Arquitectura de Computadores

Grado en Ingeniería Informática

Práctica 3

UNIVERSIDAD NEBRIJA

Fundamentos básicos de paralelización: Modelo de comunicación colectivo

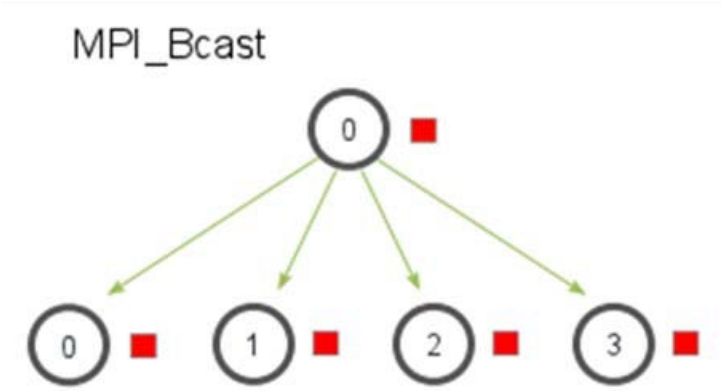
Introducción

La comunicación colectiva involucra a todos los procesos de un comunicador. Se define un proceso maestro y procesos esclavos.
Mediante las funciones de comunicación colectiva el nodo maestro puede enviar datos a los esclavos y recoger los datos que le envían los esclavos. Estas funciones son `MPI_Bcast`, `MPI_Gather` y `MPI_Scatter`.

Broadcast

El concepto de broadcast se asocia tradicionalmente a envíos programados desde un emisor a todos los posibles receptores. Se puede traducir como difusión. MPI proporciona una función para poder realizar este tipo de envíos en una única llamada en la que el emisor envía el mismo dato a todos los procesos pertenecientes al comunicador.

```
MPI_Bcast(void*buf, int count, MPI_Datatype type, int source, MPI_Commcomm)
```

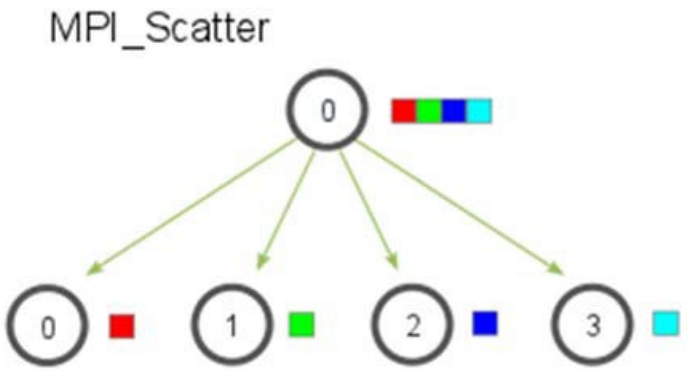


Scatter

De forma similar a la función de broadcast, un nodo designado como root envía datos al resto de nodos del comunicador. La diferencia principal consiste en que los datos enviados a cada nodo son distintos. La función es la siguiente:

```
MPI_Scatter(void*bufSource, int count, MPI_Datatype type, void *bufDest, int count, MPI_Datatype type, int dest, MPI_Commcomm)
```

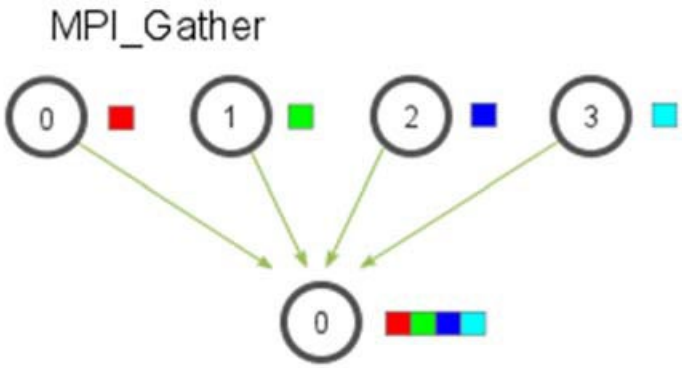
- `send_data`, es el array de datos que están en el root.
- `send_count` y `send_datatype`, definen cuantos elementos y tipo, que serán enviados a cada nodo.
  - Si `send_count` es uno y `send_datatype` es `MPI_INT`, entonces el proceso 0 coge el primer dato entero del array, el proceso 1 coge el segundo entero, y así sucesivamente.
  - Si `send_count` es dos, entonces el proceso 0 coge los dos primeros enteros, el proceso 1 coge el tercer y cuarto datos, y así sucesivamente.
  - En la práctica, `send_count` es definido por el ratio entre el número total de datos y el número de procesos.
- `recv_data`, `recv_count` y `recv_datatype` son idénticos con respecto a los de envío.



Gather

Es la función inversa a Scatter. Los distintos nodos del comunicador envían sus datos a un solo nodo. La función es la siguiente:

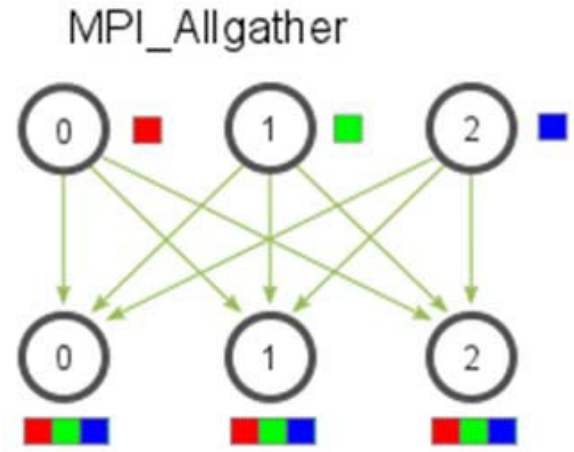
```
MPI_Gather(void*bufSource, int count, MPI_Datatype type, void *bufDest, int count, MPI_Datatype type, int dest, MPI_Commcomm)
```



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`.

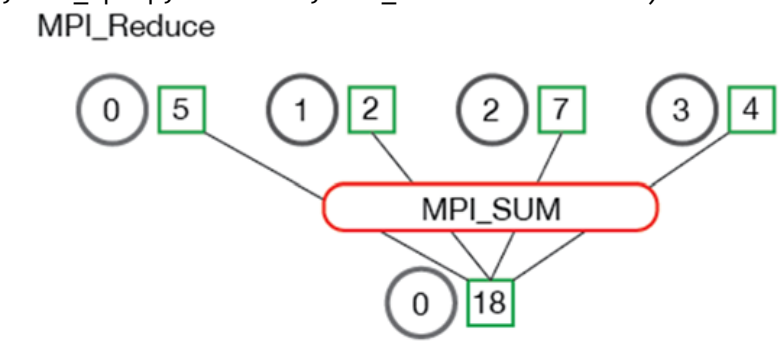
```
MPI_Allgather( void* send_data, int send_count, MPI_Datatype send_datatype, void* recv_data, int recv_count, MPI_Datatype recv_datatype, MPI_Comm communicator)
```

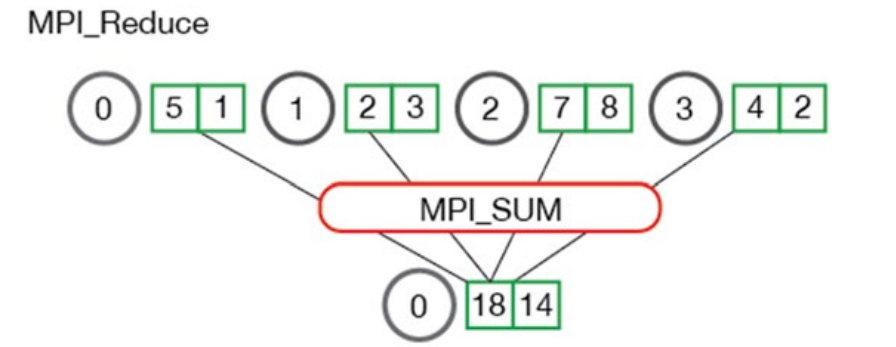


Reduce

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)
```





Objetivos

En esta práctica se aprenderá el concepto de comunicación colectiva 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_Bcast
  - MPI_Gather
  - MPI_Scatter
  - MPI_Reduce

Compilación y ejecución de programas MPI

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

Entregables

- Memoria
- Código fuente de los siguientes ejercicios:

Ejercicio 1 (1 punto)

Parte 1: Explicar en qué consiste la comunicación colectiva, las primitivas que hay en MPI y las ventajas y desventajas de este tipo de comunicación.

Parte 2: Implementar un programa donde el **nodo 0** inicializa una variable con la frase "Hola mundo" lo envía al resto de nodos del comunicador.

Cada proceso deberá imprimir por consola:

"Soy el proceso X y he recibido el dato Y"

NOTA: Al inicio del programa, el contenido de la variable solo lo conoce el proceso 0.

Ejercicio 2 (2 puntos)

Implementar un programa donde el nodo 0 inicializa un array unidimensional asignando a cada valor su índice. Este array es dividido en partes, donde cada una de ellas será mandada a un proceso/nodo diferente. Después de que cada nodo haya recibido su porción de datos, los actualiza sumando a cada valor su rank. Por último, cada proceso envía su porción modificada al proceso root.

(Hacerlo para que el número de datos total (N) sea múltiplo del número de procesos).

Ejercicio 3 (2 puntos)

Implementar un programa donde cada proceso inicializa un array de una dimensión, asignando a todos los elementos el valor de su rank+1. Después el proceso 0 (root) ejecuta dos operaciones de reducción (suma y después producto) sobre los arrays de todos los procesos.

Ejercicio 4 (5 puntos)

Crear un programa que, **haciendo uso de las funciones de comunicación colectiva** que se considere necesario, calcule el factorial del número total de procesos. Esto es, si se ejecuta la aplicación con 4 procesos, uno de los procesos tiene que mostrar el resultado 24, si se ejecuta con 5, el resultado sería 120.

Solo un proceso debe conocer el resultado final y deberá imprimir por pantalla:
"Soy el proceso X y el factorial de Y es Z"