ECE-5772- HIGH PERFORMANCE EMBEDDED PROGRAMMING

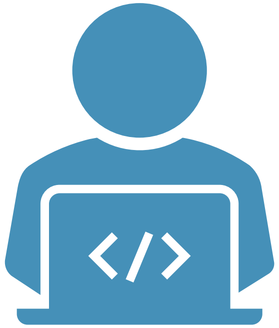
ENHANCING MATRIX INVERSION EFFICIENCY

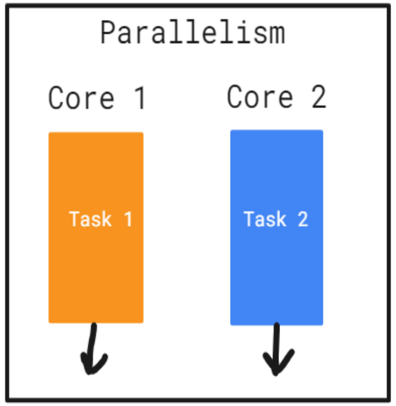
A PARALLEL IMPLEMENTATION OF LU DECOMPOSITION AND CHOLESKY DECOMPOSITION

INTRODUCTION

- Matrix inversion is a critical computational task in various scientific and engineering applications, such as embedded systems, signal processing, control systems, machine learning, real-time systems etc.,
- We have implemented two matrix inversion algorithms- Cholesky and LU decomposition in sequential and parallel.
- The significant performance enhancements are attained through parallel programming.

Inverse of $\begin{pmatrix} 1, 1, 1 \\ 4, 3, -1 \\ 3, 5, 3 \end{pmatrix}$ is $\begin{pmatrix} 7/5, 1/5, -2/5 \\ -3/2, 0, 1/2 \\ 11/10, -1/5, -1/10 \end{pmatrix}$





PROGRAMMING LANGUAGE: C,C++
LANGUAGE

PARALLELISATION STRATEGY:
Intel TBB –Parallel_for

SOFTWARE REQUIREMENTS

LU DECOMPOSITION MATRIX INVERSION

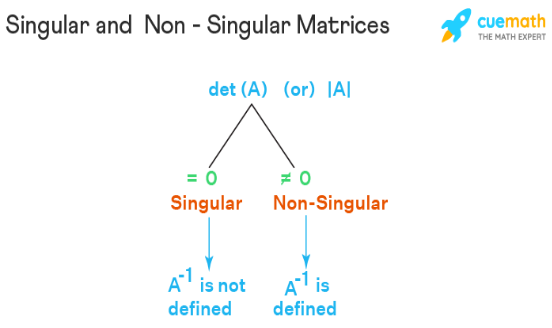
An LU decomposition of a matrix A is the product of a lower triangular matrix(L) and an upper triangular matrix(U) that is equal to A.
A=LU;

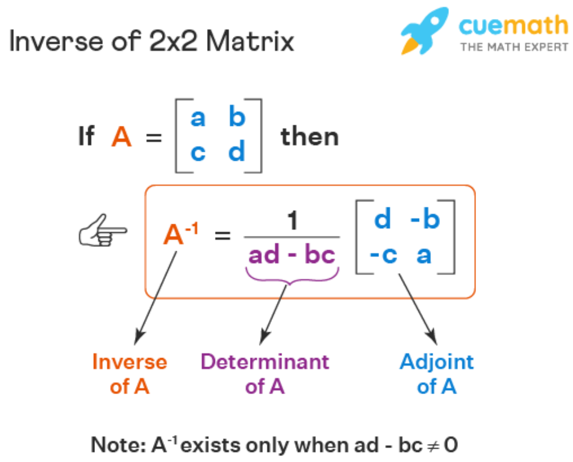
Lower Triangular

$$\begin{bmatrix} A00 & A01 & A02 \\ A10 & A11 & A12 \\ A20 & A21 & A22 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L10 & 1 & 0 \\ L20 & L21 & 1 \end{bmatrix} \begin{bmatrix} U00 & U01 & U02 \\ 0 & U11 & U12 \\ 0 & 0 & U22 \end{bmatrix}$$

Upper Triangular

LU DECOMPOSITION MATRIX INVERSION-ALGORITHM EXPLANATION





$$L=\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix} \qquad U=\begin{pmatrix}1, & 1, & 1\\ 4, & 3, & -1\\ 3, & 5, & 3\end{pmatrix}$$

Copy of input matrix

Row operation to get L and U matrices:

R2=R2-(4*R1); R3=R3-(3*R1);R3=R3-(-2*R2);-->gives U matrix

Put this multiplier in corresponding L matrix –gives L matrix

$$L=\begin{pmatrix}1 & 0 & 0\\ 4 & 1 & 0\\ 3 & -2 & 1\end{pmatrix} \qquad U=\begin{pmatrix}1, & 1, & 1\\ 0, & -1, & -5\\ 0, & 0, & -10\end{pmatrix}$$

A=L*U

LU DECOMPOSITION MATRIX INVERSION--ALGORITHM EXPLANATION

A*X=I; A=L*U; (LU)*X=I; U*X=d; L*d=I.

FORWARD SUBSTITUTION (solve from top to bottom)

L*d=I; to get d, where L- lower triangular matrix, d-intermediate matrix, I-identity matrix
Solve column by column to compute d matrix.

$$L=\begin{pmatrix}1&0&0\\4&1&0\\3&-2&1\end{pmatrix}\quad d=\begin{pmatrix}d00&d01&d02\\d10&d11&d12\\d20&d21&d22\end{pmatrix}\quad I=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}$$

BACKWARD SUBSTITUTION (solve from bottom to top)

U*X=d; to get X, where U- upper triangular matrix, X-inverse matrix, d-intermediate matrix(received from forward substitution)
Solve column by column to compute X matrix.

$$U=\begin{pmatrix}1&1&1\\0&-1&-5\\0&0&-10\end{pmatrix}\quad x=\begin{pmatrix}x00&x01&x02\\x10&x11&x12\\x20&x21&x22\end{pmatrix}\quad d=\begin{pmatrix}1&0&0\\-4&1&0\\-11&2&1\end{pmatrix}$$

VERIFICATION (verified using MATLAB)

A*X=I; where A- input matrix, X-inverse of A, I- Identity matrix.
Identity matrix computed from c code is written to a binary file and this binary file is compared with the identity matrix generated by the MATLAB code and the difference between those is displayed.

$$A*X=\begin{pmatrix}1&1&1\\4&3&-1\\3&5&3\end{pmatrix} * \begin{pmatrix}\textbf{7/5, 1/5,–2/5}\\\textbf{is -3/2, 0, 1/2}\\\textbf{11/10, -1/5, -1/10}\end{pmatrix}=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}$$

RESULTS OF TIME STAMP ANALYSIS OF SEQUENTIAL IMPLEMENTATION

- Slower processing speed
- Limited Real time application usage
- Scalability issues
- Inefficient Resource Utilization

SIZE(number of rows and columns)	Computation time(in milliseconds)
100	6
500	297
1000	2428
1500	9499
1750	18699
2000	42017

PARALLELISATION STRATEGY-MAP PATTERN

After initialization, we used **Intel TBB parallel_for (map pattern)** for decomposition of A into L and U, **Forward substitution** to get d, **backward substitution** to compute X(output).

DECOMPOSING A INTO L AND U: parallel_for-row
FORWARD AND BACKWARD SUBSTITUTION: parallel_for-column

For decomposing A into L and U, by using map pattern we have parallelized the rows to run in a parallel way in order to simultaneous update the values of L and U matrices.

For forward, backward substitution, main task may get divided into 3 tasks (theses 3 tasks will run in parallel however each task will run sequentially inside it), so each task may handle each column values to compute d, X from Ld=I; UX=d;

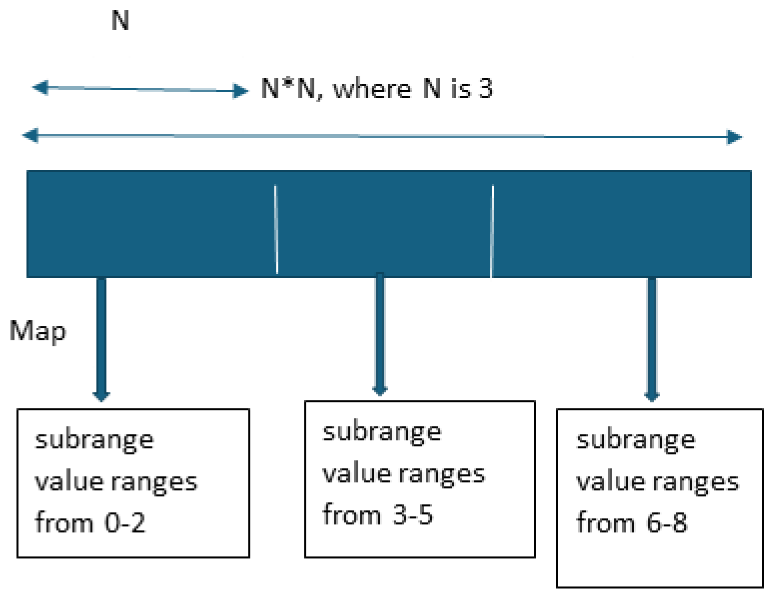


Fig. 2 depicts one possible implementation of the map pattern.

THREAD SAFE IMPLEMENTATION
where L, I- for forward and U, d-for backward, matrices are shared among 3 tasks, but these tasks won't modify/update the L, I in forward and U, d in backward, it will just read values from L and I, and U and d to compute d [3][3] and X [3][3]so that we can prevent the race condition.

Map pattern will divide the rows/columns into different subranges and it will assign these subranges into different threads and the scheduler will assign these different threads into different cores available in the system

PARALLELISATION STRATEGY-PARALLEL PIPELINE

After initialization and decomposition of A into L and U, we used **Intel TBB parallel pipeline** , for **Forward substitution** to get d, and **backward substitution** to compute X(output).

We have implemented the forward and backward substitution by using parallel pipeline.

- **First stage:** Giving columns indices as input
- **Second stage:** Forward substitution to compute intermediate matrix(d)
- **Third stage:** Backward substitution to compute inverse matrix(x) using intermediate matrix(d) from forward substitution.

Fig. 2 depicts one possible implementation of the parallel pipeline

PIPELINE IMPLEMENTATION

We have given number of token as 16, in the first stage 16 columns indices will be processed and given as an input to the second stage where the forward substitution will solve 16 columns in parallel to compute the intermediate matrix (d) and once the each columns is processed it will be send as an input to the final stage where backward substitution takes place to compute inverse matrix(x) using intermediate matrix(d) values from previous stage like this till the column values raeches the size of given matrix this process will continue to compute the inverse.

RESULTS
TIME STAMP ANALYSIS OF SERIAL AND PARALLEL

Intell_TBB-parallel_for

SIZE (number of rows and columns)	Computation time(ms)
100	2
500	63
1000	877
1500	4295
1750	8140
2000	14378

Intell_TBB-parallel_pipeline

SIZE(number of rows and columns)	Computation time(ms)
100	5
500	71
1000	841
1500	4274
1750	8126
2000	14257

Programming methods for size=2000	Time to compute inverse (in milliseconds) and speed improvement via parallelization (in %)
Sequential	42017
Intel-TBB- Parallel_for	14378(65%)
Intel-TBB- Parallel_pipeline	14257(66%)

RESULT
LU DECOMPOSITION MATRIX INVERSION- TERMINAL OUTPUT FOR SIZE 2000

```
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/pipe$ make clean
rm -f pipe *.o core
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/pipe$ make all
g++ -O3 -Wall -std=c++11 -c pipe_fun.cpp -ltbb -lm
g++ -O3 -Wall -std=c++11 main.cpp pipe_fun.o -lm  -ltbb -o pipe
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/pipe$ ./pipe
Enter the size of the matrix (n x n): 2000
Starting LU decomposition and inversion with pipeline...
Pipeline elapsed time: 14415 ms
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/pipe$ cd ..
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project$ cd paralle_for
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/paralle_for$ make clean
rm -f for *.o core
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/paralle_for$ make all
g++ -O3 -Wall -std=c++11 -c par_fun.cpp -ltbb -lm
g++ -O3 -Wall -std=c++11 main.cpp  par_fun.o -lm  -ltbb -o for
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/paralle_for$ ./for
Enter the size of the matrix (n x n): 2000
Starting LU decomposition and inversion...
Sequential elapsed time: 14493 ms
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/paralle_for$ cd ..
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project$ cd sequential
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/sequential$ make clean
rm -f seq *.o core
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/sequential$ make all
g++ -O3 -Wall  -c seq_fun.cpp  -lm
g++ -O3 -Wall  main.cpp  seq_fun.o -lm   -o seq
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/sequential$ ./seq
Enter the size of the matrix (n x n): 2000
Starting LU decomposition and inversion...
Sequential elapsed time: 40531 ms
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/sequential$
```

RESULT
LU DECOMPOSITION MATRIX INVERSION- TERMINAL OUTPUT FOR SIZE 3

```
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/pipe$ make clean
rm -f pipe *.o core
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/pipe$ make all
g++ -O3 -Wall -std=c++11 -c pipe_fun.cpp -ltbb -lm
g++ -O3 -Wall -std=c++11 main.cpp pipe_fun.o -lm  -ltbb -o pipe
ramya-rajaraman@ramya-rajaraman-Latitude-3500:~/hpep_project/pipe$ ./pipe
Enter the size of the matrix (n x n): 3
Input Matrix:
7.00 4.00 1.00
3.00 5.00 5.00
5.00 9.00 2.00
Starting LU decomposition and inversion with pipeline...
Inverse Matrix:
0.21 -0.01 -0.09
-0.11 -0.05 0.19
-0.01 0.26 -0.14
Pipeline elapsed time: 2 ms
Verification Matrix (A * A_inv):
1.00 0.00 0.00
-0.00 1.00 0.00
-0.00 -0.00 1.00
```

ANSWER

The inverse matrix is $\begin{bmatrix} \frac{35}{167} & -\frac{1}{167} & -\frac{15}{167} \\ -\frac{19}{167} & -\frac{9}{167} & \frac{32}{167} \\ -\frac{2}{167} & \frac{43}{167} & -\frac{23}{167} \end{bmatrix} \approx$

$$\begin{bmatrix} 0.209580838323353 & -0.005988023952096 & -0.089820359281437 \\ -0.11377245508982 & -0.053892215568862 & 0.191616766467066 \\ -0.011976047904192 & 0.25748502994012 & -0.137724550898204 \end{bmatrix}.$$

A

MATLAB RESULT
LU DECOMPOSITION MATRIX INVERSION-SEQUENTIAL

/ ▸ home ▸ ramya-rajaraman ▸ hpep_project ▸ sequential ▸

Current Folder

Name

- seq_fulltime
- final_with_tolerance.m
- final_without_pivot.m
- iss.bof
- iss_100.bof
- iss_500.bof
- iss_1000.bof
- iss_1500.bof
- iss_1750.bof
- iss_2000.bof
- lu_decompose.m
- main.cpp
- makefile
- matlab_result_matrix.bof
- matlab_result_matrix_no_pivoting.bof
- project_1.m
- seq
- seq_fun.cpp
- seq_fun.o
- seq_header.h

lu_decompose.m (Script)

MATLAB script to replicate the C code functionality

Editor - /home/ramya-rajaraman/hpep_project/sequential/lu_decompose.m

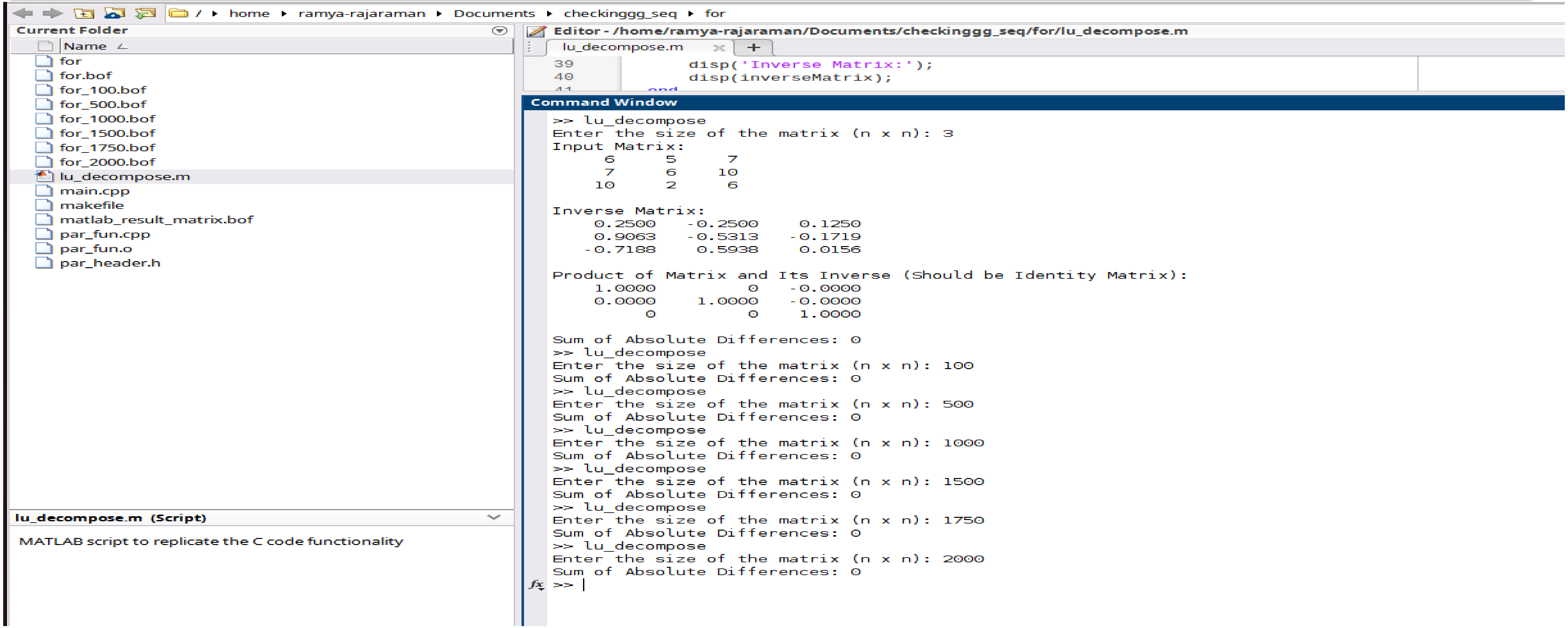
lu_decompose.m

```
    disp('Product of Matrix and Its Inverse (Should be Identity Matrix):');
    disp(resultMatrix);
end

% Write the result matrix to a binary file
filename = 'matlab_result_matrix.bof';
fileID = fopen(filename, 'wb');
fwrite(fileID, resultMatrix, 'double');
fclose(fileID);

% Read the binary file generated by the C code
```

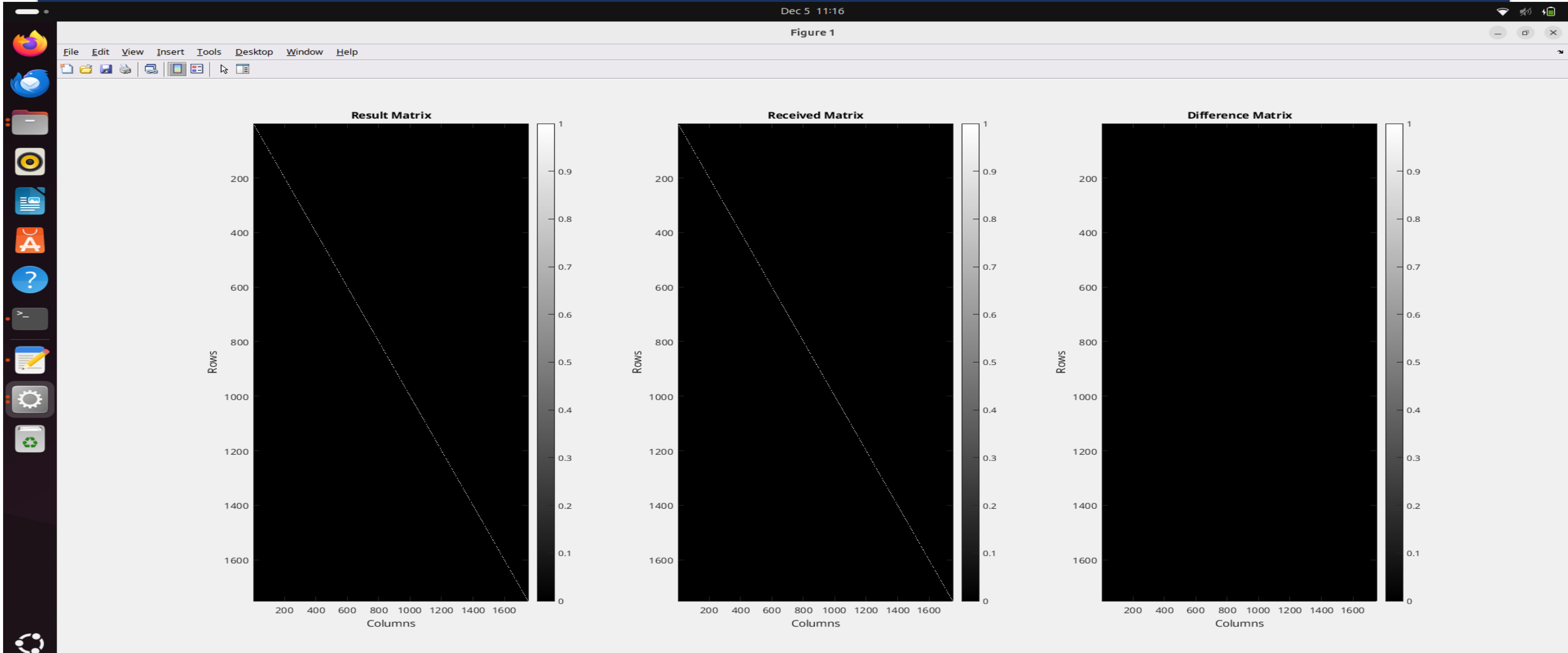
Command Window

```
>> lu_decompose
Enter the size of the matrix (n x n): 3
Input Matrix:
     8     6     8
     8     9     7
     4     3     6

Inverse Matrix:
    0.6875   -0.2500   -0.6250
   -0.4167    0.3333    0.1667
   -0.2500         0    0.5000

Product of Matrix and Its Inverse (Should be Identity Matrix):
     1     0     0
     0     1     0
     0     0     1

Sum of Absolute Differences: 0
>> lu_decompose
Enter the size of the matrix (n x n): 2000
Sum of Absolute Differences: 0
>> lu_decompose
Enter the size of the matrix (n x n): 1500
Sum of Absolute Differences: 0
>> lu_decompose
Enter the size of the matrix (n x n): 1750
Sum of Absolute Differences: 0
>> |
```

MATLAB RESULT
LU DECOMPOSITION MATRIX INVERSION-PARALLEL

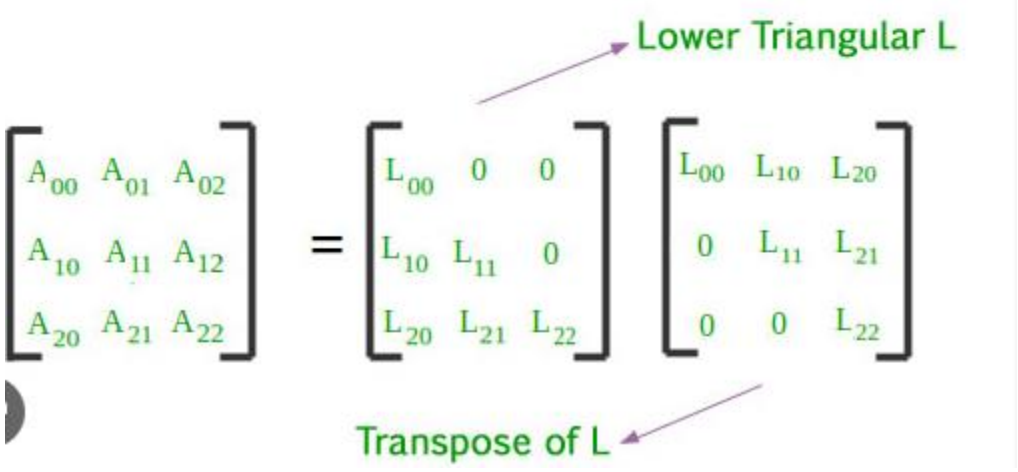


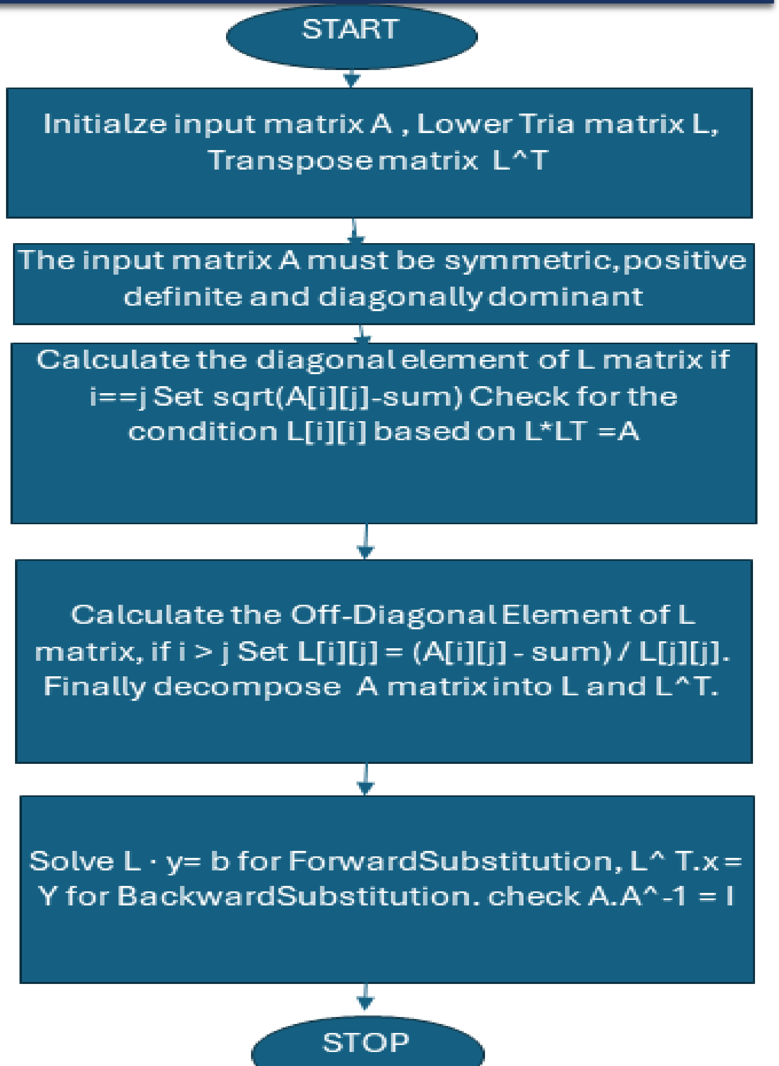
MATLAB RESULT
LU DECOMPOSITION MATRIX INVERSION-PARALLEL



CHOLESKY DECOMPOSITION MATRIX INVERSION

A cholesky decomposition is a mathematical method used to decompose a symmetric, positive definite matrix A in to the product of a lower triangular matrix(L) and its transpose L^T.
A=L* L^T;





CHOLESKY DECOMPOSITION MATRIX INVERSION ALGORITHM STEPS

STEPS TO FOLLOW

Symmetric matrix A = A^T (A[i][j]=A[j][i])

$$A = \begin{pmatrix} 4 & 2 & 3 \\ 2 & 5 & 6 \\ 3 & 6 & 7 \end{pmatrix} \qquad A^T = \begin{pmatrix} 4 & 2 & 3 \\ 2 & 5 & 6 \\ 3 & 6 & 7 \end{pmatrix}$$

To verify the symmetry of the matrix manually, we check the following:

1. The element at position $(1, 2)$ in A is 2, and it is equal to the element at position $(2, 1)$ in A.
2. The element at position $(1, 3)$ in A is 3, and it is equal to the element at position $(3, 1)$ in A.
3. The element at position $(2, 3)$ in A is 6, and it is equal to the element at position $(3, 2)$ in A.

Since all corresponding off-diagonal elements are equal, the matrix is symmetric.

Positive-Definite matrix

If $A = \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}$:

- $D_1 = 4$ (the top-left element of A).
- $D_2 = \det(A) = (4)(3) - (2)^2 = 12 - 4 = 8$.

Since $D_1 > 0$ and $D_2 > 0$, the matrix is **positive definite**.

Diagonal values must be positive

- symmetric and positive definite matrix.

CHOLESKY DECOMPOSITION ALGORITHM EXPLANATION

Decompose A = L * L^T

$$A = \begin{bmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{bmatrix}$$

$$L = \begin{bmatrix} ? & 0 & 0 \\ ? & ? & 0 \\ ? & ? & ? \end{bmatrix}$$

Compute the diagonal element L[0][0], where i==j

- Take A[0][0]=4 , Since it's the diagonal element, calculate:
- L[0][0]=sqrt(A[0][0]) =sqrt(4) =2

Compute the Off -diagonal element L[1][0], where i > j

- **L[1][0] = A[1][0] - sum / L[0][0] ,**
- Here, sum - sum of product of already computed elements of col1 and row1
- L[1][0] = 12 – 0 / 2 => 6

Compute the Off -diagonal element L[2][0], where i > j

- L[2][0] = A[2][0] - sum / L[0][0] ,
- L[2][0] = -16 – 0/2 => -8

Compute the diagonal element L[1][1]

$$L[1][1] = \sqrt{A[1][1] - \text{sum}}$$

Where, Sum - sum of squares of already calculated value in the same row

L[1][1] = sqrt(A[1][1] - L[1][0] * L[1][0])

sqrt(37 - (6*6)) = sqrt(1) =>1

Compute the Off -diagonal element L[2][1], where i > j

L[2][1] = A[2][1] - sum / L[1][1] , where sum = L[2][0]*L[1][0]

L[2][1] = - 43 - (-8 * 6) = - 43 + 48 / 1 => 5

Similarly for L[2][2] -

sqrt(A[2][2] - L[2][0]* L[2][0] + L[2][1]*L[2][1]

sqrt(98-(8*8) + (5*5)) = sqrt(98 - 89) = sqrt(9) => 3

Diagonal computation(Serial)

- Sequential dependency & Smaller workload
- Numerical stability considerations Therefore ,

$$L = \begin{bmatrix} 2 & 0 & 0 \\ 6 & 1 & 0 \\ -8 & 5 & 3 \end{bmatrix}$$

CHOLESKY DECOMPOSITION MATRIX INVERSION ALGORITHM EXPLANATION

A = L* L^T ; L* Y = B; L^T * X = Y ; A*X=I;

FORWARD SUBSTITUTION (solve from top to bottom)

L*Y=B; to get Y, where L- lower triangular matrix, Y-intermediate matrix, B-Identity matrix. Solve column by column to compute Y matrix.

$$L=\begin{pmatrix} 2 & 0 & 0 \\ 6 & 1 & 0 \\ -8 & 5 & 3 \end{pmatrix} \quad y=\begin{pmatrix} y00 & y01 & y02 \\ y11 & y12 & y13 \\ y21 & y22 & y23 \end{pmatrix} \quad b=\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

BACKWARD SUBSTITUTION (solve from bottom to top)

L^T*X=Y; to get X, where L^T- Transpose matrix, X-inverse matrix, Y-intermediate matrix(received from forward substitution)
Solve column by column to compute X matrix.

$$L\text{^}t=\begin{pmatrix} 2 & 6 & -8 \\ 0 & 1 & 5 \\ 0 & 0 & 3 \end{pmatrix} \quad x=\begin{pmatrix} x00 & x01 & x02 \\ x11 & x12 & x13 \\ x21 & x22 & x23 \end{pmatrix} \quad y=\begin{pmatrix} 1/2 & 0 & 0 \\ -3 & 1 & 0 \\ 19/3 & -5/3 & 1/3 \end{pmatrix}$$

VERIFICATION (verified using MATLAB)

A*X=I; where A- input matrix, X-inverse of A, I- Identity matrix.
Identity matrix computed from c code is written to a bof file and this bof file is compared with the identity matrix generated by the MATLAB code bof file , and the difference between those will displayed.

$$A*X=\begin{pmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{pmatrix} * \begin{pmatrix} 1/18 , & -122/9, & 19/9 \\ -62/9, & 34/9, & -5/9 \\ 19/9 , & -5/9, & 1/9 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

CHOLESKY SEQUENTIAL IMPLEMENTATION

- Long Execution Time
- Poor Fault Tolerance
- Limited Throughput

SIZE(number of rows and columns)	Computation time(ms)
100	5
500	338
750	988
1000	2217
1500	7104
2000	17180

CHOLESKY PARALLELISATION STRATEGY-MAP PATTERN

After initialization , we used Intel TBB parallel_for (map pattern) for computing **cholesky decomposition, Foward substitution** to get Y, **backward substitution** to get X(inverse).

To decompose A in to L and L^T , we used row wise parallelization to compute the off-diagonal elements. Each thread will assigned to one or more rows and processed simultaneously.

For computing the inverse of matrix, main task may get divided into 3 tasks (these 3 tasks will run in parallel, however each task will run sequentially inside it), so each task may handle each column values to compute Y, X from Ly=b; L^T*X= y;

THREAD SAFE IMPLEMENTATION

where L, b- for forward and L^T, Y-for backward, matrices areshared among 3 tasks, but these tasks won't modify/update the L, b in forward and L^T, Y in backward, it will just read values from L and b, and L^T and Y to compute Y [3][3] and X [3][3]so that we can prevent the race condition.

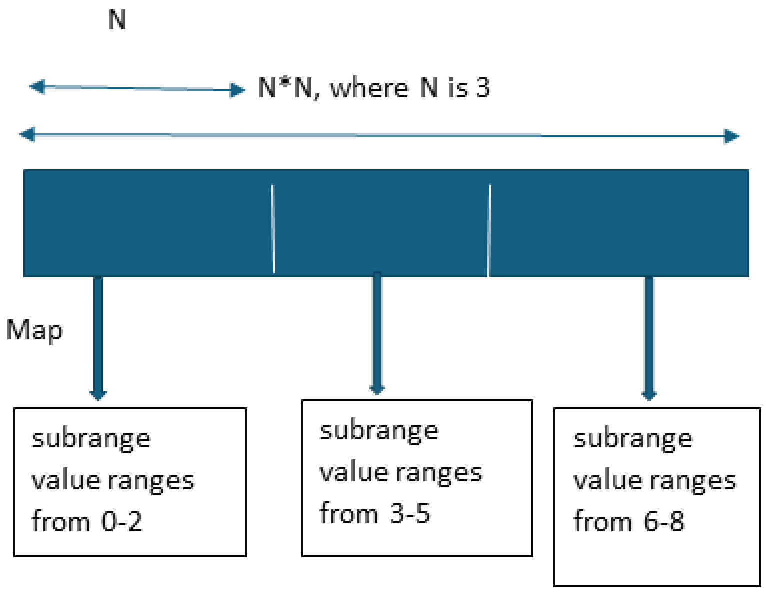


Fig. 2 depicts one possible implementation of the map pattern.

CHOLESKY PARALLELISATION STRATEGY- PARALLEL PIPELINE

After initialization , we used Parallel pipeline for computing **Foward substitution** to get Y, **backward substitution** to get X(inverse).

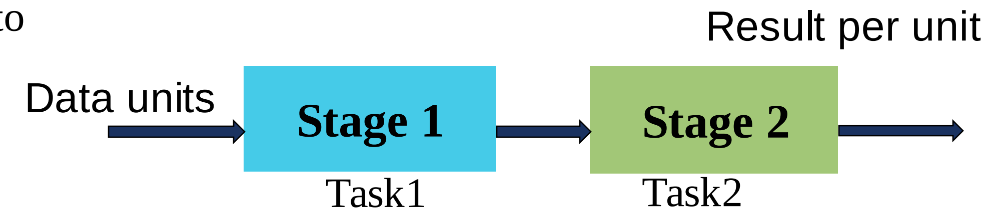


Fig. Two stage Parallel Pipeline

We use 2 main stages in the Parallel pipeline implementation.
Stage 1 : Generates column indices col represents the column of the inverse matrix A^-1
Stage 2: Compute the inverse of the given matrix by doing forward and backward substitution by getting the column indices from stage1, and it will store the inverse

IMPLEMENTATION OF PIPELINE

The pipeline implementation involves stage-2 pipeline. The stage-1 is serial stage which generates col indices, that represents the current column to process. The stage-2 is parallel stage , it takes the column index (col) from Stage 1 and performs the calculations to compute the inverse of that column using forward and backward substitution. Combine the work of forward substitution, backward substitution, and assignment to the inverse matrix A^-1 in a single stage. Col 1 - solve L* y = e , col2- solve L^T*X = Y, col 3 - store X in the col(column of A^-1).

CHOLESKY RESULTS
TIME STAMP ANALYSIS OF SERIAL AND PARALLEL

Intell_TBB-parallel_for

SIZE (number of rows and columns)	Computation time(ms)
100	3
500	111
750	168
1000	330
1500	1094
2000	2675

Intell_TBB-parallel_pipeline

SIZE(number of rows and columns)	Computation time(ms)
100	1
500	40
750	137
1000	313
1500	1039
2000	3015

Programming methods for size=2000	**Time to compute inverse (in seconds) and speed improvement via parallelization (in %)**
Sequential	17180
Intel-TBB- Parallel_for	2675 (84%)
Intel-TBB- Parallel_pipeline	3015 (82%)

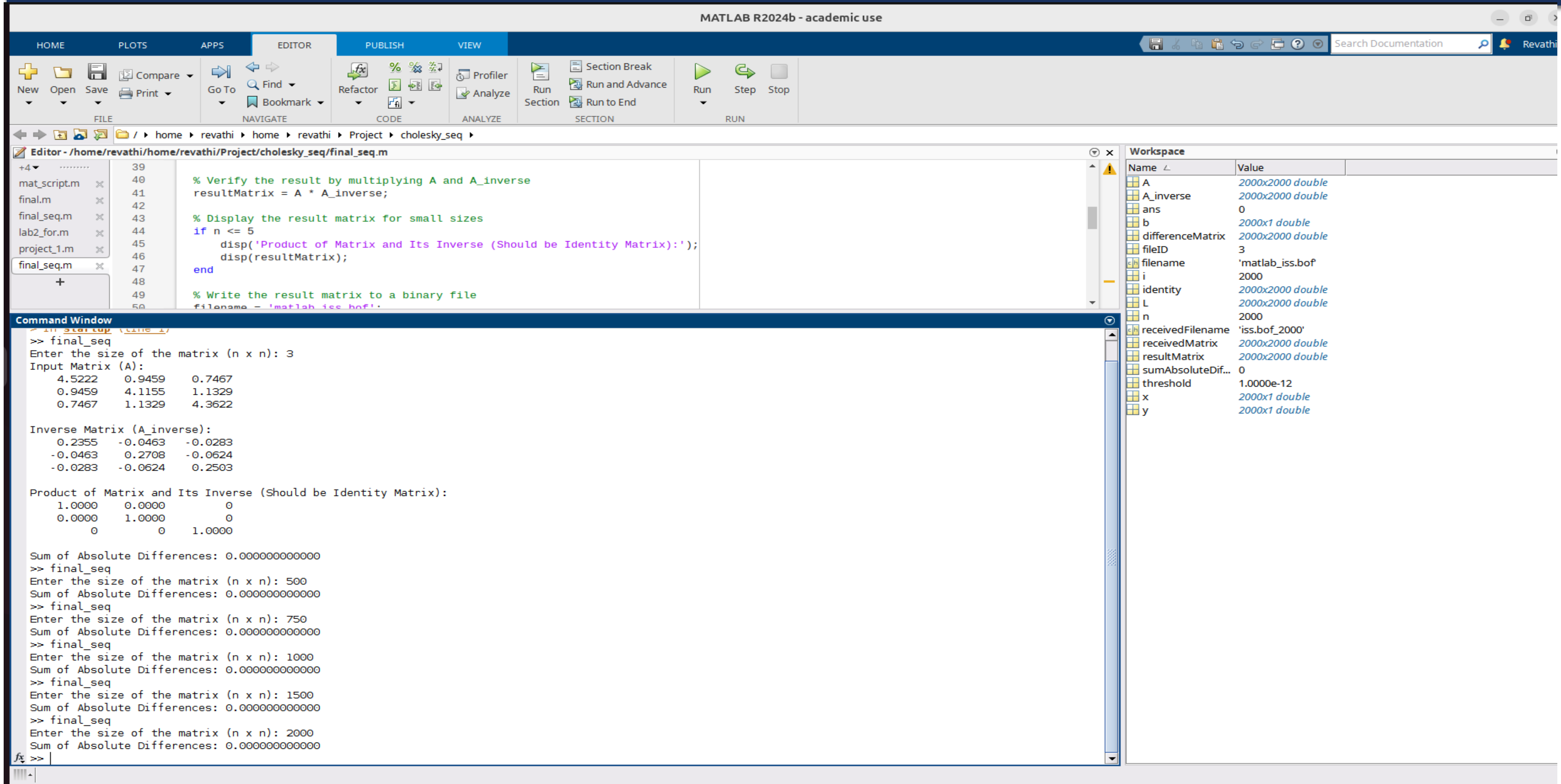
RESULT- SEQ, PARALLEL,PIPELINE CHOLESKY DECOMPOSITION MATRIX INVERSION- TERMINAL OUTPUT FOR SIZE 2000

```
revathi@revathi-Latitude-3510: ~/home/revathi/Projec
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_seq$ make clean
rm -f seq *.o core
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_seq$ make all
g++ -O3 -Wall -c seq_fun.cpp -lm
g++ -O3 -Wall -o seq seq.cpp seq_fun.o -lm
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_seq$ ./seq
Enter the size of the matrix: 2000
The inverse of the matrix A has been computed.
Elapsed time: 17816 ms
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_seq$ cd ..
revathi@revathi-Latitude-3510:~/home/revathi/Project$ cd cholesky_para/
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_para$ make clean
rm -f tbb_for *.o core
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_para$ make all
g++ -O3 -Wall -std=c++11 -c tbb_fun.cpp -ltbb
g++ -O3 -Wall -std=c++11 tbb_for.cpp tbb_fun.o -ltbb -o tbb_for
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_para$ ./tbb_for
Enter the size of the matrix: 2000
The inverse of the matrix A has been computed.
Elapsed time: 2684 ms
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_para$ cd ..
revathi@revathi-Latitude-3510:~/home/revathi/Project$ cd cholesky_pipeline/
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_pipeline$ make clean
rm -f pipe *.o core
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_pipeline$ make all
g++ -O3 -Wall -std=c++11 -c pipe_fun.cpp -ltbb -lm
g++ -O3 -Wall -std=c++11 pipe.cpp pipe_fun.o -lm  -ltbb -o pipe
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_pipeline$ ./pipe
Enter the size of the matrix: 2000
The inverse of the matrix A has been computed successfully.
Elapsed time: 2563 ms
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_pipeline$ cd ..
```

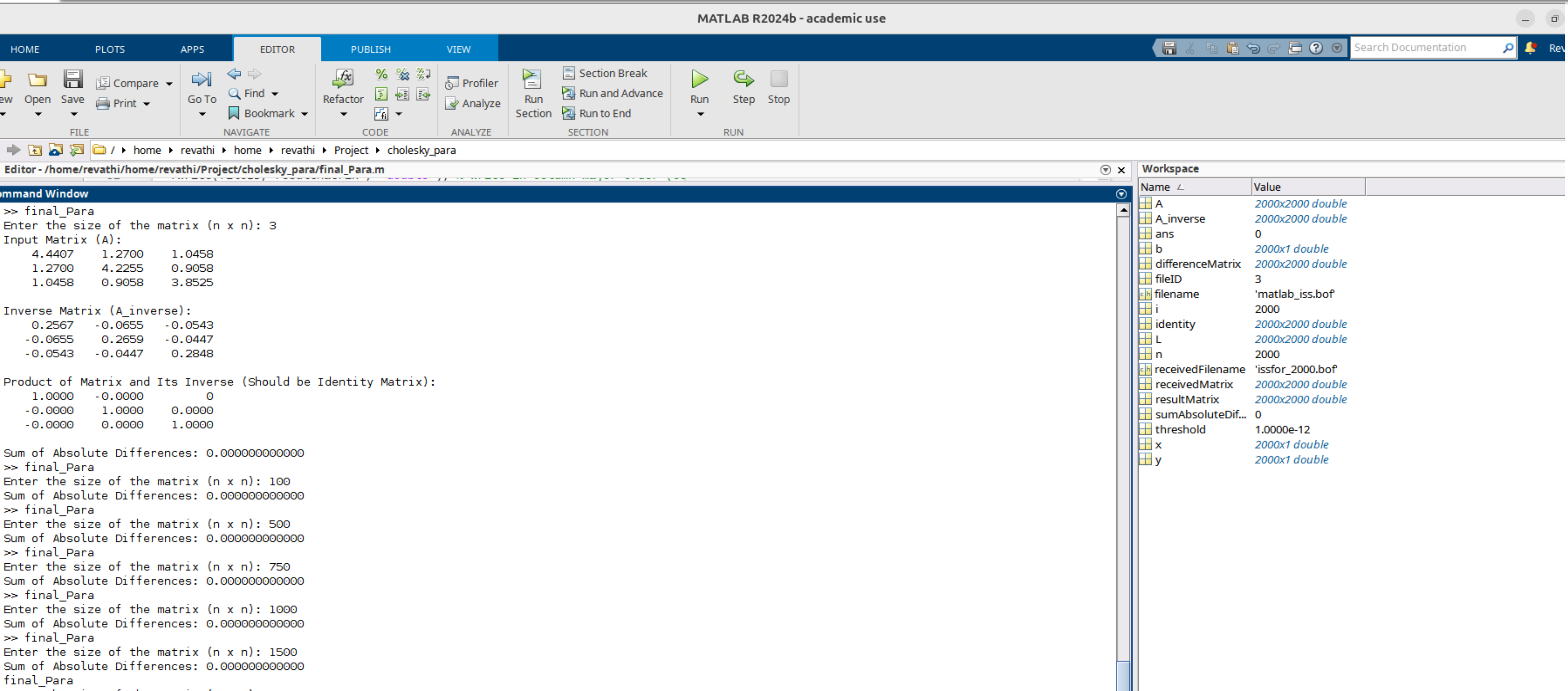
RESULT
CHOLESKY DECOMPOSITION MATRIX INVERSION- TERMINAL OUTPUT FOR SIZE 3

```
revathi@revathi-Latitude-3510: ~/home/revathi/Pro
rm -f tbb_for *.o core
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_para$ make all
g++ -O3 -Wall -std=c++11 -c tbb_fun.cpp -ltbb
g++ -O3 -Wall -std=c++11 tbb_for.cpp tbb_fun.o -ltbb -o tbb_for
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_para$ ./tbb_for
Enter the size of the matrix: 3
Input matrix (A):
              94               7               8
               7              64               6
               8               6              40
The inverse of the matrix A has been computed.
Elapsed time: 0 ms
Inverse matrix (A_inverse):
       0.0108853     -0.00100055     -0.00202698
     -0.00100055       0.0159398     -0.00219086
     -0.00202698     -0.00219086        0.025734
Identity matrix (A * A_inverse):
               1               0               0
               0               1               0
               0               0               1
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_para$ ./tbb_for
Enter the size of the matrix: 100
The inverse of the matrix A has been computed.
Elapsed time: 3 ms
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_para$ ./tbb_for
Enter the size of the matrix: 500
The inverse of the matrix A has been computed.
Elapsed time: 111 ms
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_para$ ./tbb_for
Enter the size of the matrix: 750
The inverse of the matrix A has been computed.
Elapsed time: 168 ms
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_para$ ./tbb_for
Enter the size of the matrix: 1000
The inverse of the matrix A has been computed.
Elapsed time: 330 ms
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_para$ ./tbb_for
Enter the size of the matrix: 1500
The inverse of the matrix A has been computed.
Elapsed time: 1094 ms
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_para$ ./tbb_for
Enter the size of the matrix: 2000
The inverse of the matrix A has been computed.
Elapsed time: 2675 ms
revathi@revathi-Latitude-3510:~/home/revathi/Project/cholesky_para$
```

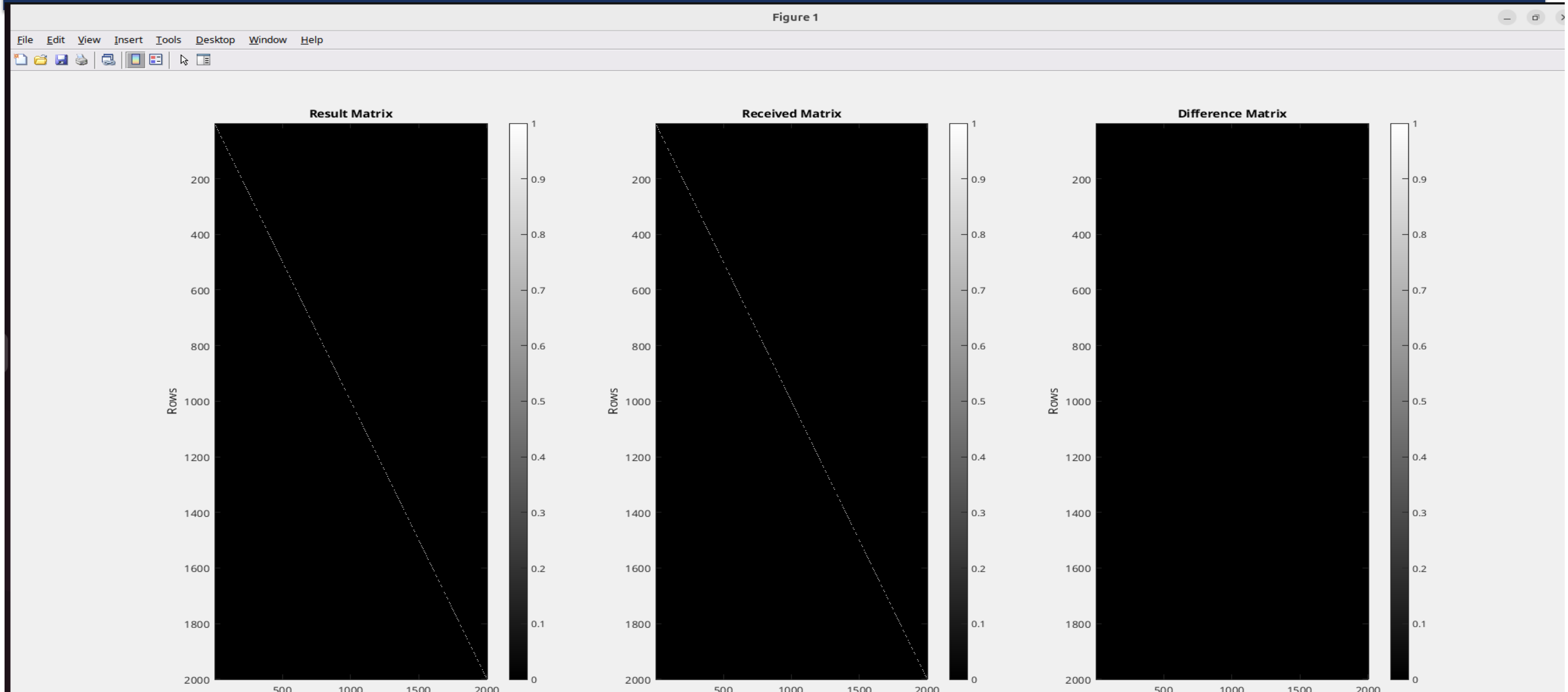
MATLAB RESULT
CHOLESKY DECOMPOSITION MATRIX INVERSION-SEQUENTIAL



MATLAB RESULT
CHOLESKY DECOMPOSITION MATRIX INVERSION-PARALLEL



MATLAB RESULT
LU DECOMPOSITION MATRIX INVERSION-PARALLEL



Advantages of using parallelization techniques

- Improved Performance
- Scalability
- Resource Utilization

Applications

- Machine Learning
- Computer Graphics
- Signal Processing

CONCLUSION

We conclude that our work shows how parallel programming is used to increase the productivity of matrix inversion operations. Significant processing time savings are achieved through parallelization, underscoring the advantages of concurrent execution over several threads. Our work highlights how parallel programming can revolutionize matrix inversion processing capabilities and how important it is to improve performance in challenging computational jobs.

REFERENCES

1) “ECE-5772-Lecture notes unit 4-Map Pattern”- https://moodle.oakland.edu/pluginfile.php /9501748/mod_resource/content/5/Notes% 20-%20Unit%204.pdf

2) "Matrix Row Operations," Khan Academy. Available: https://www.khanacademy.org/math/algebra-home/alg-matrices/alg-elementary-matrix-row-operations/a/matrix-row-operations

3) "Matrix Inversion and Eigenvalue," SRM Institute of Science and Technology. Available: https://webstor.srmist.edu.in/web_assets/srm_mainsite/files/2018 MatrixInversionandeigenvalue.pdf

4) Y. Zhang, "LU Decomposition," CAAM, Rice University, Fall 2009. Available: https://www.cmor-faculty.rice.edu~zhang/caam335/F09/handouts/lu.pdf

5)A. Ziefert, "Cholesky Decomposition," Matrix Algebra, Oct. 13, 2020. [Online].Available: https://zief0002.github.io/matrix-algebra/cholesky-decompostion.html

6) "Triangular matrix," Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/Triangular_matrix. [Accessed: Nov. 14, 2024].

DONE BY

RAMYA RAJARAMAN

REVATHY SEKAR

UNDER GUIDANCE OF

PROF.DANIEL LLAMOCCA

DEMO

THANK YOU

ANY QUESTIONS?