UNIVERSIDADE FEDERAL DE SÃO PAULO - INSTITUTO DE CIÊNCIA E TECNOLOGIA

Aluna: Rosangela Miyeko Shigenari 92334

Tarefa 8 - MPI Send e Receive

Altere o programa "Ring" do tutorial para que execute da seguinte forma:

1. Execute o anel de forma invertida (4-3-2-1-0).

O trecho de código abaixo foi modificado na linha 16, onde se inicia no último (word_rank +1) até o 0, para que o anel seja de maneira inversa, onde o token "-1" é enviado sempre de um valor de processo para o seu antecessor.

O código é uma pequena modificação do ring.c, já que o envio do token é realizado de maneira inversa, do processo com maior id para o menor.

```
13
14  int token;
15  if (world_rank != world_size-1) {
16    MPI_Recv(&token, 1, MPI_INT, world_rank+1, 0, MPI_COMM_WORLD,
17          MPI_STATUS_IGNORE);
18    printf("Process %d received token %d from process %d\n", world_rank, token,
19    world_rank+1);
20  } else {
21    token = -1;
22  }
23  if((world_rank - 1) > -1) {
24    MPI_Send(&token, 1, MPI_INT, world_rank - 1, 0,MPI_COMM_WORLD);
25  }
26  else MPI_Send(&token, 1, MPI_INT, world_size-1, 0,MPI_COMM_WORLD);
27  if (world_rank == world_size-1) {
28    MPI_Recv(&token, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,MPI_STATUS_IGNORE);
29    printf("Process %d received token %d from process %d\n", world_size-1, token, 0);
30  }
31  MPI_Finalize();
```

A figura abaixo mostra a execução do código no terminal, a execução no sentido inverso (4-3-2-1-0) do anel. O código está em anexo como (ring-2.c)

```
[ROSANGELA@AIRDEROSANGELA: ~/DOWNLOADS$ MPICC RING-2.C -O RING
[ROSANGELA@AIRDEROSANGELA: ~/DOWNLOADS$ MPIRUN -NP 5 RING
PROCESS 3 RECEIVED TOKEN -1 FROM PROCESS 4
PROCESS 2 RECEIVED TOKEN -1 FROM PROCESS 3
PROCESS 1 RECEIVED TOKEN -1 FROM PROCESS 2
PROCESS 0 RECEIVED TOKEN -1 FROM PROCESS 1
PROCESS 4 RECEIVED TOKEN -1 FROM PROCESS 0
```

2. Execute o anel N vezes, porém invertendo a direção a cada execução.

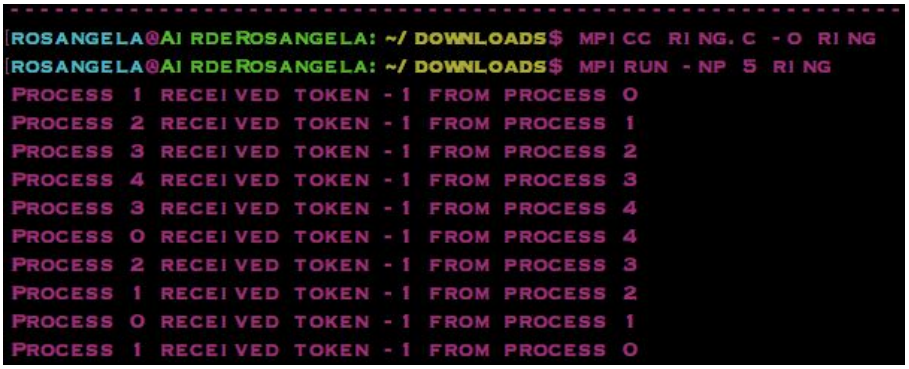
É implementado uma iteração usando o laço for, que é executado N vezes, inserido como um define no código. Se o i, que é o contador da iteração é par, o anel é executado no sentido normal se iniciando do 0 até o último processo, como mostrado no trecho abaixo.

```
int token, i;
    for(i = 0; i< N; i++) {

        if((i%2)==0){
         if (world_rank != 0) {
             MPI_Recv(&token, 1, MPI_INT, world_rank - 1, 0, MPI_COMM_WORLD,
                     MPI_STATUS_IGNORE);
             printf("Process %d received token %d from process %d\n", world_rank, token,
                     world_rank - 1);
         } else {
             token = -1;
         }
         MPI_Send(&token, 1, MPI_INT, (world_rank + 1) % world_size, 0,
                 MPI_COMM_WORLD);

         if (world_rank == 0) {
             MPI_Recv(&token, 1, MPI_INT, world_size - 1, 0, MPI_COMM_WORLD,
                     MPI_STATUS_IGNORE);
             printf("Process %d received token %d from process %d\n", world_rank, token,
                     world_size - 1);
         }
        }
```

Caso contrário, se o número da iteração for ímpar é executado de forma inversa, do último processo ao 0. Como observado abaixo.

```
 else if(i%2!=0){
        if (world_rank != world_size-1) {
            MPI_Recv(&token, 1, MPI_INT, world_rank+1, 0, MPI_COMM_WORLD,
                    MPI_STATUS_IGNORE);
            printf("Process %d received token %d from process %d\n", world_rank, token,
            world_rank+1);
        } else {
            token = -1;
        }
        if((world_rank - 1) > -1) {
            MPI_Send(&token, 1, MPI_INT, world_rank - 1, 0,MPI_COMM_WORLD);
        }
        else MPI_Send(&token, 1, MPI_INT, world_size-1, 0,MPI_COMM_WORLD);
        if (world_rank == world_size-1) {
            MPI_Recv(&token, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,MPI_STATUS_IGNORE);
            printf("Process %d received token %d from process %d\n", world_size-1, token, 0);
        }}
```

A figura abaixo mostra a execução do código no terminal, podendo observar que o anel é executado no sentido normal e posteriormente no inverso, mostrando 2 iterações. (código em anexo como ring-2.c)



3. Gere números aleatórios nos elementos do anel, execute-o e ao final imprima qual é o maior valor e qual processo o gerou.

Foi implementado um laço com N iterações, onde primeiramente, foram definidas as ordens normais e inversas do anel, respectivamente, como mostrado abaixo nas linhas 29 a 42.

Posteriormente, o anel é estruturado, para que o valor do token seja passado para o seu adjacente, podendo ser na ordem normal ou reversa. No código abaixo podemos observar a implementação do anel com as funções *send e receive,* na linha 57 há a geração do token de valor aleatório para ser enviado entre os processos.

```
44   if (world_rank != arg0)
45   {
46      if(token > max) {
47         max = token;
48         maxrank = world_rank;
49      }
50      MPI_Recv(&token, 1, MPI_INT, world_rank + arg1, 0, MPI_COMM_WORLD,
51            MPI_STATUS_IGNORE);
52      printf("Process %d received token %d from process %d\n", world_rank, token,
53            world_rank + arg1);
54   }
55   else
56   {
57      token = rand() % 100;
58   }
59
60   if (!order)
61   {
62      MPI_Send(&token, 1, MPI_INT, (world_rank + 1) % world_size, 0,MPI_COMM_WORLD
63   }
64   else
65   {
66      if ((world_rank - 1) > -1)
67      {
68         MPI_Send(&token, 1, MPI_INT, world_rank - 1, 0, MPI_COMM_WORLD);
69      }
70      else MPI_Send(&token, 1, MPI_INT, world_size - 1, 0, MPI_COMM_WORLD);
71   }
```

A figura abaixo mostra a execução de um exemplo do algoritmo implementado, onde foi gerado um valor aleatório 52, onde na primeira linha o processo 1 recebeu o valor 52 pelo processo 0, primeiramente, logo, o maior valor 52 foi gerado pelo processo 0, retornando ao final. Logo em seguida, é realizada execução na ordem inversa do anel, se iniciando pelo processo 4 que reenvia o valor do token da primeira iteração para o 3, e assim em diante, como ele iniciou o envio nesta iteração, o responsável pelo envio do maior valor será o 4. (Código em anexo como ring-3.c)

```
ROSANGELA@AIRDEROSANGELA:~/DOWNLOADS$ MPIRUN -NP 5 RING
PROCESS 1 RECEIVED TOKEN 52 FROM PROCESS 0
PROCESS 2 RECEIVED TOKEN 52 FROM PROCESS 1
PROCESS 3 RECEIVED TOKEN 52 FROM PROCESS 2
PROCESS 4 RECEIVED TOKEN 52 FROM PROCESS 3
PROCESS 0 RECEIVED TOKEN 52 FROM PROCESS 4
MAIOR ELEMENTO =  52 GERADO POR 0
PROCESS 3 RECEIVED TOKEN 52 FROM PROCESS 4
PROCESS 2 RECEIVED TOKEN 52 FROM PROCESS 3
PROCESS 1 RECEIVED TOKEN 52 FROM PROCESS 2
PROCESS 0 RECEIVED TOKEN 52 FROM PROCESS 1
PROCESS 4 RECEIVED TOKEN 52 FROM PROCESS 0
MAIOR ELEMENTO =  52 GERADO POR 4
```