Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programação Concorrente

Relatório de Entrega de Atividades

Aluno(s): Amanda Oliveira Alves e Fillype Alves do Nascimento
Matrícula: 15/0116276 e 16/0070431
Atividade: Aula Prática 08 - MPICH

1. 5 Threads:

0 (d3f1f7d0da3c) de 5 - Ola, MPI.
4 (d3f1f7d0da3c) de 5 - Ola, MPI.
2 (d3f1f7d0da3c) de 5 - Ola, MPI.
1 (d3f1f7d0da3c) de 5 - Ola, MPI.
3 (d3f1f7d0da3c) de 5 - Ola, MPI.

4 Threads:

0 (d3f1f7d0da3c) de 4 - Ola, MPI.
2 (d3f1f7d0da3c) de 4 - Ola, MPI.
3 (d3f1f7d0da3c) de 4 - Ola, MPI.
1 (d3f1f7d0da3c) de 4 - Ola, MPI.

3 Threads:

2 (d3f1f7d0da3c) de 3 - Ola, MPI.
0 (d3f1f7d0da3c) de 3 - Ola, MPI.
1 (d3f1f7d0da3c) de 3 - Ola, MPI.

2 Threads:

1 (d3f1f7d0da3c) de 2 - Ola, MPI.
0 (d3f1f7d0da3c) de 2 - Ola, MPI.

1 Thread:

0 (d3f1f7d0da3c) de 1 - Ola, MPI.

```
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 1.1.c
// atividade: 1.1
```

```c
#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv){
   int rank, size, namelen;
   char name[100];

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Get_processor_name(name, &namelen);

   printf("%d (%s) de %d - Ola, MPI.\n", rank, name, size);

   MPI_Finalize();

   return 0;
}
```

2.

2.1. Processo 0 de 2 enviando mensagem ao processo 1. Mensagem: 47.
Processo 1 de 2 recebeu mensagem do processo 0. Mensagem: 47.
Processo 1 de 2 enviando mensagem ao processo 0. Mensagem: 47.
Processo 0 de 2 recebeu mensagem do processo 1. Mensagem: 47.

```c
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 2.1.1.c
// atividade: 2.1.1

#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv){
   int rank, np, valor1;
   MPI_Status status;
```

```
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    if (rank == 0) {
        scanf("%d", &valor1);
        MPI_Send(&valor1, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        printf("Processo %d de %d enviando mensagem ao processo 1.
Mensagem: %d.\n", rank, np, valor1);
        MPI_Recv(&valor1, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &status);
        printf("Processo %d de %d recebeu mensagem do processo 1. Mensagem:
%d.\n", rank, np, valor1);
    }
    else if (rank == 1) {
        MPI_Recv(&valor1, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        printf("Processo %d de %d recebeu mensagem do processo 0. Mensagem:
%d.\n", rank, np, valor1);
        MPI_Send(&valor1, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        printf("Processo %d de %d enviando mensagem ao processo 0.
Mensagem: %d.\n", rank, np, valor1);
    }

    MPI_Finalize();

    return 0;
}
```

2.2. O buffer de de envio e de recebimento, já que o programa escreve no mesmo tanto para enviar quanto para receber a mensagem.

3. Processo 0 de 4 enviando mensagem ao processo 1. Mensagem: 1.
Processo 1 de 4 recebeu mensagem do processo 0. Mensagem: 1.
Processo 1 de 4 enviando mensagem ao processo 2. Mensagem: 2.
Processo 2 de 4 recebeu mensagem do processo 1. Mensagem: 2.
Processo 2 de 4 enviando mensagem ao processo 3. Mensagem: 3.
Processo 3 de 4 recebeu mensagem do processo 2. Mensagem: 3.
Processo 3 de 4 enviando mensagem ao processo 4. Mensagem: 4.
Processo 4 de 4 recebeu mensagem do processo 3. Mensagem: 4.
Processo 4 de 4 enviando mensagem ao processo 0. Mensagem: 5.
Processo 0 de 4 recebeu mensagem do processo 4. Mensagem: 5.

```c
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 1.3.c
// atividade: 1.3

#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv){
   int rank, size, valor;
   MPI_Status status;

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);

   if (rank == 0) {
       scanf("%d", &valor);
       MPI_Send(&valor, 1, MPI_INT, 1, 14, MPI_COMM_WORLD);
       printf("Processo %d de %d enviando mensagem ao processo 1.
Mensagem: %d.\n", rank, size-1, valor);
       MPI_Recv(&valor, 1, MPI_INT, size-1, 14, MPI_COMM_WORLD, &status);
       printf("Processo %d de %d recebeu mensagem do processo %d.
Mensagem: %d.\n", rank, size-1, size-1, valor);
   }
```

```
    else {
        MPI_Recv(&valor, 1, MPI_INT, rank-1, 14, MPI_COMM_WORLD, &status);
        printf("Processo %d de %d recebeu mensagem do processo %d.
Mensagem: %d.\n", rank, size-1, rank-1, valor);
        valor+=1;
        MPI_Send(&valor, 1, MPI_INT, (rank+1)%size, 14, MPI_COMM_WORLD);
        printf("Processo %d de %d enviando mensagem ao processo %d.
Mensagem: %d.\n", rank, size-1, (rank+1)%size, valor);
    }

    MPI_Finalize();

    return 0;
}
```

4.

4.1. Processo 0. Valor: 1.
Processo 2. Valor: 1.
Processo 4. Valor: 1.
Processo 1. Valor: 1.
Processo 3. Valor: 1.

```
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 4.1.1.c
// atividade: 4.1.1

#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv){
    int rank, size, valor;
    const int root = 0;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
    if(rank == 0){
        scanf("%d", &valor);
    }

    MPI_Bcast(&valor,1,MPI_INT,root,MPI_COMM_WORLD);


    printf("Processo %d. Valor: %d.\n", rank, valor);

    MPI_Finalize();
    return 0;
}
```

4.2.

```
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: 4.1.2.c
// atividade: 4.1.2

#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main(int argc, char **argv){
    int rank, size, valor, n, soma=0;
    int total_sum = 0, *values;
    const int root = 0;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if(rank == 0){
        scanf("%d", &n);
```

```
        int aux;
        values = malloc(sizeof(int)*n);

        for(int i=0;i<n;i++){
            scanf("%d", &aux);
            values[i] = aux;
        }
    }

    int* n_subset = malloc(sizeof(int) * (n/size));


MPI_Scatter(values,(n/size),MPI_INT,n_subset,(n/size),MPI_INT,0,MPI_COMM_W
ORLD);

    int partial_sum = 0;

    for(int j=0;j<(n/size);j++){
        partial_sum+=n_subset[j];
    }

    int* n_sub_total = NULL;

    if(rank == 0){
        n_sub_total = malloc(sizeof(int)*size);
    }


MPI_Gather(&partial_sum,1,MPI_INT,n_sub_total,1,MPI_INT,0,MPI_COMM_WORLD);

    if(rank == 0){
        total_sum+=*n_sub_total;
    }

    printf("Processo %d. Valor: %d.\n", rank, total_sum);

    MPI_Finalize();
```

```
    return 0;
}
```