Assignment 3

Inderpreet Singh Chera - 160101035

Different parallel implementations of adding numbers

Three different parallel implementations are tried and are compared in the following report.

System Settings

All tests were done on **Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz** processor. This computer is **dual core** where each core has **2 threads**. Also during each experiment it was insured that **no other applications** were running in the background, so that we don't have any biased readings.

Experiments

Implementation 1: Naive implementation to add n numbers.

1. P_o sends n/p items to each processor.
2. Each PE sends n/p items.
3. Each PE sends a partial sum to P_o.
4. P_o adds partial sum.

Implementation 2: Recursive partitioning.

Implementation 3: MPI API MPI_Reduce to compute the sum.

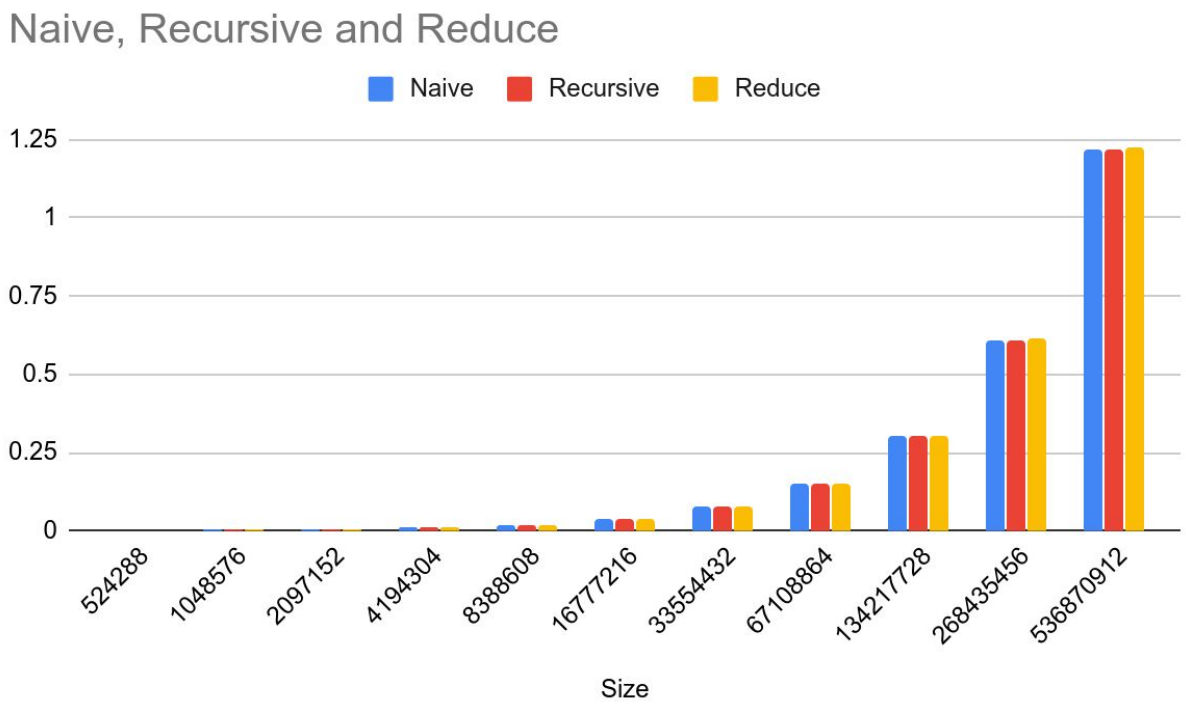
Results

We will be varying input size along with the number of threads to compare. Input size varies from $\mathbf{2^{17}}$ to $\mathbf{2^{29}}$. And the number of threads will be varied as 2, 4, 8 and 16.

N vs Time(in sec) using 2 Threads

Size	Naive	Recursive	Reduce
131072	0.000319	0.000436	0.000408
262144	0.000805	0.001104	0.000809

524288	0.001261	0.001513	0.001508
1048576	0.002406	0.002372	0.002411
2097152	0.006029	0.004767	0.004821
4194304	0.009606	0.009564	0.009583
8388608	0.019864	0.018982	0.019181
16777216	0.038042	0.037962	0.038311
33554432	0.07643	0.07616	0.076532
67108864	0.152524	0.15203	0.152976
134217728	0.305374	0.305578	0.306139
268435456	0.607599	0.607906	0.612472
536870912	1.21803	1.216854	1.225096

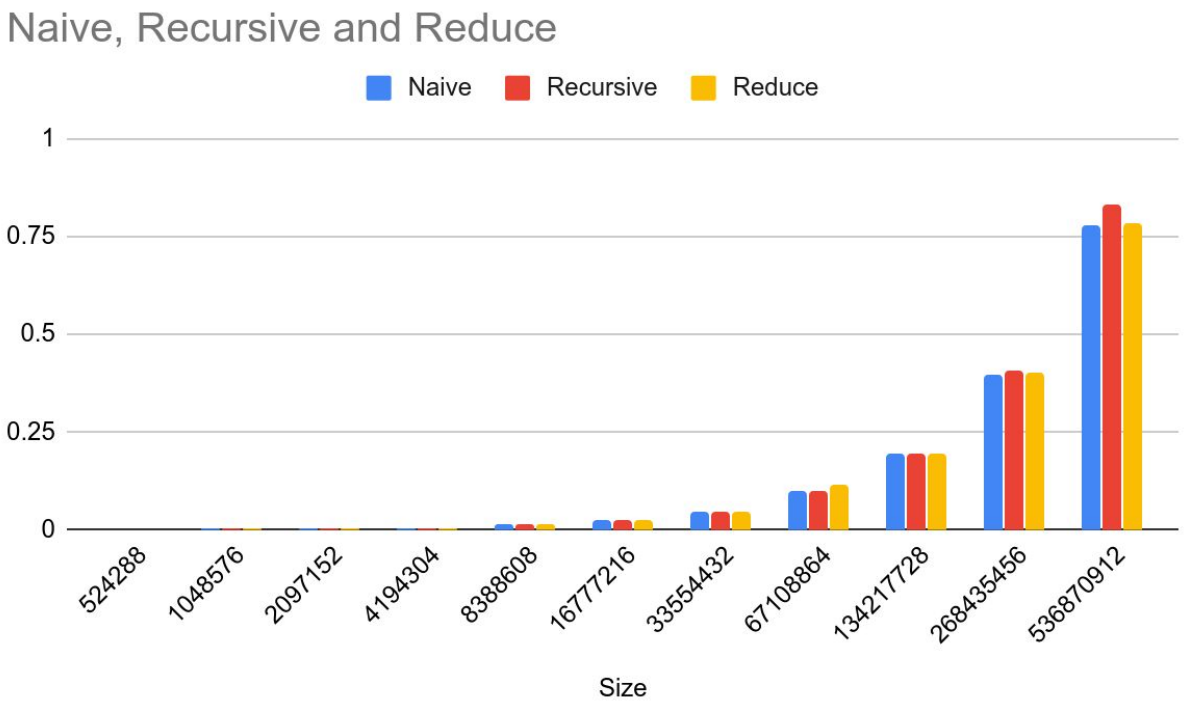


Observations:

We can observe that all the three implementations take almost the same time.There is no clear winner between the algorithms in this case.

N vs Time(in sec) using 4 Threads

Size	Naive	Recursive	Reduce
131072	0.000235	0.000488	0.000247
262144	0.000877	0.000462	0.000471
524288	0.001426	0.000825	0.000887
1048576	0.001548	0.001568	0.001598
2097152	0.003249	0.00321	0.003197
4194304	0.006205	0.006193	0.006202
8388608	0.012727	0.012275	0.012272
16777216	0.024572	0.024327	0.02448
33554432	0.048706	0.049145	0.048725
67108864	0.097857	0.097947	0.114488
134217728	0.197602	0.194183	0.193117
268435456	0.399318	0.407124	0.401727
536870912	0.780325	0.832949	0.783422



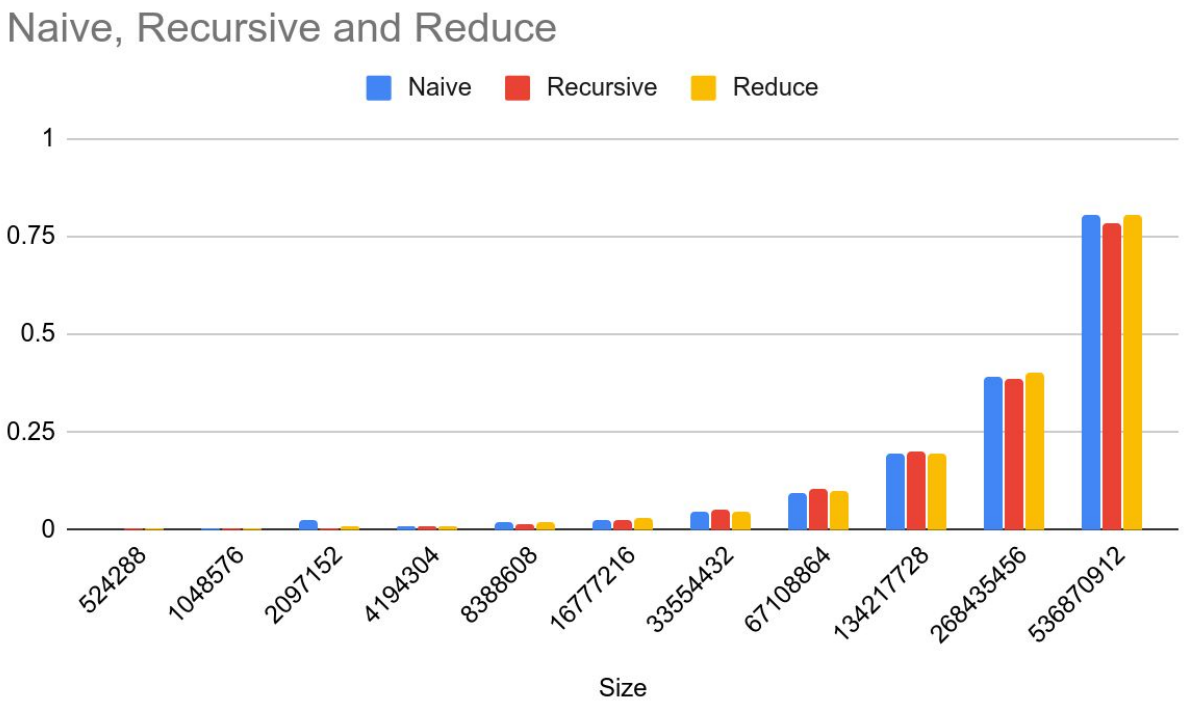
Observations:

No generic trend can be observed from the above graph. There are some n where we can see that recursive implementation takes a bit more time than other two implementations. Similarly there is no generic trend of time between naive and MPI_Reduce implementations.

N vs Time(in sec) using 8 Threads

Size	Naive	Recursive	Reduce
131072	0.000743	0.000658	0.000404
262144	0.000668	0.000655	0.000846
524288	0.001338	0.001513	0.00189
1048576	0.002516	0.002645	0.003382
2097152	0.02369	0.003328	0.010039
4194304	0.009302	0.00949	0.008833
8388608	0.019176	0.014954	0.019102
16777216	0.024854	0.024644	0.030498

33554432	0.048583	0.050128	0.048696
67108864	0.096525	0.102409	0.097231
134217728	0.196909	0.199087	0.196105
268435456	0.391312	0.386506	0.403606
536870912	0.80408	0.785891	0.803099



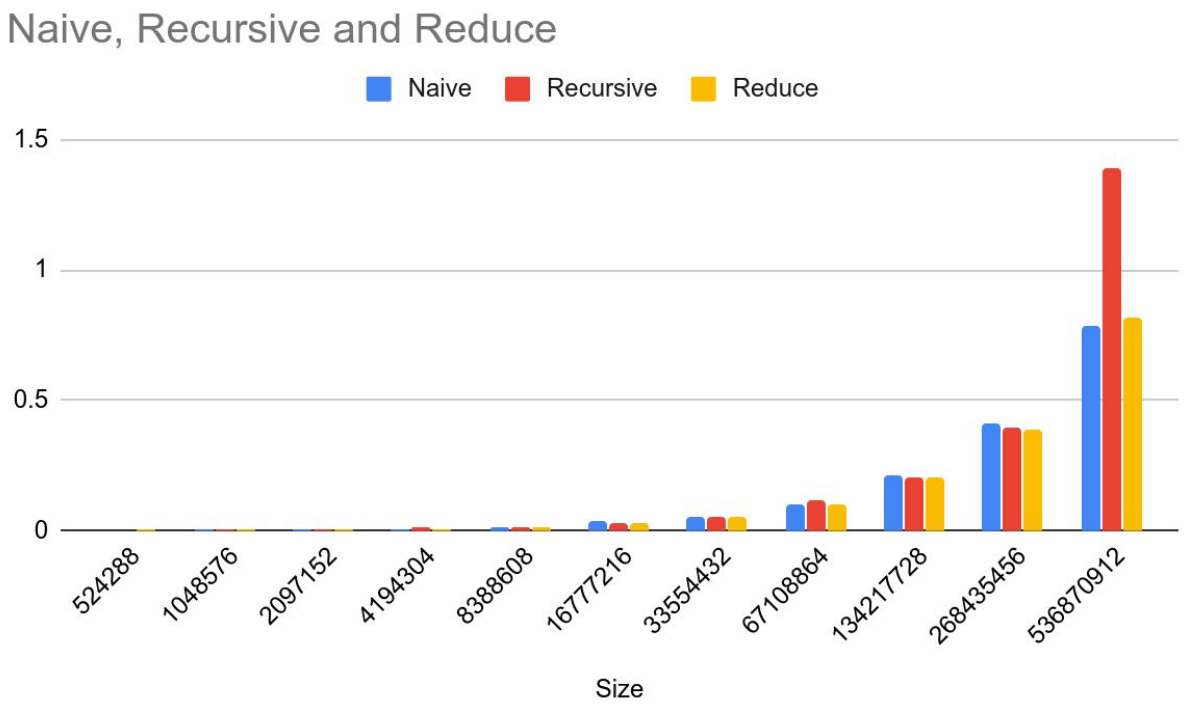
Observations:

We can observe that Naive algorithm takes more time than the rest of the algorithms in most instances. The recursive implementation takes less time than other two implementations most of the times.

N vs Time(in sec) using 16 threads

Size	Naive	Recursive	Reduce
131072	0.001302	0.000892	0.001237
262144	0.001387	0.000975	0.001231

524288	0.002048	0.001944	0.002376
1048576	0.003146	0.00297	0.003928
2097152	0.006385	0.00419	0.005237
4194304	0.009004	0.011226	0.010165
8388608	0.015921	0.013662	0.012953
16777216	0.037891	0.030622	0.029189
33554432	0.050081	0.049993	0.056077
67108864	0.101246	0.117038	0.101149
134217728	0.21015	0.20799	0.206037
268435456	0.408255	0.395883	0.390376
536870912	0.786714	1.392167	0.81485

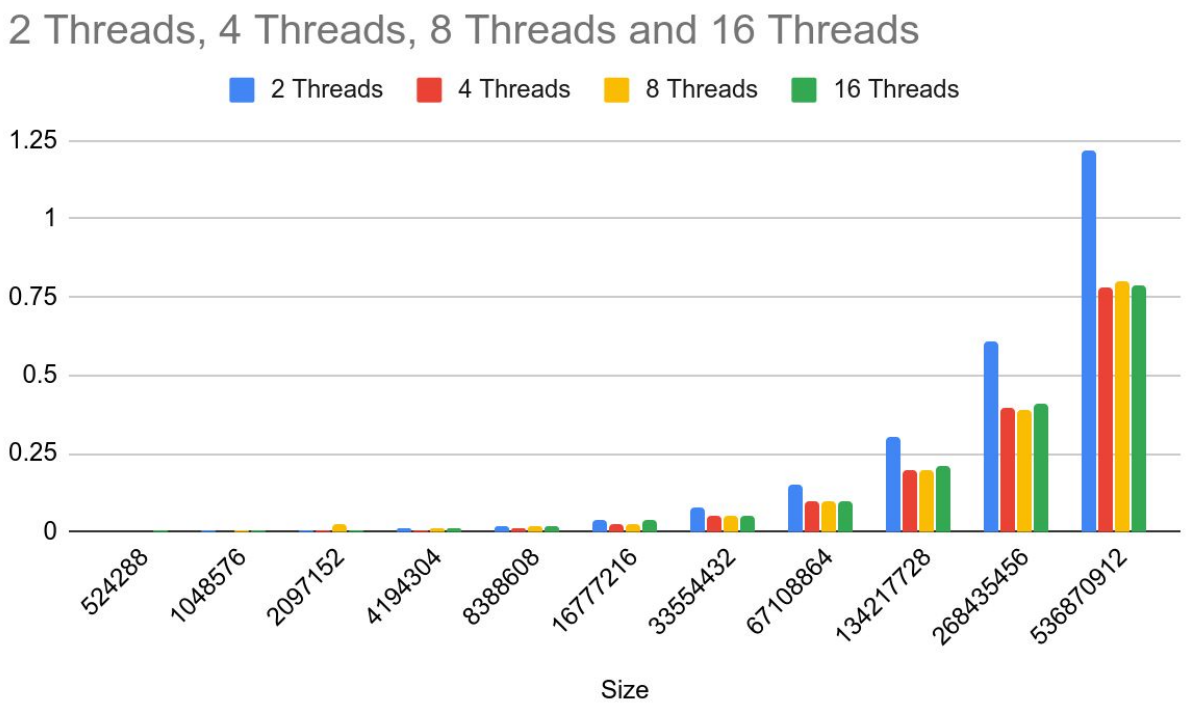


Observations:

Here recursive implementation takes more time than Naive and MPI_Reduce implementations when n is very large. And in most cases, MPI_Reduce implementation takes less time than Naive implementations.

N vs No. of threads for Naive implementation

Size	2 Threads	4 Threads	8 Threads	16 Threads
131072	0.000319	0.000235	0.000743	0.001302
262144	0.000805	0.000877	0.000668	0.001387
524288	0.001261	0.001426	0.001338	0.002048
1048576	0.002406	0.001548	0.002516	0.003146
2097152	0.006029	0.003249	0.02369	0.006385
4194304	0.009606	0.006205	0.009302	0.009004
8388608	0.019864	0.012727	0.019176	0.015921
16777216	0.038042	0.024572	0.024854	0.037891
33554432	0.07643	0.048706	0.048583	0.050081
67108864	0.152524	0.097857	0.096525	0.101246
134217728	0.305374	0.197602	0.196909	0.21015
268435456	0.607599	0.399318	0.391312	0.408255
536870912	1.21803	0.780325	0.80408	0.786714



Observations:

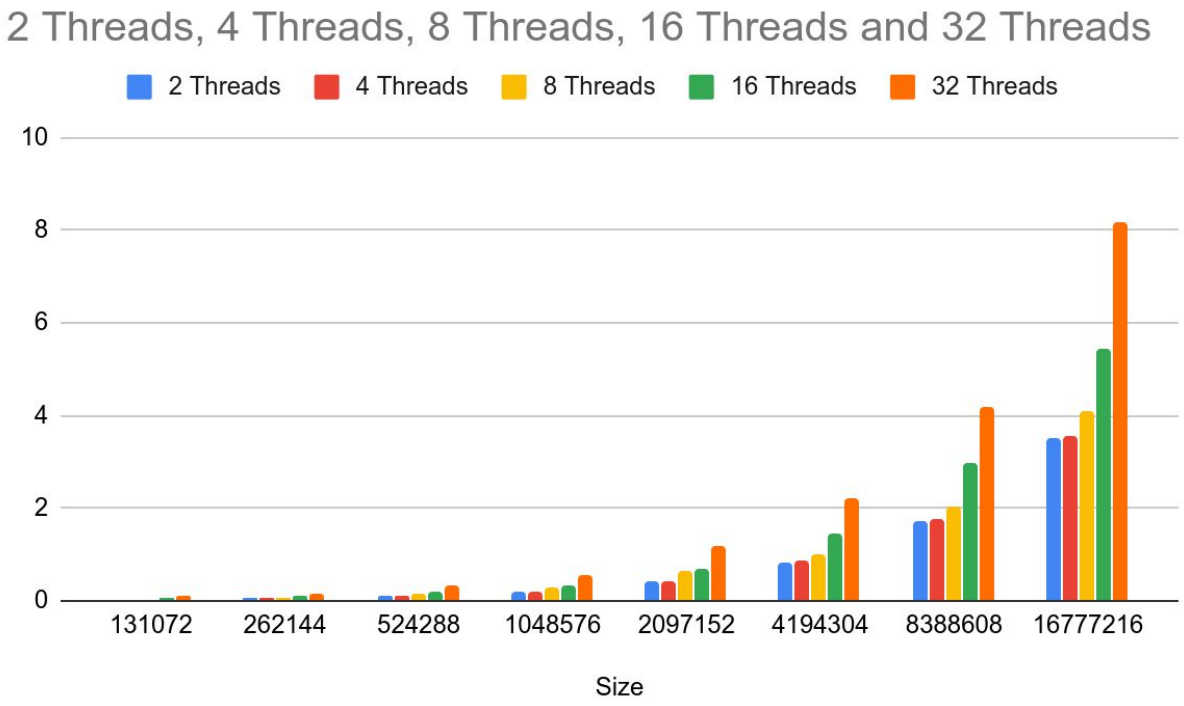
The general trend is that time should decrease on increasing the number of processors which happens, but then time requires remains almost stagnant. This maybe because my processors support maximum 4 threads in parallel. Now, if we increase the number of threads beyond that then thread context switching time also comes into play and may increase the total time by a small amount. We have observed that after 4 threads, time hasn't decreased drastically which happened when we increased threads from 2 to 4.

Odd Even Sorting:

Different values of n tried are $\mathbf{2^{17}}$ to $\mathbf{2^{25}}$ and number of threads tried are **2,4,8,16** and **32**.

Size	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads
131072	0.02262	0.025179	0.039622	0.058954	0.116084
262144	0.05974	0.048886	0.081275	0.097263	0.172117
524288	0.09701	0.099508	0.149919	0.212645	0.323227
1048576	0.198089	0.20515	0.292639	0.355571	0.564787
2097152	0.406528	0.419917	0.64568	0.68422	1.184755
4194304	0.833918	0.863701	1.016915	1.462978	2.23145

8388608	1.718373	1.753384	2.021018	2.991224	4.167669
16777216	3.529526	3.566595	4.118968	5.429082	8.154064
33554432	7.24952	7.309651	8.361685	15.501664	163.449615



Observations:

We can see that from 2 to 4 processors time decreased a bit, but after that on increasing the number of threads, time goes on increasing. Probable reason for this can be that because I have a quad core processor, and hence after that one processor is going to handle more than one thread and hence increases the total time because now context switch time also gets included in it.