Programación Concurrente, Junio 2023

Práctica 8

Martínez Buenrostro Jorge Rafael.
Universidad Autónoma Metropolitana
Unidad Iztapalapa, México
molap96@gmail.com

Todos los códigos para está práctica se encuentra en el servidor ***upc2@pacifico.izt.uam.mx***. Dentro de la carpeta Practica8 de mi carpeta personal

Actividad B

El objetivo de esta actividad es usar como base el programa de ejemplo ***EjemploBcast.c*** para que cada proceso sume k números generados aleatoriamente y despliegue el resultado, haciendo que el valor de k lo ingrese el usuario

1. La primer modificación es dentro del main, la cual consiste en inicializar la semilla para los números random .

```
srand(time(NULL));
MPI_Init(&argc, &ar
```

2. Para que cada proceso sume k números generados aleatoriamente se modifica la línea en la que se realiza la suma por la siguiente línea.

```
for(i=0; i<k; i++)
    suma += rand()%100;
```

Código Fuente

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

#define EMISOR 0


void emisor(int mi_id)
{   int k;
```

```
    printf("¿Cuantos k números quieres sumar?\n");
    scanf("%d",&k);

    MPI_Bcast(&k,1,MPI_INT,mi_id,MPI_COMM_WORLD);

    printf("Emisor: mande k=%d\n", k);

    int i,suma=0;

    for(i=0; i<k; i++)
        suma += rand()%100;

    printf("Proceso %d: mi suma es %d\n", mi_id, suma);

}


void receptor(int mi_id)
{   int k;


    MPI_Bcast(&k,1,MPI_INT,EMISOR,MPI_COMM_WORLD);

    printf("Receptor %d: recibi %d\n", mi_id, k);

    int i,suma=0;

    for(i=0; i<k; i++)
        suma += rand()%100;

    printf("Proceso %d: mi suma es %d\n", mi_id, suma);
}


int main(int argc, char *argv[])
{
```

```
    int rank, size;

    srand(time(NULL));
    MPI_Init(&argc, &argv); // Inicializacion del entorno MPI

    MPI_Comm_size(MPI_COMM_WORLD, &size); // Obtenemos el numero de procesos en el
comunicador global
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Obtenemos la identificacion de nuestro
proceso en el comunicador global

    if ( rank == EMISOR)
        emisor(rank);
    else
        receptor(rank);

    MPI_Finalize();
    return 0;
}
```

Actividad C

Esta actividad requiere que modifiquemos el programa ejemplo ***EjemploScatter_Gather.c***, en el que el proceso 0 inicialice con números aleatorios enteros un arreglo de tamaño K (donde K es un múltiplo del número de procesos). Después debe repartir el arreglo entre todos los procesos para que simultáneamente calculen la suma parcial de los elementos recibidos. Al final el proceso 0 recupera las sumas parciales y calcula la suma global.

1. Primero se agrega la inicialización de la semilla para la generación de los números random.

```
srand(time(NULL));
```

2. En el proceso emisor se modifica la inicialización del arreglo del índice a un número generado aleatoriamente.

```
for(i=0; i<TAM; i++)
    datosOut[i] = rand() % TAM;
```

3. También se agrega un ciclo en el cual se calculará la suma global.

```
suma=0;
for(i=0; i<numProcesos; i++){
    printf("%d: res[%d]:%d \n",mi_id, i,res[i]);
    suma+=res[i];
}

printf("La suma global es: %d\n", suma);
```

4. Para el proceso receptor se modifica el tamaño del arreglo en el que se guarda el arreglo que se recibe, el tamaño del nuevo arreglo será un múltiplo del número de proceso.

```
int i, datosIn[k/numProcesos];
```

Código Fuente

```
#include <mpi.h>
#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#define EMISOR 0
#define TAM 40

void emisor(int mi_id, int numProcesos)
{   int k = (rand() % 20) * numProcesos;
    int datosOut[k];
    int i, datosIn[k/numProcesos], res[numProcesos];

    for(i=0; i<TAM; i++)
        datosOut[i] = rand() % TAM;

    MPI_Scatter(datosOut,10,MPI_INT,datosIn, 10, MPI_INT, mi_id,MPI_COMM_WORLD);
```

```
    printf("Emisor %d: recibi\n", mi_id);

    int suma = 0;
    for(i=0; i<10; i++){
        printf("%d: arreglo[%d]:%d \n",mi_id, i,datosIn[i]);
        suma += datosIn[i];
    }

    printf("Emisor %d: mandaré %d\n", mi_id, suma);

    MPI_Gather(&suma,1,MPI_INT,res,1,MPI_INT,EMISOR,MPI_COMM_WORLD);

    suma=0;
    for(i=0; i<numProcesos; i++){
        printf("%d: res[%d]:%d \n",mi_id, i,res[i]);
        suma+=res[i];
    }

    printf("La suma global es: %d\n", suma);
}


void receptor(int mi_id, int numProcesos)
{
    int k = (rand() % 20) * numProcesos;
    int i, datosIn[k/numProcesos];

    MPI_Scatter(datosIn,10,MPI_INT,datosIn,10, MPI_INT, EMISOR,MPI_COMM_WORLD);

    printf("Receptor %d: recibi\n", mi_id);

    int suma = 0;
    for(i=0; i<10; i++){
        printf("%d: arreglo[%d]:%d \n",mi_id, i,datosIn[i]);
        suma += datosIn[i];
    }
```

```
    printf("Receptor %d: mandaré %d\n", mi_id, suma);

    MPI_Gather(&suma,1,MPI_INT,&suma,1,MPI_INT,EMISOR,MPI_COMM_WORLD);
}

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

    srand(time(NULL));

    MPI_Init(&argc, &argv); // Inicializacion del entorno MPI

    MPI_Comm_size(MPI_COMM_WORLD, &size); // Obtenemos el numero de procesos en el
comunicador global
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Obtenemos la identificacion de nuestro
proceso en el comunicador global

    if ( rank == EMISOR)
        emisor(rank, size);
    else
        receptor(rank, size);

    MPI_Finalize();
    return 0;
}
```

Actividad D

Esta actividad nos pide un programa que dada una matriz de enteros aleatorios de tamaño NxN (donde N es el número de procesos), la generación de la matriz debe ser colectiva, todos los procesos imprimen los datos con los que van a trabajar pero solo el proceso 0 imprime el resultado

1. Para empezar agregamos la inicialización de la semilla para la generación de números aleatorios

```
srand(time(NULL));
```

2. Dentro del proceso se agrega la creación de cada fila de matriz con números aleatorios

```
// Generar la fila de la matriz para el proceso actual
for (i = 0; i < MATRIX_SIZE; i++) {
    fila[i] = rand() % 100 + i + mi_id;
    printf("Proceso %d - fila[%d]: %d\n", mi_id, i, fila[i]);
}
```

3. Por medio de ***MPI_Allgather*** se envía la fila creada a todos los demás procesos para poder tener toda la matriz creada.

```
// Enviar la fila a todos los procesos para construir la matriz completa
MPI_Allgather(fila, MATRIX_SIZE, MPI_INT, matriz, MATRIX_SIZE, MPI_INT, MPI_COMM_WORLD);
```

4. Cada proceso calcula el máximo de la fila que creo

```
// Calcular el máximo de cada fila en cada proceso
int maximo_fila = fila[0];
for (i = 1; i < MATRIX_SIZE; i++) {
    if (fila[i] > maximo_fila) {
        maximo_fila = fila[i];
    }
}
```

5. Por medio de ***MPI_Reduce*** se calcula el minimo de los máximos en el proceso 0

```
// Calcular el mínimo de los máximos de las filas en el proceso 0
int minimo_maximos;
MPI_Reduce(&maximo_fila, &minimo_maximos, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);
```

6. Casa proceso imprime el máximo de su fila

```
// Imprimir el máximo de la fila en cada proceso
printf("Proceso %d: Máximo de la fila: %d\n", mi_id, maximo_fila);
```

7. Para terminar el proceso 0 es el responsable de imprimir el mínimo de los máximos

```
// El proceso 0 imprime el mínimo de los máximos
if (mi_id == 0) {
    printf("Proceso 0: Mínimo de los máximos: %d\n", minimo_maximos);
}
```

Código Fuente

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



#define MATRIX_SIZE 4

void emisor_receptor(int mi_id, int num_proc)
{
    int i, j;
    int fila[MATRIX_SIZE];
    int matriz[MATRIX_SIZE][MATRIX_SIZE];
    int recv_matriz[MATRIX_SIZE][MATRIX_SIZE];

    // Generar la fila de la matriz para el proceso actual
    for (i = 0; i < MATRIX_SIZE; i++) {
        fila[i] = rand() % 100 + i + mi_id;
        printf("Proceso %d - fila[%d]: %d\n", mi_id, i, fila[i]);
    }

    // Enviar la fila a todos los procesos para construir la matriz completa
    MPI_Allgather(fila, MATRIX_SIZE, MPI_INT, matriz, MATRIX_SIZE, MPI_INT,
MPI_COMM_WORLD);
```

```
    // Calcular el máximo de cada fila en cada proceso
    int maximo_fila = fila[0];
    for (i = 1; i < MATRIX_SIZE; i++) {
        if (fila[i] > maximo_fila) {
            maximo_fila = fila[i];
        }
    }

    // Calcular el mínimo de los máximos de las filas en el proceso 0
    int minimo_maximos;
    MPI_Reduce(&maximo_fila, &minimo_maximos, 1, MPI_INT, MPI_MIN, 0,
MPI_COMM_WORLD);

    // Imprimir el máximo de la fila en cada proceso
    printf("Proceso %d: Máximo de la fila: %d\n", mi_id, maximo_fila);

    // El proceso 0 imprime el mínimo de los máximos
    if (mi_id == 0) {
        printf("Proceso 0: Mínimo de los máximos: %d\n", minimo_maximos);
    }
}

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

    srand(time(NULL));

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

    if (size < MATRIX_SIZE) {
        printf("El número de procesos debe ser al menos %d\n", MATRIX_SIZE);
        MPI_Finalize();
        return 1;
```

```
    }

    emisor_receptor(rank, size);

    MPI_Finalize();
    return 0;
}
```

Actividad E

Esta actividad nos pide un programa para que dada una matriz de enteros aleatorios de tamaño NxN (donde N es el número de procesos), calcule el determinante de la matriz. La generación de la matriz debe ser colectiva, todos los procesos imprimen los datos con los que van a trabajar.

1. Primero se inicializa la semilla para la generación de números aleatorios.

```
srand(time(NULL));
```

2. Cada proceso genera la fila que le corresponde con números aleatorios.

```
// Generar la fila de la matriz para el proceso actual
for (i = 0; i < MATRIX_SIZE; i++) {
    fila[i] = rand()%100 + i + mi_id;
    printf("fila i [%d]", fila[i]);
}
```

3. Por medio de ***MPI_Allgather*** se envía la fila creada a todos los demás procesos para construir la matriz completa.

```
// Enviar la fila a todos los procesos para construir la matriz completa
MPI_Allgather(fila, MATRIX_SIZE, MPI_INT, matriz, MATRIX_SIZE, MPI_INT, MPI_COMM_WORLD);
```

4. Calculamos el determinante de la matriz de cada proceso.

```
int determinante = 0;
for (i = 0; i < MATRIX_SIZE; i++) {
    int prod_principal = 1;
    int prod_secundaria = 1;
    for (j = 0; j < MATRIX_SIZE; j++) {
        prod_principal *= matriz[j][(j + i) % MATRIX_SIZE];
        prod_secundaria *= matriz[j][(MATRIX_SIZE - j + i) % MATRIX_SIZE];
    }
    determinante += prod_principal - prod_secundaria;
}
```

5. Para terminar el proceso 0 imprimirá el determinante de la matriz.

```
if(mi_id == 0){
    // Imprimir el determinante calculado por cada proceso
    printf("Proceso %d: El determinante es %d\n", mi_id, determinante);
}
```

Código Fuente

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

#define MATRIX_SIZE 4

void emisor_receptor(int mi_id, int num_proc)
{
    int i, j;
    int fila[MATRIX_SIZE];
    int matriz[MATRIX_SIZE][MATRIX_SIZE];
    int recv_matriz[MATRIX_SIZE][MATRIX_SIZE];

    // Generar la fila de la matriz para el proceso actual
    for (i = 0; i < MATRIX_SIZE; i++) {
        fila[i] = rand()%100 + i + mi_id;
        printf("fila i [%d]", fila[i]);
    }

    // Enviar la fila a todos los procesos para construir la matriz completa
    MPI_Allgather(fila, MATRIX_SIZE, MPI_INT, matriz, MATRIX_SIZE, MPI_INT,
MPI_COMM_WORLD);

    // Calcular el determinante de la matriz en cada proceso
    int determinante = 0;
    for (i = 0; i < MATRIX_SIZE; i++) {
        int prod_principal = 1;
        int prod_secundaria = 1;
        for (j = 0; j < MATRIX_SIZE; j++) {
            prod_principal *= matriz[j][(j + i) % MATRIX_SIZE];
            prod_secundaria *= matriz[j][(MATRIX_SIZE - j + i) % MATRIX_SIZE];
        }
        determinante += prod_principal - prod_secundaria;
    }
```

```
    if(mi_id == 0){
        // Imprimir el determinante calculado por cada proceso
        printf("Proceso %d: El determinante es %d\n", mi_id, determinante);
    }
}

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

    srand(time(NULL));

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

    if (size < MATRIX_SIZE) {
        printf("El número de procesos debe ser al menos %d\n", MATRIX_SIZE);
        MPI_Finalize();
        return 1;
    }

    emisor_receptor(rank, size);

    MPI_Finalize();
    return 0;
}
```