Índice General

1. Introducción e instalación.
2. Comunicación punto a punto bloqueante.
3. Comunicación punto a punto no bloqueante.
4. Llamadas colectivas.
   - Barrier, Broadcast, gather, scatter.
5. Llamadas colectivas.
   - Todos con todos, reduction.
6. Tipos de datos / Comunicadores.

4. Llamadas colectivas

1. Introducción.
2. Sincronización (barreras).
3. Difusión (Broadcast).
4. Recolección (Gather).
5. Dispersión (Scatter).
6. Todos a todos (all to all).
7. Reducción (Reduction).

4.1 Introducción

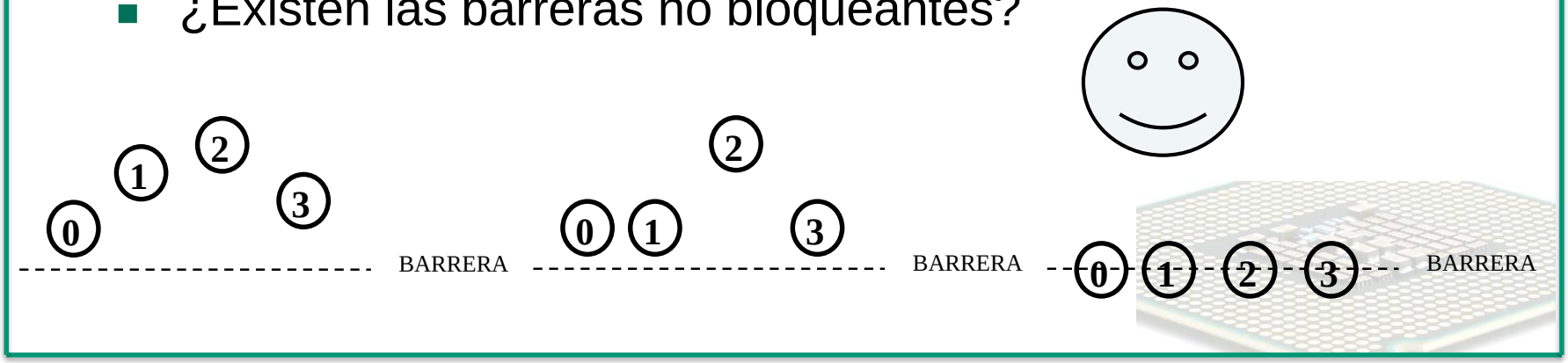
- Una comunicación puede involucrar a varios procesos dentro de un comunicador. Esto se puede realizar con una serie de comunicaciones punto a punto o con una llamada colectiva.
- Se suele hacer con la última por dos razones:
  - **Comodidad**.
  - **Rendimiento**. La comunicación colectiva está optimizada en las implementaciones de MPI.
- Al contrario de cómo se podría pensar, todos los procesos involucrados en una comunicación colectiva hacen la **misma** llamada, ya sean emisores o receptores (ver más adelante*).

4.1 Introducción

- Simplificaciones en las llamadas colectivas:
  - La cantidad de datos que se envía, es la **misma** que se recibe.
  - Las comunicaciones son **bloqueantes**.
  - Por consistencia, los **argumentos** en las llamadas son los mismos que en las punto a punto, salvo que se prescinde de la **etiqueta**.
  - Sólo existe un **modo** de comunicación, la llamada puede terminar cuando su participación en la comunicación global haya terminado. Es decir, cuando se pueda usar el buffer utilizado por otra llamada, sería análogo al modo **básico**.
  - Cualquier otro comportamiento no será portable.

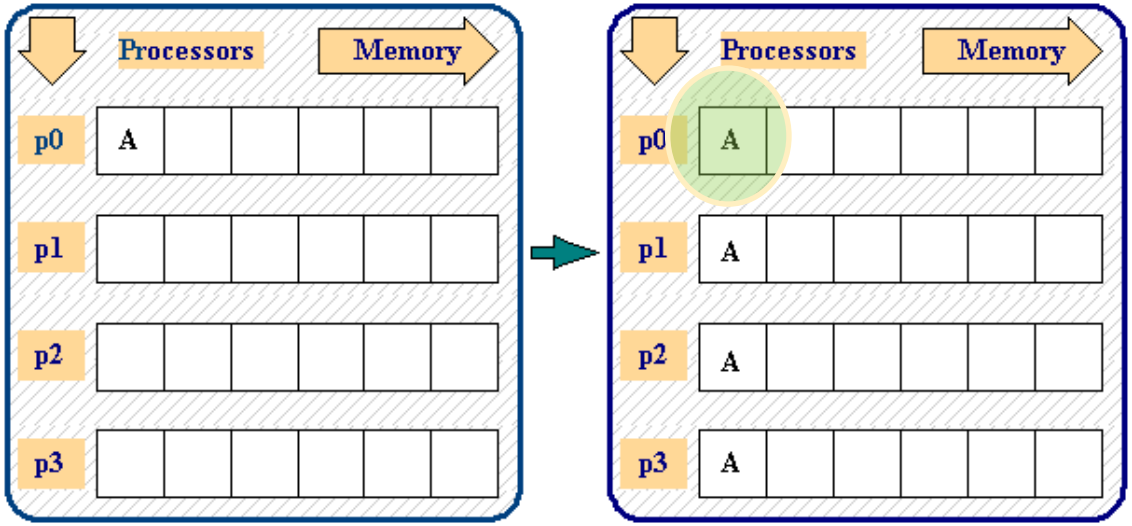
4.2 Sincronización (barreras)

- La operación de sincronización o barrera no implica comunicación (transmisión de datos), sólo bloqueará a los procesos del comunicador hasta que todos hayan pasado por la llamada.
  - `int MPI_Barrier(MPI_Comm comm);`
- Suele emplearse para terminar una etapa y asegurarse de que todos los procesos empiecen la siguiente al mismo tiempo.
- Análogo a las señales o las barrier de pthreads…
- ¿Existen las barreras no bloqueantes?

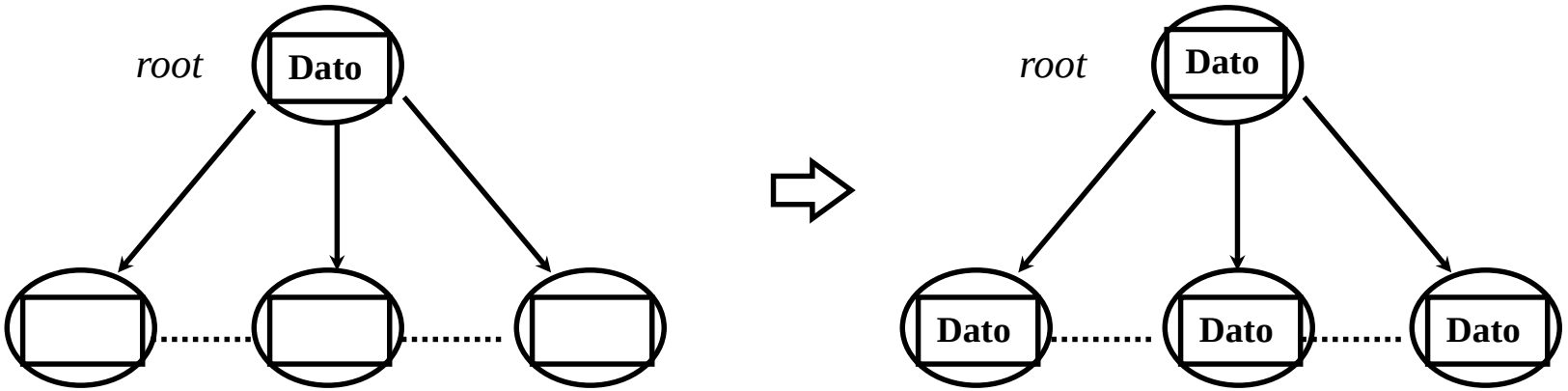


4.3 Difusión (broadcast)

- Es la comunicación colectiva más sencilla, un proceso raíz manda un mensaje a un grupo de procesos definido por el comunicador.
- **Peculiaridad***: Todos los procesos utilizan la misma función, indicando quien es el proceso **raíz** (el que emite el mensaje) e indirectamente quien lo va a recibir.



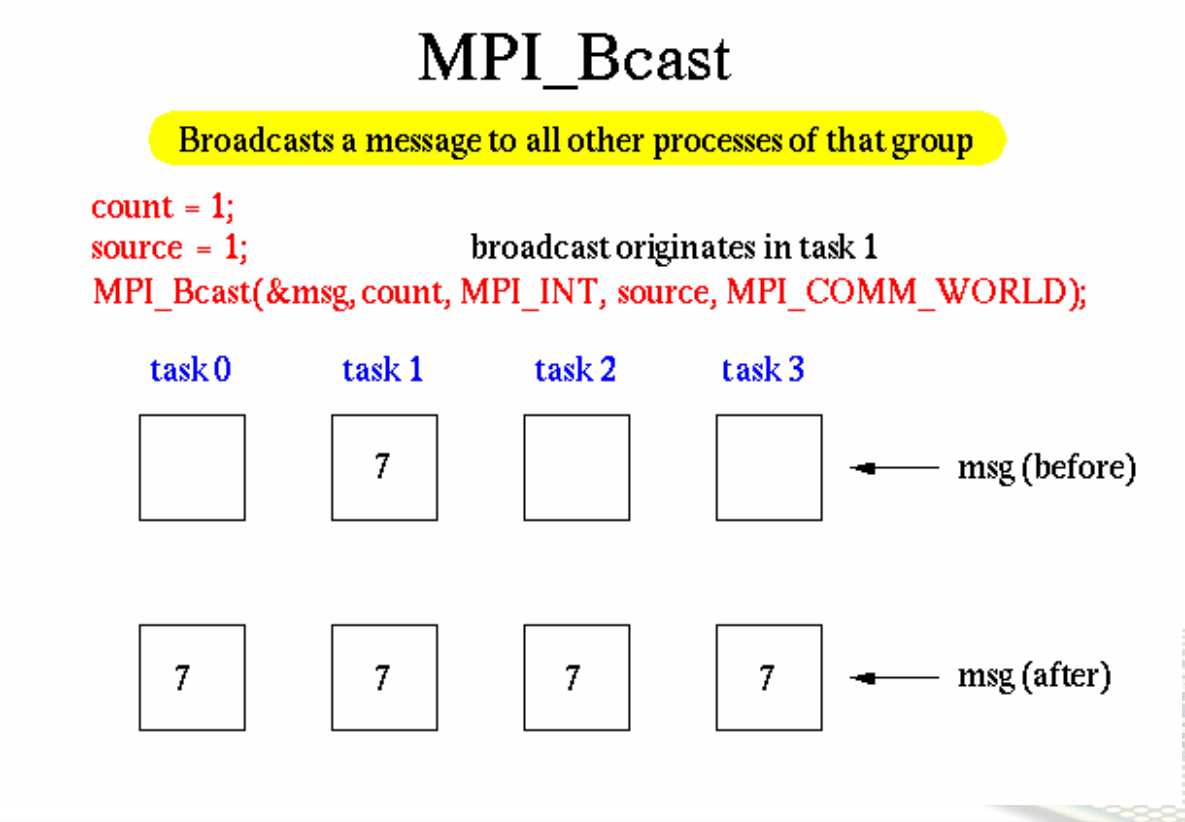
4.3 Difusión (broadcast)



- Todos los procesos deben poner el mismo root.
  - `int` **`MPI_Bcast`**`(void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm);`

4.3 Difusión (broadcast)

- Ejemplo de broadcast, se transmiten 1 entero, el root es la tarea 1:



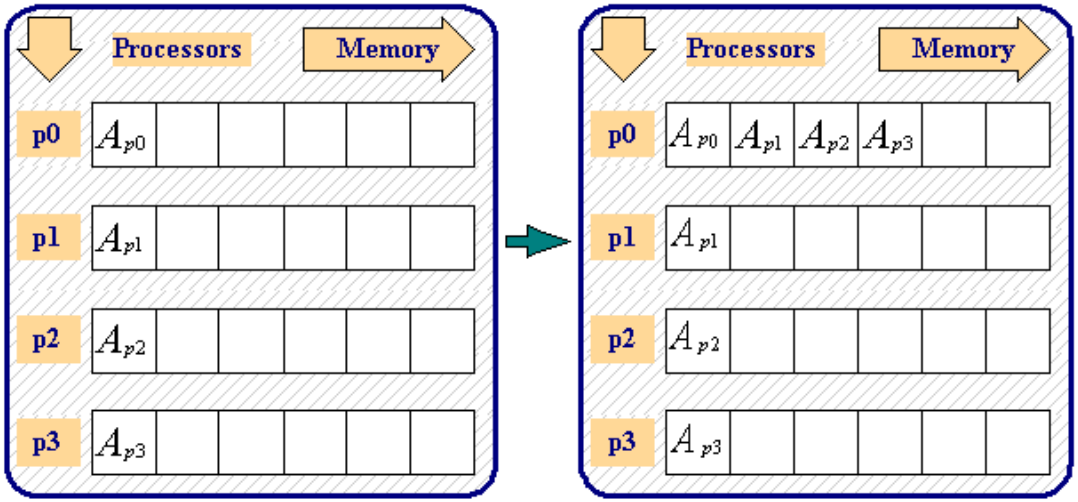
Práctica 3

- Práctica 6: Uso de la difusión y las barreras. Repartiendo datos de entrada
  A. En un programa se toma un dat… eclado en el proceso raíz. Ese dato es trans… rocesos del problema. Em… Todos los procesos pintar… …n y después de ella.
  B. … …a barrera?
  C. C… …blema de los trapezoides para que hagan un br…dcast de los datos de entrada.
  D. ¿Podemos hacer el scanf en todos los procesos?

Ver ejercicio 1

4.4 Recolección (gather)

- Cuando tenemos un proceso raíz que recoge datos de los otros procesos.
- Todos los procesos contribuyen con la misma cantidad de datos.
- Se necesitan dos buffers distintos (de envío y recepción más grande para el root):
  - `int MPI_Gather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm);`

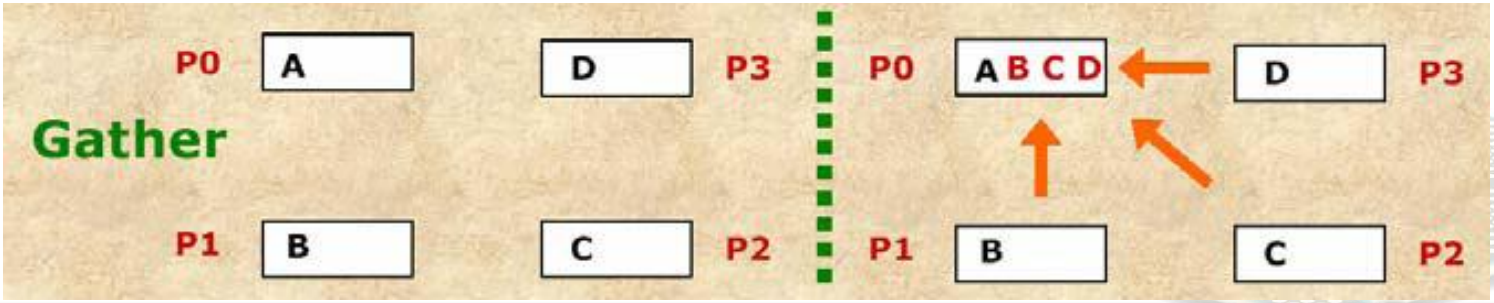


4.4 Recolección (gather)

- Ejemplo de gather, se transmiten 100 enteros:

```
…
MPI_Comm comm;
int gsize,sendarray[100];
int root, myrank, *rbuf;
...
MPI_Comm_rank( comm, &myrank);
if ( myrank == root)
{
   MPI_Comm_size( comm, &gsize);
   rbuf = (int *)malloc(gsize*100*sizeof(int));
}
MPI_Gather( sendarray, 100, MPI_INT, rbuf, 100, MPI_INT,
   root, comm);
…
```

¡Ojo! Todos hacen la misma llamada , pero cada uno tiene una función. De hecho sólo el raíz tiene definido el buffer de recepción. Se ponen los datos a recibir de cada proceso, NO el total.



4.4 Recolección (gather)

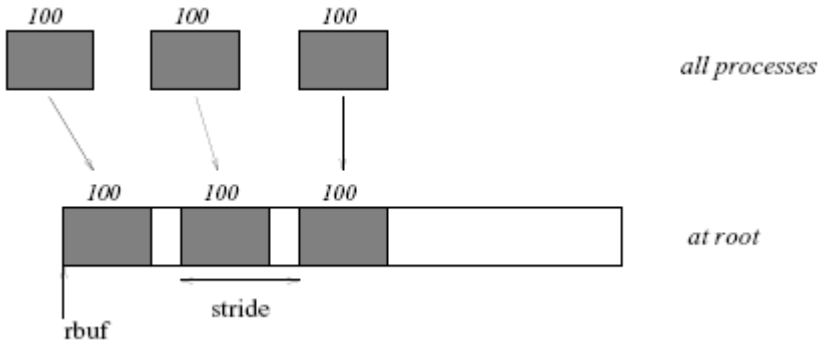
- Una versión más compleja de gather es gatherv que permite enviar distintas cantidades de datos y colocar los datos en el raíz con desplazamientos.
  - `int` **`MPI_Gatherv`**`(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int *recvcounts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm);`
- Esto se hace a través de dos arrays (vectores):
  - Recvcounts: Array de números de datos que se han enviado.
  - Displs: Array de desplazamientos en root en tipo de dato, no en bytes.

4.4 Recolección (gather)

- Ejemplo de gather variable, se transmiten 100 enteros con el mismo desplazamiento:

```
…
MPI_Comm comm;
int gsize,sendarray[100];
int root, *rbuf, stride = lo que sea > 100;
int *displs,i,*rcounts;
...
MPI_Comm_size( comm, &gsize);
rbuf = (int*)malloc(gsize*stride*sizeof(int));
rcounts = (int *)malloc(gsize*sizeof(int));
displs = (int *)malloc(gsize*sizeof(int));
for (i=0; i<gsize; ++i) {
    displs[i] = i*stride;
    rcounts[i] = 100;
}
MPI_Gatherv( sendarray, 100, MPI_INT, rbuf, rcounts, displs, MPI_INT,
root, comm);
…
```

Solo el raíz lo necesita (con if)



Práctica 3

hola_sec.c

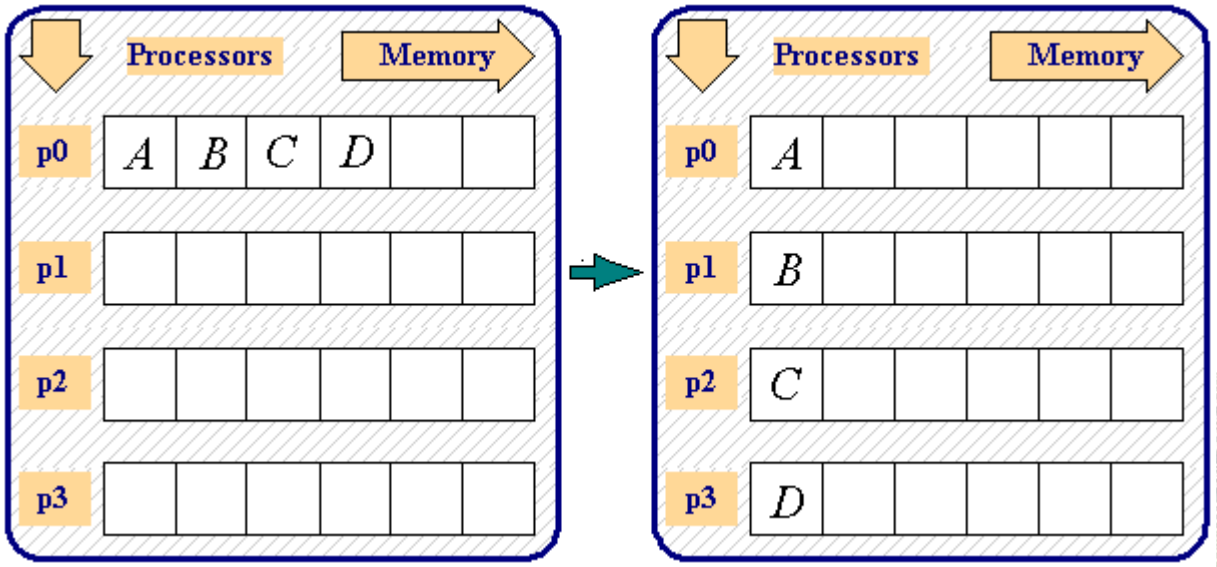
- Práctica 7: Uso de la recolección. Como divido datos.
  A. Adapta el programa “hola mundo” para que todos los procesos menos el raíz construyan su mensaje y lo envíen.
  B. Realiza un programa para el producto de una matriz por un vector utilizando datos divididos por bloques de filas. Para simplificar se supone que las dimensiones son múltiplos del número de procesos. El vector es pasado a cada proceso con el gather. Como ayuda se rellena la matriz como sus índices. Comprueba con el secuencial.
  C. La solución estaría repartida entre los procesos. ¿qué haríamos para tenerla en un proceso?

Ver ejercicio 2

producto_sec.c

4.5 Dispersión (scatter)

- Es la operación inversa a gather, ahora el raíz es el que envía datos que se reparten en partes iguales a los otros procesos.
  - `int` **`MPI_Scatter`**`(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm);`

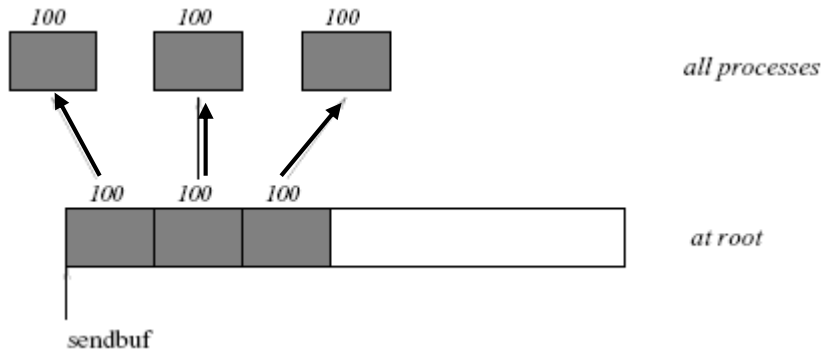


4.5 Dispersión (scatter)

- Ejemplo de scatter, se transmiten 100 enteros:

```
...
MPI_Comm comm;
int gsize,*sendbuf;
int root, rbuf[100];
...
if ( myrank == root)
{
  MPI_Comm_size( comm, &gsize);
  sendbuf = (int *)malloc(gsize*100*sizeof(int));
}
...
MPI_Scatter( sendbuf, 100, MPI_INT, rbuf, 100, MPI_INT,
    root, comm);
...
```

¡Ojo! Todos hacen la misma llamada , pero cada uno tiene una función. De hecho sólo el raíz tiene definido el buffer de envío.



4.5 Dispersión (scatter)

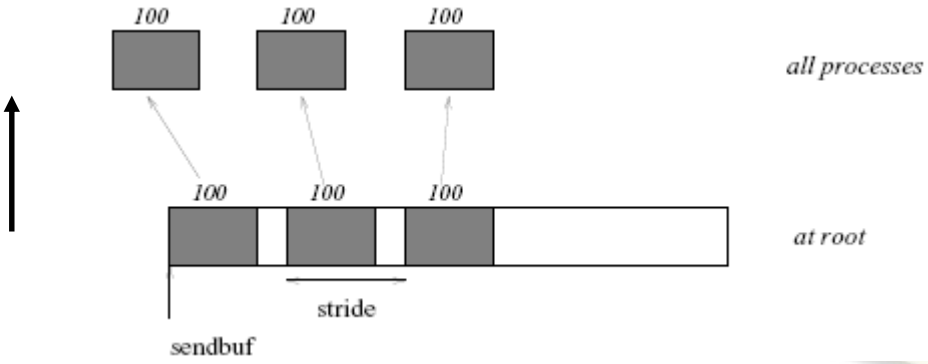
- De la misma forma que con gather existe una forma más compleja para diferentes tamaños de datos la **scatterv**:
  - `int MPI_Scatterv(void* sendbuf, int *sendcounts, int *displs, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm);`
- Se hace a través de dos arrays:
  - Sendcounts: array de número de datos a enviar a cada proceso.
  - Displs: Desplazamientos para enviar esos datos.

4.5 Dispersión (scatter)

- Ejemplo de scatter variable, se transmiten 100 enteros con el mismo desplazamiento:

```
…
MPI_Comm comm;
int gsize,*sendbuf;
int root, rbuf[100], i, *displs, *scounts, stride=algo>100;
...
MPI_Comm_size( comm, &gsize);
sendbuf = (int *)malloc(gsize*stride*sizeof(int));
...
displs = (int *)malloc(gsize*sizeof(int));
scounts = (int *)malloc(gsize*sizeof(int));
for (i=0; i<gsize; ++i)
{
    displs[i] = i*stride;
    scounts[i] = 100;
}
MPI_Scatterv( sendbuf, scounts, displs, MPI_INT, rbuf, 100, MPI_INT,
root, comm);
…
```

Solo el raíz lo necesita (con if)



Práctica 3

- Práctica 8: Uso de la dispersión. Repartiendo datos de entrada.
  - A. Se te da un programa secuencial que escribe 100 dobles en un fichero cuyo nombre se pasa como argumento. Además se comprueba el resultado.
  - B. Realiza un programa [illegible] raíz toma los datos de ese fichero [illegible] y los reparte (scatter) [illegible] un divisor de 100), que los pintarán

Ver ejercicio 3

fiche.c