Paleti Krishnasai
CED18I039

HPC LAB - Sum of N Numbers MPI

Name: Paleti Krishnasai
Roll No: CED18I039
Programming Environment: MPI
Problem:Sum of N numbers

Hardware Configuration:
PU NAME: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
Number of Sockets: 1
Cores per Socket: 4
Threads per core: 2
L1d cache: 128 KiB
L1i cache: 128 KiB
L2 cache: 1 MiB
L3 cache: 8 MiB

```
paleti@paleti-Lenovo-ldeapad-330-15ICH:~$ likwid-topology
--------------------------------------------------------------------------------
CPU name:       Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
CPU type:       Intel Coffeelake processor
CPU stepping:   10
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:                1
Cores per socket:       4
Threads per core:       2
--------------------------------------------------------------------------------
HWThread        Thread          Core            Socket          Available
0               0               0               0               *
1               0               1               0               *
2               0               2               0               *
3               0               3               0               *
4               1               0               0               *
5               1               1               0               *
6               1               2               0               *
7               1               3               0               *
--------------------------------------------------------------------------------
Socket 0:               ( 0 4 1 5 2 6 3 7 )
--------------------------------------------------------------------------------
********************************************************************************
Cache Topology
********************************************************************************
Level:                  1
Size:                   32 kB
Cache groups:           ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
--------------------------------------------------------------------------------
Level:                  2
Size:                   256 kB
Cache groups:           ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
--------------------------------------------------------------------------------
Level:                  3
Size:                   8 MB
Cache groups:           ( 0 4 1 5 2 6 3 7 )
--------------------------------------------------------------------------------
********************************************************************************
NUMA Topology
********************************************************************************
NUMA domains:           1
--------------------------------------------------------------------------------
Domain:                 0
Processors:             ( 0 1 2 3 4 5 6 7 )
Distances:              10
Free memory:            3546.2 MB
Total memory:           7831.84 MB
--------------------------------------------------------------------------------
```

```
**************************************************
Graphical Topology
**************************************************
Socket 0:
+-------------------------------------------------+
| +---------+ +---------+ +---------+ +---------+ |
| |  0 4    | |  1 5    | |  2 6    | |  3 7    | |
| +---------+ +---------+ +---------+ +---------+ |
| +---------+ +---------+ +---------+ +---------+ |
| |  32 kB  | |  32 kB  | |  32 kB  | |  32 kB  | |
| +---------+ +---------+ +---------+ +---------+ |
| +---------+ +---------+ +---------+ +---------+ |
| | 256 kB  | | 256 kB  | | 256 kB  | | 256 kB  | |
| +---------+ +---------+ +---------+ +---------+ |
| +---------------------------------------------+ |
| |                    8 MB                     | |
| +---------------------------------------------+ |
+-------------------------------------------------+
```

No of nodes: 12 (4 for each as written in the machine file).

Paleti Krishnasai
CED18I039

Serial Code:

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
int main(int argc, char *argv[])
{
    double start,end;
    MPI_Init(&argc,&argv);
    start=MPI_Wtime();
    int i = 1, n = 100000;
    double a[n], sum = 0;
    for(i=0;i<n;i++)
    {
        a[i] = (i+1)*5.658;
        sum += a[i]
    }
    end=MPI_Wtime();
    printf("\nTime= %f",end-start);
    return 0;
}
```

Paleti Krishnasai
CED18I039

Parallel Code : [MPI_Reduce]

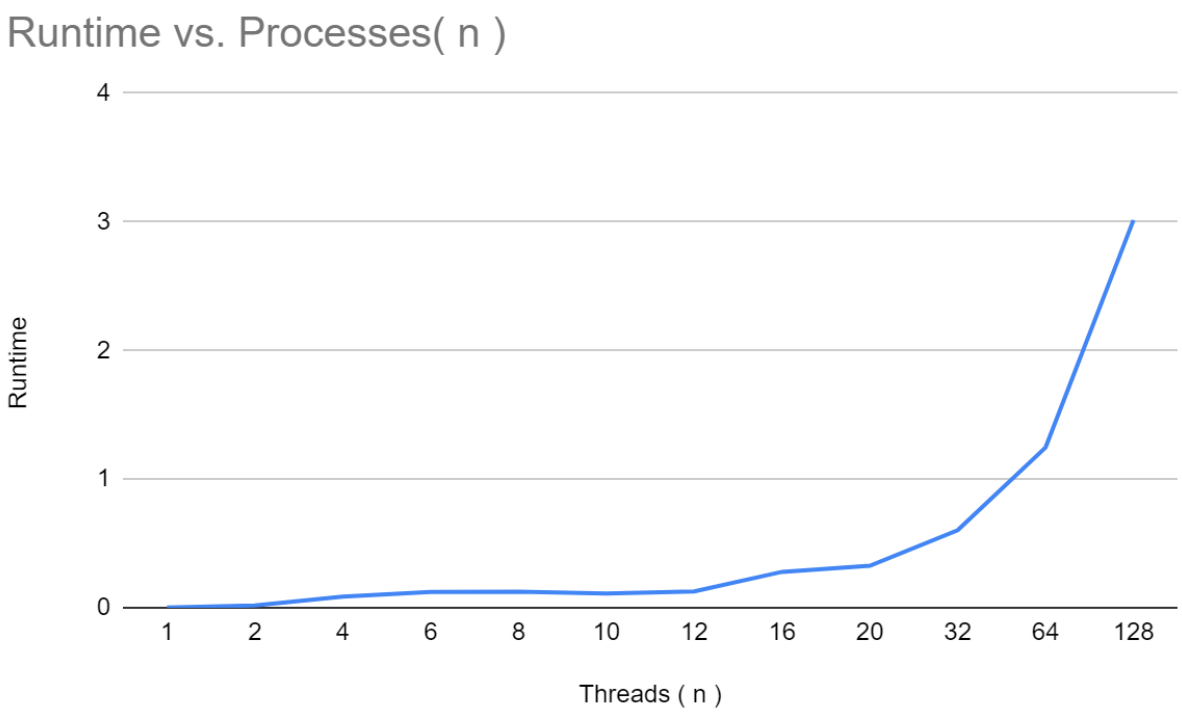
```
#include <mpi.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define ARRAY_SIZE 100000

int main(int argc, char *argv[])
{
int myid, numprocs, s;
double vector1[ARRAY_SIZE + 50], sum = 0, vector1Receive[ARRAY_SIZE + 50],
fSum =
0;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
double start = MPI_Wtime();
for (int i = 0; i < ARRAY_SIZE; i++)
vector1[i] = i + 1;
if (myid == 0)
s = (int)floor(ARRAY_SIZE / numprocs);
MPI_Bcast(&s, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Scatter(vector1, s, MPI_DOUBLE, vector1Receive, s, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
for (int i = 0; i < s; i++)
sum += vector1Receive[i];
MPI_Reduce(&sum, &fSum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
if (myid == 0)
{
for (int i = s * numprocs; i < ARRAY_SIZE; i++)
fSum += vector1[i];
printf("%f\n", fSum);
double end = MPI_Wtime();
printf("%f\n", end - start);
}
MPI_Finalize();
}
```

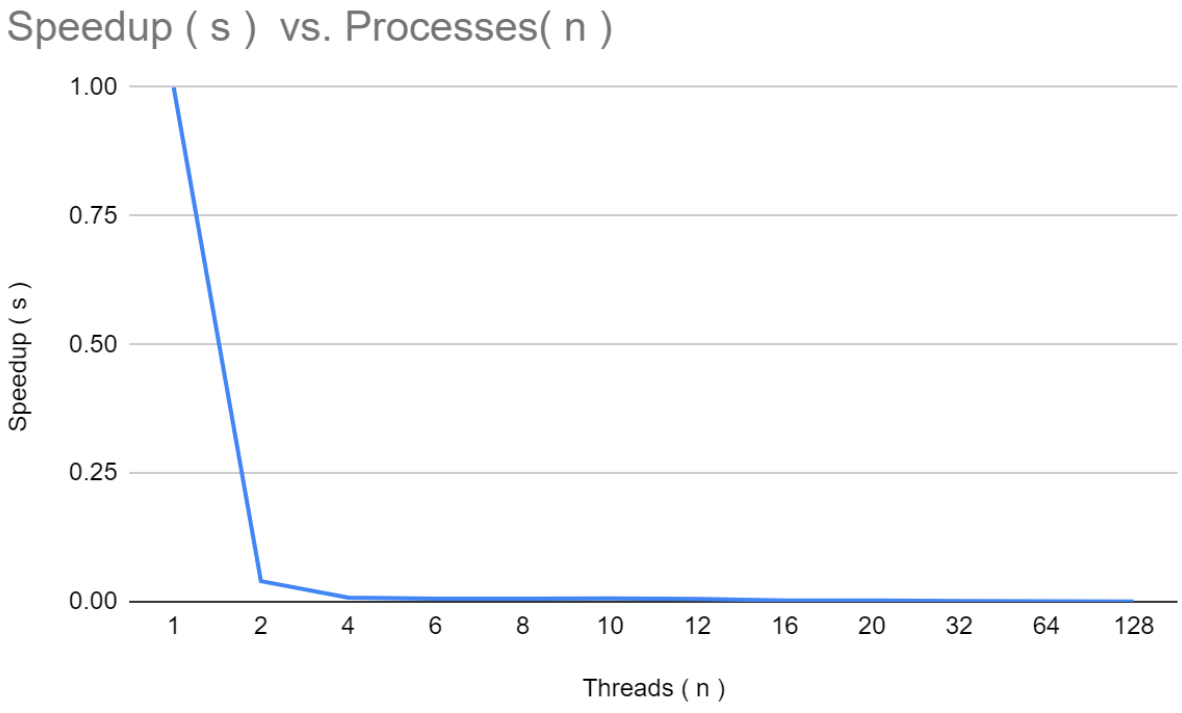
Paleti Krishnasai
CED18I039

Observations : [MPI_Reduce]

Processes (n)	Runtime	Speedup (s)
1	0.000698	1
2	0.017489	0.03991080107
4	0.086921	0.00803028037
6	0.121665	0.005737064891
8	0.124371	0.005612240796
10	0.110482	0.006317771221
12	0.126299	0.005526567906
16	0.278594	0.002505438021
20	0.325962	0.0021413539
32	0.602774	0.001157979608
64	1.246792	0.0005598367651
128	3.013827	0.0002315992258

Paleti Krishnasai
CED18I039





Paleti Krishnasai
CED18I039

Parallel Code : [Without MPI_Reduce]

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

// size of array
#define n 10000

int a[10000];

// Temporary array for slave process
int a2[100000000];

int main(int argc, char* argv[])
{
    double start,end;
    int pid, np,
        elements_per_process,
        n_elements_recieved;
    // np -> no. of processes
    // pid -> process id

    MPI_Status status;

    // Creation of parallel processes
    MPI_Init(&argc, &argv);
    start=MPI_Wtime();
     for (int i = 0; i < 10000; i++)
            a[i] = (i + 1)*8.754;
    // find out process ID,
    // and how many processes were started
    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    // master process
    if (pid == 0) {
        int index, i;
        elements_per_process = n / np;
```

```
        // check if more than 1 processes are run
        if (np > 1) {
            // distributes the portion of array
            // to child processes to calculate
            // their partial sums
            for (i = 1; i < np - 1; i++) {
                index = i * elements_per_process;

                MPI_Send(&elements_per_process,
                         1, MPI_INT, i, 0,
                         MPI_COMM_WORLD);
                MPI_Send(&a[index],
                         elements_per_process,
                         MPI_INT, i, 0,
                         MPI_COMM_WORLD);
            }

            // last process adds remaining elements
            index = i * elements_per_process;
            int elements_left = n - index;

            MPI_Send(&elements_left,
                     1, MPI_INT,
                     i, 0,
                     MPI_COMM_WORLD);
            MPI_Send(&a[index],
                     elements_left,
                     MPI_INT, i, 0,
                     MPI_COMM_WORLD);
        }

        // master process add its own sub array
        int sum = 0;
        for (i = 0; i < elements_per_process; i++)
            sum += a[i];

        // collects partial sums from other processes
        int tmp;
        for (i = 1; i < np; i++) {
            MPI_Recv(&tmp, 1, MPI_INT,
```

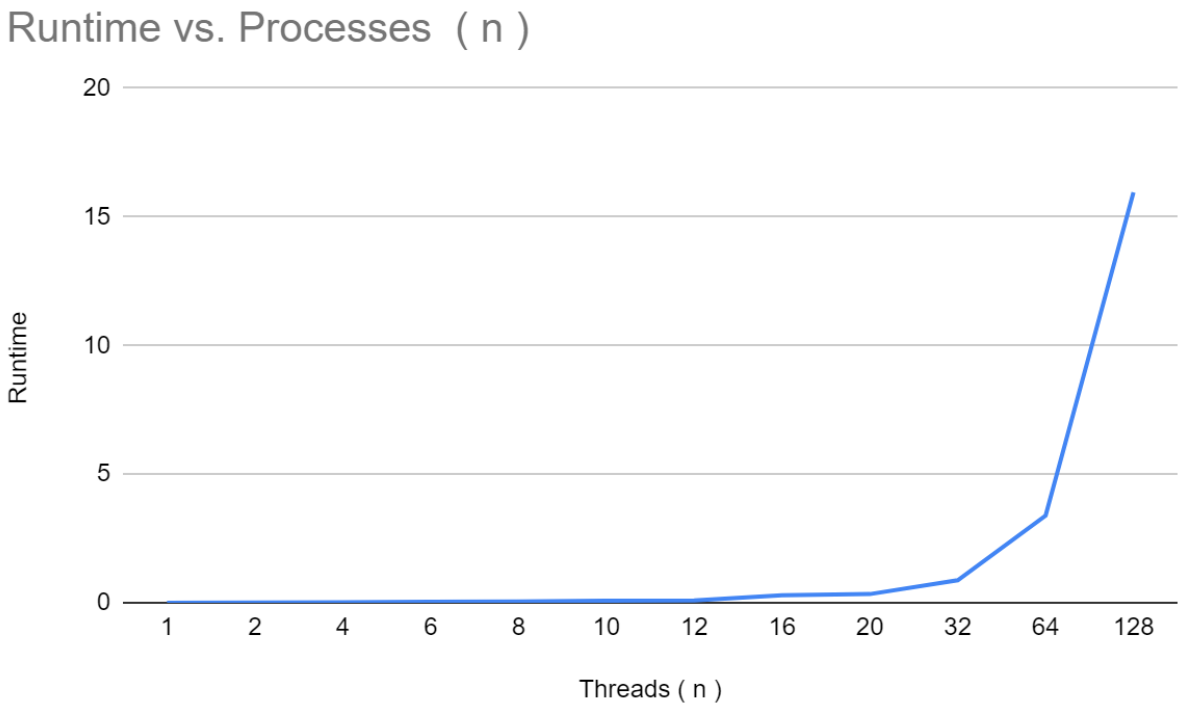
```
                    MPI_ANY_SOURCE, 0,
                    MPI_COMM_WORLD,
                    &status);
            int sender = status.MPI_SOURCE;

            sum += tmp;
        }

        // prints the final sum of array
        printf("Sum of array is : %d\n", sum);
         end=MPI_Wtime();
        printf("\nTime_hello= %f",end-start);
    }
    // slave processes
    else {
        MPI_Recv(&n_elements_recieved,
                 1, MPI_INT, 0, 0,
                 MPI_COMM_WORLD,
                 &status);

        // stores the received array segment
        // in local array a2
        MPI_Recv(&a2, n_elements_recieved,
                 MPI_INT, 0, 0,
                 MPI_COMM_WORLD,
                 &status);

        // calculates its partial sum
        int partial_sum = 0;
        for (int i = 0; i < n_elements_recieved; i++)
            partial_sum += a2[i];

        // sends the partial sum to the root process
        MPI_Send(&partial_sum, 1, MPI_INT,
                 0, 0, MPI_COMM_WORLD);
    }
    // cleans up all MPI state before exit of process
    MPI_Finalize();
    return 0;
}
```

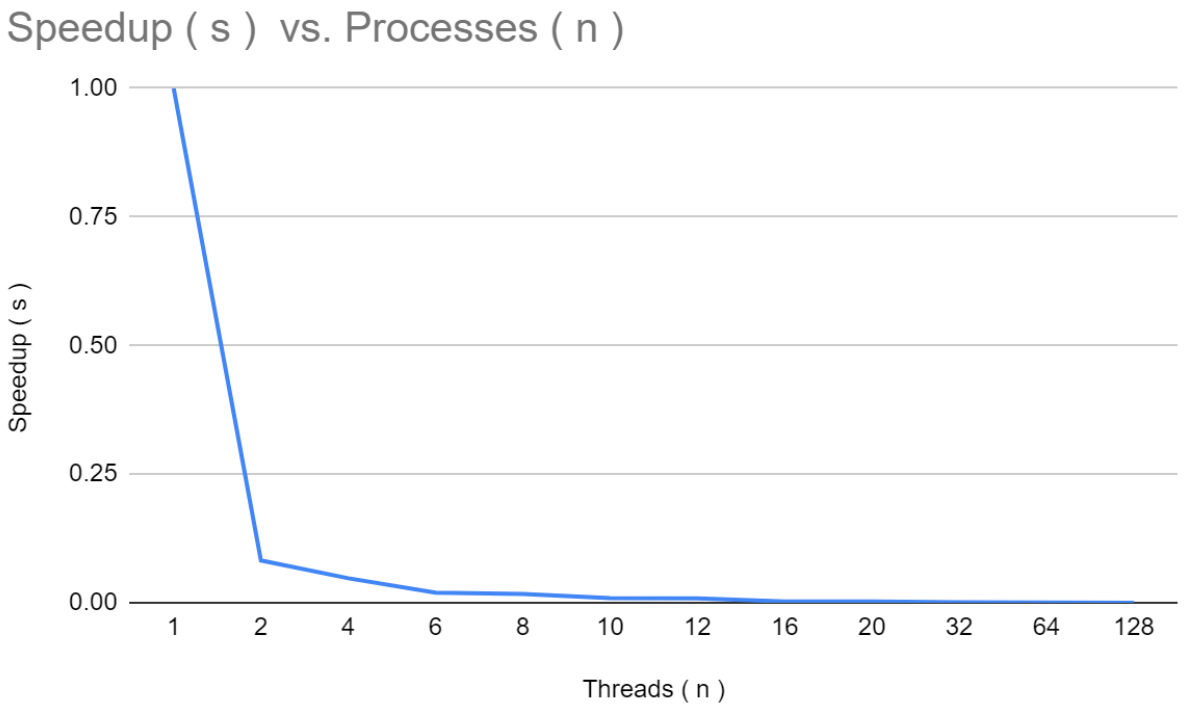
Paleti Krishnasai
CED18I039

Observations : [Without MPI_Reduce]

Processes (n)	Runtime	Speedup (s)
1	0.000698	1
2	0.008503	0.08208867459
4	0.014682	0.04754120692
6	0.036009	0.01938404288
8	0.04158	0.01678691679
10	0.078148	0.008931770487
12	0.082783	0.008431682833
16	0.290735	0.002400811736
20	0.334831	0.002084633741
32	0.873777	0.000798830823
64	3.385408	0.0002061789894
128	15.950274	0.00004376100373

Paleti Krishnasai
CED18I039





Paleti Krishnasai
CED18I039

Inference: (**Note**: Execution time, graph, and inference will be based on hardware configuration)

- Since MPI is a distributed memory architecture, the communication overhead between nodes causes the parallel code to run slower compared to serial code (running in 1 node or only in master).

.