Práctica 2

Sesión 3

Objetivos

- Resolver sistemas de ecuaciones lineales mediante métodos iterativos
- Transformar implementaciones paralelas implementadas mediante comunicaciones punto a punto en otras implementadas con comunicaciones colectivas

Descripción del problema

- Resolver sistemas de ecuaciones lineales

$$Ax = b;\ A \square\square^{n \square n}, b \square\square^{n}$$

mediante un método iterativo:

- Partir de una "solución inicial" x^0
- Generar una sucesión de soluciones, usando expresiones del tipo

$$x^{k+1} = Mx^k + v;\ M \in \Re^{n \times n}, v \in \Re^n, k = 1,2,3,\mathrm{L}$$

hasta alcanzar la convergencia a la solución del sistema (si la tuviera)

Algoritmo secuencial

```
Inicializar M,x,v
Para iter=1,2,…num_iter
     x=M*x+v
Fin_para
Norma=suma de los valores absolutos de x
Mostrar norma
```

Figura 6

- `num_iter:` número fijo de iteraciones a realizar
- La norma nos sirve como *comprobación* para saber si los resultados obtenidos son correctos
- Tenemos una implementación paralela por filas (`mxv1.c`) y otra por columnas (`mxv2.c`) del algoritmo de la figura 6

Ejercicios 1, 2 y 3

- En la primera implementación del código anterior (mxv1.c) la matriz M está almacenada por filas.
- La matriz M es repartida por bloques de filas y el vector v por bloques de componentes.
- En cada iteración se obtienen los vectores xloc que deben recogerse para todos los procesos en el vector x

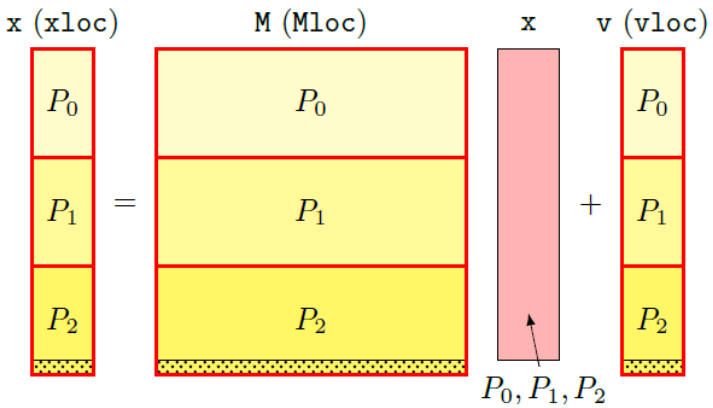


Figura 7: Distribución de datos en `mxv1.c`.

```
11  /* NOTE: We work with matrices stored by rows */
12  #define M(i,j) M[(i)*n + (j)]
13  #define Mloc(i,j) Mloc[(i)*n + (j)]
14
```

Ejercicios 1 y 2

- Compilar el programa mxv1.c:

```
$ mpicc mxv1.c –o mxv1
```

- Ejecución:

  mpiexec… mxv1 [–s semilla] [-n tamanyo] [-i iteraciones]

- Ejemplo (ejecución sobre 3 procesos):

```
$ mpiexec –n 3 mxv1 –n 5 –i 5
```

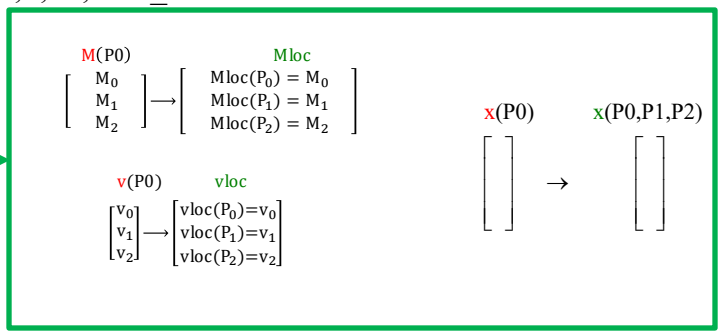
- Para ejecuciones más largas utiliza el sistema de colas

Ejercicio 3

- Sustituir en el código de **mxv1.c** las operaciones punto a punto que sean factibles de ser substituidas por operaciones colectivas de comunicación (pasos en rojo). En el código estas operaciones están marcadas con el comentario previo /* COMUNICACIONES */
- Código a implementar: $x^{k+1} = Mx^k + v;\ M \in \Re^{n\times n}, v \in \Re^n, k = 1,2,3,\text{L}\ ,\text{num\_iter}$
- Algoritmo:

P0 obtiene la matriz M, el vector v y el vector inicial x

(1) P0 reparte M y v (Mloc, vloc) y difunde el vector inicial x



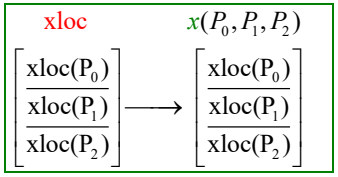
Para i=1,2,…, num_iter

Cada Pi **calcula** su parte local:

$$\begin{bmatrix} \text{xloc}(P_0) \\ \text{xloc}(P_1) \\ \text{xloc}(P_2) \end{bmatrix} = \begin{bmatrix} \text{Mloc}(P_0) \\ \text{Mloc}(P_1) \\ \text{Mloc}(P_2) \end{bmatrix} \begin{bmatrix} x \end{bmatrix} + \begin{bmatrix} \text{vloc}(P_0) \\ \text{vloc}(P_1) \\ \text{vloc}(P_2) \end{bmatrix}$$

$$\text{xloc}(P_i) = \text{Mloc}(P_i)x + \text{vloc}(P_i)$$

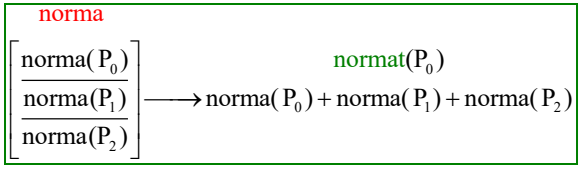
(2) Multirecogida de los vectores xloc, para que todos los procesos tengan el vector completo x:



Fin para

(3)Todos los procesos ayudan a **calcular la 1-norma de x**, quedando en P0 el resultado obtenido $\left(\|x\|_1 = |x_1| + |x_2| + \text{L}\ + |x_n|\right)$

P0 imprime el valor de la 1-norma del vector calculado



Ejercicio 3

- Al ejecutar el código original mxv1 sin ningún parámetro, se obtiene el siguiente resultado:

  1-norm (50 iterations): 42393.21405

- Hay que realizar los tres pasos señalados con la numeración (1), (2) y (3)
- En cada paso hay que eliminar/comentar las comunicaciones punto a punto, sustituyéndolas por comunicaciones colectivas.
- En cada paso hay que comprobar que se ha efectuado bien la sustitución. Para ello, ejecuta el código modificado, comprobando que se obtienen los mismos resultados que en el código original:

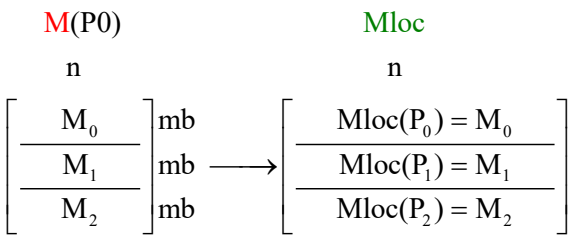
  1-norm (50 iterations): 42393.21405

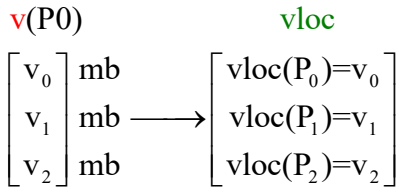
- Como la práctica es larga, realiza posteriormente las ejecuciones en Kahan

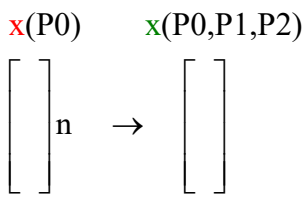
Ejercicio 3: primer paso

1. P0 reparte M y v (Mloc, vloc) y difunde el vector inicial x: sustituir el código por operaciones colectivas

```
135   /* COMMUNICATIONS */
136   /* Distribute M and v by blocks of rows (mb rows for each process)
137    * and send the initial x to all */
138   if (me == 0) {
139     /* For process 0, copy the data */
140     for (i = 0; i < mb; i++) {
141       for (j = 0; j < n; j++)
142         Mloc(i, j) = M(i, j);
143       vloc[i] = v[i];
144     }
145     /* For other processes, send the data */
146     for (proc = 1; proc < num_procs; proc++) {
147       MPI_Send(&M[proc*sz], sz, MPI_DOUBLE, proc, 13, MPI_COMM_WORLD);
148       MPI_Send(&v[proc*mb], mb, MPI_DOUBLE, proc, 89, MPI_COMM_WORLD);
149       MPI_Send(x, n, MPI_DOUBLE, proc, 25, MPI_COMM_WORLD);
150     }
151   } else {
152     MPI_Recv(Mloc, sz, MPI_DOUBLE, 0, 13, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
153     MPI_Recv(vloc, mb, MPI_DOUBLE, 0, 89, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
154     MPI_Recv(x, n, MPI_DOUBLE, 0, 25, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
155   }
```





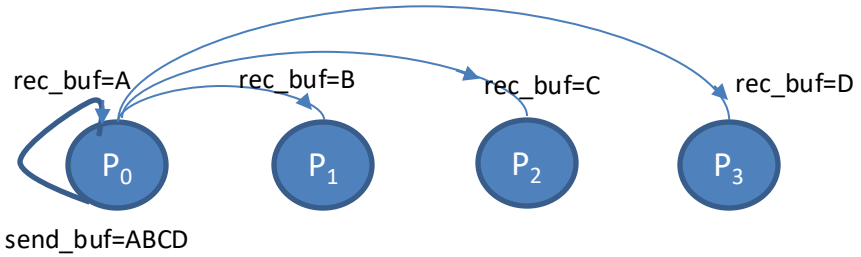


- n=nº de columnas de las matrices M y Mloc
- num_procs= nº de procesos
- proc= índice de proceso
- mb=nº de componentes de vloc=nº de filas de Mloc=n/num_procs
- sz= mb*n=nº de elementos de cada bloque de la matriz M=tamaño de las matrices Mloc
- &M[proc*sz] apunta al primer elemento del bloque de la matriz **M** que P0 envía a Pi
- Para acceder al elemento (i,j) de M y Mloc se usa la macro: #define M(i,j) M[(i)*n + (j)]
- &v[proc*mb] apunta al primer elemento del bloque del vector **v** que P0 envía a Pi

Ejercicio 3: primer paso (reparto de datos)

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
send_buf(P_0)	ABCD
send_buf(P_1)	
send_buf(P_2)	
send_buf(P_3)	

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

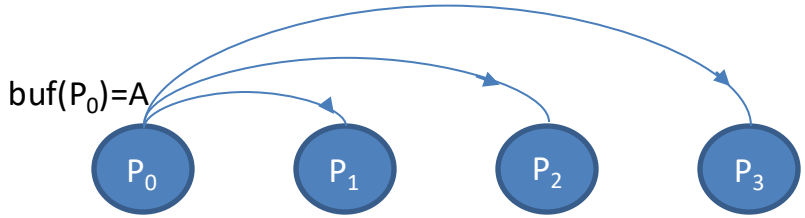
Código a implementar:

$$\underset{n}{M(P0)} \begin{bmatrix} M_0 \\ \hline M_1 \\ \hline M_2 \end{bmatrix} \begin{matrix} mb \\ mb \\ mb \end{matrix} \longrightarrow \underset{n}{Mloc} \begin{bmatrix} Mloc(P_0) = M_0 \\ \hline Mloc(P_1) = M_1 \\ \hline Mloc(P_2) = M_2 \end{bmatrix} \qquad v(P0) \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} \begin{matrix} mb \\ mb \\ mb \end{matrix} \longrightarrow vloc \begin{bmatrix} vloc(P_0)=v_0 \\ vloc(P_1)=v_1 \\ vloc(P_2)=v_2 \end{bmatrix}$$

Ejercicio 3: primer paso (Difusión)

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

Ejemplo:

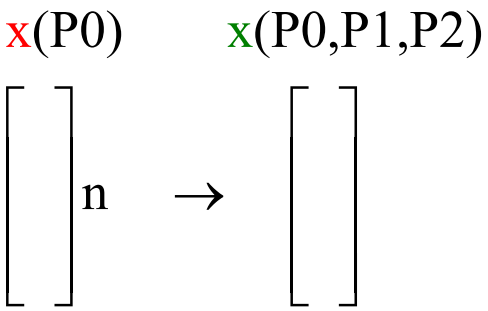


count=|A|
root=0

	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

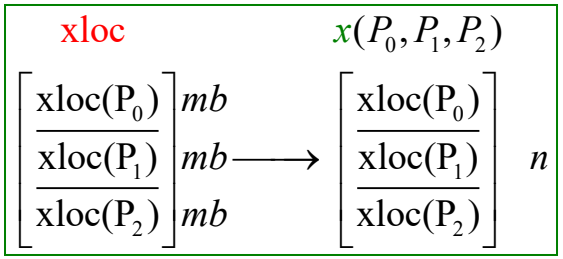
Código a implementar:



Ejercicio 3: segundo paso

2. Formar el vector x a partir de los vectores locales xloc, de manera que todos los procesos tengan el vector completo x (multirecogida): sustituir el código por una operación colectiva

```
170     /* COMMUNICATIONS */
171     /* Prepare for next iteration. Form the complete vector x (x)
172      * by assembling the fragments of x (xloc) in each process,
173      * and replicate it in all the processes */
174     if (me == 0) {
175       /* Put all the fragments of x in their place */
176       /* The first fragment is mine, so copy it */
177       for (i = 0; i < mb; i++)
178         x[i] = xloc[i];
179       /* The rest of the fragments must be received from other processes */
180       for (proc = 1; proc < num_procs; proc++)
181         MPI_Recv(&x[proc*mb], mb, MPI_DOUBLE, proc, 49,
182                  MPI_COMM_WORLD, MPI_STATUS_IGNORE);
183       /* Send the complete x to everyone */
184       for (proc = 1; proc < num_procs; proc++)
185         MPI_Send(x, n, MPI_DOUBLE, proc, 53, MPI_COMM_WORLD);
186     } else {
187       /* Send my fragment of x */
188       MPI_Send(xloc, mb, MPI_DOUBLE, 0, 49, MPI_COMM_WORLD);
189       /* Receive the complete vector x */
190       MPI_Recv(x, n, MPI_DOUBLE, 0, 53, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
191     }
```



num_procs = nº de procesos

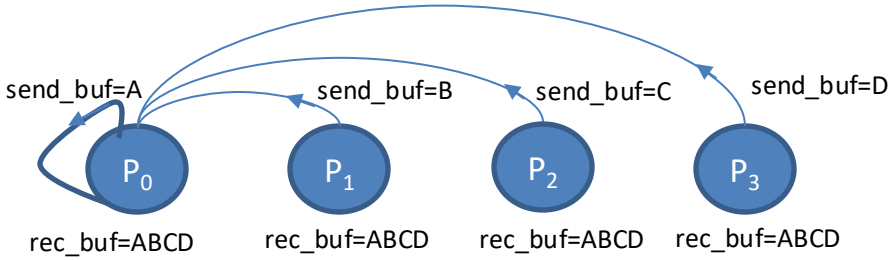
&v[proc*mb] apunta al primer elemento del bloque del vector **x** que P0 recibe del proceso **proc**

mb=nº de componentes de xloc

Ejercicio 3: segundo paso (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
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)	ABCD
rec_buf(P_1)	ABCD
rec_buf(P_2)	ABCD
rec_buf(P_3)	ABCD

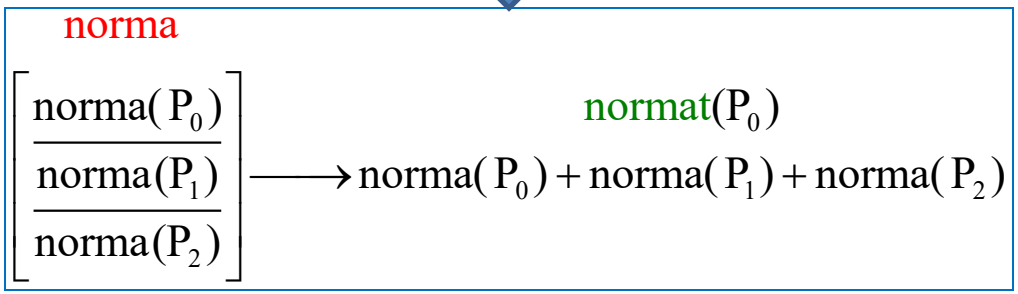
Código a implementar:

$$\text{xloc} \qquad x(P_0, P_1, P_2)$$
$$\begin{bmatrix} \text{xloc}(P_0) \\ \hline \text{xloc}(P_1) \\ \hline \text{xloc}(P_2) \end{bmatrix} \begin{matrix} mb \\ mb \\ mb \end{matrix} \longrightarrow \begin{bmatrix} \text{xloc}(P_0) \\ \hline \text{xloc}(P_1) \\ \hline \text{xloc}(P_2) \end{bmatrix} n$$

Ejercicio 3: tercer paso

3. Todos los procesos ayudan a calcular la 1-norma de x, quedando en P0 el resultado obtenido

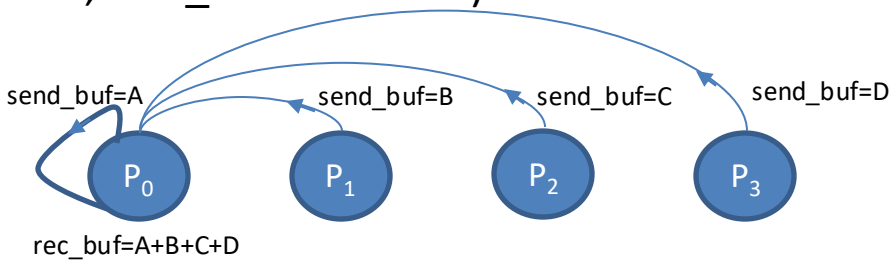
```
194 #define ABS(a) ((a) >= 0 ? (a) : -(a))
195   /* Compute the 1-norm of the local part of x */
196   norma = 0;
197   for (i = 0; i < mLoc; i++)
198     norma += ABS(xloc[i]);
199
200   /* COMMUNICATIONS */
201   /* Compute the 1-norm of the complete x from the 1-norm of each fragment,
202    * i.e. compute the sum of the local norms, leaving the result in process 0 */
203   if (me == 0) {
204     for (proc = 1; proc < num_procs; proc++) {
205       MPI_Recv(&aux, 1, MPI_DOUBLE, proc, 65, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
206       norma += aux;
207     }
208   } else {
209     MPI_Send(&norma, 1, MPI_DOUBLE, 0, 65, MPI_COMM_WORLD);
210   }
```



Ejercicio 3: tercer paso (Reducción)

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

Ejemplo (Suma):



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

	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+B+C+D
rec_buf(P_1)	
rec_buf(P_2)	
rec_buf(P_3)	

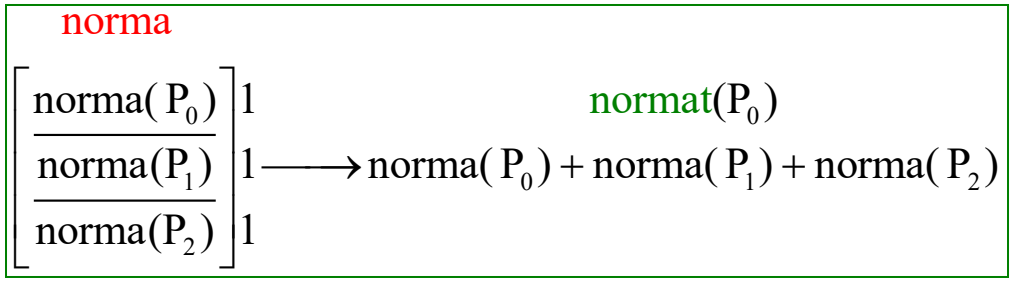
Código a implementar:

norma

$$\begin{bmatrix} \frac{\text{norma}(P_0)}{\text{norma}(P_1)} \\ \frac{\text{norma}(P_1)}{\text{norma}(P_2)} \end{bmatrix} \begin{matrix} 1 \\ 1 \\ 1 \end{matrix} \longrightarrow \text{normat}(P_0) \quad \text{norma}(P_0) + \text{norma}(P_1) + \text{norma}(P_2)$$

Ejercicio 3: tercer paso (Reducción)

Código a implementar:



Como para la reducción hay que declarar y usar una nueva variable (normat), debes cambiar la variable norma por normat en el printf que aparece a continuación en el código:

```
if (me == 0) {
  printf("1-norm (%d iterations): %.10g\n", num_iter, normat);
  printf("Time using %d processes: %.2f\n", num_procs, t);
}
```

Ejercicio 4

- El código de mxv2.c resuelve también sistemas de ecuaciones lineales, pero la matriz A se almacena por columnas y se reparte dicha matriz entre los procesos por bloques de columnas.
- Sustituir en el código mxv2.c todas las comunicaciones que son susceptibles de ser realizadas mediante operaciones colectivas por las correspondientes llamadas a las funciones de MPI de comunicación colectiva.
- Las comunicaciones que se deben sustituir se encuentran especificadas mediante: /* COMUNICACIONES */
- Compila y ejecuta la implementación realizada
- Una vez que se tengan las versiones modificadas de mxv1.c y mxv2.c, es interesante comparar los tiempos respecto a las versiones originales, usando el sistema de colas

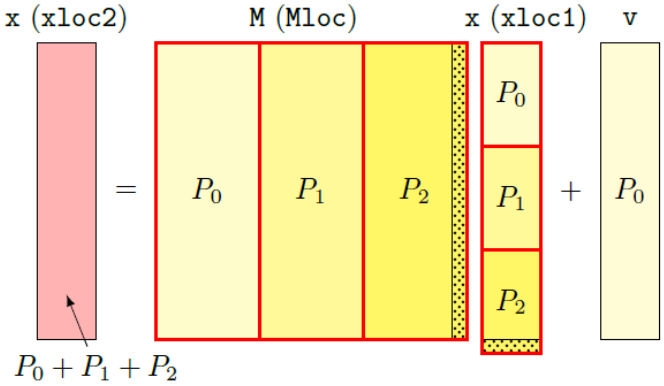


Figura 8: Distribución de datos en `mxv2.c`

```
11  /* NOTE: We work with matrices stored by columns */
12  #define M(i,j) M[(i) + (j)*n]
13  #define Mloc(i,j) Mloc[(i) + (j)*n]
```

Ejercicio 4

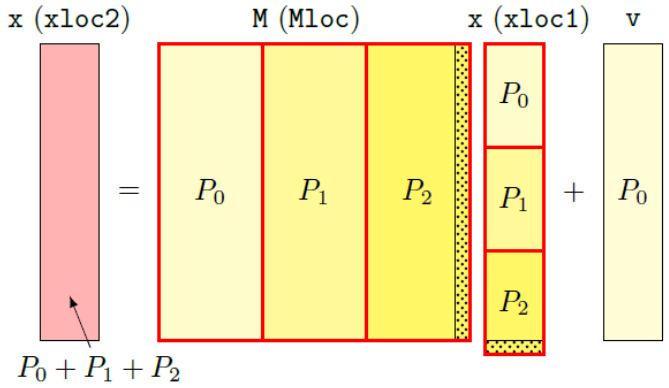


Figura 8: Distribución de datos en `mxv2.c`

```
/* Calcular Mloc*xloc1 en xloc2 */
for ( i = 0 ; i < n ; i++ ) {
  aux = 0;
  for ( j = 0 ; j < ncoluscalculo ; j++ )
    aux += Mloc(i,j) * xloc1[j];
  xloc2[i] = aux;
}
```

```
if ( me == 0 )
  for ( i = 0 ; i < n ; i++ )
    x[i] += v[i];
```

Algoritmo:

P0 reparte la matriz M quedando en Mloc(Pi) las matrices repartidas

P0 reparte el vector x quedando en xloc1(Pi) los vectores repartidos

Primer COMMUNICATIONS

Para i=1,2, …, num_iter (bucle de la línea 160)

En dos bucles anidados, cada Pi calcula: xloc2(Pi)=Mloc(Pi)*xloc1(Pi)

x(P0)=Σxloc2(Pi) (Reducción)

Mediante un bucle, P0 calcula: x(P0)=x(P0)+v(P0)

P0 reparte el vector x quedando en xloc1(Pi) los vectores repartidos

Segundo COMMUNICATIONS

Calcular la 1-norma del vector s (tercer COMMUNICATIONS)