B.E (Information Technology) Final Year Project
Report on

Study of Time Taken By Different Matrix Multiplication Approach For Different value N of N Square Matrix

By

BITTU GOND (001611001013)

&

NITYANANDA DEBSHARMA (001611001033)

Under The Guidance Of

Prof. UTPAL KUMAR RAY

Department of Information Technology
Faculty of Engineering and Technology
Jadavpur University
Kolkata, India

2016-2020

Declaration by the Students

We hereby declare that this project report entitled "Study of time taken by different matrix multiplication approach for different value of n of n square matrix" contains only the work completed by us as a part of the Bachelor of Engineering course, during this year 2019-2020, under the supervision of Prof. Utpal Kumar Ray, Department of Information Technology, Jadavpur University.

All information, materials and methods that are not original to this work have been properly referenced and cited. All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.

We also declare that no part of this project work has been submitted for the award of any other degree prior to this date.

Signature: Bittu Gond

Nityananda Debsharma

Date: 12/6/2020

Certificate

This is to certify that the project entitled “Study of time taken by different matrix multiplication approach for different value of n of n square matrix” was carried out by Bittu Gond (001611001013) and Nityananda Debsharma (001611001033) and submitted to the Jadavpur University during the year 2019-2020 for the award of the degree of Bachelor’s of Engineering, is a bona fide record of work done by them under my supervision. The contents of this project, in full or in parts, have not been submitted to any other Institute or University for the award of any other degree.

Signature of the supervisor:

Date:

Acknowledgement

We would like to express our deep and sincere gratitude to our respected Prof. Utpal Kumar Ray of the Information Technology Department, Jadavpur University for giving us this opportunity to work on an equally challenging and informative project and providing his continuous support, encouragement, and valuable guidance throughout the project.

Furthermore, we would also like to thank Mr. Rayan Ray and our classmates who helped us in our hours of need.

Thank you.

Table of Contents

Introduction

Matrix multiplication: The product of two matrix a and b (a*b) is Only possible if number of columns of First matrix ‘a’ is equal to number of rows of second matrix ‘b’. Let ‘a’ be a matrix of size m*n and ‘b’ be a matrix of size n*o then the product a*b will produce ‘c’ matrix of size m*o.

Such that

cij=ai1*b1j+ai2*b2j+ai3*b3j..................+ain*bnj

For i=1, 2, 3....m and j=1, 2, 3.......o

Example:

$$A=\begin{pmatrix} 3, 1 \\ 0, 3 \end{pmatrix} \qquad B=\begin{pmatrix} 0, 2 \\ 1, 0 \end{pmatrix}$$

$$C=\begin{pmatrix} 3*0+1*1,\ 3*2+1*0 \\ 0*0+3*1,\ 0*2+3*0 \end{pmatrix} = \begin{pmatrix} 1, 6 \\ 3, 0 \end{pmatrix}$$

Applications of Matrix Multiplication

- In graph most of the data is represented in the form of matrix. Matrix multiplication is one of the most important operation on matrix. Lots of matrix-graph based algorithms are based on matrix multiplication.egg Transitive closure

- It is used in network theory.

- It is used in solution of linear system of equation.

- It is used in transformation of coordinate system.

- It is used in statistics and linear algebra.

- It is used in geology.

- It is used in robotics, AI, Machine learning.

- It is used in cryptography .For example it is used in encryption and decryption in Hill Cipher and other techniques.

Why lots of effort and time is invested in finding New approach of matrix multiplication by researchers

In today's world data is becoming bigger and bigger.
Many of the data are being stored in matrix format.

Matrix multiplication is one of the important operation
So it is being used in lots of algorithms of graph,
Networking, AI, machine learning etc.

Time complexity of matrix multiplication is O (n^3).
Since size of date is increasing day by day operation
Execution time increasing drastically thus the performance
Of algorithms using matrix multiplication becoming poor.

To tackle this problem researchers are trying their best to
Minimise time complexity of matrix multiplication from O (n^3)
Or reduce the execution time by parallel and distributed
algorithms.

Pre-Requisites

(Overview of required pre requisites are explained below.)

- C
- OpenMpi should be installed.

Some pre requisites concepts are explained henceforth.

Mpi

What is Mpi?

Mpi is a library of routines which can be used to create parallel programs in c. It runs with standard c programs, using Commonly-available operating system services to create parallel processes and exchange information among these processes. It can also support distributed program execution on heterogeneous hardware.

Basic process of running program in c using mpi is shown below

```
addr:192.168.43.204>mpicc matrixMultiplication4.c -o 4
addr:192.168.43.204>mpirun 4
value of n =2
matrix a:
3   2
3   1
matrix b:
1   0
0   2
resultant matrix:
3   4
3   2
time taken by matrix multiplication using mpi=0.000109 sec
addr:192.168.43.204>
```

Objective of project

- Implement matrix multiplication of two n square matrix using different approaches.
- Create n square matrix using dynamic memory allocation
- Use rand ( ) function to input values to matrix form 0 to 4
- Use gettimeofday ( ) function to calculate execution of matrix multiplication.
- Collect data of execution time for different value of n.

Normal Matrix Multiplication

- Use scan ( ) function to ask for size of n.
- Create matrix a, b, c dynamically using format of instruction given below

```
int **a=(int **)malloc(n*sizeof(int *));
for(int i=0;i<n;i++)
a[i]=(int *)malloc(n*sizeof(int));
```

- Assign value to matrix a and b using rand( )%5
- Create start and end variable of type struct timeval. Get starting time in start before calling matrix multiplication function and collect time after execution of function in end by using gettimeofday (). From this time calculate execution time. Put it in a variable created to store execution time. Print this time on monitor screen.
- We made a condition of printing a, b, c matrix on screen when value of n is <=4 to verify function is working correctly or not.
- Pseudo code of normal matrix multiplication function

```
For i from 0 to n-1
        For j from 0 to n-1
                c[i] [j] =0;
                For k from 0 to n-1
                        c[i][j] =c[i][j] +a[i][k]*b[k][j]
```

Time complexity of this approach is O [n^3].

Strassen's Algorithm

- Matrix multiplication using normal divide and conquer approach takes T (n) =8T (n/2) + O (n^2) .By master method this time came out to be O (n^3). Strassen's algorithm reduced this 8T (n/2) to 7T (n/2) by decreasing number of matrix multiplication of sub-matrices 8 to 7 by incorporation of more addition and subtraction of in a manner so that it does not affect the matrix multiplication result. Since sub-matrices are divided in ½ in each recursion so in order to divide it correctly value of n should be power of 2.

- We used scan ( ) function to ask for size of n.

- We checked value of n is power of 2 or not by using bit wise "and" operation on n, n-1.if (n&n-1!=0) then print n is not power of 2 after that exited the program .

- If bit wise 'and' operation on n was 0.then

- Create matrix a, b, c dynamically using format of instruction given below

```
int **a=(int **)malloc(n*sizeof(int *));
for(int i=0;i<n;i++)
a[i]=(int *)malloc(n*sizeof(int));
```

  - Assign value to matrix a and b using rand( )%5

  - Create start and end variable of type struct timeval. Get starting time in start before calling matrix multiplication function and collect time after execution of function in end by using gettimeofday (). From this time calculate execution time. Put it in a variable created to store execution time. Print this time on monitor screen.

  - We made a condition of printing a, b, c matrix on screen when value of n is <=4 to verify function is working correctly or not.

  - Description of matrix multiplication function of Strassen's algorithm

- Multiply function contain 4 arguments a, b, c matrix and size=n.
- Check if value of n = 1 then c[0][0] =a[0][0]*b[0][0] then return .
- Else

- Divide matrix a virtually into 4 (n/2) ^2 matrix a11, a12, a21, a22. Similarly divide b virtually into b11, b12, b21, b22.we did this by creating 8 (n/2) ^2 matrices dynamically and assigning values to them from a and b matrix from their respective position.

- Dynamically create 7 (n/2) ^2 matrix for 7 matrix multiplication m, m2, m3, m4, m5, m6, m7.

- Dynamically create temp1 (n/2) ^2 matrix. Then call add function to add a11 and a22 and store sum in temp1.This add function take 4 arguments 3 matrix and size of matrix. This function adds 2 matrix and store in 3rd matrix.

- Dynamically create temp2 (n/2) ^2 matrix. Then call add function with b11, b22, temp2 and n/2.

- Then recursively call multiply function with temp1, temp2, m1 and n/2.

- Free temp1 and temp2.

- Dynamically create temp3 (n/2) ^2 matrix. Then call add function with a21, a22, temp3 and n/2.

- Then recursively call multiply function with temp3, b11, m2 and n/2.

- Free temp3.

- Dynamically create temp4 (n/2) ^2 matrix. Then call subtract function with b12, b22, temp4 and n/2. This subtract function subtract 2nd matrix from 1st matrix and store it in 3rd matrix.

- Then recursively call multiply function with a11, temp4, m3 and n/2.

- Free temp4.

- Dynamically create temp5 (n/2) ^2 matrix. Then call subtract function with b21, b11, temp5 and n/2.

- Then recursively call multiply function with a22, temp5, m4 and n/2.

- Free temp5.

- Dynamically create temp6 (n/2) ^2 matrix. Then call add function with a11, a12, temp6 and n/2.

- Then recursively call multiply function with temp6, b22, m5 and n/2.

- Free temp6.

- Dynamically create temp7 (n/2) ^2 matrix. Then call subtract function with a21, a11, temp7 and n/2.
-
- Dynamically create temp8 (n/2) ^2 matrix. Then call add function with b11, b12, temp8 and n/2.

- Then recursively call multiply function with temp7, temp8, m6 and n/2.

- Free temp7 and temp8.

- Dynamically create temp9 (n/2) ^2 matrix. Then call subtract function with a12, a22, temp9 and n/2.
-
- Dynamically create temp10 (n/2) ^2 matrix. Then call add function with b21, b22, temp10 and n/2.

- Then recursively call multiply function with temp9, temp10, m7 and n/2.

- Free temp9 and temp10.

- Now to assign value in c matrix. Virtually c matrix in c11, c22, c33, c44 each with size (n/2) ^2. i.e.
  c11 is matrix portion of c[i][j] where i= 0 to n/2-1, j = 0 to n/2-1.
  c12 is matrix portion c[i][j] where i=0 to n/2-1, j=n/2 to n-1.
  c21 is matrix portion c[i][j] where i=n/2 to n-1, j=0 to n/2-1.

c22 is matrix portion c[i][j] where i=n/2 to n-1, j=n/2 to n-1.

- In 2 for loops assign value to c in following way:
- For c11 portion
  c[i][j] = m1[i][j]+m4[i][j]-m5[i][j]+m7[i][j];

- For c12 portion
  c[i][j] = m3[i][j-n/2]+m5[i][j-n/2];

- For c21 portion
  c[i][j] = m2[i-n/2][j]+m4[i-n/2][j];

- For c22 portion

c[i][j] =m1[i-n/2][j-n/2]-m2[i-n/2][j-n/2]+m3[i-n/2][j-n/2]+m6[i-n/2][j-n/2];

Time complexity of this approach is O (nlog7) which
Is approximately O (n^2.8074).

Matrix multiplication using threads

- Globally create an offset value to for threads to work on correct partition.
- Use scan ( ) function to ask for size of n.
- Create matrix a, b, c dynamically using format of instruction given below

```
int **a=(int **)malloc(n*sizeof(int *));
for(int i=0;i<n;i++)
a[i]=(int *)malloc(n*sizeof(int));
```

- Assign value to matrix a and b using rand( )%5
- Create array of 4 threads.
- Create start and end variable of type struct timeval.
- Get starting time in start by using gettimeofday (&start, NULL);
- In for loop i=0 to 3

Run pthread_create(&threads[i],NULL,matrixmultiplication,(void *) p);
In this function we called matrixmultiplication function and passed value of n as argument .It also contain thread id.

- Run pthread_join ( ) function all 4 threads. It blocks the calling of thread until the thread with identifier equal to 1st argument terminates.
- Get time in end by using gettimeofday(&end, NULL); function. From this time calculate execution time. Put it in a variable created to store execution time. Print this time on monitor screen.
- We made a condition of printing a, b, c matrix on screen when value of n is <=4 to verify function is working correctly or not.

- Description of matrix multiplication function

- This function divide 1st matrix into 4 parts of size N/4*N
- Each thread multiplies one of the part 1st matrix with 2nd matrix in normal method concurrently by using a globally declared offset value such that each thread independent parts and none of them interfere with each others

Here matrix multiplication is done in parallel manner it increases the throughput.

Matrix multiplication using Mpi

- Mpi creates multiple instances of program. These instances are separate processes, each process has its own memory space. Each process has its own copy of every variable, including global variable because of that in this approach we were not able to create a, b, c matrices dynamically.
  So we globally declared size of n and a, b, c .
- Create global variable MPI_Status status; to store status of send and receive.
- Call MPI_Init(&argc, &argv); to initialize mpi computation.
- Call MPI_Comm_rank(MPI_COMM_WORLD,&pid); to get process id in pid.
- Call MPI_Comm_size(MPI_COMM_WORLD,&np); to get number of process in np.
- In process 0
  - We initialise a and b matrix with the help of rand ( )%5;
  - We get the time before send process start in start variable of type timeval struct
  - We sent equal rectangular portion of 1st matrix (&a[p][0]) of size rows*n and complete b matrix to each remaining processes to do normal matrix multiplication by using MPI_Send
  - ( ) function. Then increment p+=row where row=n/(np-1).

- Format of MPI_Send:

  MPI_Send(&p, 1, MPI_INT, i, 1, MPI_COMM_WORLD);
  1st argument = address of data to be sent
  2nd argument = number of data to be sent
  3rd argument =data type of data to be sent in of mpi data set
  4th argument = destination id
  5th argument =message tag
  6th argument =communicator

- Wait for result from each process and collect result of multiplication by using MPI_Recv ( ) function
- Format of MPI_Recv :

MPI_Recv(&c[p][0], rows*n, MPI_INT, i, 2, MPI_COMM_WORLD, &status);
1st argument = address of data to be received
2nd argument = number of data to be expected
3rd argument =data type of data to be received in of mpi data set
4th argument = source id
5th argument =message tag
6th argument =communicator
7th argument=status struct

  - Get time after all results from other processes is received in end variable of type timeval struct.
  - Calculate execution time from start and end and print on screen.

  - We made a condition of printing a, b, c matrix on screen when value of n is <=4 to verify function is working correctly or not.

- In pid >0
  - We receive data from to process o through MPI_Recv and did matrix multiplication. Then sent it back to process 0.
- Then call MPI_Finalize(); then return

Input/output of normal matrix multiplication

1. 1st input n=2.

Output:-

```
addr:192.168.43.204>gcc matrixMultiplication1.c -o 1
addr:192.168.43.204>./1
enter value of n for n^2 matrix a and b=2
matrix a:
 3 2
 3 1
matrix b:
 1 0
 0 2
done
time taken by odanary method of matrix multiplication=0.000003 sec
matrix c:
 3 4
 3 2
addr:192.168.43.204>
```

So at n=2, execution time=0.000003 sec

2. 2nd input n=4.

   Output:-

```
addr:192.168.43.204>gcc matrixMultiplication1.c -o 1
addr:192.168.43.204>./1
enter value of n for n^2 matrix a and b=4
matrix a:
 3 2 3 1
 4 2 0 3
 0 2 1 2
 2 2 2 4
matrix b:
 1 0 0 2
 1 2 4 1
 1 1 3 4
 0 3 0 2
done
time taken by odanary method of matrix multiplication=0.000003 sec
matrix c:
 8 10 17 22
 6 13 8 16
 3 11 11 10
 6 18 14 22
addr:192.168.43.204>
```

So at n=4, execution time= 0.000003 sec

3. 3rd input n=16

   Output:-

```
addr:192.168.43.204>gcc matrixMultiplication1.c -o 1
addr:192.168.43.204>./1
enter value of n for n^2 matrix a and b=16
done
time taken by odanary method of matrix multiplication=0.000134 sec
addr:192.168.43.204>
```

   So at n=16, execution time=0.000134 sec

4. 4th input n=256

   Output:-

```
addr:192.168.43.204>gcc matrixMultiplication1.c  -o 1
addr:192.168.43.204>./1
enter value of n for n^2 matrix a and b=256
done
time taken by odanary method of matrix multiplication=0.168110 sec
addr:192.168.43.204>
```

   So at n=256, execution time=0.168110 sec

5. 5th input n=512

   Output:-

```
addr:192.168.43.204>gcc matrixMultiplication1.c  -o 1
addr:192.168.43.204>./1
enter value of n for n^2 matrix a and b=512
done
time taken by odanary method of matrix multiplication=1.002248 sec
addr:192.168.43.204>
```

   So at n=512, execution time=1.002248 sec

6. 6th input n=1024

   Output:-

```
addr:192.168.43.204>gcc matrixMultiplication1.c  -o 1
addr:192.168.43.204>./1
enter value of n for n^2 matrix a and b=1024
done
time taken by odanary method of matrix multiplication=15.214676 sec
addr:192.168.43.204>
```

   So at n=1024, execution time=15.214676 sec

7. 7th input n=2048

   Output:-

```
addr:192.168.43.204>gcc matrixMultiplication1.c  -o 1
addr:192.168.43.204>./1
enter value of n for n^2 matrix a and b=2048
done
time taken by odanary method of matrix multiplication=123.969043 sec
addr:192.168.43.204>
```

   So at n=2048, execution time=123.969043 sec

8. 8th input n=4096

   Output:-

```
addr:192.168.43.204>gcc matrixMultiplication1.c  -o 1
addr:192.168.43.204>./1
enter value of n for n^2 matrix a and b=4096
done
time taken by odanary method of matrix multiplication=842.484933 sec
addr:192.168.43.204>
```

   So at n=4096, execution time=842.484933 sec

Input/output of Strassen’s algorithm

1. 1st input n=2

   Output:-

```
>gcc matrixMultiplication3.c -o 3
>./3
Enter n of n sqare  matrix which is power of 2 =2
matrix a:
 3 2
 3 1
matrix b:
 1 0
 0 2
done
time taken by strassen's algorithm for matrix multiplication=0.000026 sec
matrix c:
 3 4
 3 2
>
```

So at n=2, execution time =0.000026 sec

2. 2nd input n=4

   Output:-

```
>gcc matrixMultiplication3.c -o 3
>./3
Enter n of n sqare  matrix which is power of 2 =4
matrix a:
 3 2 3 1
 4 2 0 3
 0 2 1 2
 2 2 2 4
matrix b:
 1 0 0 2
 1 2 4 1
 1 1 3 4
 0 3 0 2
done
time taken by strassen's algorithm for matrix multiplication=0.000139 sec
matrix c:
 8 10 17 22
 6 13 8 16
 3 11 11 10
 6 18 14 22
>
```

So at n=4, execution time=0.000139 sec

3. 3rd input n=16

   Output:-

```
>gcc matrixMultiplication3.c -o 3
>./3
Enter n of n sqare  matrix which is power of 2 =16
done
time taken by strassen's algorithm for matrix multiplication=0.004274 sec
>
```

   So at n=16, execution time=0.004274 sec

4. 4th input n=256

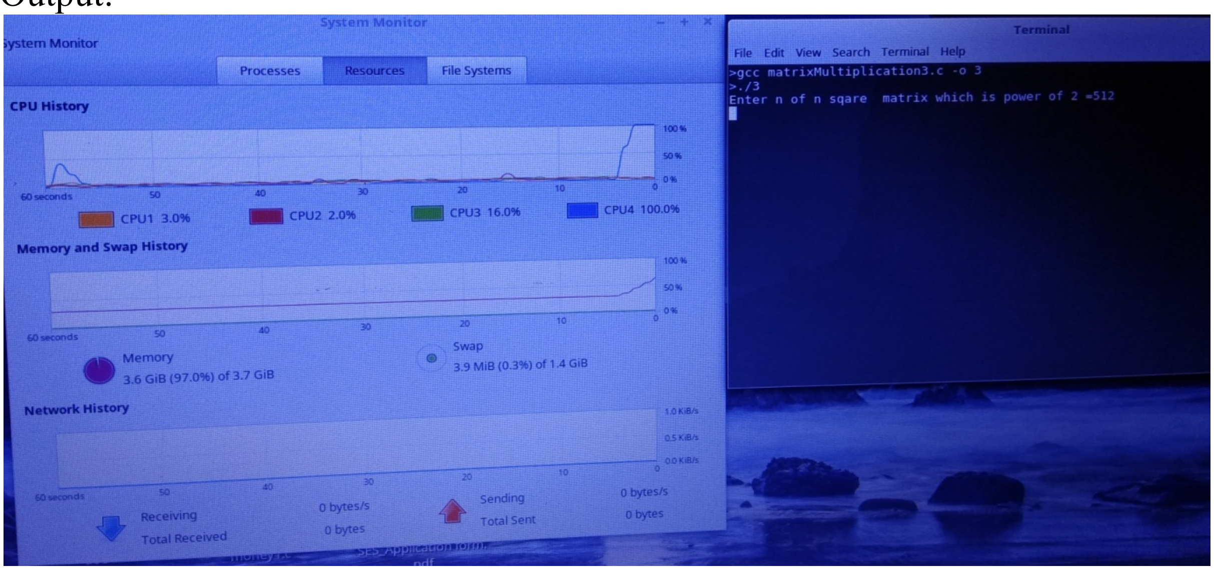
   Output:-

```
>gcc matrixMultiplication3.c -o 3
>./3
Enter n of n sqare  matrix which is power of 2 =256
done
time taken by strassen's algorithm for matrix multiplication=2.584046 sec
>
```

   So at n=256, execution time=2.584046 sec

5. 5th input n=512

   Output:-



   So at n=512 memory consumption becomes 100% and system start hanging. Which shows Strassen's algorithm consume lots of memory.

Input/output of matrix multiplication using threads

1. 1st input n=2.

   Output:-

```
addr:192.168.43.204>gcc -pthread matrixMultiplication2a.c -o 2
addr:192.168.43.204>./2
enter value of n for n^2 matrix a and b=2
matrix a:
 3 2
 3 1
matrix b:
 1 0
 0 2
done
time taken for  matrix multiplication using  n square thread =0.000857 sec
matrix c:
 3 4
 3 2
addr:192.168.43.204>
```

   So at n=2, execution time=0.000857 sec

2. 2nd input n=4.

   Output:-

```
addr:192.168.43.204>gcc -pthread matrixMultiplication2a.c -o 2
addr:192.168.43.204>./2
enter value of n for n^2 matrix a and b=4
matrix a:
 3 2 3 1
 4 2 0 3
 0 2 1 2
 2 2 2 4
matrix b:
 1 0 0 2
 1 2 4 1
 1 1 3 4
 0 3 0 2
done
time taken for  matrix multiplication using  n square thread =0.000440 sec
matrix c:
 8 10 17 22
 6 13 8 16
 3 11 11 10
 6 18 14 22
addr:192.168.43.204>
```

   So at n=4, execution time=0.000440 sec

3. 3^{rd} input n=16

Output:-

```
addr:192.168.43.204>gcc -pthread matrixMultiplication2a.c -o 2
addr:192.168.43.204>./2
enter value of n for n^2 matrix a and b=16
done
time taken for  matrix multiplication using  n square thread =0.000618 sec
addr:192.168.43.204>
```

So at n=16, execution time=0.000618 sec

4. 4^{th} input n=256

Output:-

```
addr:192.168.43.204>gcc -pthread matrixMultiplication2a.c -o 2
addr:192.168.43.204>./2
enter value of n for n^2 matrix a and b=256
done
time taken for  matrix multiplication using  n square thread =0.099006 sec
addr:192.168.43.204>
```

So at n=256, execution time=0.099006 sec

5. 5^{th} input n=512

Output:-

```
addr:192.168.43.204>gcc -pthread matrixMultiplication2a.c -o 2
addr:192.168.43.204>./2
enter value of n for n^2 matrix a and b=512
done
time taken for  matrix multiplication using  n square thread =0.502232 sec
addr:192.168.43.204>
```

So at n=512, execution time=0.502232 sec

6. 6th input n=1024

Output:-

```
addr:192.168.43.204>gcc -pthread matrixMultiplication2a.c -o 2
addr:192.168.43.204>./2
enter value of n for n^2 matrix a and b=1024
done
time taken for  matrix multiplication using  n square thread =5.958136 sec
addr:192.168.43.204>
```

So at n=1024, execution time=5.958136 sec

7. 7th input n=2048

Output:-

```
addr:192.168.43.204>gcc -pthread matrixMultiplication2a.c -o 2
addr:192.168.43.204>./2
enter value of n for n^2 matrix a and b=2048
done
time taken for  matrix multiplication using  n square thread =51.464284 sec
addr:192.168.43.204>
```

So at n=2048, execution time=51.464284 sec

8. 8th input n=4096

Output:-

```
addr:192.168.43.204>gcc -pthread matrixMultiplication2a.c -o 2
addr:192.168.43.204>./2
enter value of n for n^2 matrix a and b=4096
done
time taken for  matrix multiplication using  n square thread =528.827028 sec
addr:192.168.43.204>
```

So at n=4096, execution time=528.827028 sec

Input/output of matrix multiplication using mpi

1. 1st input n=2.

   Output:-

```
addr:192.168.43.204>mpicc matrixMultiplication4.c -o 4
addr:192.168.43.204>mpirun 4
value of n =2
matrix a:
3   2
3   1
matrix b:
1   0
0   2
resultant matrix:
3   4
3   2
time taken by matrix multiplication using mpi=0.000109 sec
addr:192.168.43.204>
```

   So at n=2, execution time=0.000109 sec

2. 2nd input n=4.

   Output:-

```
addr:192.168.43.204>mpicc matrixMultiplication4.c -o 4
addr:192.168.43.204>mpirun 4
value of n =4
matrix a:
3   2   3   1
4   2   0   3
0   2   1   2
2   2   2   4
matrix b:
1   0   0   2
1   2   4   1
1   1   3   4
0   3   0   2
resultant matrix:
8   10   17   22
6   13   8   16
3   11   11   10
6   18   14   22
time taken by matrix multiplication using mpi=0.000063 sec
addr:192.168.43.204>
```

   So at n=4, execution time=0.000063 sec

3. 3rd input n=16

Output:-

```
addr:192.168.43.204>mpicc matrixMultiplication4.c -o 4
addr:192.168.43.204>mpirun 4
value of n =16
time taken by matrix multiplication using mpi=0.000185 sec
addr:192.168.43.204>
```

So at n=16, execution time=0.000185 sec

4. 4th input n=256

Output:-

```
addr:192.168.43.204>mpicc matrixMultiplication4.c -o 4
addr:192.168.43.204>mpirun 4
value of n =256
time taken by matrix multiplication using mpi=0.096881 sec
addr:192.168.43.204>
```

So at n=256, execution time=0.096881 sec

5. 5th input n=512

Output:-

```
addr:192.168.43.204>mpicc matrixMultiplication4.c -o 4
addr:192.168.43.204>mpirun 4
value of n =512
time taken by matrix multiplication using mpi=0.918421 sec
addr:192.168.43.204>
```

So at n=512, execution time=0.918421 sec

6. 6^{th} input n=1024

Output:-

```
addr:192.168.43.204>mpicc matrixMultiplication4.c -o 4
addr:192.168.43.204>mpirun 4
value of n =1024
time taken by matrix multiplication using mpi=18.204945 sec
addr:192.168.43.204>
```

So at n=1024, execution time=18.204945 sec

7. 7^{th} input n=2048

Output:-

```
addr:192.168.43.204>mpicc matrixMultiplication4.c -o 4
addr:192.168.43.204>mpirun 4
value of n =2048
time taken by matrix multiplication using mpi=176.621725 sec
addr:192.168.43.204>
```

So at n=2048, execution time=176.621725 sec

8. 8^{th} input n=4096

Output:-

```
addr:192.168.43.204>mpicc matrixMultiplication4.c -o 4
addr:192.168.43.204>mpirun 4
value of n =4096
time taken by matrix multiplication using mpi=1447.589081 sec
addr:192.168.43.204>
```

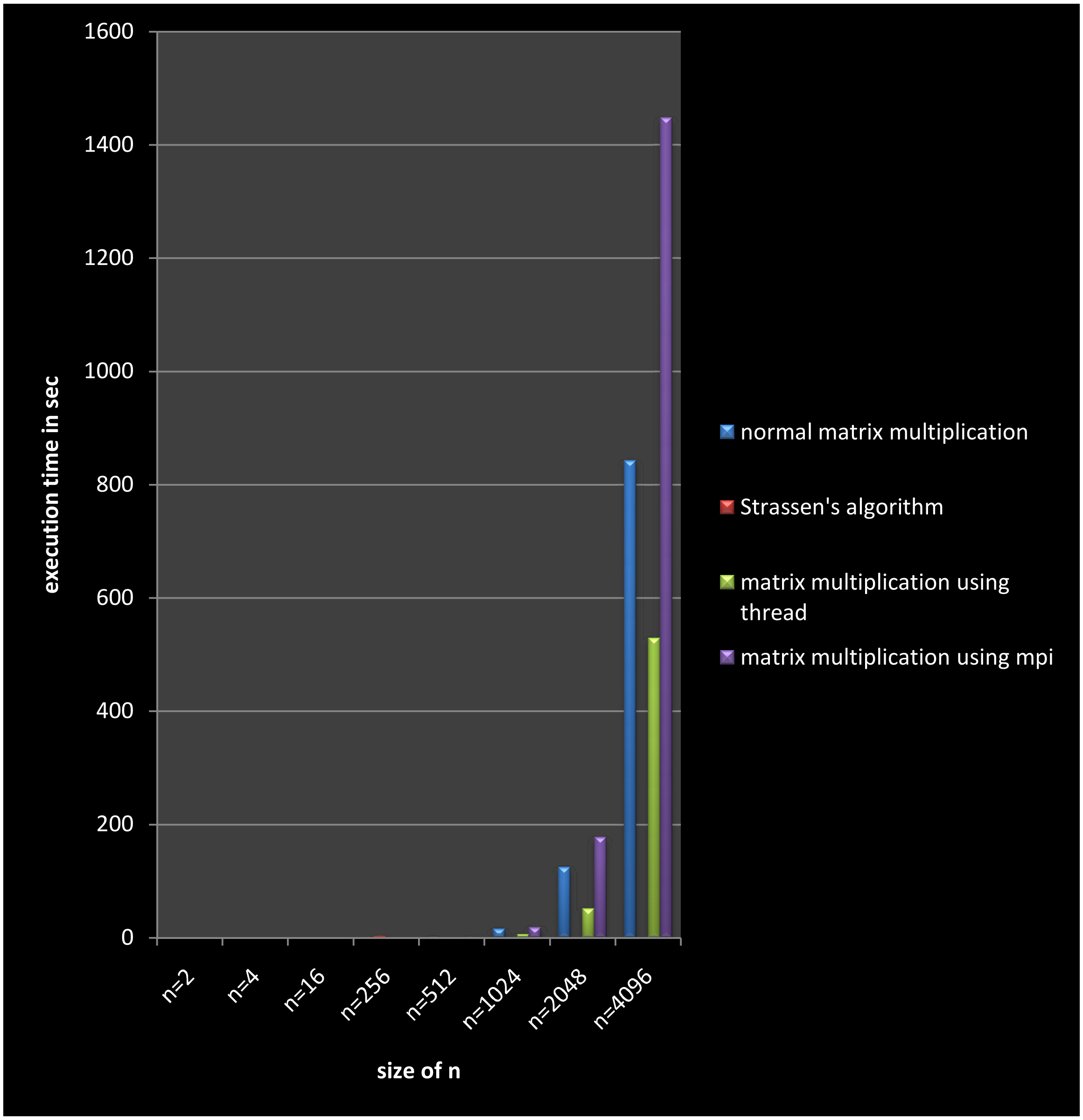
So at n=4096, execution time=1447.589281 sec

Data table

Date of execution time of matrix multiplication in all 4 approaches.

Value of n	Execution time in sec in normal matrix multiplication	Execution time in sec in Strassen's Algorithm	Execution time in sec in matrix multiplication using thread	Execution time in sec in matrix multiplication using mpi
2	0.000003	0.000026	0.000857	0.000109
4	0.000003	0.000139	0.000440	0.000063
16	0.000134	0.004274	0.000618	0.000185
256	0.168110	2.584046	0.099006	0.096881
512	1.002248		0.502232	0.918421
1024	15.214676		5.958136	18.204945
2048	123.969043		51.464284	176.621725
4096	842.484933		528.827028	1447.589281

Chart representations of data



Conclusion

Execution time may vary from system to system. Depending on system conditions while taking the Output data may vary but according to result on this system we can conclude following points:-

- Strassen's algorithm needs lots memory and also throughput is lower than remaining approaches for value of n <=256. Due to it is rarely used.

- Matrix multiplication using thread which does multiplication in parallel manner works better than normal and multiplication using mpi .we can't say about its performance is better than Strassen's algorithm or not we didn't got Strassen's output after n=256.

- Mpi used process which is high weighted and Thread is light weighted in this sense thread is better than mpi.

- We had lack of resources to go for distributed approach using mpi otherwise mpi might have given better performance than thread.

- Performance comparison: 1st thread, 2nd normal, 3rd mpi and Strassen's might perform better then all 3 if we could get more memory.

Reference

- https://en.m.wikipedia.org/wiki/Matrix_multiplication_algorithm
- https://www.geeksforgeeks.org/dynamically-allocate-2d-array-c/
- https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/
- https://www.geeksforgeeks.org/multithreading-c-2/
- https://en.m.wikipedia.org/wiki/Strassen_algorithm?wprov=sfla1
- http://condor.cc.ku.edu/~grobe/docs/intro-MPI-C.shtml

System info used for project