Seminario práctico 2: Introducción a la Interfaz de Paso de Mensajes (MPI):

UNIVERSIDAD DE GRANADA

Luis González Romero, XXXXXXXXX, luisgonromero@correo.ugr.es

Escuela Técnica Superior de Ingeniería informática y Telecomunicaciones

29 de junio de 2021

Índice

1. Ejercicio 1

```
if(rank == 0 || rank == 1)
{
  if(rank+2 <= size-1)
  {
    MPI_Send(&rank //referencia al vector de elementos a enviar
             ,1 // tamaño del vector a enviar
             ,MPI_INT // Tipo de dato que envias
             ,rank+2 // pid del proceso destino
             ,0 //etiqueta
             ,MPI_COMM_WORLD); //Comunicador por el que se manda
    cout<< "Soy el proceso "<<rank<<" y he enviado "<<rank<<endl;
  }
}else{
  MPI_Recv(&dato // Referencia al vector donde se almacenara lo recibido
          ,1 // tamaño del vector a recibir
          ,MPI_INT // Tipo de dato que recibe
          ,rank-2 // 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 "<<dato<<endl;

  if(rank+2 <= size-1)
  {
    MPI_Send(&dato //referencia al vector de elementos a enviar
             ,1 // tamaño del vector a enviar
             ,MPI_INT // Tipo de dato que envias
             ,rank+2 // pid del proceso destino
             ,0 //etiqueta
             ,MPI_COMM_WORLD); //Comunicador por el que se manda
    cout<< "Soy el proceso "<<rank<<" y he enviado "<<rank<<endl;
  }
}
```

He añadido los ifs correspondientes para que solo los proceso 0 y 1 envíen sus *ids*(solo cuando haya mas procesos pares o impares) y los correspondientes reciban ese *id*(y envien al siguiente cuando proceda).

2. Ejercicio 2

```
if(rank == 0)
{
  cout<<"introduce la precision del calculo (n > 0): ";
  cin>>n;
}
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

double PI25DT = 3.14159265358979323846264 3;
double h = 1.0 / (double) n;
double sum = 0.0;

int istart = (n/size)*rank+1, iend = (n/size)*(rank+1);
cout << "istart: " << istart << " iend:  " << iend << " para " << rank << endl;
for (int i = istart; i <= iend; i++) {
  double x = h * ((double)i - 0.5);
  sum += (4.0 / (1.0 + x*x));
}

double pi_parcial = sum * h;
double pi_reduce;
MPI_Allreduce(&pi_parcial, &pi_reduce, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
```

Se difunde el valor introducido por teclado al resto de procesos. Luego, cada proceso calcula su intervalo y se reduce para todos ya que se pide que la solución se obtenga para todos los procesos.

3. Ejercicio 3

```
long istart = rank * tama/size; // comienzo del bloque local
long iend = istart + tama/size; // final del bloque local
// calculo del bloque de b local
for (long i= istart; i < iend; ++i)
  VectorB[i] = (i + 1)*10;

// Calculo de la multiplicacion escalar entre vectores
long producto = 0;
for (long i = 0; i < tama/size; ++i) {
//  cout << "rank: " << rank << " " << i << " " << i+istart << " " <<
    producto += VectorALocal[i] * VectorB[i+istart];
}
long total;

// Reunimos los datos en un solo proceso, aplicando una operacion
// aritmetica, en este caso, la suma.
MPI_Reduce(&producto, // Elemento a enviar
        &total, // Variable donde se almacena la reunion de los datos
        1, // Cantidad de datos a reunir
        MPI_LONG, // Tipo del dato que se reunira
        MPI_SUM, // Operacion aritmetica a aplicar
        0, // Proceso que recibira los datos
        MPI_COMM_WORLD); // Comunicador
```

Para el vectorB, ahora se inicializa cada segmento s_i por el proceso p_i y el producto lo hace cada proceso multiplicando su vectorA local por su segmento s_i del vectorB.

4. Ejercicio 4

```
float v[size_nuevo];
if(rank==1) // primer proceso impar
{
  for(int i=0; i < size_nuevo; ++i)
    v[i] = i*0.7 + 3;
}

float v_parcial;
MPI_Scatter(v, 1, MPI_FLOAT, &v_parcial, 1, MPI_FLOAT, 0, comm);
```

Dejo que el primer proceso impar(rank global = 1) inicialice el vector y se reparten al resto de impares usando el comunicador *comm* con color=rank %2. Así el root 0, que corresponde al primero de los impares lo reparte al resto.