Gaussian Elimination

OpenMP vs MPI

PDC Project

Huzaifa Quaid Johar (K21-3272)

Saleh Shamoon (K21-3333)

Ahmed Saif (K21-3232)

Problem:

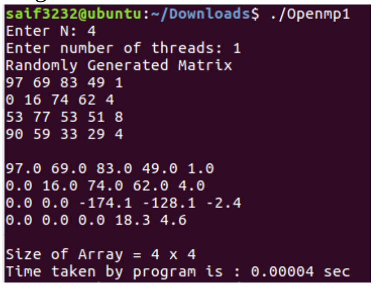
The problem presented by us in our submitted proposal was the solving a matrix using Gaussian Elimination via OpenMP and MPI and comparing the two.

Testing and Report:

We used the following references for the OMP and MPI codes for solving the aforementioned problem.

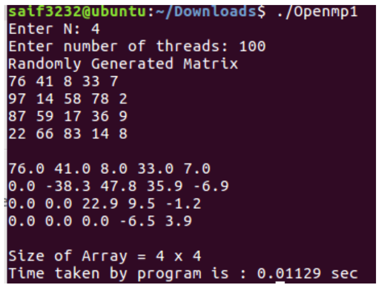
The rubrics provided asked us to run the program with the following:

- Single Thread:



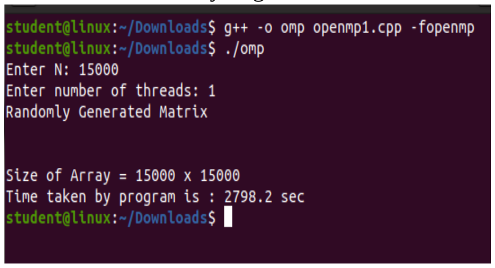
```
saif3232@ubuntu:~/Downloads$ ./Openmp1
Enter N: 4
Enter number of threads: 1
Randomly Generated Matrix
97 69 83 49 1
0 16 74 62 4
53 77 53 51 8
90 59 33 29 4

97.0 69.0 83.0 49.0 1.0
0.0 16.0 74.0 62.0 4.0
0.0 0.0 -174.1 -128.1 -2.4
0.0 0.0 0.0 18.3 4.6

Size of Array = 4 x 4
Time taken by program is : 0.00004 sec
```

- Multithreads:



```
saif3232@ubuntu:~/Downloads$ ./Openmp1
Enter N: 4
Enter number of threads: 100
Randomly Generated Matrix
76 41 8 33 7
97 14 58 78 2
87 59 17 36 9
22 66 83 14 8

76.0 41.0 8.0 33.0 7.0
0.0 -38.3 47.8 35.9 -6.9
0.0 0.0 22.9 9.5 -1.2
0.0 0.0 0.0 -6.5 3.9

Size of Array = 4 x 4
Time taken by program is : 0.01129 sec
```

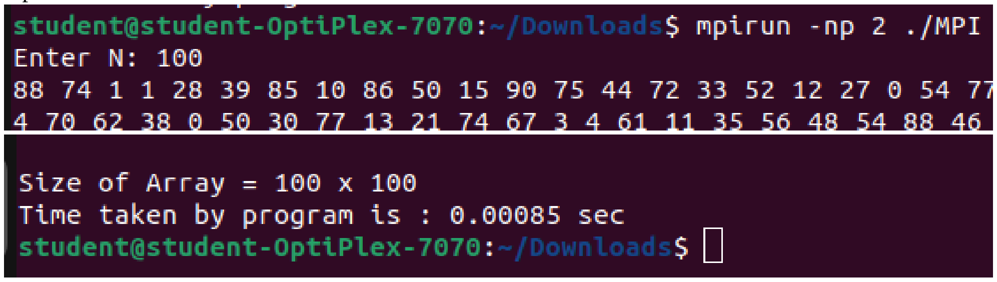
- Performance on a very large data-set with 1 thread (46 minutes):



```
student@linux:~/Downloads$ g++ -o omp openmp1.cpp -fopenmp
student@linux:~/Downloads$ ./omp
Enter N: 15000
Enter number of threads: 1
Randomly Generated Matrix


Size of Array = 15000 x 15000
Time taken by program is : 2798.2 sec
student@linux:~/Downloads$ 
```

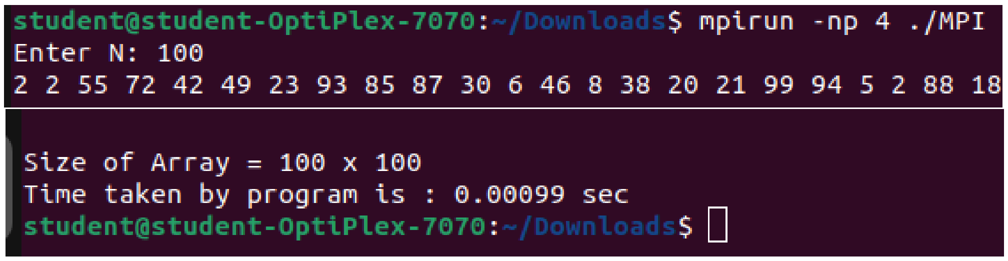
- Speedup of Parallel versions:

2 processes for 100x100 matrix



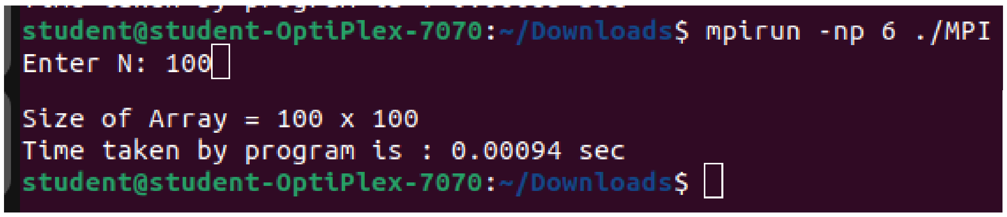
```
student@student-OptiPlex-7070:~/Downloads$ mpirun -np 2 ./MPI
Enter N: 100
88 74 1 1 28 39 85 10 86 50 15 90 75 44 72 33 52 12 27 0 54 77
4 70 62 38 0 50 30 77 13 21 74 67 3 4 61 11 35 56 48 54 88 46

Size of Array = 100 x 100
Time taken by program is : 0.00085 sec
student@student-OptiPlex-7070:~/Downloads$
```

4 processes for 100x100 matrix



```
student@student-OptiPlex-7070:~/Downloads$ mpirun -np 4 ./MPI
Enter N: 100
2 2 55 72 42 49 23 93 85 87 30 6 46 8 38 20 21 99 94 5 2 88 18

Size of Array = 100 x 100
Time taken by program is : 0.00099 sec
student@student-OptiPlex-7070:~/Downloads$
```

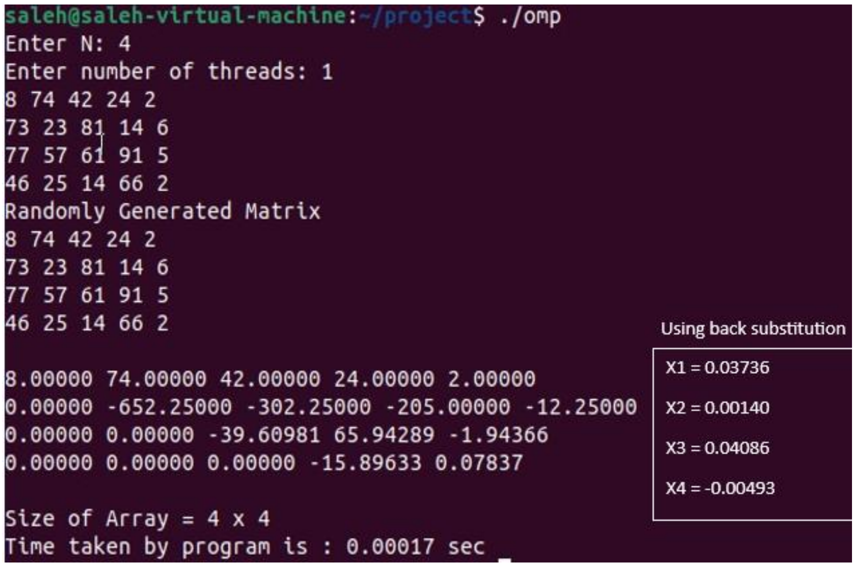
6 processes for 100x100 matrix



```
student@student-OptiPlex-7070:~/Downloads$ mpirun -np 6 ./MPI
Enter N: 100

Size of Array = 100 x 100
Time taken by program is : 0.00094 sec
student@student-OptiPlex-7070:~/Downloads$
```

- Validating outputs generated by both methods:

1 thread



```
saleh@saleh-virtual-machine:~/project$ ./omp
Enter N: 4
Enter number of threads: 1
8 74 42 24 2
73 23 81 14 6
77 57 61 91 5
46 25 14 66 2
Randomly Generated Matrix
8 74 42 24 2
73 23 81 14 6
77 57 61 91 5
46 25 14 66 2

8.00000 74.00000 42.00000 24.00000 2.00000
0.00000 -652.25000 -302.25000 -205.00000 -12.25000
0.00000 0.00000 -39.60981 65.94289 -1.94366
0.00000 0.00000 0.00000 -15.89633 0.07837

Size of Array = 4 x 4
Time taken by program is : 0.00017 sec
```

Using back substitution

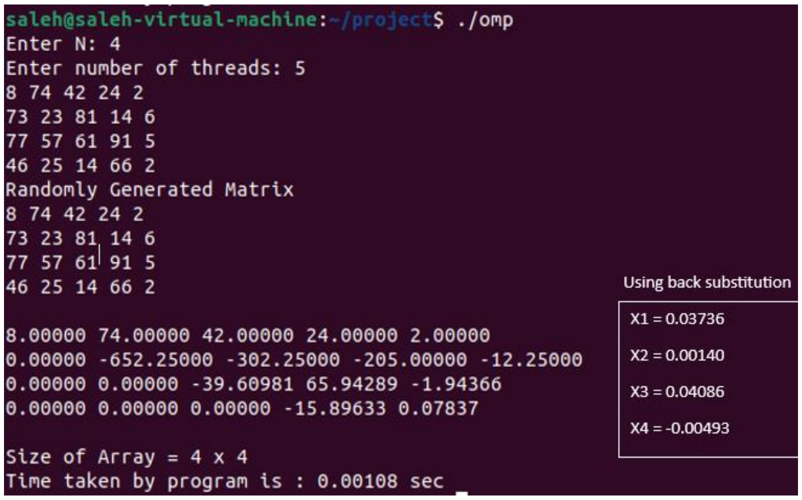
X1 = 0.03736

X2 = 0.00140

X3 = 0.04086

X4 = -0.00493

5 threads

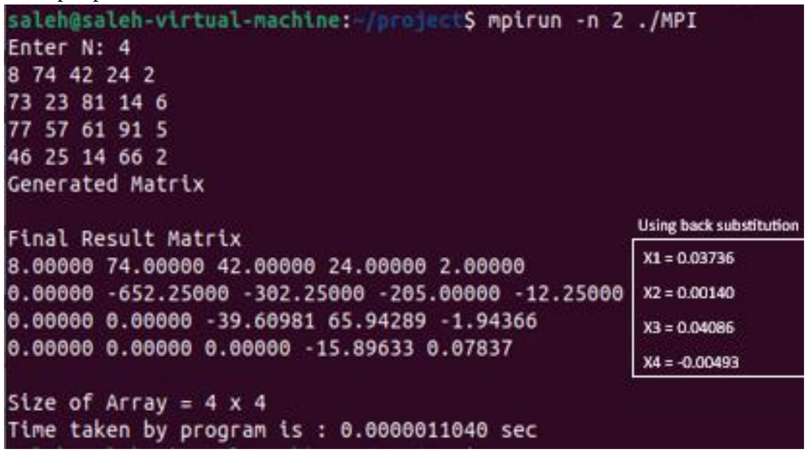


```
saleh@saleh-virtual-machine:~/project$ ./omp
Enter N: 4
Enter number of threads: 5
8 74 42 24 2
73 23 81 14 6
77 57 61 91 5
46 25 14 66 2
Randomly Generated Matrix
8 74 42 24 2
73 23 81 14 6
77 57 61 91 5
46 25 14 66 2

8.00000 74.00000 42.00000 24.00000 2.00000
0.00000 -652.25000 -302.25000 -205.00000 -12.25000
0.00000 0.00000 -39.60981 65.94289 -1.94366
0.00000 0.00000 0.00000 -15.89633 0.07837

Size of Array = 4 x 4
Time taken by program is : 0.00108 sec
```

Using back substitution

```
X1 = 0.03736
X2 = 0.00140
X3 = 0.04086
X4 = -0.00493
```

Multiple processes

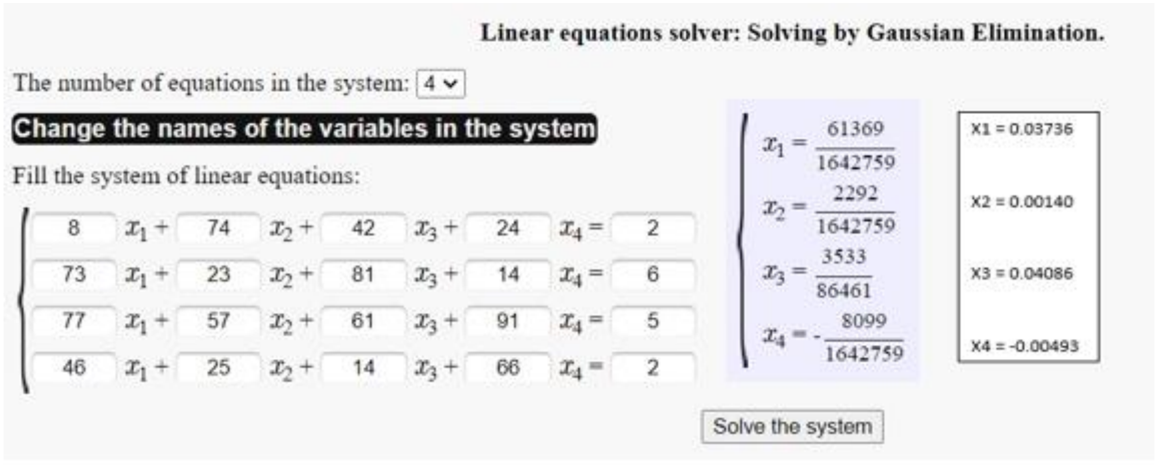


```
saleh@saleh-virtual-machine:~/project$ mpirun -n 2 ./MPI
Enter N: 4
8 74 42 24 2
73 23 81 14 6
77 57 61 91 5
46 25 14 66 2
Generated Matrix

Final Result Matrix
8.00000 74.00000 42.00000 24.00000 2.00000
0.00000 -652.25000 -302.25000 -205.00000 -12.25000
0.00000 0.00000 -39.60981 65.94289 -1.94366
0.00000 0.00000 0.00000 -15.89633 0.07837

Size of Array = 4 x 4
Time taken by program is : 0.0000011040 sec
```

Using back substitution

```
X1 = 0.03736
X2 = 0.00140
X3 = 0.04086
X4 = -0.00493
```

Proof:



Linear equations solver: Solving by Gaussian Elimination.

The number of equations in the system: 4

Change the names of the variables in the system

Fill the system of linear equations:

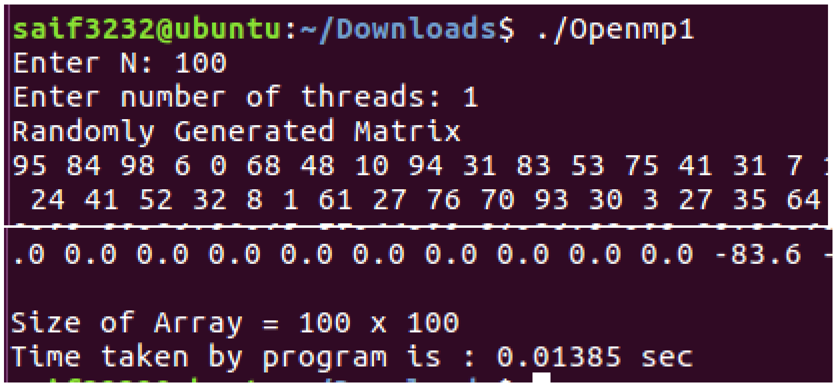
$$\begin{cases} 8x_1 + 74x_2 + 42x_3 + 24x_4 = 2 \\ 73x_1 + 23x_2 + 81x_3 + 14x_4 = 6 \\ 77x_1 + 57x_2 + 61x_3 + 91x_4 = 5 \\ 46x_1 + 25x_2 + 14x_3 + 66x_4 = 2 \end{cases}$$

$$\begin{cases} x_1 = \frac{61369}{1642759} \\ x_2 = \frac{2292}{1642759} \\ x_3 = \frac{3533}{86461} \\ x_4 = -\frac{8099}{1642759} \end{cases}$$

```
X1 = 0.03736
X2 = 0.00140
X3 = 0.04086
X4 = -0.00493
```

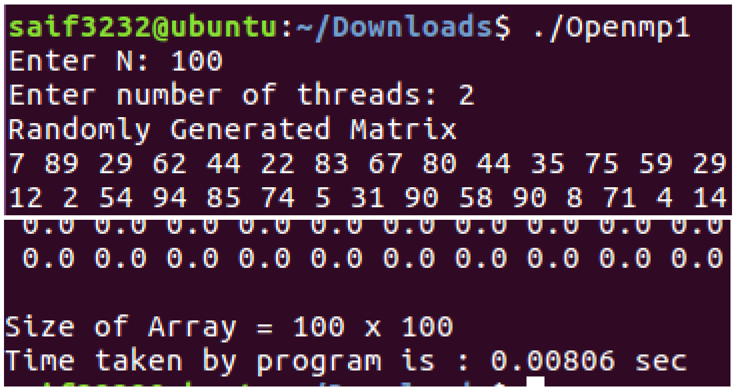
- m-cut off maximum number of threads and processes after which performance falls:
  Checking one-by-one at what number of threads and processes does the program performance falls

  1 thread



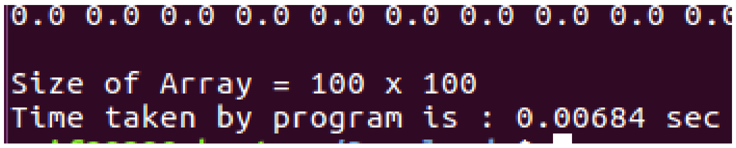
```
saif3232@ubuntu:~/Downloads$ ./Openmp1
Enter N: 100
Enter number of threads: 1
Randomly Generated Matrix
95 84 98 6 0 68 48 10 94 31 83 53 75 41 31 7
 24 41 52 32 8 1 61 27 76 70 93 30 3 27 35 64
.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -83.6

Size of Array = 100 x 100
Time taken by program is : 0.01385 sec
```

  2 threads



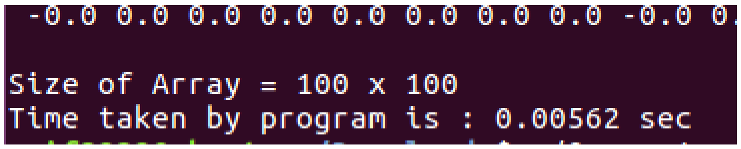
```
saif3232@ubuntu:~/Downloads$ ./Openmp1
Enter N: 100
Enter number of threads: 2
Randomly Generated Matrix
7 89 29 62 44 22 83 67 80 44 35 75 59 29
12 2 54 94 85 74 5 31 90 58 90 8 71 4 14
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Size of Array = 100 x 100
Time taken by program is : 0.00806 sec
```

  3 threads



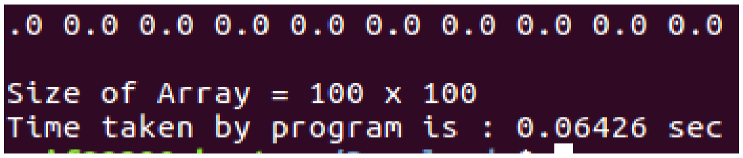
```
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Size of Array = 100 x 100
Time taken by program is : 0.00684 sec
```

  4 threads



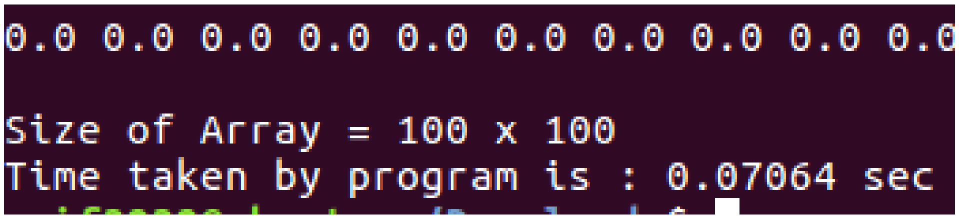
```
 -0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.0 0.

Size of Array = 100 x 100
Time taken by program is : 0.00562 sec
```

  5 threads



```
.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Size of Array = 100 x 100
Time taken by program is : 0.06426 sec
```

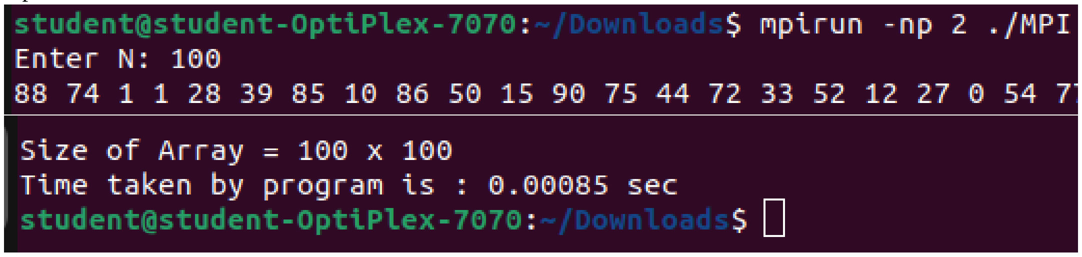
We can see that at this point the performance fell, not by much but it is a considerable time that was increased to solve the problem.
Testing on greater number of threads.
10 threads



```
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Size of Array = 100 x 100
Time taken by program is : 0.07064 sec
```

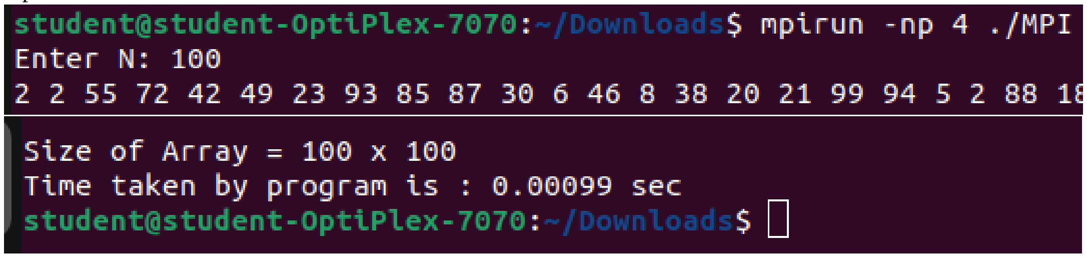
The reason for this spike in time consumption will be discussed in the write-up.

2 processes



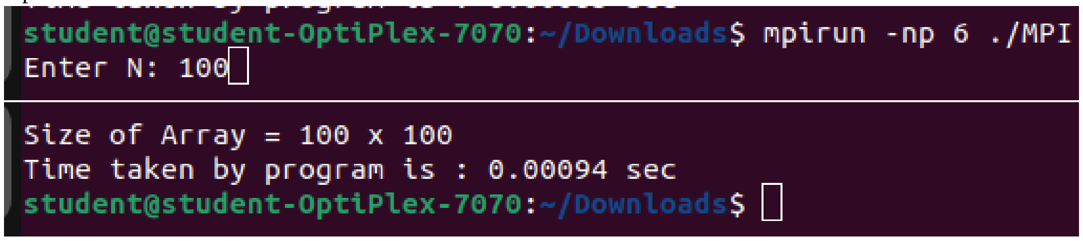
```
student@student-OptiPlex-7070:~/Downloads$ mpirun -np 2 ./MPI
Enter N: 100
88 74 1 1 28 39 85 10 86 50 15 90 75 44 72 33 52 12 27 0 54 7

Size of Array = 100 x 100
Time taken by program is : 0.00085 sec
student@student-OptiPlex-7070:~/Downloads$
```

4 processes



```
student@student-OptiPlex-7070:~/Downloads$ mpirun -np 4 ./MPI
Enter N: 100
2 2 55 72 42 49 23 93 85 87 30 6 46 8 38 20 21 99 94 5 2 88 1

Size of Array = 100 x 100
Time taken by program is : 0.00099 sec
student@student-OptiPlex-7070:~/Downloads$
```

6 processes



```
student@student-OptiPlex-7070:~/Downloads$ mpirun -np 6 ./MPI
Enter N: 100

Size of Array = 100 x 100
Time taken by program is : 0.00094 sec
student@student-OptiPlex-7070:~/Downloads$
```

We can see that increasing the number of processes, much like with threads, increases the time taken to solve the Gaussian elimination problem.

Write-Up:

We learned from our course that OpenMP simplifies parallel programming by adding directives to a program, enabling shared memory multiprocessing, while MPI focuses on explicit message passing between separate processes, facilitating communication in distributed memory systems. Developers often use a combination of both models to leverage the advantages of shared and distributed memory architectures in parallel computing. We were tasked to create a project and observe how the 2 differ when it comes to application on complex problems.

The project we chose differentiates how OpenMP and MPI take up on the task to solve Gaussian elimination. We saw and were able to deduce that given a smaller data-set, both OpenMP and MPI take quite very little time to solve the problem. Referring to the screenshots attached, we can clearly see how efficiently both the methods work in order to solve multiple variations i.e. different ranges of inputs and sizes, in very minimal time. Although OpenMP was found to be slower than MPI since OpenMP emphasizes parallelizing tasks within a single node's shared memory but may face scalability issues with larger datasets due to memory overhead and resource contention. MPI, focusing on communication between distributed memory systems, excels in scaling across multiple nodes. MPI often outperforms OpenMP in scenarios requiring inter-node communication.

Increasing threads in OpenMP for Gaussian elimination can lead to longer execution times due to overheads in thread creation, contention for resources, potential work imbalance, and saturation of memory bandwidth. Balancing workload distribution, minimizing unnecessary synchronization, optimizing memory access, and experimenting with thread counts can help mitigate these issues for improved performance. With more MPI processes, communication overhead increases due to more inter-process communication, synchronization, and potential contention for network resources. This can lead to higher latency, increased message passing time, and scalability issues, impacting the performance of Gaussian elimination as the number of processes grows. Optimizing communication patterns, reducing unnecessary message passing, and balancing workload distribution can alleviate these bottlenecks.