FACTORIZACIÓN LU

UNA APLICACIÓN CON MPI

Descomposición Lu

$$A = \begin{pmatrix} 1 & 2 & 4 \\ 3 & 8 & 14 \\ 2 & 6 & 13 \end{pmatrix}$$

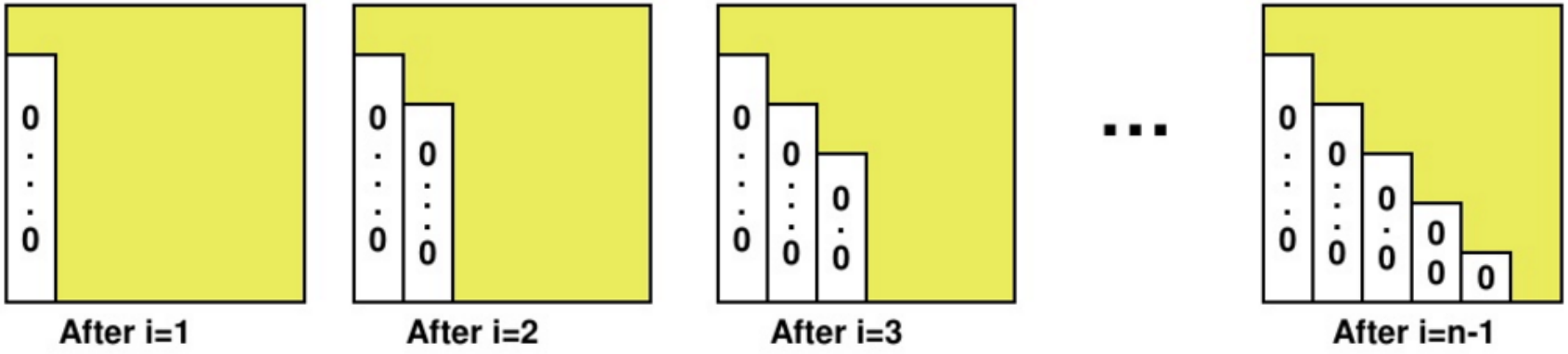
$$U = \begin{pmatrix} 1 & 2 & 4 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 2 & 1 & 1 \end{pmatrix}$$

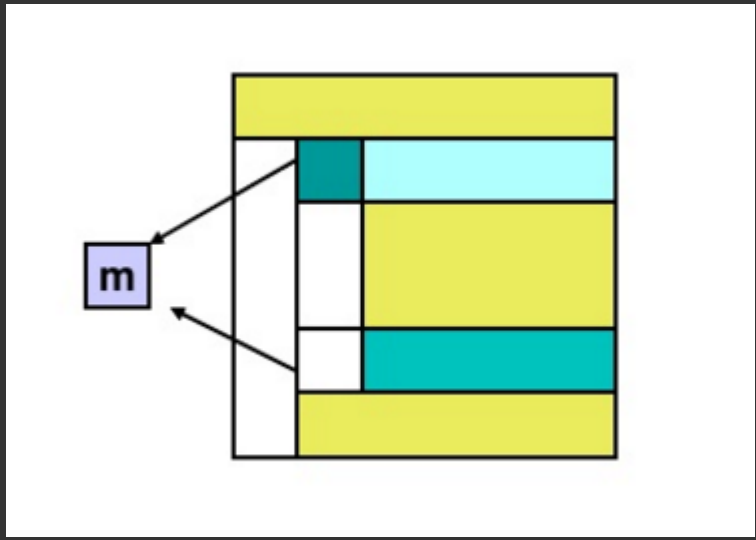
Suponemos matrices cuadradas con entradas en la diagonal no cero

Enfoque secuencial

Matriz U:



Matriz L:



Código

```
void factoriza_lu(double** A, long n)
{
    long i,j,k;
    for(k=0;k<n;k++)
    {

        for(j=k+1;j<n;j++)
        {
            A[k][j]=A[k][j]/A[k][k]; /*Factores de multiplicación para la matriz L*/
        }
        for(i=k+1;i<n;i++)
        {
            for(j=k+1;j<n;j++) /*j son las columnas*/
            {
                A[i][j]=A[i][j] - A[i][k] * A[k][j]; /*Le hace la operación al renglón correspondiente*/
            }
        }
    }
}
```

Salida

```
_Metodos_Numericos_y_Optimizacion/Proyecto_final/Docker/MPI$ ./lu_secuencial.out 4

 *************** MATRIZ INICIAL****************
2.000000 2.000000 3.000000 4.000000
5.000000 6.000000 7.000000 8.000000
9.000000 10.000000 1.000000 2.000000
3.000000 4.000000 5.000000 8.000000

 *************** MATRIZ L:*****************
1.000000 0.000000 0.000000 0.000000
2.500000 1.000000 0.000000 0.000000
4.500000 1.000000 1.000000 0.000000
1.500000 1.000000 -0.083333 1.000000

 *************** MATRIZ U:*****************
2.000000 2.000000 3.000000 4.000000
0.000000 1.000000 -0.500000 -2.000000
0.000000 0.000000 -12.000000 -14.000000
0.000000 0.000000 0.000000 2.833333

*************************************************
Tamaño de la matriz :4
Tiempo usado en la factorización LU : 0.000001 seconds
*************************************************
```

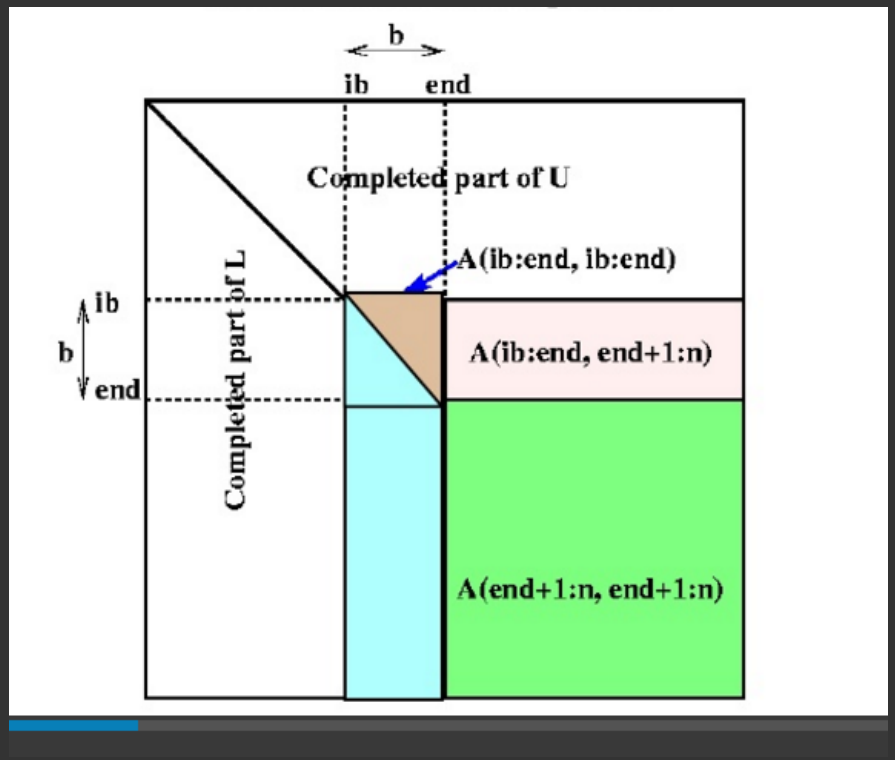
Enfoque Paralelo

$$A = \left(\begin{array}{c|c} a_{11} & a_{12} \ldots a_{1n} \\ \hline a_{21} & a_{22} \ldots a_{2n} \\ a_{n1} & a_{n2} \ldots a_{nn} \end{array} \right) = \begin{pmatrix} a_{11} & w^T \\ v & A' \end{pmatrix}$$

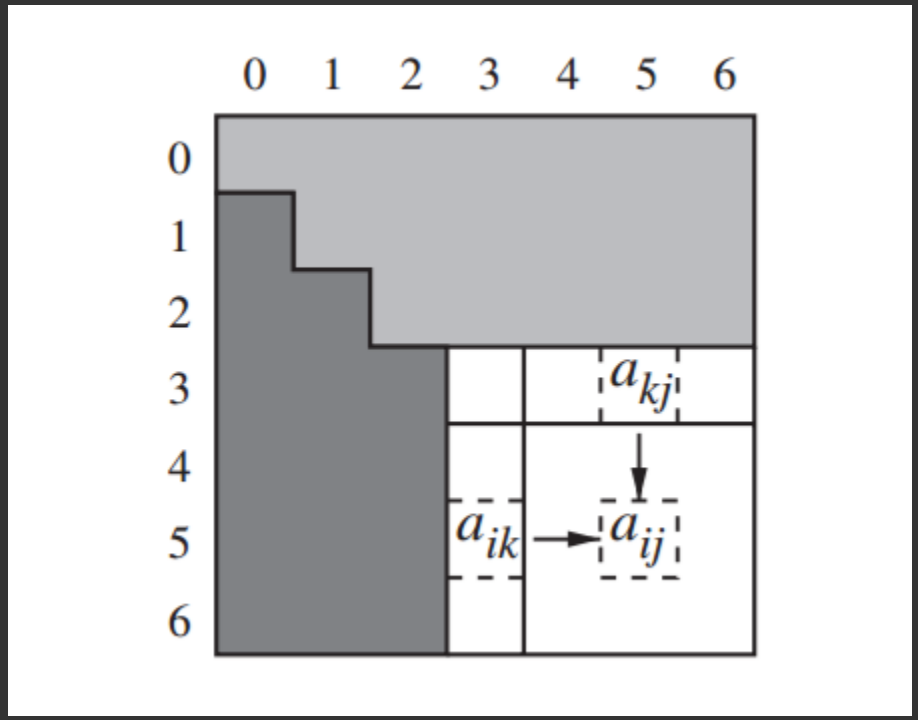
$$= \begin{pmatrix} 1 & 0 \\ \frac{v}{a_{11}} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & A' - \frac{vw^T}{a_{11}} \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 0 \\ \frac{v}{a_{11}} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & L'U' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{v}{a_{11}} & L' \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & U' \end{pmatrix} = LU$$

Enfoque Paralelo

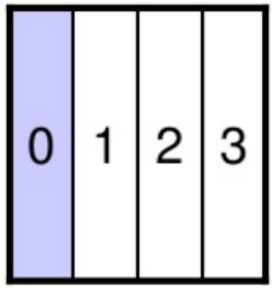


Se van actualizando los factores del renglón

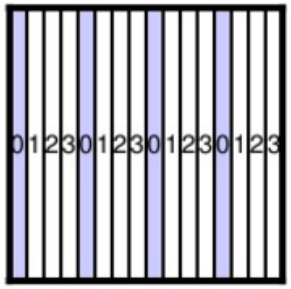


Formas de paralelizar

Mal balance de carga:
P0 ocioso luego de
n/4 pasos



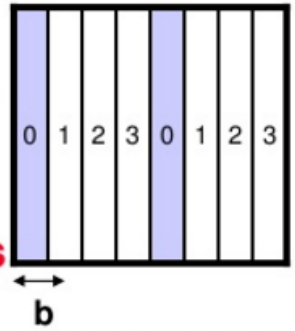
1) Bloque de columnas 1D



Mejor balance, pero
difícil usar BLAS2 o
BLAS3

2) Columnas 1D cíclicas

Se puede negociar
balance de carga y
performance BLAS2/3
cambiando b, pero la
factorización del
bloque de columnas es
el cuello de botella



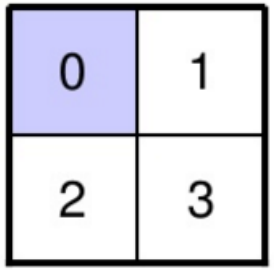
3) Bloques cíclicos de columnas 1D

0	1	2	3
3	0	1	2
2	3	0	1
1	2	3	0

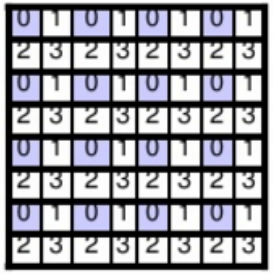
Difícil de
direccionar

4) Bloques de diagonales

Mal balance de carga:
P0 ocioso luego de
n/2 pasos



5) Bloques de filas y columnas 2D



El mejor!

6) Bloques cíclicos de filas
y columnas 2D

```
for(k=0;k<n;k++)
{
    if(k>=mymin && k<=mymax)
    {
        for(j=k+1;j<n;j++)
        {
            A_local[k][j]=A_local[k][j] / A_local[k][k];
        }

        if(nprocs>1) /*Si el número de procesadores es 1, entonces no hay nada que enviar*/
        {
            for(p=pid;p<nprocs;p++) /*El último procesador no envía mensaje sólo lo recibe*/
            {
                MPI_Send(&A_local[k][0], n , MPI_DOUBLE, p, data_tag, MPI_COMM_WORLD);
            }
        }
    }
 if(nprocs>1)
 {
     if(k<=mymax)
     {
         MPI_Recv(&rowk, n, MPI_DOUBLE, MPI_ANY_SOURCE, data_tag, MPI_COMM_WORLD, &status);

         if(pid==nprocs-1)
             {reemplaza(pid, rowk,k,n);}
     }
 }

 for(i=(((k+1) > mymin) ? (k+1) : mymin);i<=mymax;i++)
 {
     for(j=k+1;j<n;j++)
     {
         if(nprocs>1)
         {
             A_local[i][j] = A_local[i][j] - A_local[i][k] * rowk[j];
         }
         else
         {
             A_local[i][j] = A_local[i][j] - A_local[i][k] * A_local[k][j];
         }
     }
 }
```

Distribuido con un procesador

```
mpi_user@master:~$ mpiexec -n 2 lu_mpi.out 4

 *************** MATRIZ INICIAL***************
PID0: 2.000000 PID0: 2.000000 PID0: 3.000000 PID0: 4.000000
PID0: 5.000000 PID0: 6.000000 PID0: 7.000000 PID0: 8.000000
PID0: 9.000000 PID0: 10.000000 PID0: 1.000000 PID0: 2.000000
PID0: 3.000000 PID0: 4.000000 PID0: 5.000000 PID0: 8.000000

*****************************************************
Tamaño de la matriz :4
Tiempo usado en la factorización LU : 0.019049 seconds
*****************************************************

 *************** MATRIX L:****************
1.000000 0.000000 0.000000 0.000000
2.500000 1.000000 0.000000 0.000000
4.500000 1.000000 1.000000 0.000000
1.500000 1.000000 -0.083333 1.000000

 *************** MATRIX U:****************
2.000000 2.000000 3.000000 4.000000
0.000000 1.000000 -0.500000 -2.000000
0.000000 0.000000 -12.000000 -14.000000
0.000000 0.000000 0.000000 2.833333

*****************************************************
Tamaño de la matriz :4
Tiempo usado en la factorización LU : 0.018125 seconds
*****************************************************
```

Pseudo-distribuido

```
mpi_user@master:~$ mpirun --prefix /opt/openmpi-2.1.0/ -n 2 -H master,nodo1 lu_mpi.out
4

 *************** MATRIZ INICIAL***************
PID0: 2.000000 PID0: 2.000000 PID0: 3.000000 PID0: 4.000000
PID0: 5.000000 PID0: 6.000000 PID0: 7.000000 PID0: 8.000000
PID0: 9.000000 PID0: 10.000000 PID0: 1.000000 PID0: 2.000000
PID0: 3.000000 PID0: 4.000000 PID0: 5.000000 PID0: 8.000000

*****************************************************
Tamaño de la matriz :4
Tiempo usado en la factorización LU : 0.023544 seconds
*****************************************************

 *************** MATRIX L:****************
1.000000 0.000000 0.000000 0.000000
2.500000 1.000000 0.000000 0.000000
4.500000 1.000000 1.000000 0.000000
1.500000 1.000000 -0.083333 1.000000

 *************** MATRIX U:****************
2.000000 2.000000 3.000000 4.000000
0.000000 1.000000 -0.500000 -2.000000
0.000000 0.000000 -12.000000 -14.000000
0.000000 0.000000 0.000000 2.833333

*****************************************************
Tamaño de la matriz :4
Tiempo usado en la factorización LU : 0.023040 seconds
*****************************************************
```