Paleti Krishnasai
CED18I039

HPC LAB - Vector Multiplication MPI

Name: Paleti Krishnasai
Roll No: CED18I039
Programming Environment: MPI
Problem: Vector Multiplication
Date: 21-10-2021

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-ideapad-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 machine file).

Paleti Krishnasai
CED18I039

Serial Code:

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

#define VSIZE 1000
#define MASTER 0
#define FROM_MASTER 1
#define FROM_WORKER 2

//serial code
int main(int argc, char *argv[]){
        double start,end;
    MPI_Init(&argc, &argv);
        start=MPI_Wtime();
    int i;
    long double a[VSIZE], b[VSIZE], c[VSIZE];
    for (i = 0; i < VSIZE; i++){
                a[i] = i * 156.678;
                }
        for (i = 0; i < VSIZE; i++){
                b[i] = i * 2.0078;
                }
        for (i = 0; i < VSIZE; i++)
            c[i] = a[i] * b[i];
        printf("\nResultant Vector:\n");

        for (i = 0; i < VSIZE; i++)
            printf("\n%Lf +%Lf = %Lf   ", a[i],b[i],c[i]);

        printf("\nFinished.\n");
    end=MPI_Wtime();
    printf("\nTime= %f",end-start);
    return 0;
    }
```

Paleti Krishnasai
CED18I039

Parallel Code: [Point to Point]

```
#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>



#define VSIZE 1000

#define MASTER 0

#define FROM_MASTER 1

#define FROM_WORKER 2


int main(int argc, char *argv[])

{

    int numtasks, taskid, numworkers, source, dest, mtype, segment,
aveseg, extra, offset, i, j, k, rc;

    long double a[VSIZE], b[VSIZE], c[VSIZE];

    MPI_Status status;

    double start,end;



    MPI_Init(&argc, &argv);

    start=MPI_Wtime();

    MPI_Comm_rank(MPI_COMM_WORLD, &taskid);

    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
```

```
    if (numtasks < 2)

    {

        printf("Need atleast two MPI tasks. Quiting...\n");

        MPI_Abort(MPI_COMM_WORLD, rc);

        exit(1);

    }



    char pro_name[MPI_MAX_PROCESSOR_NAME];

    int name_len;

    MPI_Get_processor_name(pro_name,&name_len);



    printf("-From from  %s, rank %d, out of %d
processors\n",pro_name,taskid,numtasks);

    numworkers = numtasks - 1;



    //master task:



    if (taskid == MASTER)

    {

        for (i = 0; i < VSIZE; i++){
```

```
                a[i] = i * 156.678;

                }

        for (i = 0; i < VSIZE; i++){

                b[i] = i * 2.0078;

                }



        aveseg = VSIZE / numworkers;

        extra = VSIZE % numworkers;

        offset = 0;

        mtype = FROM_MASTER;

        for (dest = 1; dest <= numworkers; dest++)

        {

            segment = (dest <= extra) ? aveseg + 1 : aveseg;



            MPI_Send(&offset, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);

            MPI_Send(&segment, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);

            MPI_Send(&a[offset], segment, MPI_LONG_DOUBLE, dest, mtype,
MPI_COMM_WORLD);

            MPI_Send(&b[offset], segment, MPI_LONG_DOUBLE, dest, mtype,
MPI_COMM_WORLD);

            offset = offset + segment;
```

```
        }



        //receive from worker:



        mtype = FROM_WORKER;

        for (i = 1; i <= numworkers; i++)

        {

            source = i;

            MPI_Recv(&offset, 1, MPI_INT, source, mtype, MPI_COMM_WORLD,
&status);

            MPI_Recv(&segment, 1, MPI_INT, source, mtype, MPI_COMM_WORLD,
&status);

            MPI_Recv(&c[offset], segment, MPI_LONG_DOUBLE, source, mtype,
MPI_COMM_WORLD, &status);

        }



        printf("\nResultant Vector:\n");



        for (i = 0; i < VSIZE; i++)

            printf("\n%Lf +%Lf = %Lf   ", a[i],b[i],c[i]);



        printf("\nFinished.\n");
```

```
    end=MPI_Wtime();

    printf("\nTime= %f",end-start);

    }

    //Worker task:



    if (taskid > MASTER)

    {

        mtype = FROM_MASTER;

        MPI_Recv(&offset, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD,
&status);

        MPI_Recv(&segment, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD,
&status);

        MPI_Recv(&a, segment, MPI_LONG_DOUBLE, MASTER, mtype,
MPI_COMM_WORLD, &status);

        MPI_Recv(&b, segment, MPI_LONG_DOUBLE, MASTER, mtype,
MPI_COMM_WORLD, &status);



        //mat multiplication



        for (i = 0; i < segment; i++)

            c[i] = a[i] * b[i];
```

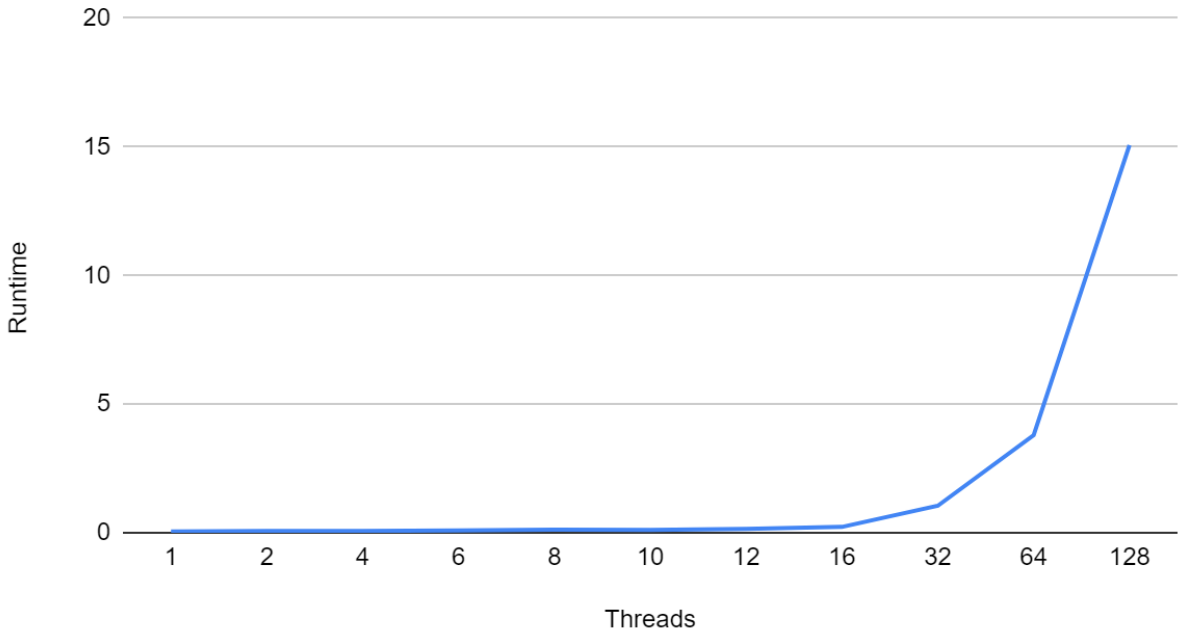
```
        mtype = FROM_WORKER;

        MPI_Send(&offset, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD);

        MPI_Send(&segment, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD);

        MPI_Send(&c, segment, MPI_LONG_DOUBLE, MASTER, mtype,
MPI_COMM_WORLD);

    }

    MPI_Finalize();
    return 0;
}
```

Observations: [Point to Point]

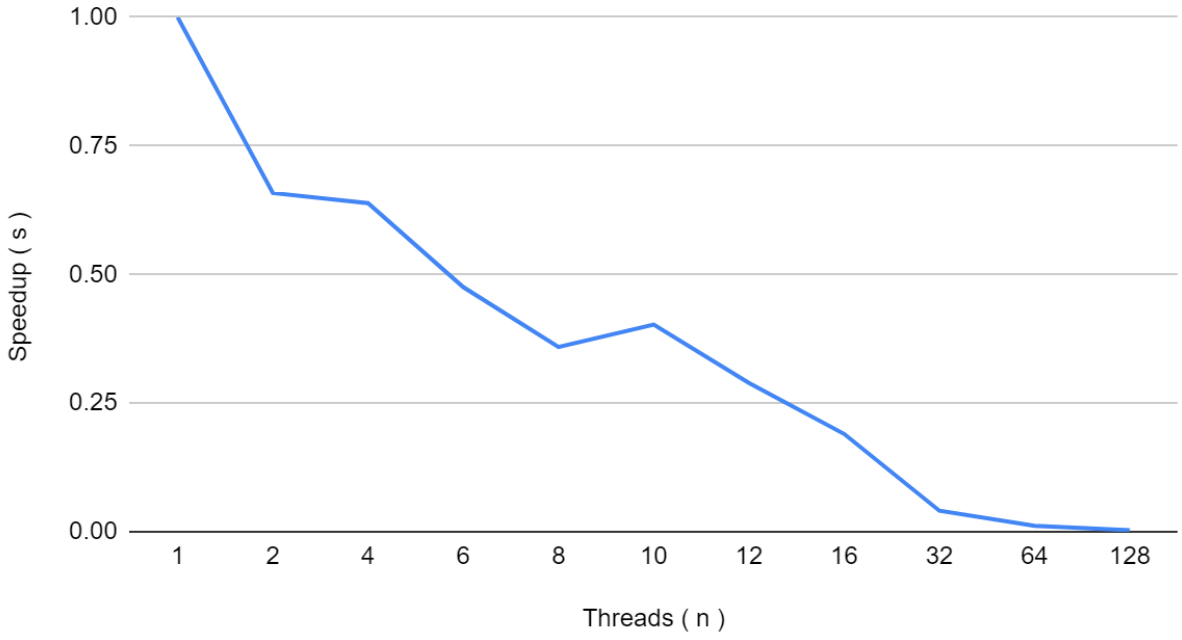
Processes(n)	Runtime	Speedup (s)
1	0.042741	1
2	0.064913	0.6584351363
4	0.066917	0.6387166191
6	0.089911	0.4753700882
8	0.119081	0.3589237578
10	0.106206	0.4024348907
12	0.147787	0.2892067638
16	0.2249	0.1900444642
32	1.046669	0.04083525928
64	3.7924277	0.01127008961
128	15.069948	0.002836174352

Paleti Krishnasai
CED18I039

Runtime vs. Processes



Speedup (s) vs. Processes(n)



Paleti Krishnasai
CED18I039

Parallel Code: [Collective]

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
     int myid, numprocs;
    double startwtime,
    endwtime;
    int  namelen;
    int ARRAY_SIZE = 100000;
    int vector1[ARRAY_SIZE];
    int vector2[ARRAY_SIZE];
    int vector3[ARRAY_SIZE];
     int i, j;
     int s, s0;
    double totalTime;
    double start, end;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc,&argv);
    start=MPI_Wtime();
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
MPI_Get_processor_name(processor_name,&namelen);
    fprintf(stderr,"Process %d on %s\n", myid, processor_name);
    fflush(stderr);
 // Vector 1 Reading
        for (i=0; i < ARRAY_SIZE; i++)
        vector1[i] =i * 156.678;
    // Vector 2 Reading
        for (i=0; i < ARRAY_SIZE; i++)
        vector2[i] =i * 2.0078;

if(myid == 0)
{
    MPI_Bcast(&ARRAY_SIZE, 1, MPI_INT, 0, MPI_COMM_WORLD);
    s = (int) floor(ARRAY_SIZE/numprocs);
    s0 = ARRAY_SIZE%numprocs;
    int vector1Receive[s];
    int vector2Receive[s];
```

```
    int vector3Receive[s];
    if (s0 != 0)
    {
        s = s + 1;
        for(i=0; i < ((s * numprocs) - ARRAY_SIZE); i++)
        {
            vector1[ARRAY_SIZE + i] = 0;
            vector2[ARRAY_SIZE + i] = 0;
        }
    }
    MPI_Bcast(&s, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Scatter(vector1, s, MPI_INT, vector1Receive, s, MPI_INT, 0,
MPI_COMM_WORLD);
    MPI_Scatter(vector2, s, MPI_INT, vector2Receive, s, MPI_INT, 0,
MPI_COMM_WORLD);
    for(i=0; i<s; i++)
    {
        vector3Receive[i] = vector1Receive[i] * vector2Receive[i];
    }
    MPI_Gather(vector3Receive, s, MPI_INT, vector3, s, MPI_INT, 0,
MPI_COMM_WORLD);

    for(i=0; i<ARRAY_SIZE; i++)
        printf("%d\n", vector3[i]);
}
else
{
    MPI_Bcast(&ARRAY_SIZE, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&s, 1, MPI_INT, 0, MPI_COMM_WORLD);
    int vector1Receive[s];
    int vector2Receive[s];
    int vector3Receive[s];
    MPI_Scatter(vector1, s, MPI_INT, vector1Receive, s, MPI_INT, 0,
MPI_COMM_WORLD);
    MPI_Scatter(vector2, s, MPI_INT, vector2Receive, s, MPI_INT, 0,
MPI_COMM_WORLD);
    for(i=0; i<s; i++)
    {
        vector3Receive[i] = vector1Receive[i] * vector2Receive[i];
    }
```

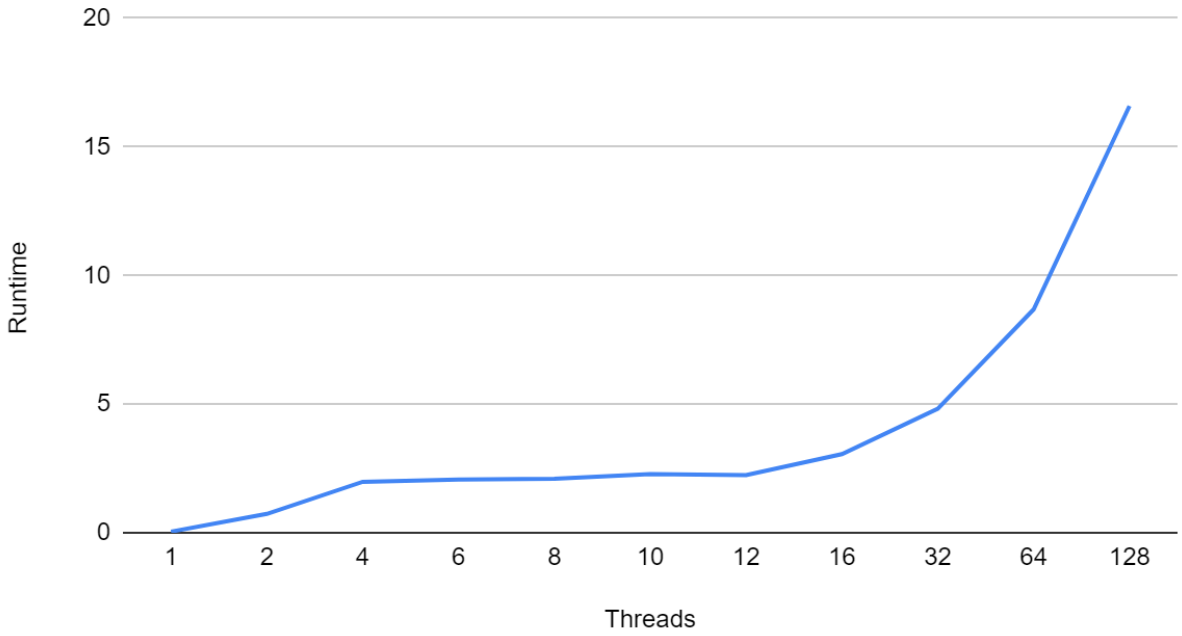
```
    MPI_Gather(vector3Receive, s, MPI_INT, vector3, s, MPI_INT, 0,
MPI_COMM_WORLD);

}
end=MPI_Wtime();
    printf("\nTime= %f",end-start);
MPI_Finalize();
}
```

Observations [Collective] :

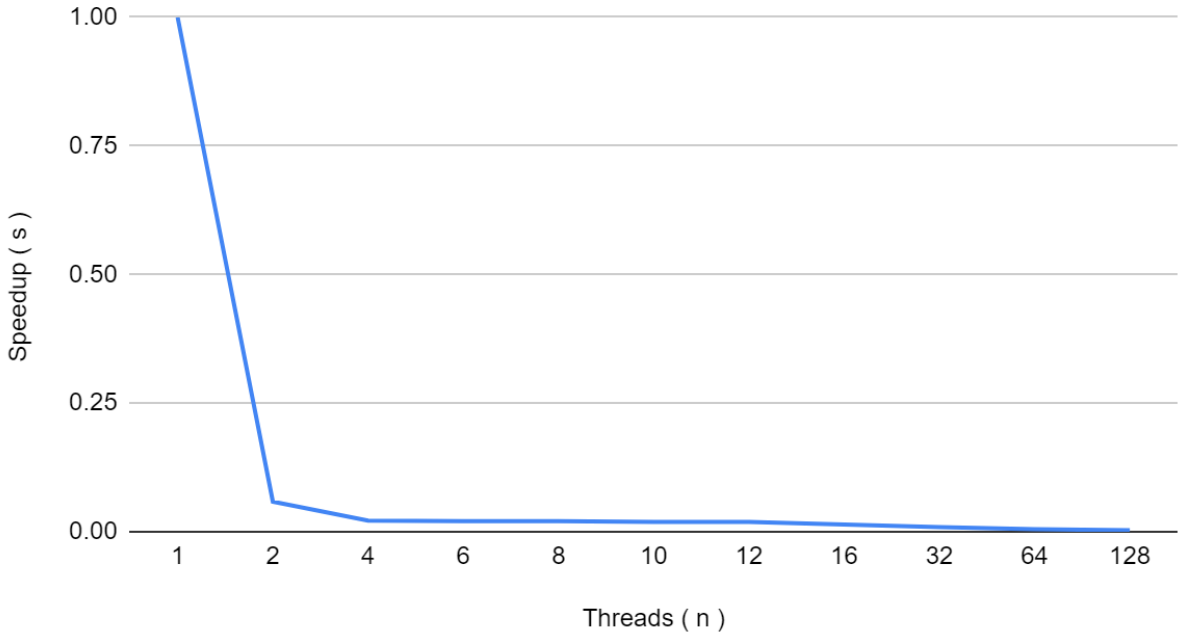
Processes(n)	Runtime	Speedup (s)
1	0.042741	1
2	0.73142	0.05843564573
4	1.977666	0.02161183941
6	2.065529	0.02069251993
8	2.099447	0.02035821814
10	2.279888	0.01874697354
12	2.240546	0.01907615376
16	3.053671	0.01399659623
32	4.816833	0.008873257595
64	8.681692	0.004923118673
128	16.589044	0.002576459499

Paleti Krishnasai
CED18I039

Runtime vs. Processes



Speedup (s) vs. Processes(n)



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)