

High Performance Computing

COM403P

Week-6

Matrix Addition

Kadali Paul Babu

CED19I002

Objective

Matrix Addition for given $n \times n$ double precision floating point numbers.

Serial Code

```
#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;
}
```

Parallel Code

```
#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);

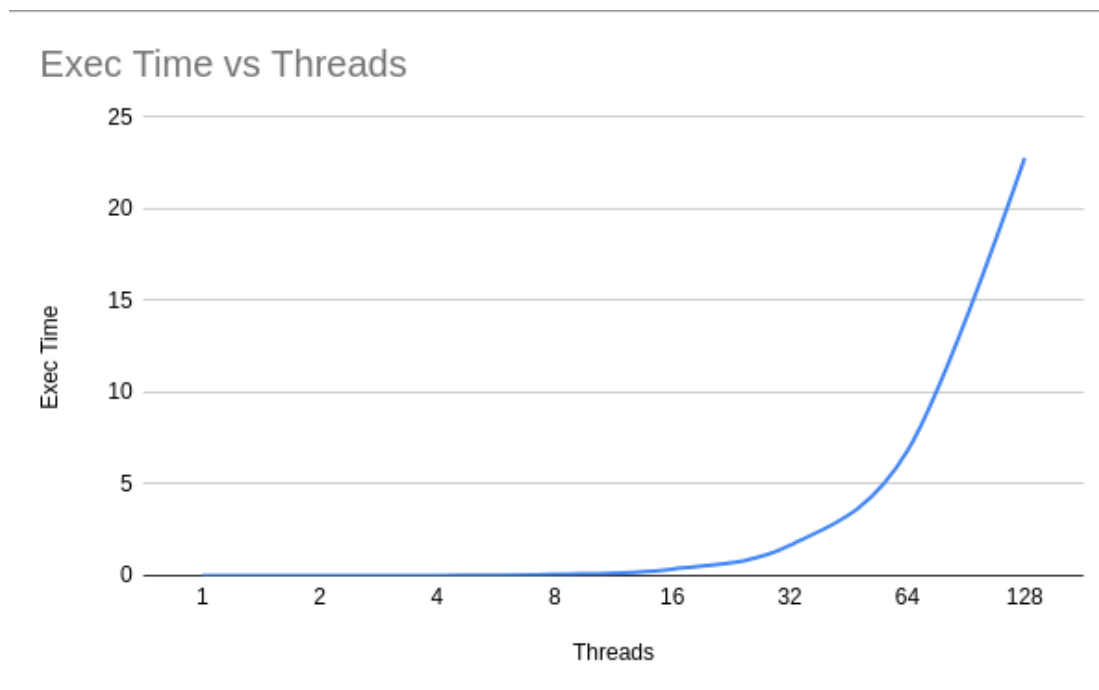
}
MPI_Finalize();

return 0;
}
```

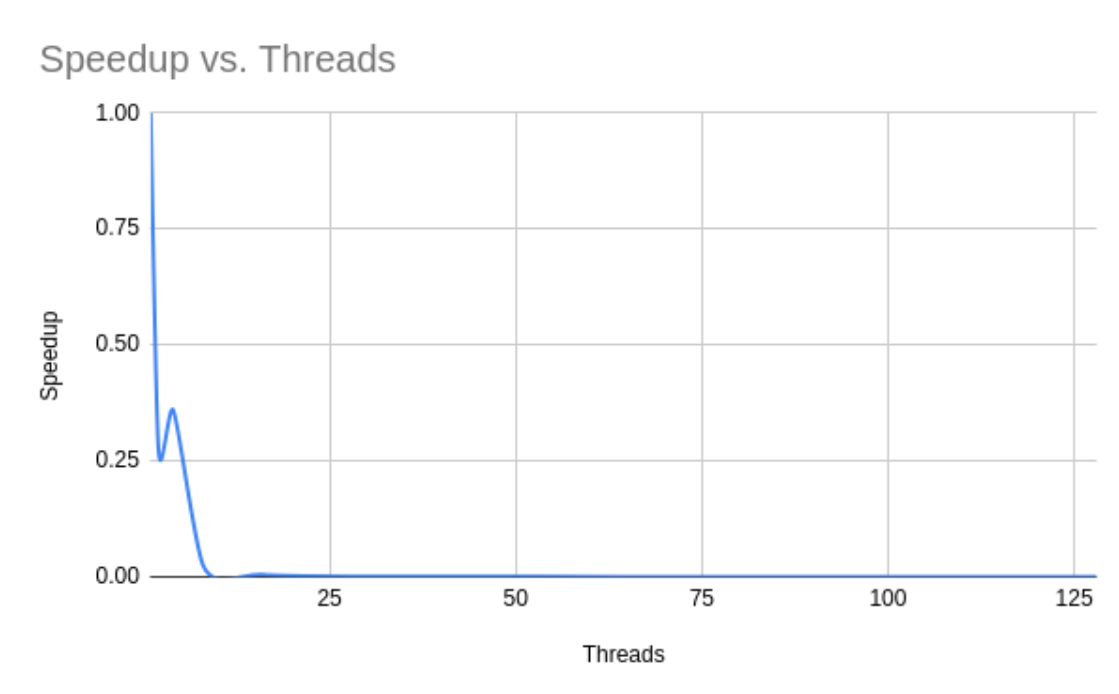
Observations

no of processors	Exec Time	Speedup	Parallelization Factor
1	0.000477	1	0
2	0.005553	0.2856113812	-2.084384195
4	0.004405	0.360045403	-1.18495166
8	0.062945	0.02519660021	-12.89596469
16	0.357613	0.004434961816	74.82702816
32	1.640225	0.0009669405112	1721.982976
64	6.780678	0.0002338999138	18522.10929
128	22.776056	0.00006963453198	138810.3468

Threads vs Time



Speedups vs Threads



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)