Práctica 2
Sesión 4

Objetivos

- Resolver sistemas de ecuaciones lineales mediante la descomposición LU
- Estudio del problema y de la implementación paralela basada en el reparto por bloques de filas
- Transformación a la implementación paralela basada en el reparto cíclico de filas

4. Sistemas de ecuaciones lineales

- Resolver sistemas de ecuaciones lineales *Ax=b,* donde $A \in \Re^{n \times n}$, $b \in \Re^{n}$ mediante la descomposición LU:

$$\underset{A}{\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}} = \underset{L}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{pmatrix}} \underset{U}{\begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix}}$$

- El material de partida para realizar la práctica consiste en el fichero **sistbf.c**, que implementa una versión paralela donde la matriz del problema **se distribuye por bloques de filas**.
- La práctica consistirá en modificarla para que utilice en su lugar una **distribución cíclica por filas**.
- Se sugiere que hagas una copia del fichero sistbf.c, por ejemplo con el nombre **sistcf.c**, que contendrá la implementación cíclica por filas.

4.1 Resolución de sistemas de ecuaciones lineales

- Pasos para resolver *Ax=b*:
  - Obtener la descomposición *LU de A:*

$$A = LU$$

  - Desacoplar el sistema *Ax=b:*

$$Ax = b \leftrightarrow L\underbrace{Ux}_{y} = b \leftrightarrow \begin{cases} Ly = b \\ Ux = y \end{cases}$$

    - Resolver el sistema triangular inferior unidad ***Ly=b***
    - Resolver el sistema triangular superior ***Ux=y***

4.2 Programa paralelo proporcionado

El programa **sistbf.c** genera un sistema lineal *Ax=b* y lo resuelve realizando una serie de pasos marcados en el texto con "STEP":

1. **Generar los datos**. El proceso P_0 genera la matriz ***A***, y el vector ***b*** completos. Todos los procesos (incluido P_0), reservan memoria para su matriz local (***Aloc***). Este paso es común para las dos implementaciones.
2. **Distribuir los datos.** La matriz ***A*** 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 sobre-escribe por ***L*** y por ***U***. Los elementos de ***L*** quedan en la parte inferior de ***A*** y los de ***U*** en la superior.
4. **Resolver el sistema triangular inferior *Ly=b***. El vector ***y*** se almacena en *b*.
5. **Resolver el sistema triangular superior *Ux=*y**. El vector ***y***, el cual se encuentra almacenado en el vector ***b***, se almacena en el vector ***x***.

4.2 Programa paralelo proporcionado

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

```
mpicc –o sistbf sistbf.c
mpiexec –n 3 sistbf 5
```

$$\begin{bmatrix} 25 & 4 & 3 & 2 & 1 \\ 4 & 25 & 4 & 3 & 2 \\ 3 & 4 & 25 & 5 & 3 \\ 2 & 3 & 4 & 25 & 4 \\ 1 & 2 & 3 & 4 & 25 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 35 \\ 38 \\ 39 \\ 38 \\ 35 \end{bmatrix}$$

4.2 Programa paralelo proporcionado

- Al ejecutar el programa, aparecerán el contenido de la matriz *A* y el vector *b*, en distintos puntos del programa.
- Al final, el vector b contendrá la solución del sistema, que es un vector de unos. El sistema generado está preparado para que, independientemente del tamaño del sistema, la solución sea siempre un vector de unos.
- Finalmente, el programa informa del error de la solución obtenida, que debería ser cero.

```
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

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

Matrix LU:
---- proc. 0 ----
 25.000   4.000   3.000   2.000   1.000
  0.080   0.110   0.140  24.074   3.352
---- proc. 1 ----
  0.160  24.360   3.520   2.680   1.840
  0.040   0.076   0.108   0.139  24.071
---- proc. 2 ----
  0.120   0.144  24.131   3.373   2.614

Vector b after triInf:
---- proc. 0 ----
 35.000  32.400  30.118  27.426  24.071
---- proc. 1 ----
 35.000  32.400  30.118  27.426  24.071
---- proc. 2 ----
 35.000  32.400  30.118  27.426  24.071

Vector b after triSup (system solution):
---- proc. 0 ----
  1.000   1.000   1.000   1.000   1.000
---- proc. 1 ----
  1.000   1.000   1.000   1.000   1.000
---- proc. 2 ----
  1.000   1.000   1.000   1.000   1.000

Total accumulated error: 0.000000
```

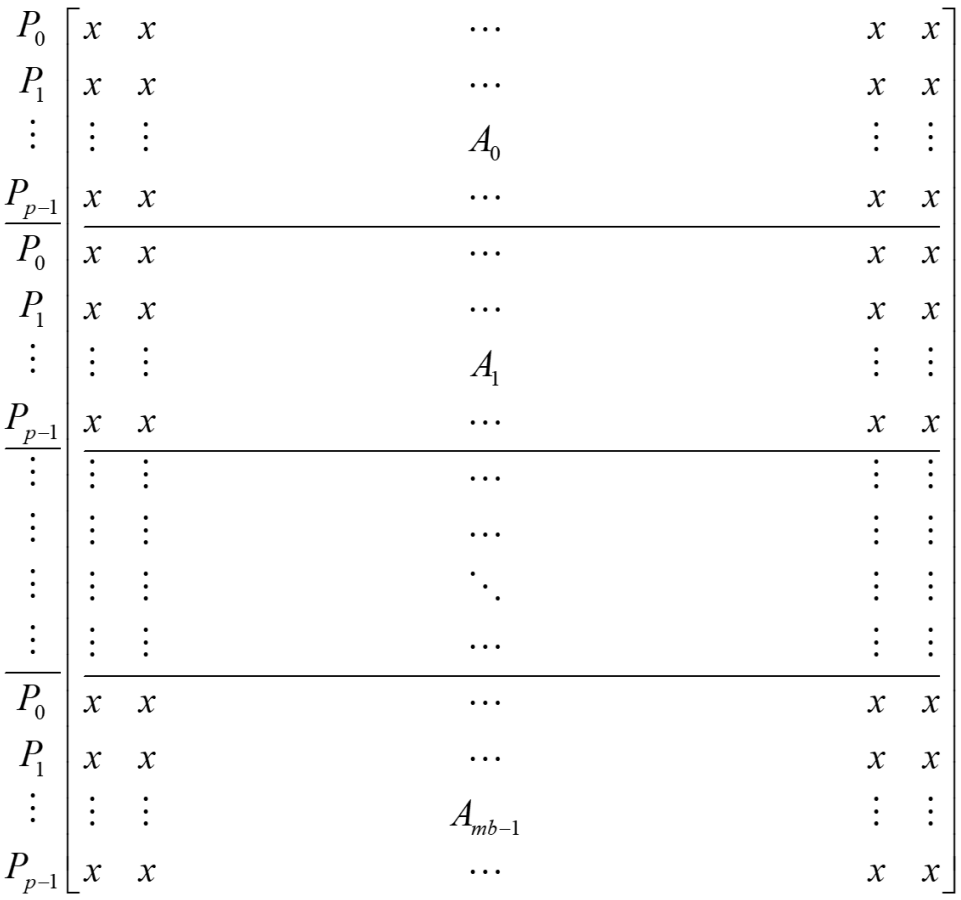
4.3 Fase de distribución de datos

- El objetivo de los siguientes ejercicios es modificar el código del programa para que **utilice una distribución cíclica por filas**, **en vez de una distribución por bloques de filas**.
- Esto requiere introducir cambios en el paso 2 (distribución de datos) y en los pasos 3, 4 y 5 (descomposición LU y resolución de sistemas de ecuaciones triangulares).
- 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 (MPI_Scatter, como a continuación se explica, y otra mediante tipos derivados(MPI_Type_vector)).

4.3 Fase de distribución de datos

- Cambia la forma de distribuir la matriz en la fase 2, para que se haga un reparto cíclico por filas mediante comunicaciones colectivas MPI_Scatter

```
344 |  MPI_Scatter(A[0], mb * n, MPI_DOUBLE,
345 |              Aloc[0], mb * n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```



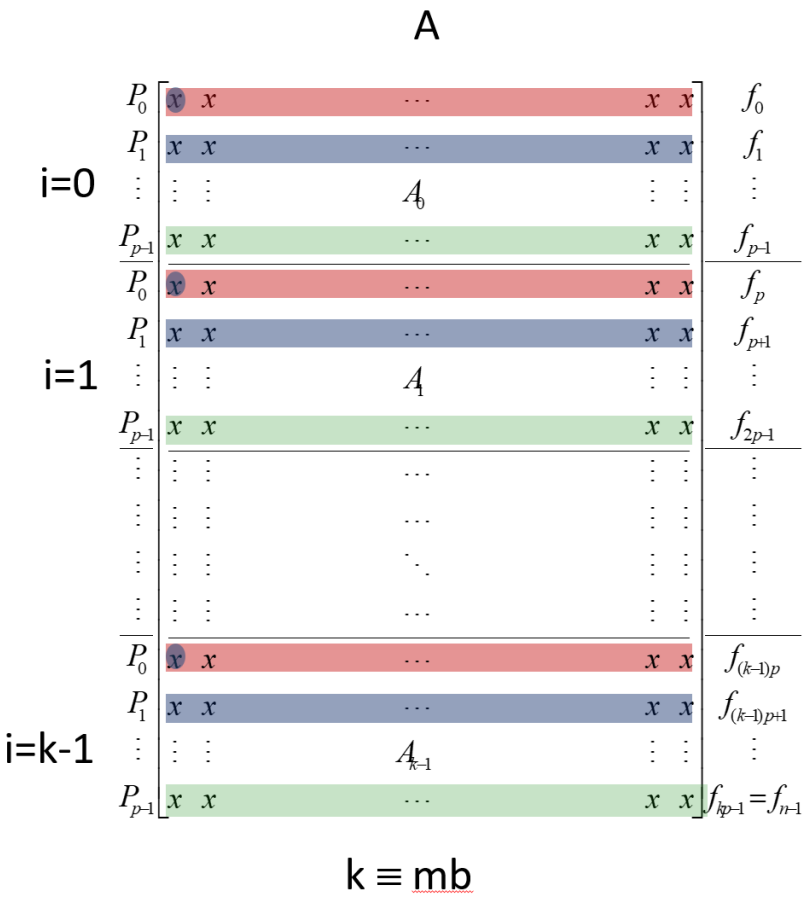
Cambiarlo por:

Para i=0 hasta mb-1
  Reparto del bloque A_i mediante MPI_Scatter
Fin para

Notas:

- **mb**=nº de filas de Aloc=n/p (n filas, p procesos)
- Si pruebas ahora con la modificación hecha, los resultados son incorrectos, pues se deben modificar las otras tres fases

4.3 Fase de distribución de datos



$$Al(P_0) = \begin{bmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{bmatrix} \quad \begin{aligned} \bar{f}_0 &= f_0 \\ \bar{f}_1 &= f_p \\ &\vdots \\ \bar{f}_{k-1} &= f_{(k-1)p} \end{aligned}$$

$$Al(P_1) = \begin{bmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{bmatrix} \quad \begin{aligned} \bar{f}_0 &= f_1 \\ \bar{f}_1 &= f_{p+1} \\ &\vdots \\ \bar{f}_{k-1} &= f_{(k-1)p+1} \end{aligned}$$

$$\vdots$$

$$Al(P_{p-1}) = \begin{bmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{bmatrix} \quad \begin{aligned} \bar{f}_0 &= f_{p-1} \\ \bar{f}_1 &= f_{2p-1} \\ &\vdots \\ \bar{f}_{k-1} &= f_{kp-1} \end{aligned}$$

Para i=0 hasta mb-1

Reparto del bloque A_i mediante MPI_Scatter

Fin para

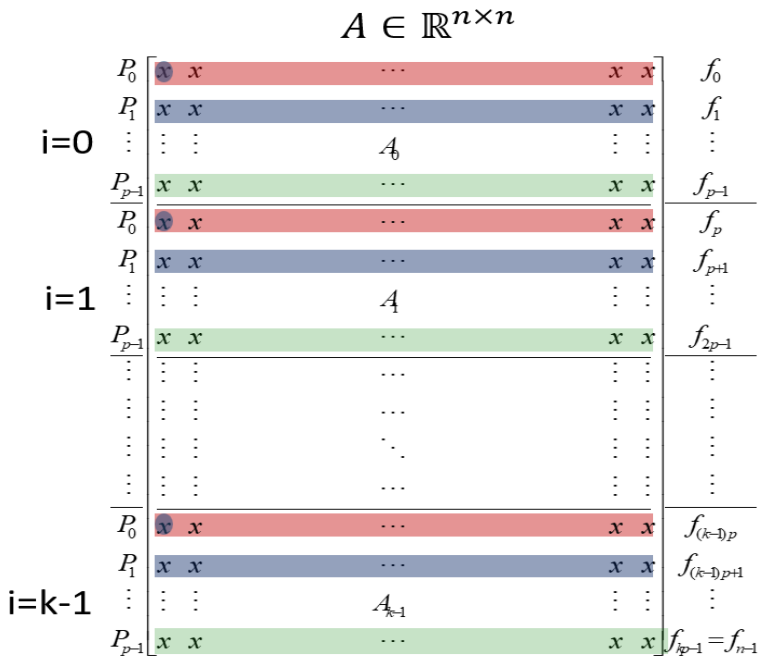
Para cada i tienes que identificar el:

- inicio del bloque que repartes (elementos con círculo de la matriz A)
- Inicio del lugar de Al en donde lo recibirás (elemento con cuadrado)

Para i=0, 1, 2, …, mb-1

Repartir A_i mediante MPI_Scatter

Fin para



$$Al(P_0) = \begin{bmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{bmatrix} \quad \begin{matrix} \bar{f}_0 = f_0 \\ \bar{f}_1 = fp \\ \vdots \\ \bar{f}_{k-1} = f_{(k-1)p} \end{matrix}$$

$$Al(P_1) = \begin{bmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{bmatrix} \quad \begin{matrix} \bar{f}_0 = f_1 \\ \bar{f}_1 = f_{p+1} \\ \vdots \\ \bar{f}_{k-1} = f_{(k-1)p+1} \end{matrix}$$

$$\vdots$$

$$Al(P_{p-1}) = \begin{bmatrix} x & x & \cdots & x \\ x & x & \cdots & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & x \end{bmatrix} \quad \begin{matrix} \bar{f}_0 = f_{p-1} \\ \bar{f}_1 = f_{2p-1} \\ \vdots \\ \bar{f}_{k-1} = f_{kp-1} \end{matrix}$$

i	Primer elemento de A	Primer elemento de Al
0	A[0][0]	Al[0][0]
1	A[p][0]	Al[1][0]
2	A[2*p][0]	Al[2][0]
…	…	…
k-1	A[(k-1)*p][0]	Al[k-1][0]

k ≡ mb

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

4.3 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
```

4.4. Fases de descomposición LU y sistemas triangulares

```
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
```

Figura 9: Algoritmo secuencial de descomposición LU.

```
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
```


Figura 11: Algoritmo paralelo de descomposición LU.

4.4. Fases de descomposición LU y 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
```

Figura 12: Algoritmos secuenciales de resolución de sistemas triangulares.

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
```

Figura 13: Algoritmos paralelos de resolución de sistemas triangulares.

4.4. Fases de descomposición LU y sistemas triangulares

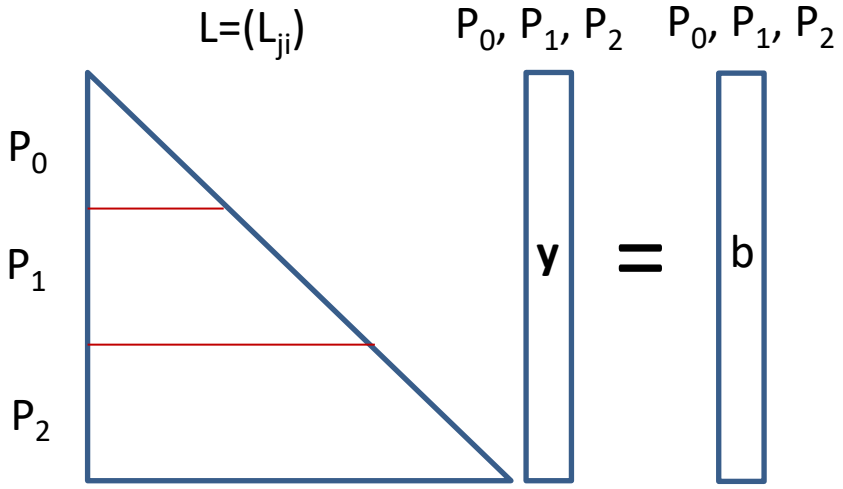
Código secuencial

```
Para i = 0, 1, ..., n-1
   Para j = i+1, ..., n-1
      bj = bj - Lji*bi
   Fin para
Fin para
```

Código paralelo

```
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
```

- La matriz L está distribuida por bloques de filas
- El vector solución (**y**) es calculado por P_k, quedando almacenado en b, debiendo ser conocido por todos los procesos



4.4. Fases de descomposición LU y sistemas triangulares

Reparto por bloques de un conjunto de índices

	k=12/3=4				k=12/3=4				k=12/3=4			
Indice global (i_g)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Indice proceso (i_p)	0	0	0	0	1	1	1	1	2	2	2	2
Indice local (i_l)	0	1	2	3	0	1	2	3	0	1	2	3

N= número de índices (12)
p= número de procesos (3)
k=N/p= nº de índices que le corresponden a cada proceso (4)

$$\begin{matrix} i_g & \lfloor k \\ i_l & i_p \end{matrix} \Leftrightarrow \begin{matrix} i_g = ki_p + i_l \\ i_p = i / k \\ i_l = i\%k \end{matrix}$$

```
Para i = 0, 1, ..., n-1
    difundir b(i) (El propietario de la fila i, difunde b(i) al resto)
    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
```

```c
/* Solves the unit lower triangular system L*x = b
 * Note: overwrites b with the solution x */
int triInf(double **L, double *b, int n, int iproc, int p, int mb)
{
  int i, j;

  if (L == NULL)
    return -2;

  if (n <= 0)
    return -3;

  for (i = 0; i < n; i++) {
    /* Broadcast b[i] */
    MPI_Bcast(&b[i], 1, MPI_DOUBLE, owner(i,p,mb), MPI_COMM_WORLD);
    /* Modify elements at rows i+1...n-1 */
    for (j = i + 1; j < n; j++) {
      if (owner(j,p,mb) == iproc)
        b[j] -= L[localIndex(j,p,mb)][i] * b[i];
    }
  }
  return 0;
}
```

- La implementación es independiente de la distribución utilizada
- Solo hay que cambiar las funciones **owner** (propietario) e **localIndex** para cambiar de distribución

Matriz L distribuida por bloques de filas:

```
234  /* Solves the unit lower triangular system L*x = b
235   * Note: overwrites b with the solution x */
236  int triInf(double **L, double *b, int n, int iproc, int p, int mb)
237  {
238    int i, j;
239
240    if (L == NULL)
241      return -2;
242
243    if (n <= 0)
244      return -3;
245
246    for (i = 0; i < n; i++) {
247      /* Broadcast b[i] */
248      MPI_Bcast(&b[i], 1, MPI_DOUBLE, owner(i,p,mb), MPI_COMM_WORLD);
249      /* Modify elements at rows i+1...n-1 */
250      for (j = i + 1; j < n; j++) {
251        if (owner(j,p,mb) == iproc)
252          b[j] -= L[localIndex(j,p,mb)][i] * b[i];
253      }
254    }
255    return 0;
256  }
```

```
166  /* Returns the process that owns row i */
167  int owner(int i, int p, int mb) {
168     /* Block distribution */
169     return i/mb;
170  }
```

$$i_p = i / k$$

$$i_l = i \% k$$

k ≡ mb

```
172  /* Returns the local index of row i in the local matrix of its owner process */
173  int localIndex(int i, int p, int mb) {
174     /* Block distribution */
175     return i%mb;
176  }
```

Reparto cíclico de un conjunto de índices

Indice global (i_g)	0	1	2	3	4	5	6	7	8	9	10	11
Indice proceso (i_p)	0	1	2	0	1	2	0	1	2	0	1	2
Indice local (i_l)	0	0	0	1	1	1	2	2	2	3	3	3

N= número de índices (12)

p= número de procesos (3)

k= N/p=nº de índices que le corresponden a cada proceso (4)

$$\begin{array}{l} i_g \;\; \big\lfloor p \\ i_p \;\; i_l \end{array} \Leftrightarrow \begin{array}{l} i_g = pi_l + i_p \\ i_p = i\%p \\ i_l = i \,/\, p \end{array}$$

Ejercicio 3

- Haz las modificaciones necesarias para que la función **trinf** funcione correctamente para una distribución cíclica de filas
  - Modifica las funciones **owner** y **localIndex** en el código proporcionado, respectivamente para que correspondan a una distribución cíclica por filas, en vez de a una distribución por bloques

```
/* Returns the process that owns row i */
int owner(int i, int p, int mb) {
   /* Block distribution */
   return i/mb;
}
/* Returns the local index of row i in the local matrix of its owner process */
int localIndex(int i, int p, int mb) {
   /* Block distribution */
   return i%mb;
}
```

$$i_p = i / k \qquad i_l = i\%k$$

$$\rightarrow \quad i_p = i\%p \qquad i_l = i / p$$

k ≡ mb

Ejercicio 3

- Haz las modificaciones necesarias para que la función **trinf** funcione correctamente para una distribución cíclica de filas
  - Modifica la función **numLocalRows**, la cual devuelve el número de filas locales de la matriz en un proceso (ese número puede ser distinto de **mb**, puesto que el número de filas puede no ser divisible entre el número de procesos), comentando y des-comentando su código:

```
int numLocalRows(int n, int mb, int p, int iproc) {
   int mloc;
   /* Comment out the code for the distribution that is not wanted */

   /* Block distribution */
   mloc = MIN(mb, n - mb * iproc);
   if (mloc < 0) mloc = 0;
   return mloc;

   /* Cyclic distribution */
   //mloc = n/p;
   //if (iproc < n%p) mloc++;
   //return mloc;
}
```

- Tras cambiar estas funciones, comprueba que los algoritmos de descomposición LU y resolución de sistemas triangulares funcionan correctamente.
- Al ejecutar el programa con 3 procesos y un sistema de 5 ecuaciones, el resultado de la descomposición LU y los sistemas triangulares debería dar:

```
Matrix LU:
---- proc. 0 ----
 25.000   4.000   3.000   2.000   1.000
  0.080   0.110   0.140  24.074   3.352
---- proc. 1 ----
  0.160  24.360   3.520   2.680   1.840
  0.040   0.076   0.108   0.139  24.071
---- proc. 2 ----
  0.120   0.144  24.131   3.373   2.614

Vector b after triInf:
---- proc. 0 ----
 35.000  32.400  30.118  27.426  24.071
---- proc. 1 ----
 35.000  32.400  30.118  27.426  24.071
---- proc. 2 ----
 35.000  32.400  30.118  27.426  24.071

Vector b after triSup (system solution):
---- proc. 0 ----
  1.000   1.000   1.000   1.000   1.000
---- proc. 1 ----
  1.000   1.000   1.000   1.000   1.000
---- proc. 2 ----
  1.000   1.000   1.000   1.000   1.000

Total accumulated error: 0.000000
```

Ejercicio 4

- Una vez realizada la versión cíclica, obtén resultados experimentales de las dos variantes, comparando las prestaciones de ambas, empleando el sistema de colas de Kahan.
- Analiza cuál de las dos versiones es más eficiente y trata de razonar a qué puede deberse.

Ejercicio 4

- Como debes realizar toma de tiempos, debes de añadir/modificar las siguientes líneas de código en las dos implementaciones:

Añadir

```
double t = MPI_Wtime();
/* STEP 2: Distribute data (A, b) */
```

Añadir

```
.....
t = MPI_Wtime() - t;
```

Mantener

```
if (iproc == 0) {
  dError = 0.0;
  for (i = 0; i < n; i++)
    dError += fabs(b[i] - 1.0);
```

Modificar

```
  printf("Tiempo=%f. Total accumulated error: %f\n", t, dError);
}
```

- Utiliza tamaños del sistema suficientemente grandes (entre 1000 y 2000)
- Es importante evitar que el programa muestre por pantalla las matrices y vectores, ya que eso hará que tarde mucho más. Para ello, basta con comentar la línea: #define VERBOSE en los dos códigos:

```
11    //#define VERBOSE
```

Ejercicio 4: script y resultados

```
#!/bin/sh
#SBATCH --nodes=2
#SBATCH --ntasks=4
#SBATCH --time=10:00
#SBATCH --partition=cpa
#SBATCH --output=s4.txt
scontrol show hostnames $SLURM_JOB_NODELIST
echo "sisbf: 1000 "
mpiexec sistbf 1000
echo "siscf: 1000 "
mpiexec sistcf 1000
echo "sisbf: 2000 "
mpiexec sistbf 2000
 echo "siscf: 2000 "
mpiexec sistcf 2000
```

```
sisbf: 1000 T=1.685481. Total accumulated error: 0.000000
siscf: 1000 T=1.383355. Total accumulated error: 0.000000
sisbf: 2000 T=13.419227. Total accumulated error: 0.000000
siscf: 2000 T=10.764285. Total accumulated error: 0.000000
```

Nota:

- Observa que en la resolución del sistema Ax=b hay que resolver dos sistemas triangulares ¿Esto puede tener algo que ver con los tiempos obtenidos?