Name: Andrew Tee
Github Account Name: atee001
Github Repo for HW2: https://github.com/CS211-Fall2023/hw2-atee001/tree/main

HW# 2

PROBLEM 1

Image shows all intermediate calculations regarding the non-blocked LU factorization A = LU.

$$A = \begin{array}{c|c} & \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 1 & 2 & 3 \\ 4 & 13 & 18 \\ 7 & 54 & 78 \end{bmatrix} \end{array}$$

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
    for j = i+1 to n
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

$i = 1$

$j = i+1$

$j=1 \Rightarrow A(2,1) = A(2,1)/A(1,1) = 4$

$j=2 = A(3,1) = A(3,1)/A(1,1) = 7$

$j = i+1 = 2$

$k = i+1 = 2$

$j=2, k=2 \Rightarrow A(2,2) = A(2,2) - 4A(1,2) = 13 - 4(2) = 5$

$j=2, k=3 \Rightarrow A(2,3) = A(2,3) - 4A(1,3) = 18 - 4(3) = 6$

$j = 3$

$j=3, k=2 \Rightarrow A(3,2) = A(3,2) - A(3,1) * A(1,2) = 54 - 7(2) = 40$

$j=3, k=3 \Rightarrow A(3,3) = A(3,3) - 7A(1,3) = 78 - 7(3) = 57$

$$A' = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 40 & 57 \end{bmatrix}$$

$i = 2$

$j = 3$

$A(3,2) = A(3,2)/A(2,2) = \frac{40}{5} = 8$

$j = 3$

$k = 3$

$A(3,3) = A(3,3) - 8A(2,3) = 57 - 8(6) = 9$

$$A'' = \boxed{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 8 & 1 \end{bmatrix}}_{L} \times \underbrace{\begin{bmatrix} 1 & 2 & 3 \\ 0 & 5 & 6 \\ 0 & 0 & 9 \end{bmatrix}}_{U}}$$

PROBLEM 2

Output for N = 1000, 2000, 3000, 4000

```
output_6684-cluster-001-compute-001.txt
rm -f main
gcc main.c -o main -I /act/opt/intel/composer_xe_2013.3.163/mkl/include \
-L /act/opt/intel/composer_xe_2013.3.163/mkl/lib/intel64 \
-O3 -DMKL_ILP64 -lmkl_avx2 -lmkl_intel_lp64 -lmkl_sequential -lmkl_core -lpthread -lm -m64
n=1000, pad=1
time=0.059905s
n=1000, pad=1
time=0.157215s
n=2000, pad=1
time=0.206416s
n=2000, pad=1
time=1.684254s
n=3000, pad=1
time=0.660857s
n=3000, pad=1
time=6.487672s
n=4000, pad=1
time=1.328586s
n=4000, pad=1
time=16.376956s
```

TA said to assume the same flops for LAPACK and my GEPP algorithms.

TOTAL FLOP COUNT = FACTORIZATION + L + U

1. **PART 1. FACTORIZATION (mydgetrf) FLOP Count**

Only this part of code for mydgetrf has floating point operations:

```c
for (int j = i + 1; j < n; j++)
{
    A[j * n + i] = A[j * n + i] / A[i * n + i];
    for (int k = i + 1; k < n; k++)
    {
        A[j * n + k] = A[j * n + k] - (A[j * n + i] * A[i * n + k]);
    }
}
```

<u>For i = 0</u>
Inner Loop Flop: 2 flops per iteration
Inner Loop Iterations: N-1
=> 2*(N-1)

Outer Loop Flops: 1 flop per iteration
Outer Loop Iterations: (N-1)

Flops for i = 0: (2*(N-1))*(N-1)

<u>For i = 0 : N-2</u>

(2*(N-1))*(N-1) + (2*(N-2))*(N-2) … (2*(N-(N-1))*(N-(N-1))

=> **LU Factorization Flops =** $2\sum_{i=1}^{N-1}(N-i)^2$

2. **PART 2. Forward Substitution**

```
for (int i = 1; i < n; i++)
{
    double sum = 0.0;
    for (int j = 0; j < i; j++)
    {
        sum += y[j] * A[i * n + j];
    }
    y[i] = B[ipiv[i]] - sum;
}
```

Inner Loop Flop: 2

Inner Loop Iterations: 1+2+ 3+4….N-1 => $\sum_{i=1}^{N-1} i = (N-1)*(N)/2$

Outer Loop Flops: 1
Outer Loop Iterations: N-1

Forward Substitution Flops = $(2*(N-1)*(N/2)) + (1*(N-1))$ **=** $N(N-1) + N-1$

3. **Part 3. Backward Substitution**

```
double x[n];
x[n - 1] = y[n - 1] / A[(n * n) - 1];

for (int i = n - 2; i >= 0; i--)
{
    double sum = 0.0;
    for (int j = i + 1; j < n; j++)
    {
        sum += x[j] * A[i * n + j];
    }
    x[i] = (y[i] - sum) / A[i * n + i];
}
```

1 Flop outside nested loops

Inner Loop Flop: 2

Inner Loop Iterations: 1, 2, 3, … N-1 => $\sum_{i=1}^{N-1} i = (N - 1) * (N)/2$

Outer Loop Flops: 2

Outer Loop Iterations: N-1

Backward Substitution Flops = $(2 * (N - 1) * (N/2)) + (2 * (N - 1)) + 1$ **=** $(N - 1)N + 2(N - 1) + 1$

TOTAL FLOPS = LU + FORWARD + BACKWARD

=> $(2 \sum_{i=1}^{N-1} (N - i)^2$ **+** $N(N - 1) + N - 1$ **+** $(N - 1)N + 2(N - 1) + 1$

TA said to assume the same flops for LAPACK and my GEPP algorithms.

MY GFLOPS PER SECOND

n	LU Flops	Forward Flops	Backward Flops	Total Flops	Time	Gflops	**Gflops per second**
1000	665667000	999999	1000999	667667998	0.156361	0.667667998	**4.27004175**
2000	5329334000	3999999	4001999	5337335998	1.680327	5.337335998	**3.176367456**
3000	17991001000	8999999	9002999	18009003998	6.541332	18.009004	**2.753109611**
4000	42650668000	15999999	16003999	42682671998	16.312457	42.682672	**2.616569165**
5000	83308335000	24999999	25004999	83358339998	32.099974	83.35834	**2.596835125**

LAPACK GLOPS PER SECOND

n	LU Flops	Forward Flops	Backward Flops	Total Flops	Time	Gflops	**Gflops per second**
1000	665667000	999999	1000999	667667998	0.063136	0.667667998	**10.57507599**
2000	5329334000	3999999	4001999	5337335998	0.20903	5.337335998	**25.53382767**
3000	17991001000	8999999	9002999	18009003998	0.663943	18.009004	**27.12432242**
4000	42650668000	15999999	16003999	42682671998	1.336713	42.682672	**31.93106673**
5000	83308335000	24999999	25004999	83358339998	2.658872	83.35834	**31.35101652**

FLOP Spreadsheet Link

PERFORMANCE ANALYSIS: The LAPACK version has significantly better computation density (gflops per second) than my code. This is because my algorithm is the non blocked GEPP version and isn't fully optimized compared to the LAPACK version. The execution time of the LAPACK program is also significantly faster than my version.

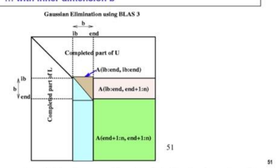
PROBLEM 3

Image demonstrates all intermediate states of A after each step of the blocked GEPP algorithm:

Problem 3 (10 points):
The following slide shows the blocked GEPP algorithm.

Blocked GEPP (www.netlib.org/lapack/single/sgetrf.f)

```
for  ib = 1 to n-1 step b     ... Process matrix b columns at a time
   end = ib + b-1             ... Point to end of block of b columns
   apply BLAS2 version of GEPP to  get A(ib:n , ib:end) = P' * L' * U'
   ... let LL denote the strict lower triangular part of A(ib:end , ib:end) + I
   A(ib:end , end+1:n) = LL^-1 * A(ib:end , end+1:n)      ... update next b rows of U
   A(end+1:n , end+1:n ) = A(end+1:n , end+1:n )
          - A(end+1:n , ib:end) * A(ib:end , end+1:n)
                  ... apply delayed updates with single matrix-multiply
                  ... with inner dimension b
```



51

Perform this algorithm on the following matrix A with block size $b = 2$ and without pivoting to achieve the LU factorization result $A = LU$:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 9 & 12 & 15 \\ 3 & 26 & 41 & 49 \\ 5 & 40 & 107 & 135 \end{bmatrix}$$

Your calculation process should show the intermediate state of A after each step (line) of the blocked GEPP algorithm.
Hint: The non-zero elements of L and U are positive integers no larger than 10.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 9 & 12 & 15 \\ 3 & 26 & 41 & 49 \\ 5 & 40 & 107 & 135 \end{bmatrix}$$

① divide by A_{11} first column

$$\Rightarrow A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 9 & 12 & 15 \\ 3 & 26 & 41 & 49 \\ 5 & 40 & 107 & 135 \end{bmatrix} \end{array}$$

since $A_{11} = 1$ matrix stays same

② Gaussian elimination 2nd column.

$$A(2,2) = A(2,2) - A(2,1)\,A(1,2) = 9 - 4 = 5$$

$$A(3,2) = A(3,2) - A(3,1)\,A(1,2) = 26 - 6 = 20$$

$$A(4,2) = A(4,2) - A(4,1)\,A(1,2) = 40 - 10 = 30$$

$$\Rightarrow A = \begin{array}{c} 1 \\ 2 \\ \\ \\ \end{array} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 12 & 15 \\ 3 & 20 & 41 & 49 \\ 5 & 30 & 107 & 135 \end{bmatrix}$$

divide column below $A_{2,2}$ by $A_{2,2} = 5$

$$\Rightarrow A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 12 & 15 \\ 3 & 4 & 41 & 49 \\ 5 & 6 & 107 & 135 \end{bmatrix}$$

$A_{3,2} = 20/5 = 4$

$A_{4,2} = 30/5 = 6$

(3) Gaussian elimination 2nd row

$$A(2,3) = A(2,3) - A(2,1)\,A(3,1) = 12 - 6 = 6$$

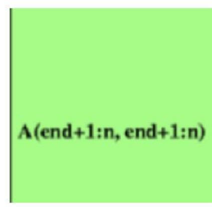
$$A(2,4) = A(2,4) - A(2,1)\,A(4,1) = 15 - 8 = 7$$

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 7 \\ 3 & 4 & 41 & 49 \\ 5 & 6 & 107 & 135 \end{bmatrix}$$

(4)

A(end+1:n , end+1:n) = A(end+1:n , end+1:n)
- A(end+1:n , ib:end) * A(ib:end , end+1:n)

Green block part.



$$A_{2\times2} = \begin{bmatrix} 41 & 49 \\ 107 & 135 \end{bmatrix} - \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 3 & 4 \\ 6 & 7 \end{bmatrix}$$

$$A_{2\times2} = \begin{bmatrix} 8 & 9 \\ 56 & 73 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 7 \\ 3 & 4 & 8 & 9 \\ 5 & 6 & 56 & 73 \end{bmatrix}$$

(5) Divide 3rd column below $A_{3,3}$ by $A_{3,3} = 8$ $\quad A_{4,3} = 56/8 = 7$

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 7 \\ 3 & 4 & 8 & 9 \\ 5 & 6 & 7 & 73 \end{bmatrix}$$

(6) Gaussian elemination on $A_{4,4}$

$$A(4,4) = A(4,4) - A(4,3)A(3,4)$$
$$= 73 - 63 = 10$$

$$\boxed{A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 7 \\ 3 & 4 & 8 & 9 \\ 5 & 6 & 7 & 10 \end{bmatrix}}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ 5 & 6 & 7 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 5 & 6 & 7 \\ 0 & 0 & 8 & 9 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$