Transparencias MPI

Ejercicios Comunicaciones Colectivas

Reparto: **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

$t_c = (p-1)(t_s + \text{sendcount} \cdot t_w)$

	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

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

$t_c = (p-1)(t_s + \text{sendcount} \cdot t_w)$

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

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

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

Ejemplo:
count=|A|
root=0

Tiempo de comunicaciones:
$t_c = (p-1)(t_s + \text{count} \cdot t_w)$

	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

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

Ejemplo:
count=|A|=|B|=|C|=|D|
op= MPI_SUM
root=0

Tiempo de comunicaciones:
$t_c = (p-1)(t_s + \text{count} \cdot t_w)$

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

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

Cuestiones relativas a las comunicaciones colectivas

- Todos los procesos deben ejecutar la comunicación colectiva: no se puede encontrar una comunicación colectiva dentro la condición de una instrucción if que dependa de algún índice de proceso.
- Los pasos que se siguen al implementar código de comunicaciones colectivas son:
  1. Reparto y/o difusión de datos por parte de un proceso
  2. Cada proceso realiza operaciones locales con sus datos
  3. Los resultados se obtienen o bien recogiendo los datos locales o realizando una operación sobre los datos locales; de manera que los resultados se almacenan sobre un solo proceso o bien en todos (operaciones del tipo MPI_Allgather o MPI_Allreduce)

Ejercicio 1

La siguiente función devuelve el número de componentes de un vector **v** coincidentes con un entero **a**. Realiza una implementación MPI, suponiendo que inicialmente **v** y **a** están inicialmente almacenados en la memoria local de P_0. Calcular el coste de las comunicaciones.

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

$$v(P_0) = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \xrightarrow{\text{MPI\_Scatter}} \begin{bmatrix} vl(P_0) = v_0 \\ vl(P_1) = v_1 \\ vl(P_2) = v_2 \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \Bigg\} \xrightarrow{\begin{matrix} \text{Número de componentes} \\ \text{de } vl \text{ igual al valor } a \end{matrix}} \begin{matrix} 1 \\ \begin{bmatrix} \mathrm{c}(P_0) \\ \mathrm{c}(P_1) \\ \mathrm{c}(P_2) \end{bmatrix} \end{matrix} \begin{matrix} 1 \\ 1 \\ 1 \end{matrix} \xrightarrow{\text{MPI\_Reduce}} \mathrm{ct}(P_0) = \mathrm{c}(P_0) + \mathrm{c}(P_1) + \mathrm{c}(P_2)$$

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

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

Ejercicio 1

$$v(P_0) = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} k \xrightarrow{\text{MPI\_Scatter}} \begin{bmatrix} vl(P_0) = v_0 \\ vl(P_1) = v_1 \\ vl(P_2) = v_2 \end{bmatrix} k \xrightarrow{\text{Número de componentes de } vl \text{ igual al valor } a} \begin{matrix} 1 \\ \begin{bmatrix} c(P_0) \\ c(P_1) \\ c(P_2) \end{bmatrix} \end{matrix} 1 \xrightarrow{\text{MPI\_Reduce}} ct(P_0) = c(P_0) + c(P_1) + c(P_2)$$

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

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

```
int contp(int v[N], int a){
   int p, id;
   MPI_Comm_size(MPI_COMM_WORLD, &p);
   MPI_Comm_rank(MPI_COMM_WORLD, &id);
   int k=N/p;
   int c=0, ct, 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);
   for(i=0; i<k; i++)
      if (vl[i]==a)
         c++;
   MPI_Reduce(&c, &ct, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
   return total_cont;
}
```

```
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)
```

Ejercicio 1

$$v(P_0) = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} k \xrightarrow{\text{MPI\_Scatter}} \begin{bmatrix} vl(P_0) = v_0 \\ vl(P_1) = v_1 \\ vl(P_2) = v_2 \end{bmatrix} k \xrightarrow{\text{Número de componentes de } vl \text{ igual al valor } a} \begin{bmatrix} \mathrm{c}(P_0) \\ \mathrm{c}(P_1) \\ \mathrm{c}(P_2) \end{bmatrix} \begin{matrix} 1 \\ 1 \\ 1 \end{matrix} \xrightarrow{\text{MPI\_Reduce}} \mathrm{ct}(P_0) = \mathrm{c}(P_0) + \mathrm{c}(P_1) + \mathrm{c}(P_2)$$

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

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

MPI_Scatter(vector de dimensión N): $t_{c_s} = (p-1)\left(t_s + \frac{N}{p} t_w\right)$

MPI_Bcast(dato simple): $t_{c_g} = (p-1)(t_s + t_w)$

MPI_Reduce(dato simple): $t_{c_r} = (p-1)(t_s + t_w)$

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

Ejercicio 2

La siguiente función devuelve el valor máximo de una matriz de cuadrada de orden N.

```
int max_mat(int A[N][N]){
    int i, j;
    double maxi=A[0][0];
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
        if (A[i][j]>maxi)
            maxi=A[i][j];
    return maxi;
}
```

Realiza una implementación MPI sabiendo que inicialmente el único proceso que tiene almacenada la matriz A es P0, en los siguientes casos:

a) El valor máximo quede almacenado en P0.

b) El valor máximo quede almacenado en todos los procesos.

c) Calcula el tiempo de comunicaciones en los dos casos anteriores.

$$A(P_0) = \begin{matrix} N \\ \begin{bmatrix} A_0 \\ A_1 \\ A_2 \end{bmatrix} \end{matrix} \begin{matrix} k \\ k \\ k \end{matrix} \xrightarrow{\text{MPI\_Scatter}} \begin{matrix} N \\ \begin{bmatrix} Al(P_0) = A_0 \\ Al(P_1) = A_1 \\ Al(P_2) = A_2 \end{bmatrix} \end{matrix} \begin{matrix} k \\ k \\ k \end{matrix} \xrightarrow{\text{Máximo local}} \begin{matrix} 1 \\ \begin{bmatrix} \text{maxl}(P_0) \\ \text{maxl}(P_1) \\ \text{maxl}(P_2) \end{bmatrix} \end{matrix} \begin{matrix} 1 \\ 1 \\ 1 \end{matrix} \xrightarrow{\text{MPI\_Reduce}} \text{maxg}(P_0) = \max\{\text{maxl}(P_0), \text{maxl}(P_1), \text{maxl}(P_2)\}$$

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

Ejercicio 2

$$A(P_0) = \overset{N}{\begin{bmatrix} A_0 \\ A_1 \\ A_2 \end{bmatrix}}\begin{matrix} k \\ k \\ k \end{matrix} \xrightarrow{\text{MPI\_Scatter}} \overset{N}{\begin{bmatrix} Al(P_0) = A_0 \\ Al(P_1) = A_1 \\ Al(P_2) = A_2 \end{bmatrix}}\begin{matrix} k \\ k \\ k \end{matrix} \xrightarrow{\text{Máximo local}} \overset{1}{\begin{bmatrix} \text{maxl}(P_0) \\ \text{maxl}(P_1) \\ \text{maxl}(P_2) \end{bmatrix}}\begin{matrix} 1 \\ 1 \\ 1 \end{matrix} \xrightarrow{\text{MPI\_Reduce}} \text{maxg}(P_0 = \max\{\text{maxl}(P_0), \text{maxl}(P_1), \text{maxl}(P_2)$$

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

a)

```
double contp(double A[N][N]){
    int p, id, i, j, k;
    double Al[N][N];
    MPI_Comm_size(MPI_COMM_WORLD,&p);
    MPI_Comm_rank(MPI_COMM_WORLD,&id);
    k=N/p;
    MPI_Scatter(A, k*N, MPI_DOUBLE, Al, k*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    double maxl=Al[0][0], maxg;
    for(i=0; i<k; i++)
        for(j=0; j<N; j++)
            if (Al[i][j]>maxl)
                maxl=Al[i][j];
    MPI_Reduce(&maxl, &maxg, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
    return maxg;
}
```

```
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)
```

Ejercicio 2

$$A(P_0)=\begin{matrix}N\\ \left[\begin{matrix}A_0\\A_1\\A_2\end{matrix}\right]\end{matrix}k \xrightarrow[\text{MPI\_Scatter}]{\text{Reparto}} \begin{matrix}N\\ \left[\begin{matrix}Al(P_0)=A_0\\Al(P_1)=A_1\\Al(P_2)=A_2\end{matrix}\right]\end{matrix}k \xrightarrow[\text{máximo local}]{\text{Cálculo del}} \begin{matrix}1\\ \left[\begin{matrix}\text{maxl}(P_0)\\\text{maxl}(P_1)\\\text{maxl}(P_2)\end{matrix}\right]\end{matrix}1 \xrightarrow[\text{MPI\_Reduce}]{\text{Cálculo del máximo global}} \text{maxg}(P_0)=\max\{\text{maxl}(P_0),\text{maxl}(P_1),\text{maxl}(P_2)\}$$

$k = N / p$ (p = número de procesos)

b)

```
double contp(double A[N][N]){
    int p, id, i, j, k;
    double Al[N][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);
    double maxl=A[0][0], maxg;
    for(i=0; i<k; i++)
        for(j=0; j<N; j++)
        if (Al[i][j]>maxl)
            maxl=Al[i][j];
    MPI_Allreduce(&maxi, &maxg, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
    return maxg;
}
```

```
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)
```

MPI_Allreduce: mismos componentes que MPI_Reduce, salvo que no tiene el argumento int root

Ejercicio 2

$$A(P_0)=\begin{matrix}N\\ \left[\begin{matrix}A_0\\ A_1\\ \hline A_2\end{matrix}\right]\end{matrix}\begin{matrix}k\\k\\k\end{matrix}\xrightarrow[\text{MPI\_Scatter}]{\text{Reparto}}\begin{matrix}N\\ \left[\begin{matrix}Al(P_0)=A_0\\ Al(P_1)=A_1\\ \hline Al(P_2)=A_2\end{matrix}\right]\end{matrix}\begin{matrix}k\\k\\k\end{matrix}\xrightarrow[\text{máximo local}]{\text{Cálculo del}}\begin{matrix}1\\ \left[\begin{matrix}\text{maxl}(P_0)\\ \text{maxl}(P_1)\\ \hline \text{maxl}(P_2)\end{matrix}\right]\end{matrix}\begin{matrix}1\\1\\1\end{matrix}\xrightarrow[\text{MPI\_Reduce}]{\text{Cálculo del máximo global}}\text{maxg}(P_0)=\max\left\{\text{maxl}(P_0),\text{maxl}(P_1),\text{maxl}(P_2)\right\}$$

$k = N / p$ (p = número de procesos)

c) Costes de comunicaciones

$$\left.\begin{array}{l}\text{MPI\_Scatter(matriz de dimensión } kN): t_{c_s}=(p-1)\left(t_s+\dfrac{N^2}{p}t_w\right)\\ \text{MPI\_Reduce(dato simple): } t_{c_r}=(p-1)\left(t_s+t_w\right)\end{array}\right\} t_c=(p-1)\left(2t_s+\left(\frac{N^2}{p}+1\right)t_w\right)$$

Allreduce se puede realizar mediante una operación **reduce** sobre el proceso 0, seguida de un **broadcast** del resultado. Para el **broadcast**, suponemos que el proceso 0 envía un mensaje de un elemento a cada uno de los demás procesos:

$$\left.\begin{array}{l}\text{MPI\_Scatter(vector de dimensión } N): t_{c_s}=(p-1)\left(t_s+\dfrac{N^2}{p}t_w\right)\\ \text{MPI\_Allreduce(dato simple): } t_{c_r}=2(p-1)\left(t_s+t_w\right)\end{array}\right\} t_c=\left(p-1\right)\left(3t_s+\left(\frac{N^2}{p}+2\right)t_w\right)$$

Ejercicio 3

Dada la siguiente función

```
void func(double A[M][N], int sup[M]) {
  int i, j;
  double s = 0, m;
  for (i=0; i<M; i++)
    for (j=0; j<N; j++)
      s += A[i][j];
  m = s/(M*N);
  for (i=0; i<M; i++) {
    sup [i] = 0;
    for (j=0; j<N; j++)
      if (A[i][j]>m) sup[i]++;
  }
}
```

a) ¿Qué hace la función?
b) Escribe un programa MPI suponiendo que **A** se encuentra inicialmente en el proceso 0, y que al finalizar la función el vector **sup** debe estar también en el proceso 0. Se puede suponer que el número de filas de la matriz es divisible entre el número de procesos.
c) Calcula el tiempo paralelo

Solución:

a) Dada la matriz A, en primer lugar, calcula y almacena en **m** el valor medio de los elementos de **A**; a continuación, genera un vector **sup** de manera que su componente i-ésima contiene el nº de elementos de la fila i-ésima de la matriz **A** que son mayores que la media de los elementos de **A**

```
void func(double A[M][N], int sup[M]) {
   int i, j;
   double s = 0, m;
   for (i=0; i<M; i++)
      for (j=0; j<N; j++)
         s += A[i][j];
   m = s/(M*N);
   for (i=0; i<M; i++) {
      sup [i] = 0;
      for (j=0; j<N; j++)
         if (A[i][j]>m) sup[i]++;
   }
}
```

$$A(P_0) = \begin{matrix} N \\ \left[\begin{matrix} A_0 \\ A_1 \\ A_2 \end{matrix}\right] \end{matrix} k \xrightarrow[\text{MPI\_Scatter}]{\text{Reparto de A}} \begin{matrix} N \\ \left[\begin{matrix} Al(P_0) = A_0 \\ Al(P_1) = A_1 \\ Al(P_2) = A_2 \end{matrix}\right] \end{matrix} k \xrightarrow[\text{la suma local}]{\text{Cálculo de}} \begin{matrix} 1 \\ \left[\begin{matrix} \mathrm{s}(P_0) \\ \mathrm{s}(P_1) \\ \mathrm{s}(P_2) \end{matrix}\right] \end{matrix} 1$$

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

$$\xrightarrow[\text{MPI\_Allreduce}]{\text{Suma global}} m(P_0, P_1, P_2) = \mathrm{s}(P_0) + \mathrm{s}(P_1) + \mathrm{s}(P_1) \longrightarrow m(P_0, P_1, P_2) = \frac{m(P_0, P_1, P_2)}{MN}$$

$$\xrightarrow[\text{de supl}]{\text{Cálculo local}} \begin{matrix} 1 \\ \left[\begin{matrix} \mathrm{supl}(P_0) \\ \mathrm{supl}(P_1) \\ \mathrm{supl}(P_2) \end{matrix}\right] \end{matrix} k \xrightarrow[\text{MPI\_Gather}]{\text{Recogida vectores locales supl}} \mathrm{sup}(P_0) = \begin{matrix} 1 \\ \left[\begin{matrix} \mathrm{supl}(P_0) \\ \mathrm{supl}(P_1) \\ \mathrm{supl}(P_2) \end{matrix}\right] \end{matrix} k$$

```
void funcp(double A[M][N], int sup[M]) {
  double Al[M][N];
  int supl[M];
  double s=0;
  int i, j, p, k;
  MPI_Comm_size(MPI_COMM_WORLD, &p); k=M/p;
```

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

$$A(P_0) = \begin{matrix} N \\ \left[\begin{matrix} A_0 \\ A_1 \\ A_2 \end{matrix}\right] \begin{matrix} k \\ k \\ k \end{matrix} \end{matrix} \xrightarrow[\text{MPI\_Scatter}]{\text{Reparto de A}} \begin{matrix} N \\ \left[\begin{matrix} Al(P_0)=A_0 \\ Al(P_1)=A_1 \\ Al(P_2)=A_2 \end{matrix}\right] \begin{matrix} k \\ k \\ k \end{matrix} \end{matrix}$$

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

```
  for (i=0; i<k; i++)
      for (j=0; j<N; j++)
        s += Al [i][j];
```

Almacenar en s la suma local de los elementos de Al

$$\left[\begin{matrix} s(P_0) \\ s(P_1) \\ s(P_2) \end{matrix}\right] 1$$

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

$$\left[\begin{matrix} s(P_0) \\ s(P_1) \\ s(P_2) \end{matrix}\right] 1$$

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

↓ Reducción (suma) para todos

$$m(P_0,P_1,P_2) = s(P_0) + s(P_1) + s(P_1)$$

```
  m = m/(M*N)
```

$$m(P_0,P_1,P_2) = \frac{m(P_0,P_1,P_2)}{MN}$$

```
  for (i=0; i<k; i++) {
      supl[i] = 0;
      for (j=0; j<N; j++)
        if (Al[i][j]>m) supl [i]++;
    }
```

Para cada componente, **supl** contiene el nº de elementos de la fila mayores que la media

$$\left[\begin{matrix} supl(P_0) \\ supl(P_1) \\ supl(P_2) \end{matrix}\right] k$$

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

```
  MPI_Gather(supl, k, MPI_INT, sup, k, MPI_INT, 0, MPI_COMM_WORLD);
```

$$\left[\begin{matrix} supl(P_0) \\ supl(P_1) \\ supl(P_2) \end{matrix}\right] k \xrightarrow[\text{MPI\_Gather}]{\text{Recogida de supl en } P_0} sup(P_0) = \left[\begin{matrix} supl(P_0) \\ supl(P_1) \\ supl(P_2) \end{matrix}\right] k$$

```
}
```

```
void funcp(double A[M][N], int sup[M]) {
   double Al[M][N];
   int supl[M];
   double s=0;
   int i, j, p;
   MPI_Comm_size(MPI_COMM_WORLD, &p);
   MPI_Scatter(A, k*N, MPI_DOUBLE, Al, k*N, MPI_DOUBLE, 0, MPI_COMM_WORLD)

   for (i=0; i<k; i++)
        for (j=0; j<N; j++)
           s += Al [i][j];

   MPI_Allreduce(&s, &m, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

    m = m/(M*N)

   for (i=0; i<k; i++) {
        supl[i] = 0;
        for (j=0; j<N; j++)
           if (Al[i][j]>m) supl [i]++;
     }
   MPI_Gather(supl, M/p, MPI_INT, sup, M/p, MPI_INT, 0, MPI_COMM_WORLD);
}
```

$$t_a(M,N,p) = \sum_{i=0}^{M/p}\sum_{j=0}^{N} 1 + p - 1 + 2 \approx \frac{MN}{p} \text{ flops}$$

MPI_Scatter(matriz de dimensión MN): $t_{c_s} = (p-1)\left(t_s + \frac{MN}{p}t_w\right)$

MPI_Allreduce(dato simple): $t_{c_r} = 2(p-1)(t_s + t_w)$

MPI_Gather: $t_{c_g} = (p-1)\left(t_s + \frac{M}{p}t_w\right)$

$$t(M,N,p) \approx \frac{MN}{p} \text{ flops} + t_{c_s} + t_{c_r} + t_{c_g}$$

Ejercicio 4 (Parcial 2020)

Observa la siguiente función, la cual cuenta el número de apariciones de un número en una matriz e indica también la primera fila en la que aparece:

```
void search(double A[M][N], double x) {
  int i, j, first, count;
  first = M ; count = 0;
  for (i=0; i<M; i++)
    for (j=0; j<N; j++)
      if (A[i][j] == x) {
          count++;
          if (i < first) first = i;
      }
  printf("%g está %d veces, la primera vez en la fila %d.\n", x, count, first);
}
```

Ejercicio 4 (Parcial 2020)

Paralelízala mediante MPI repartiendo la matriz **A** entre todos los procesos disponibles. Tanto la matriz como el valor a buscar están inicialmente disponibles únicamente en el proceso **owner**. Asumimos que el número de filas y columnas de la matriz es un múltiplo exacto del número de procesos. El **printf** que muestra el resultado por pantalla debe hacerlo únicamente un proceso. Utiliza operaciones de comunicación colectiva allí donde sea posible. Para ello, completa esta función:

```
void par_search(double A[M][N], double x, int owner) {
double Al[M][N];
```

Nota: por simplicidad, en el siguiente esquema supondremos que el propietario (**owner**) es el proceso 0.

Reparto y Difusión inicial

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

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

$k = M / p$ (p = número de procesos)

Cálculo local

```
firstl = M ; countl = 0;
  for (i=0; i<k; i++)
    for (j=0; j<N; j++)
      if (Al[i][j] == x) {
          countl++;
          if (i < firstl) firstl = i;
      }
firstl=k*id+firstl /*índice global*/
```

$$\begin{matrix} i_g & \lfloor k \\ i_l & i_p \end{matrix} \rightarrow i_g = k i_p + i_l$$

Obtención de resultados

$$\text{first} = \min\{\text{firstl}(P_0), \text{firstl}(P_1), \text{firstl}(P_2)\}$$
$$\text{count} = \text{countl}(P_0) + \text{countl}(P_1) + \text{countl}(P_2)$$

$$A(P_0) = \begin{bmatrix} \frac{A_0}{A_1} \\ \frac{A_1}{A_2} \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \xrightarrow[\text{MPI\_Scatter}]{\text{Reparto}} \begin{bmatrix} \frac{Al(P_0)=A_0}{Al(P_1)=A_1} \\ \frac{Al(P_1)=A_1}{Al(P_2)=A_2} \end{bmatrix} \begin{matrix} k \\ k \\ k \end{matrix} \xrightarrow{\text{Cálculo local}} \begin{Bmatrix} \text{firstl} \\ \text{contl} \end{Bmatrix} \rightarrow \begin{matrix} \text{firstl=k*id+firstl} \\ \text{(índice global)} \end{matrix} \rightarrow \begin{matrix} \text{first}(P_0) = \min\{\text{firstl}(P_0), \text{firstl}(P_1), \text{firstl}(P_2)\} \\ \text{cont}(P_0) = \text{contl}(P_0) + \text{contl}(P_1) + \text{contl}(P_2) \end{matrix}$$

$$x(P_0) \xrightarrow[\text{MPI\_Bcast}]{\text{Difusión}} x(P_1), x(P_2) \qquad (k = M/p,\ p = \text{número de procesos})$$

```
void par_search(double A[M][N], double x, int owner) {
   double Al[M][N];
   int i, j, first,  count, firstl, countl, id, p;
   MPI_Comm_rank(MPI_COMM_WORLD, &id);
   MPI_Comm_size(MPI_COMM_WORLD, &p);
   k=M/p;
   MPI_Scatter(A, k*N, MPI_DOUBLE, Al, k*N, MPI_DOUBLE, owner, MPI_COMM_WORLD);
   MPI_Bcast(&x, 1 ,MPI_DOUBLE, owner, MPI_COMM_WORLD);
   firstl = M ; countl = 0;
   for (i=0; i<k; i++)
      for (j=0; j<N; j++)
         if (Al[i][j] == x) {
            countl++;
            if (i < firstl) firstl = i;
         }
    firstl = k*id + firstl;
   MPI_Reduce(&firstl, &first, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);
   MPI_Reduce(&countl, &count, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
   if (id == 0) printf("%g está %d veces, la primera vez en la fila %d.\n", x, count, first);
}
```

```
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)
```

Ejercicio 5 (Parcial 2021)

La siguiente función desplaza un vector restándole a todos sus elementos el menor de sus valores:

```
void desplaza(int n, double v[MAXN]){
    int i;
    double m=v[0];
    for ( i = 1 ; i < n ; i++ )
        if ( v[i] < m ) m = v[i];
    for ( i = 0 ; i < N ; i++ )
        v[i] -= m;
}
```

Paraleliza la función utilizando para las comunicaciones únicamente operaciones de **comunicación colectiva de MPI**. Todos los procesos deben colaborar en la realización de los dos bucles. Los argumentos de la función (**ambos**) sólo son correctos en el proceso 0. El resultado (el vector desplazado) se desea únicamente en el proceso 0. Asume que el tamaño del vector (n) es un múltiplo exacto del número de procesos.

Ejercicio 5 (Parcial 2021)

```
void desplaza(int n, double v[MAXN]){
    int i;
    double m=v[0];
    for ( i = 1 ; i < n ; i++ )
        if ( v[i] < m ) m = v[i];
    for ( i = 0 ; i < N ; i++ )
        v[i] -= m;
}
```

Los argumentos de la función (**ambos**) sólo son correctos en el proceso 0.

$$n(P_0) \xrightarrow[\text{MPI\_Bcast}]{\text{Difusión}} n(P_0,P_1,P_2) \rightarrow k(P_0,P_1,P_2) = \frac{n(P_0,P_1,P_2)}{p}$$

p=número de procesos

$$\rightarrow v(P_0) = \begin{bmatrix} v_0 \\ \hline v_1 \\ \hline v_2 \end{bmatrix}^{k}_{k}_{k} \xrightarrow[\text{MPI\_Scatter}]{\text{Reparto}} \begin{bmatrix} vl(P_0)=v_0 \\ \hline vl(P_1)=v_1 \\ \hline vl(P_2)=v_2 \end{bmatrix}^{1}_{k}_{k}_{k} \rightarrow$$

```
ml = vl[0];
for ( i = 1 ; i < k ; i++ )
    if ( vl[i] < ml) ml = vl[i];
```

$$\rightarrow \begin{bmatrix} ml(P_0) \\ \hline ml(P_1) \\ \hline ml(P_2) \end{bmatrix}^{1}_{1}_{1} \xrightarrow[\text{MPI\_Allreduce}]{\text{Reducción total}} m(P_0,P_1,P_2) = \min\{ml(P_0), ml(P_1), ml(P_2)\}$$

$$\rightarrow$$

```
for ( i = 0 ; i < k ; i++ )
    vl[i] -= m;
```

$$\rightarrow \begin{bmatrix} vl(P_0) \\ \hline vl(P_1) \\ \hline vl(P_2) \end{bmatrix}^{1}_{k}_{k}_{k} \xrightarrow[\text{MPI\_Gather}]{\text{Recogida}} v(P_0) = \begin{bmatrix} vl(P_0) \\ vl(P_1) \\ vl(P_2) \end{bmatrix}^{1}_{k}_{k}_{k}$$

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

```
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)
```

$$n(P_0) \xrightarrow[\text{MPI\_Bcast}]{\text{Difusión}} n(P_0,P_1,P_2) \rightarrow k(P_0,P_1,P_2) = \frac{n(P_0,P_1,P_2)}{p}$$

$$1 \qquad\qquad 1$$

$$\rightarrow v(P_0) = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} k \xrightarrow[\text{MPI\_Scatter}]{\text{Reparto}} \begin{bmatrix} vl(P_0)=v_0 \\ vl(P_1)=v_1 \\ vl(P_2)=v_2 \end{bmatrix} k \rightarrow$$

```
ml = vl[0];
for ( i = 1 ; i < k ; i++ )
    if ( vl[i] < ml) ml = vl[i];
```

$$1$$

$$\rightarrow \begin{bmatrix} ml(P_0) \\ ml(P_1) \\ ml(P_2) \end{bmatrix} 1 \xrightarrow[\text{MPI\_Allreduce}]{\text{Reducción total}} m(P_0,P_1,P_2) = \min\{ml(P_0), ml(P_1), ml(P_2)\}$$

$$1 \qquad\qquad 1$$

$$\rightarrow$$

```
for ( i = 0 ; i < k ; i++ )
    vl[i] -= m;
```

$$\rightarrow \begin{bmatrix} vl(P_0) \\ vl(P_1) \\ vl(P_2) \end{bmatrix} k \xrightarrow[\text{MPI\_Gather}]{\text{Recogida}} v(P_0) = \begin{bmatrix} vl(P_0) \\ vl(P_1) \\ vl(P_2) \end{bmatrix} k$$

```
void desplaza(int n,double v[MAXN]){
  int i, p, k;
  double m, vl[MAXN], ml;
  MPI_Comm_size(MPI_COMM_WORLD, &p);
  MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
  k = n / p;
  MPI_Scatter(v, k, MPI_DOUBLE, vl ,k,
                  MPI_DOUBLE, 0, MPI_COMM_WORLD);
  ml = vl[0];
  for ( i = 1 ; i < k ; i++ )
    if ( vl[i] < ml) ml = vl[i];
  MPI_Allreduce(&ml, &m, 1, MPI_DOUBLE, MPI_MIN,
                  MPI_COMM_WORLD);
  for ( i = 0 ; i < k ; i++ )
    vl[i] -= m;
  MPI_Gather(vl,k, MPI_DOUBLE, v, k, MPI_DOUBLE, 0,
                  MPI_COMM_WORLD);}
```