Analysis and Design of Parallel Algorithms

Practica Nº 3

M. en C. Sandra Luz Morales Güitrón.

Instrucciones.
Copie y ejecute los siguientes programas, esta práctica tiene el objetivo de que usted analice las funciones enviar y recibir de MPI. No olvide enviar el reporte correspondiente, entrega hoy hasta las 12 de la noche.

Suma algoritmo serial.

```
#include <iostream.h>

int main(int argc, char **argv)
{
	int sum = 0;
	for(int i=1;i<=1000;i=i+1)
	sum = sum + i;
	cout << "The sum from 1 to 1000 is: "<< sum << endl;
}
```

Resultado: The sum from 1 to 1000 is: 500500

Suma algoritmo paralelo.

```
#include<iostream.h>
#include<mpi.h>

int main(int argc, char ** argv)
{
	int mynode, totalnodes;
	int sum = 0,startval,endval,accum;
	MPI_Status status;
	MPI_Init(&argc,&argv);
	MPI_Comm_size(MPI_COMM_WORLD, &totalnodes);
	MPI_Comm_rank(MPI_COMM_WORLD, &mynode);
	startval = 1000*mynode/totalnodes+1;
	endval = 1000*(mynode+1)/totalnodes;

	for(int i=startval;i<=endval;i=i+1)
	sum = sum + i;

	if(mynode!=0)
		MPI_Send(&sum, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
	else
		for(int j=1;j<totalnodes;j=j+1)
		{
```

```
            MPI_Recv(&accum, 1, MPI_INT, j, 1,MPI_COMM_WORLD, &status);
            sum = sum + accum;
            }

      if(mynode == 0)
       cout << "The sum from 1 to 1000 is: " << sum << endl;
      MPI_Finalize();
}
```

Resultado: The sum from 1 to 1000 is: 500500

Enviar y Recibir

```
#include "mpi.h"
#include <iostream>
using namespace std;

int main(int argc, char *argv[])
{
    int rank, size, contador;
    MPI_Status estado;

    MPI_Init(&argc, &argv); // Inicializamos la comunicacion de los procesos
    MPI_Comm_size(MPI_COMM_WORLD, &size); // Obtenemos el numero total de hebras
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Obtenemos el valor de nuestro
identificador

    if(rank == 0){

        MPI_Send(&rank //referencia al vector de elementos a enviar
                ,1 // tamaño del vector a enviar
                ,MPI_INT // Tipo de dato que envias
                ,rank+1 // pid del proceso destino
                ,0 //etiqueta
                ,MPI_COMM_WORLD); //Comunicador por el que se manda

    }else{

        MPI_Recv(&contador // Referencia al vector donde se almacenara lo recibido
                ,1 // tamaño del vector a recibir
                ,MPI_INT // Tipo de dato que recibe
                ,rank-1 // pid del proceso origen de la que se recibe
                ,0 // etiqueta
                ,MPI_COMM_WORLD // Comunicador por el que se recibe
                ,&estado); // estructura informativa del estado

        cout<<"Soy el proceso "<<rank<<" y he recibido "<<contador<<endl;
        contador++;
        if(rank != size-1)
            MPI_Send(&contador, 1 ,MPI_INT ,rank+1 , 0 ,MPI_COMM_WORLD);

    }

    // Terminamos la ejecucion de las hebras, despues de esto solo existira
```

```
    // la hebra 0
    // ¡Ojo! Esto no significa que las demas hebras no ejecuten el resto
    // de codigo despues de "Finalize", es conveniente asegurarnos con una
    // condicion si vamos a ejecutar mas codigo (Por ejemplo, con "if(rank==0)".
    MPI_Finalize();
    return 0;
}
```

Ping – Pong

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

int main(int argc, char** argv) {
  const int PING_PONG_LIMIT = 10;

  // Initialize the MPI environment
  MPI_Init(NULL, NULL);
  // Find out rank, size
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // We are assuming at least 2 processes for this task
  if (world_size != 2) {
    fprintf(stderr, "World size must be two for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }

  int ping_pong_count = 0;
  int partner_rank = (world_rank + 1) % 2;
  while (ping_pong_count < PING_PONG_LIMIT) {
    if (world_rank == ping_pong_count % 2) {
      // Increment the ping pong count before you send it
      ping_pong_count++;
      MPI_Send(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD);
      printf("%d sent and incremented ping_pong_count %d to %d\n",
             world_rank, ping_pong_count, partner_rank);
    } else {
      MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,
               MPI_STATUS_IGNORE);
      printf("%d received ping_pong_count %d from %d\n",
             world_rank, ping_pong_count, partner_rank);
    }
  }
  MPI_Finalize();
}
```