Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

Parallel Processing: Assignment 4

Objective:

Computers usually solve system of Linear Equations using LU Decomposition as explained in Chapter 8 of Introduction to Parallel Processing. It is also a key step in computing determinant of the matrix. LU Decomposition can be viewed as the matrix form of Gaussian Elimination.

The goal of this assignment is to find the determinant of a matrix A of size n*n after performing LU Decomposition of the matrix.

Det(A) = det(L)det(U) = 1. Det(U) = $\prod_{i=1}^{n} u_{ii}$

To solve the above problem, below are the details for the formulation along with the MPI calls used.

Commonly used MPI Calls:

MPI Calls	Description
int MPI_Init(int *argc, char ***argv)	used to Initialize the MPI execution environment
Int MPI_Comm_size(MPI_Comm comm, int *size)	Determines the size of the group associated with a communicator
int MPI_Comm_rank(MPI_Comm comm, int *rank)	Determines the rank of the calling process in the communicator
double MPI_Wtime()	Returns an elapsed time on the calling processor.
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)	Blocking receive for a message
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)	Broadcasts a message from the process with rank "root" to all other processes of the communicator
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)	Performs a blocking send
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)	Blocking receive for a message
int MPI_Barrier(MPI_Comm comm)	Blocks until all processes in the communicator have reached this routine.
int MPI_Finalize()	Terminates MPI execution environment

Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

LU Decomposition:

The LU Decompositon algorithm decomposes the matrix A of size n*n into a lower triangular matrix L and upper triangular matrix U.

A recursive way to calculate the upper triangle matrix $U = A^n$ can be given by:

1. Define $A^{(1)} = a_{i,j}^1 \coloneqq A$
2. Calculate for k = 1,2,…n-1 values $l_{i,k}, i = k+1, \ldots., n$ and the matrices $A^{(k+1)} = \left(a_{i,j}^{(k+1)}\right)$iteratively with:

$$l_{i,k} \coloneqq \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}$$

$$a_{i,j}^{(k+1)} \coloneqq \begin{cases} a_{i,j}^{(k)} - l_{i,k}a_{k,j}^{k} \\ a_{i,j}^{k} \qquad otherwise \end{cases} for\ i\ \epsilon\ \{k+1, \ldots, n\}, j\ \epsilon\{k, \ldots, n\}$$

A serial implementation of this iterative algorithm can have the following form:

for k = 1 to n -1:

i= k+1 to n

$l_{i,k} \coloneqq \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}$

for j = k to n:

$a_{i,j} \coloneqq a_{i,j} - l_{i,k}a_{k,j}$

Parallel Implementation:

Gaussian Elimination Algorithm was implemented that uses the block wise decomposition in parallel. Different processes work on different blocks of the matrices, thus parallelizing the work. In Gaussian Elimination the L and U matrices are computed separately. Cyclic distribution was used to accomplish LU decomposition. Each node is responsible for computing on the block assigned to it and post the computation it broadcasts the results to the rest of the nodes in the system. Thus, in essence, matrix A

is scattered among all processes by dividing it into blocks. Each node computes $l_{i,k}$ of column $L_{:,1}$. Broadcasts the column and then updates all entries on all nodes. After LU Decomposition, determinant of the matrix is calculated.

Parameter Ranges:

To test the run times for both serial and parallel formulations, we have considered the following ranges for the 3 parameters of matrix size, and the number of processors.

Matrix size: From 32 x 32 to 1024 x 1024

Number of processors: 4 to 64

Results:

Matrix Size	Number of Processes	Serial Time (Ts)	Parallel time	Communication time	Speedup	Efficiency
32 * 32	4	0.028241	0.011976	0.001994	2.3581	0.5895
64 * 64	16	0.047434	0.034805	0.005091	1.3628	0.0851
128 * 128	16	0.145151	0.133734	0.005232	1.0581	0.0661
256 * 256	64	0.567519	0.430462	0.065949	1.3183	0.0205
512 * 512	64	2.270888	1.099640	0.066834	2.066	0.0322
1024 * 1024	64	9.468472	3.118609	0.699093	3.0445	0.0475

Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

Graphs:

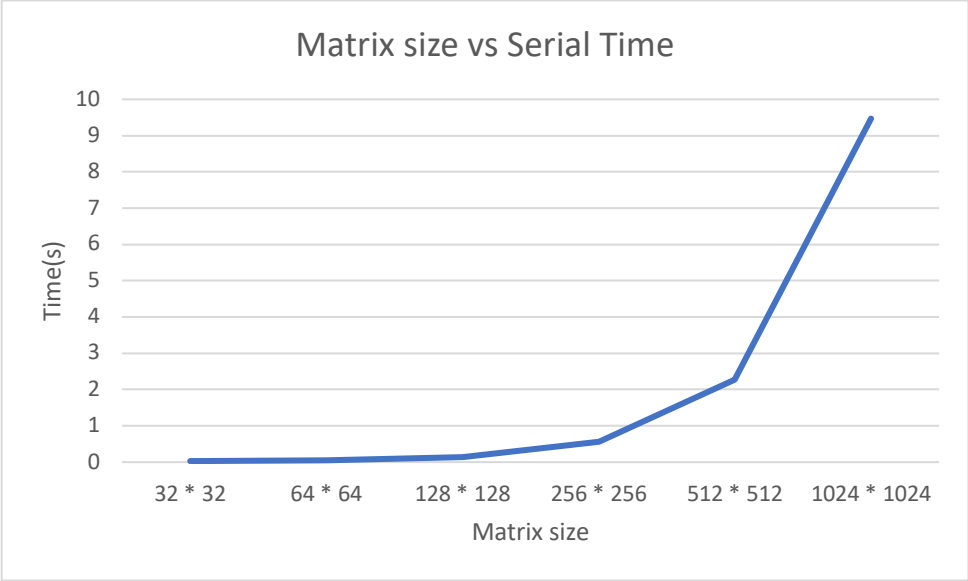

FIGURE 1

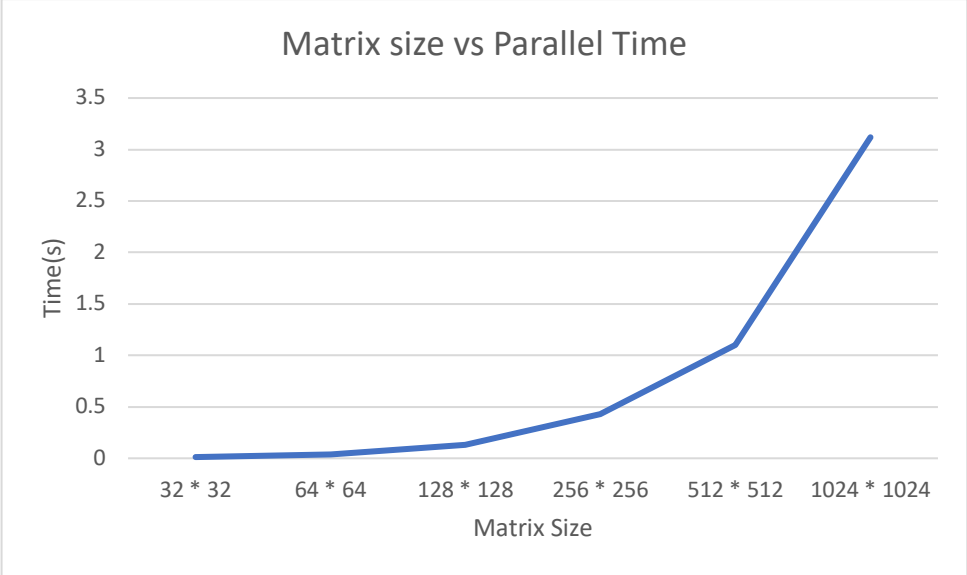

FIGURE 2

Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

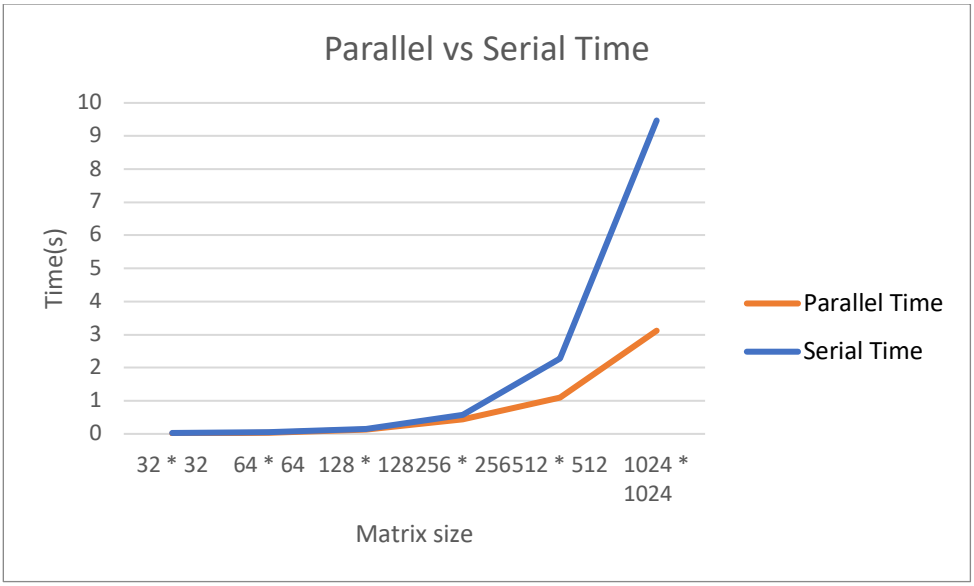

FIGURE 3

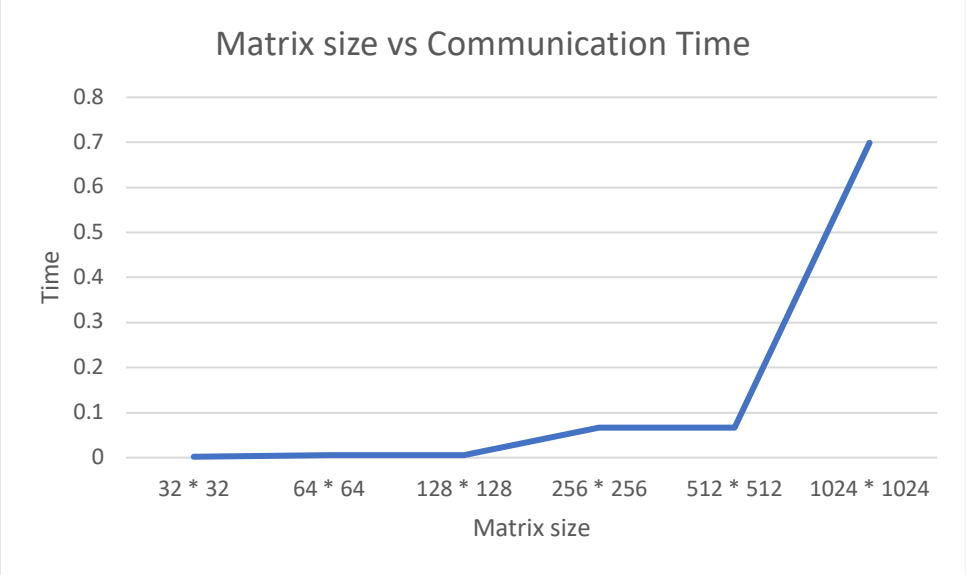

FIGURE 4

Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

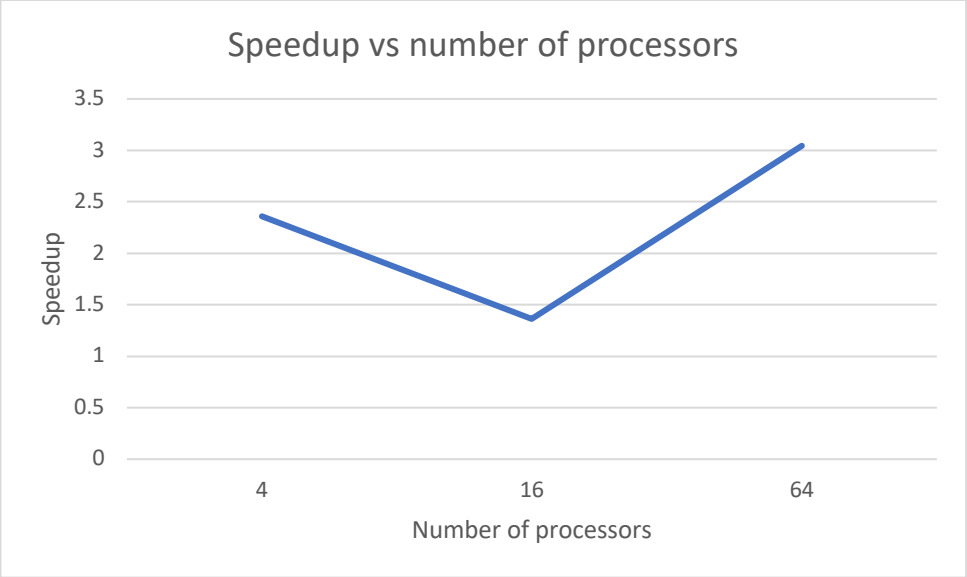

FIGURE 5

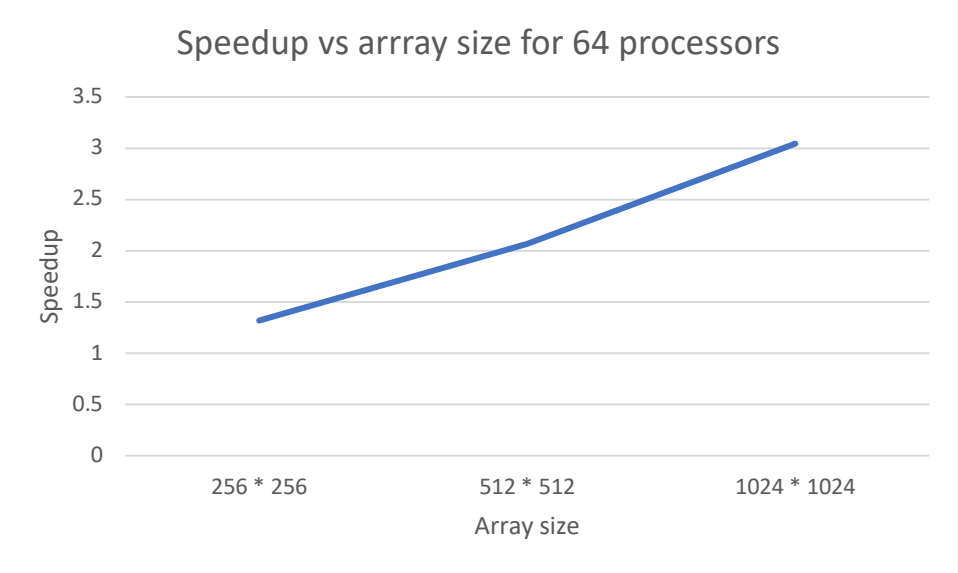

Figure 6

Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

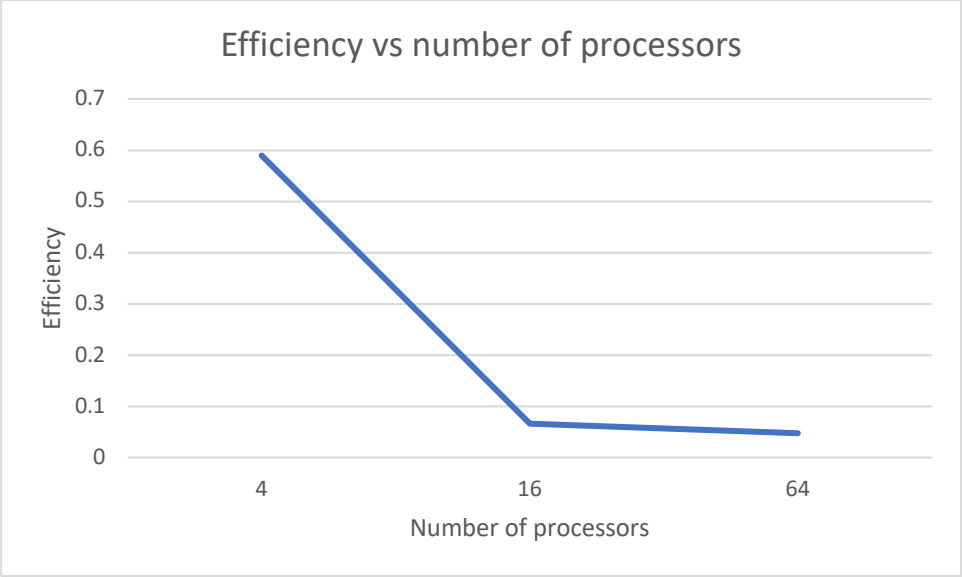

FIGURE 7

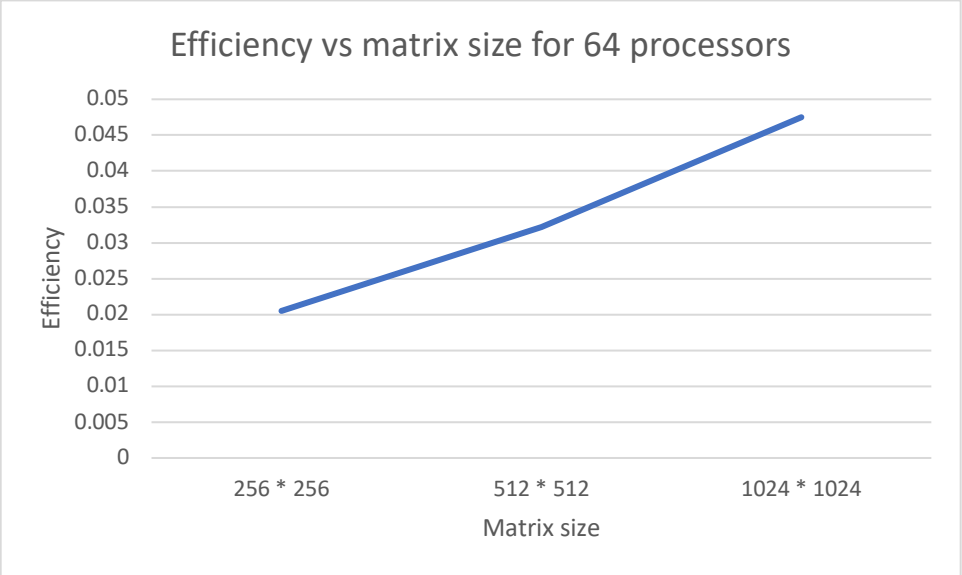

Figure 8

Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

Analysis:

1. The above graphs show the serial run time, parallel runtime, Speedup, and Efficiency of LU Decomposition and calculating determinant of the matrix.
2. From Figure 1 and 2, it is evident that as the size of the matrix increases, both serial and parallel runtime increases. It is clear that with increase in workload, the time needed for the implementation also increases.
3. From Figure 3, we can see the serial run time is almost similar (varies very little) compared to other parallel algorithms upto matrix size of 256*256 since the communication time between different processes in a parallel system nullify the advantage of parallel system. Thus, the communication time is a overhead here which increases the overall time.
4. From Figure 3, we can observe that for a large matrix size of 512*512 and 1024 * 1024 there is a huge increase in the serial time however parallel runtime is comparatively lower. Thus, the division of workload between processes in the parallel system saves time when they work in parallel.
5. From Figure 4, we can see that the communication time is very high when the size of the matrix is 512*512 and 1024 * 1024. With increase in the size of the matrix, more workload needs to be communicated between different processes.
6. From Figure 5, speedup is always above 1 for all matrix sizes, but we can see a significate increase when matrix size is larger.
7. Especially for 64 processes, we see a huge increase in speedup as evident in Figure 6.
8. From Figure 7, we can see that efficiency drops as number of processes increase and matrix size increases. This is due to the overhead in parallel system.
9. Efficiency is directly proportional to speedup and inversely proportional to number of processes.
10. Finally, in the last figure we see as the matrix size increases, the efficiency also increases keeping the number of processes constant.

Lessons Learnt:

1. In this lab assignment we learnt various parallel formulation of LU Decomposition and calculating determinant of the matrix.
2. We understood the various factors that affects the time taken to compute the same.
3. This assignment also helped understand how to decompose matrix into blocks, and how to communicate between processes.
4. In conclusion, parallelization decreases the time taken to calculate matrix multiplication however as matrix size increases, there is a overhead associated with communication time. In comparison with serial time, there is a significant decrease in the time taken overall.
5. Time increases if we increase number of process nodes since communication time increases.
6. In this assignment we learnt how to calculate speedup and efficiency, , and the factors that affects the two.