Práctica 3

Sesión 3

Objetivo

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

Descripción del problema

- Resolver sistemas de ecuaciones lineales

$$Ax = b;\ \ A \in \Re^{n\times n}, b \in \Re^{n}$$

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)

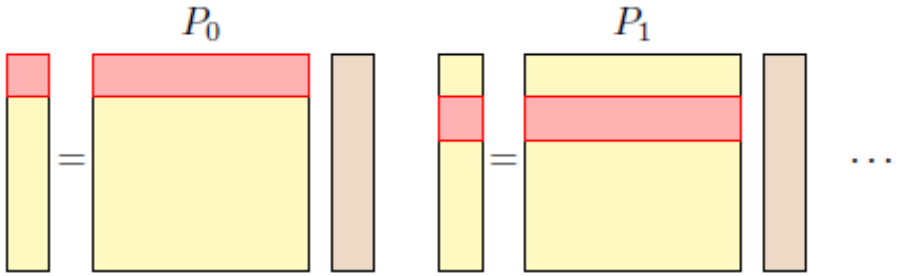
Algoritmo secuencial

```
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
```

- `num_iter:` haremos un número fijo de iteraciones.
- La norma nos sirve como *hash*, para saber si las ejecuciones son correctas
- Tenemos una implementación paralela por filas (mxv1.c) y otra por columnas (`mxv2.c`)

Descripción de las implementaciones de partida

- Disponemos de dos versiones que difieren en la forma de almacenar y repartir los datos entre los distintos procesadores (el resultado de las dos versiones no es comparable entre sí, porque cada una trabaja con un problema diferente).
  - En la primera versión (mxv1.c) almacenamos la matriz M por filas y la repartimos por bloques de filas entre los procesadores:



  - En la segunda versión (mxv2.c) la matriz se almacena por columnas y se reparte por bloques de columnas entre los procesadores.
- Los ejercicios 3 y 4 consisten en sustituir las operaciones de comunicación punto a punto por operaciones colectivas.

Ejercicios 1 y 2

- Compilar y ejecutar el programa mxv1.c

```
$ mpicc mxv1.c –o mxv1
```

Lo ejecutamos sobre el frontend para probar:

```
$ mpiexec –n 3 mxv1 –n 5 –i 5
```

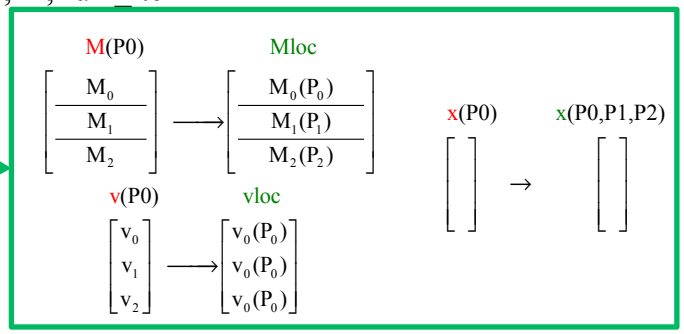
- Analizar el código mxv1.c para entender como funciona.

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

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

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



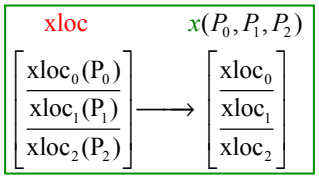
Para i=1,2,…, num_iter

Cada Pi **calcula** su parte local:

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

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

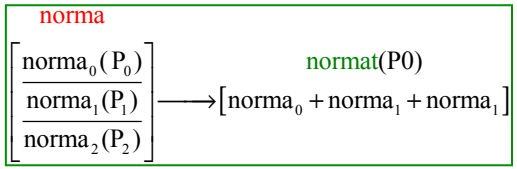
Multirecogida de los vectores xloc, para que todos los procesos tengan el vector completo x:



Fin para

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

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



Ejercicio 3

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

```
/* COMUNICACIONES */
/* Repartir la M y la v por filas (nfilas a cada proceso)
 * y enviar la x inicial a todos */
if (yo == 0) {
  /* Para el 0, no enviamos sino que copiamos */
  for (i = 0; i < nfilas; i++) {
    for (j = 0; j < n; j++)
      Mloc(i, j) = M(i, j);
    vloc[i] = v[i];
  }
  /* Para el resto, enviamos */
  for (proc = 1; proc < num_procs; proc++) {
    MPI_Send(&M[proc*tama], tama, MPI_DOUBLE, proc, 13, MPI_COMM_WORLD);
    MPI_Send(&v[proc*nfilas], nfilas, MPI_DOUBLE, proc, 89, MPI_COMM_WORLD);
    MPI_Send(x, n, MPI_DOUBLE, proc, 25, MPI_COMM_WORLD);
  }
} else {
  MPI_Recv(Mloc, tama, MPI_DOUBLE, 0, 13, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  MPI_Recv(vloc, nfilas, MPI_DOUBLE, 0, 89, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  MPI_Recv(x, n, MPI_DOUBLE, 0, 25, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```

Ejercicio 3

- Formar el vector x a partir de los vectores locales xloc, de manera que todos los procesos tengan el vector completo x (multirecogida):

```
/* COMUNICACIONES */
/* Preparar la siguiente iteración. Esto es formar la x completa (x)
 * a partir de los fragmentos de x (xloc) que tiene cada proceso
 * y dejarla replicada en todos */
if (yo == 0) {
  /* Poner todos los fragmentos de x cada uno en su sitio */
  /* El primer fragmento lo tengo yo -> copiarlo */
  for (i = 0; i < nfilas; i++)
    x[i] = xloc[i];
  /* El resto de fragmentos hay que recibirlos de los otros procesos */
  for (proc = 1; proc < num_procs; proc++)
    MPI_Recv(&x[proc*nfilas], nfilas, MPI_DOUBLE, proc, 49,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  /* Enviar la x ya completa a todos */
  for (proc = 1; proc < num_procs; proc++)
    MPI_Send(x, n, MPI_DOUBLE, proc, 53, MPI_COMM_WORLD);
} else {
  /* Enviar mi fragmento de x */
  MPI_Send(xloc, nfilas, MPI_DOUBLE, 0, 49, MPI_COMM_WORLD);
  /* Recibir toda la x */
  MPI_Recv(x, n, MPI_DOUBLE, 0, 53, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```

Ejercicio 3

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

```
#define ABS(a) ((a) >= 0 ? (a) : -(a))
  /* Calcular la 1-norma de la parte de x que tiene cada uno */
  norma = 0;
  for (i = 0; i < nfilascalculo; i++)
    norma += ABS(xloc[i]);

  /* COMUNICACIONES */
  /* Calcular la 1-norma de toda la x a partir de la 1-norma de cada fragmento.
   * Esto es calcular la suma de todas dejándola en el proceso 0. */
  if (yo == 0) {
    for (proc = 1; proc < num_procs; proc++) {
      MPI_Recv(&aux, 1, MPI_DOUBLE, proc, 65, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      norma += aux;
    }
  } else {
    MPI_Send(&norma, 1, MPI_DOUBLE, 0, 65, MPI_COMM_WORLD);
  }
```

Ejercicio 3

- Etapa inicial:
  - Utilizar MPI_Scatter para la distribución
  - Utilizar MPI_Bcast para la difusión

$$Mx^{(0)} + v;\ M \in \Re^{n \times n}, v \in \Re^{n}$$

P_0 difunde el vector inicial x y distribuye la matriz M y el vector v:

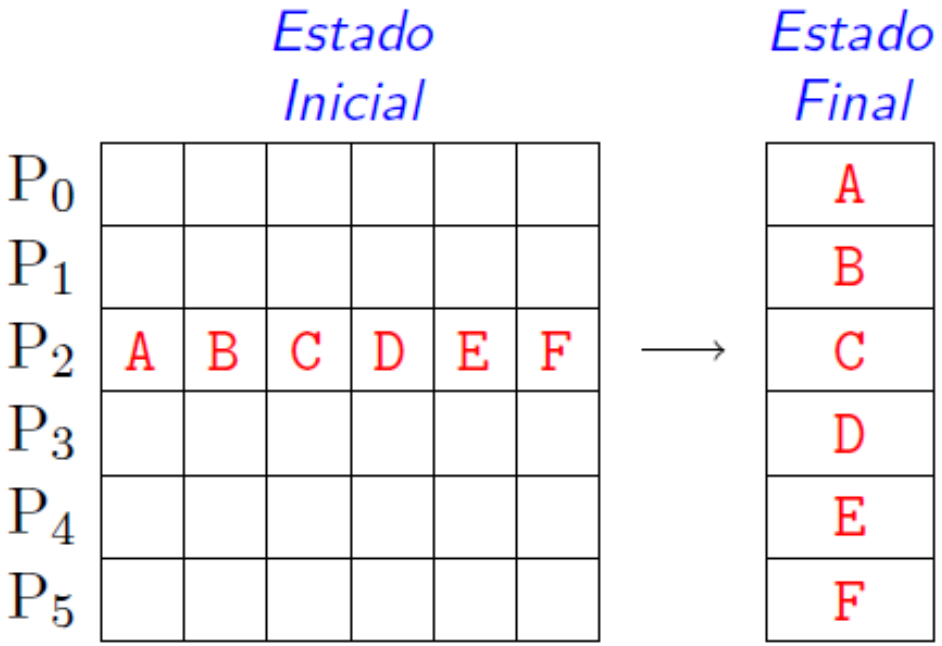
M distribuida x difundido v distribuido:

$$\left[\begin{array}{c} Mloc(P_0) \\ \hline Mloc(P_1) \\ \hline Mloc(P_2) \end{array}\right] \qquad \left[\ \right] \qquad \left[\begin{array}{c} vloc(P_0) \\ vloc(P_1) \\ vloc(P_2) \end{array}\right]$$

Reparto

```
MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf,
            recvcount, recvtype, root, comm)
```

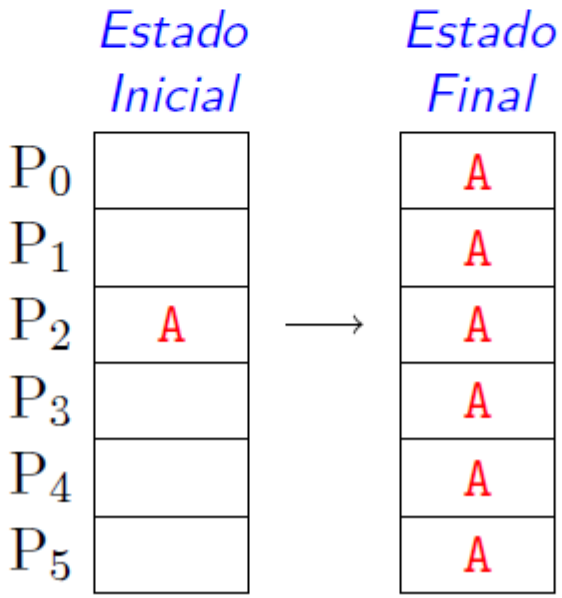
El proceso `root` distribuye una serie de datos al resto de procesos



Difusión

```
MPI_Bcast(buffer, count, datatype, root, comm)
```

El proceso `root` difunde al resto de procesos el mensaje definido por los 3 primeros argumentos



Ejercicio 3

- Sustituir las operaciones punto a punto por una operación colectiva (MPI_Allgather):

Todos recogen el vector x a partir de los vectores $xloc$:

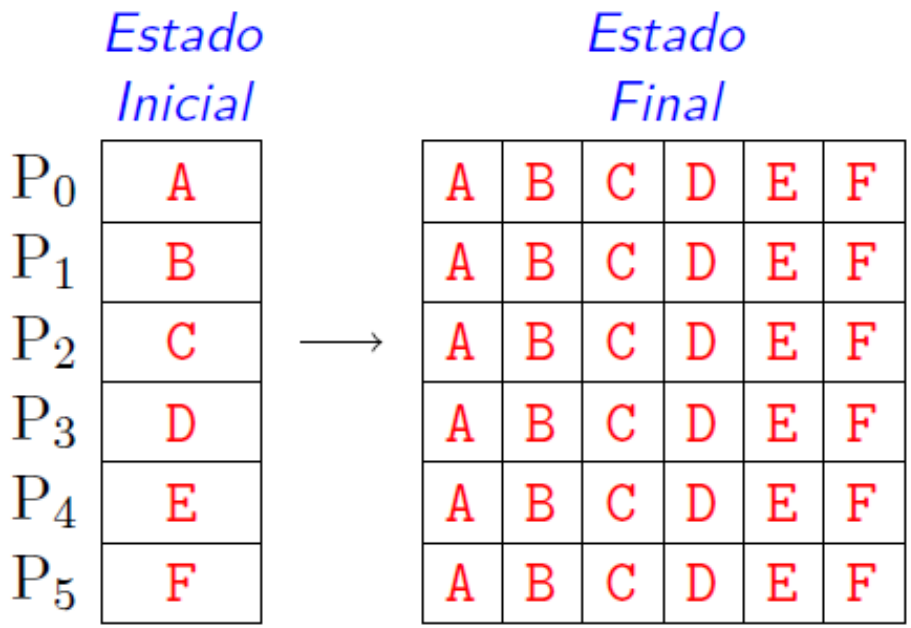
$$x(P_0, P_1, P_2)$$

$$\begin{bmatrix} xloc(\mathrm{P}_0) \\ xloc(\mathrm{P}_1) \\ xloc(\mathrm{P}_2) \end{bmatrix} \longrightarrow \begin{bmatrix} \\ \\ \end{bmatrix}$$

Multi-Recogida

```
MPI_Allgather(sendbuf, sendcount, sendtype, recvbuf,
              recvcount, recvtype, comm)
```

Similar a la operación `MPI_Gather`, pero todos los procesos obtienen el resultado



Ejercicio 3

- Calcular la 1-norma del vector x

```
#define ABS(a) ((a) >= 0 ? (a) : -(a))
  /* Calcular la 1-norma de la parte de x que tiene cada uno */
  norma = 0;
  for (i = 0; i < nfilascalculo; i++)
    norma += ABS(xloc[i]);

  /* COMUNICACIONES */
  /* Calcular la 1-norma de toda la x a partir de la 1-norma de cada fragmento.
   * Esto es calcular la suma de todas dejándola en el proceso 0. */
  if (yo == 0) {
    for (proc = 1; proc < num_procs; proc++) {
      MPI_Recv(&aux, 1, MPI_DOUBLE, proc, 65, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      norma += aux;
    }
  } else {
    MPI_Send(&norma, 1, MPI_DOUBLE, 0, 65, MPI_COMM_WORLD);
  }
```

Ejercicio 3

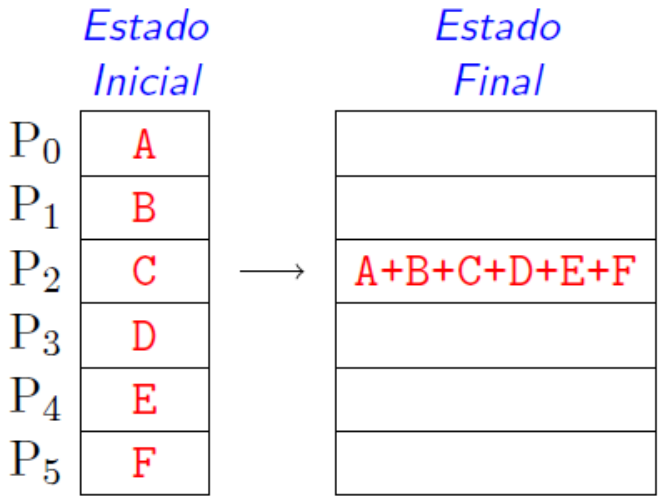
- Sustituir las comunicaciones punto a punto por una operación colectiva (reducción):

Reducción

```
MPI_Reduce(sendbuf, recvbuf, count, datatype, op,
           root, comm)
```

Además de la comunicación se realiza una operación aritmética o lógica (suma, max, and, ..., o definida por el usuario)

El resultado final se devuelve en el proceso `root`



Ejercicio 3

- Forma de ejecutarlo:

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

- Por defecto:
  - semilla→ 0
  - Tamaño→10000
  - Iteraciones→50

- Recordad lanzarlo por cola.

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 */

Ejercicio 4

- Forma de ejecutarlo:

  mpiexec… mxv2 [−s semilla] [-n tamanyo] [-i iteraciones]

- Por defecto:
  - semilla→ 0
  - Tamaño→10000
  - Iteraciones→50

- Recordad lanzarlo por cola.