEIS	TEMPO DE PROCESSAMENTO (ms)
(1)	0.05
(2)	4.98
(3)	6284.30

Assim, a partir da análise dos tempos de processamento das tabelas acima foi possível perceber que, quando é utilizado o MPI, devido a programação ser paralela e com memória distribuída, os tempos de processamentos foram inferiores ao serem comparados quando não utilizado a biblioteca "mpi.h", com exceção de valores muitos baixos para as variáveis, como por exemplo o caso encontrado em **(1)**, onde os tempos de processamento não apresentam significativa diferença, porém para resultados superiores ao valor máximo das variáveis encontradas em **(2)** e **(3)**, nota-se uma melhora nos tempos de processamento quando usado a biblioteca "mpi.h", por conta da comunicação de dados entre processos paralelos, que podemos ver sua utilização na Imagem 1 abaixo e nos resultados do Gráfico 1.

Imagem 1 - MPI_Bcast utilizada para transmitir uma mensagem para todos os processos.

```
110     MPI_Bcast(matrizA, y*w, MPI_INT, 0, MPI_COMM_WORLD);
111     MPI_Bcast(matrizB, w*v, MPI_INT, 0, MPI_COMM_WORLD);
112     MPI_Bcast(matrizC, v*1, MPI_INT, 0, MPI_COMM_WORLD);
113     MPI_Bcast(matrizD, y*1, MPI_INT, 0, MPI_COMM_WORLD);
```

Gráfico 1 - Tempos de Processamento da multiplicação e cálculo da redução pela soma da matriz D.



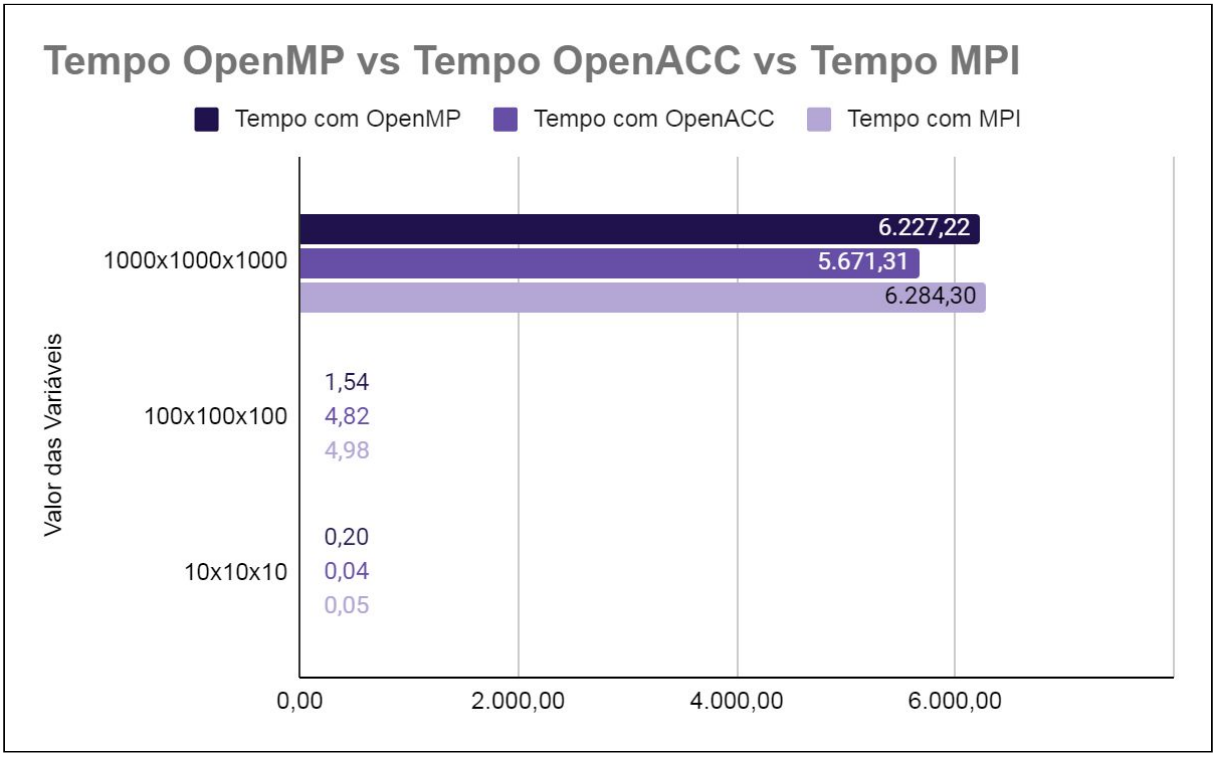
Fonte: elaborado pelas autoras deste documento.

No laboratório anterior vimos que, o OpenMP permite a programação paralela em plataformas de computação de memória compartilhada, e o OpenACC permite que o código seja executado por um acelerador conectado, possibilitando a programação híbrida

de CPU + GPU (como dito anteriormente), o que melhora a sequenciação e o paralelismo, e que o tempo de processamento para variáveis iguais a **(3)** ou superiores a esse valor, apresentam melhor desempenho quando utilizado o OpenACC. Porém, para valores menores entre **(2)** e **(3)**, é expressiva a diferença entre os tempos, o que mostra que um melhor desempenho do uso de multi-thread no código.

Agora, ao compararmos essas bibliotecas acima (OpenMP e OpenACC) com o MPI, que é um modelo de programação paralela de passagem de mensagens, onde os dados são movidos do espaço de endereço de um processo para o de outro processo por meio de operações cooperativas em cada processo, podemos perceber que a biblioteca "`mpi.h`" não apresenta um melhor desempenho para os valores dessas variáveis, devido justamente ao gasto de tempo dessas passagens de mensagem, para realizar as operações de multiplicação presentes no código, com exceção de valores baixos para as variáveis, como em **(1)**, no qual demonstra uma diminuição no tempo de processamento se comparado com a biblioteca OpenMP.

Gráfico 2 - Comparação entre os tempos de Processamento utilizando OpenMP, OpenACC e MPI.



Fonte: elaborado pelas autoras deste documento.

4. Conclusão

A partir do Laboratório III de MPI da disciplina de Tópicos Especiais em Telecomunicações I (Programação de Alto Desempenho), o grupo Optimus Tech foi capaz de ter como

experiência a aplicação de conceitos estudados durante as vídeo aulas de MPI e pelos exemplos apresentados no repositório disponibilizado pelo professor no github, e assim correlacionando-os com a prática, sendo possível através da criação de um código em linguagem C que utilizou o MPI para exemplificar o conceito de memória distribuída.

Portanto, através da prática foi possível ter uma melhor fixação dos conteúdos trabalhados na disciplina, além de ter explorado o emprego do MPI e utilizado novamente a ferramenta MobaXterm como suporte.