Brandon Hernández
Computacion Paralela y Distribuida

Hoja de Trabajo 2: OpenMPI y Computación Distribuida

Parte 3:

En la siguiente imagen se observa el output del programa sin modificar:

```
batouuz@batouuz-Inspiron-13-5378:~/Desktop/HDT2_CPD$ mpicc mpiHello.c -o mpiHello.exe
batouuz@batouuz-Inspiron-13-5378:~/Desktop/HDT2_CPD$ mpirun -np 2 mpiHello.exe
Hello from process 0 of 2
Hello from process 1 of 2
batouuz@batouuz-Inspiron-13-5378:~/Desktop/HDT2_CPD$
```

Se observa como cada uno de los procesos imprime en pantalla el mensaje, se aclara que aquí se usaron dos debido a que la máquina solamente cuenta con dos núcleos.

Parte 4:

Luego se observa al correrlo con el código proporcionado. Aquí se observa cómo es que se comunican dos procesos, en el que el proceso con rank 0 le envía un mensaje al rank 1 y luego este mismo lo imprime.

```
batouuz@batouuz-Inspiron-13-5378:~/Desktop/HDT2_CPD$ mpicc mpiHello4.c -o mpiHello4.exe
batouuz@batouuz-Inspiron-13-5378:~/Desktop/HDT2_CPD$ mpirun -np 2 mpiHello4.exe
0 sent Hello World
Hello from process 0 of 2
1 received Hello World
Hello from process 1 of 2
batouuz@batouuz-Inspiron-13-5378:~/Desktop/HDT2_CPD$
```

```c
//----(2) CAPTURA DE DATOS DEL COMUNICADOR----//
int rank, num, i;
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &num);

if (rank == 0) {
  char mess[] = "Hello World";
  printf("%i sent %s\n", rank, mess);
  for (i = 1; i < num; i++)
  MPI_Send(mess, strlen(mess) + 1, MPI_CHAR, i, MESSTAG, MPI_COMM_WORLD);
} else {
  char mess[MAXLEN];
  MPI_Status status;
  MPI_Recv (mess, MAXLEN, MPI_CHAR, 0, MESSTAG, MPI_COMM_WORLD, &status);
  printf ("%i received %s\n", rank, mess);
}

//----(3) DISTRIBUCION DEL TRABAJO----//
printf ("Hello from process %i of %i\n", rank, num);
```

Es interesante como es que este tipo de comunicación se parece a la comunicación de procesos por medio de un socket, puede que por detrás lo este usando y con las funciones de MPI_Send y MPI_Recv se realice la lógica de estos.

Brandon Hernández
Computacion Paralela y Distribuida

Parte 5:

Aquí se observa que solamente tres procesos lograron llegar al final de programa, este se quedo corriendo y se tuvo que forzar el cierre de la corrida.

```
batouuz@batouuz-Inspiron-13-5378:~/Desktop/HDT2_CPD$ mpicc mpiHello5.c -o mpiHello5.exe
batouuz@batouuz-Inspiron-13-5378:~/Desktop/HDT2_CPD$ mpirun --use-hwthread-cpus -np 4 mpiHello5.exe
0 sent Hello World from brandon 5.1
Hello from process 0 of 4
1 received Hello World from brandon 5.1
Hello from process 1 of 4
2 sent Hello World from brandon 5.1
Hello from process 2 of 4
batouuz@batouuz-Inspiron-13-5378:~/Desktop/HDT2_CPD$
```

```c
//----(2) CAPTURA DE DATOS DEL COMUNICADOR----//
int rank, num, i;
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &num);

if (rank % 2 == 0) {
  char mess[] = "Hello World from brandon 5.1";
  printf("%i sent %s\n", rank, mess);
  MPI_Send(mess, strlen(mess) + 1, MPI_CHAR, rank+1, MESSTAG, MPI_COMM_WORLD);
} else {
  char mess[MAXLEN];
  MPI_Status status;
  MPI_Recv (mess, MAXLEN, MPI_CHAR, 0, MESSTAG, MPI_COMM_WORLD, &status);
  printf ("%i received %s\n", rank, mess);
}
```

Lo que se observa es que el ultimo proceso que esta escuchando se queda esperando el mensaje, aunque se le indico al proceso que se lo enviara al proceso con un id mayor por 1.

Brandon Hernández
Computacion Paralela y Distribuida

Parte 5.2:

Aquí se observa como los cuatro procesos lograron terminar y no se tuvo que salir de manera prematura del programa.

```
batouuz@batouuz-Inspiron-13-5378:~/Desktop/HDT2_CPD$ mpicc mpiHello5v2.c -o mpiHello5v2.exe
batouuz@batouuz-Inspiron-13-5378:~/Desktop/HDT2_CPD$ mpirun --use-hwthread-cpus -np 4 mpiHello5v2.
exe
0 sent Hello World from brandon 5.1
2 sent Hello World from brandon 5.1
Hello from process 0 of 4
Hello from process 2 of 4
3 received Hello World from brandon 5.1
Hello from process 3 of 4
1 received Hello World from brandon 5.1
Hello from process 1 of 4
batouuz@batouuz-Inspiron-13-5378:~/Desktop/HDT2_CPD$
```

```c
//----(2) CAPTURA DE DATOS DEL COMUNICADOR----//
int rank, num, i;
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &num);

if (rank % 2 == 0) {
  char mess[] = "Hello World from brandon 5.1";
  printf("%i sent %s\n", rank, mess);
  MPI_Send(mess, strlen(mess) + 1, MPI_CHAR, rank + 1, MESSTAG, MPI_COMM_WORLD);
} else {
  char mess[MAXLEN];
  MPI_Status status;
  MPI_Recv (mess, MAXLEN, MPI_CHAR, rank - 1, MESSTAG, MPI_COMM_WORLD, &status);
  printf ("%i received %s\n", rank, mess);
}
```

Lo que estaba ocurriendo es que se estaba mandando el mensaje, pero nunca se le indico al rank impar que tiene que escuchar al proceso que tiene un id menor al de él por uno. Que es lo que se hizo en este caso.

Esta es una manera bastante interesante de realizar la comunicación entre procesos. Porque la manera a la que estoy acostumbrado es a abrir dos terminales y programar un montón de lógica para poder hacer esta comunicación, en lo personal ha sido por medio de sockets.