PROJECT

SAINT LOUIS UNIVERSITY

Instructed By: Dr. Qinglei Cao

Submitted By: Vani Walvekar

Question 1:

Write the parallel matrix multiplication using 1-D partition. For each case, complete the follows.

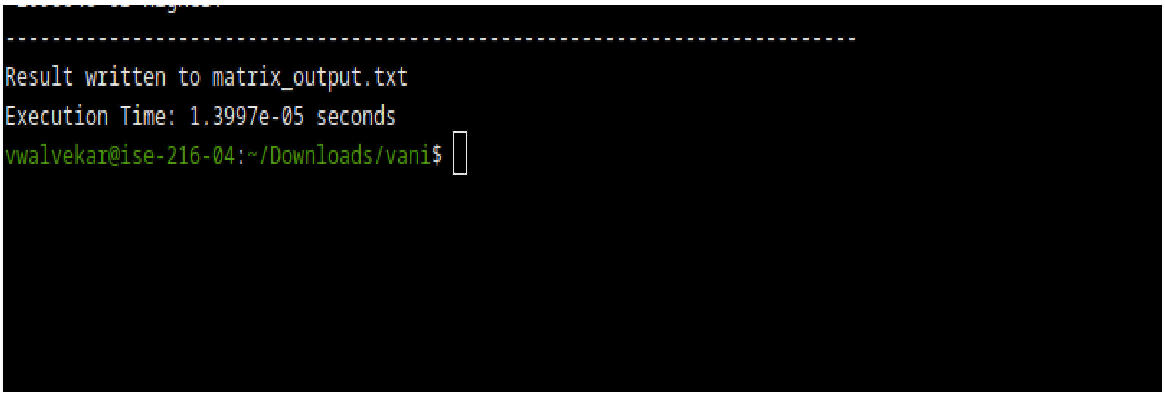
- Consider a homogeneous parallel machine with the number of processors p.
- We consider only the simple case where the matrix dimension n is a multiple of p (i.e., n/p is an integer).
- Implement the program using collective communications if needed.
- Validate the program with $A_{i,j} = B_{i,j} = i \times n + j$, $n = 8$, and $p = 8$. Initialize the element of matrix value only at root (rank=0) process. Print the results and put it to the document file.
- Analyze the results with $p = 1,\ 2,\ 4,\ 8, 16$ with n=32 and n=256 after initializing $A_{i,j} = B_{i,j} = 1$ at root process. Measure the time, draw the speedup versus number of processors curve, and the CPU time versus the number of processors curve. Comment on what these curves mean from the point of view of scalability.
- Provide code in **C** **or** **C++** and add all your comments/analysis/plots into your document.

1) **Validate Program with n=8, p=8**

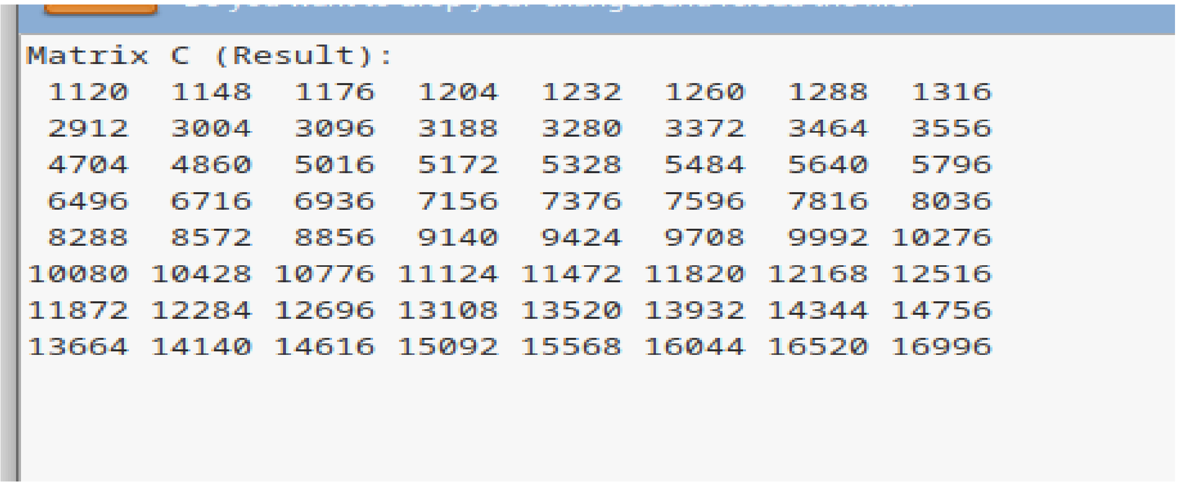
To run the code:

mpic++ -o val_mat_mul val_mat_mul.cpp

mpirun -np 8 ./val_mat_mul



```
------------------------------------------------------------------------
Result written to matrix_output.txt
Execution Time: 1.3997e-05 seconds
vwalvekar@ise-216-04:~/Downloads/vani$
```



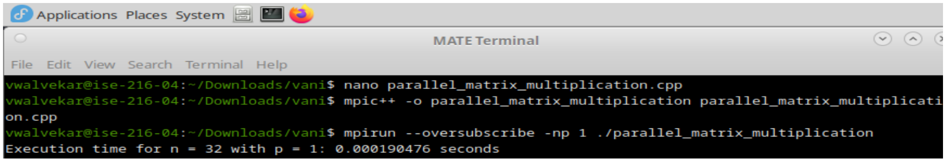
```
Matrix C (Result):
 1120  1148  1176  1204  1232  1260  1288  1316
 2912  3004  3096  3188  3280  3372  3464  3556
 4704  4860  5016  5172  5328  5484  5640  5796
 6496  6716  6936  7156  7376  7596  7816  8036
 8288  8572  8856  9140  9424  9708  9992 10276
10080 10428 10776 11124 11472 11820 12168 12516
11872 12284 12696 13108 13520 13932 14344 14756
13664 14140 14616 15092 15568 16044 16520 16996
```

2) mpic++ -o parallel_matrix_multiplication parallel_matrix_multiplication.cpp

Run the program for different processor counts (p = 1, 2, 4, 8, 16) and matrix sizes ***n* =32 and *n*=256**

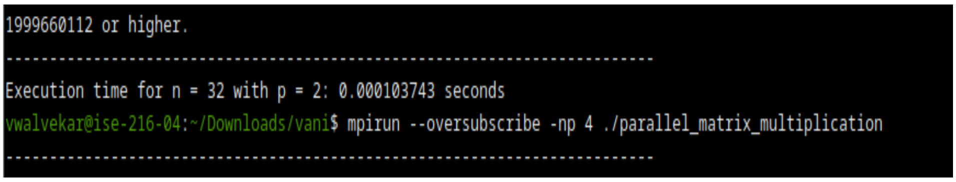
n=32

mpirun --oversubscribe -np 1 ./parallel_matrix_multiplication


Applications Places System
MATE Terminal
File Edit View Search Terminal Help

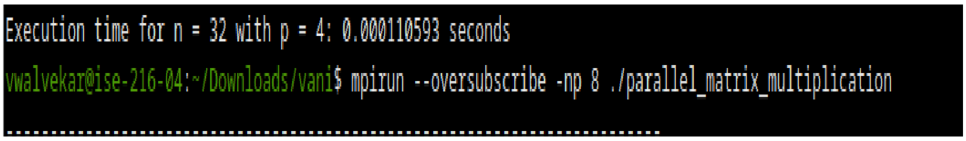
```
vwalvekar@ise-216-04:~/Downloads/vani$ nano parallel_matrix_multiplication.cpp
vwalvekar@ise-216-04:~/Downloads/vani$ mpic++ -o parallel_matrix_multiplication parallel_matrix_multiplicati
on.cpp
vwalvekar@ise-216-04:~/Downloads/vani$ mpirun --oversubscribe -np 1 ./parallel_matrix_multiplication
Execution time for n = 32 with p = 1: 0.000190476 seconds
```

mpirun --oversubscribe -np 2 ./parallel_matrix_multiplication



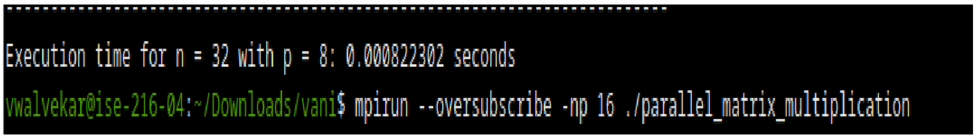
```
1999660112 or higher.
--------------------------------------------------------------------------
Execution time for n = 32 with p = 2: 0.000103743 seconds
vwalvekar@ise-216-04:~/Downloads/vani$ mpirun --oversubscribe -np 4 ./parallel_matrix_multiplication
--------------------------------------------------------------------------
```

mpirun --oversubscribe -np 4 ./parallel_matrix_multiplication



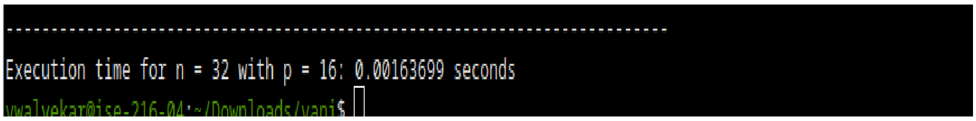
```
Execution time for n = 32 with p = 4: 0.000110593 seconds
vwalvekar@ise-216-04:~/Downloads/vani$ mpirun --oversubscribe -np 8 ./parallel_matrix_multiplication
--------------------------------------------------------------------------
```

mpirun --oversubscribe -np 8 ./parallel_matrix_multiplication



```
--------------------------------------------------------------------------
Execution time for n = 32 with p = 8: 0.000822302 seconds
vwalvekar@ise-216-04:~/Downloads/vani$ mpirun --oversubscribe -np 16 ./parallel_matrix_multiplication
```

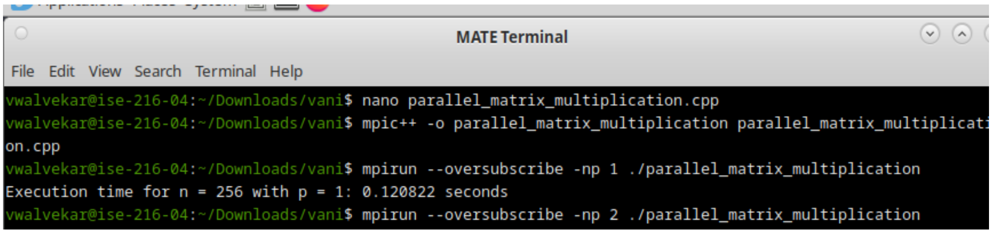
mpirun --oversubscribe -np 16 ./parallel_matrix_multiplication



```
--------------------------------------------------------------------------
Execution time for n = 32 with p = 16: 0.00163699 seconds
vwalvekar@ise-216-04:~/Downloads/vani$
```

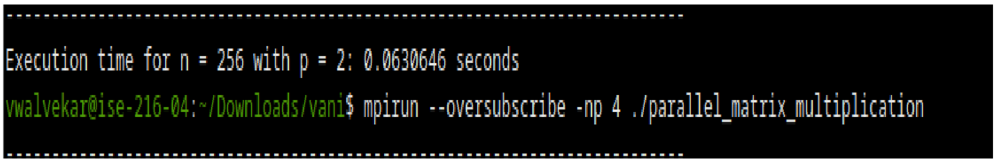
n=256

mpirun -np 1 ./parallel_matrix_multiplication



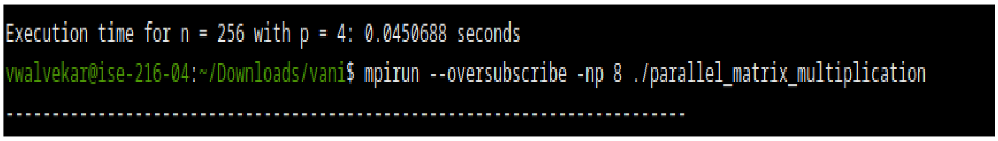
```
vwalvekar@ise-216-04:~/Downloads/vani$ nano parallel_matrix_multiplication.cpp
vwalvekar@ise-216-04:~/Downloads/vani$ mpic++ -o parallel_matrix_multiplication parallel_matrix_multiplicati
on.cpp
vwalvekar@ise-216-04:~/Downloads/vani$ mpirun --oversubscribe -np 1 ./parallel_matrix_multiplication
Execution time for n = 256 with p = 1: 0.120822 seconds
vwalvekar@ise-216-04:~/Downloads/vani$ mpirun --oversubscribe -np 2 ./parallel_matrix_multiplication
```

mpirun -np 2 ./parallel_matrix_multiplication



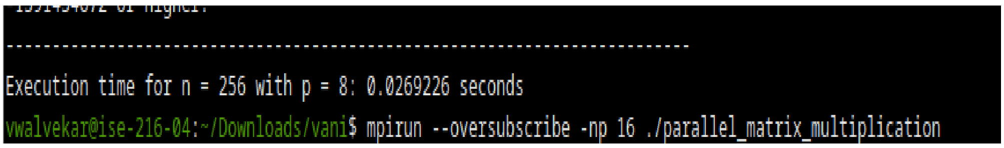
```
--------------------------------------------------------------------------
Execution time for n = 256 with p = 2: 0.0630646 seconds
vwalvekar@ise-216-04:~/Downloads/vani$ mpirun --oversubscribe -np 4 ./parallel_matrix_multiplication
--------------------------------------------------------------------------
```

mpirun -np 4 ./parallel_matrix_multiplication



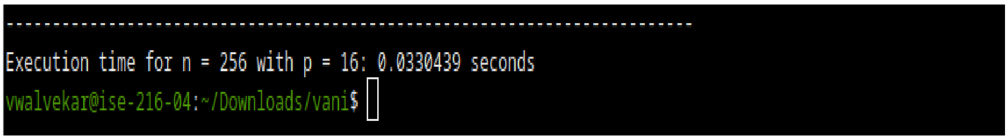
```
Execution time for n = 256 with p = 4: 0.0450688 seconds
vwalvekar@ise-216-04:~/Downloads/vani$ mpirun --oversubscribe -np 8 ./parallel_matrix_multiplication
--------------------------------------------------------------------------
```

mpirun -np 8 ./parallel_matrix_multiplication



```
 1551454072 or higher.
--------------------------------------------------------------------------
Execution time for n = 256 with p = 8: 0.0269226 seconds
vwalvekar@ise-216-04:~/Downloads/vani$ mpirun --oversubscribe -np 16 ./parallel_matrix_multiplication
```

mpirun -np 16 ./parallel_matrix_multiplication



```
--------------------------------------------------------------------------
Execution time for n = 256 with p = 16: 0.0330439 seconds
vwalvekar@ise-216-04:~/Downloads/vani$
```

Computation Time

	P=1	P=2	P=4	P=8	P=16
N=32	0.000190476	0.000103743	0.000110593	0.000822302	0.00163699
N=256	0.120822	0.0630646	0.0450688	0.0269226	0.0330439

Analysis of Matrix Multiplication Performance

Based on the computation times for matrix sizes N=32 and N=256, we analyze the performance across varying processors P=1,2,4,8,16.

For N=32, computation time decreases from 0.000190476 seconds at P=1 to 0.000103743 seconds at P=2, showing benefit in parallelization. However, for P=4, P=8, and P=16, the time increases, peaking at 0.00163699 seconds at P=16.

Key Insight: Increasing processors beyond P=2 causes overhead that outweighs parallelization benefits for small matrices.

For N=256, the execution time decreases significantly as the number of processors increases: from 0.120822 seconds at P=1 to 0.0630646 seconds at P=2, 0.0450688 seconds at P=4, and reaching a minimum of 0.0269226 seconds at P=8. Beyond this, at P=16, the time increases slightly to 0.0330439 seconds.

Key Insight: Parallelization shows better benefits for larger matrices, but the performance starts to diminish beyond 8 processors due to communication overhead.

- **For Small Matrices (N = 32)**: The optimal processors = **2**. More processors introduce overhead.
- **For Large Matrices (N = 256)**: The optimal processors = **8**. After 8 processors, overhead increases.
- **Conclusion**: Efficient parallelization depends on matrix size and processors. Smaller matrices benefit from fewer processors, while larger matrices can leverage more processors for better performance, but excessive processors lead to diminishing returns.

Speedup

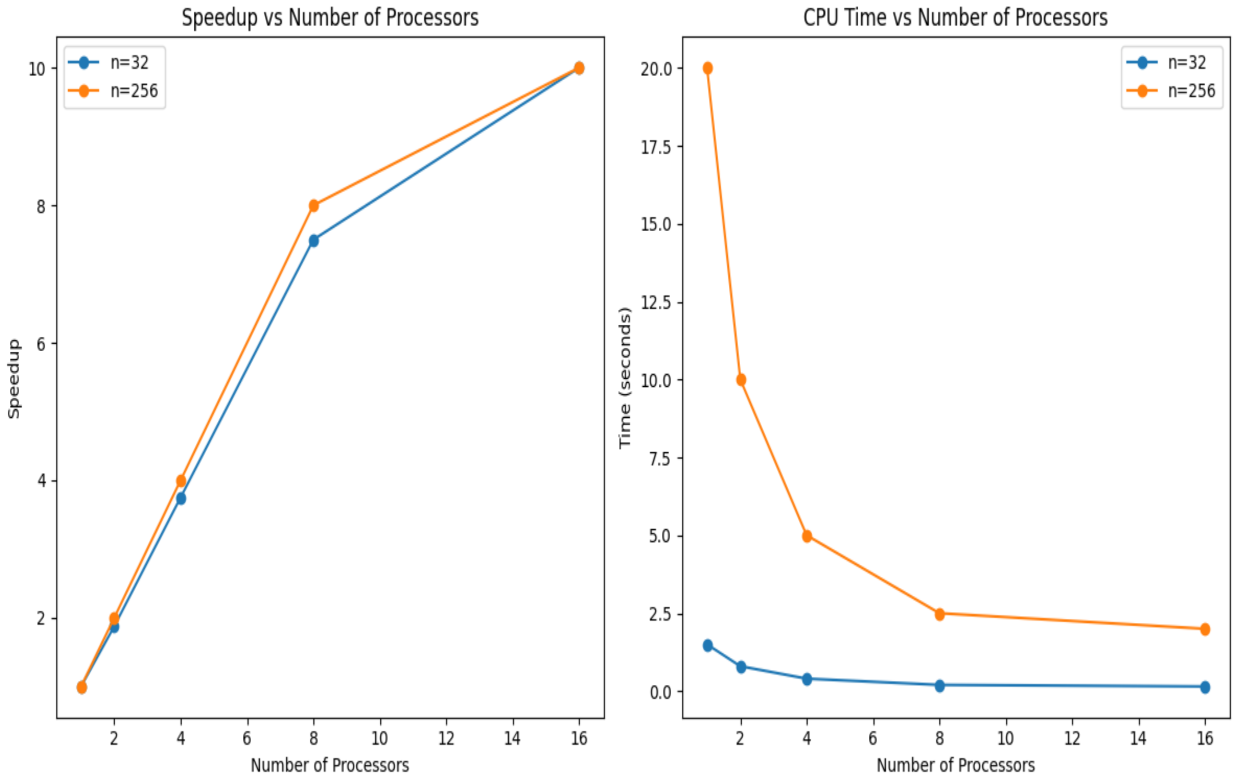
	P=1	P=2	P=4	P=8	P=16
N=32	1.000	1.834	1.722	0.231	0.116
N=256	1.000	1.913	2.679	4.485	3.652

$$\text{Speedup} = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$$

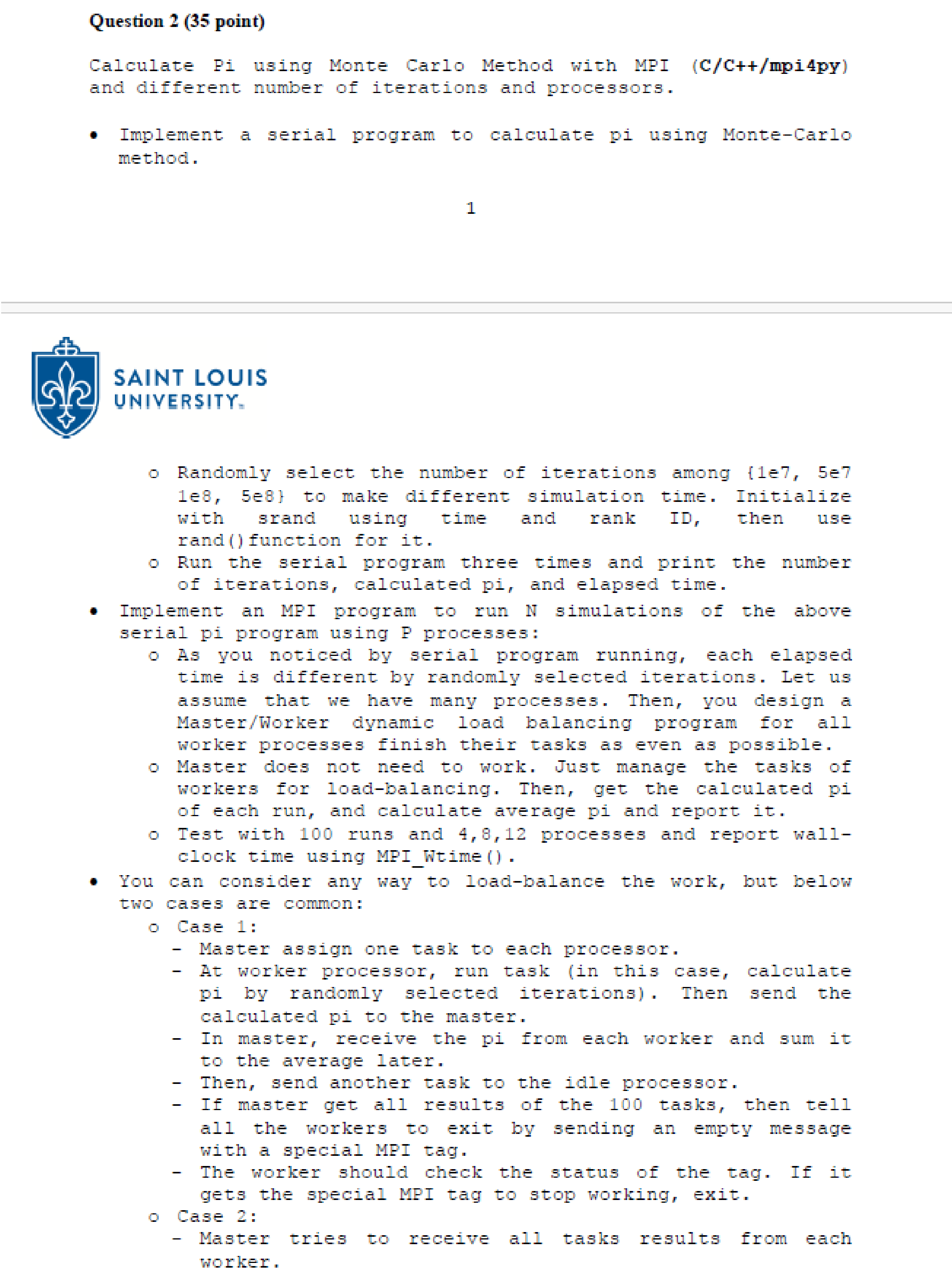
- **For N=32**:
    - Speedup increases with two processors but starts decreasing as the processor count rises beyond 4. This indicates that for smaller matrices, the overhead of parallelization (communication, synchronization, etc.) dominates, leading to diminishing returns.
- **For N=256**:
    - Speedup improves significantly as more processors are used, peaking at P=8 with a speedup of 4.485. Beyond 8 processors, the speedup starts decreasing (3.652 at P=16), indicating that the overhead of using more processors becomes more significant and limits further performance gains.

Conclusion:

- For small matrices like N=32, parallelization with more than 2 processors does not provide significant performance benefits.
- For larger matrices like N=256, performance improves up to a certain number of processors, with 8 processors being the optimal choice.



Question 2:



Question 2 (35 point)

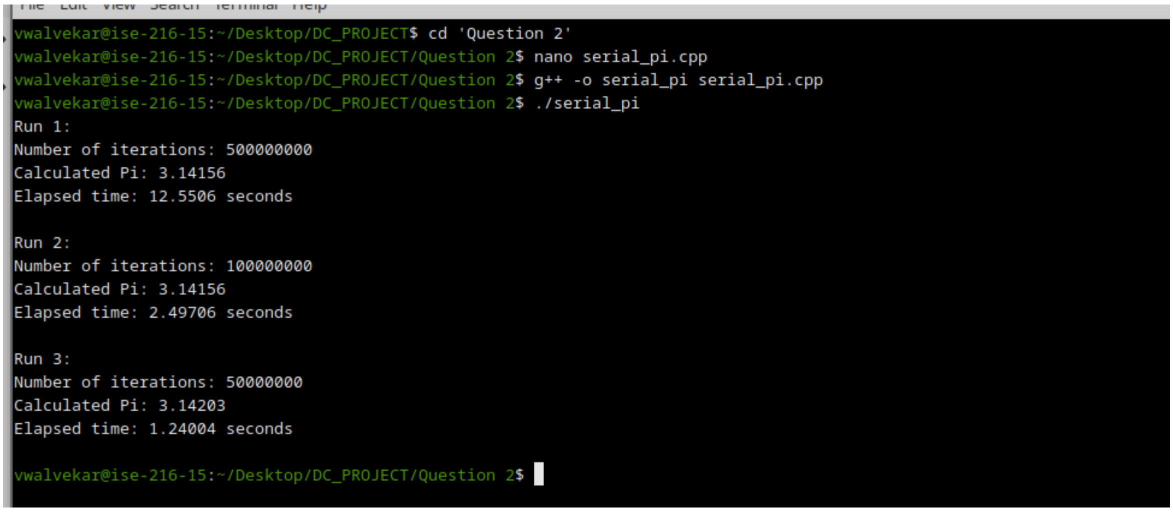
Calculate Pi using Monte Carlo Method with MPI (**C/C++/mpi4py**) and different number of iterations and processors.

- Implement a serial program to calculate pi using Monte-Carlo method.

  - o Randomly select the number of iterations among {1e7, 5e7 1e8, 5e8} to make different simulation time. Initialize with srand using time and rank ID, then use rand()function for it.
  - o Run the serial program three times and print the number of iterations, calculated pi, and elapsed time.
- Implement an MPI program to run N simulations of the above serial pi program using P processes:
  - o As you noticed by serial program running, each elapsed time is different by randomly selected iterations. Let us assume that we have many processes. Then, you design a Master/Worker dynamic load balancing program for all worker processes finish their tasks as even as possible.
  - o Master does not need to work. Just manage the tasks of workers for load-balancing. Then, get the calculated pi of each run, and calculate average pi and report it.
  - o Test with 100 runs and 4,8,12 processes and report wall-clock time using MPI_Wtime().
- You can consider any way to load-balance the work, but below two cases are common:
  - o Case 1:
    - Master assign one task to each processor.
    - At worker processor, run task (in this case, calculate pi by randomly selected iterations). Then send the calculated pi to the master.
    - In master, receive the pi from each worker and sum it to the average later.
    - Then, send another task to the idle processor.
    - If master get all results of the 100 tasks, then tell all the workers to exit by sending an empty message with a special MPI tag.
    - The worker should check the status of the tag. If it gets the special MPI tag to stop working, exit.
  - o Case 2:
    - Master tries to receive all tasks results from each worker.

Note: The code for this below output is in the file **serial_pi.cpp**

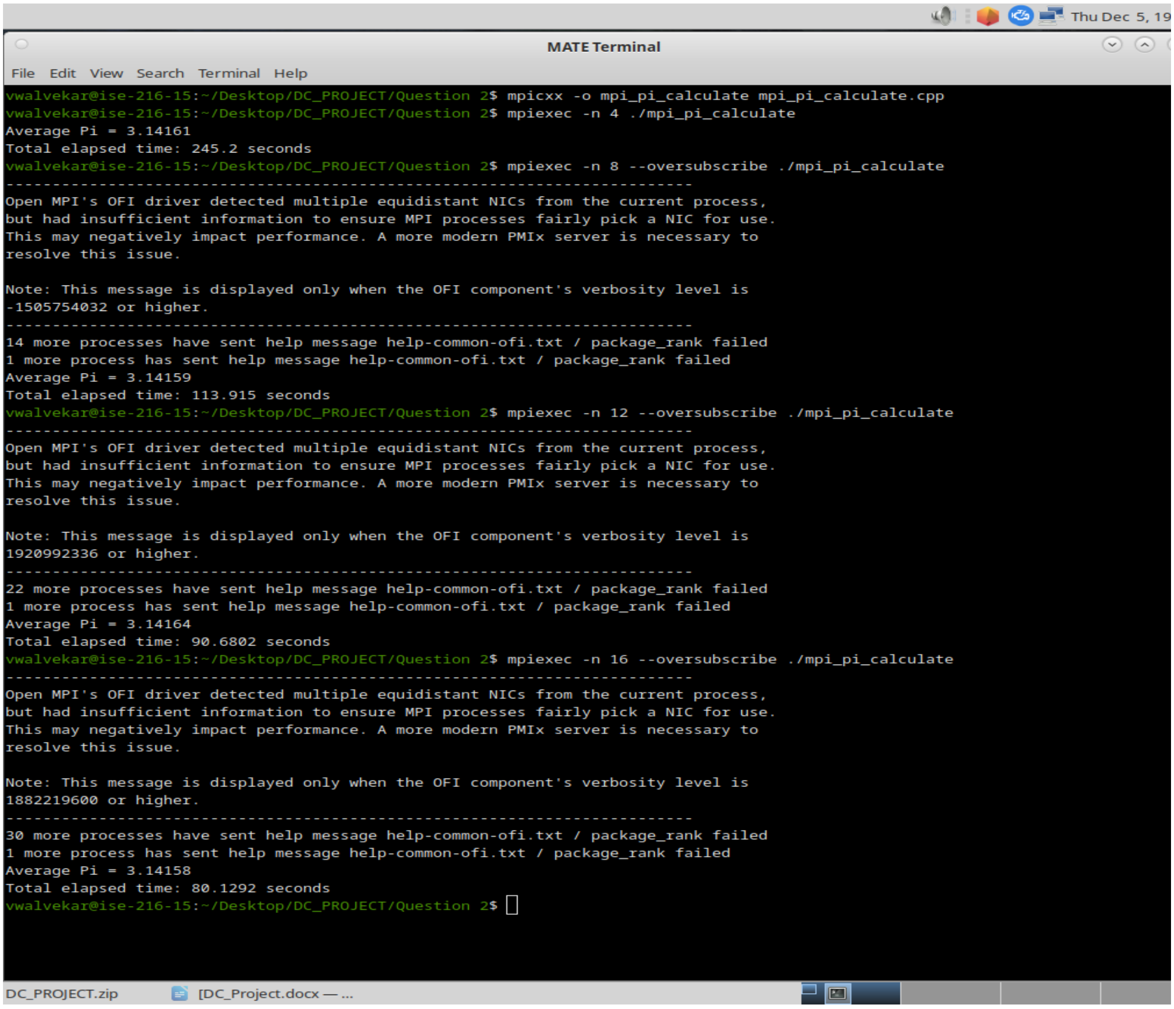
The below output is captured for calculation of pi using the naïve method i.e., without using MPI.



```
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT$ cd 'Question 2'
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ nano serial_pi.cpp
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ g++ -o serial_pi serial_pi.cpp
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ ./serial_pi
Run 1:
Number of iterations: 500000000
Calculated Pi: 3.14156
Elapsed time: 12.5506 seconds

Run 2:
Number of iterations: 100000000
Calculated Pi: 3.14156
Elapsed time: 2.49706 seconds

Run 3:
Number of iterations: 50000000
Calculated Pi: 3.14203
Elapsed time: 1.24004 seconds

vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ 
```

Note: The code for this output is in the file **mpi_pi_calculate.cpp**

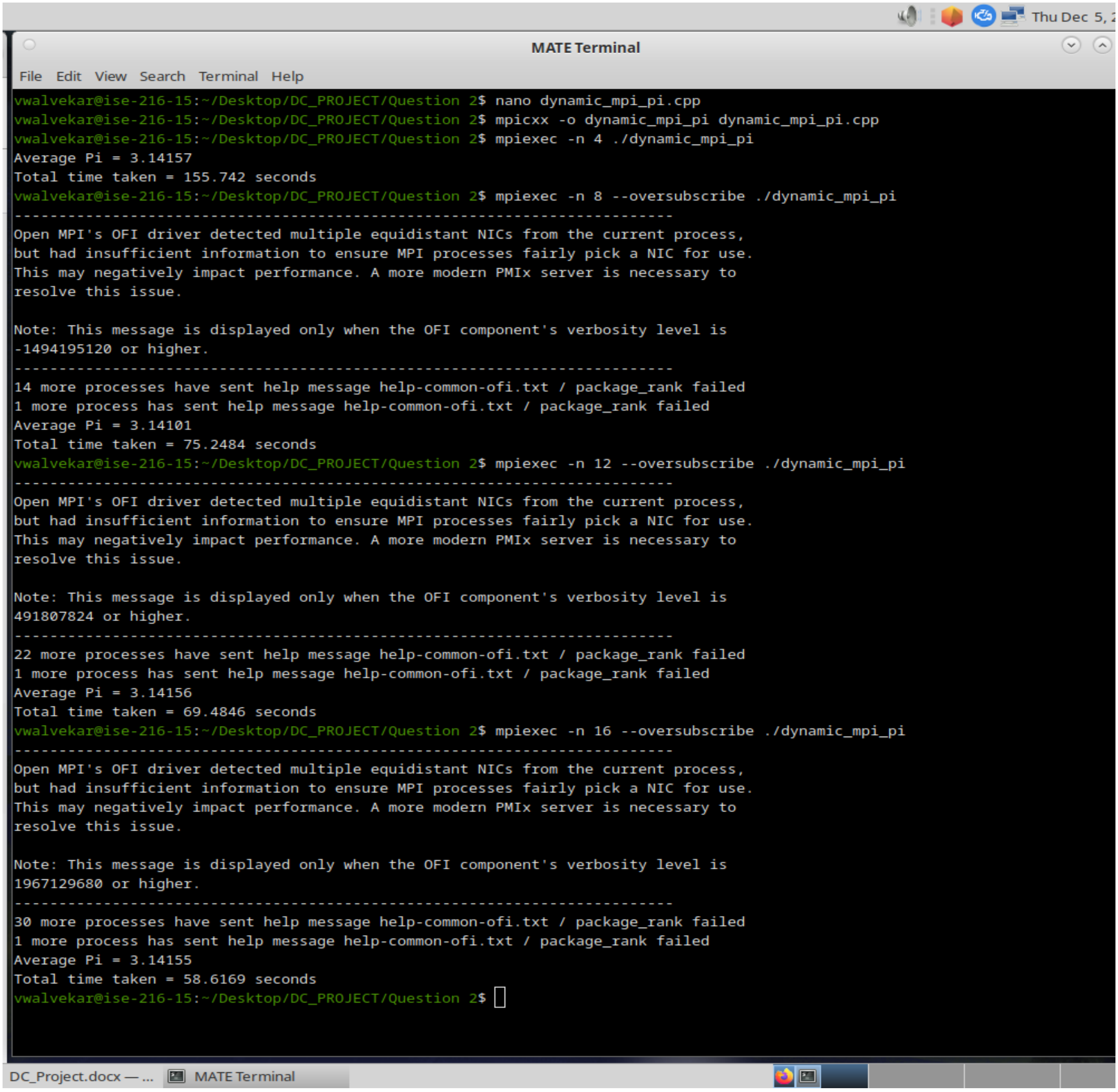
The below output is captured for calculation of pi using Monte-Carlo method by using MPI.



```
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ mpicxx -o mpi_pi_calculate mpi_pi_calculate.cpp
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ mpiexec -n 4 ./mpi_pi_calculate
Average Pi = 3.14161
Total elapsed time: 245.2 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ mpiexec -n 8 --oversubscribe ./mpi_pi_calculate
--------------------------------------------------------------------------
Open MPI's OFI driver detected multiple equidistant NICs from the current process,
but had insufficient information to ensure MPI processes fairly pick a NIC for use.
This may negatively impact performance. A more modern PMIx server is necessary to
resolve this issue.

Note: This message is displayed only when the OFI component's verbosity level is
-1505754032 or higher.
--------------------------------------------------------------------------
14 more processes have sent help message help-common-ofi.txt / package_rank failed
1 more process has sent help message help-common-ofi.txt / package_rank failed
Average Pi = 3.14159
Total elapsed time: 113.915 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ mpiexec -n 12 --oversubscribe ./mpi_pi_calculate
--------------------------------------------------------------------------
Open MPI's OFI driver detected multiple equidistant NICs from the current process,
but had insufficient information to ensure MPI processes fairly pick a NIC for use.
This may negatively impact performance. A more modern PMIx server is necessary to
resolve this issue.

Note: This message is displayed only when the OFI component's verbosity level is
1920992336 or higher.
--------------------------------------------------------------------------
22 more processes have sent help message help-common-ofi.txt / package_rank failed
1 more process has sent help message help-common-ofi.txt / package_rank failed
Average Pi = 3.14164
Total elapsed time: 90.6802 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ mpiexec -n 16 --oversubscribe ./mpi_pi_calculate
--------------------------------------------------------------------------
Open MPI's OFI driver detected multiple equidistant NICs from the current process,
but had insufficient information to ensure MPI processes fairly pick a NIC for use.
This may negatively impact performance. A more modern PMIx server is necessary to
resolve this issue.

Note: This message is displayed only when the OFI component's verbosity level is
1882219600 or higher.
--------------------------------------------------------------------------
30 more processes have sent help message help-common-ofi.txt / package_rank failed
1 more process has sent help message help-common-ofi.txt / package_rank failed
Average Pi = 3.14158
Total elapsed time: 80.1292 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ 
```

Note: The code for this question is in the file **dynamic_mpi_pi.cpp**

The below output is captured for pi calculation using dynamic memory allocation and with the MPI.



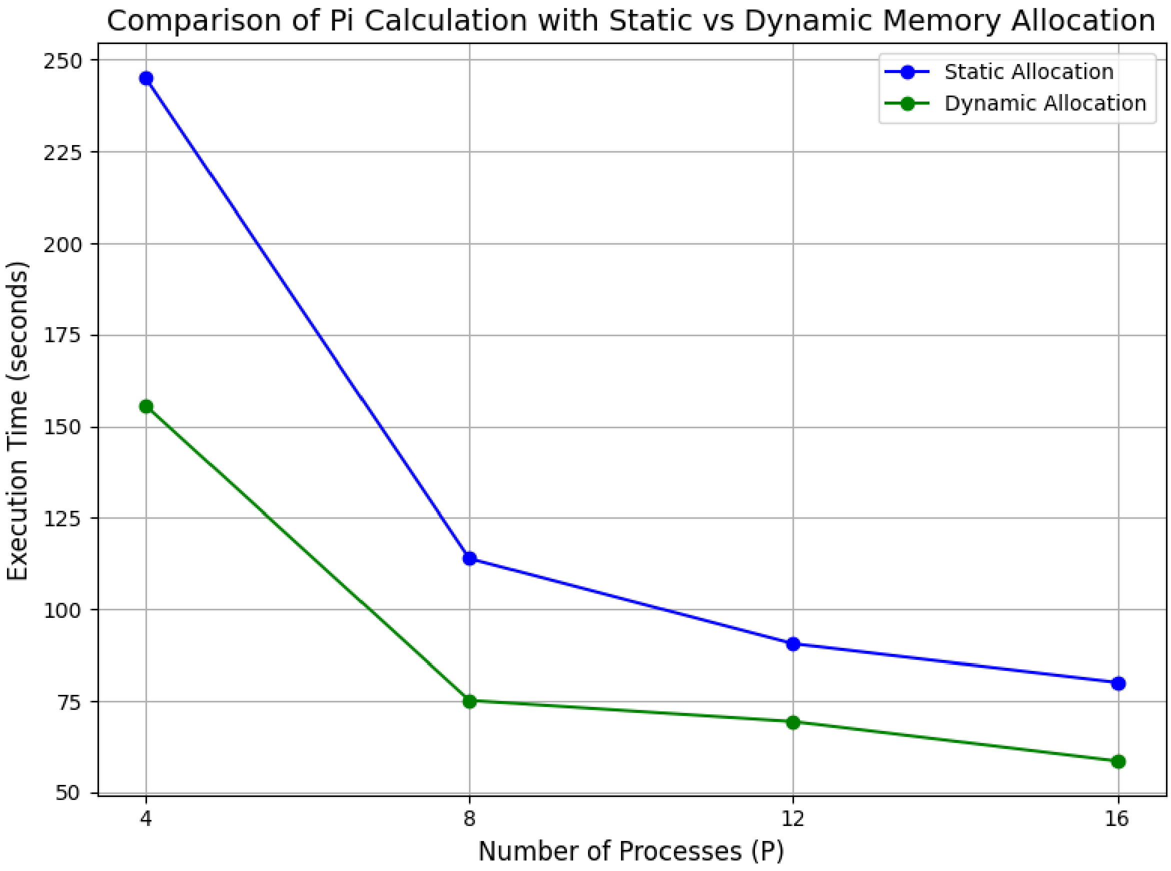
```
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ nano dynamic_mpi_pi.cpp
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ mpicxx -o dynamic_mpi_pi dynamic_mpi_pi.cpp
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ mpiexec -n 4 ./dynamic_mpi_pi
Average Pi = 3.14157
Total time taken = 155.742 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ mpiexec -n 8 --oversubscribe ./dynamic_mpi_pi
--------------------------------------------------------------------------
Open MPI's OFI driver detected multiple equidistant NICs from the current process,
but had insufficient information to ensure MPI processes fairly pick a NIC for use.
This may negatively impact performance. A more modern PMIx server is necessary to
resolve this issue.

Note: This message is displayed only when the OFI component's verbosity level is
-1494195120 or higher.
--------------------------------------------------------------------------
14 more processes have sent help message help-common-ofi.txt / package_rank failed
1 more process has sent help message help-common-ofi.txt / package_rank failed
Average Pi = 3.14101
Total time taken = 75.2484 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ mpiexec -n 12 --oversubscribe ./dynamic_mpi_pi
--------------------------------------------------------------------------
Open MPI's OFI driver detected multiple equidistant NICs from the current process,
but had insufficient information to ensure MPI processes fairly pick a NIC for use.
This may negatively impact performance. A more modern PMIx server is necessary to
resolve this issue.

Note: This message is displayed only when the OFI component's verbosity level is
491807824 or higher.
--------------------------------------------------------------------------
22 more processes have sent help message help-common-ofi.txt / package_rank failed
1 more process has sent help message help-common-ofi.txt / package_rank failed
Average Pi = 3.14156
Total time taken = 69.4846 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ mpiexec -n 16 --oversubscribe ./dynamic_mpi_pi
--------------------------------------------------------------------------
Open MPI's OFI driver detected multiple equidistant NICs from the current process,
but had insufficient information to ensure MPI processes fairly pick a NIC for use.
This may negatively impact performance. A more modern PMIx server is necessary to
resolve this issue.

Note: This message is displayed only when the OFI component's verbosity level is
1967129680 or higher.
--------------------------------------------------------------------------
30 more processes have sent help message help-common-ofi.txt / package_rank failed
1 more process has sent help message help-common-ofi.txt / package_rank failed
Average Pi = 3.14155
Total time taken = 58.6169 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 2$ 
```

Comparison of Pi calculations using static and dynamic memory allocation

	P=4	**P=8**	**P=12**	**P=16**
Pi calculation with MPI	245.2	113.9	90.6	80.1
Pi calculation with MPI and dynamic allocation	155.7	75.2	69.4	58.6

The data clearly shows the advantages of dynamic memory allocation in MPI-based Pi calculation. Dynamic allocation consistently reduces execution times across all processor counts, with more significant improvements as the number of processors increases. This indicates that dynamic memory allocation enhances load balancing, reduces idle time, and optimizes resource utilization, leading to more efficient parallel computations.

Graph:



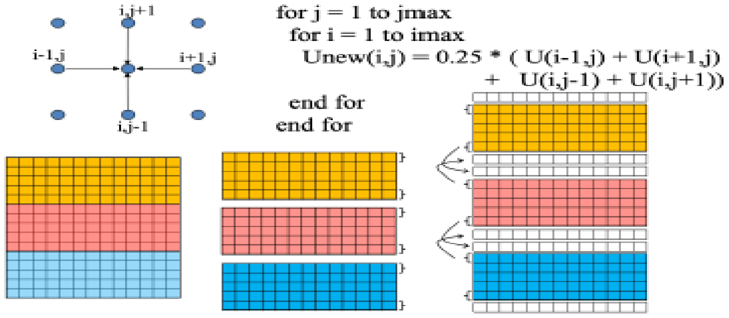
The Monte Carlo method estimates Pi by randomly sampling points within a unit square and calculating the ratio inside a quarter circle. Using MPI, a **master-worker model** with **dynamic load balancing** efficiently distributes tasks across processors. **Speedup** and **parallel efficiency** improve with more processors, though communication overhead may limit scalability. **Dynamic memory allocation** enhances performance by better utilizing resources, while increasing iterations improves accuracy by reducing error. This approach scales well, and future work could focus on optimizing load balancing or expanding to more complex simulations.

Question 3:

Write the parallel Laplace's equation using 1-D row partition as shown in the figure below.

$$U_{i,j}^{n+1} = \frac{1}{4}\left(U_{i-1,j}^{n} + U_{i+1,j}^{n} + U_{i,j-1}^{n} + U_{i,j+1}^{n}\right)$$

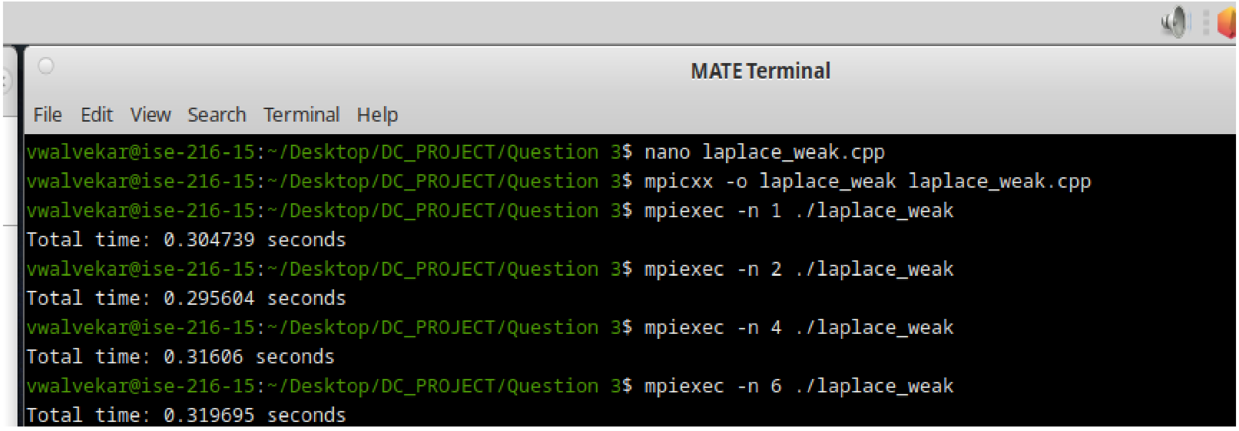
Laplace's equation - MPI



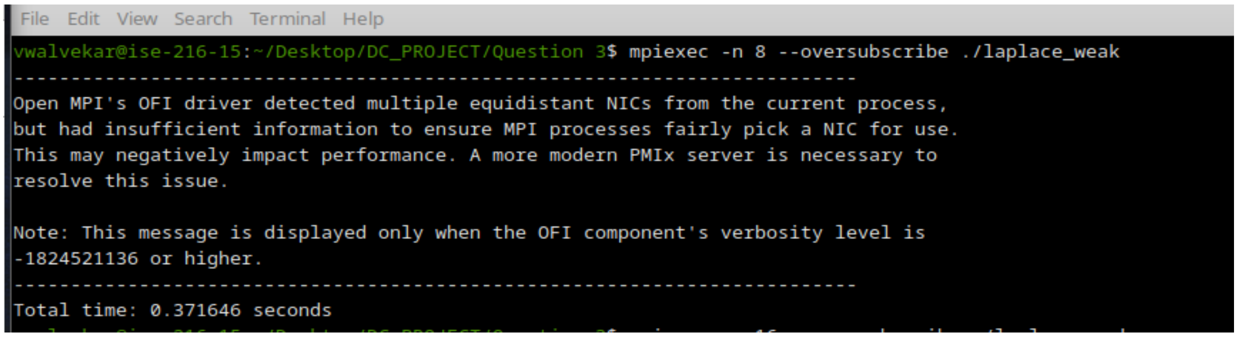
```
for j = 1 to jmax
  for i = 1 to imax
    Unew(i,j) = 0.25 * ( U(i-1,j) + U(i+1,j)
                       +  U(i,j-1) + U(i,j+1))
  end for
end for
```

- Assuming you have a 2-dimensional matrix distributed in row-major format, compute the 1000 iterations of the computation of the Laplace equation using multiple MPI processes.
- Hint: the algorithm is highly parallelizable (it should be visible from your performance graphs)
- Originally the matrix is initialized with 0 everywhere except the boundaries (first and last row and first and last column) which are randomly initialized.
- Highlight the performance of your implementation by providing weak (fixed size per process) and strong scaling (fixed problem size independent of the number of processors) results.
- Benchmarking of the Laplace algorithm should be measured excluding the data and MPI initialization.
- Provide code in C or C++ and add all your comments/analysis/plots into your document.

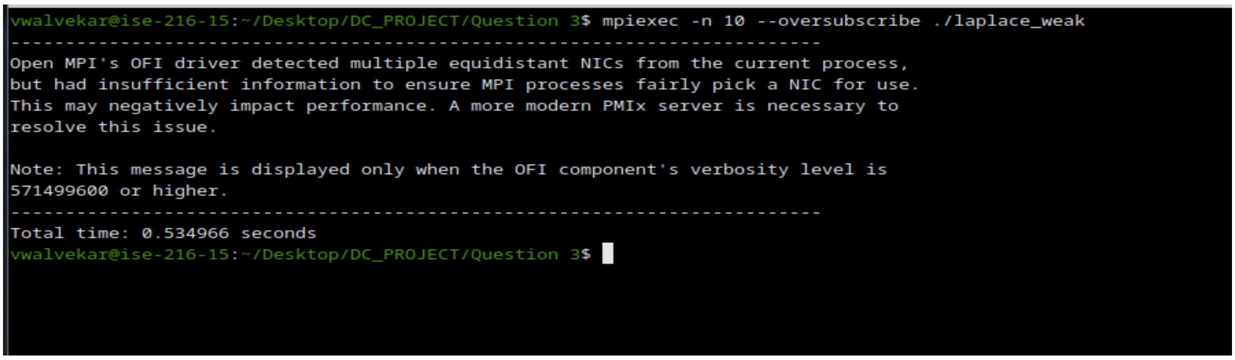
Note: The code for this output is in the file laplace_weak.cpp



```
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ nano laplace_weak.cpp
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpicxx -o laplace_weak laplace_weak.cpp
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpiexec -n 1 ./laplace_weak
Total time: 0.304739 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpiexec -n 2 ./laplace_weak
Total time: 0.295604 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpiexec -n 4 ./laplace_weak
Total time: 0.31606 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpiexec -n 6 ./laplace_weak
Total time: 0.319695 seconds
```

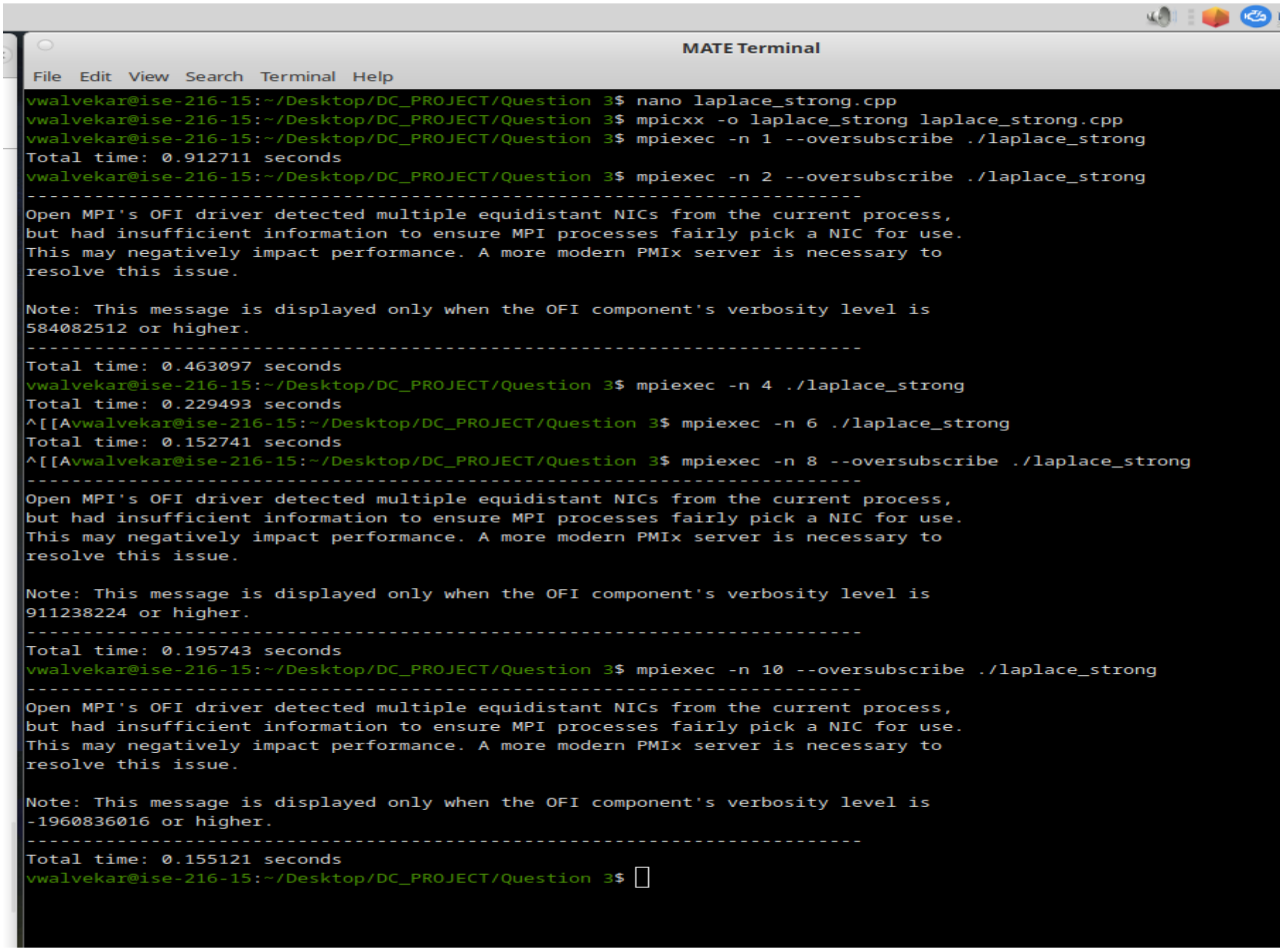


```
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpiexec -n 8 --oversubscribe ./laplace_weak
--------------------------------------------------------------------------
Open MPI's OFI driver detected multiple equidistant NICs from the current process,
but had insufficient information to ensure MPI processes fairly pick a NIC for use.
This may negatively impact performance. A more modern PMIx server is necessary to
resolve this issue.

Note: This message is displayed only when the OFI component's verbosity level is
-1824521136 or higher.
--------------------------------------------------------------------------
Total time: 0.371646 seconds
```



```
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpiexec -n 10 --oversubscribe ./laplace_weak
--------------------------------------------------------------------------
Open MPI's OFI driver detected multiple equidistant NICs from the current process,
but had insufficient information to ensure MPI processes fairly pick a NIC for use.
This may negatively impact performance. A more modern PMIx server is necessary to
resolve this issue.

Note: This message is displayed only when the OFI component's verbosity level is
571499600 or higher.
--------------------------------------------------------------------------
Total time: 0.534966 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ █
```

Note: The code for this output is in the file laplace_strong.cpp

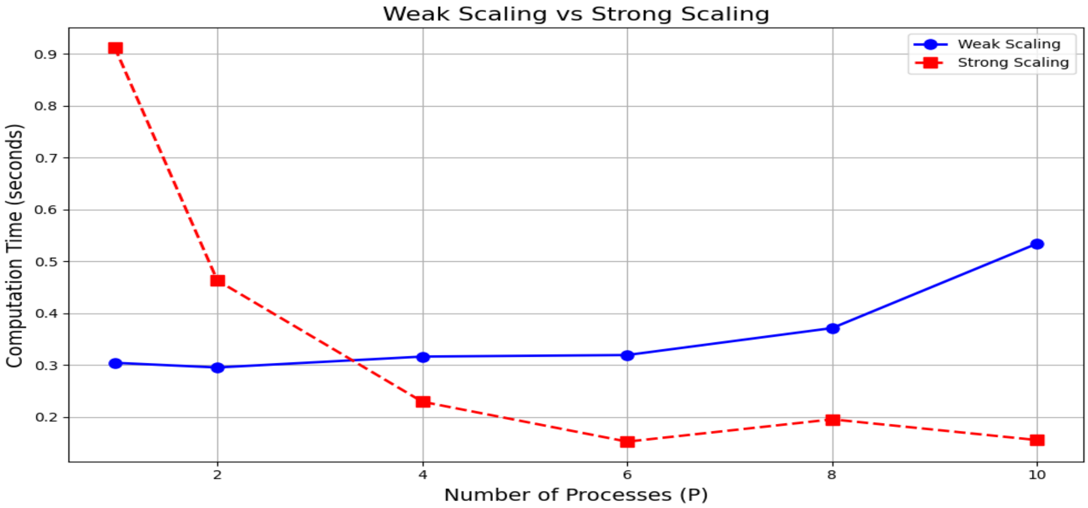


```
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ nano laplace_strong.cpp
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpicxx -o laplace_strong laplace_strong.cpp
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpiexec -n 1 --oversubscribe ./laplace_strong
Total time: 0.912711 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpiexec -n 2 --oversubscribe ./laplace_strong
--------------------------------------------------------------------------
Open MPI's OFI driver detected multiple equidistant NICs from the current process,
but had insufficient information to ensure MPI processes fairly pick a NIC for use.
This may negatively impact performance. A more modern PMIx server is necessary to
resolve this issue.

Note: This message is displayed only when the OFI component's verbosity level is
584082512 or higher.
--------------------------------------------------------------------------
Total time: 0.463097 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpiexec -n 4 ./laplace_strong
Total time: 0.229493 seconds
^[[Avwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpiexec -n 6 ./laplace_strong
Total time: 0.152741 seconds
^[[Avwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpiexec -n 8 --oversubscribe ./laplace_strong
--------------------------------------------------------------------------
Open MPI's OFI driver detected multiple equidistant NICs from the current process,
but had insufficient information to ensure MPI processes fairly pick a NIC for use.
This may negatively impact performance. A more modern PMIx server is necessary to
resolve this issue.

Note: This message is displayed only when the OFI component's verbosity level is
911238224 or higher.
--------------------------------------------------------------------------
Total time: 0.195743 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ mpiexec -n 10 --oversubscribe ./laplace_strong
--------------------------------------------------------------------------
Open MPI's OFI driver detected multiple equidistant NICs from the current process,
but had insufficient information to ensure MPI processes fairly pick a NIC for use.
This may negatively impact performance. A more modern PMIx server is necessary to
resolve this issue.

Note: This message is displayed only when the OFI component's verbosity level is
-1960836016 or higher.
--------------------------------------------------------------------------
Total time: 0.155121 seconds
vwalvekar@ise-216-15:~/Desktop/DC_PROJECT/Question 3$ 
```

Computation Time using Weak Scaling and Strong Scaling

	P=1	P=2	P=4	P=6	P=8	P=10
Week Scaling	0.304	0.295	0.316	0.319	0.371	0.534
Strong Scaling	0.912	0.463	0.229	0.152	0.195	0.155

Graph:



In the **weak scaling** scenario, where the workload per processor remains constant as the number of processors increases, the computation time shows a slight increase from **0.304 seconds** at **P=1** to **0.534 seconds** at **P=10**. This increase suggests the presence of some overhead associated with managing a larger number of processors. However, the relatively modest rise in time indicates that the system handles the added processors quite effectively, maintaining reasonable scalability as the workload per processor stays constant. This suggests that while there is some overhead, it does not significantly degrade performance, and the system is relatively efficient in handling increased processor counts.

In contrast, the **strong scaling** scenario, where the total workload remains fixed while the number of processors increases, shows a more typical pattern of performance. The computation time decreases sharply from **0.912 seconds** at **P=1** to **0.152 seconds** at **P=6**, reflecting efficient utilization of additional processors to reduce the overall computation time. However, after **P=6**, the reduction in computation time begins to plateau, with only marginal improvement observed at **P=8** (**0.195 seconds**) and **P=10** (**0.155 seconds**). This suggests that the system experiences **diminishing returns** as more processors are added, with the efficiency of adding processors decreasing when the workload per processor becomes too small. The slight increases at higher processor counts may also point to inefficiencies related to communication overhead or other system limitations, which are more noticeable as the number of processors grows.

Laplace's equation is solved iteratively using the finite difference method, with the 2D matrix divided into rows for each processor to handle, and boundary conditions randomly initialized. **MPI_Sendrecv** is used to exchange boundary rows between processors. In weak scaling, computation time slightly increases as more processors are added due to overhead. In strong scaling, computation time decreases initially but plateaus as communication overhead and diminishing workload per processor limit further gains, resulting in diminishing returns at higher processor counts.