Report for HPC LAB

Name: Bhaskar R
Roll No: CED18I009
Programming Environment: MPI
Problem: Vector 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:

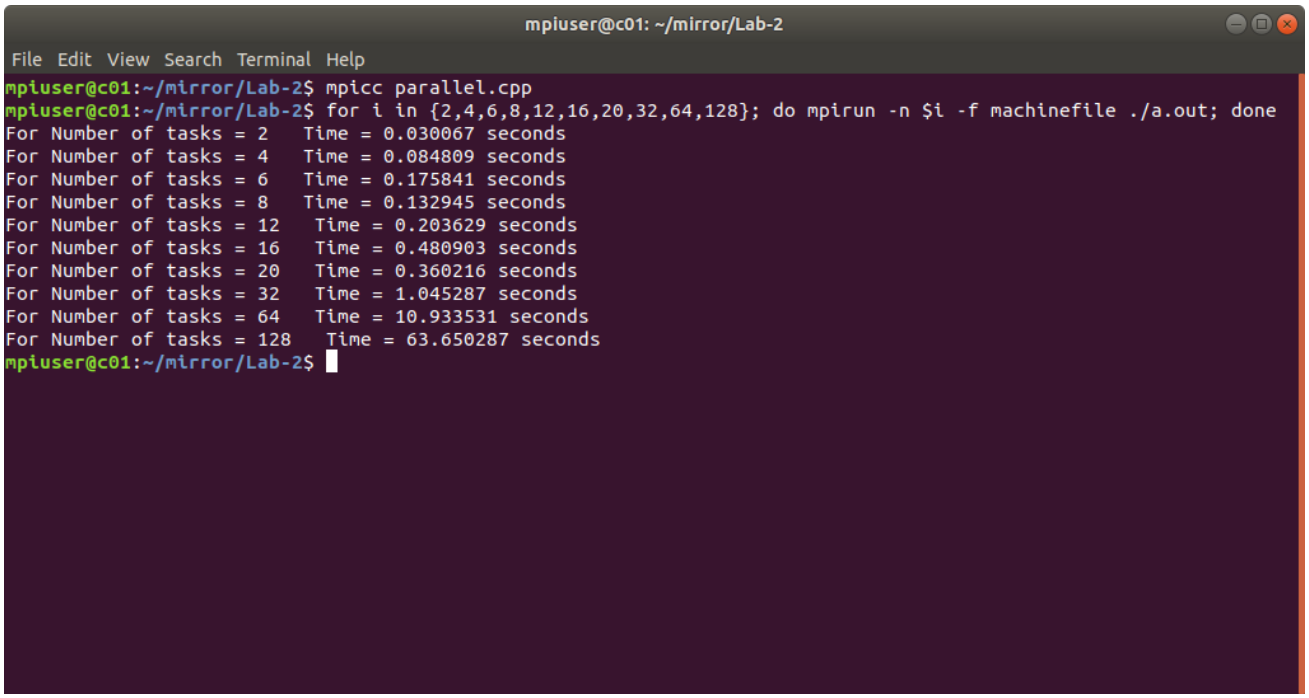
```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define n 100
#define m 100
int main()
{
	double a[n], b[n], c[n];
	float startTime, endTime, execTime;
	int i, k;
	float rtime;
	startTime = MPI_WTIME();
	for (i = 0; i < n; i++)
	{
		a[i] = i * 10.236;	// Use Random function and assign a[i]
		b[i] = i * 152.123; // Use Random function and assign b[i]
		for (int j = 0; j < m; j++)
			c[i] = a[i] * b[i];
		//printf("The value of a[%d] = %lf and b[%d] = %lf and result c[%d] = %lf done
by worker Thread ID = % d\n ", i, a[i], i, b[i], i, c[i], omp_rank);
	}
	endTime = MPI_WTIME();
	execTime = endTime - startTime;
	rtime = execTime;
	printf("\n rtime=%f\n", rtime);
	return (0);
}
```

Parallel Code :

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define array_size 100
#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;
        double starttime, endtime;
        long double a[array_size], b[array_size], c[array_size];
        MPI_Status status;
        MPI_Init(&argc, &argv);
        starttime = 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);
        }
        numworkers = numtasks - 1;
        //master task:
        if (taskid == MASTER)
        {
                for (i = 0; i < array_size; i++)
                        a[i] = i;
                for (j = 0; j < array_size; j++)
                        b[j] = j;
                aveseg = array_size / numworkers;
                extra = array_size % 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 < array_size; i++)
			printf("%6.2Lf ", c[i]);
		endtime = MPI_Wtime();
		printf("That took %f seconds\n", endtime - starttime);
		printf("\nDone.\n");
	}
	//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 addition
		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();
}
```

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	0.01	1	
2	0.03	0.3333	-4.0006
4	0.08	0.125	-9.3333
6	0.17	0.0588	-19.2082
8	0.13	0.0769	-13.7187
12	0.20	0.05	-20.7273
16	0.48	0.0208	-50.2154
20	0.36	0.0278	-36.8118
32	1.04	0.0096	-106.4946
64	10.9	0.0009	-1127.7319
128	63.65	0.0002	-5038.3622

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)

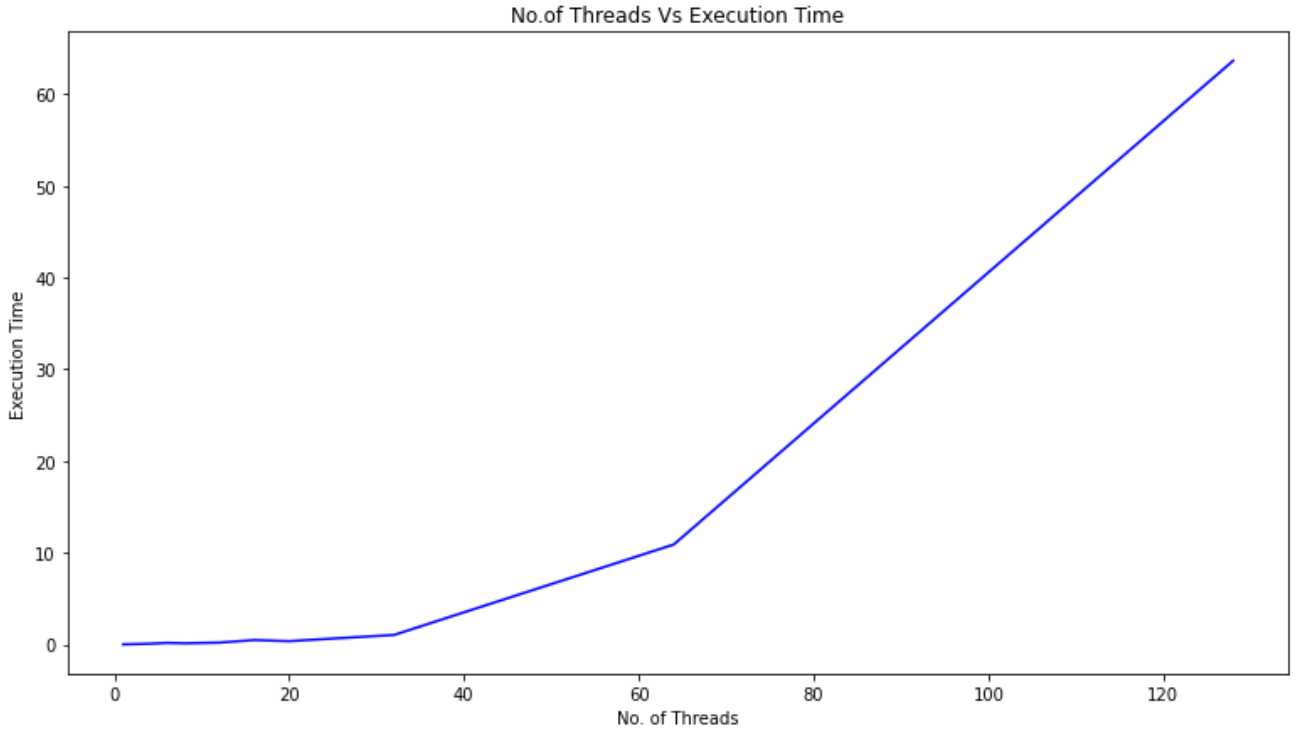
Parallelization Fraction can be found using the following formula, **S(n)=1/((1 - p) + p/n)**

where, S(n) = Speedup for thread count ‘n’

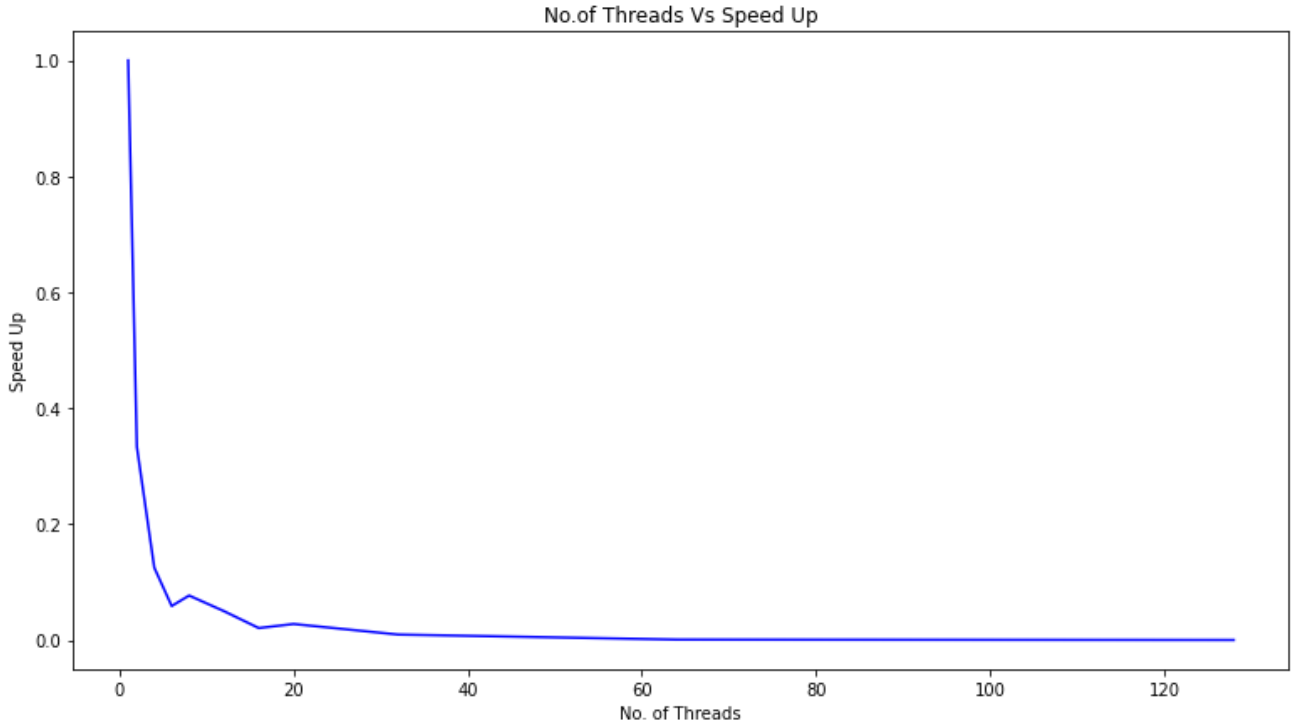
n = Number of threads

p = Parallelization fraction

Number of Threads vs Execution Time:



Number of Threads vs Speed Up:



Inference:

•Execution time increases 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.