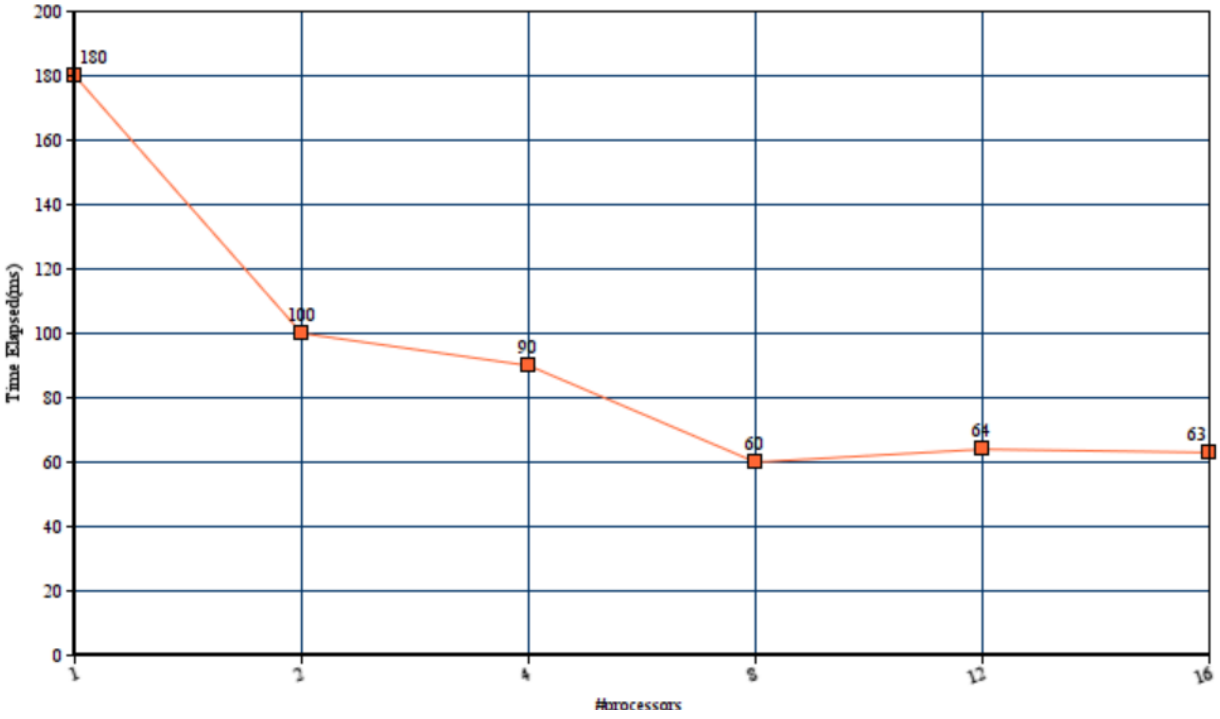
Performance Report

T_s= 123.2 ms

(a)

Sp= T_s/T_p

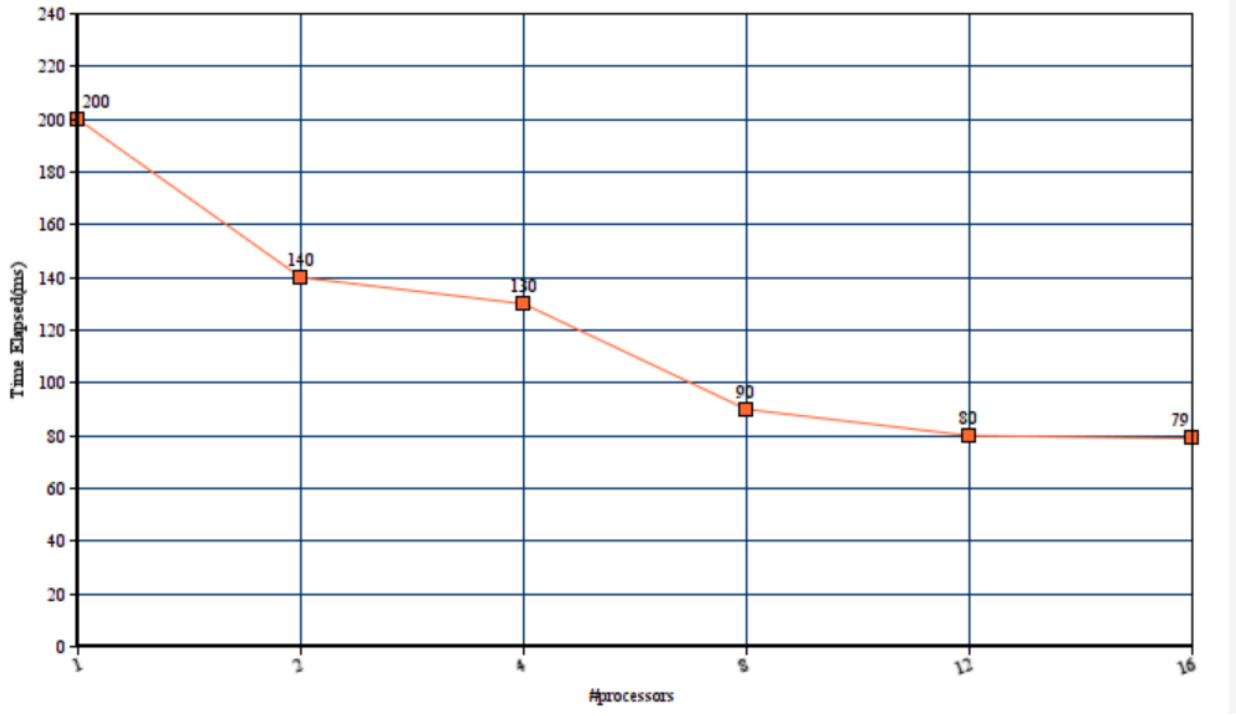
# processors	Elapsed time (ms)	Communication time(ms)	Computation Time(ms)	Speedup
1	180	4	176	0.7
2	100	10	90	1.2
4	90	9.4	80.6	1.4
8	60	7.4	52.6	2.1
12	64	6	58	1.9
18	63	4.1	58.9	2



(b)

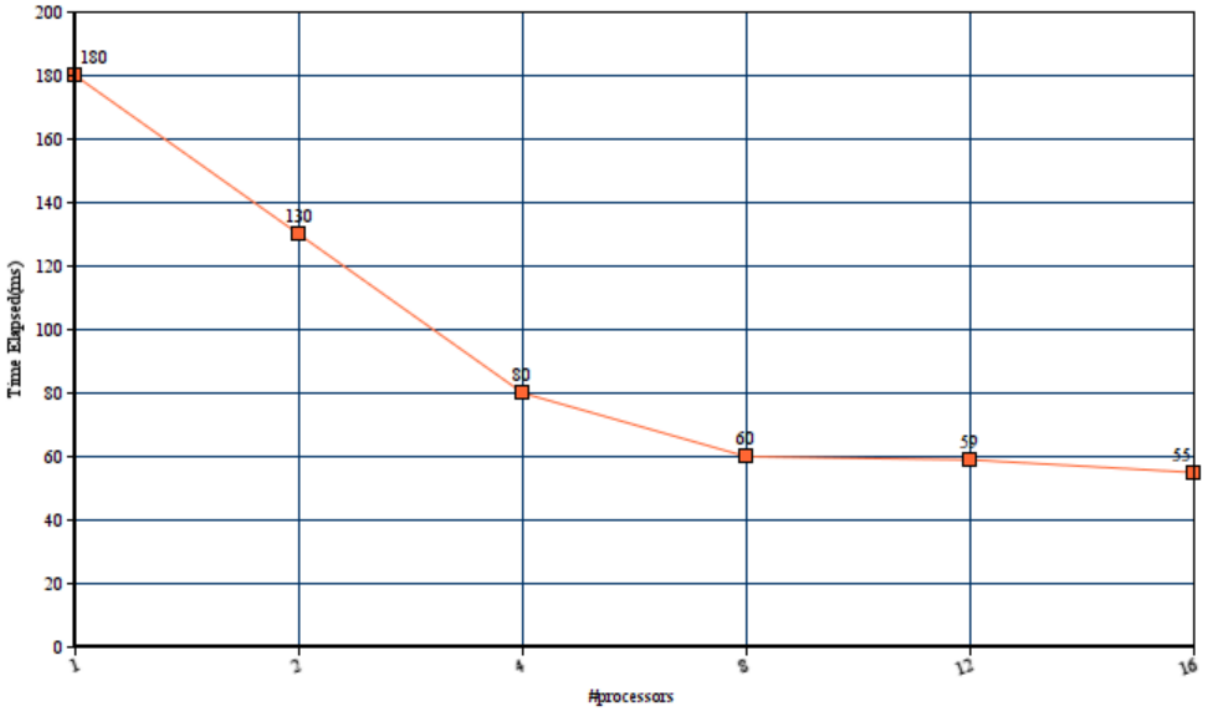
# processors	Elapsed time (ms)	Speedup
1	200	0.61
2	140	0.8
4	130	0.9
8	90	1.4
12	80	1.5

16	79	1.5



(c) Time elapsed using openMP+MPI communicator

# processors	Elapsed time (ms)	Speedup
1	180	0.7
2	130	0.9
4	80	1.5
8	60	2.1
12	59	2.1
16	55	2.2



When P1=P2=P3=P4=2

➔2*2*2*2=8 processors

Running the program on 8 processors. All the 4 tasks are divided so that they run on each **group** of processors.

Each **group** of processors contain 2 processors, which implies that every task is being handled by 2 processors.

Time taken for P1=P2=P3=2 is 0.17s

(d)

Comparing (a) and (c), for 8 processors shows that there is not much difference –

- When openMP + MPI is used **vs** only MPI_Send and MPI_Recv only.

The potential root cause for this might be –

- Huge data files being loaded in each iteration
- Usage of MPI_Send and MPI_Recv. The network utilization is not happening because it's a node to node transfer. Instead if we use MPI_bcast this could have been faster because of its tree- based communication algorithm.

- When tasks are divided over grouped processors, the performance is reduced. The potential reason might be because –

- There are always free processors in (c) task parallelism. Whereas case (a) all the processors are utilized most of the time.
- If some large computation needs to be done then we can argue that approach (c) can utilize the other processors.in parallel and free up as or when needed and then it can be efficient.

Testing of code has been done by-

1. Checking with 1 processor and comparing the result with serial code for 32X32
2. Subsequently increasing the number of processors and observing the time elapsed, if its increasing or decreasing.

- Tried generating the part 3 with MPI_ Gather and MPI_Scatter, but getting error during compilation for destination. So dropped the approach.
- Do not see a significant improvement by increasing the number of processors by 12 or 16, maybe because the algorithm is not very effective or communication time is taking longer than expected on increasing the processors compared to computation time.