CS211, Assignment #2 **Name: Pinxue Du**
Student ID: 862226602

Github Repository: https://github.com/UCR-HPC/cs211-hw2-solving-large-linear-system-pdu016

Notice: All GFlops are calculated by coding. Something went wrong with my laptop's python env, I can't use starter.py to test any code. So I just create new main.c and put them in a folder named by the question number. The modified file all stored in github.

Q1:

Step 1: Eliminate the 1st column

Use the 1st row to eliminate the 1st element of the 2nd and 3rd rows.

- For the 2nd row, the element in the 1st column is 4. Divide by the pivot (which is 1) to get the factor: $l_{21} = 4$. Subtract $4\times$ (1st row) from the 2nd row:

$$R_2 = \begin{bmatrix} 4 & 13 & 18 \end{bmatrix} - 4 \times \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 5 & 6 \end{bmatrix}$$

- For the 3rd row, the element in the 1st column is 7. Divide by the pivot to get the factor: $l_{31} = 7$. Subtract $7\times$ (1st row) from the 3rd row:

$$R_3 = \begin{bmatrix} 7 & 54 & 78 \end{bmatrix} - 7 \times \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 40 & 57 \end{bmatrix}$$

Now the matrix looks like:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 5 & 6 \\ 0 & 40 & 57 \end{bmatrix}$$

The partial lower triangular matrix L is:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 0 & 1 \end{bmatrix}$$

Step 2: Eliminate the 2nd column

Next, use the 2nd row to eliminate the element in the 2nd column of the 3rd row.

- For the 3rd row, the element in the 2nd column is 40. Divide by the pivot (which is 5) to get the factor: $l_{32} = 8$. Subtract $8\times$ (2nd row) from the 3rd row:

$$R_3 = \begin{bmatrix} 0 & 40 & 57 \end{bmatrix} - 8 \times \begin{bmatrix} 0 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 9 \end{bmatrix}$$

Now the matrix looks like:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 5 & 6 \\ 0 & 0 & 9 \end{bmatrix}$$

The partial lower triangular matrix L is:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 8 & 1 \end{bmatrix}$$

Final Result

We have the upper triangular matrix U and the lower triangular matrix L:

$$U = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 5 & 6 \\ 0 & 0 & 9 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 8 & 1 \end{bmatrix}$$

So the matrix A can be written as:

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

Q2:

Custom Implementation Performance

Matrix Size	Time Taken (seconds)	Performance (Gflops)
1000	0.14	4.90
2000	1.63	3.28
3000	6.49	2.78
4000	16.36	2.61
5000	32.28	2.58

Table 1: Performance of Custom LU Factorization

LAPACK Implementation Performance

Matrix Size	Time Taken (seconds)	Performance (Gflops)
1000	0.056	11.85
2000	0.229	23.29
3000	0.665	27.08
4000	1.362	31.32
5000	2.691	30.96

Table 2: Performance of LAPACK LU Factorization

Comparison of Performance (Gflops)

The results from Tables **??** and **??** show significant differences in the performance (measured in Gflops) between the custom implementation and the LAPACK implementation:

- **Custom Implementation**: The performance of the custom implementation decreases as the matrix size increases, starting at 4.90 Gflops for a matrix of size 1000 and dropping to 2.58 Gflops for a matrix of size 5000. This indicates inefficiencies in the custom implementation that become more pronounced with larger matrix sizes.

- **LAPACK Implementation**: The LAPACK implementation achieves significantly higher Gflops across all matrix sizes. For a matrix of size 1000, the performance is 11.85 Gflops, and it increases to 31.32 Gflops for a matrix of size 4000. The performance remains high even for larger matrices, with 30.96 Gflops for a matrix of size 5000.
- **Performance Gap**: The LAPACK implementation outperforms the custom implementation by a wide margin. For example, for a matrix of size 5000, the LAPACK implementation achieves a performance of 30.96 Gflops compared to only 2.58 Gflops for the custom implementation. This demonstrates the superior efficiency of LAPACK, which benefits from optimized algorithms, better cache utilization, and parallel execution.

Q3:

Step 1: Partitioning Matrix A

Given the block size $b = 2$, partition matrix A into four 2×2 submatrices:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

Where:

$$A_{11} = \begin{bmatrix} 1 & 2 \\ 2 & 9 \end{bmatrix}, \quad A_{12} = \begin{bmatrix} 3 & 4 \\ 12 & 15 \end{bmatrix}, \quad A_{21} = \begin{bmatrix} 3 & 26 \\ 5 & 40 \end{bmatrix}, \quad A_{22} = \begin{bmatrix} 41 & 49 \\ 107 & 135 \end{bmatrix}$$

Step 2: Factorize A_{11} into L_{11} and U_{11}

We aim to decompose A_{11} as:

$$A_{11} = L_{11}U_{11}$$

Where:

$$L_{11} = \begin{bmatrix} 1 & 0 \\ l_{21} & 1 \end{bmatrix}, \quad U_{11} = \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}$$

Calculations:

$$\begin{aligned} u_{11} &= 1 \\ u_{12} &= 2 \\ l_{21} &= \frac{2}{1} = 2 \\ u_{22} &= 9 - 2 \times 2 = 5 \end{aligned}$$

Resulting Matrices:

$$L_{11} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}, \quad U_{11} = \begin{bmatrix} 1 & 2 \\ 0 & 5 \end{bmatrix}$$

Intermediate State of A after Step 2:

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

Step 3: Compute $L_{21} = A_{21}U_{11}^{-1}$

First, compute the inverse of U_{11}:

$$U_{11}^{-1} = \begin{bmatrix} 1 & -\frac{2}{5} \\ 0 & \frac{1}{5} \end{bmatrix}$$

Now, calculate L_{21}:

$$L_{21} = A_{21}U_{11}^{-1} = \begin{bmatrix} 3 & 26 \\ 5 & 40 \end{bmatrix} \begin{bmatrix} 1 & -\frac{2}{5} \\ 0 & \frac{1}{5} \end{bmatrix} = \begin{bmatrix} 3 \times 1 + 26 \times 0 & 3 \times \left(-\frac{2}{5}\right) + 26 \times \frac{1}{5} \\ 5 \times 1 + 40 \times 0 & 5 \times \left(-\frac{2}{5}\right) + 40 \times \frac{1}{5} \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Resulting L_{21}:

$$L_{21} = \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Intermediate State of A after Step 3:

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

Step 4: Compute $U_{12} = L_{11}^{-1}A_{12}$

First, compute the inverse of L_{11}:

$$L_{11}^{-1} = \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix}$$

Now, calculate U_{12}:

$$U_{12} = L_{11}^{-1}A_{12} = \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 3 & 4 \\ 12 & 15 \end{bmatrix} = \begin{bmatrix} 3 \times 1 + 0 \times 12 & 4 \times 1 + 0 \times 15 \\ -2 \times 3 + 1 \times 12 & -2 \times 4 + 1 \times 15 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 6 & 7 \end{bmatrix}$$

Resulting U_{12}:

$$U_{12} = \begin{bmatrix} 3 & 4 \\ 6 & 7 \end{bmatrix}$$

Intermediate State of A after Step 4:

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

Step 5: Update $A_{22} = A_{22} - L_{21}U_{12}$

Calculate $L_{21}U_{12}$:

$$L_{21}U_{12} = \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 3 & 4 \\ 6 & 7 \end{bmatrix} = \begin{bmatrix} 3 \times 3 + 4 \times 6 & 3 \times 4 + 4 \times 7 \\ 5 \times 3 + 6 \times 6 & 5 \times 4 + 6 \times 7 \end{bmatrix} = \begin{bmatrix} 33 & 40 \\ 51 & 62 \end{bmatrix}$$

Now, update A_{22}:

$$A_{22} = \begin{bmatrix} 41 & 49 \\ 107 & 135 \end{bmatrix} - \begin{bmatrix} 33 & 40 \\ 51 & 62 \end{bmatrix} = \begin{bmatrix} 8 & 9 \\ 56 & 73 \end{bmatrix}$$

Updated A_{22}:

$$A_{22} = \begin{bmatrix} 8 & 9 \\ 56 & 73 \end{bmatrix}$$

Intermediate State of A after Step 5:

$$A^{(4)} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 9 & 12 & 15 \\ 3 & 26 & 8 & 9 \\ 5 & 40 & 56 & 73 \end{bmatrix}$$

Step 6: Factorize Updated A_{22} into L_{22} and U_{22}

We aim to decompose A_{22} as:

$$A_{22} = L_{22}U_{22}$$

Where:

$$L_{22} = \begin{bmatrix} 1 & 0 \\ l_{32} & 1 \end{bmatrix}, \quad U_{22} = \begin{bmatrix} u_{33} & u_{34} \\ 0 & u_{44} \end{bmatrix}$$

Calculations:

$$\begin{aligned} u_{33} &= 8 \\ u_{34} &= 9 \\ l_{32} &= \frac{56}{8} = 7 \\ u_{44} &= 73 - 7 \times 9 = 73 - 63 = 10 \end{aligned}$$

Resulting Matrices:

$$L_{22} = \begin{bmatrix} 1 & 0 \\ 7 & 1 \end{bmatrix}, \quad U_{22} = \begin{bmatrix} 8 & 9 \\ 0 & 10 \end{bmatrix}$$

Intermediate State of A after Step 6:

$$A^{(5)} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 9 & 12 & 15 \\ 3 & 26 & 8 & 9 \\ 5 & 40 & 56 & 73 \end{bmatrix} = \begin{bmatrix} L_{11} & U_{12} \\ L_{21} & L_{22}U_{22} \end{bmatrix}$$

Step 7: Assemble Final L and U Matrices

Combine all the computed submatrices to form the final L and U matrices.

Lower Triangular Matrix L:

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ 5 & 6 & 7 & 1 \end{bmatrix}$$

Upper Triangular Matrix U:

$$U = \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 5 & 6 & 7 \\ 0 & 0 & 8 & 9 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

Intermediate State of A after Step 7:

$$A^{(6)} = LU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ 5 & 6 & 7 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 5 & 6 & 7 \\ 0 & 0 & 8 & 9 \\ 0 & 0 & 0 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 9 & 12 & 15 \\ 3 & 26 & 41 & 49 \\ 5 & 40 & 107 & 135 \end{bmatrix} = A$$

Verification

To ensure the correctness of the factorization, we verify that $LU = A$:

$$LU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ 5 & 6 & 7 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 5 & 6 & 7 \\ 0 & 0 & 8 & 9 \\ 0 & 0 & 0 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 9 & 12 & 15 \\ 3 & 26 & 41 & 49 \\ 5 & 40 & 107 & 135 \end{bmatrix} = A$$

Since $LU = A$, the factorization is correct.

Final Matrices L and U

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

Constraints Verification:
- All non-zero elements of L and U are positive integers. - No element exceeds 10.
Thus, the LU factorization satisfies the given constraints.

Q4:

Code part:

check repo please.

Performance Comparison: Optimized vs Un-optimized (Q2)

The following tables summarize the performance comparison between the optimized version using blocked LU decomposition and the un-optimized version from Q2 for different matrix sizes. The performance metric used is Gflops (billion floating-point operations per second).

Analysis

From the performance comparison, it is evident that the un-optimized version achieves significantly higher Gflops for all matrix sizes compared to the optimized version using blocked LU decomposition. Specifically:

Matrix Size	Time Taken (seconds)	Performance (Gflops)
1000	0.792828	0.84
2000	6.442068	0.83
3000	21.934598	0.82
4000	52.082852	0.82
5000	101.900013	0.82

Table 3: Performance of Optimized Version (Blocked LU)

Matrix Size	Time Taken (seconds)	Performance (Gflops)
1000	0.14	4.90
2000	1.63	3.28
3000	6.49	2.78
4000	16.36	2.61
5000	32.28	2.58

Table 4: Performance of Un-optimized Version (Q2)

- For the un-optimized version, the performance starts at 4.90 Gflops for a matrix size of 1000 and decreases gradually to 2.58 Gflops for a matrix size of 5000. - In contrast, the optimized version has a consistent performance around 0.82 Gflops across all matrix sizes.

This indicates that the blocked LU decomposition may have introduced additional overhead, which outweighed the benefits of blocking for this specific implementation and matrix sizes. The un-optimized version, despite lacking blocking, performed better due to fewer data movement and computational overhead.