MA513

Parallel Computing

Assignment-3 Report

Paranjay Bagga
160101050

Problem Statement 1 :-

Implementation of three different parallel algorithms for ***Adding n numbers*** to observe their associated overhead and speedup using MPI programming.

System Specifications :-

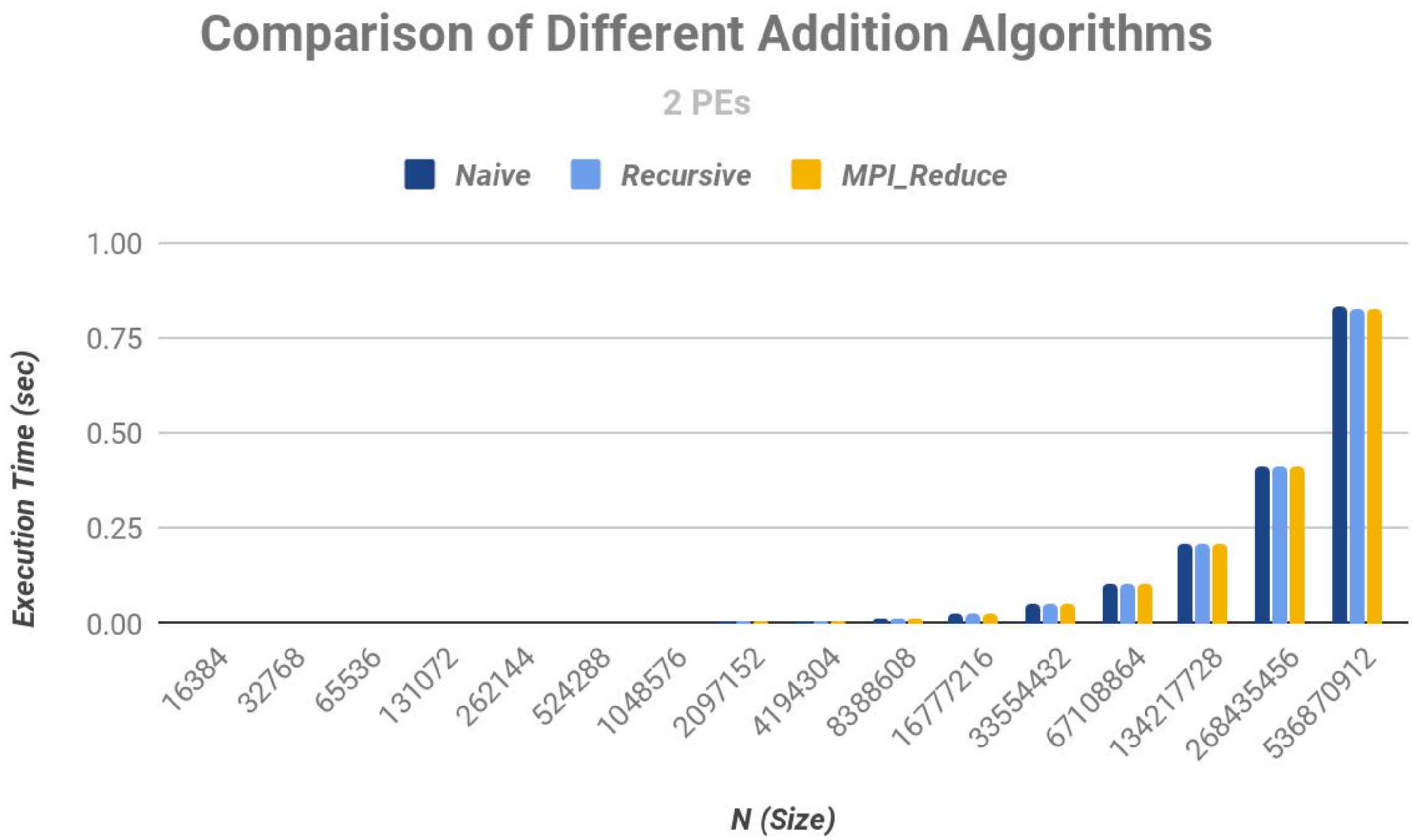
All the experiments have been performed on **Dell Inspiron 5559** laptop with **Intel i5-6200U dual core processor** and **8 GB RAM**. Each core has **2 threads.**

Experiments :-

The task is to add ***N*** numbers using parallel algorithms having ***p*** processing elements, thus giving each processor ***N/p*** elements to add locally and store the total final result in a single processor (say p0).
The value of *N* is varied from **2^14(16284)** upto **2^29(536870912)**.
The results are shown in the tables and graphs below. The values in the table are average values of 10 executions to avoid any unexpected behaviour introduced as a result of thread switching.

1. Using 2 PEs :

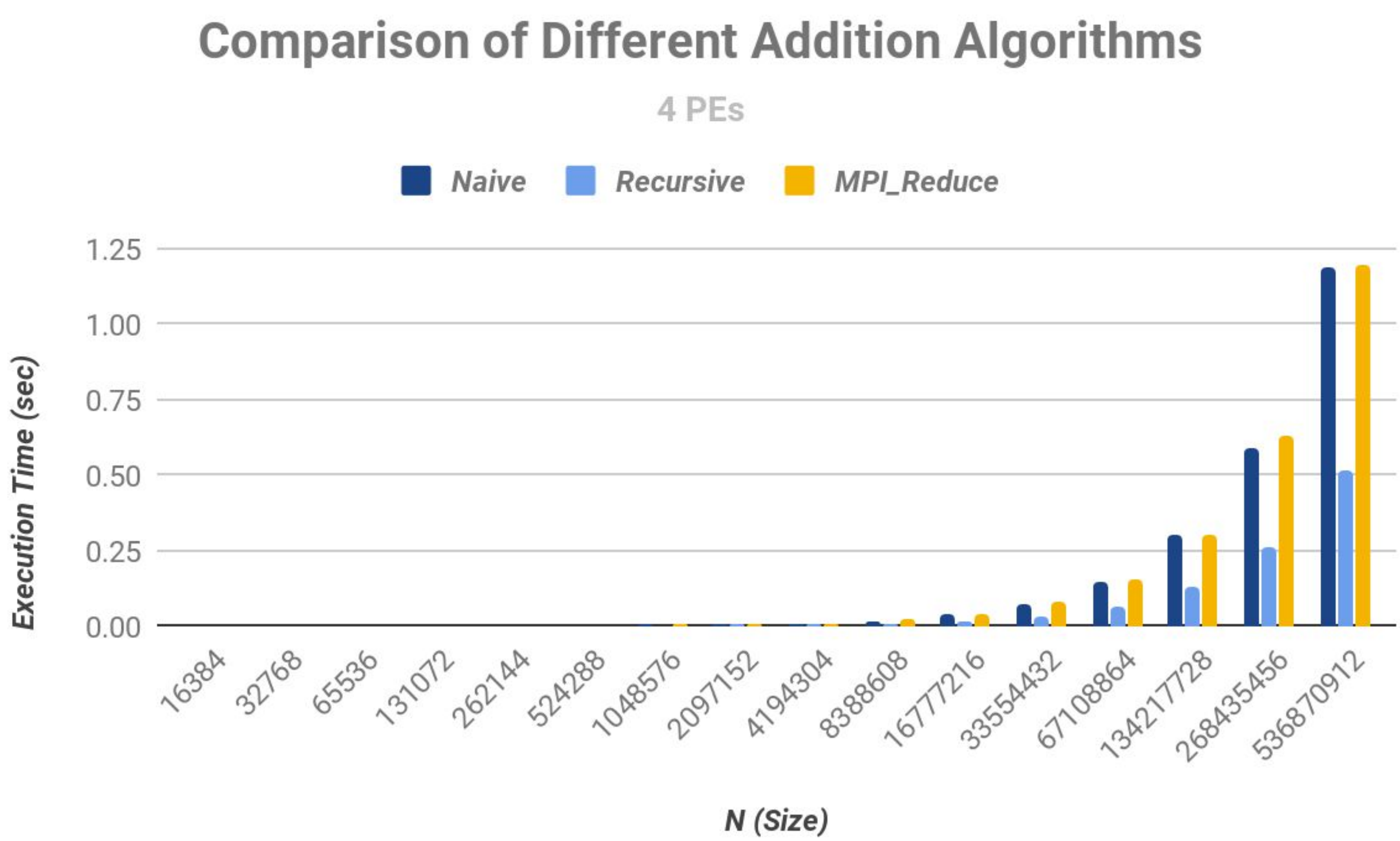
Size	Naive	Recursive	MPI_Reduce
16384	4.88E-05	4.71E-05	6.64E-05
32768	6.26E-05	6.23E-05	7.62E-05
65536	0.0001143	0.0001148	0.0001382
131072	0.0002319	0.0002361	0.0002513
262144	0.0005612	0.0005384	0.0005902
524288	0.00094	0.0009366	0.0009848
1048576	0.0017957	0.0017998	0.0017382
2097152	0.0036845	0.0034984	0.0034298
4194304	0.0067282	0.0065276	0.006687
8388608	0.0144151	0.0132142	0.0134266
16777216	0.025950333333333	0.025850666666667	0.025854333333333
33554432	0.051859	0.051747666666667	0.051652666666667
67108864	0.103662	0.103233333333333	0.103311333333333
134217728	0.207275	0.206592666666667	0.206695333333333
268435456	0.414490333333333	0.413236333333333	0.413548666666667
536870912	0.828969666666666	0.826194333333333	0.826867666666666



2. Using 4 PEs :

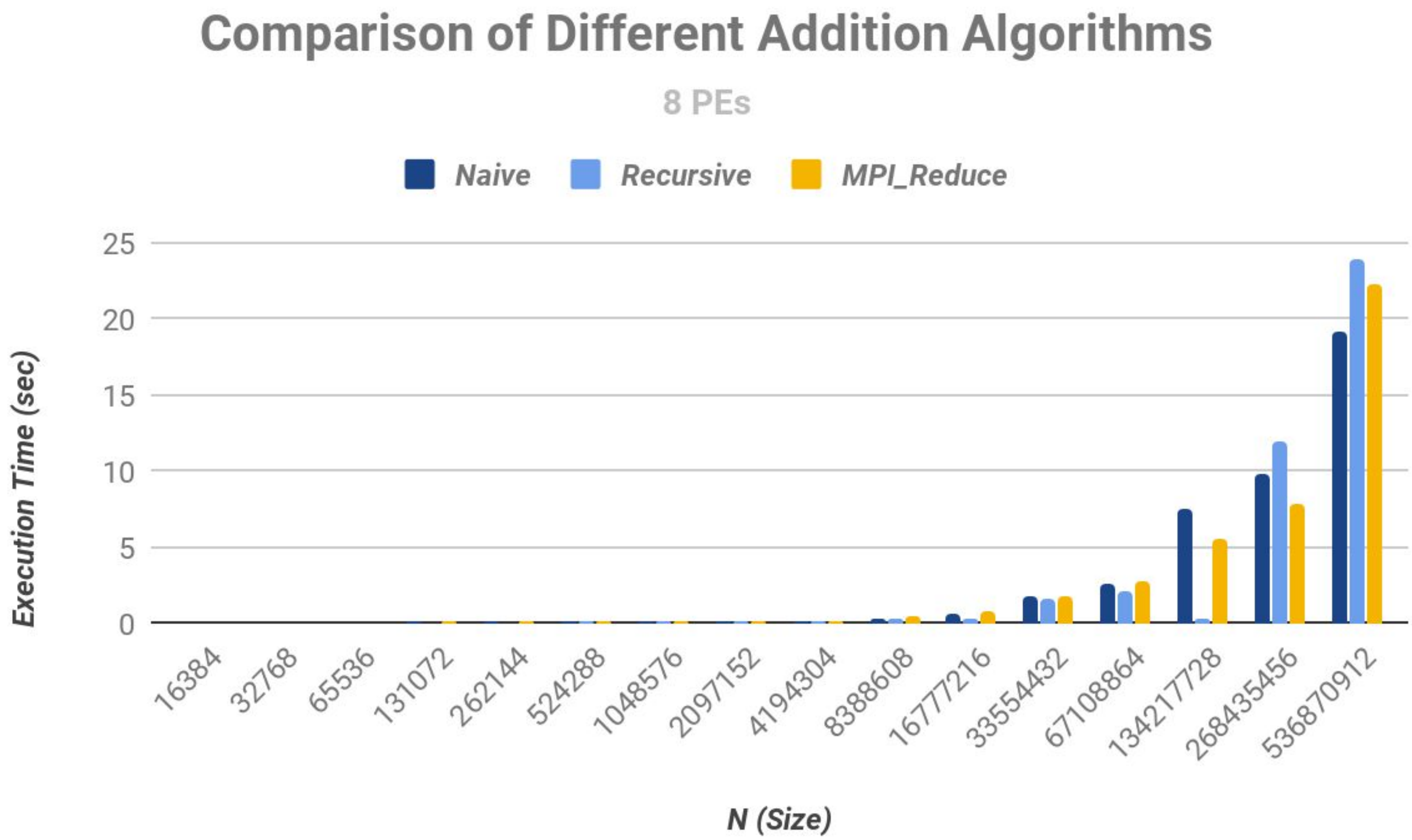
Size	Naive	Recursive	MPI_Reduce
16384	4.22E-05	6.18E-05	0.0001196
32768	0.0002983	6.77E-05	9.92E-05
65536	0.0004554	9.44E-05	0.000307
131072	0.0005396	0.00019	0.000543
262144	0.0009877	0.0006173	0.0008069
524288	0.0013642	0.0009127	0.0010791
1048576	0.0023475	0.0012678	0.0023437
2097152	0.0045709	0.0023522	0.0047969

4194304	0.0093651	0.0042822	0.0095758
8388608	0.0185459	0.0083094	0.0189615
16777216	0.036737333333333	0.016291	0.037569666666667
33554432	0.073483	0.032346333333333	0.078657333333333
67108864	0.149285	0.06432	0.157021333333333
134217728	0.298021333333333	0.128403	0.300406666666667
268435456	0.592301333333333	0.2577	0.626952333333333
536870912	1.18435633333333	0.514624333333333	1.19726133333333



3. Using 8 PEs :

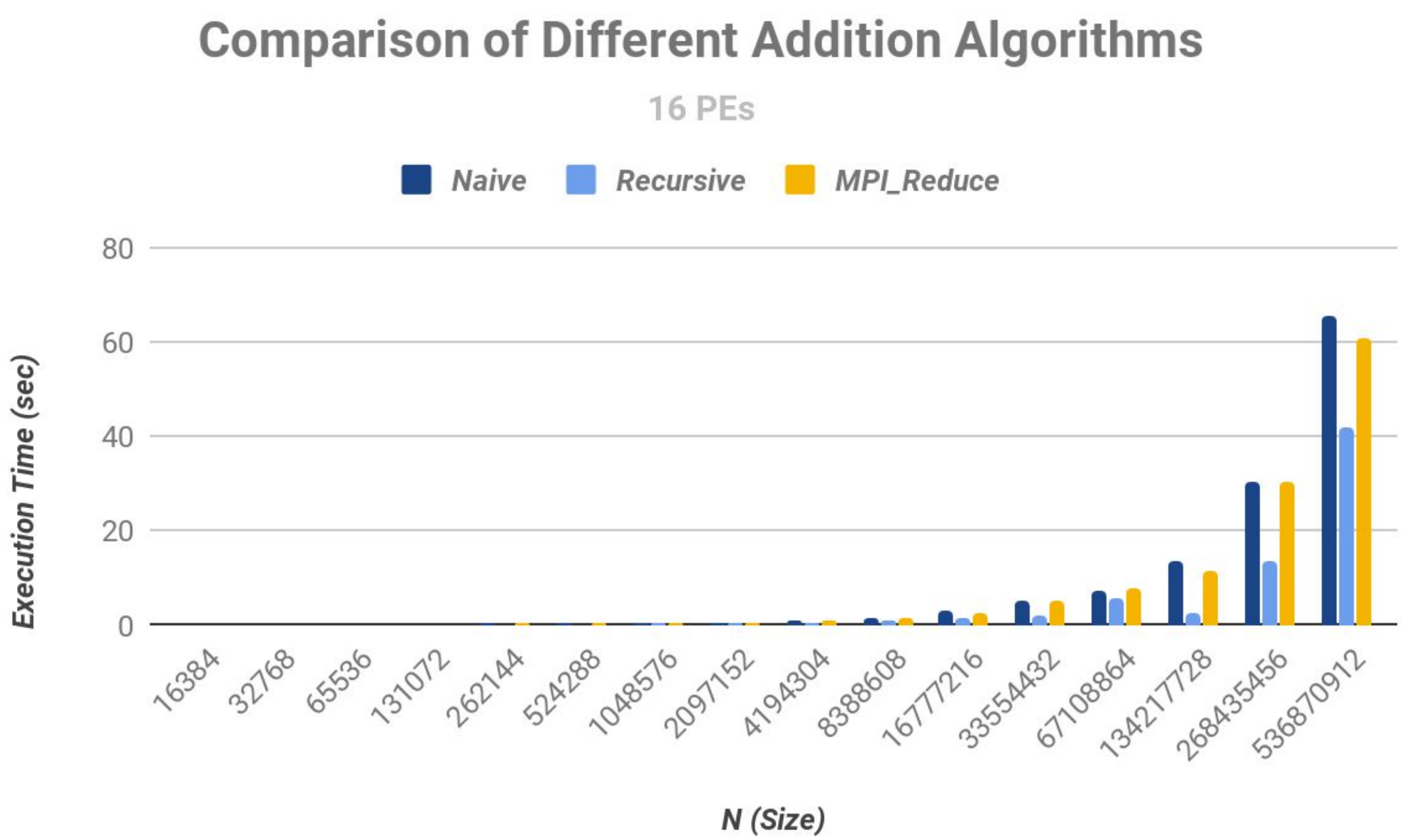
Size	Naive	Recursive	MPI_Reduce
16384	0.0165203	0.0293351	0.0243278
32768	0.0251192	0.0290242	0.022982
65536	0.0221907	0.039446	0.0243137
131072	0.0654992	0.0458373	0.0779505
262144	0.0758046	0.0441269	0.0756427
524288	0.0544796	0.0491985	0.0756757
1048576	0.0798632	0.0986791	0.1096879
2097152	0.1025362	0.0873132	0.1188049
4194304	0.1508483	0.1341912	0.1952913
8388608	0.3647934	0.3355703	0.3792638
16777216	0.683318	0.303059666666667	0.754793333333333
33554432	1.792066	1.63174533333333	1.802101
67108864	2.66656533333333	2.15018566666667	2.759532
134217728	7.43782	0.277295333333333	5.52239233333333
268435456	9.822782	11.8591996666667	7.85536166666667
536870912	19.073651	23.9431146666667	22.282449



4. Using 16 PEs :

Size	Naive	Recursive	MPI_Reduce
16384	0.0335505	0.0896714	0.0540944
32768	0.0310624	0.0891196	0.0608533
65536	0.0307123	0.0846621	0.0582272
131072	0.0365996	0.1198436	0.0648739
262144	0.3045195	0.1304432	0.3667313
524288	0.3160244	0.1361906	0.3315552
1048576	0.30018	0.1842294	0.3531471
2097152	0.6157985	0.2762402	0.6345363

4194304	0.9218865	0.3831931	1.089952
8388608	1.6584083	0.7320114	1.7182964
16777216	3.05105566666667	1.37290633333333	2.487336
33554432	5.262478	1.929877	5.01835133333333
67108864	7.24301833333333	5.43546733333333	7.64733366666667
134217728	13.2798516666667	2.766041	11.5052596666667
268435456	30.2539623333333	13.2957993333333	30.4286976666667
536870912	65.3617066666667	41.8135663333333	60.758743

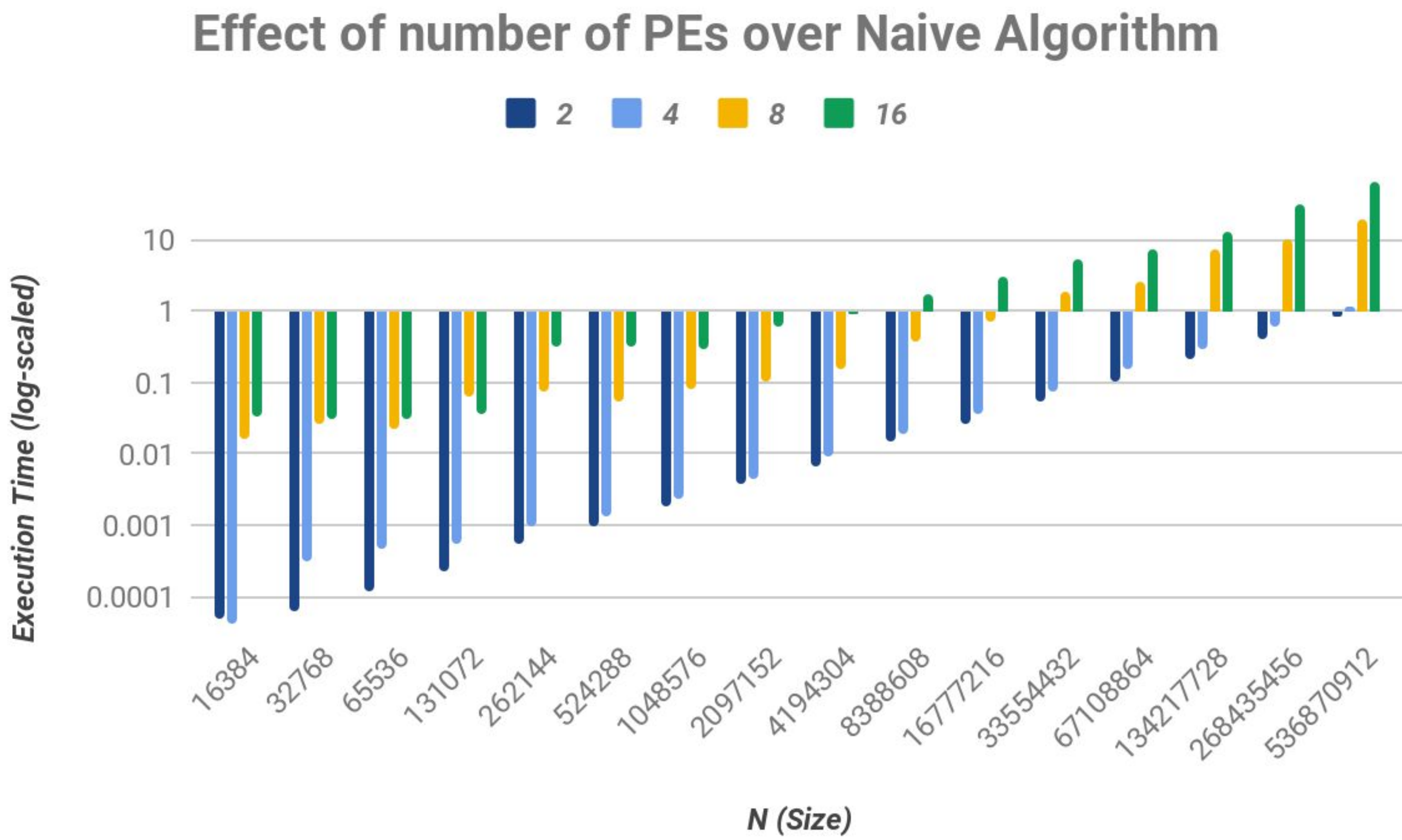


Observations :-

- The Recursive algorithm performed exceptionally better than the other two in case of 4 and 16 PEs. This can be attributed to the fact that the system was dual core and thus had 4 threads in total.
- In each of the above cases, Naive and MPI_Reduce algorithms were observed to be taking approx. same time with the Naive one performing slightly better in case of 4 PE, but a bit worse in case of 16 PE with increasing size.
- On the other hand, it was observed that the Recursive algorithm performed worst in case of 8 PEs with increasing size, with the Naive algo performing best among them.
- In case of 2 PEs, the Naive algorithm performed a bit worse than the rest with the other two performing approx. at the same level.
- There were some slight exceptions in the middle possibly due to unexpected thread switching.

Effect of number of PEs over Naive Algorithm :-

Size	2	4	8	16
16384	4.88E-05	4.22E-05	0.0165203	0.0335505
32768	6.26E-05	0.0002983	0.0251192	0.0310624
65536	0.0001143	0.0004554	0.0221907	0.0307123
131072	0.0002319	0.0005396	0.0654992	0.0365996
262144	0.0005612	0.0009877	0.0758046	0.3045195
524288	0.00094	0.0013642	0.0544796	0.3160244
1048576	0.0017957	0.0023475	0.0798632	0.30018
2097152	0.0036845	0.0045709	0.1025362	0.6157985
4194304	0.0067282	0.0093651	0.1508483	0.9218865
8388608	0.0144151	0.0185459	0.3647934	1.6584083
16777216	0.0259503333333333	0.0367373333333333	0.683318	3.05105566666667
33554432	0.051859	0.073483	1.792066	5.262478
67108864	0.103662	0.149285	2.66656533333333	7.24301833333333
134217728	0.207275	0.298021333333333	7.43782	13.2798516666667
268435456	0.414490333333333	0.592301333333333	9.822782	30.2539623333333
536870912	0.828969666666666	1.18435633333333	19.073651	65.3617066666667



Observations :-

- The Naive algorithm performed significantly better with 2 and 4 PEs in comparison to 8 and 16 threads, since the system was a dual core with 2 threads in each processor and thus better parallelism was observed.
- Performance was best with 2 PEs, followed by 4 PEs, and further greatly reduced with 8 PEs and worst with 16 PEs. The main time that was elapsed can be greatly attributed to thread-switching as observed with increase in PEs.

Problem Statement 2 :-

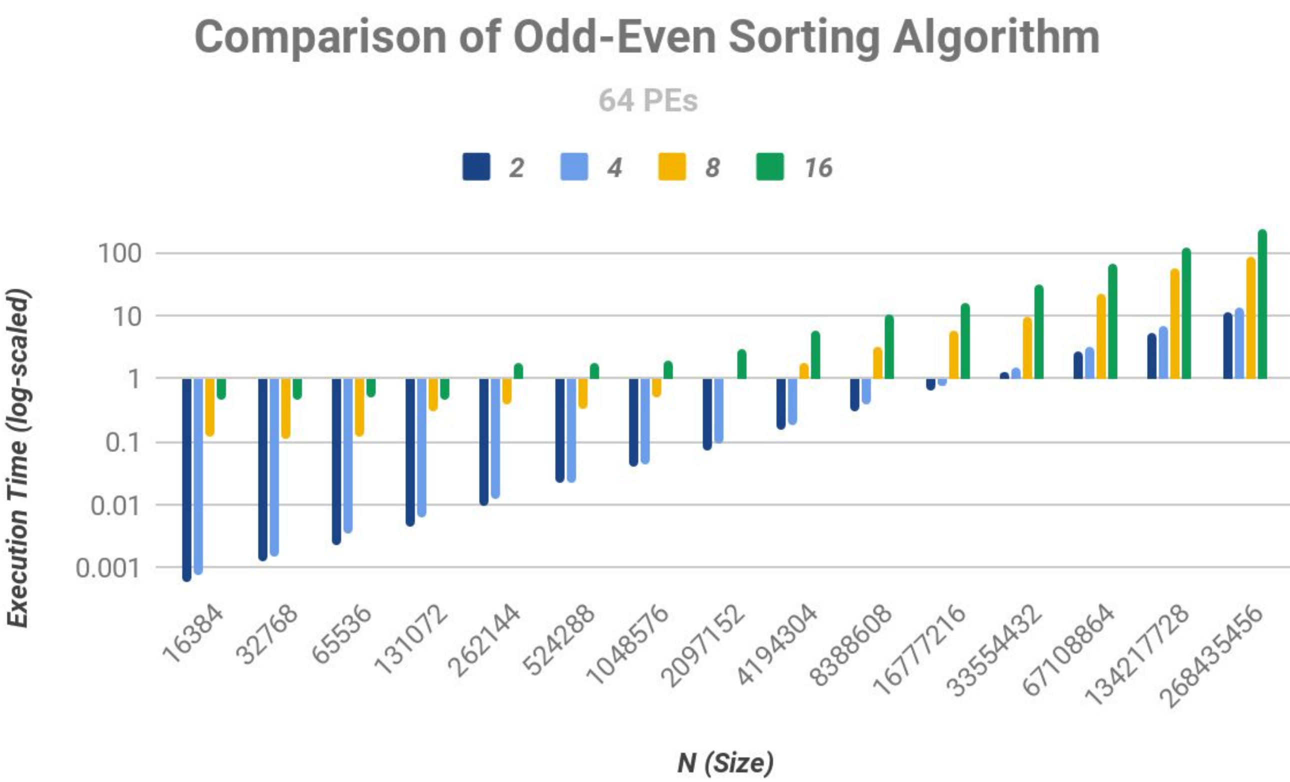
Implementation of the parallel ***Odd-Even Sorting algorithm*** using MPI programming.

Experiments :-

The task is to sort ***N*** numbers using the parallel odd-even sorting algorithm having ***p*** processing elements, thus giving each processor ***N/p*** elements to sort locally (initially) and further run the algorithm accordingly.
The value of *N* is varied from **2^14(16284)** upto **2^28(268435456)**.
The results are shown in the table and graph below. The values in the table are average values of 10 executions to avoid any unexpected behaviour introduced as a result of thread switching.

Size	2	4	8	16
16384	0.0006007	0.000754	0.1200431	0.4659713
32768	0.0012454	0.001459	0.1121843	0.4594921
65536	0.0023372	0.0035058	0.1163661	0.4799176
131072	0.0045537	0.0060901	0.3092717	0.4617221
262144	0.0095641	0.0120463	0.3852264	1.7329111
524288	0.0212409	0.0223659	0.3145488	1.8052235
1048576	0.0398044	0.0448955	0.4969631	1.8753551
2097152	0.0747058	0.0903885	1.0206497	3.0378519
4194304	0.1524034	0.1846829	1.7398297	5.8680199
8388608	0.305325333333333	0.375288	3.29774633333333	10.325815
16777216	0.624571666666667	0.757359666666667	5.65031533333333	16.0411156666667
33554432	1.29342233333333	1.54666433333333	9.292489	32.1871866666667

67108864	2.65438833333333	3.14701933333333	21.6402146666667	66.2461503333333
134217728	5.43754666666667	6.68883433333333	53.883429	115.382181
268435456	11.3676263333333	13.1708766666667	86.9731526666667	228.543194



Observations :-

- The time of execution was significantly better with 2 and 4 PEs in comparison to 8 and 16 threads, since the system was a dual core with 2 threads in each processor and thus better parallelism was observed.
- Performance was best with 2 PEs, followed by 4 PEs, and further greatly reduced with 8 PEs and worst with 16 PEs. The main time that was elapsed can be greatly attributed to thread-switching overhead as observed with increase in PEs.