High Performance Computing

COM403P

Week-6

Matrix Multiplication

Kadali Paul Babu

CED19I002

CED19I002
Kadali Paul Babu

Objective

Matrix Multiplication for given n x n double precision floating point numbers.

Serial Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

#define SIZE 100

int main()
{
   double a[SIZE][SIZE], b[SIZE][SIZE], c[SIZE][SIZE], rand_a, rand_b;
   double start, end, exec;

   start = omp_get_wtime();

   for (int i = 0; i < SIZE; i++)
   {
       for(int j = 0; j < SIZE; j++)
       {
           rand_a = rand();
           rand_b = rand();

           a[i][j] = i*rand_a;
           b[i][j] = i*rand_b;

           // for(int k = 0; k < 100000; k++)
           c[i][j] = a[i][j] * b[i][j];
       }
   }

   end = omp_get_wtime();

   exec = end - start;

   printf("Serial Exec time - %f\n", exec);

   return 0;
}
```

CED19I002
Kadali Paul Babu

Parallel Code

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

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

#define SIZE 100

int main()
{
   MPI_Init(NULL, NULL);

   long double a[SIZE][SIZE], b[SIZE][SIZE], c[SIZE][SIZE], rand_a, rand_b;
   double start, end, exec;
   int i,j,k;

   int avgrow, extra;
   int offset, mtype;
   int dest, rows;
   int source;
   MPI_Status status;

   start = MPI_Wtime();

   int taskid;
   MPI_Comm_rank(MPI_COMM_WORLD,&taskid);

   int numtasks;
   MPI_Comm_size(MPI_COMM_WORLD,&numtasks);

   int workers = numtasks -1;

   if(taskid == MASTER)
   {
       for (i = 0; i < SIZE; i++)
       {
```

```
            for(j = 0; j < SIZE; j++)
            {
                a[i][j] = (i+j)*1.22;
                b[i][j] = (i+j)*1.22;
            }
        }

//-------------------------------------------------------------------------------------------
--------------
        // Master sending Data to Worker tasks

        avgrow = SIZE/workers;
        extra = SIZE%workers;

        offset = 0;
        mtype = FROM_MASTER;

        for(dest = 1; dest <= workers; dest++)
        {
            rows = (dest <= extra)?avgrow+1:avgrow;

            // 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*SIZE, MPI_LONG_DOUBLE, dest, mtype,
MPI_COMM_WORLD);
            MPI_Send(&b[offset][0], rows*SIZE, MPI_LONG_DOUBLE, dest, mtype,
MPI_COMM_WORLD);

            offset += rows;
        }


//-------------------------------------------------------------------------------------------
--------------
        //Receiving results from Workers

        mtype = FROM_WORKER;
        for(i = 1; i <= workers; 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*SIZE, MPI_LONG_DOUBLE, source, mtype,
MPI_COMM_WORLD, &status);

            // printf("Received results from task=%d\n", source);
        }
        // printf("Result Matrix\n");

        // for(i = 0; i < SIZE;i++)
        // {
        //     // printf("\n");
        //     for(int j = 0; j < SIZE;j++)
        //     {
        //         // printf("%6.2Lf  ",c[i][j]);
        //     }
        // }
        end = MPI_Wtime();

        exec = end - start;
        printf("MPI Exec time - %f\n", exec);
    }


//-------------------------------------------------------------------------------------------
-------------




//-------------------------------------------------------------------------------------------
--------------
    //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*SIZE, MPI_LONG_DOUBLE, MASTER, mtype, MPI_COMM_WORLD,&status);
        MPI_Recv(&b, rows*SIZE, MPI_LONG_DOUBLE, MASTER, mtype, MPI_COMM_WORLD,&status);

        for(k = 0; k < SIZE; k++)
        {
```

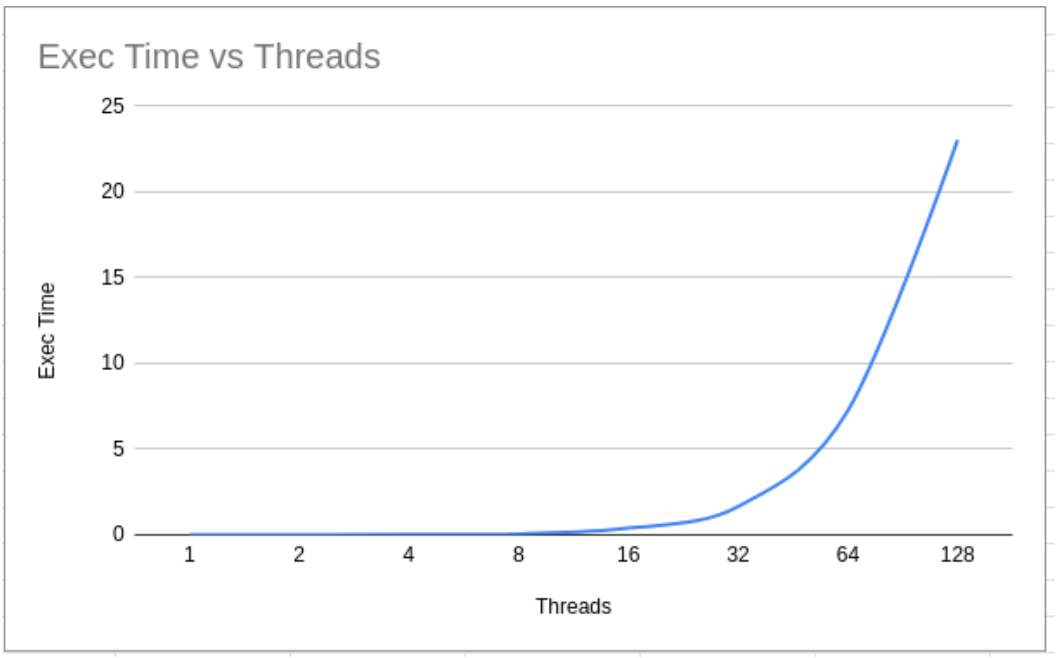
```
            for(i = 0; i < rows; i++)
            {
                c[i][k] = a[i][k] * b[i][k];
            }
        }

        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*SIZE, MPI_LONG_DOUBLE, dest, mtype, MPI_COMM_WORLD);

    }
    MPI_Finalize();

    return 0;
}
```

Observations

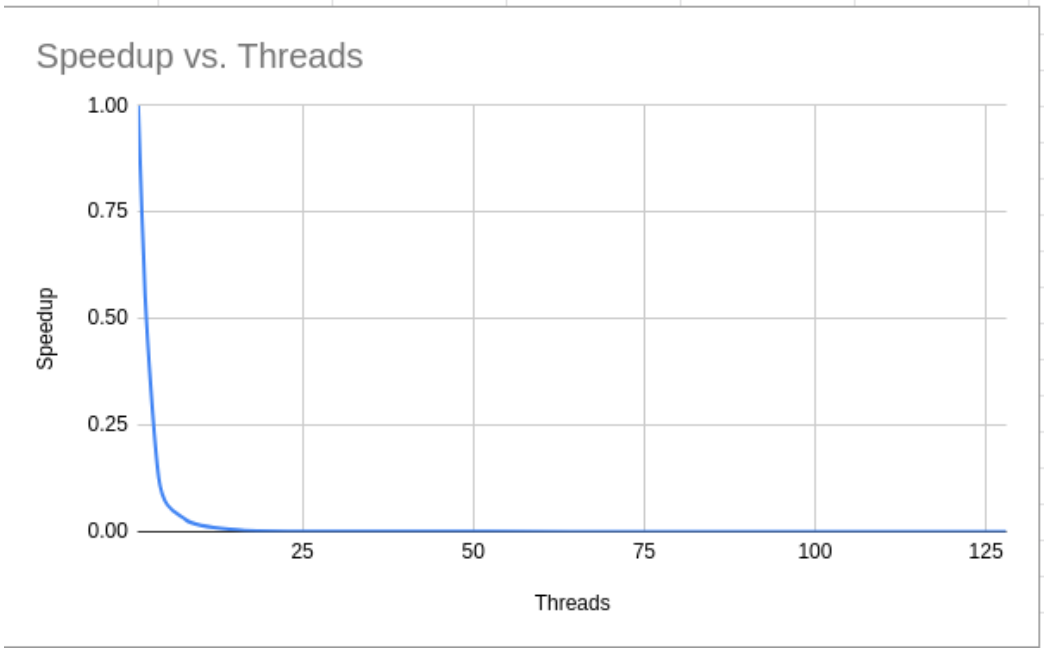
no of processors	Exec Time	Speedup	Parallelization Factor
1	0.000359	1	0
2	0.002854	0.5557112824	-0.666246322
4	0.0126	0.1258730159	-4.629676335
8	0.057482	0.02759124596	-11.74779319
16	0.392194	0.004043917041	82.0949979
32	1.652586	0.000959707997	1734.972678
64	7.239875	0.0002190645557	19776.7459
128	23.03238	0.00006885957943	140372.641

Threads vs Time

CED19I002
Kadali Paul Babu



Speedups vs Threads



CED19I002
Kadali Paul Babu

Inferences

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)