PRACTICA 4

Resolución de problemas Mediante el modelo de comunicación colectivo

María González Herrero

Santiago Molpeceres Díaz

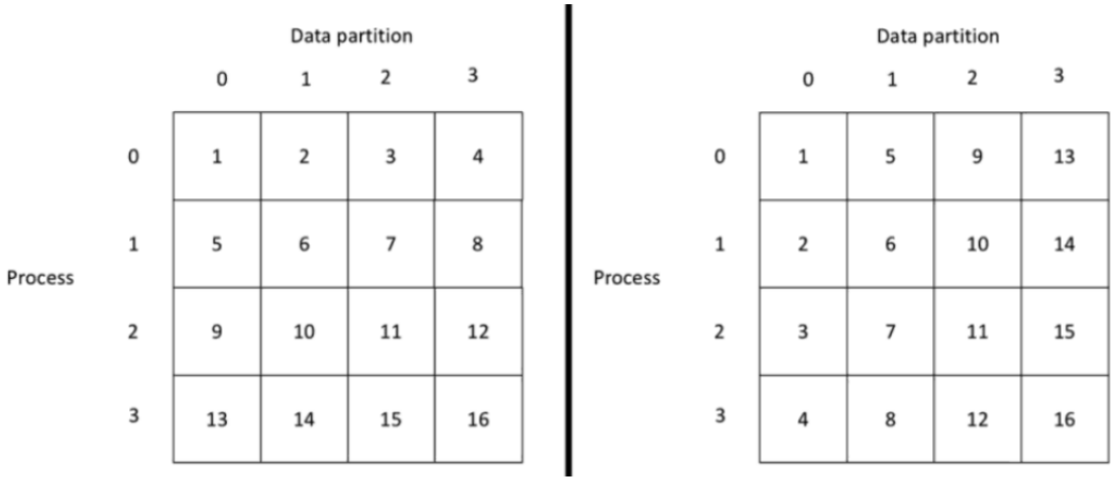
UNIVERSIDAD NEBRIJA

INDICE

Ejercicio 1

Se pide:

Implementar un programa que realice la transposición de la matriz inicial mostrada en la figura (parte izquierda).



Para esta práctica tenemos que interiorizar la función MPI_Alltoall

Esta función se define por esta estructura:

```
1. int MPI_Alltoall(const void* buffer_de_envio,
2.                  int número_de_elementos_que_envía_cada_proceso,
3.                  MPI_Datatype tipo de dato de envio,
4.                  void* buffer_de_recepción,
5.                  int
número_de_elemtos_en_el_mensaje_de_recepción_de_cada_proceso,
6.                  MPI_Datatype tipo_de_dato_de_recepción,
7.                  MPI_Comm comunicador);
```

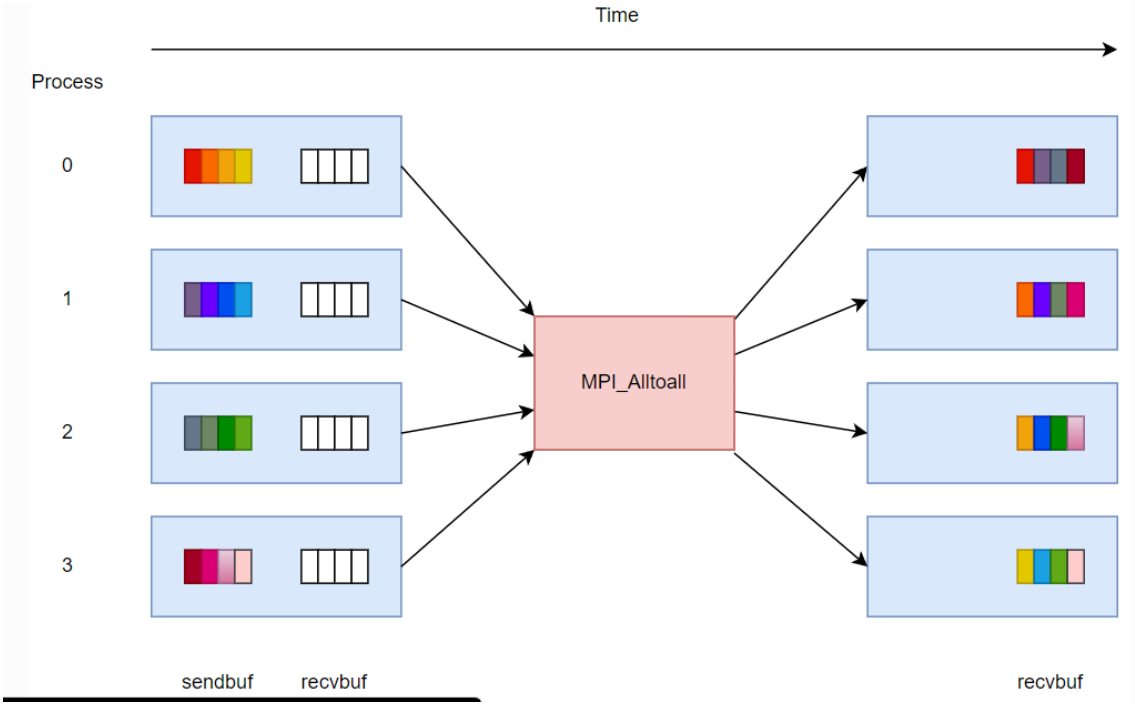
Lo que esta función es que todos los elementos de cada proceso se distribuyen entre todos los procesos.

Cada paquete es enviado al proceso que equivale su posición.

Para un ejercicio complejo con arrays, exponemos este ejemplo si el proceso 0 tiene 3 paquetes

Siguiendo las normas de las posiciones en los arrays:

- El paquete en la posición 0- >Proceso 0
- El paquete en la posición 1 ->Proceso 1
- El paquete en la posición 2 -> Proceso 2



Código

Primero declaramos las variables que vamos a utilizar.

```
int rank, size;
```

- Rank: valor del número de proceso.
- Size: El número de procesos totales.

```
int send[4], recv[4];
```

- Send: Array de 4 elementos que recogerá los números que va a enviar.
- Recv: Arrray de 4 elementos que recogerá los números que va a recibir.

Como controlador para la matriz, declaramos una sentencia en la que obligamos al usuario a generar 4 procesos obligatoriamente.

```
if (size == 4)
```

En caso de no usar 4 procesos, de diremos al usuario que deben ser 4 procesos obligatoriamente.

```
}
else
{
    printf("El numero de procesos debe ser 4.\n");
}
```

En caso de ser 4 procesos, rellenamos cada proceso con sus respectivos valores.

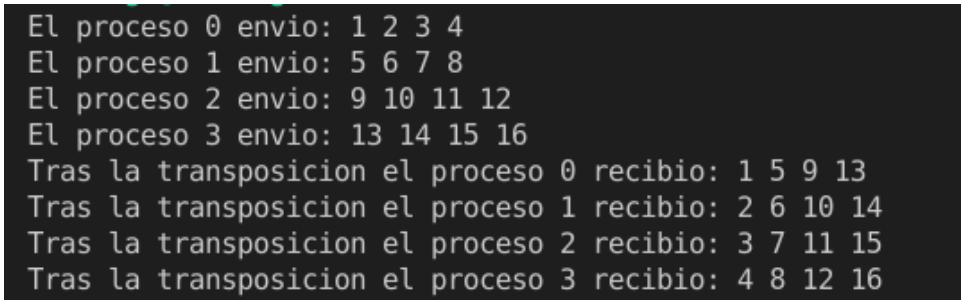
```
    for (int i = 0; i < size; i++)
    {
        send[i] = (i + 1) + rank * size;
    }
```

Y ejecutamos la función MPI_ALLtoall.

```
MPI_Alltoall(&send, 1, MPI_INT, &recv, 1, MPI_INT, MPI_COMM_WORLD);
```

Del array de envío, enviamos un sólo paquete a cada proceso de tipo integer, y los datos que recibamos se guardarán en el array “recv”. Vamos a recibir un solo elemento de cada proceso de tipo int, y declaramos el comunicador.

Salida por pantalla



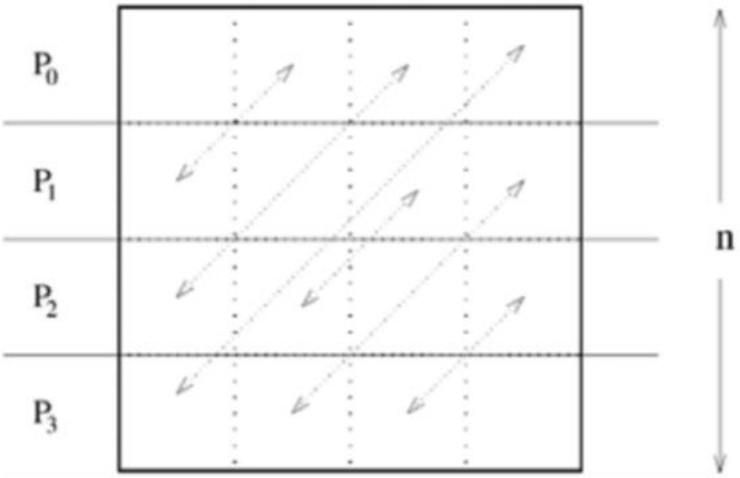
Las 4 primeras líneas representan el array original y las 4 últimas líneas representan la matriz traspuesta.

Ejercicio 2

Se pide:

Implementar un programa donde se resuelva el problema de la transposición de matrices cuadradas de dimensión arbitraria NxN (siendo N múltiplo del número de P de procesos). La matriz se dividirá entre P procesos (p<<N) que la inicializarán, de tal modo que cada elemento sea único (para ello usar la posición de la fila y la columna y el número de rank, como, por ejemplo: 1000 * i + j + n/p * rank).

Para ello, distribuir A y B por columnas (o filas), entre los procesos y haciendo uso de las operaciones colectivas que se consideren oportunas llevar a cabo el proceso de transposición.



Código

Primero declaramos las variables que vamos a utilizar

```
int size, rank;
```

- Rank: valor del número de proceso.
- Size: El número de procesos totales.

Para que el código sea más legible y adaptado a la práctica declaramos las variables

```
int P = size;
int N = 0;
```

- P es el número de procesos
- N es la nueva dimensión que vamos a calcular

Para calcular N, como debe ser cuadrada debemos tener en cuenta que debemos cumplir la condición.

- Que el número sea divisible por el número de procesos y su modulo(resto) de 0.

Por tanto:

```
for (int i = 1; i % P != 0 || i == P; i++)
{
   N = i + 1;
}
```

Aumentamos el valor de N cumpliendo:

- Modulo entre I y P no sea 0
- Que el valor de I sea igual de P

Mientras que una de estas 2 condiciones se cumpla, el “for” seguirá funcionando, generando que si tenemos un número de procesos P=2 => N=4.

También debemos declarar las variables que van a almacenar los datos de envío y recepción.

```
int send[N][P], recv[N][P];
```

- Send: Array de datos de envió
- Recv: Array de datos de recepción

Rellenamos la matriz send.

```
for (int i = 0; i < N; i++)
  {
     printf("\nEl proceso %d ha enviado:", rank);
     for (int j = 0; j < P; j++)
     {
        send[i][j] = 1000 * i + j + N / P * rank;
        printf("%d ", send[i][j]);
     }
  }
```

Y llamamos a la función MPI_Alltoall, explicada anteriormente.

```
MPI_Alltoall(&send, P * P, MPI_INT, &recv, P * P,
      MPI_INT, MPI_COMM_WORLD);
```

```
void transpuesta(int *a, int n)
{
   int i, j;
   int ij, ji, l;
   double tmp;
   ij = 0;
   l = -1;
   for (i = 0; i < n; i++)
   {
      l += n + 1;
      ji = l;
      ij += i + 1;
      for (j = i + 1; j < n; j++)
      {
         tmp = a[ij];
         a[ij] = a[ji];
         a[ji] = tmp;
         ij++;
         ji += n;
      }
   }
}
```

Para la salida por consola, utilizamos la función proporcionada en clase para terminar de ordenar la matriz.

Ejemplo

Para este ejemplo utilizaremos una cantidad de procesos tal que P=2.

1º Calcular N

Seguín nuestro for, mientras (i%P ¡=0 || I==P), este seguirá funcionando. Por tanto

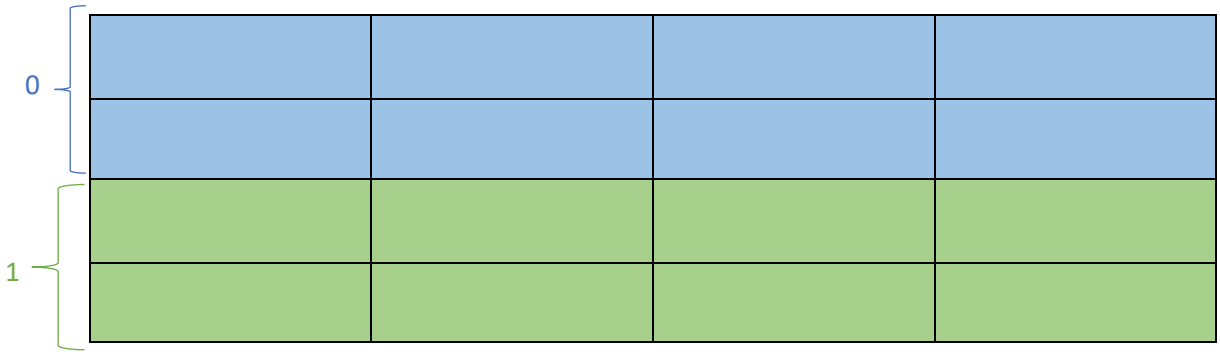
1. i=1, P=2
    a. i%P =1 se cumple => N=i+1=>1+1=2
2. i=2, P=2
    a. i%P=0
    b. i==P se cumple => N=i+1 => 2+1=3
3. i=3, P=2
    a. i%P =1 se cumple => N=i+1=>3+1=4
4. i=4, P=2
    a. i%P=0 no se cumple
    b. i no es igual a P
        ⇨ Salimos del for

Por tanto, ya tenemos P=2 y N=4, que es lo mismo que decir que tenemos un array de 4x4 ya que tiene que ser una matriz cuadrada, pero las dimensiones del array se dividen entre los procesos.

Como tenemos 2 procesos, a cada proceso le corresponden 2 arrays.

Proceso 0 =2 Array

Proceso 1= 2 Aray



Rellenamos la matriz

Que tendrá este aspecto:

(Tener en cuenta que nuestro programa lo dividimos por columnas)

0	1	2	3
1000	1001	1002	1003
2000	2001	2002	2003
3000	2001	3002	3003

Y cuando ejecutamos la impresión, nos devuelve la matriz traspuesta.

0	1000	2000	3000
1	1001	2001	3001
2	1002	2002	3002
3	1003	2003	3003

Consola_Envio

```
El proceso 0 envia:
Rank = 0 Fila= 0  Columna=0  =0
Rank = 0 Fila= 0  Columna=1  =1
El proceso 0 envia:
Rank = 0 Fila= 1  Columna=0  =1000
Rank = 0 Fila= 1  Columna=1  =1001
El proceso 0 envia:
Rank = 0 Fila= 2  Columna=0  =2000
Rank = 0 Fila= 2  Columna=1  =2001
El proceso 0 envia:
Rank = 0 Fila= 3  Columna=0  =3000
N=42 3 1002 1003 2002 2003 3002 3003
El proceso 1 envia:
Rank = 1 Fila= 0  Columna=0  =2
Rank = 1 Fila= 0  Columna=1  =3
El proceso 1 envia:
Rank = 1 Fila= 1  Columna=0  =1002
Rank = 1 Fila= 1  Columna=1  =1003
El proceso 1 envia:
Rank = 1 Fila= 2  Columna=0  =2002
Rank = 1 Fila= 2  Columna=1  =2003
El proceso 1 envia:
Rank = 1 Fila= 3  Columna=0  =3002
Rank = 0 Fila= 3  Columna=1  =3001Rank = 1 Fila= 3  Columna=1  =3003
```

Consola Recepción

```
El proceso 1 ha recibido:
Rank = 1 Fila= 0  Columna=0  =2000
Rank = 1 Fila= 0  Columna=1  =3000
El proceso 1 ha recibido:
Rank = 1 Fila= 1  Columna=0  =2001
Rank = 1 Fila= 1  Columna=1  =3001
El proceso 1 ha recibido:
Rank = 1 Fila= 2  Columna=0  =2002
Rank = 1 Fila= 2  Columna=1  =3002
El proceso 1 ha recibido:
Rank = 1 Fila= 3  Columna=0  =2003
N=40 1 1000 1001 2000 2001 3000 3001
El proceso 0 ha recibido:
Rank = 0 Fila= 0  Columna=0  =0
Rank = 0 Fila= 0  Columna=1  =1000
El proceso 0 ha recibido:
Rank = 0 Fila= 1  Columna=0  =1
Rank = 0 Fila= 1  Columna=1  =1001
El proceso 0 ha recibido:
Rank = 0 Fila= 2  Columna=0  =2
Rank = 0 Fila= 2  Columna=1  =1002
El proceso 0 ha recibido:
Rank = 0 Fila= 3  Columna=0  =3
Rank = 1 Fila= 3  Columna=1  =3003Rank = 0 Fila= 3  Columna=1  =1003
```