REPORT FOR CS 359

(PARALLEL COMPUTING LAB)

IMPLEMENTATION OF

BIDIRECTIONAL LU FACTORIZATION

Under Guidance of:

DR. SURYA PRAKASH

TEAM MEMBERS:

Bikash Kumar Tudu (150001006)

C. Chakradhar Reddy (150001007)

OBJECTIVE:

In this project, we will implement the parallel solution for Bidirectional LU Factorization, which is used to solve system of linear equations.

INTRODUCTION:

A system of linear algebraic equations has form $Ax = b$, where A is given $m\ x\ m$ matrix, b is given m-vector, and x is unknown solution m-vector to be computed. If a unique solution is known to exist, and the coefficient matrix is full, a direct method for solving general linear system is by computing LU factorization.

BIDIRECTIONAL LU FACTORIZATION:

The basic idea is to use left-multiplication of $A \in \mathbb{C}^{mxm}$ by (elementary) lower triangular matrices, $L_1, L_2, \ldots, L_{m-1}$ to convert A to upper triangular form, i.e.,

$$L_{m-1}\, L_{m-2} \ldots L_2 L_1 A = U \quad \Leftrightarrow \quad \tilde{L}A = U$$

Note that the product of lower triangular matrices is a lower triangular matrix, and the inverse of a lower triangular matrix is also lower triangular. Therefore,

$$\tilde{L}A = U \quad \Leftrightarrow \quad A = LU$$

where $L = \tilde{L}^{-1}$ is unit lower triangular and U is upper triangular.

This approach can be viewed as triangular triangularization.

Linear system $Ax = b$ then becomes $LUx = b$.

$$LUx = b, where\ Ux = y$$

Now by performing two steps:

1. Solve lower triangular system $Ly = b$ by forward-substitution to obtain vector y.
2. Solve upper triangular system $Ux = y$ by back-substitution to obtain solution x to original system.

The value of x is obtained.

Moreover, consider the problem $AX = B$ (i.e., many different right-hand sides that are associated with the same system matrix). In this case we need to compute the factorization $A = LU$ only once, and then

$$AX = B \quad \Leftrightarrow \quad LUX = B$$

and we proceed as before:

1. Solve LY = B by many forward substitutions (in parallel).
2. Solve UX = Y by many back substitutions (in parallel).

Example:

Let,

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

Step-1:

We choose L_1 such that left-multiplication corresponds to subtracting multiples of row 1 from the rows below such that the entries in the first column of A are zeroed out.

$$(L_1A) = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 5 \\ 4 & 6 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 3 \\ 0 & 2 & 4 \end{bmatrix}$$

Step-2:

We repeat this operation analogously for L_2 (in order to zero what is left in column 2 of the matrix on the right-hand side above):

$$L_2(L_1A) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 3 \\ 0 & 2 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & -2 \end{bmatrix} = U$$

Now $L = (L_2L_1)^{-1} = {L_1}^{-1}{L_2}^{-1}$ with

$${L_1}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 0 & 1 \end{bmatrix} \; {L_2}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix}$$

So that

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 2 & 1 \end{bmatrix}$$

L always is a unit lower triangular matrix, i.e., it has ones on the diagonal. Moreover, L is always obtained as above matrix, i.e., the multipliers are accumulated into the lower triangular part with a change of sign.

We note that the multipliers in L_k are of the form

$$l_{jk} = \frac{a_{jk}}{a_{kk}}, \quad j = k+1, \dots\dots, m$$

We formulate the factorization in **Algorithm** (LU Factorization)

SEQUENTIAL ALGORITHM – (A)

```
Initialize U = A, L = I
for k = 1: m – 1                                   -- Step - A1
        for j = k + 1 : m                          -- Step - A2
                L(j, k) = U(j, k)/U(k, k)          -- Step - A3
                U(j, k : m) = U(j, k : m) − L(j, k)U(k, k : m)   -- Step - A4
        end for
end for
end
```

PARALLEL ALGORITHM – (A)

In the above sequential algorithm, we can parallelize the following steps.

Step - A2: In this **for** loop, we update all the rows below a diagonal element. Updating a row is not dependent on any other rows except diagonal row, hence all the rows can be updated in parallel.

Step - A4: In this step we update all the elements in a row, and updating elements in a row is dependent only on a factor L(j, k), which is constant for a particular row, hence all the elements can be updated in parallel.

NOTE:

Step - A1 can't be done in parallel because Guass-elimination step cant be done in parallel.

Step - A3 is single step. There is no meaning for it to do in parallel.

COMPLEXITY ANALYSIS – (A)

Work Complexity:

In order to appreciate the usefulness of this approach note that the operations count for the matrix factorization is $O(\frac{2}{3}m^3)$, while that for forward and back substitution is $O(m^2)$.

Time Complexity:

In a row, all the elements can be updated in parallel, & rows can be updated in parallel. Hence the time complexity is

$$T(m) = O(m) * O(1) * O(1) = O(m)$$

COMPLICATIONS WITH LU FACTORIZATION:

LU Factorization is not guaranteed to be stable. The following example illustrate that

Example: A fundamental problem is given if we encounter a zero pivot (from earlier example) as in

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 5 \\ 4 & 6 & 8 \end{bmatrix} \quad \Rightarrow \quad L_1 A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 5 \\ 0 & 2 & 4 \end{bmatrix}$$

Now the (2,2) position contains a zero and the algorithm will break down since it will attempt to divide by zero.

Hence, we do Pivoting to avoid such scenarios.

PIVOTING:

The breakdown of the algorithm in our earlier example with

$$L_1 A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 3 \\ 0 & 2 & 3 \end{bmatrix}$$

can be prevented by simply swapping rows, i.e., instead of trying to apply L_2 to $L_1 A$ we first create

$$PL_1 A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad L_1 A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 3 \\ 0 & 0 & 3 \end{bmatrix}$$

and are done.

More generally, stability problems can be avoided by swapping rows before applying L_k, i.e., we perform

$$L_{m-1} P_{m-1} \dots \dots L_2 P_2 L_1 P_1 A = U$$

The strategy we use for swapping rows in step k is to find the largest element in column k below (and including) the diagonal, the so-called pivot element and swap its row with row k. This process is referred as partial (row) pivoting.

SOLUTION WITH PARTIAL PIVOTING:

Since we have the factorization PA = LU, we can solve the linear system $Ax = b$ as

$$PA\boldsymbol{x} = P\boldsymbol{b} \Leftrightarrow LU\boldsymbol{x} = P\boldsymbol{b},$$

and apply the usual two-step procedure

1. Solve the lower triangular system $L\boldsymbol{y} = P\boldsymbol{b}$ for **y**.

2. Solve the upper triangular system $U\boldsymbol{x} = \boldsymbol{y}$ for **x**.

From above, we can formulate an **Algorithm** with **Partial Pivoting**

SEQUENTIAL ALGORITHM – (B)

```
Initialize U = A, L = I, P = I
for k = 1 : m – 1                                    --Step – B1
    find i ≥ k to maximize |U(i, k)|                 --Step – B2
    U(k, k : m) ↔ U(i, k : m)                        --Step – B3
    L(k, 1 : k − 1) ↔ L(i, 1 : k − 1)                --Step – B4
    P(k, :) ↔ P(i, :)                                --Step – B5
    for j = k + 1 : m                                --Step – B6
        L(j, k) = U(j, k)/U(k, k)                    --Step – B7
        U(j, k : m) = U(j, k : m) − L(j, k)U(k, k : m)   --Step – B8
    end for
end for
end
```

PARALLEL ALGORITHM – (B)

In the above sequential algorithm, we can parallelize the following steps.

Step - B2: In this step, we find the row index of the largest element below diagonal element, which can be done in parallel using reduction.

Step – B3: In this step, we swap the diagonal row with row searched in Step – B2, and swapping of all elements of rows can be swapped in parallel.

Step – B4: This step is similar to Step – B3.

Step – B5: This step is similar to Step – B3 & Step - B4.

Step – B6: This step is similar to Step – A2 in Algorithm – (A).

Step – B8: This step is similar to Step – A4 in Algorithm – (A).

NOTE:

Step – B1 is similar to Step – A1 in Algorithm – (A).

Step – B7 is similar to Step – A3 in Algorithm – (A).

COMPLEXITY ANALYSIS – (B)

Work Complexity:

In order to appreciate the usefulness of this approach note that the operations count for the matrix factorization is $O(m^3)$, while that for swapping in partial pivoting is $O(m^2)$.

Time Complexity:

In a row, all the elements can be updated in parallel, & rows can be updated in parallel. Hence the time complexity is

$$T(m) = O(m) * O(1) * O(1) = O(m)$$

SOLUTION OF LINEAR SYSTEM OF EQUATIONS (USING L & U):

Since, we have found L & U for the linear system of equations, now we are going to find the solution for system of linear equations using these L & U.

⇒ First we have to find **y**, such that $L\boldsymbol{y} = \boldsymbol{b}$ $(or\ P\boldsymbol{b},\ in\ partial\ pivoting)$

SEQUENTIAL ALGORITHM – (C)

for $i = 1\ to\ n$ --Step – C1

$y_i = b_i$ (or $y_i = b'_i$) --Step – C2

for $j = 1\ to\ i-1$ --Step – C3

$y_i = y_i - l(i,j) * y_j$ --Step – C4

end for

end for

Note: $\boldsymbol{b}' = P\boldsymbol{b}$

PARALLEL ALGORITHM – (C)

In the above sequential algorithm, we can parallelize the following steps.

Step – C3: In this step, $l(i,j)$ is exclusive read by each thread, hence it can done in parallel using reduction.

NOTE:

Step – C1: y_i is dependent on the value of y_{i-1}. Hence cannot be done in parallel.

Step – C2 & Step – C4 are single steps. There is no meaning for it to do in parallel.

COMPLEXITY ANALYSIS – (C)

Work Complexity:

The operations count for the forward substitution is $O(\ m^2)$.

Time Complexity:

In a row, all the elements can be updated in parallel, & rows can be updated in parallel. Hence the time complexity is

$$T(m) = O(m) * O(1) = \ O(m)$$

NOTE: Reduction can be done in $O(1)$ in CRCW.

⇒ Secondly, we can find **x**, such that $U\boldsymbol{x} \ = \ \boldsymbol{y}$

SEQUENTIAL ALGORITHM – (D)

for $i \ = \ \ n\ to\ 1$ --Step – D1

  $x_i = y_i$ --Step – D2

  for $j \ = i + 1\ to\ n$ --Step – D3

    $x_i = x_i \ - U(i,j) * x_j$ --Step – D4

  end for

  $x_i = x_i\ /U(i,i)$ --Step – D5

end for

PARALLEL ALGORITHM – (D)

In the above sequential algorithm, we can parallelize the following steps.

Step – D3: In this step, $U(i,j)$ is exclusive read by each thread, hence it can done in parallel using reduction.

NOTE:

Step – D1: x_i is dependent on the value of x_{i+1}. Hence can't be done in parallel.

Step – D2, Step – D4 & Step – D5 are single steps. There is no meaning for it to do in parallel.

COMPLEXITY ANALYSIS – (D)

Work Complexity:

The operations count for the backward substitution is $O(\ m^2)$.

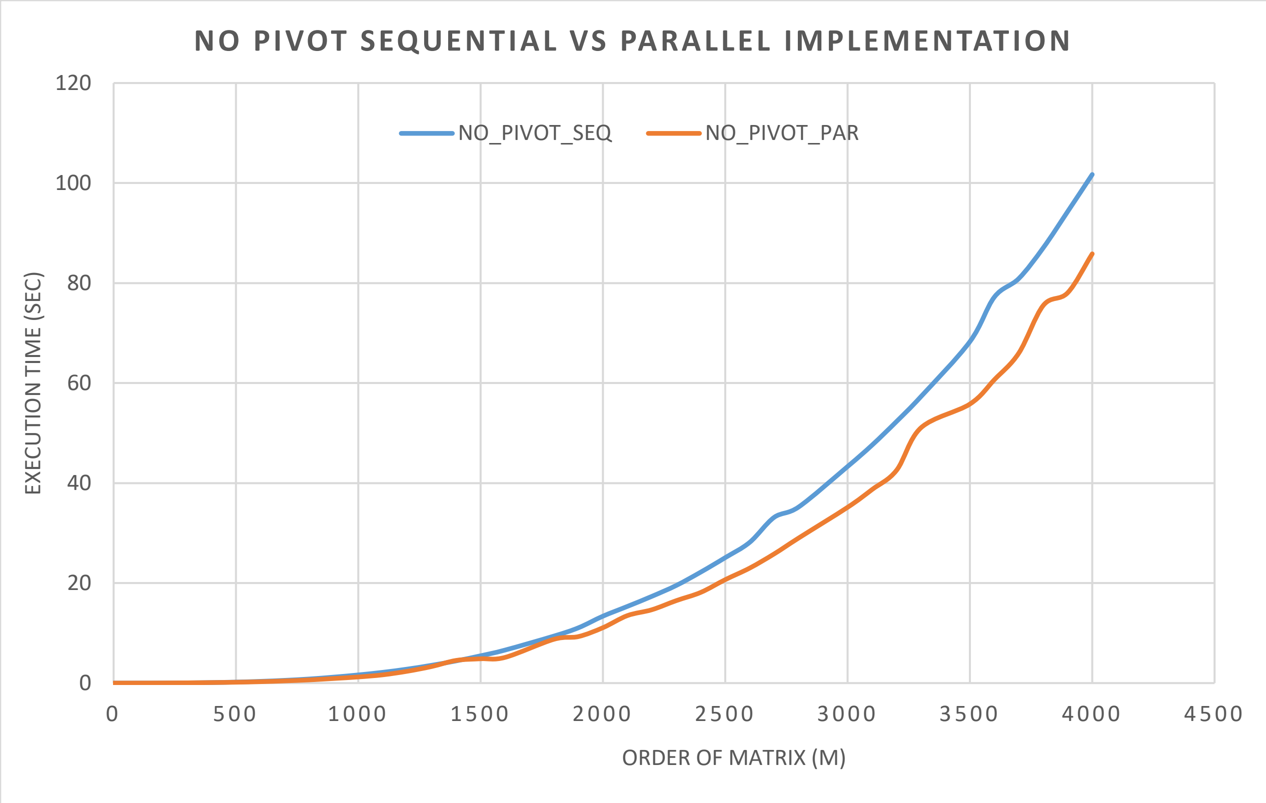
Time Complexity:

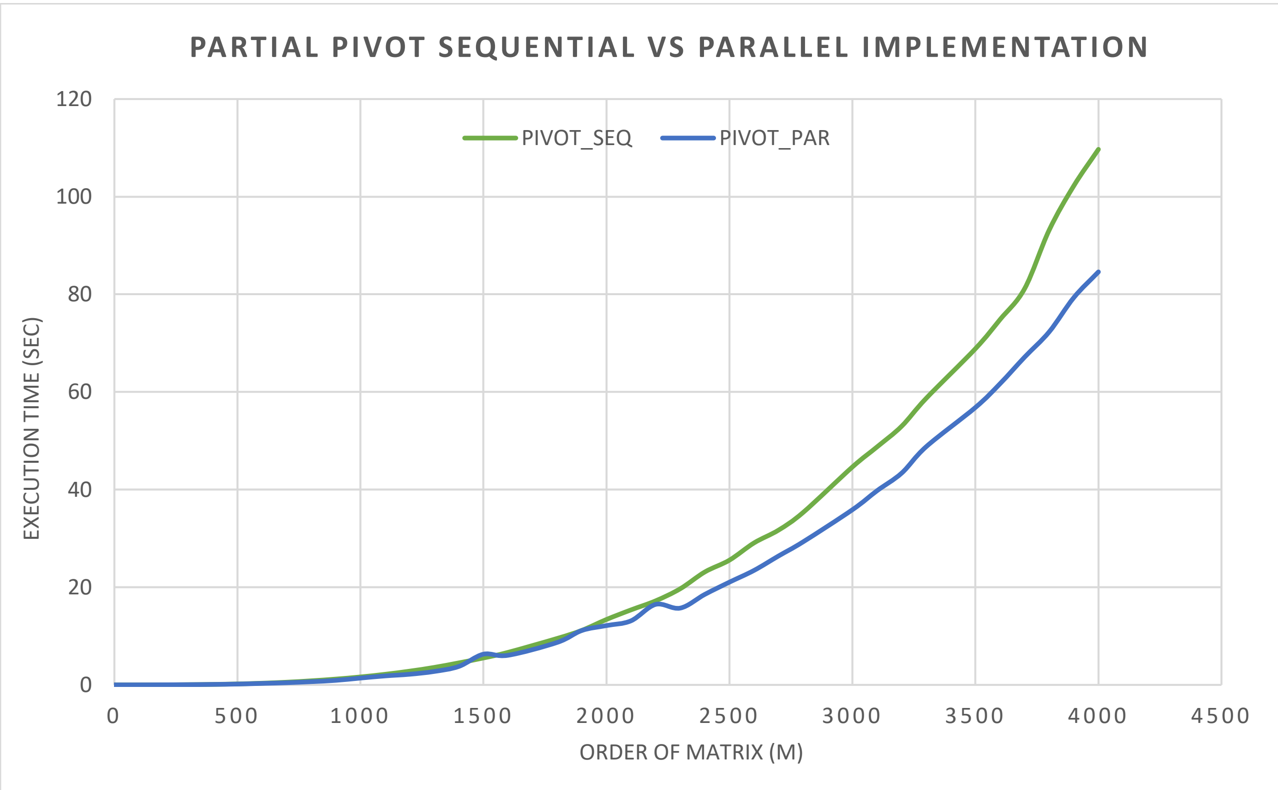
In a row, all the elements can be updated in parallel, & rows can be updated in parallel. Hence the time complexity is

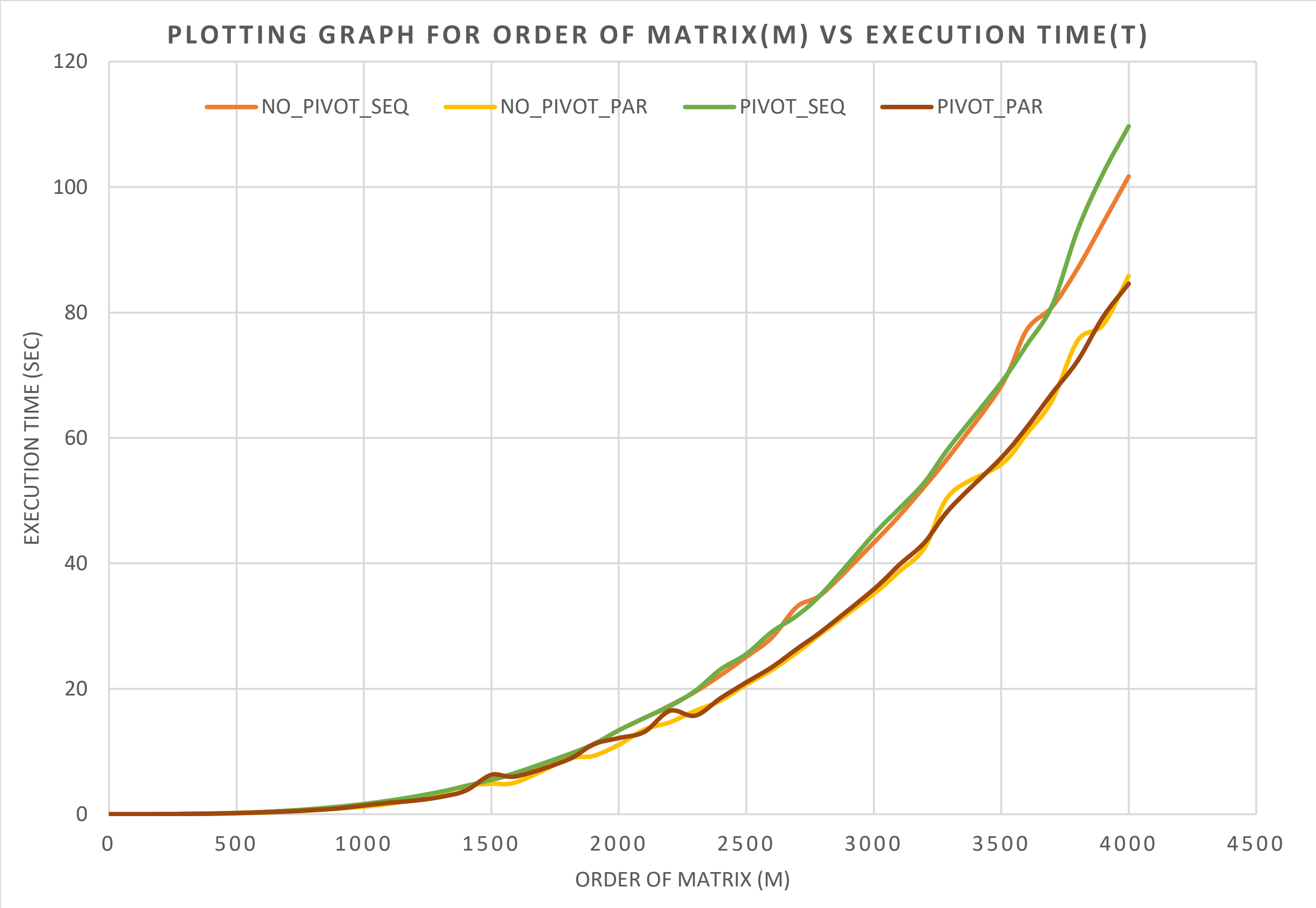
$$T(m) = O(m) * O(1) = \ O(m)$$

NOTE: Reduction can be done in $O(1)$ in CRCW.

GRAPHS PLOTTING ORDER OF MATRIX (M) VS EXECUTION TIME (SEC):







OBSERVATION:

For example,

if m = 3000,

No_Pivot Implementation,

$T_s = 43.308$ s, $T_p = 35.146$ s

Speedup = $\frac{Ts}{Tp} = \frac{43.308}{35.146} = 1.232$

Pivot Implementation,

$T_s = 44.638$ s, $T_p = 35.876$ s

Speedup = $\frac{Ts}{Tp} = \frac{44.638}{35.876} = 1.244$

CONCLUSION:

In this project, we have showed that by using parallel implementation (OpenMP) we can reduce the execution time.

FUTURE SCOPE:

In Future, we would try to implement efficiently the MPI implementation of LU Factorisation.

SUMMARY:

In this project, we parallelized LU Factorization, forward and backward substitutions using OpenMP parallel library and we observed speedup compared to sequential implementation.

NOTE:

All the programs are executed in machine configuration

- 5th Gen Intel® CoreTM i7-5500U Processor 4M Cache, up to 3.00 GHz
- 8 GB DDR3 Memory
- 4 Hyperthreads

⇒ Project Repository for this project can be found at the following link

LU Factorisation Project Repository