Report for HPC LAB

Name: Bhaskar R
Roll No: CED18I009
Programming Environment: MPI
Problem: Matrix Multiplication
Date: 22nd October 2021

Hardware Configuration:

CPU NAME : Intel core i5 – 8250U @ 1.60 GHz
Number of Sockets : 1
Cores per Socket : 4
Threads per core : 2
L1 Cache size : 32KB (Per Core)
L2 Cache size : 256KB (Per Core)
L3 Cache size : 6MB (Shared)
RAM : 8 GB

Serial Code:

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define NR 5
#define NC 5
#define MASTER 0
#define FROM_MASTER 1
#define FROM_WORKER 2int main()
{
	double start, end;
	int i, j;
	long double a[NR][NC], b[NC][NC], c[NR][NC];
	start = MPI_Wtime();
	for (i = 0; i < NR; i++)
		for (j = 0; j < NC; j++)
			a[i][j] = i + j * 1.785;
	for (i = 0; i < NR; i++)
		for (j = 0; j < NC; j++)
			b[i][j] = i + j * 0.987;
	for (i = 0; i < NR; i++)
		for (j = 0; j < NC; j++)
		{
			c[i][j] = a[i][j] * b[i][j];
		}
	printf("\nResultant Matrix:\n");
	for (i = 0; i < NR; i++)
	{
		printf("\n");
		for (j = 0; j < NC; j++)
			printf("%3.1Lf ", c[i][j]);
	}
	printf("\nFinished.\n");
```

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

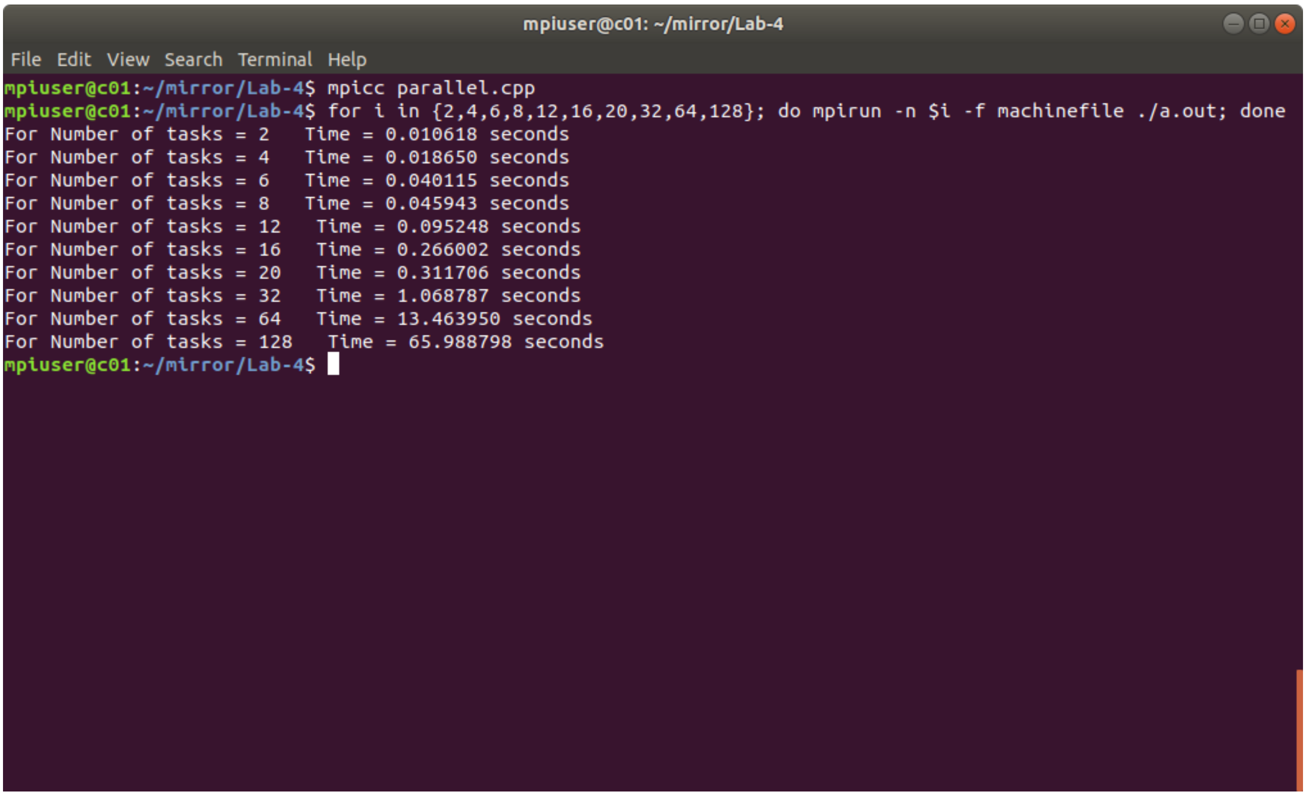
Parallel Code :

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define MASTER 0
#define FROM_MASTER 1
#define FROM_WORKER 2
#define MatA_rows 5
#define MatA_cols 5
#define MatB_cols 5
// cluster /parallel code
int main(int argc, char *argv[])
{
        int numtasks, taskid, numworkers, source, dest, mtype, rows, averow, extra, offset, i,
j,
                k, rc;
        long double a[MatA_rows][MatA_cols], b[MatA_cols][MatB_cols],
                c[MatA_rows][MatB_cols];
        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\n");
                MPI_Abort(MPI_COMM_WORLD, rc);
                exit(1);
        }
        numworkers = numtasks - 1;
        char pro_name[MPI_MAX_PROCESSOR_NAME];
        int name_len;
        MPI_Get_processor_name(pro_name, &name_len);
        printf("Working in Processor %s\n", pro_name);
        if (taskid == MASTER)
        {
                for (i = 0; i < MatA_rows; i++)
                        for (j = 0; j < MatA_cols; j++)
                                a[i][j] = i + j;
                for (i = 0; i < MatA_cols; i++)
                        for (j = 0; j < MatB_cols; j++)
                        {
                                b[i][j] = i * j;
                                c[i][j] = 0.0;
                        }
```

```
		averow = MatA_rows / numworkers;
		extra = MatA_rows % numworkers;
		offset = 0;
		mtype = FROM_MASTER;
		for (dest = 1; dest <= numworkers; dest++)
		{
			rows = (dest <= extra) ? averow + 1 : averow;
			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 * MatA_cols, MPI_LONG_DOUBLE,
dest, mtype,
					MPI_COMM_WORLD);
			MPI_Send(&b, MatA_cols * MatB_cols, MPI_LONG_DOUBLE, dest,
mtype,
					MPI_COMM_WORLD);
			offset = offset + rows;
		}
		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 * MatB_cols, MPI_LONG_DOUBLE,
source, mtype, MPI_COMM_WORLD, &status);
		}
		printf("Result Matrix:\n");
		for (i = 0; i < MatA_rows; i++)
		{
			printf("\n");
			for (j = 0; j < MatB_cols; j++)
				printf("%3.2Lf ", c[i][j]);
		}
		end = MPI_Wtime();
		printf("\nTime= %f", end - start);
	}
	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 * MatA_cols, MPI_LONG_DOUBLE, MASTER, mtype,
				MPI_COMM_WORLD, &status);
		MPI_Recv(&b, MatA_cols * MatB_cols, MPI_LONG_DOUBLE, MASTER,
mtype,
				MPI_COMM_WORLD, &status);
		for (k = 0; k < MatB_cols; k++)
			for (i = 0; i < rows; i++)
			{
```

```
                        c[i][k] = 0.0;
                        for (j = 0; j < MatA_cols; 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 * MatB_cols, MPI_LONG_DOUBLE, MASTER, mtype,
                         MPI_COMM_WORLD);
        }
        MPI_Finalize();
}
```

Output :



Compilation and Execution:

Compiling using mpic++ parallel.cpp

For execution use

for i in {2,4,6,8,12,16,20,32,64,128}; do mpirun -n $i -f machinefile ./a.out

Observations:

Number of Threads	Execution Time	Speed-up	Parallelization Fraction
1	0.008	1	
2	0.010	0.8	-0.5
4	0.018	0.4444	-1.667
6	0.040	0.2	-4.8
8	0.045	0.1778	-5.2849
12	0.095	0.0842	-11.8653
16	0.266	0.0301	-34.3708
20	0.311	0.0257	-39.9058
32	1.068	0.0075	-136.6022
64	13.46	0.0006	-1692.1058
128	65.98	0.0001	-10077.7323

Speed up can be found using the following formula,

$S(n)=T(1)/T(n)$

where, $S(n)$ = Speedup for thread count 'n'

$T(1)$ = Execution Time for Thread count '1' (serial code)

$T(n)$ = Execution Time for Thread count 'n' (serial code)

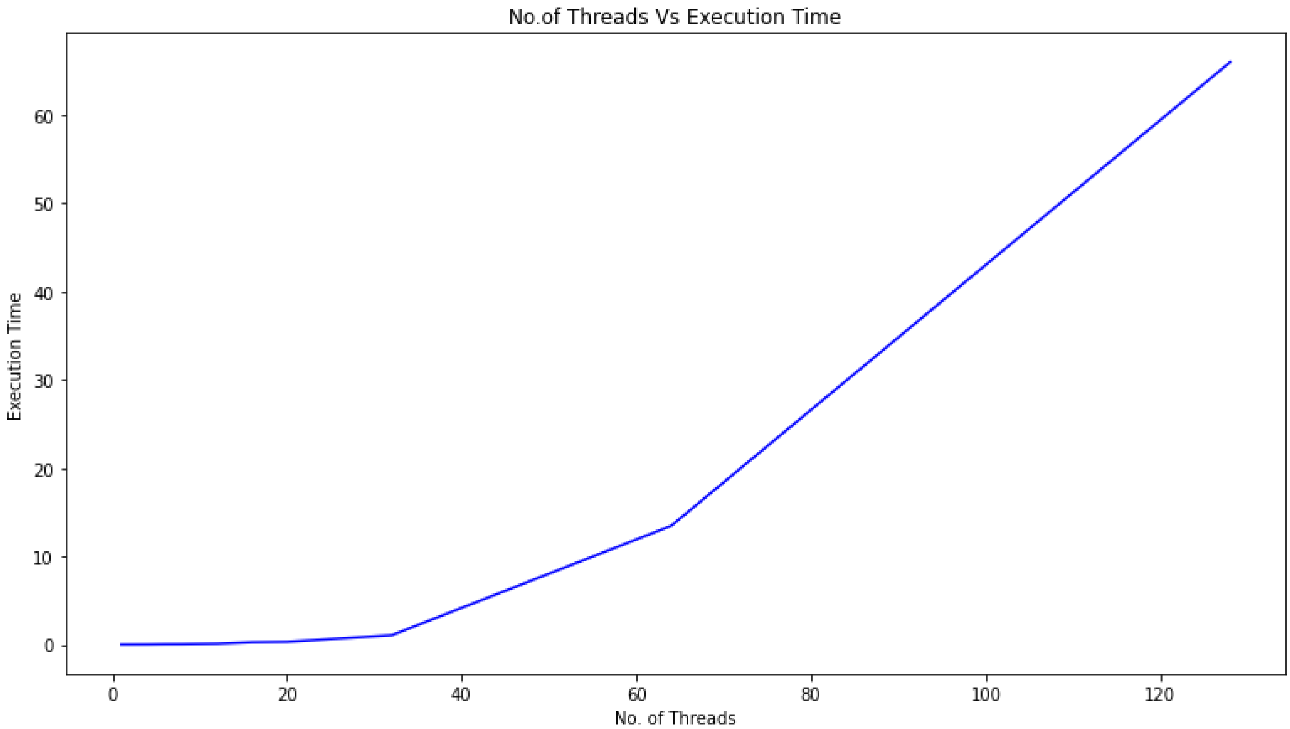
Parallelization Fraction can be found using the following formula, **$S(n)=1/((1 - p) + p/n)$**

where, $S(n)$ = Speedup for thread count 'n'

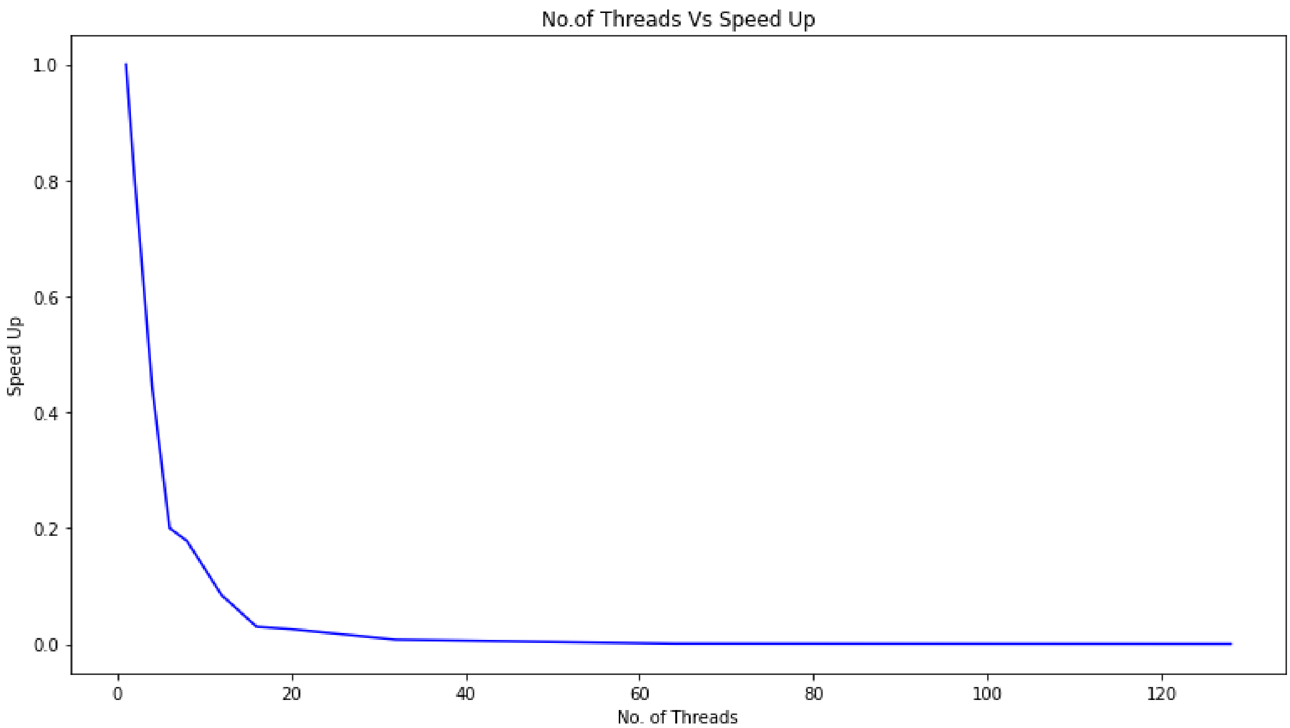
n = Number of threads

p = Parallelization fraction

Number of Threads vs Execution Time:



Number of Threads vs Speed Up:



Inference:

•Execution time is increasing with an increase in the number of threads.

Since the problem is of smaller complexity the overheads of parallelization seem to have more effects here.