Arquitectura de Computadores

Grado en Ingeniería Informática

Práctica 4

UNIVERSIDAD NEBRIJA

Resolución de problemas mediante el modelo de comunicación colectivo

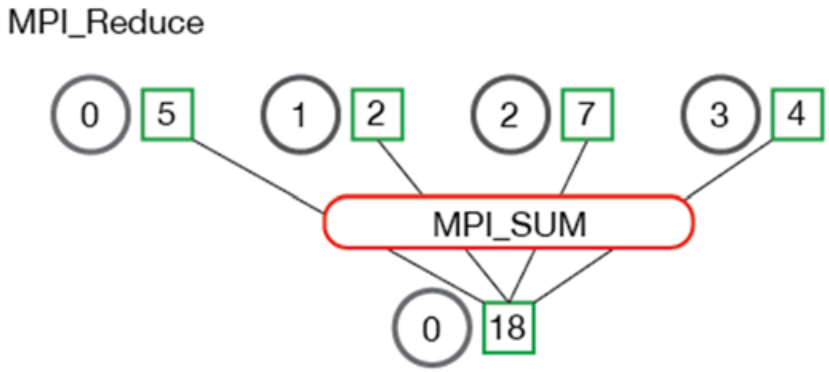
Introducción

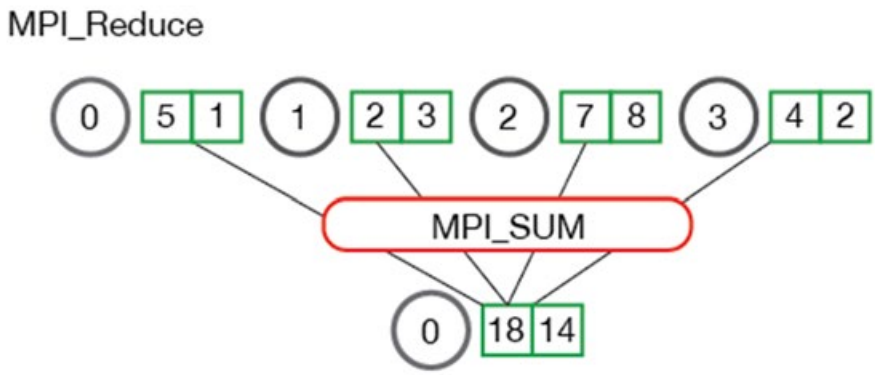
La presente práctica continúa profundizando en el modelo de comunicación colectivo que fue presentado en la práctica anterior.

Reduce

Realiza una operación de reducción global (como puede ser calcular el máximo, la suma, hacer un AND lógico, etc) sobre cada uno de los miembros del grupo. Se puede notar que únicamente se especifica una vez el número de datos y el tipo de datos, esto es porque el número de datos es el mismo tanto en la emisión como en la recepción, al igual que el tipo. Si el número de datos es mayor que uno, la operación de reducción se realiza uno a uno cada elemento con su posición correspondiente (es decir, el elemento 1 de envío se reduce con cada elemento en la posición 1 de cada sendbuf).

```
MPI_Reduce( void* send_data, void* recv_data, int count, MPI_Datatype
datatype, MPI_Op op, int root, MPI_Comm communicator)
```

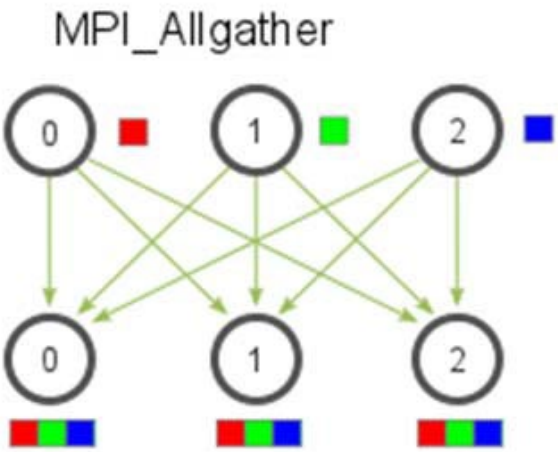




MPI_Allgather

Dado un conjunto de datos distribuidos entre los diferentes nodos del comunicador, todos los datos se recogerán en todos los nodos. `MPI_Allgather` es un `MPI_Gather` seguido por un `MPI_Bcast`.

```
MPI_Allgather( void* send_data, int send_count, MPI_Datatype send_datatype,
void* recv_data, int recv_count, MPI_Datatype recv_datatype, MPI_Comm
communicator)
```




```
rank    send buf                        recv buf
----    --------                        --------
 0      a,b,c          MPI_Allgather    a,b,c,A,B,C,#,@,%
 1      A,B,C         ----------------> a,b,c,A,B,C,#,@,%
 2      #,@,%                           a,b,c,A,B,C,#,@,%
```

MPI_Alltoall

Funciona como MPI_Scatter y MPI_Gather combinados. El búfer de envío en cada proceso se divide como en MPI_Scatter y luego cada columna de fragmentos se recopila mediante el proceso respectivo, cuyo rango coincide con el número de la columna de fragmentos.

```
int MPI_Alltoall(const void *send_data, int send_count, MPI_Datatype
send_datatype, void *recv_data, int recv_count, MPI_Datatype recv_datatype,
MPI_Comm communicator)
```

```
rank    send buf                        recv buf
----    --------                        --------
 0      a,b,c          MPI_Alltoall     a,A,#
 1      A,B,C         ----------------> b,B,@
 2      #,@,%                           c,C,%

(a more elaborate case with two elements per process)

rank    send buf                        recv buf
----    --------                        --------
 0      a,b,c,d,e,f    MPI_Alltoall     a,b,A,B,#,@
 1      A,B,C,D,E,F   ----------------> c,d,C,D,%,$
 2      #,@,%,$,&,*                     e,f,E,F,&,*
```

Objetivos

En esta práctica se aprenderá el concepto de comunicación colectivo entre procesos de MPI.

Los objetivos fijados son los siguientes:

- Compilación de programas MPI.
- Ejecución de programas en varios procesos de forma paralela.
- Estructura de un programa MPI.
  - Iniciar y finalizar el entorno MPI con MPI_Init y MPI_Finalize.
  - Identificador (rango) del proceso con MPI_Comm_rank.
  - Consultar el número de procesos lanzados con MPI_Comm_size.
- Aprender operaciones de comunicación colectivas:
  - MPI_Allgather
  - MPI_Alltoall

Compilación y ejecución de programas MPI

- Compilación: `mpicc codigo_fuente.c -o ejecutable`
- Ejecución: `mpirun –np <number> ejecutable`

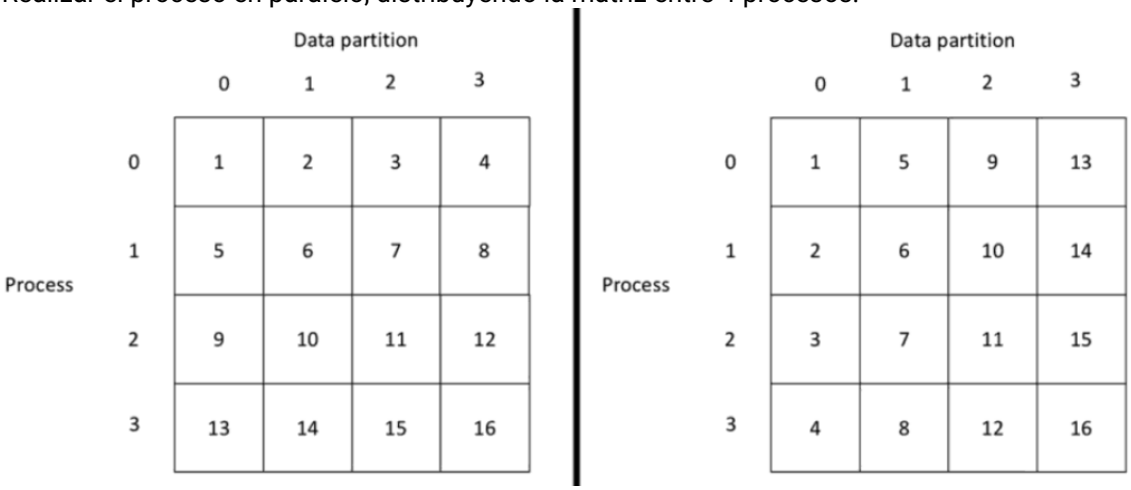
Ejercicio 1 (1 punto)

Crear un ejemplo sencillo de uso de cada primitiva explicada en esta práctica (MPI_Reduce, MPI_Allgather y MPI_Alltoall) y utilizarlo para explicar el funcionamiento de cada una de ellas. Estos ejemplos deben funcionar correctamente con cualquier número de procesos.

Ejercicio 2 (3 puntos)

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

Realizar el proceso en paralelo, distribuyendo la matriz entre 4 procesos.



Ejercicio 2 (6 puntos)

Conversión a decimal un número binario.
Haciendo uso de las funciones explicadas en este guion, implementar un programa donde los procesos calculan en paralelo la representación decimal de un número binario:

- El número de procesos con el que se lance la aplicación será igual que el número de bits del número binario.

- Al iniciar la aplicación cada proceso genera un valor aleatorio (0 o 1) y se lo envía al resto. Uno de los procesos deberá imprimir por pantalla el número que se va a pasar a decimal.
- Una vez que todos los procesos tienen el dato, hacen las operaciones necesarias para calcular el valor decimal **de forma paralela**.
- Solo un proceso conoce el resultado final en decimal y lo imprime por pantalla.

Ejemplo de salida por consola con 2, 6 y 10 procesos:

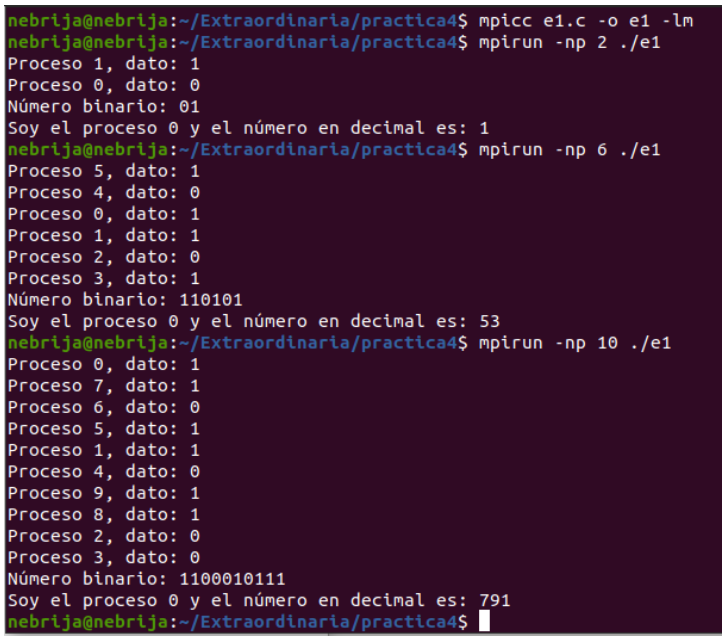


Fig 1: Salida por consola ejercicio 2