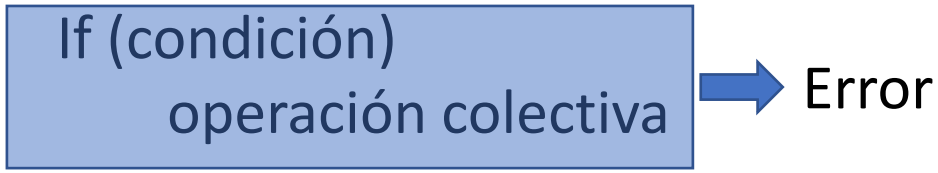
Transparencias MPI

Comunicaciones Colectivas

Operaciones de Comunicación Colectiva

- Involucran a todos los procesos del comunicador: todos ellos deben ejecutar la operación
- Operaciones disponibles:
  - Sincronización (Barrier)
  - Difusión (Bcast)
  - Reparto (Scatter)
  - Recogida (Gather)
  - Multi-recogida (Allgather)
  - Todos a todos (Alltoall)
  - Reducción (Reduce)
  - Prefijación (Scan)
- Algunas de estas operaciones tienen un argumento (root) el cual tiene un papel especial
- Algunas de estas operaciones tienen el sufijo “v”, el cual indica que cantidad de datos de una variable es distinta para los diferentes procesos
- Un error muy común consiste en utilizar una operación colectiva dentro de una sentencia del tipo **if** cuya condición dependa del identificador del proceso, ya que todos los procesos del comunicador deben realizar esa operación esa operación colectiva:



Sincronización

MPI_Barrier(MPI Comm comm)

- Todos los procesos del comunicador se detienen hasta que todos ellos han invocado esta operación
- Ejemplo: medición de tiempos

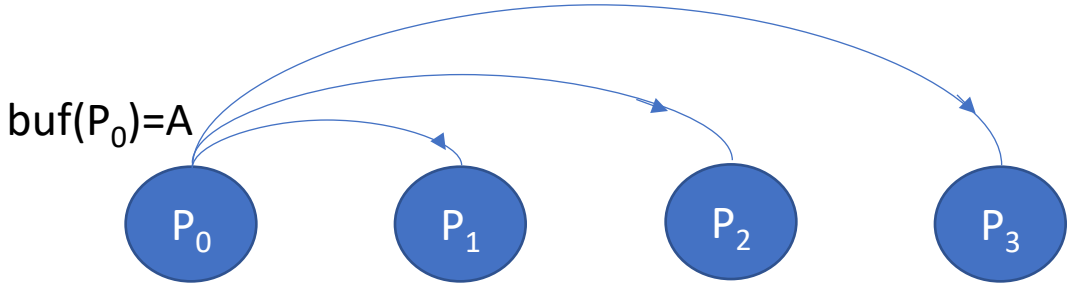
```
MPI_Barrier(comm);
t1 = MPI_Wtime();
/*
...
*/
MPI_Barrier(comm);
t2 = MPI_Wtime();
if (!rank) printf("Tiempo transcurrido: %f s.\n", t2-t1);
```

Difusión

MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int root, MPIComm comm)

- buf=variable local que contiene el dato aportado por el proceso root
- count=número de elementos de la variable buf
- datatype=tipo de datos de buf
- root=proceso que realiza la difusión. Puede ser cualquiera, por ejemplo, el proceso P_0
- Comm= comunicador (habitualmente el comunicador universal MPI_COMM_WORLD)

Ejemplo:



count=|A|=nº datos de A
root=0

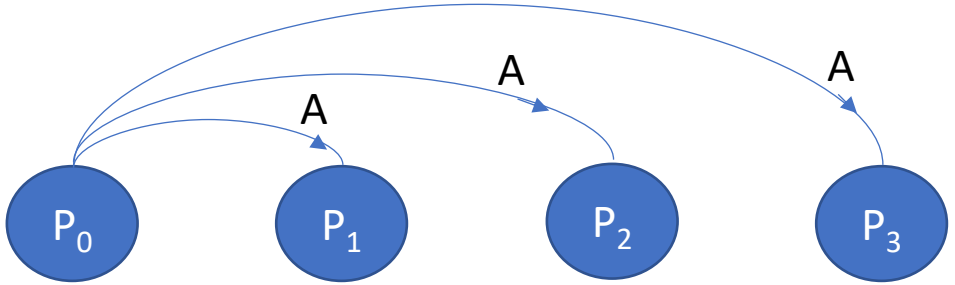
	Antes
buf(P_0)	A
buf(P_1)	
buf(P_2)	
buf(P_3)	

	Después
buf(P_0)	A
buf(P_1)	A
buf(P_2)	A
buf(P_3)	A

Ejercicio 1

Supongamos que en MPI se dispone de **p** procesos y que el proceso P0 contiene una matriz **A** de dimensión MxN de tipo double que debe ser difundida al resto de procesos. Usa la llamada a la función de comunicación colectiva adecuada ¿Cómo podrías sustituir esa comunicación colectiva mediante comunicaciones punto a punto? Calcula el tiempo de comunicaciones.

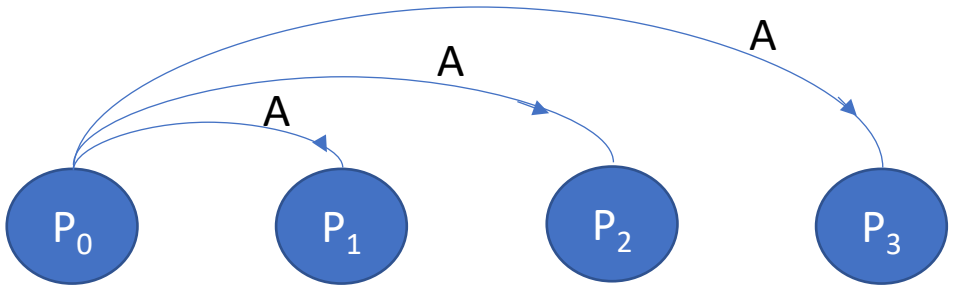
$$A(P_0) \xrightarrow[\text{MPI\_Bcast}]{\text{Difusión}} A(P_1), A(P_2), A(P_3)$$




```
int MPI_Bcast(void *buf, int count,
    MPI_Datatype datatype, int root,
    MPI_Comm comm)
```

MPI_Bcast(A, M*N, MPI_DOUBLE, 0, MPI_COMM_WORLD)

Ejercicio 1




```
int MPI_Send(void *buf, int count,
    MPI_Datatype datatype, int dest, int
    tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count,
    MPI_Datatype datatype, int source, int
    tag, MPI_Comm comm, MPI_Status *status)
```

Implementación usando comunicaciones punto a punto:

```
double A[M][N];
MPI_Init(&argc, &argv);
int id, p;
MPI_Status   stat;
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_rank(MPI_COMM_WORLD, &id);
if (id==0) /*P0 envía la matriz A al resto de procesos*/
    for(i=1; i<p; i++)/*P0 envía la matriz A al resto de procesos*/
        MPI_Send(A, M*N, MPI_DOUBLE, i, 100, MPI_COMM_WORLD);
else /* El resto de los procesos reciben la matriz A */
        MPI_Recv(A, M*N, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD, &stat);
MPI_Finalize();
```

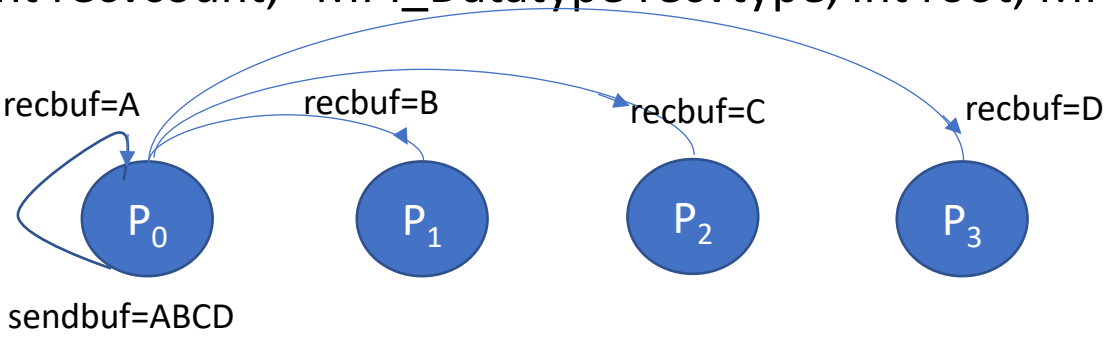
Tiempo de comunicaciones:

$$t_c = (p-1)(t_s + MNt_w)$$

Reparto: El proceso root reparte un vector/matriz entre todos los procesos

MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype,
void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Ejemplo:



sendcount=recvcount=|A|=|B|=|C|=|D|
sendtype= rectype
root=0

	Antes
sendbuf(P_0)	ABCD
sendbuf(P_1)	
sendbuf(P_2)	
sendbuf(P_3)	

	Después
recbuf(P_0)	A
recbuf(P_1)	B
recbuf(P_2)	C
recbuf(P_3)	D

Ejercicio 2

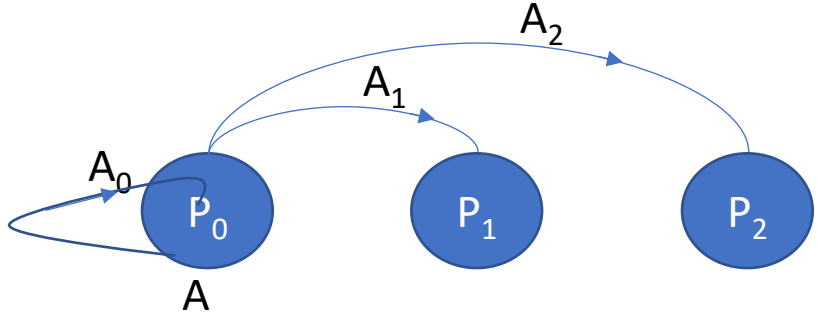
Supongamos que en MPI se dispone de **p** procesos y que el proceso P_0 contiene a una matriz **A** de dimensión MxN de tipo double, la cual debe ser repartida entre todos los procesos. Usa una operación de comunicación colectiva para realizar este reparto ¿Cómo podrías sustituir esa operación mediante comunicaciones punto a punto, suponiendo que el número de procesos **p** divide a **M**? Calcula el tiempo de comunicaciones.

Nota: Las matrices locales quedarán almacenadas en las matrices **Al**. Por comodidad, supondremos que esas matrices son también de dimensión MxN.

$$A(P_0) = \begin{bmatrix} \overbrace{A_0}^{N} \\ \overline{A_1} \\ \overline{A_2} \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \xrightarrow[\text{MPI\_Scatter}]{\text{Reparto}} \begin{bmatrix} \overbrace{Al(P_0) = A_0}^{N} \\ \overline{Al(P_1) = A_1} \\ \overline{Al(P_2) = A_2} \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix}$$

$$k = M / p$$

(k=nº de filas de A_i=nº de filas de $Al(P_i)$, p=número de procesos)




```
int MPI_Scatter(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
```

MPI_Scatter(A, k*N, MPI_DOUBLE, Al, k*N, MPI_DOUBLE, 0, MPI_COMM_WORLD)

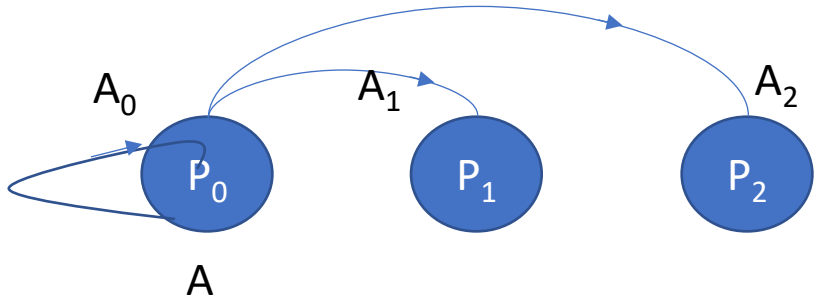
Ejercicio 2

$$A(P_0) = \begin{bmatrix} A_0 \\ A_1 \\ A_2 \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \xrightarrow{\text{MPI\_Scatter}} \begin{bmatrix} Al(P_0) = A_0 \\ Al(P_1) = A_1 \\ Al(P_2) = A_2 \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix}$$

(N sobre $A(P_0)$ y sobre $Al(P_j)$)

$k = M/p$

(k=nº de filas de A_i=nº de filas de $Al(P_j)$, p=número de procesos)



Bloque	1ª fila del bloque
0	0
1	k
2	2k
3	3k
……	…..
i	i*k

Ejemplo: M=12, N=5, p=3→ k=12/3=4

A(P0)=

A[0][0] x x x x x
A[1][0] x x x x x
A[2][0] x x x x x
A[3][0] x x x x x
A[4][0] x x x x x
A[5][0] x x x x x
A[6][0] x x x x x
A[7][0] x x x x x
A[8][0] x x x x x
A[9][0] x x x x x
A[10][0] x x x x x
A[11][0] x x x x x

Al(P0)=
x x x x x
x x x x x
x x x x x
x x x x x

Al(P1)=
x x x x x
x x x x x
x x x x x
x x x x x

Al(P2)=
x x x x x
x x x x x
x x x x x
x x x x x

P0:
for(i=1; i<p; i++)
Enviar bloque de k*N elementos con origen A[i*k][0] a Pi
Copiar sus 1ª k filas de A en Al (k*N elementos consecutivos)
Resto:
Recibir en Al k*N elementos enviados por P0

Ejercicio 2

K=M/p

P0:

for(i=1; i<p; i++)

Enviar el bloque de k*N elementos con origen A[k*i][0] a Pi

Copiar sus 1ª k filas de A en Al (k*N elementos consecutivos)

Resto:

Recibir en Al k*N elementos de P0

```
int MPI_Send(void *buf, int count,
    MPI_Datatype datatype, int dest, int
    tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count,
    MPI_Datatype datatype, int source, int
    tag, MPI_Comm comm, MPI_Status *status)

int MPI_Sendrecv(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, int
    dest, int sendtag, void *recvbuf, int
    recvcount, MPI_Datatype recvtype, int
    source, int recvtag, MPI_Comm comm,
    MPI_Status *status)
```

```
MPI_Init(&argc, &argv);
double A[M][N], Al[M][N];
int id, p;
MPI_Status   stat;
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_rank(MPI_COMM_WORLD, &id);
int k=M/p; /* k= nº de filas de los bloques que se envían/reciben*/
if (id==0){
    for(i=1; i<p; i++)
        MPI_Send(&A[k*i][0], k*N, MPI_DOUBLE,   i, 100, MPI_COMM_WORLD);
    MPI_Sendrecv(&A[0][0], k*N, MPI_ DOUBLE, 0, 100, Al, k*N, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD, &stat);
}
else
    MPI_Recv(Al, k*N, MPI_DOUBLE,   0, 100, MPI_COMM_WORLD, &stat);
MPI_Finalize();
```

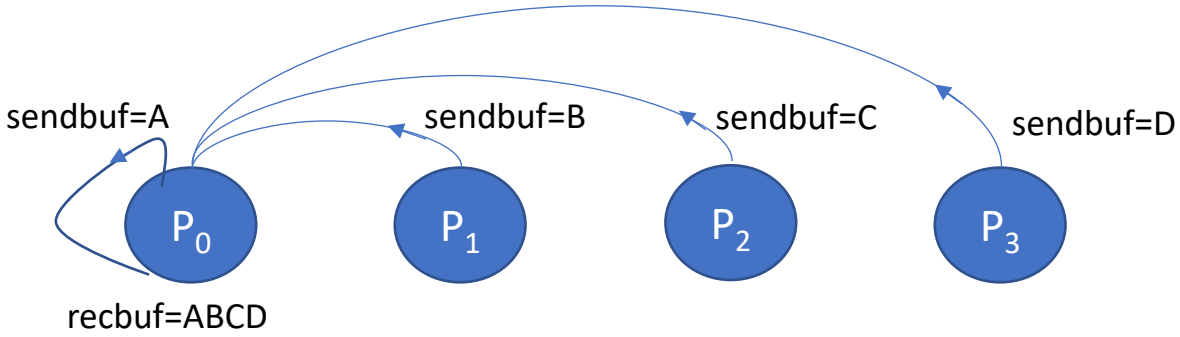
Tiempo de comunicaciones:

$$t_c = (p-1)\left(t_s + \frac{MN}{p} t_w\right)$$

Recogida

MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype sendtype,
void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Ejemplo:



sendcount=recvcount=|A|=|B|=|C|=|D|
sendtype= rectype
root=0

	Antes
sendbuf(P_0)	A
sendbuf(P_1)	B
sendbuf(P_2)	C
sendbuf(P_3)	D

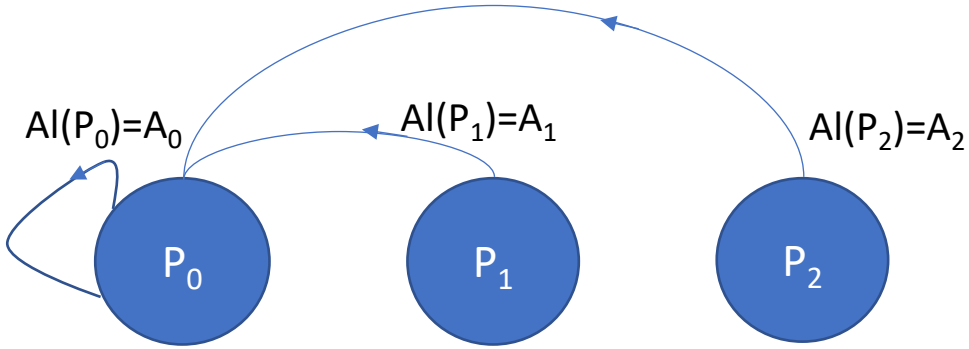
	Después
recbuf(P_0)	ABCD
recbuf(P_1)	
recbuf(P_2)	
recbuf(P_3)	

Ejercicio 3

Supongamos que en MPI se dispone de **p** procesos y una matriz **A** de dimensión MxN de tipo double se encuentra repartida en matrices locales **Al** también de tamaños kxN ¿Qué función de comunicaciones colectivas deberían ejecutar todos los procesos para que P_0 recogiera en **A** todas las matrices? ¿Cómo podrías sustituir esa comunicación colectiva mediante comunicaciones punto a punto, suponiendo que el número de procesos **p** divide a **M**? Calcula el tiempo de comunicaciones.

$$\begin{matrix} N \\ \begin{bmatrix} Al(P_0) = A_0 \\ Al(P_1) = A_1 \\ Al(P_2) = A_2 \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \end{matrix} \xrightarrow{\text{MPI\_Gather}} A(P_0) = \begin{matrix} N \\ \begin{bmatrix} A_0 \\ A_1 \\ A_2 \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \end{matrix}$$

$$k = \frac{M}{p} (p = \text{nº procesos})$$



$$A = \begin{bmatrix} A_0 \\ A_1 \\ A_2 \end{bmatrix}$$

```
int MPI_Gather(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
```

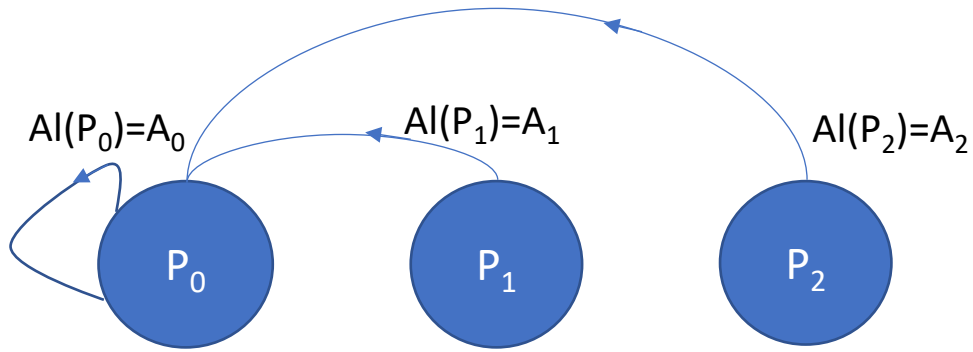
MPI_Gather(Al, k*N, MPI_DOUBLE, A, k*N, MPI_DOUBLE, 0, MPI_COMM_WORLD)

Ejercicio 3

$$\begin{matrix} N \\ \begin{bmatrix} Al(P_0) = A_0 \\ Al(P_1) = A_1 \\ Al(P_2) = A_2 \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \end{matrix} \xrightarrow{\text{RecogidaMPI\_Gather}} A(P_0) = \begin{matrix} N \\ \begin{bmatrix} A_0 \\ A_1 \\ A_2 \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \end{matrix}$$

$$k = \frac{M}{p} (p = \text{nº procesos})$$

$$A = \begin{bmatrix} A_0 \\ A_1 \\ A_2 \end{bmatrix}$$



Ejemplo: M=12, N=6, p=3→ k=12/3=4

Al(P0)=
x x x x x x
x x x x x x
x x x x x x
x x x x x x

Al(P1)=
x x x x x x
x x x x x x
x x x x x x
x x x x x x

Al(P2)=
x x x x x x
x x x x x x
x x x x x x
x x x x x x

A[0][0] x x x x x
A[1][0] x x x x x
A[2][0] x x x x x
A[3][0] x x x x x
A[4][0] x x x x x
A[5][0] x x x x x
A[6][0] x x x x x
A[7][0] x x x x x
A[8][0] x x x x x
A[9][0] x x x x x
A[10][0] x x x x x
A[11][0] x x x x x

Bloque	1ª fila del bloque
0	0
1	k
2	2k
3	3k
……	…..
i	i*k

P_0:

for (i=1; i<p; i++)

Recibir en A[i*k][0] el bloque de k*N elementos de Pi

Copiar Al a partir de la primera posición de A

Resto:

Enviar los k*N elementos de Al a P0

Ejercicio 3

P0:
for (i=1; i<p; i++)
Recibir en A[k*i][0] bloque de kN elementos de Pi
Copiar Al a partir de la primera posición de A
Resto:
Enviar los kN elementos de Al a P0

```
int MPI_Send(void *buf, int count,
    MPI_Datatype datatype, int dest, int
    tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count,
    MPI_Datatype datatype, int source, int
    tag, MPI_Comm comm, MPI_Status *status)

int MPI_Sendrecv(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, int
    dest, int sendtag, void *recvbuf, int
    recvcount, MPI_Datatype recvtype, int
    source, int recvtag, MPI_Comm comm,
    MPI_Status *status)
```

Tiempo de comunicaciones:

$$t_c = (p-1)\left(t_s + \frac{MN}{p} t_w\right)$$

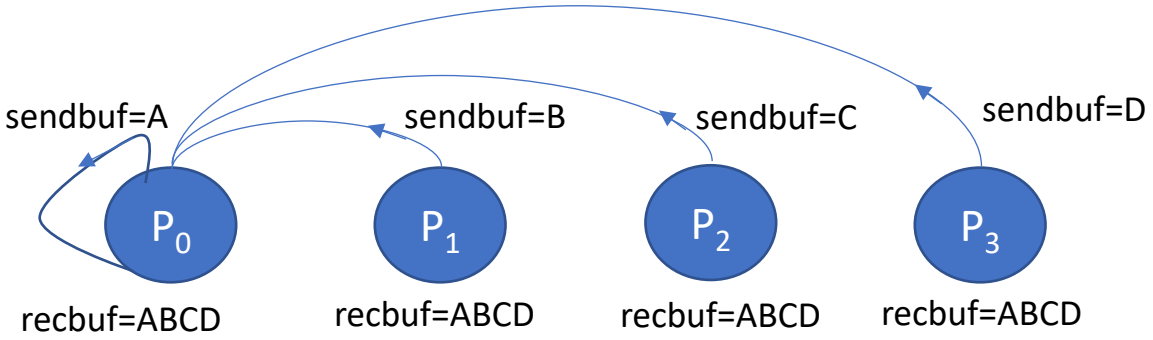
```
MPI_Init(&argc, &argv);
double A[M][N], Al[M][N];
int id, p;
MPI_Status   stat;
MPI_Comm_size(MPI_COMM_WORLD,&p);
MPI_Comm_rank(MPI_COMM_WORLD,&id);
int k=M/p; /* k= nº de filas de los bloques que se envían/reciben*/
if (id==0){
    for(i=1; i<p; i++)
        MPI_Recv(&A[i*k][0], k*N, MPI_DOUBLE, i, 100, MPI_COMM_WORLD, &stat);
    MPI_Sendrecv(Al, k*N, MPI_ DOUBLE, 0, 100, A, k*N, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD, &stat);
}
else
    MPI_Send(Al, k*N, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD);

MPI_Finalize();
```

Multi-recogida

MPI_AllGather(void *sendbuf, int sendcount, MPI_Datatype sendtype,
void *recvbuf, int recvcount, MPI_Datatype, recvtype, MPI_Comm comm)

Ejemplo:



sendcount=recvcount=|A|=|B|=|C|=|D|
sendtype= rectype
root=0

	Antes
sendbuf(P_0)	A
sendbuf(P_1)	B
sendbuf(P_2)	C
sendbuf(P_3)	D

	Después
recbuf(P_0)	ABCD
recbuf(P_1)	ABCD
recbuf(P_2)	ABCD
recbuf(P_3)	ABCD

Ejercicio 4

¿Qué cambiaras en el código que usa comunicaciones colectivas del ejercicio 8 para que la matriz A la tuviesen todos los procesos?

$$\begin{matrix} N \\ \begin{bmatrix} Al(P_0) = A_0 \\ Al(P_1) = A_1 \\ Al(P_2) = A_2 \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \end{matrix} \xrightarrow{\text{MPI\_Allgather}} A(P_0, P_1, P_2) = \begin{matrix} N \\ \begin{bmatrix} A_0 \\ A_1 \\ A_2 \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \end{matrix}$$

$$M = kp(p = \text{nº procesos})$$

MPI_Gather(Al, k*N, MPI_DOUBLE, A, k*N, MPI_DOUBLE, 0, MPI_COMM_WORLD)

MPI_Allgather(Al, k*N, MPI_DOUBLE, A, k*N, MPI_DOUBLE, MPI_COMM_WORLD)

Todos a todos

MPI_Alltoall(void *sendbuf, int sendcount, MPI_Datatype sendtype,
void *recvbuf, int recvcount, MPI_Datatype, recvtype, MPI_Comm comm)

Ejemplo:

sendcount=recvcount=$|A_i|=|B_i|=|C_i|=|D_i|$
sendtype= rectype
root=0

	Antes
sendbuf(P_0)	$A_0A_1A_2A_3$
sendbuf(P_1)	$B_0B_1B_2B_3$
sendbuf(P_2)	$C_0C_1C_2C_3$
sendbuf(P_3)	$D_0D_1D_2D_3$

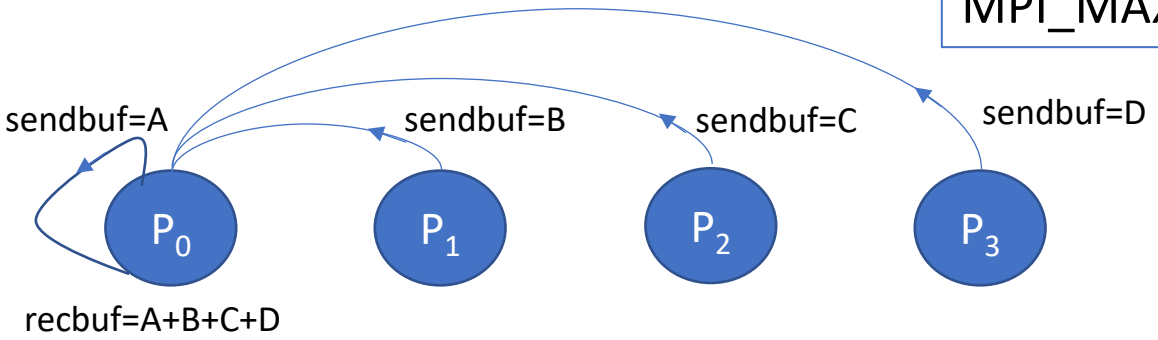
	Después
recbuf(P_0)	$A_0B_0C_0D_0$
recbuf(P_1)	$A_1B_1C_1D_1$
recbuf(P_2)	$A_2B_2C_2D_2$
recbuf(P_3)	$A_3B_3C_3D_3$

Reducción

MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, …..

Ejemplo: Suma



sendcount=|A|=|B|=|C|=|D|
op= MPI_SUM
root=0

	Antes
sendbuf(P_0)	A
sendbuf(P_1)	B
sendbuf(P_2)	C
sendbuf(P_3)	D

	Después
recbuf(P_0)	A+B+C+D
recbuf(P_1)	
recbuf(P_2)	
recbuf(P_3)	

Ejercicio 5 (Suma elementos de un vector)

Realiza una implementación paralela mediante MPI de la función **suma_vec**, la cual tiene como argumento de entrada un vector **v** de dimensión **N** y devuelve la suma de los elementos del vector **v**. Supondremos que inicialmente **v** está almacenada en P0 y que **N** es divisible entre el número de procesos **p**. Calcula el tiempo paralelo.

```
double suma_vec(double v[N]){
    int i;  /*k=nº de filas de los v_i*/
    double sl=0;
    for (i=0; i<N; i++)  /*calcular suma local*/
            sl+=vl[i];
    return sl;
}
```

Solución:

$$v(P_0)=\begin{bmatrix}v_0\\v_1\\v_2\end{bmatrix}k \xrightarrow{\text{MPI\_Scatter}} \begin{bmatrix}vl(P_0)=v_0\\vl(P_1)=v_1\\vl(P_2)=v_2\end{bmatrix}k \xrightarrow{\text{suma local}} \begin{bmatrix}sl(P_0)\\sl(P_1)\\sl(P_2)\end{bmatrix}1 \xrightarrow{\text{MPI\_Reduce}} s(P_0)=sl(P_0)+sl(P_1)+sl(P_2)$$

$$k=\frac{N}{p}\,(p=\text{número de procesos})$$

$$v(P_0) = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \xrightarrow{\text{MPI\_Scatter}} \begin{bmatrix} vl(P_0) = v_0 \\ vl(P_1) = v_1 \\ vl(P_2) = v_2 \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \xrightarrow{\text{Suma local}} \begin{bmatrix} sl(P_0) \\ sl(P_1) \\ sl(P_2) \end{bmatrix} \begin{matrix} 1 \\ 1 \\ 1 \end{matrix} s(P_0) = sl(P_0) + sl(P_1) + sl(P_2)$$

$k = N/p$(p=número de procesos)

```
double suma_vecp(double v[N]){
    int p;
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    int i, k;  /*k=nº de filas de los v_i*/
    /*En algunas ocasiones por simplicidad declaramos: double vl[N];*/
    double *vl = (double*) malloc(sizeof(double)*k);
    double sl=0, s; /* sl=suma local, s= suma total*/
    k=N/p;
    MPI_Scatter(v, k, MPI_DOUBLE, vl, k, MPI_ DOUBLE, 0, MPI_COMM_WORLD);
    for (i=0; i<k; i++)  /*calcular suma local*/
        sl+=vl[i];
    MPI_Reduce(&sl, &s, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    return s;
}
```

```
int MPI_Scatter(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
```

```
int MPI_Reduce(void *sendbuf, void
    *recvbuf, int count, MPI_Datatype
    datatype, MPI_Op op, int root, MPI_Comm
    comm)
```

$$t_{\text{Scatter}} = (p-1)\left(t_s + \frac{N}{p}t_w\right)$$

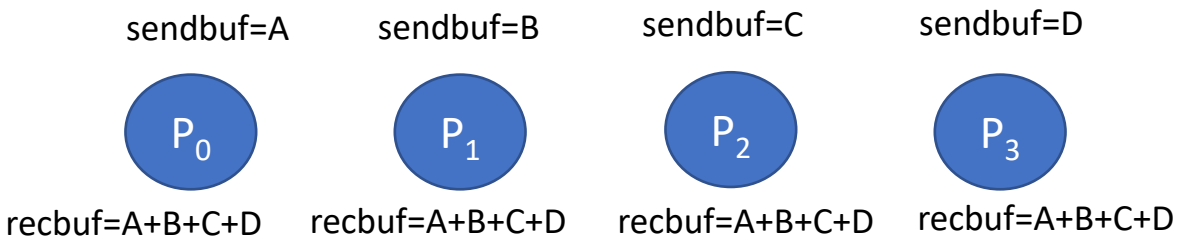
$$t_{\text{Reduce}} = (p-1)(t_s + t_w)$$

$$t_c(N,p) = (p-1)\left(2t_s + \left(1 + \frac{N}{p}\right)t_w\right)$$

Multi-Reducción

MPI_Allreduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

Ejemplo: Suma



count=|A|=|B|=|C|=|D|
op= MPI_SUM

	Antes
sendbuf(P_0)	A
sendbuf(P_1)	B
sendbuf(P_2)	C
sendbuf(P_3)	D

	Después
recbuf(P_0)	A+B+C+D
recbuf(P_1)	A+B+C+D
recbuf(P_2)	A+B+C+D
recbuf(P_3)	A+B+C+D

Ejercicio 6 (Suma elementos de un vector)

¿Qué modificarías en la implementación del ejercicio 9 para que todos recibiesen la suma?

$$v(P_0) = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} k \xrightarrow{\text{MPI\_Scatter}} \begin{bmatrix} vl(P_0) \\ vl(P_1) \\ vl(P_2) \end{bmatrix} k \xrightarrow{\text{Suma local}} \begin{bmatrix} sl(P_0) \\ sl(P_1) \\ sl(P_2) \end{bmatrix} k \xrightarrow{\text{MPI\_Allreduce}} s(P_0, P_1, P_2) = sl(P_0) + sl(P_1) + sl(P_2)$$

$k = N/p$(p=número de procesos)

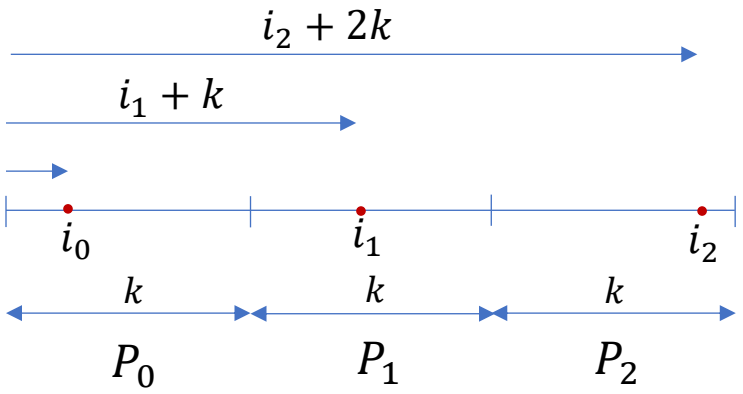
```
MPI_Reduce(&sl, &s, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

```
MPI_Allreduce(&sl, &s, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
```

Ejercicio 7 (Búsqueda lineal)

La siguiente función devuelve la última posición del vector v coincidente con **a** o **-1** si no la hubiese. Realiza una implementación MPI, suponiendo que inicialmente **v** y **a** están inicialmente almacenados en la memoria local de P0. Calcular el coste de las comunicaciones.

```
int pos(int v[N], int a){
    int pos=−1, i;
    for(i=0; i<N; i++)
        if (v[i]==a){
            pos=i;
        }
    return pos;
}
```



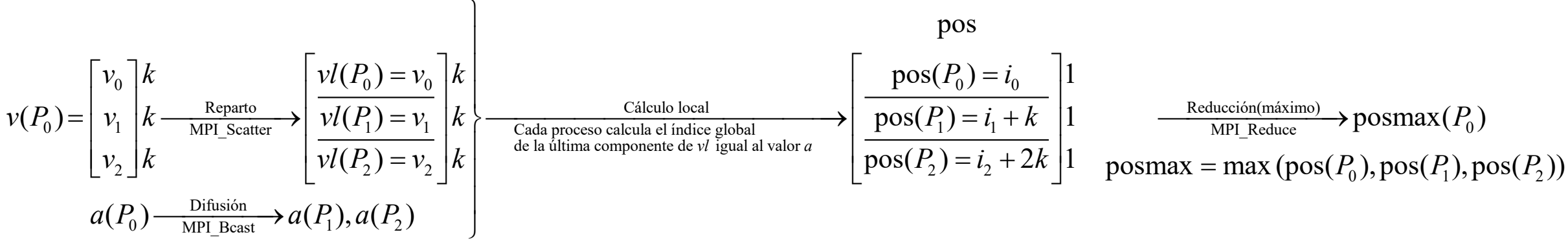
$$(P_0)=\begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix}k \xrightarrow[\text{MPI\_Scatter}]{\text{Reparto}} \begin{bmatrix} vl(P_0)=v_0 \\ vl(P_1)=v_1 \\ vl(P_2)=v_2 \end{bmatrix}k \xrightarrow[\text{Cada proceso calcula el índice global de la 1ª componente de } vl \text{ igual al valor } a]{\text{Cálculo local}} \begin{bmatrix} i_0 \\ i_1+k \\ i_2+2k \end{bmatrix}1 \quad \text{pos}$$

$$a(P_0) \xrightarrow[\text{MPI\_Bcast}]{\text{Difusión}} a(P_1), a(P_2)$$

$$\xrightarrow[\text{MPI\_Reduce}]{\text{Reducción(máximo)}} \text{posmax}(P_0)$$

$$\text{posmax}(P_0)=\max(\text{pos}(P_0),\text{pos}(P_1),\text{pos}(P_2))$$

$= N / p$ ($p =$ número de procesos)



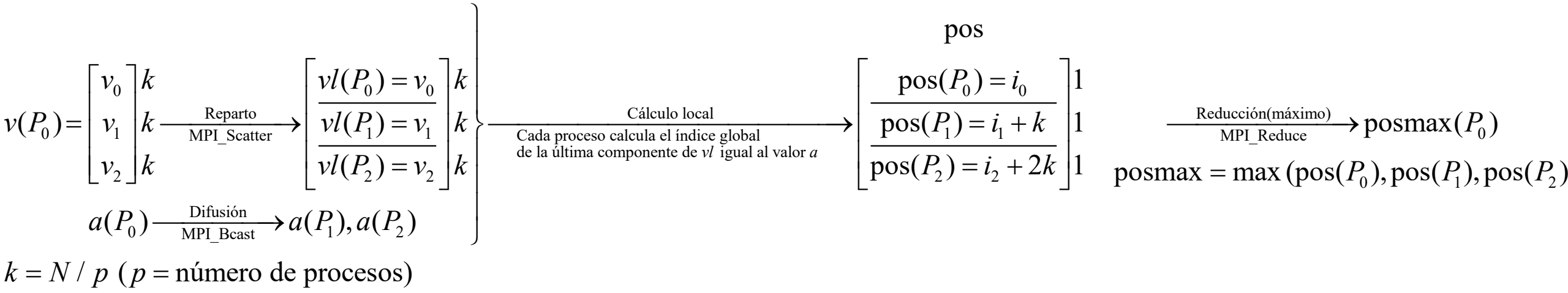
$k = N / p$ (p = número de procesos)

```
int posp(int v[N], int a){
   int p;
   MPI_Comm_size(MPI_COMM_WORLD,&p);
   MPI_Comm_rank(MPI_COMM_WORLD,&id);
   int k=N/p; /* Número de componentes de vl que le corresponden a cada proceso*/
   int pos,  posmax, i;
   int *vl = (int*) malloc(sizeof(int)*k);
   MPI_Scatter(v, k, MPI_INT, vl, k, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&a, 1,  MPI_INT, 0, MPI_COMM_WORLD);
   int pos=-1;
   for(i=0; i<k; i++)
      if (vl[i]==a){
         pos=i+id*k; /* posición global=posición local+k*id  */
      }
   MPI_Reduce(&pos, &posmax, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);
   return posmax;
}
```

```
int MPI_Scatter(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
```

```
int MPI_Bcast(void *buf, int count,
    MPI_Datatype datatype, int root,
    MPI_Comm comm)
```

```
int MPI_Reduce(void *sendbuf, void
    *recvbuf, int count, MPI_Datatype
    datatype, MPI_Op op, int root, MPI_Comm
    comm)
```



Costes de comunicaciones:

$$\left.\begin{aligned} &\text{MPI\_Scatter(vector de dimensión } N): t_{c_s} = (p-1)\left(t_s + \frac{N}{p} t_w\right) \\ &\text{MPI\_Bcast(dato simple)}: t_{c_g} = (p-1)\left(t_s + t_w\right) \\ &\text{MPI\_Reduce(dato simple)}: t_{c_r} = (p-1)\left(t_s + t_w\right) \end{aligned}\right\} t_c = (p-1)\left(3t_s + \left(\frac{N}{p}+2\right) t_w\right)$$

Ejercicio 8

La siguiente función calcula el producto de una matriz cuadrada por un vector, ambos de dimensión N:

```
void prod_mat_vec(double A[N][N], double v[N], double x[N]){
    int i, j;
    for (i=0;i<N;i++) {
        x[i]=0;
        for (j=0; j<N; j++)
          x[i] += A[i][j]*v[j];
    }
}
```

Paraleliza la función anterior mediante MPI, teniendo en cuenta que el proceso P0 contiene a la matriz **A** y al vector **v**, y que el vector **x** debe quedar almacenado en P0. Calcula el coste paralelo.

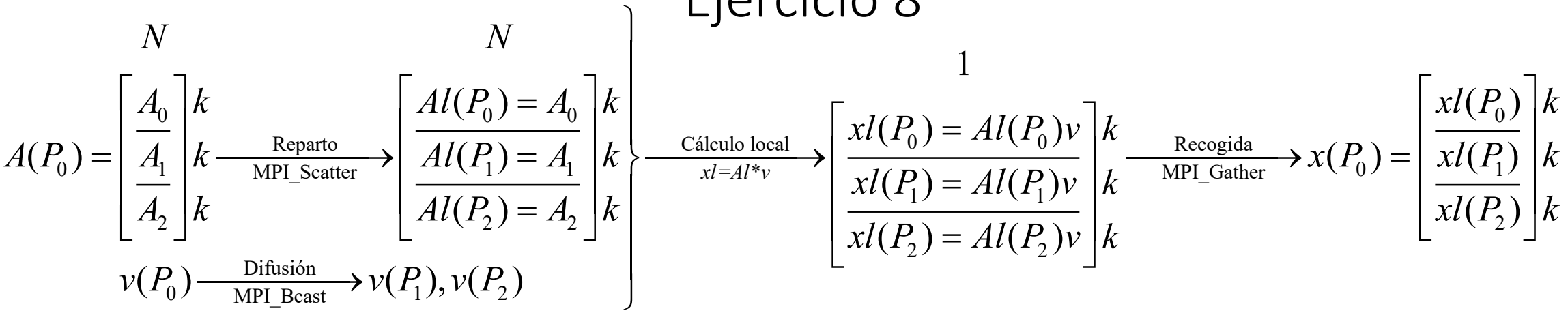
$$A(P_0) = \begin{bmatrix} A_0 \\ A_1 \\ A_2 \end{bmatrix}_k \xrightarrow[\text{MPI\_Scatter}]{\text{Reparto}} \begin{bmatrix} Al(P_0) = A_0 \\ Al(P_1) = A_1 \\ Al(P_2) = A_2 \end{bmatrix}_k$$

$$v(P_0) \xrightarrow[\text{MPI\_Bcast}]{\text{Difusión}} v(P_1), v(P_2)$$

$$k = N / p$$

$$\xrightarrow[xl=Al*v]{\text{Cálculo local}} \begin{bmatrix} xl(P_0) = Al(P_0)v \\ xl(P_1) = Al(P_1)v \\ xl(P_2) = Al(P_2)v \end{bmatrix}_k \xrightarrow[\text{MPI\_Gather}]{\text{Recogida}} x(P_0) = \begin{bmatrix} xl(P_0) \\ xl(P_1) \\ xl(P_2) \end{bmatrix}_k$$

Ejercicio 8




```
void prod_mat_vec(double A[N][N], double v[N], double x[N]){
int i, j, k, p;
double Al[N][N], xl[N];
MPI_Comm_size(MPI_COMM_WORLD, &p);
k=N/p;
MPI_Scatter(A, k*N, MPI_DOUBLE, Al, k*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(v, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
for (i=0; i<k; i++) {
      xl[i]=0;
      for (j=0;j<N;j++)
            xl[i] += Al[i][j]*v[j];
}
MPI_Gather(xl, k, MPI_DOUBLE, x, k, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

```
int MPI_Scatter(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
```

```
int MPI_Bcast(void *buf, int count,
    MPI_Datatype datatype, int root,
    MPI_Comm comm)
```

```
int MPI_Gather(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
```

Ejercicio 8

$$A(P_0)=\begin{bmatrix}\frac{A_0}{A_1}\\ \frac{}{A_2}\end{bmatrix}\begin{matrix}k\\k\\k\end{matrix}\xrightarrow[\text{MPI\_Scatter}]{\text{Reparto}}\begin{bmatrix}Al(P_0)=A_0\\ \hline Al(P_1)=A_1\\ \hline Al(P_2)=A_2\end{bmatrix}\begin{matrix}k\\k\\k\end{matrix}\qquad v(P_0)\xrightarrow[\text{MPI\_Bcast}]{\text{Difusión}}v(P_1),v(P_2)$$

$$\xrightarrow[\text{matriz por vector}]{\text{Cálculo local}}\text{Productos locales}\begin{bmatrix}xl(P_0)=Al(P_0)v\\ \hline xl(P_1)=Al(P_1)v\\ \hline xl(P_2)=Al(P_2)v\end{bmatrix}\begin{matrix}k\\k\\k\end{matrix}\xrightarrow[\text{MPI\_Gather}]{\text{Recogida}}x(P_0)=\begin{bmatrix}xl(P_0)\\ \hline xl(P_1)\\ \hline xl(P_2)\end{bmatrix}\begin{matrix}k\\k\\k\end{matrix}$$

```
for (i=0;i<k;i++) {
      xl[i]=0;
      for (j=0;j<N;j++)
            xl[i] += Al[i][j]*v[j];
}
```

Tiempo aritmético:

$$t_a=\sum_{i=0}^{k-1}\sum_{j=0}^{N}2=2kN=\frac{2N^2}{p}\ \text{flops}$$

Tiempo de comunicaciones:

$$\text{MPI\_Scatter}: t_{c_s}=(p-1)\left(t_s+\frac{N^2}{p}t_w\right)$$

$$\text{MPI\_Bcast}: t_{c_b}=(p-1)\left(t_s+Nt_w\right)$$

$$\text{MPI\_Gatter}: t_{c_g}=(p-1)\left(t_s+\frac{N}{p}t_w\right)$$

$$t_c=(p-1)\left(3t_s+\left(\frac{N^2}{p}+N+\frac{N}{p}\right)t_w\right)$$

Tiempo paralelo:

$$t_p=t_a+t_c$$

Ejercicio 9

Realiza una implementación paralela mediante MPI de la función **suma_mat**, la cual tiene como argumento de entrada una matriz A de dimensión MxN y devuelve la suma de los elementos de la matriz A. Supondremos que inicialmente A está almacenada en P0 y que **M** es divisible entre el número de procesos **p**. Calcula el tiempo paralelo.

Solución:

```
double suma_mat (double A[M][N]){
    ……..
    return 0;
}
```

$$A(P_0)=\begin{matrix}N\\ \left[\begin{matrix}A_0\\ A_1\\ A_2\end{matrix}\right]\begin{matrix}k\\k\\k\end{matrix}\end{matrix} \xrightarrow[\text{MPI\_Scatter}]{\text{Reparto}} \begin{matrix}N\\ \left[\begin{matrix}Al(P_0)\\ \hline Al(P_1)\\ \hline Al(P_2)\end{matrix}\right]\begin{matrix}k\\k\\k\end{matrix}\end{matrix} \xrightarrow[\text{suma local}]{\text{Calcular}} \begin{matrix}1\\ \left[\begin{matrix}s(P_0)\\ \hline s(P_1)\\ \hline s(P_2)\end{matrix}\right]\begin{matrix}1\\1\\1\end{matrix}\end{matrix} \xrightarrow[\text{MPI\_Reduce}]{\text{Reducción}} st(P_0)=s(P_0)+s(P_1)+s(P_2)$$

$$k = M / p \quad (p\text{=número de procesos})$$

Ejercicio 9

$$A(P_0)=\begin{matrix}N\\ \begin{bmatrix}A_0\\A_1\\A_2\end{bmatrix}\\ \end{matrix}\begin{matrix}k\\k\\k\end{matrix} \xrightarrow[\text{MPI\_Scatter}]{\text{Reparto}} \begin{matrix}N\\ \begin{bmatrix}Al(P_0)\\Al(P_1)\\Al(P_2)\end{bmatrix}\\ \end{matrix}\begin{matrix}k\\k\\k\end{matrix} \xrightarrow[\text{suma local}]{\text{Calcular}} \begin{matrix}1\\ \begin{bmatrix}s(P_0)\\s(P_1)\\s(P_2)\end{bmatrix}\\ \end{matrix}\begin{matrix}1\\1\\1\end{matrix} \xrightarrow[\text{MPI\_Reduce}]{\text{Reducción}} st(P_0)=s(P_0)+s(P_1)+s(P_2)$$

$$k = M / p \quad (p\text{=número de procesos})$$

```
int MPI_Scatter(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)

int MPI_Reduce(void *sendbuf, void
    *recvbuf, int count, MPI_Datatype
    datatype, MPI_Op op, int root, MPI_Comm
    comm)
```

```
double suma_mat p(double A[M][N]){
    int p;
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    int i, j, k= M/p; /*k=nº de filas de Ai = nº de filas de Al(Pi)*/
    double Al[M][N], s=0, st;/* s=suma local, st= suma total*/
    MPI_Scatter(A, k*N, MPI_DOUBLE, Al, k*N, MPI_ DOUBLE, 0, MPI_COMM_WORLD);
    for (i=0; i<k; i++) /*calcular suma local*/
        for (j=0; j<N; j++)
            s+=Al[i][j];
    MPI_Reduce(&s, &st, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    return st;
}
```

Ejercicio 9

$$A(P_0)=\begin{matrix}N\\ \begin{bmatrix}A_0\\A_1\\A_2\end{bmatrix}\\ k\\k\\k\end{matrix} \xrightarrow[\text{MPI\_Scatter}]{\text{Reparto}} \begin{matrix}N\\ \begin{bmatrix}\overline{Al(P_0)}\\\overline{Al(P_1)}\\\overline{Al(P_2)}\end{bmatrix}\\ k\\k\\k\end{matrix} \xrightarrow[\text{suma local}]{\text{Calcular}} \begin{matrix}1\\ \begin{bmatrix}\overline{s(P_0)}\\\overline{s(P_1)}\\\overline{s(P_2)}\end{bmatrix}\\ 1\\1\\1\end{matrix} \xrightarrow[\text{MPI\_Reduce}]{\text{Reducción}} st(P_0)=s(P_0)+s(P_1)+s(P_2)$$

$k = M / p$ (p=número de procesos)

Costes de comunicaciones:

$$t_{\text{Scatter}} = (p-1)\left(t_s + \frac{MN}{p}t_w\right)$$

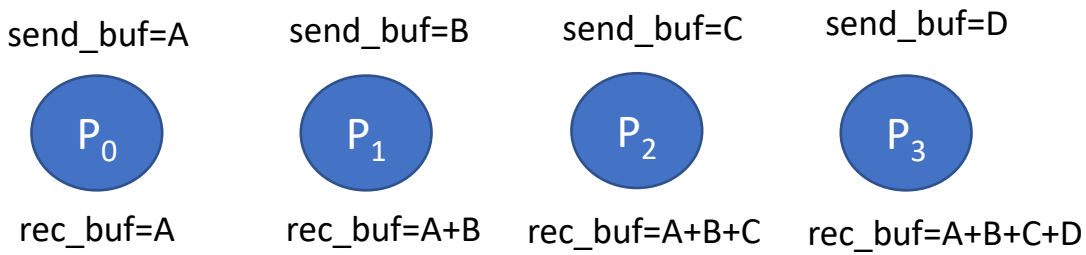
$$t_{\text{Reduce}} = (p-1)(t_s + t_w)$$

$$t_{\text{c}}(N,p) = (p-1)\left(2t_s + \left(1+\frac{MN}{p}\right)t_w\right)$$

Prefijación

MPI_Scan(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

Ejemplo: Suma



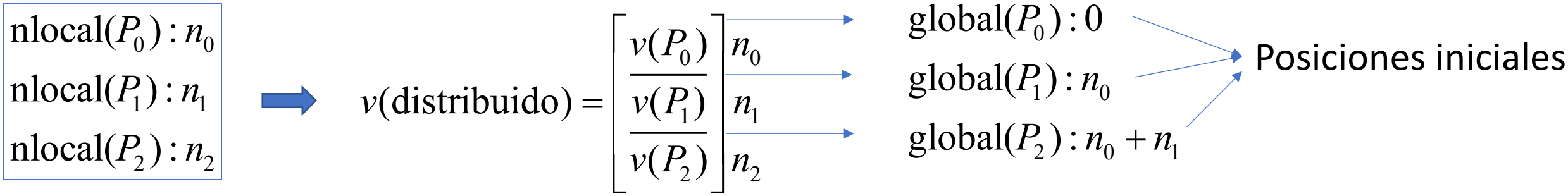
count=|A|=|B|=|C|=|D|
op= MPI_SUM

	Antes
send_buf(P_0)	A
send_buf(P_1)	B
send_buf(P_2)	C
send_buf(P_3)	D

	Después
rec_buf(P_0)	A
rec_buf(P_1)	A+B
rec_buf(P_2)	A+B+C
rec_buf(P_3)	A+B+C+D

Ejercicio 10

Dado un vector v de longitud *N*, distribuido entre los procesos, donde cada proceso tiene *n*local elementos consecutivos del vector, se quiere obtener la posición inicial del subvector local. Suponed que el número de elementos nlocal se puede conocer al realizar la siguiente llamada: calcula_nlocal(N,&nlocal).



Ejercicio 10

```
int global, nlocal, N;
calcula_nlocal(N,&nlocal);
MPI_Scan(&nlocal,&global,1,MPI_INT,MPI_SUM,comm);
global -= nlocal;
```

$$nlocal(P_0) = n_0$$
$$nlocal(P_1) = n_1$$
$$nlocal(P_2) = n_2$$

$$global(P_0) = n_0$$
$$global(P_1) : n_0 + n_1$$
$$global(P_2) : n_0 + n_1 + n_2$$

$$global(P_0) = n_0 - n_0 = 0$$
$$global(P_1) = n_0 + n_1 - n_1 = n_0$$
$$global(P_2) : n_0 + n_1 + n_2 - n_2 = n_0 + n_1$$

MPI_Scan

	Estado Inicial		Estado Final
P_0	A		A
P_1	B		A+B
P_2	C	⟶	A+B+C
P_3	D		A+B+C+D
P_4	E		A+B+C+D+E
P_5	F		A+B+C+D+E+F

Posiciones globales: