

Report for HPC LAB

Name: Bhaskar R

Roll No: CED18I009

Programming Environment: MPI

Problem: Vector Addition

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 <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <omp.h>
#define n 100000
#define m 100000
int main()
{
    double a[n], b[n], c[n];
    float startTime, endTime, execTime;
    int i, k;
    int omp_rank;
    float rtime;
    startTime = omp_get_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\n", i, a[i], i, b[i], i, c[i], omp_rank);
    }
    endTime = omp_get_wtime();
    execTime = endTime - startTime;
    rtime = execTime;
    printf("\n rtime=%f\n", rtime);
    return (0);
}
```

Parallel Code :

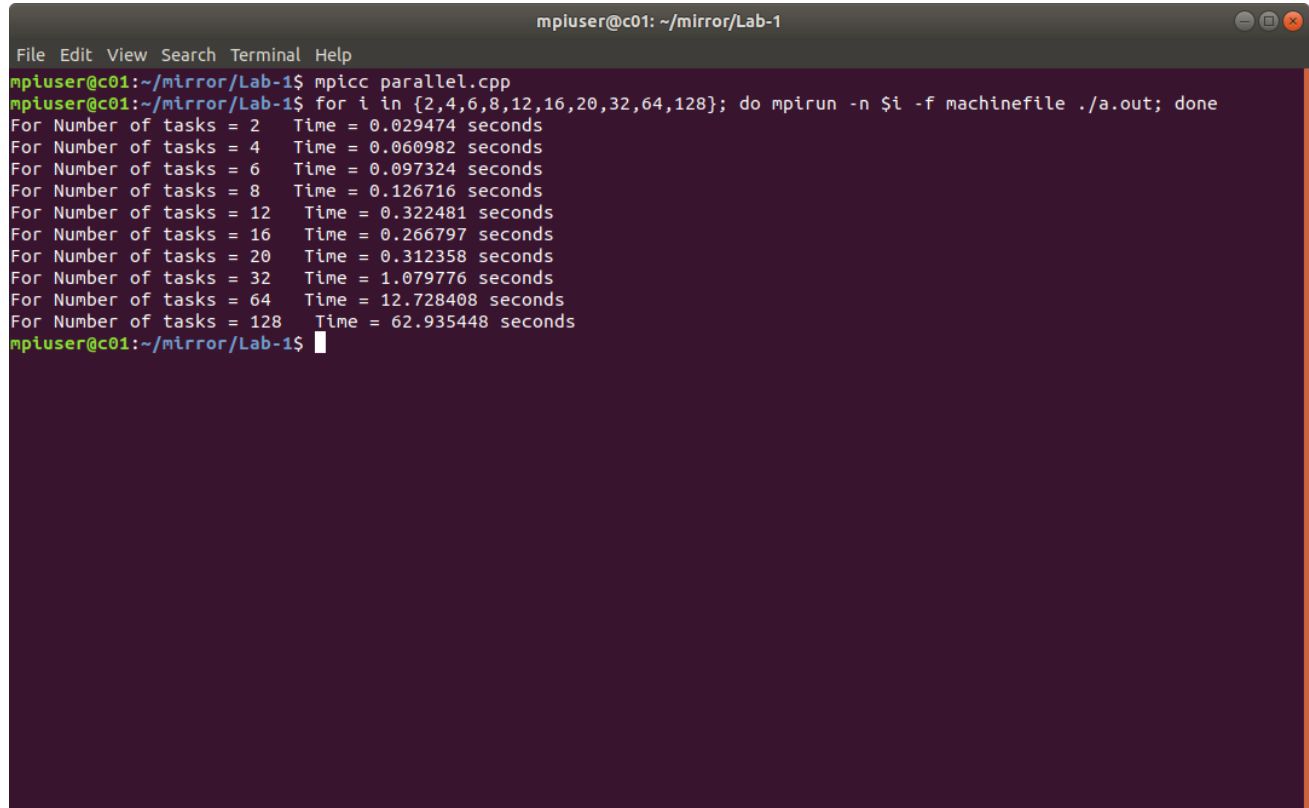
```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define array_size 1000
#define MASTER 0
#define FROM_MASTER 1
#define FROM_WORKER 2
int main(int argc, char *argv[])
{
    int numtasks, taskid, numworkers, source, dest, mtype, segment, aveseq, extra,
        offset, i, j, k, rc;
    double starttime, endtime;
    long double a[array_size], b[array_size], c[array_size];
    MPI_Status status;
    //starttime = MPI_Wtime();
    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. Quitting...\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;
        aveseq = array_size / numworkers;
        extra = array_size % numworkers;
        offset = 0;
        mtype = FROM_MASTER;
        for (dest = 1; dest <= numworkers; dest++)
        {
            segment = (dest <= extra) ? aveseq + 1 : aveseq;
            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 \n", 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 :



```
File Edit View Search Terminal Help
mpiuser@c01: ~/mirror/Lab-1
mpiuser@c01:~/mirror/Lab-1$ mpicc parallel.cpp
mpiuser@c01:~/mirror/Lab-1$ for i in {2,4,6,8,12,16,20,32,64,128}; do mpirun -n $i -f machinefile ./a.out; done
For Number of tasks = 2    Time = 0.029474 seconds
For Number of tasks = 4    Time = 0.060982 seconds
For Number of tasks = 6    Time = 0.097324 seconds
For Number of tasks = 8    Time = 0.126716 seconds
For Number of tasks = 12   Time = 0.322481 seconds
For Number of tasks = 16   Time = 0.266797 seconds
For Number of tasks = 20   Time = 0.312358 seconds
For Number of tasks = 32   Time = 1.079776 seconds
For Number of tasks = 64   Time = 12.728408 seconds
For Number of tasks = 128  Time = 62.935448 seconds
mpiuser@c01:~/mirror/Lab-1$
```

Compilation and Execution:

Compiling using mpicc 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	0.012	1	
2	0.029	0.41	-2.878
4	0.06	0.2	-5.3333
6	0.09	0.13	-8.0308
8	0.12	0.1	-10.2857
12	0.32	0.04	-26.1818
16	0.26	0.05	-20.2667
20	0.31	0.04	-25.2632
32	1.07	0.01	-102.1935
64	12.72	0.0009	-1127.7319
128	62.93	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)

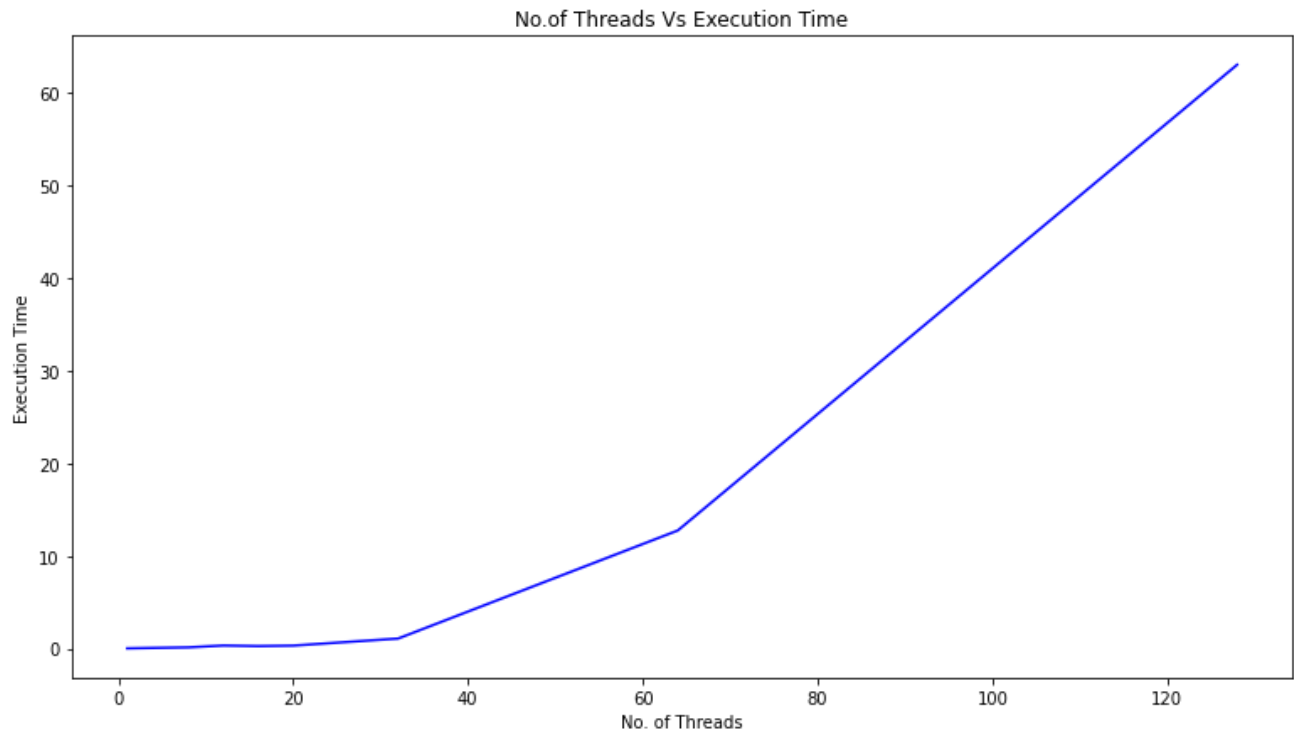
Parallelization Fraction can be found using the following formula, $S(n)=1/((1 - p) + p/n)$

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

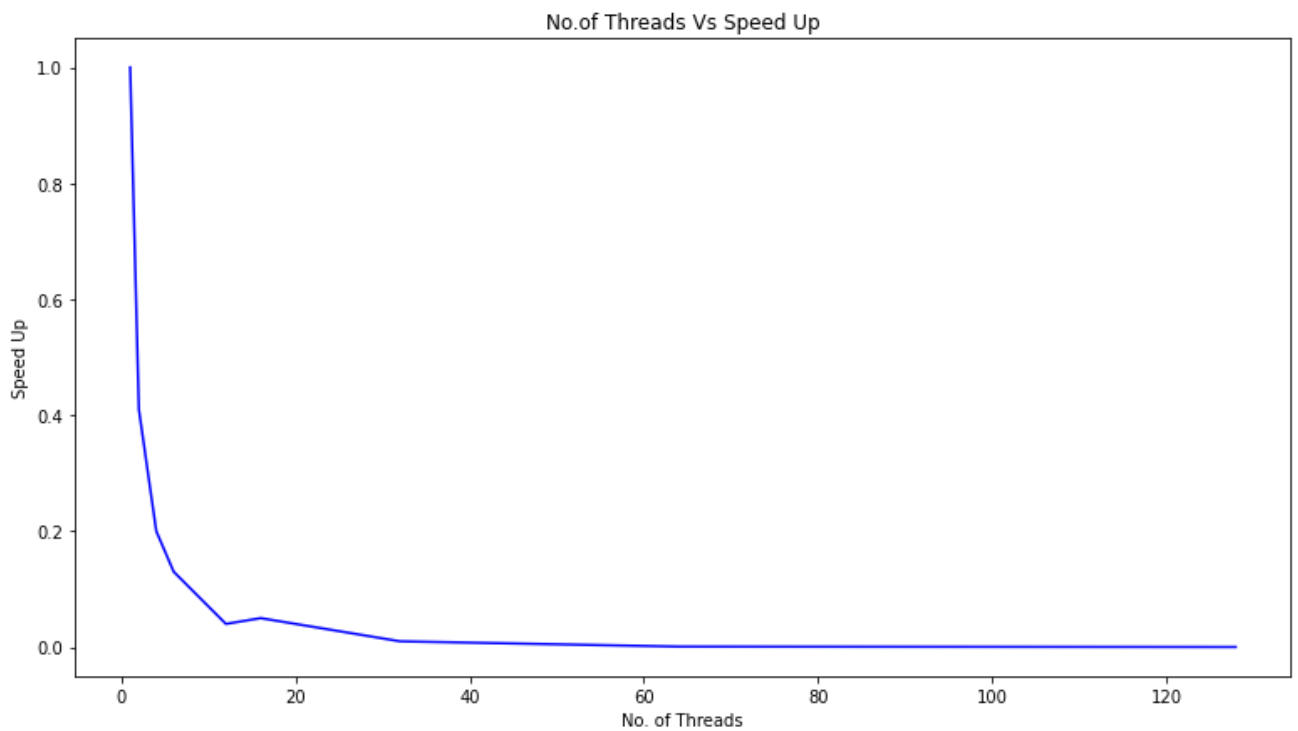
n = Number of threads

p = Parallelization fraction

Number of Threads vs Execution Time:



Number of Threads vs Speed Up:



Inference:

- Execution time increasing with increase in number of threads

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