



**UNIVERSIDADE FEDERAL DE ITAJUBÁ – UNIFEI – CAMPUS ITABIRA**  
**CURSO DE ENGENHARIA DA COMPUTAÇÃO - ECO**  
**SISTEMAS PARALELOS E DISTRIBUÍDOS - ECO036**  
**PROF. JULIANO MONTE-MOR**

## **PROJETO PRÁTICO Nº 02**

YAN ZINGRA PEREIRA - 31455  
DOUGLAS SANTOS DE OLIVEIRA VENÂNCIO - 28614

### Projeto 2 - MPI

Itabira

Maio de 2018

## Introdução

O MPI é um padrão de comunicação por troca de mensagens em ambientes paralelos geralmente utilizado em ambientes com memória distribuída.

Em uma aplicação ou código onde contém uma ou mais tarefas que são divididas em tarefas que se comunicam entre si, o MPI é utilizado para trocar informações ponto a ponto de forma síncrona travando os outros processos de forma segurar a execução deles até o término da comunicação , assíncrona com todos processos funcionando livremente entre as trocas de mensagem.

A comunicação do MPI é dividida em canais de comunicação o que permite modularizar a comunicação permitindo encapsular a comunicação interna entre determinados envios e recebimentos de mensagem.

## Multiplicação de matrizes utilizando troca de mensagens por MPI

Conforme o proposto foi implementado um programa de multiplicação de matrizes utilizando mais de um processo em paralelo, para a comunicação entre os processos foi utilizado troca de mensagens utilizando MPI, inicialmente após a declaração das matrizes inicia o ambiente de troca de mensagens com o MPI\_Init passando para ele o número de processos envolvidos dessa forma designando um rank para cada processo, utilizando o MPI\_Comm\_size é atrelado a variável size para cada processo o número total de processos e com MPI\_Comm\_rank qual é o rank(número de identificação do processo). A utilização desses comandos é mostrada abaixo.

```
//MPI
MPI_Status status;
//iniciando ambiente de troca de mensagens recebendo quantos o numero de processos
envolvidos
MPI_Init(&argc, &argv);
//atrelando a size o numero de processos envolvidos
MPI_Comm_size(MPI_COMM_WORLD, &size);
//atrelando a rank o qual processo esta sendo executado
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
inicio=clock();
```

Trecho de código 1:Inicialização do ambiente de troca mensagens

Após iniciado o ambiente de troca de mensagens e declarado todas as matrizes que serão utilizadas é a multiplicação das matrizes é feita dividida em partes neste código foi dividida em linhas onde cada processo é responsável pelo cálculo do resultado por determinadas linhas que são distribuídas entre os processos conforme seu rank.

```
//dividindo a tarefa da multiplicação entre os processos
for (int i = rank; i < tammat; i = i + size)
{
    for (int j = 0; j < tammat; j++)
    {
        for (int m = 0; m < tammat; m++)
        {
            result[i][j] = result[i][j] + mat1[i][m] + mat1[m][j];
        }
    }
}
```

Trecho de código 2: Cálculo da multiplicação das matrizes

Determinando o processo de rank 0 como um processo mestre e os demais como escravo são enviadas as mensagens dos processos escravos para o processo mestre com as linhas calculadas utilizando MPI\_Send.

```
//verifica se processo é escravo
if (rank != 0)
{
    //envia a mensagem de cada processo escravo conforme a divisão das tarefas feita
    na multiplicação para o processo mestre
    for (int i = rank; i < tammat; i = i + size)
    {
        //envia uma mensagem que inicia no endereço &result[i][0] com tamanho
        tammat, a variavel da mensagem é do tipo MPI_INT
        //A mensagem é enviada para o processo de rank0(processo mestre) com a
        tag de identificação i utilizando o canal de comunicação MPI_COMM_WORLD
        MPI_Send(&result[i][0], tammat, MPI_INT, 0, i, MPI_COMM_WORLD);
    }
}
```

Trecho de código 3: Envio de Mensagens

Após o envio das mensagens é recebido pelo processo mestre todas as mensagens.

```
//verifica se processo é mestre
if (rank == 0)
{
    //o processo mestre o recebe a mensagem de cada processo escravo conforme a
    divisão das tarefas feita na multiplicação
    for (int j = 1; j < size; j++)
    {
        for (int i = j; i < tammat; i = i + size)
        {
            //recebe uma mensagem que inicia no endereço &result[i][0] com tamanho
            tammat, a variavel da mensagem é do tipo MPI_INT
            //A mensagem recebida tem como origem o processo de rank[i](processos
            escravos) com a tag de identificação i utilizando o canal de comunicação MPI_COMM_WORLD
            MPI_Recv(&result[i][0], tammat, MPI_INT, j,i, MPI_COMM_WORLD, &status);
        }
    }
}
```

Trecho de código 4:Recebimento de Mensagens

E é utilizado uma barreira de sincronia com o comando MPI\_Barrier para espere que todos os processos tenham se comunicado.

```
//barreira de sincronismo bloqueia a execução da programa até que todos os processos
tenham se comunicado
MPI_Barrier(MPI_COMM_WORLD);
```

Trecho de código 5: Barreira de sincronismo

Concluída a troca de mensagens o ambiente de troca de mensagens é finalizado com MPI\_Finalize().

```
MPI_Finalize();
```

Trecho de código 6: Finalização do ambiente de troca de mensagens

O código foi compilado e executado utilizando 8 processos.

```
douglinha5k@douglinha5k-To-be-filled-by-O-E-M: ~/Desktop/paralelotrabalho3
File Edit View Search Terminal Help
douglinha5k@douglinha5k-To-be-filled-by-O-E-M:~/Desktop/paralelotrabalho3$ mpic++ matriz.cpp -o matriz
douglinha5k@douglinha5k-To-be-filled-by-O-E-M:~/Desktop/paralelotrabalho3$ mpirun -np 8 matriz
20 20 20 20 20
20 20 20 20 20
20 20 20 20 20
20 20 20 20 20
20 20 20 20 20
20 20 20 20 20
duração:3614ms
douglinha5k@douglinha5k-To-be-filled-by-O-E-M:~/Desktop/paralelotrabalho3$
```

Figura 1: Compilação e execução do programa de multiplicação de matrizes

## Busca dos 2 maiores números de um vetor utilizando troca de mensagens por MPI

Conforme o proposto foi implementado um programa para Busca dos 2 maiores números de um vetor utilizando mais de um processo em paralelo, para a comunicação entre os processos foi utilizado troca de mensagens utilizando MPI, inicialmente após a declaração dos vetores inicia o ambiente de troca de mensagens da mesma forma como foi iniciado na multiplicação de matrizes.

```
//MPI
MPI_Status status;
//iniciando ambiente de troca de mensagens recebendo quantos o numero de processos
envolvidos
MPI_Init(&argc, &argv);
//atrelando a size o numero de processos envolvidos
MPI_Comm_size(MPI_COMM_WORLD, &size);
//atrelando a rank o qual processo esta sendo executado
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
inicio=clock();
```

Trecho de código 7:Inicialização do ambiente de troca mensagens

Após iniciado o ambiente de troca de mensagens e declarado vetor em que se deseja executar a busca o vetor é dividido em vetores menores onde cada processo é designado para uma dessas divisões conforme o seu rank e busca em seu vetor designado o maior e o segundo maior do vetor e os coloca em uma posição do vetor maior e segmaior conforme seu rank.

```
//vetor que irá os maiores numeros achados em cada processo
int *maior = new int[size];
```

```

//vetor que irá os segundo maiores numeros achados em cada processo
int *Segmaior = new int[size];
//encontrando o numero de posições que cada processo ira verificar
divisao = tamvet / size;
sobra = tamvet % size;
//caso numero de posições que cada processo ira verificar não seja igual
if (sobra != 0 && rank == (size - 1))
{
    divisaofim = (divisao * (rank + 1)) + sobra;
}
else
{
    divisaofim = divisao * (rank + 1);
}

//verificação de qual numero é maior numero e o segundo maior em cada processo
for (int i = divisao * rank; i < divisaofim; i++)
{

    if ((i == divisao * rank) || vet[i] > maior[rank])
    {
        Segmaior[rank] = maior[rank];
        maior[rank] = vet[i];
    }
}

```

Trecho de código 8: Divisão do vetor entre o rank e busca do maior e segundo maior número desses vetores

As mensagens são enviadas e recebidas da mesma forma que no multiplicação de matrizes.

```

//verifica se processo é escravo
if (rank != 0)
{
    //envia uma mensagem que inicia no endereço &maior[i] com tamanho 1, a variavel
da mensagem é do tipo MPI_INT
    //A mensagem é enviada para o processo de rank0(processo mestre) com a tag de
identificação rank utilizando o canal de comunicação MPI_COMM_WORLD
    MPI_Send(&maior[rank], 1, MPI_INT, 0, rank, MPI_COMM_WORLD);
    //envia uma mensagem que inicia no endereço &Segmaior[i] com tamanho 1, a
variavel da mensagem é do tipo MPI_INT
    //A mensagem é enviada para o processo de rank0(processo mestre) com a tag de
identificação rank+size utilizando o canal de comunicação MPI_COMM_WORLD

```

```

        MPI_Send(&Segmaior [rank], 1, MPI_INT, 0, rank + size, MPI_COMM_WORLD);
    }

    //verifica se processo é mestre
    if (rank == 0)
    {
        for (int i = 1; i < size; i++)
        {
            //recebe uma mensagem que inicia no endereço &maior[i] com tamanho 1, a
            //variavel da mensagem é do tipo MPI_INT
            //A mensagem recebida tem como origem o processo de i(processos escravos) com
            //a tag de identificação i utilizando o canal de comunicação MPI_COMM_WORLD
            MPI_Recv(&maior[i], 1, MPI_INT, i, i, MPI_COMM_WORLD, &status);
            //recebe uma mensagem que inicia no endereço &2maior[i] com tamanho 1, a
            //variavel da mensagem é do tipo MPI_INT
            //A mensagem recebida tem como origem o processo de i(processos escravos) com
            //a tag de identificação i+size utilizando o canal de comunicação MPI_COMM_WORLD
            MPI_Recv(&Segmaior [i], 1, MPI_INT, i, i + size, MPI_COMM_WORLD, &status);
        }
    }

    //barreira de sincronismo bloqueia a execução da programa até que todos os processos
    tenham se comunicado
    MPI_Barrier(MPI_COMM_WORLD);

```

Trecho de código 9:Envio e recebimento de mensagens com barreira de sincronismo

Com todas as mensagens recebidas o processo mestre procura o maior número e o segundo maior número no vetor maior dessa forma encontrando o maior número do vetor em que se efetuou a busca, e também procura no vetor segmaior seu maior número e compara com o segundo número maior do vetor maior dessa forma encontrando o segundo maior número do vetor em que se efetuou a busca. Após isso o ambiente de troca de mensagens é finalizado.

```

//verifica se processo é mestre
if (rank == 0)
{
    int maiorT = 0;
    int maiorsegT=0;
    int SegmaiorT = 0;
    for (int j = 0; j < size; j++)
    {
        //encontra o maior e o segundo maior numero no vetor maior[]
        if (maior[j] > maiorT || j == 0)

```

```

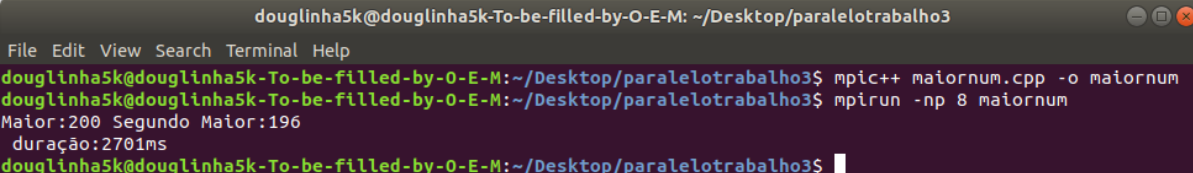
        {
            maiorsegT=maiorT;
            maiorT = maior[j];
        }
        //encontra o maior numero no vetor Segmaior[]
        if (Segmaior[j] > SegmaiorT || j == 0)
        {
            SegmaiorT = Segmaior [j];
        }
    }

    //encontra qual é maior entre o segundo maior numero do vetor maior[] e o maior
    numero do vetor Segmaior
    if(SegmaiorT<maiorsegT)
    {
        SegmaiorT=maiorsegT;
    }
    cout <<"Maior:"<< maiorT<<" Segundo Maior:"<<SegmaiorT;
    fim = clock();
    duracao = fim - inicio;
    cout << "\n duração:" << duracao << "ms\n";
}
MPI_Finalize();

```

Trecho de código 9:Busca do maior e segundo maior números pelo processo mestre e finalização do ambiente de mensagens.

O código foi compilado e executado utilizando 8 processos.



```

douglinha5k@douglinha5k-To-be-filled-by-O-E-M: ~/Desktop/paralelotrabalho3
File Edit View Search Terminal Help
douglinha5k@douglinha5k-To-be-filled-by-O-E-M:~/Desktop/paralelotrabalho3$ mpic++ maiornum.cpp -o maiornum
douglinha5k@douglinha5k-To-be-filled-by-O-E-M:~/Desktop/paralelotrabalho3$ mpirun -np 8 maiornum
Maior:200 Segundo Maior:196
duração:2701ms
douglinha5k@douglinha5k-To-be-filled-by-O-E-M:~/Desktop/paralelotrabalho3$

```

Figura 2: Compilação e execução do programa de busca dos 2 maiores números de um vetor