Arquitectura de Computadores

- Pablo Rayón Zapater
- Fernando Pérez Ballesteros
- José María Fernández

Práctica 1

UNIVERSIDAD NEBRIJA

Índice

Ejercicio 1

Implementar un programa usando MPI, que imprima por salida estándar:

"Hola mundo, soy el proceso X de un total de Y."

cuando el número total de tareas es Y y X un rango de 0 a Y-1.

Calcular el tiempo de ejecución **de cada proceso** (desde que se inicia hasta que termina de imprimir "Hola mundo…") cuando hay 1, 20 y 50 procesos.

Explicar los resultados según varía el número de procesos (se puede realizar una gráfica para ayudar con la explicación).

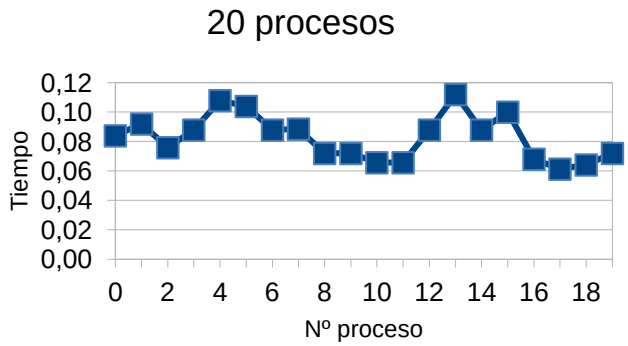
Contestar a las siguientes preguntas (ilustrándolas con los resultados obtenidos en cada ejecución):

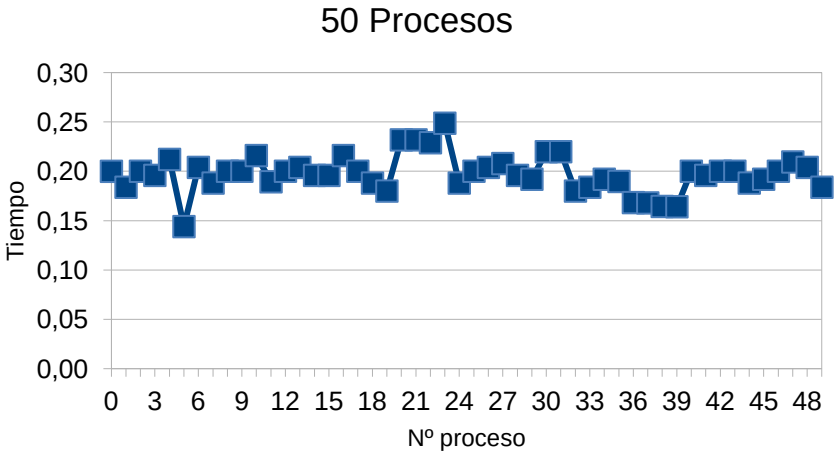
```c
#include <mpi.h>
#include <stdio.h>


int main (int argc, char** argv) {

    int nproc;
    int myrank;
    double start,finish,time;
    MPI_Init (&argc, &argv);
    MPI_Barrier(MPI_COMM_WORLD);
    start = MPI_Wtime();
    MPI_Comm_size (MPI_COMM_WORLD, &nproc);
    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 en %f tiempo.\n",
    myrank, nproc,time);
    MPI_Finalize();

}
```





1. ¿Los procesos imprimen el hola mundo y el tiempo en orden? ¿A qué se debe esto?

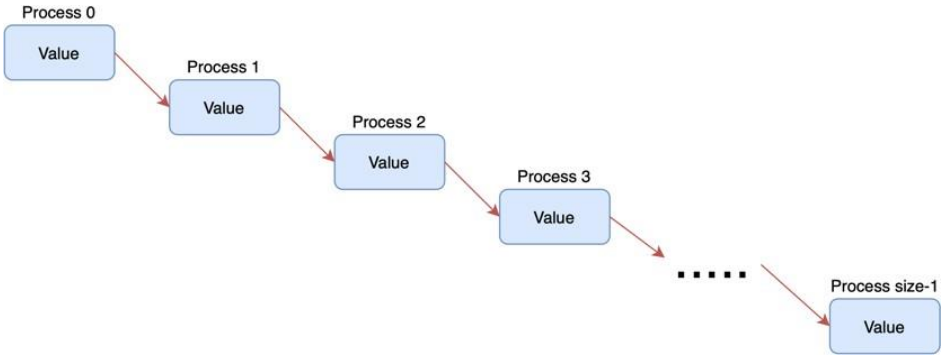
Los procesos no se imprimen en orden, ya que a la hora de crearse y entrar en ejecución dependen del planificador del propio sistema operativo, por lo que serán ejecutados dependiendo de estos criterios, esto puede darse debido a una mayor carga por proceso en función de donde esté almacenada la información de este, si se produce un miss al intentar acceder a esta, por lo que cada proceso tendrá diferentes valores en cuanto a tiempo de ejecución se refiere y serán gestionados de manera acorde en función de esto.

2. ¿Qué pasa con el tiempo de ejecución si eliminamos las barreras (MPI_Barrier())? ¿Y con el orden de ejecución? ¿Porqué?

El tiempo de ejecución se reduce de manera considerable ya que esa barrera que los procesos tienen que superar, la cual hace que se sincronicen entre si y bloqueando a cada proceso que la llame hasta que todos los procesos que están dentro del comunicador estén dentro de los límites de esa barrera, por lo que consecuentemente el tiempo de ejecución total de cada proceso aumenta considerablemente.

Ejercicio 2

Implementar un programa usando MPI, donde el proceso 0 toma un dato del usuario y lo envía al siguiente nodo, que lo envía al siguiente y así hasta llegar al último nodo. Esto es, el proceso i recibe de i-1 y transmite el dato a i+1, hasta que el dato alcanza el último nodo:



Asumir que el dato que se transmite es un entero y que el proceso cero lee el dato del usuario.

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

int main(int argc, char** argv)
{
    int nproc;
    int myrank;
    int datoInput;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    if(myrank == 0){
        printf("Intrduzca dato: ");
        scanf("%d",&datoInput);
        MPI_Send(&datoInput, 1, MPI_INT, (myrank + 1), myrank, MPI_COMM_WORLD);
    }else{
         MPI_Recv(&datoInput,1,MPI_INT,myrank-1,myrank-1,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
         printf("Proceso %d recibe dato : %d \n",myrank,datoInput);
         if(myrank!=nproc-1) MPI_Send(&datoInput, 1, MPI_INT, (myrank + 1), myrank,
MPI_COMM_WORLD);
    }
    MPI_Finalize();
}
```

Ejercicio 3

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

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

   Se tienen que realizar muchas más comprobaciones, para saber si el proceso que se esta ejecutando es el proceso con rango 0, esto para que no se produzca un error y que el último proceso del amillo sea capaz de entregar el mensaje al proceso 0 en lugar de al siguiente, ya que este proceso siguiente no existe. Al tener que realizarse tantas comprobaciones el tiempo de ejecución aumenta proporcionalmente con respecto al numero de procesos en ejecución.

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

   Cuantos menos procesos concurrentes haya a la vez será mejor, ya que se empleará menos memoria y el acceso reiterado a estas posiciones será mucho mas rápido que acceder a una nueva posición para cada iteración por el principio de localidad espacial.

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

int main(int argc, char **argv)
{
    int nproc;
    int myrank;
    int datoInput;
    int contador = 0;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    if (myrank == 0)
    {
        printf("Intrduzca dato: ");
        scanf("%d", &datoInput);
        MPI_Send(&datoInput, 1, MPI_INT, (myrank + 1), myrank, MPI_COMM_WORLD);
    }

    do
    {
        if (myrank == 0)
        {
            MPI_Recv(&datoInput, 1, MPI_INT, nproc - 1, nproc - 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("(RECIBO)Proceso %d recibe dato : %d \n", myrank, datoInput);
            datoInput--;
        }
        else
        {
            if (datoInput != 1)
            {
                MPI_Recv(&datoInput, 1, MPI_INT, myrank - 1, myrank - 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                printf("(RECIBO)Proceso %d recibe dato : %d \n", myrank, datoInput);
            }
            else
                break;
        }
        if (datoInput > 0)
        {
            if (myrank != nproc - 1)
            {
                MPI_Send(&datoInput, 1, MPI_INT, (myrank + 1), myrank, MPI_COMM_WORLD);
            }
            else
            {
                MPI_Send(&datoInput, 1, MPI_INT, 0, myrank, MPI_COMM_WORLD);
            }
        }
        else
            break;
    } while (datoInput >= 1);

    MPI_Finalize();
}
```