UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE CIENCIAS
ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACION

Apellidos/Nombres: Escobedo Vasquez, Janice Katherine Nota: …..

PROGRAMACIÓN PARALELA

Practica Calificada Nº 2

(13/10/2023)

El tratamiento de un problema enfocado desde una perspectiva computacional adopta diferente complejidad y tendencias de comportamiento a medida que se incrementa el número n de variables y su correspondiente dimensión.

Las siguientes **áreas** contienen **problemas** los cuales pueden ser resueltos a través de múltiples criterios con diferentes técnicas de resolución.

Dado el siguiente problema Factorización en LU

Implementar el método de resolución Algoritmo de Crout

Para tal efecto, se debe realizar:

1. **Analizar la técnica de resolución**

   El algoritmo de Crout en Julia se escribe como

```
using LinearAlgebra
function LU(A::Matrix{Float64})
    m,n = size(A)
    U = Matrix{Float64}(I, n, n)
    L = copy(A)
    for i = 1:n
        for j = i+1:n
            U[i,j] = L[i,j]/L[i,i]
            L[i:end,j] = L[i:end,j] - U[i,j]*L[i:end,i]
        end
    end
    return L,U
end
```

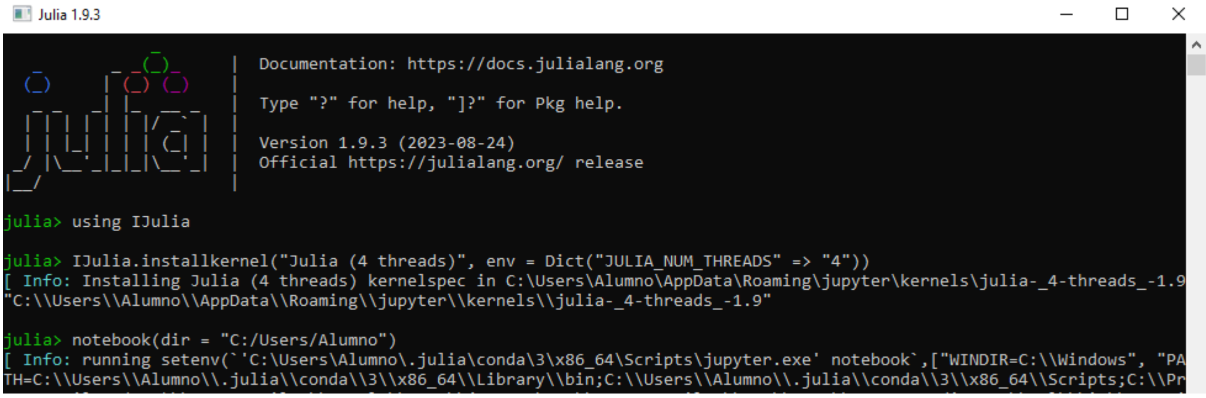
Y el algoritmo de Crout por bloques, como:

```
using LinearAlgebra

function is_singular(A)
    return det(A) == 0
end

function block_LU(A)
    n = size(A, 1) # Numero de filas, debe ser par
    if n == 1
```

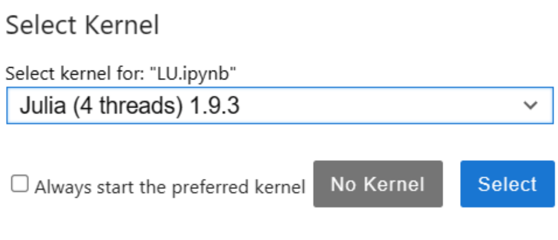
```
        return A, A, A
    end
    # Dividimos a la matriz en bloques de (n/2)*(n/2) para factorzarlo como sigue
    #     [ A_11 | A_12 ]    [ L_11 |  0  ] [ U_11 | U_12 ]
    # A = | ----------- | =  ¡ ---------- ] [ ----------- ]
    #     [ A_21 | A_22 ]    [ L_21 | L22 ] [  0   | U_22 ]

    m = n ÷ 2
    A11 = A[1:m, 1:m]
    A12 = A[1:m, m+1:end]
    A21 = A[m+1:end, 1:m]
    A22 = A[m+1:end, m+1:end]

    if is_singular(A11) || is_singular(A22)
        error("La matriz es singular")
    end

    # Efectuar el algoritmo de Crout como si tuvieramos una matriz 2x2
    L11, U11 = impLU(A11)
    U12 = L11 \ A12
    L21 = A21 / U11
    L22, U22 = impLU(A22 - L21*U12)

    # Formacion de matrices L, U por bloques tambien
    L = [L11 zeros(m, n-m);
         L21 L22]
    U = [U11 U12;
         zeros(n-m, m) U22]

    return L, U
end

# Algoritmo Crout
function impLU(A::Matrix{Float64})
    m,n = size(A)
    U = Matrix{Float64}(I, n, n)
    L = copy(A)
    for i = 1:n-1
        for j = i+1:n
            U[i,j] = L[i,j]/L[i,i]
            L[i:end,j] = L[i:end,j] - U[i,j]*L[i:end,i]
        end
    end
    return L,U
end
```

2. **Identificar bloques funcionales autónomos/independientes para la descomposición/particionamiento de la tarea en unidades o subtareas elementales.**

   Para poder implementar los hilos en Julia (con Jupiter) debemos crear un kernel con 4 hilos en este caso



En el notebook, seleccionamos el kernel con 4 hilos que creamos



La versión en paralelo del algoritmo de Crout:

```
using LinearAlgebra
using Base.Threads

function parallel_LU(A::Matrix{Float64})
    m,n = size(A)
    U = Matrix{Float64}(I, n, n)
    L = copy(A)
    num_threads = Threads.nthreads() # Obtenemos el numero de hilos disponibles en el proceso
    threads = []
    for i = 1:n-1
        for j = i+1:n
            if length(threads) < num_threads
                # Creamos una tarea en el hilo actual, en este caso  la tarea
                # es hacer: L[i,j] = U[i,j]/U[j,j] y U[i,j:end] -= L[i,j]*U[j,j:end]
                push!(threads, Threads.@spawn begin
                    U[i,j] = L[i,j]/L[i,i]
                    L[i:end,j] = L[i:end,j] - U[i,j]*L[i:end,i]
                end)
            else # Cuando ya tengamos en ejecucion los 4 hilos
                wait(threads[1]) # Esperamos a que la tarea 1 iniciada acabe
                # La tarea 1 acabo y asi lo podemos eliminar
                popfirst!(threads) # Elimina el primer elemento de la lista
                                   # threads; es decir, la primera tarea
            end
        end
    end
    for thread in threads # Recorre las tareas, una en cada hilo
        # Esperamos a que acaben de ejecutarse todos los hilos para finalizar la ejecucion
        wait(thread)
    end
    return L,U
end
```

Mientras que la versión por bloques en Paralelo es:

```julia
using Base.Threads
using LinearAlgebra

function is_singular(A)
    return det(A) == 0
end

function pseudo_parallel_LU_block_LU(A)
    n = size(A, 1)
    if n == 1
        return A, A, A
    end

    m = n ÷ 2
    A11 = A[1:m, 1:m]
    A12 = A[1:m, m+1:end]
    A21 = A[m+1:end, 1:m]
    A22 = A[m+1:end, m+1:end]

    if is_singular(A11) || is_singular(A22)
        error("La matriz es singular")
    end

    L11, U11 = parallel_LU(A11)
    U12 = L11 \ A12
    L21 = A21 / U11
    L22, U22 = parallel_LU(A22 - L21*U12)

    L = [L11 zeros(m, n-m);
         L21 L22]
    U = [U11 U12;
         zeros(n-m, m) U22]

    return L, U
end

function parallel_LU(A::Matrix{Float64})
    m,n = size(A)
    U = Matrix{Float64}(I, n, n)
    L = copy(A)
    num_threads = Threads.nthreads()
    threads = []
    for i = 1:n-1
        for j = i+1:n
            if length(threads) < num_threads
                push!(threads, Threads.@spawn begin
                    U[i,j] = L[i,j]/L[i,i]
                    L[i:end,j] = L[i:end,j] - U[i,j]*L[i:end,i]
                end)
            else
                wait(threads[1])
                popfirst!(threads)
            end
        end
    end
    for thread in threads
        wait(thread)
    end
    return L,U
end
```

3. **Implementar la solución en las versiones normal y paralela generando la salida respectiva.**

Consideramos la matriz

```
A1 = rand(1000,1000)
```

```
1000×1000 Matrix{Float64}:
 0.531245   0.840443   0.324613   …  0.231392   0.429289   0.740698
 0.199012   0.804679   0.194388      0.781432   0.406935   0.478237
 0.364188   0.414361   0.768961      0.517587   0.801675   0.377513
 0.994625   0.176044   0.502354      0.187411   0.764038   0.115481
 0.314001   0.511661   0.131091      0.176075   0.869293   0.750911
 0.978091   0.27864    0.388003   …  0.192211   0.992112   0.59318
 0.406506   0.248327   0.889899      0.133758   0.659295   0.318746
 0.585159   0.460981   0.0227243     0.508858   0.139703   0.856802
 0.104612   0.0131676  0.543797      0.314539   0.558658   0.961856
 0.604983   0.883581   0.937919      0.0600933  0.981399   0.148655
 0.342449   0.212408   0.985459   …  0.646976   0.233151   0.965976
 0.799217   0.117264   0.0777435     0.417677   0.399047   0.401566
 0.402942   0.234303   0.338502      0.859909   0.0388994  0.964968
 ⋮                                ⋱
 0.628572   0.322654   0.773479      0.141251   0.700553   0.50894
 0.606729   0.500559   0.562075      0.612121   0.0928506  0.256479
 0.0400324  0.880954   0.757279   …  0.80527    0.772681   0.550398
 0.920523   0.761366   0.151254      0.112968   0.420793   0.710674
 0.354766   0.365657   0.717905      0.0334473  0.916623   0.844074
 0.33993    0.320912   0.205217      0.814119   0.120004   0.708768
 0.926979   0.897713   0.224411      0.726363   0.935092   0.786041
 0.36965    0.932813   0.333833   …  0.195266   0.126649   0.44792
 0.237553   0.25068    0.355568      0.0888052  0.960663   0.483127
 0.325837   0.946895   0.276072      0.835243   0.334661   0.259688
 0.688088   0.744081   0.67029       0.839286   0.331659   0.760077
 0.783704   0.285326   0.0160924     0.483126   0.214749   0.148425
```

Generamos las pruebas y lo guardamos en archivos .txt adjuntados en el .zip

Tiempo de ejecución para el algoritmo de Crout serial:

```
L,U = LU(A1)
open("LUserial.txt","w") do io
    println(io,"A =", A1)
    println(io,"L=",L)
    println(io,"U=",U)
end
println("Tiempo de Ejecucion: ",@time LU(A1))
```

```
  4.588439 seconds (2.00 M allocations: 10.235 GiB, 12.88% gc time)
Tiempo de Ejecucion: ([0.5312445262029812 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -5.551115123125783e-17 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 -1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.1102230246251565e-16 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 5.55111512312
5783e-17 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -5.551115123125783e-17 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16 0.0 0.0 -2.77555756
15628914e-17 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 2.7755575615628914e-17 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 5.551115123125783e-17 0.0 0.0 5.551115123125783e-17 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16 1.387778780
```

Tiempo de ejecución para el algoritmo de Crout paralelo:

```
# misma matrizA1 de orden 1000
L,U = parallel_LU(A1)
open("LUparalelo.txt","w") do io
    println(io,"L=",L)
    println(io,"U=",U)
end
println("Tiempo de Ejecucion: ",@time parallel_LU(A1))
```

```
  2.862505 seconds (2.45 M allocations: 5.276 GiB, 17.43% gc time)
Tiempo de Ejecucion: ([0.5312445262029812 0.0 0.0 0.0 0.0 0.7888655029358267 0.0 0.7420950211252872 0.0 0.8640249894556998 0.0 0.5293833145840889 0.0
0.0027319336560808205 0.0 0.35818475868259936 0.0 0.3513952234484925 0.0 0.44455090604345504 0.0 0.33420786885252796 0.0 0.7932976786256317 -1.11022302
46251565e-16 0.921368110112549 0.0 0.7308288908320628 0.0 0.06585363328085436 0.0 0.48669411338991153 0.0 0.50923573343181 0.0 0.5974256015182012 0.0
0.9798505840846334 0.0 0.24770041343597615 0.0 0.3320463326055645 0.0 0.14638468062779897 0.0 0.8556114695798496 0.0 0.5185352297064301 1.1102230246251
565e-16 0.29644657814538944 0.0 0.18085409934746188 0.0 0.07653265003702425 0.0 0.9930716002897829 0.0 0.2292287004372452 0.0 0.18280747195816105 0.0
0.13157624023512127 0.0 0.5727067634948028 0.0 0.3329245171333801 0.0 0.46767626287366193 0.0 0.1910285891286888 0.0 0.48190840036679483 0.0 0.26741223
888310905 0.0 0.0966637122300481 0.0 0.1608484212221537 0.0 0.9204941848690853 0.0 0.3136334815758647 0.0 0.9883521490118594 0.0 0.2588313130220552 5.5
51115123125783e-17 0.8452908208309209 0.0 0.19225674831391593 0.0 0.116999924090891 0.0 0.8236110755523258 0.0 0.06685912408584171 0.0 0.67599542508638
11 0.0 0.7022214955389813 0.0 0.7834047541549856 0.0 0.7306104150208492 0.0 0.36475845840586196 0.0 0.03892838785917996 0.0 0.7205277267416165 0.0 0.76
04466508690233 0.0 0.764732144404898 0.0 0.21087131279432203 0.0 0.1647829894804328 0.0 0.8344332086513194 0.0 0.06374034632907344 0.0 0.08516311205598
06 0.0 0.6073680220301143 0.0 0.4122710103428099 0.0 0.6907043461709651 0.0 0.13350290021452127 0.0 0.8758368243997332 0.0 0.36937251644519986 0.0 0.20
268977830848012 0.0 0.761114724556432 0.0 0.05530615061857136 0.0 0.7933629492752605 0.0 0.5190248056469964 0.0 0.25549613857014297 0.0 0.5984601188714
145 0.0 0.13009271813858958 0.0 0.8516517744045822 0.0 0.3549457088046424 0.0 0.04371604541191254 0.0 0.9878868518771304 0.0 0.14852524781801835 0.0 0.
9774124642805415 0.0 0.08455605293105206 0.0 0.07279871983344932 0.0 0.5015685097387642 0.0 0.2918713936794789 0.0 0.8348197839610783 0.0 0.35771840213
557116 0.0 0.6643145231494976 0.0 0.6893424881385837 0.0 0.5548841125667096 0.0 0.9811577466369241 0.0 0.6248013179539404 0.0 0.6225604804630108 0.0 0.
8818895983874276 0.0 0.3749605715618475 0.0 0.44392300457438505 0.0 0.22438359795177887 0.0 0.6381443894533744 0.0 0.2822375153003118 0.0 0.43063271587
09303 0.0 0.9573138857586987 -1.1102230246251565e-16 0.03921758891397109 0.0 0.9832481104903099 0.0 0.7713739928909645 0.0 0.6733017194269695 0.0 0.842
```

Tiempo de ejecución para el algoritmo de Crout por bloques serial:

```julia
# misma matriz
L,U = block_LU(A1)
open("LUbloqueserial.txt","w") do io
    println(io,"L=",L)
    println(io,"U=",U)
end
println("Tiempo de Ejecucion: ",@time block_LU(A1))
```

```
  0.961884 seconds (998.04 k allocations: 2.668 GiB, 18.27% gc time)
Tiempo de Ejecucion: ([0.5312445262029812 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -5.551115123125783e-17 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 -1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.1102230246251565e-16 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 5.55111512312
5783e-17 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -5.551115123125783e-17 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16 0.0 0.0 -2.77555756
15628914e-17 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 1.1102230246251565e-16 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 2.7755575615628914e-17 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Tiempo de ejecución para el algoritmo de Crout por bloques paralelo:

```julia
# misma matriz
L,U = pseudo_parallel_LU_block_LU(A1)
open("LUbloqueparalelo.txt","w") do io
    println(io,"A =", A1)
    println(io,"L=",L)
    println(io,"U=",U)
end
println("Tiempo de Ejecucion: ",@time pseudo_parallel_LU_block_LU(A1))
```

```
  0.784273 seconds (1.22 M allocations: 1.434 GiB, 17.37% gc time)
Tiempo de Ejecucion: ([0.5312445262029812 0.0 0.0 0.0 0.0 0.7888655029358267 0.0 0.7420950211252872 0.0 0.8640249894556998 0.0 0.5293833145840889 0.0
0.0027319336560808205 0.0 0.35818475868259936 0.0 0.3513952234484925 0.0 0.44455090604345504 0.0 0.33420786885252796 0.0 0.7932976786256317 -1.11022302
46251565e-16 0.921368110112549 0.0 0.7308288908320628 0.0 0.06585363328085436 0.0 0.48669411338991153 0.0 0.50923573343181 0.0 0.5974256015182012 0.0
0.9798505840846334 0.0 0.24770041343597615 0.0 0.3320463326055645 0.0 0.14638468062779897 0.0 0.8556114695798496 0.0 0.5185352297064301 1.1102230246251
565e-16 0.29644657814538944 0.0 0.18085409934746188 0.0 0.07653265003702425 0.0 0.9930716002897829 0.0 0.2292287004372452 0.0 0.18280747195816105 0.0
0.13157624023512127 0.0 0.5727067634948028 0.0 0.3329245171333801 0.0 0.46767626287366193 0.0 0.1910285891286888 0.0 0.48190840036679483 0.0 0.26741223
888310905 0.0 0.0966637122300481 0.0 0.1608484212221537 0.0 0.9204941848690853 0.0 0.3136334815758647 0.0 0.9883521490118594 0.0 0.2588313130220552 5.5
51115123125783e-17 0.8452908208309209 0.0 0.19225674831391593 0.0 0.116999924090891 0.0 0.8236110755523258 0.0 0.06685912408584171 0.0 0.67599542508638
11 0.0 0.7022214955389813 0.0 0.7834047541549856 0.0 0.7306104150208492 0.0 0.36475845840586196 0.0 0.03892838785917996 0.0 0.7205277267416165 0.0 0.76
04466508690233 0.0 0.764732144404898 0.0 0.21087131279432203 0.0 0.1647829894804328 0.0 0.8344332086513194 0.0 0.06374034632907344 0.0 0.08516311205598
06 0.0 0.6073680220301143 0.0 0.4122710103428099 0.0 0.6907043461709651 0.0 0.13350290021452127 0.0 0.8758368243997332 0.0 0.36937251644519986 0.0 0.20
268977830848012 0.0 0.761114724556432 0.0 0.05530615061857136 0.0 0.7933629492752605 0.0 0.5190248056469964 0.0 0.25549613857014297 0.0 0.5984601188714
145 0.0 0.13009271813858958 0.0 0.8516517744045822 0.0 0.3549457088046424 0.0 0.04371604541191254 0.0 0.9878868518771304 0.0 0.14852524781801835 0.0 0.
9774124642805415 0.0 0.08455605293105206 0.0 0.07279871983344932 0.0 0.5015685097387642 0.0 0.2918713936794789 0.0 0.8348197839610783 0.0 0.35771840213
557116 0.0 0.6643145231494976 0.0 0.6893424881385837 0.0 0.5548841125667096 0.0 0.9811577466369241 0.0 0.6248013179539404 0.0 0.6225604804630108 0.0 0.
8818895983874276 0.0 0.3749605715618475 0.0 0.44392300457438505 0.0 0.22438359795177887 0.0 0.6381443894533744 0.0 0.2822375153003118 0.0 0.43063271587
09303 0.0 0.9573138857586987 -1.1102230246251565e-16 0.03921758891397109 0.0 0.9832481104903099 0.0 0.7713739928909645 0.0 0.6733017194269695 0.0 0.842
```