Práctica 3: Paralelización con MPI
Sesión 3: Producto matriz-vector

Mónica Chillarón

Computación Paralela (CPA)

Curso 2020/2021

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Índice

1. Descripción del problema

2. Programa con distribución por bloques de filas (mxv1)

3. Programa con distribución por bloques de columnas (mxv2)

4. Comunicaciones colectivas

Descripción del problema

- Resolución de sistema de ecuaciones mediante método iterativo:

$$x^{k+1} = Mx^k + v;$$

$$\mathrm{M} \in \Re^{n \times n}, v \in \Re^n, k = 1, 2, 3, ...$$

- M y v son conocidos
- Partimos de una solución inicial x^0 y vamos generando una sucesión de soluciones aproximadas
- En cada iteración: producto matriz-vector y suma vectorial

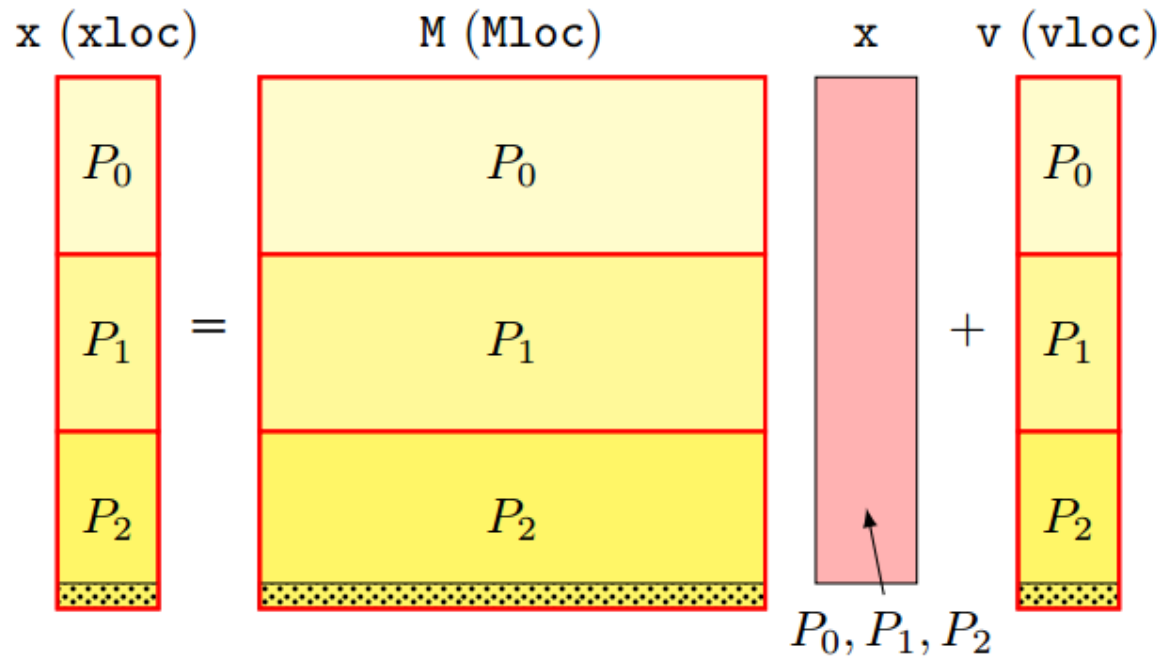
Descripción del problema

```
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
```

- No iteramos hasta convergencia, sino hasta un número máximo de iteraciones
- Se calcula 1-norma del vector x resultado (la suma de los elementos de x, en valor absoluto)
- Utilizamos la norma como hash para comprobar si el resultado es correcto

- **Dos implementaciones en MPI: reparto por bloques de filas o por bloques de columnas**

Programa con distribución por bloques de filas (mxv1)



- Cada proceso tiene un bloque de filas de M (Mloc), todo el vector x, y las componentes correspondientes de v (vloc)
- Cada proceso calcula las componentes correspondientes de la nueva solución x (xloc)
- Si n (tamaño de la matriz y vectores) no es divisible por el número de procesos, se expande para que lo sea (zona punteada)
- De esta manera todos los procesos tienen el mismo número de filas (mb), aunque no se tengan en cuenta para la solución

Programa con distribución por bloques de filas (mxv1)

Ejercicio 1: Compila el programa `mxv1.c` y ejecútalo. Puedes hacer alguna ejecución con un tamaño muy pequeño directamente en el *front-end*, por ejemplo:

```
$ mpiexec -n 3 mxv1 -n 5 -i 5
```

La orden anterior ejecuta el programa con 3 procesos, con un tamaño de matriz de 5 filas y columnas (`-n 5`) y 5 iteraciones (`-i 5`). Para ejecuciones más largas se debe utilizar el sistema de colas.

Ejercicio 2: Analiza el código de `mxv1.c` para entender cómo se realiza el producto matriz-vector y la suma de vectores.

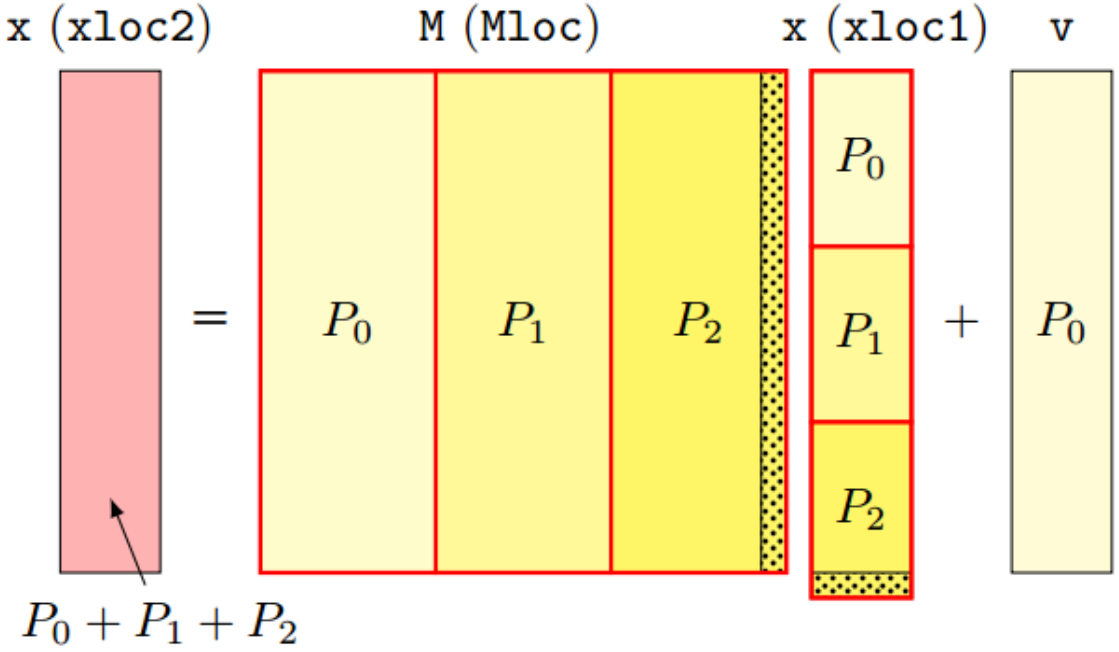
Programa con distribución por bloques de filas (mxv1)

- Todas las comunicaciones del programa mxv1.c están implementadas con operaciones punto a punto (MPI_Send y MPI_Recv):
  1. Reparto inicial de M en Mloc
  2. Reparto inicial de v en vloc
  3. Difusión inicial de x
  4. Recogida de los fragmentos de x calculados (xloc)
  5. Difusión de la nueva x para la iteración siguiente
  6. Recogida de la parte de la norma calculada por cada proceso y suma final

Ejercicio 3: Modifica el programa `mxv1.c` para sustituir las comunicaciones punto a punto por comunicaciones colectivas, en aquellos puntos del programa donde sea posible. En el código estos puntos están marcados con un comentario previo con la palabra `COMMUNICATIONS`.

Se aconseja no esperar a tener todas las comunicaciones cambiadas para probar el programa. Es conveniente ir probando el programa tras cambiar cada comunicación y así asegurarse de que no se ha alterado el resultado con el cambio.

Programa con distribución por bloques de columnas (mxv2)



- Cada proceso tiene un bloque de columnas de M (Mloc), las componentes correspondientes de x (xloc1)
- Cada proceso calcula su contribución al vector x (xloc2)
- El proceso 0 suma todas las xloc2 y el vector v (nueva solución x)
- Si n (tamaño de la matriz y vectores) no es divisible por el número de procesos, se expande para que lo sea (zona punteada)
- De esta manera todos los procesos tienen el mismo número de columnas (nb), aunque no se tengan en cuenta para la solución

Programa con distribución por bloques de columnas (mxv2)

- Todas las comunicaciones del programa mxv2.c están implementadas con operaciones punto a punto (MPI_Send y MPI_Recv):
  1. Reparto inicial de M en Mloc
  2. Reparto inicial de x en xloc1
  3. Recogida de las aportaciones a x calculadas (xloc2) y proceso 0 hace las sumas
  4. Reparto de la nueva x para la iteración siguiente (xloc1)
  5. Recogida de la parte de la norma calculada por cada proceso y suma final

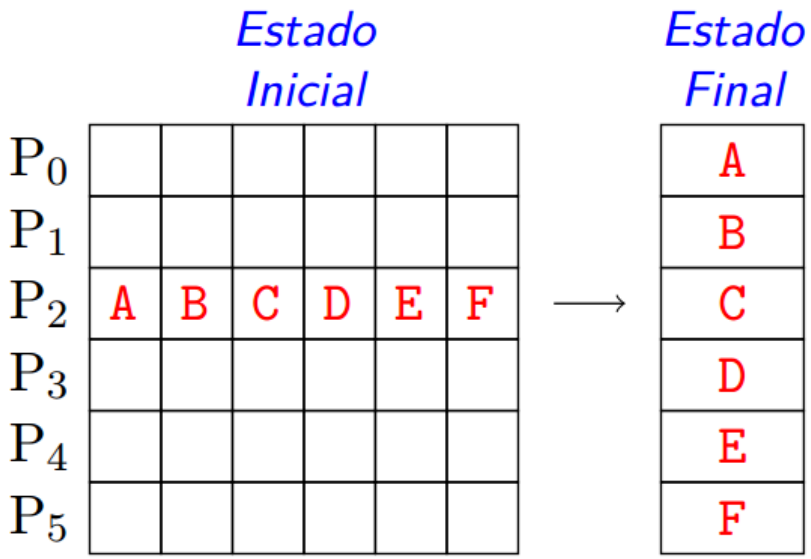
Ejercicio 4: Modifica el programa `mxv2.c` para sustituir las comunicaciones punto a punto por comunicaciones colectivas, en aquellos puntos del programa donde sea posible. En el código estos puntos están marcados con un comentario previo con la palabra `COMMUNICATIONS`.

Una vez que se tengan las versiones modificades de `mxv1.c` y `mxv2.c`, es interesante comparar los tiempos respecto a las versiones originales. Aunque es probable que no se encuentren muchas diferencias en este problema en concreto.

Reparto

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

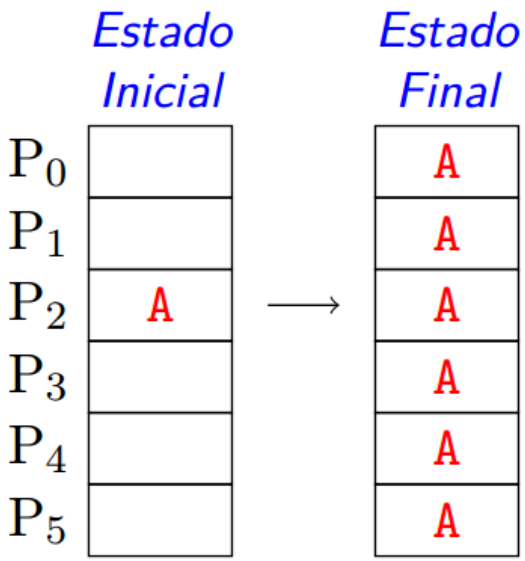
El proceso `root` distribuye una serie de fragmentos consecutivos del buffer al resto de procesos (incluyendo él mismo)



Difusión

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

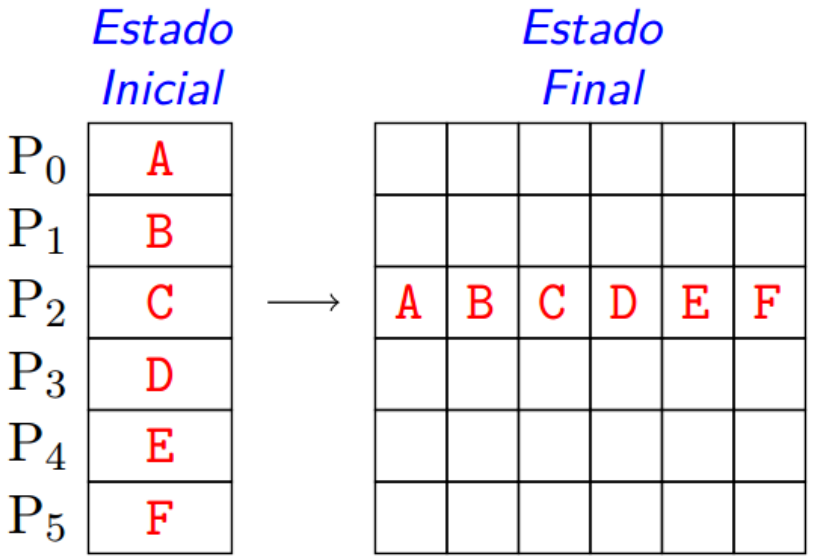
El proceso `root` difunde al resto de procesos el mensaje definido por los 3 primeros argumentos



Recogida

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

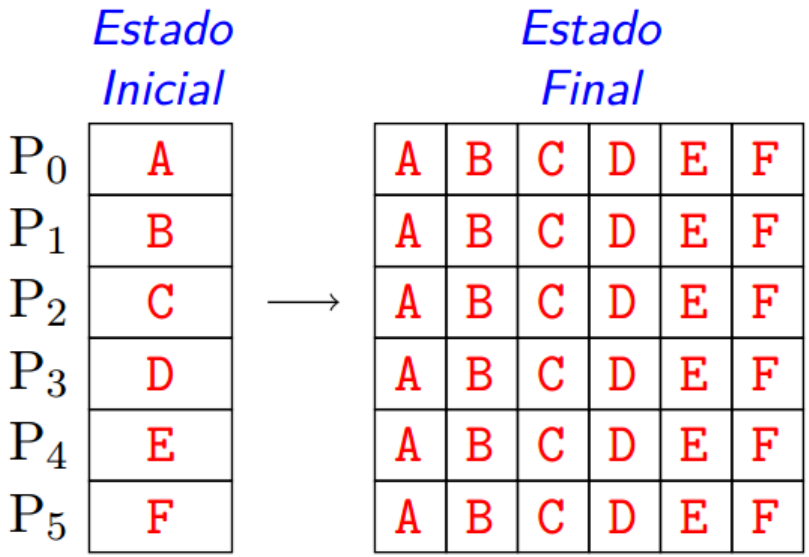
Es la operación inversa de `MPI_Scatter`: Cada proceso envía un mensaje a `root`, el cual lo almacena de forma ordenada de acuerdo al índice del proceso en el buffer de recepción



Multi-recogida

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

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



Reducción

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

Similar a `MPI_Gather`, pero en lugar de concatenación, se realiza una operación aritmética o lógica (suma, max, and, ..., o definida por el usuario)

El resultado final se devuelve en el proceso `root`