Práctica 3: Paralelización con MPI
Sesión 4: Sistemas de ecuaciones lineales

Mónica Chillarón

Computación Paralela (CPA)

Curso 2020/2021

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Índice

Resolución de sistemas de ecuaciones lineales

Resolución del sistema de ecuaciones lineales $\boldsymbol{Ax=b}$

- A es una matriz (cuadrada de tamaño n)
- b es el vector de términos independientes
- x es el vector solución

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{bmatrix}, \quad x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}, \quad b = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix}.$$

- La primera ecuación del sistema sería:

$$a_{0,0}x_0 + a_{0,1}x_1 + \cdots + a_{0,n-1}x_{n-1} = b_0.$$

- Se busca el vector x que satisface todas las ecuaciones simultáneamente

Resolución de sistemas de ecuaciones lineales

- Factorizamos la matriz A en dos matrices triangulares superior e inferior: L (lower) y U (upper)

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ L_{21} & 1 & 0 & 0 \\ L_{31} & L_{32} & 1 & 0 \\ L_{41} & L_{42} & L_{43} & 1 \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ 0 & U_{22} & U_{23} & U_{24} \\ 0 & 0 & U_{33} & U_{34} \\ 0 & 0 & 0 & U_{44} \end{bmatrix}.$$

- Resolvemos el sistema de ecuaciones:

$$\left.\begin{array}{c} A = LU \\ Ax = b \end{array}\right\} \longrightarrow \quad LUx = b \quad \longrightarrow \quad \left\{\begin{array}{c} Ly = b \\ Ux = y \end{array}\right. .$$

Programa paralelo proporcionado

El programa proporcionado (fichero `sistbf.c`) genera un sistema lineal $Ax = b$ y lo resuelve, realizando para ello los siguientes pasos, marcados en el código con comentarios con el texto "STEP":

1. **Generar los datos**. El proceso 0 genera la matriz (`A`) y el vector (`b`) completos. Todos los procesos (incluido el 0) reservan memoria para su matriz local (`Aloc`).

2. **Distribuir los datos**. La matriz se distribuye entre los procesos por bloques de `mb` filas consecutivas. El vector `b` se replica en todos los procesos.

3. **Descomposición LU**. En esta fase la matriz `A` se sobreescribe por L y U. Los elementos de L quedan en la parte inferior de `A` y los de U en la parte superior.

4. **Resolver el sistema triangular inferior** $Ly = b$. El vector y se almacena sobre `b`, sobreescribiéndolo.

5. **Resolver el sistema triangular superior** $Ux = y$. El vector y se encuentra almacenado en la variable `b`, y en esta fase se sobreescribe con el vector x.

Programa paralelo proporcionado

Ejercicio 1: Compila y ejecuta el programa. Se pueden ejecutar pruebas cortas directamente en el *front-end* de `kahan`. Por ejemplo, para resolver un sistema de 5 ecuaciones usando 3 procesos:

```
$ mpiexec -n 3 sistbf 5
```

En este caso el sistema que se resuelve es:

$$\begin{bmatrix} 25 & 4 & 3 & 2 & 1 \\ 4 & 25 & 4 & 3 & 2 \\ 3 & 4 & 25 & 4 & 3 \\ 2 & 3 & 4 & 25 & 4 \\ 1 & 2 & 3 & 4 & 25 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 35 \\ 38 \\ 39 \\ 38 \\ 35 \end{bmatrix}.$$

- Reparto de datos inicial:

```
Matrix A:
---- proc. 0 ----
25.000 4.000 3.000 2.000 1.000
11
4.000 25.000 4.000 3.000 2.000
---- proc. 1 ----
3.000 4.000 25.000 4.000 3.000
2.000 3.000 4.000 25.000 4.000
---- proc. 2 ----
1.000 2.000 3.000 4.000 25.000
```

```
Vector b:
---- proc. 0 ----
35.000 38.000 39.000 38.000 35.000
---- proc. 1 ----
35.000 38.000 39.000 38.000 35.000
---- proc. 2 ----
35.000 38.000 39.000 38.000 35.000
```

- La solución siempre es un vector de unos y el error es 0
- Objetivo: Pasar a una distrubución cíclica por filas

Fase de distribución de datos

Ejercicio 2: Cambia la forma de distribuir la matriz, para que se haga **cíclicamente por filas**. Hay distintas formas de implementar este reparto, una de las cuales consiste en hacer **múltiples operaciones tipo scatter**:

En una distribución cíclica entre p procesos, las primeras p filas de la matriz A van cada una a un proceso, lo cual corresponde a una operación scatter. Lo mismo ocurre con las siguientes p filas, y así sucesivamente. Por tanto, la distribución de la matriz entera se puede hacer mediante un **bucle en el que cada iteración corresponde a una operación scatter**. Hay que prestar atención a:

- La posición (sobre la matriz global A) donde empiezan los datos a enviar en cada scatter.
- La posición (sobre la matriz local Aloc) donde deben recibirse los datos en cada scatter.

Fase de distribución de datos

Una vez hayas cambiado esta fase, ejecuta el programa y comprueba que la distribución se realiza correctamente. Por ejemplo, al ejecutar:

```
$ mpiexec -n 3 sistcf 5
```

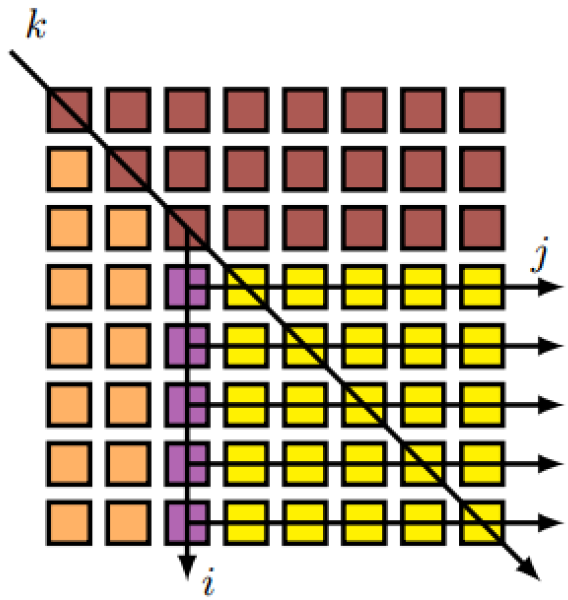
La matriz `A` debería haberse distribuido de la siguiente manera:

```
Matrix A:
---- proc. 0 ----
 25.000   4.000   3.000   2.000   1.000
  2.000   3.000   4.000  25.000   4.000
---- proc. 1 ----
  4.000  25.000   4.000   3.000   2.000
  1.000   2.000   3.000   4.000  25.000
---- proc. 2 ----
  3.000   4.000  25.000   4.000   3.000
```

- Los resultados de esta ejecución (vector solución y error) aún no son correctos puesto que hacen falta más cambios.

Fase de descomposición LU

```
Para k = 0, ..., n-2
  si A(k,k) = 0 entonces abandona
  Para i = k+1, ..., n-1
    /* Modificar fila i (elementos de la columna k a la n-1) */
    A(i,k) = A(i,k)/A(k,k)
    Para j = k+1, ..., n-1
      A(i,j) = A(i,j) - A(i,k)*A(k,j)
    Fin_para
  Fin_para
Fin_para
```



- Se paraleliza el bucle i del algoritmo secuencial. Cada iteración actualiza una fila (la fila i). La actualización de cada fila es independiente.
- Cada proceso actualiza las filas que posee (entre k+1 y n-1)
- La fila k (fila pivote) sólo la tiene uno de los procesos
- Habrá que difundir la fila k antes de empezar la actualización de filas

Fase de descomposición LU

Algortimo paralelo:

```
Para k = 0, ..., n-2
  Si propietario(k) = yo
     si A(iloc(k),k) = 0 entonces abandona
  Fin_si
  difundir fila k
  Para i = k+1, ..., n-1
    /* Modificar fila i (elementos de la columna k a la n-1) */
    Si propietario(i) = yo
      A(iloc(i),k) = A(iloc(i),k)/A(k,k)
      Para j = k+1, ..., n-1
        A(iloc(i),j) = A(iloc(i),j) - A(iloc(i),k)*A(k,j)
      Fin_para
    Fin_si
  Fin_para
Fin_para
```

- **propietario(i)** representa el proceso propietario de la fila i
- **iloc(i)** representa el índice local (en Aloc) de la fila i de A.

Fase de resolución sistemas triangulares

Triangular Inferior

```
Para i = 0, 1, ..., n-1
  Para j = i+1, ..., n-1
    b(j) = b(j) - L(j,i)*b(i)
  Fin_para
Fin_para
```

Triangular Superior

```
Para i = n-1, ..., 0
  b(i) = b(i)/U(i,i)
  Para j = i-1, ..., 0
    b(j) = b(j) - U(j,i)*b(i)
  Fin_para
Fin_para
```

- Se resuelven los sistemas Lx = b y Ux = b,
- En ambos casos, el vector b se sobreescribe con la solución del sistema.
- En cada iteración del bucle i se actualizan, mediante el bucle j, los elementos del vector b que hay por debajo del elemento i (en el sistema triangular inferior) o por encima (en el sistema triangular superior). Esta actualización requiere usar el elemento b(i).

Fase de resolución sistemas triangulares

Algoritmo paralelo:

Triangular Inferior

```
Para i = 0, 1, ..., n-1
  difundir b(i)
  Para j = i+1, ..., n-1
    Si propietario(j) = yo
      b(j) = b(j) - L(iloc(j),i)*b(i)
    Fin_si
  Fin_para
Fin_para
```

Triangular Superior

```
Para i = n-1, ..., 0
  Si propietario(i) = yo
    b(i) = b(i)/U(iloc(i),i)
  Fin_si
  difundir b(i)
  Para j = i-1, ..., 0
    Si propietario(j) = yo
      b(j) = b(j) - U(iloc(j),i)*b(i)
    Fin_si
  Fin_para
Fin_para
```

- Paralelizar el bucle j. Cada proceso actualice los elementos de las filas que posee.
- Para ello el valor de b(i) deberá ser propagado.
- Al final, todos los procesos acaban con una copia del vector de incógnitas completo.

Modificación final

Ejercicio 3: Modifica las funciones propietario e iloc (funciones **owner** y **localIndex** en el código proporcionado, respectivamente) para que correspondan a una distribución cíclica en vez de a una distribución por bloques. El comportamiento de estas funciones debe ser el siguiente:

- Dado un índice de fila i de la matriz global A, la función owner debe devolver el índice del proceso que tiene esa fila en su matriz local Aloc.
- Dado un índice de fila i de la matriz global A, la función localIndex debe devolver el índice de dicha fila en la matriz local Aloc del proceso propietario de la fila.

También es necesario modificar la función **numLocalRows**, que devuelve el número de filas locales de la matriz en un proceso. Sólo hay que descomentar la parte de código que corresponde a la distribución cíclica, y comentar o eliminar la otra parte.

Comprobar que funciona correctamente con 3 procesos y un sistema de 5 ecuaciones.

Modificación final

Ejercicio 4: Una vez realizada la versión cíclica, obtén resultados experimentales de las dos variantes, comparando las prestaciones de ambas. Utiliza un tamaño del sistema suficientemente grande (entre 1000 y 2000).

Es importante evitar que el programa muestre por pantalla las matrices y vectores, ya que eso hará que tarde considerablemente más. Para ello, basta con comentar la línea:

#define VERBOSE

Analiza cuál de las dos versiones es más eficiente y trata de razonar a qué puede deberse.