PRACTICA 1

Paso de mensajes

María González Herrero

Santiago Molpeceres Díaz

UNIVERSIDAD
NEBRIJA

Indice

Ejercicio 1

Se pide implementar un programa usando MPI, que imprima una salida estándar y cuando el número total de tareas es Y = 50 y X sea un rango de 0 a 49.

Además de calcular el tiempo de ejecución.

Para ello, hemos calculado el tiempo de ejecución desde el propio programa para ver lo que tardan en llegar los procesos, por lo que hemos declarado las variables de *finish* y *start* (permite la sincronización global entre todos los procesos del comunicador). Y la variable time como el resultado de calcular el tiempo de ejecución de *finish – start*.

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

int main (int argc, char** argv) {
    int nproc; //numero de procesos
    int myrank; //id del proceso
    double start,finish,time;

    MPI_Init (&argc, &argv); //Inicializa la aplicacion
    MPI_Barrier(MPI_COMM_WORLD);

    start = MPI_Wtime();

    MPI_Comm_size (MPI_COMM_WORLD, &nproc); //cuantos procesos participan en la aplicacion?
    MPI_Comm_rank (MPI_COMM_WORLD, &myrank);
    MPI_Barrier(MPI_COMM_WORLD);

    finish = MPI_Wtime();
    time = finish-start;

    printf("Hola mundo, soy el proceso %d de un total de %d. TIME: %f \n", myrank, nproc,time);

    MPI_Finalize(); //Fin
}
```

Estos son algunos de los procesos que nos salen, van desde el 0 al 49 en total 50.

```
soy el proceso 37 de un total de 50. TIME: 0.001156
soy el proceso 38 de un total de 50. TIME: 0.001595
soy el proceso 39 de un total de 50. TIME: 0.001162
soy el proceso 40 de un total de 50. TIME: 0.001299
soy el proceso 41 de un total de 50. TIME: 0.002298
soy el proceso 43 de un total de 50. TIME: 0.001215
soy el proceso 45 de un total de 50. TIME: 0.001186
soy el proceso 46 de un total de 50. TIME: 0.001233
soy el proceso 47 de un total de 50. TIME: 0.001869
soy el proceso 25 de un total de 50. TIME: 0.002457
soy el proceso 26 de un total de 50. TIME: 0.001860
soy el proceso 28 de un total de 50. TIME: 0.002587
soy el proceso 29 de un total de 50. TIME: 0.001021
soy el proceso 30 de un total de 50. TIME: 0.001115
soy el proceso 31 de un total de 50. TIME: 0.001890
soy el proceso 32 de un total de 50. TIME: 0.001261
soy el proceso 0 de un total de 50. TIME: 0.001255
soy el proceso 1 de un total de 50. TIME: 0.000511
soy el proceso 2 de un total de 50. TIME: 0.001342
soy el proceso 3 de un total de 50. TIME: 0.000852
soy el proceso 5 de un total de 50. TIME: 0.001137
soy el proceso 6 de un total de 50. TIME: 0.001366
soy el proceso 7 de un total de 50. TIME: 0.001157
soy el proceso 8 de un total de 50. TIME: 0.001534
soy el proceso 9 de un total de 50. TIME: 0.002285
soy el proceso 11 de un total de 50. TIME: 0.001390
soy el proceso 13 de un total de 50. TIME: 0.001570
soy el proceso 14 de un total de 50. TIME: 0.001345
soy el proceso 15 de un total de 50. TIME: 0.001824
soy el proceso 20 de un total de 50. TIME: 0.001663
soy el proceso 36 de un total de 50. TIME: 0.005017
soy el proceso 48 de un total de 50. TIME: 0.004333
soy el proceso 4 de un total de 50. TIME: 0.004764
soy el proceso 16 de un total de 50. TIME: 0.004736
soy el proceso 44 de un total de 50. TIME: 0.007688
soy el proceso 12 de un total de 50. TIME: 0.007677
soy el proceso 10 de un total de 50. TIME: 0.014930
soy el proceso 42 de un total de 50. TIME: 0.015246
```

El gráfico ha sido adaptado para su lectura. Recordemos que el valor esta multiplicado por 10^{-6}

Ejercicio 2

Se pide implementar un programa usando MPI, donde el proceso 0 toma un dato del usuario y lo envía al resto de nodos en anillo. Esto es, el proceso i recibe de i-1 y transmite el dato a i + 1, hasta que el dato alcanza el último nodo.

```c
#include <stdio.h>
#include <mpi.h>
#define TAG_MESSAGE 1
int main(int argc, char** argv)
{
int size;
int rank;
int numero;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

if(rank==0){
    printf("Con que valor quieres empezar el ciclo:\n");
    scanf("%i",&numero);
    MPI_Send(&numero, 1, MPI_INT, (rank + 1) % size, rank, MPI_COMM_WORLD);
    printf("Proceso %i envia %d\n", rank,numero);
}else if(rank < size-1){
        MPI_Recv(&numero,1,MPI_INT,rank-1,rank-1,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
        printf("Proceso: %i recibe el valor: %i\n",rank,numero);
        MPI_Send(&numero,1,MPI_INT,rank+1,rank,MPI_COMM_WORLD);
}else if(rank == size-1){
    MPI_Recv(&numero,1,MPI_INT,rank-1,rank-1,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    printf("Proceso: %i recibe el valor: %i y se acaba el envio\n",rank,numero);
}
MPI_Finalize();
return 0;
}
```

La estructura de este programa se basa en saber en qué proceso o nodo estamos. Por ello si estamos en el proceso 0, lo marcamos como maestro y es el que nos pregunta que valor queremos pasar al resto de procesos.

Por otro lado, si nos encontramos en un nodo esclavo que no es el último, le insertamos las funciones de comunicación en las que enviamos y recibimos el dato que nos pasan. El dato que nos pasan se almacena en la variable “numero” haciendo más fácil el traspaso de información.

Si el nodo es el último, no queremos que se siga propagando el mensaje, por eso el nodo solo tiene función de recepcion.

Los mensajes enviados a la consola por la función *printf*, son para que el proceso pueda ser seguido.

Ejercicio 3

Se pide:

Modificar la implementación del ejercicio 2 para que el dato introducido por el usuario dé tantas vueltas como este indique en el anillo.

Para ellos hemos modificado el ejercicio 2. El dato que pediremos al usuario es el número de vueltas que queremos dar y este será el que iremos pasando y modificando por vueltas.

```
{
  MPI_Status status;
  int num, rank, size, tag, next, from;

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

  /* Usaremos una etiqueta arbitraria de valor 201.
     Calculamos el identificador (rango) del siguiente y del anterior, suponiendo un anillo */

  tag = 201;
  next = (rank + 1) % size;
  from = (rank + size - 1) % size;
```

Primero definimos las variables:

- Num: El valor que vamos a pasar, es decir, el número de vueltas.
- Rank: El rango de valores de los nodos.
- Size: La cantidad de nodos que hay.
- Tag: Valor identificativo del mensaje.
- Next: Obtiene el valor del siguiente nodo.
- From: Obtiene el valor del nodo del que debe recibir la información

```
  if (rank == 0) {
    printf("Introduce el numero de vueltas al anillo:\n");
    scanf("%d", &num);

    printf("Proceso %d envia %d al proceso %d\n", rank, num, next);
    MPI_Send(&num, 1, MPI_INT, next, tag, MPI_COMM_WORLD);
  }
```

Volvemos a poner el nodo 0 como proceso maestro en el que le pedimos al usuario que nos indique cuantas vueltas queremos dar, siendo este el valor que se va a mantener en el anillo.

```
  do {

    MPI_Recv(&num, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status);
    printf("Proceso %d ha recibido %d\n", rank, num);

    if (rank == 0) {
      --num;
     // printf("Proceso 0 descuenta una vuelta\n");
    }
     if(num>0){
      MPI_Send(&num, 1, MPI_INT, next, tag, MPI_COMM_WORLD);
      printf("Proceso %d envia %d al proceso %d\n", rank, num, next);
     }

  } while (num > 0);
  printf("Proceso %d termina %i\n", rank,num);
  MPI_Finalize();
return 0;
}
```

Tras enviar el primer mensaje desde el nodo maestro, nos metemos en un bucle en el que vamos escalando en los diferentes nodos hasta volver a llegar al 0, en el que restamos 1 al numero de vueltas que hemos ido pasando.

Cuando llegamos al nodo maestro 0 y el numero de vueltas es 0. Salimos del bucle, indicando que la cadena de ciclos se ha terminado.

Preguntas

- ¿Qué desventaja se aprecia en este tipo de comunicaciones punto a punto a medida que aumentan el número de procesos requeridos? Razonar la respuesta.

  Este tipo de procesos a medida que aumentan los procesos es necesario tener más slots en el ordenador. A demás de que el tiempo de ejecución de estos nodos por pequeños que sean, en cadena ocuparán un gran tiempo para terminar la tarea.

- ¿Cómo podría mejorar el sistema y su implementación? Razonar la respuesta.

  En lugar de usar buffers de trasmisión con bloqueo podríamos usarlos a forma de difusión, acelerando todos los procesos a menos que estos no tengan alguna dependencia, es decir que el nodo maestro haga una difusión a todos los nodos esclavos.