Paleti Krishnasai
CED18I039

HPC LAB - Matrix Multiplication MPI

Name: Paleti Krishnasai
Roll No: CED18I039
Programming Environment: MPI
Problem: Matrix 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 ROWS_A 500                 /* number of rows in matrix A */

#define COLUMNS_A 300                /* number of columns in matrix A */

#define COLUMNS_B 150                 /* number of columns in matrix B */

#define MASTER 0               /* taskid of first task */

#define FROM_MASTER 1          /* setting a message type */

#define FROM_WORKER 2          /* setting a message type */

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

{

   double start,end;
MPI_Init(&argc,&argv);

    start=MPI_Wtime();

    int i,j,k;

    long double   a[ROWS_A][COLUMNS_A],           /* matrix A to be
multiplied */
   b[COLUMNS_A][COLUMNS_B],           /* matrix B to be multiplied */
   c[ROWS_A][COLUMNS_B];

   //printf("Initializing arrays...\n");

    for (i=0; i<ROWS_A; i++)
```

```
        for (j=0; j<COLUMNS_A; j++)

           a[i][j]= i+j;

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

        for (j=0; j<COLUMNS_B; j++)

           b[i][j]= i*j;

   for (k=0; k<COLUMNS_B; k++)

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

        {

           c[i][k] = 0.0;

           for (j=0; j<COLUMNS_A; j++)

              c[i][k] = c[i][k] + a[i][j] * b[j][k];

        }

   printf("-------------------------------------------------\n");
     printf("Result Matrix:\n");
     for (i=0; i<ROWS_A; i++)
     {
        printf("\n");

        for (j=0; j<COLUMNS_B; j++)
           printf("%6.2Lf   ", c[i][j]);
     }
     printf("\n-------------------------------------------------------\n");
     printf ("Done.\n");
   end=MPI_Wtime();
   printf("\nTime= %f",end-start);
   return 0;
}
```

Paleti Krishnasai
CED18I039

Parallel Code :

```
#include "mpi.h"

#include <stdio.h>

#include <stdlib.h>



#define ROWS_A 500                  /* number of rows in matrix A */

#define COLUMNS_A 300                  /* number of columns in matrix A */

#define COLUMNS_B 150                   /* number of columns in matrix B */

#define MASTER 0               /* taskid of first task */

#define FROM_MASTER 1          /* setting a message type */

#define FROM_WORKER 2          /* setting a message type */



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

{

int   numtasks,              /* number of tasks in partition */

   taskid,                /* a task identifier */

   numworkers,            /* number of worker tasks */

   source,                /* task id of message source */

   dest,                  /* task id of message destination */

   mtype,                 /* message type */

   rows,                  /* rows of matrix A sent to each worker */
```

```
   averow, extra, offset, /* used to determine rows sent to each worker */

   i, j, k, rc;           /* misc */

   long double a[ROWS_A][COLUMNS_A],           /* matrix A to be
multiplied */

   b[COLUMNS_A][COLUMNS_B],           /* matrix B to be multiplied */

   c[ROWS_A][COLUMNS_B];           /* result matrix C */

   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 at least two MPI tasks. Quitting...\n");

      MPI_Abort(MPI_COMM_WORLD, rc);

      exit(1);

   }

   numworkers = numtasks-1;
```

```
//------------------------------------------------ master task
------------------------------------------------------//


   if (taskid == MASTER)

   {

      //printf("mpi_mm has started with %d tasks.\n",numtasks);

      //printf("Initializing arrays...\n");

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

         for (j=0; j<COLUMNS_A; j++)

            a[i][j]= (i+j)*2.658;

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

         for (j=0; j<COLUMNS_B; j++)

            b[i][j]= (i+j)*6.658;



      /* Send matrix data to the worker tasks */

      averow = ROWS_A/numworkers;

      extra = ROWS_A%numworkers;

      offset = 0;

      mtype = FROM_MASTER;
```

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

      {

         rows = (dest <= extra) ? averow+1 : averow;

         //printf("Sending %d rows to task %d
offset=%d\n",rows,dest,offset);

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

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

         MPI_Send(&a[offset][0], rows*COLUMNS_A, MPI_LONG_DOUBLE, dest,
mtype,

                  MPI_COMM_WORLD);

         MPI_Send(&b, COLUMNS_A*COLUMNS_B, MPI_LONG_DOUBLE, dest, mtype,
MPI_COMM_WORLD);

         offset = offset + rows;

      }



      /* Receive results from worker tasks */

      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(&rows, 1, MPI_INT, source, mtype, MPI_COMM_WORLD,
&status);

         MPI_Recv(&c[offset][0], rows*COLUMNS_B, MPI_LONG_DOUBLE, source,
mtype,

                  MPI_COMM_WORLD, &status);

         printf("Received results from task %d\n",source);

      }



      /* Print results */

      printf("-------------------------------------------------\n");

      printf("Result Matrix:\n");

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

      {

         printf("\n");

         for (j=0; j<COLUMNS_B; j++)

            printf("%6.2Lf   ", c[i][j]);

      }

      printf("\n-------------------------------------------------------\n");

      printf ("Done.\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(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD, &status);

      MPI_Recv(&a, rows*COLUMNS_A, MPI_LONG_DOUBLE, MASTER, mtype,
MPI_COMM_WORLD, &status);

      MPI_Recv(&b, COLUMNS_A*COLUMNS_B, MPI_LONG_DOUBLE, MASTER, mtype,
MPI_COMM_WORLD, &status);



      for (k=0; k<COLUMNS_B; k++)

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

         {

            c[i][k] = 0.0;

            for (j=0; j<COLUMNS_A; j++)
```

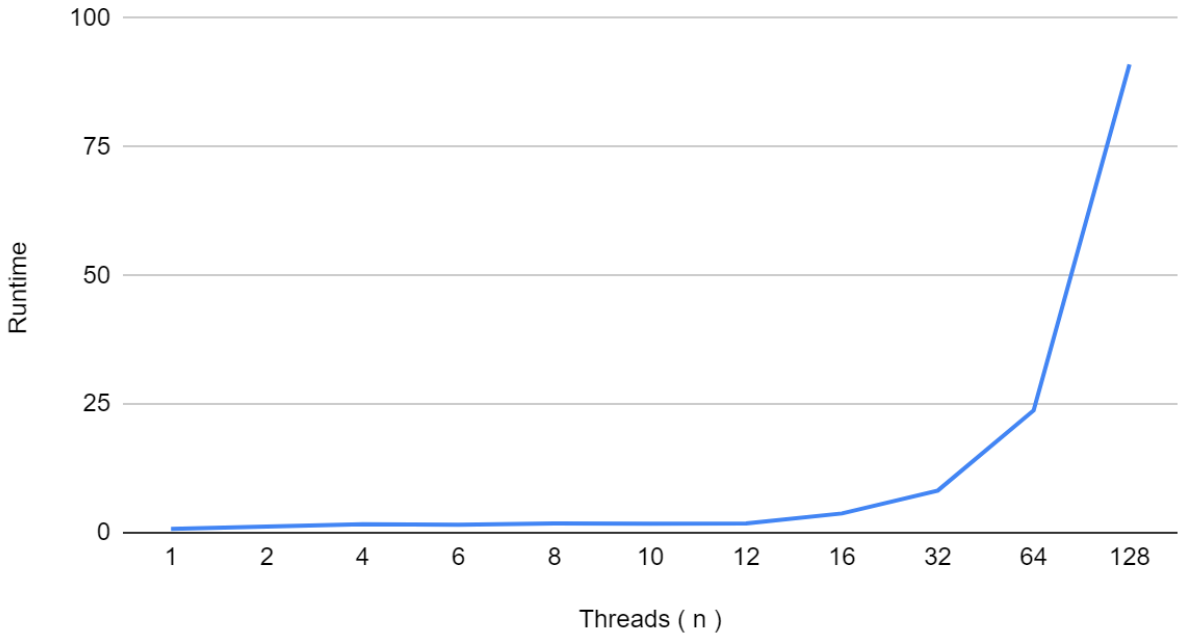
```
                c[i][k] = c[i][k] + a[i][j] * b[j][k];

            }

        mtype = FROM_WORKER;

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

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

        MPI_Send(&c, rows*COLUMNS_B, MPI_LONG_DOUBLE, MASTER, mtype,
MPI_COMM_WORLD);

    }

    MPI_Finalize();

}
```

Observations :

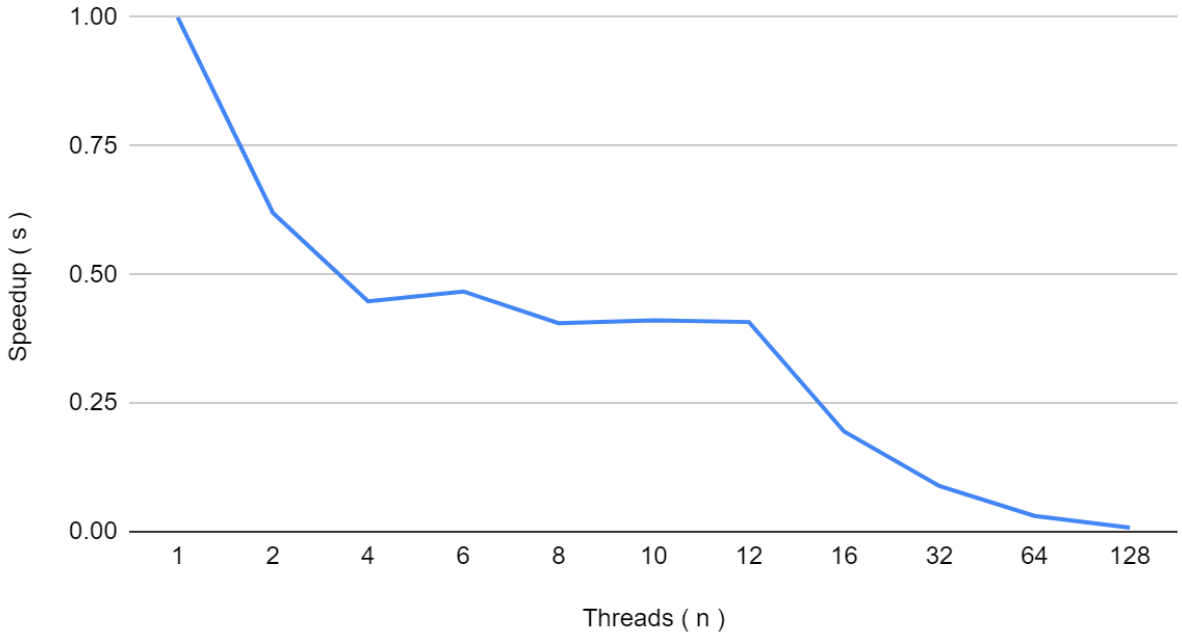
Processes(n)	Runtime	Speedup (s)
1	0.727793	1
2	1.173708	0.6200801221
4	1.625658	0.4476913348
6	1.558417	0.4670078676
8	1.795315	0.4053845704
10	1.772297	0.4106495695
12	1.78564	0.4075810354
16	3.73724	0.1947407713
32	8.186064	0.08890634131
64	23.777508	0.0306084641
128	91.022718	0.007995729154

Paleti Krishnasai
CED18I039

Runtime vs. Processes(n)



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).