

Report for HPC LAB

Name: Bhaskar R

Roll No: CED18I009

Programming Environment: MPI

Problem: Vector Dot Product

Date: 30th 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>
#include <math.h>
#define N 100
int main()
{
    double start, end;
    start = MPI_Wtime();
    float arr1[N], arr2[N], result = 0.0;
    for (int i = 0; i < N; i++)
    {
        arr1[i] = i + 1.0;
        arr2[i] = i + 1.0;
    }
    for (int i = 0; i < N; i++)
    {
        result += (arr1[i] * arr2[i]);
    }
    printf("Sum is %f.\n", result);
    end = MPI_Wtime();
    printf("\nTime= %f", end - start);
}
```

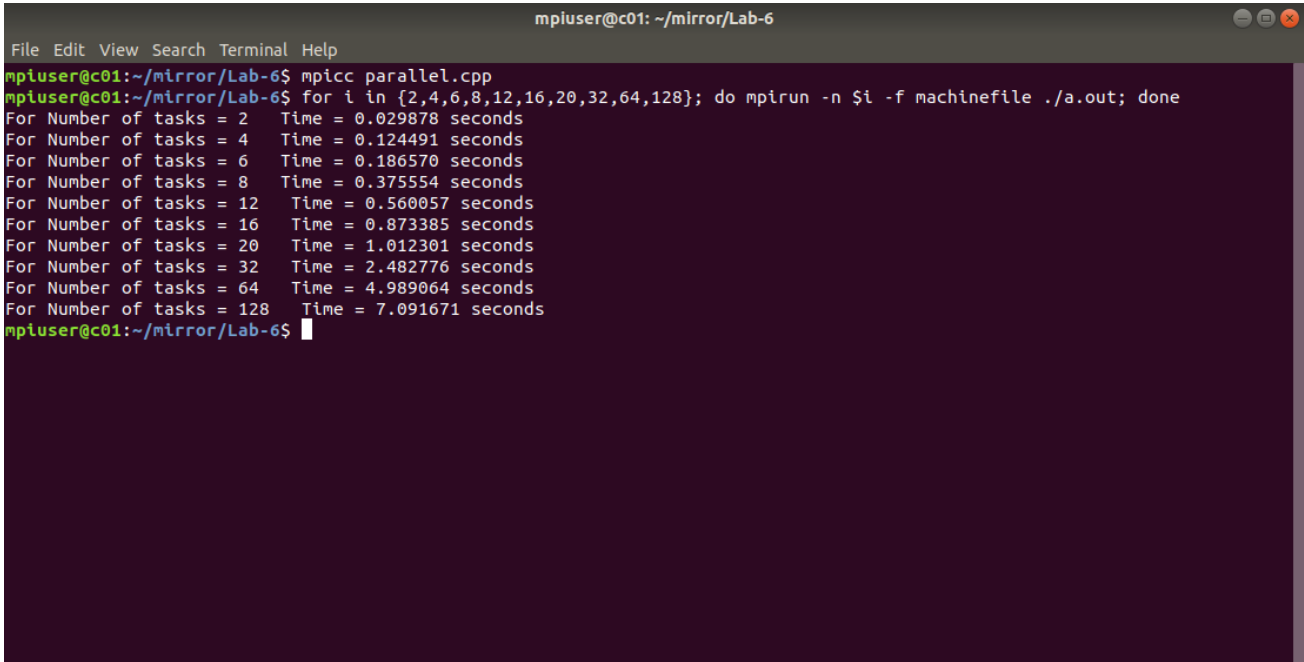
Parallel Code :

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define N 100

int main(int argc, char **argv)
{
    int myid, numprocs;
    int i, x, start, end, rem;
    long double val = 0, result, arr1[N], arr2[N];
    double st, ed;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    char pro_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(pro_name, &name_len);
    if (0 == myid)
    {
        st = MPI_Wtime();
        for (i = 0; i < N; i++)
        {
            arr1[i] = i;
            arr2[i] = i;
        }
    }
    /* broadcast arr */
    MPI_Bcast(arr1, N, MPI_LONG_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(arr2, N, MPI_LONG_DOUBLE, 0, MPI_COMM_WORLD);
    /* add portion of arr */
    x = N / numprocs;
    start = myid * x;
    end = start + x;
    for (i = start; i < end; i++)
    {
        val += (arr1[i] * arr2[i]);
    }
    printf("Calculated %Lf in %d - %s\n", val, myid, pro_name);
    /* compute global sum */
    MPI_Reduce(&val, &result, 1, MPI_LONG_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
    if (0 == myid)
    {
        rem = N % numprocs;
        for (i = N - rem; i < N; i++)
            result += (arr1[i] * arr2[i]);
        printf("Dot product is %Lf.\n", result);
        ed = MPI_Wtime();
    }
}
```

```
        printf("\nTime= %f", ed - st);  
    }  
    MPI_Finalize();  
}
```

Output :



The screenshot shows a terminal window titled 'mpiuser@c01: ~/mirror/Lab-6'. The user has compiled a program 'parallel.cpp' using 'mpicc'. They then executed a loop of commands to run the program with different numbers of tasks (2, 4, 6, 8, 12, 16, 20, 32, 64, 128) using 'mpirun -n \$i -f machinefile ./a.out'. The output shows the execution time for each task count, with the time increasing as the number of tasks increases.

```
mpiuser@c01:~/mirror/Lab-6$ mpicc parallel.cpp  
mpiuser@c01:~/mirror/Lab-6$ for i in {2,4,6,8,12,16,20,32,64,128}; do mpirun -n $i -f machinefile ./a.out; done  
For Number of tasks = 2      Time = 0.029878 seconds  
For Number of tasks = 4      Time = 0.124491 seconds  
For Number of tasks = 6      Time = 0.186570 seconds  
For Number of tasks = 8      Time = 0.375554 seconds  
For Number of tasks = 12     Time = 0.560057 seconds  
For Number of tasks = 16     Time = 0.873385 seconds  
For Number of tasks = 20     Time = 1.012301 seconds  
For Number of tasks = 32     Time = 2.482776 seconds  
For Number of tasks = 64     Time = 4.989064 seconds  
For Number of tasks = 128    Time = 7.091671 seconds  
mpiuser@c01:~/mirror/Lab-6$
```

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	1.0	
2	0.02	0.5	-2.0
4	0.12	0.0833	-14.6731
6	0.18	0.0556	-20.3827
8	0.37	0.027	-41.1852
12	0.56	0.0179	-59.8537
16	0.87	0.0115	-91.687
20	1.01	0.0099	-105.2738
32	2.48	0.004	-257.0323
64	4.98	0.002	-506.9206
128	7.09	0.0014	-718.9021

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)

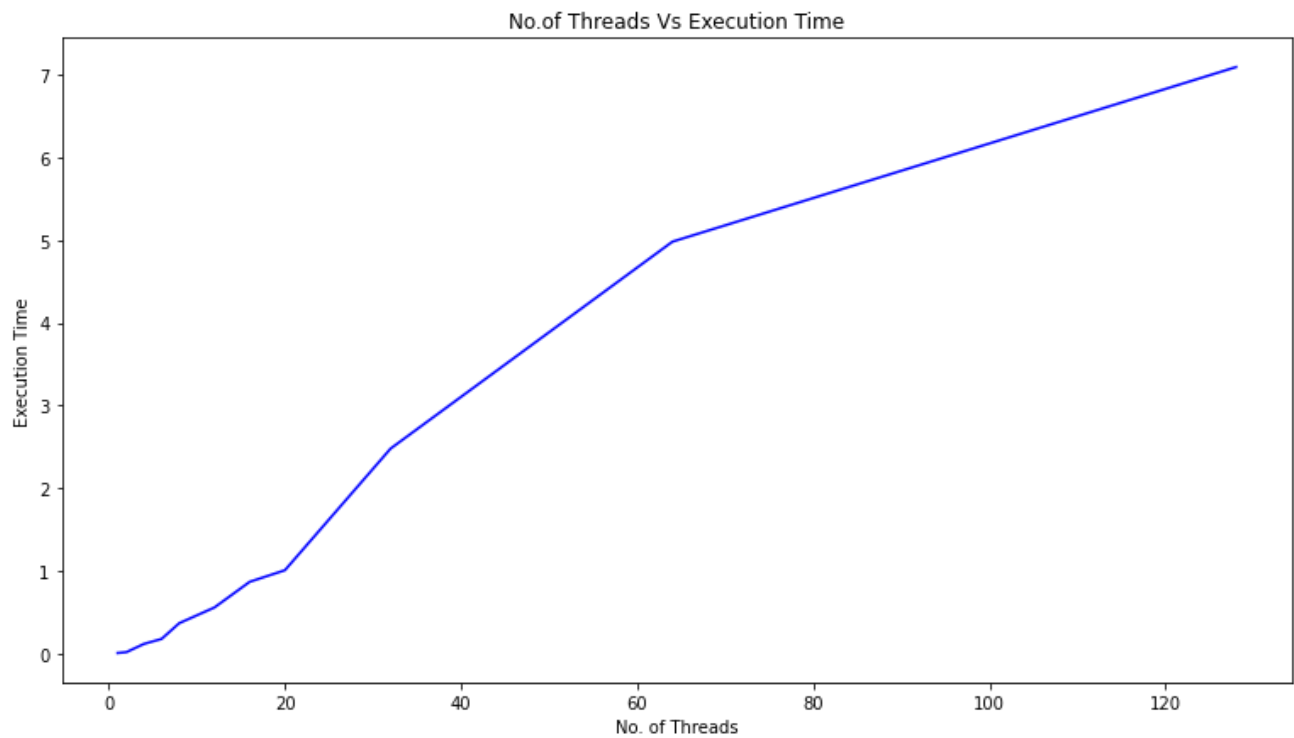
Parallelization Fraction can be found using the following formula, $S(n)=1/((1 - p) + p/n)$

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

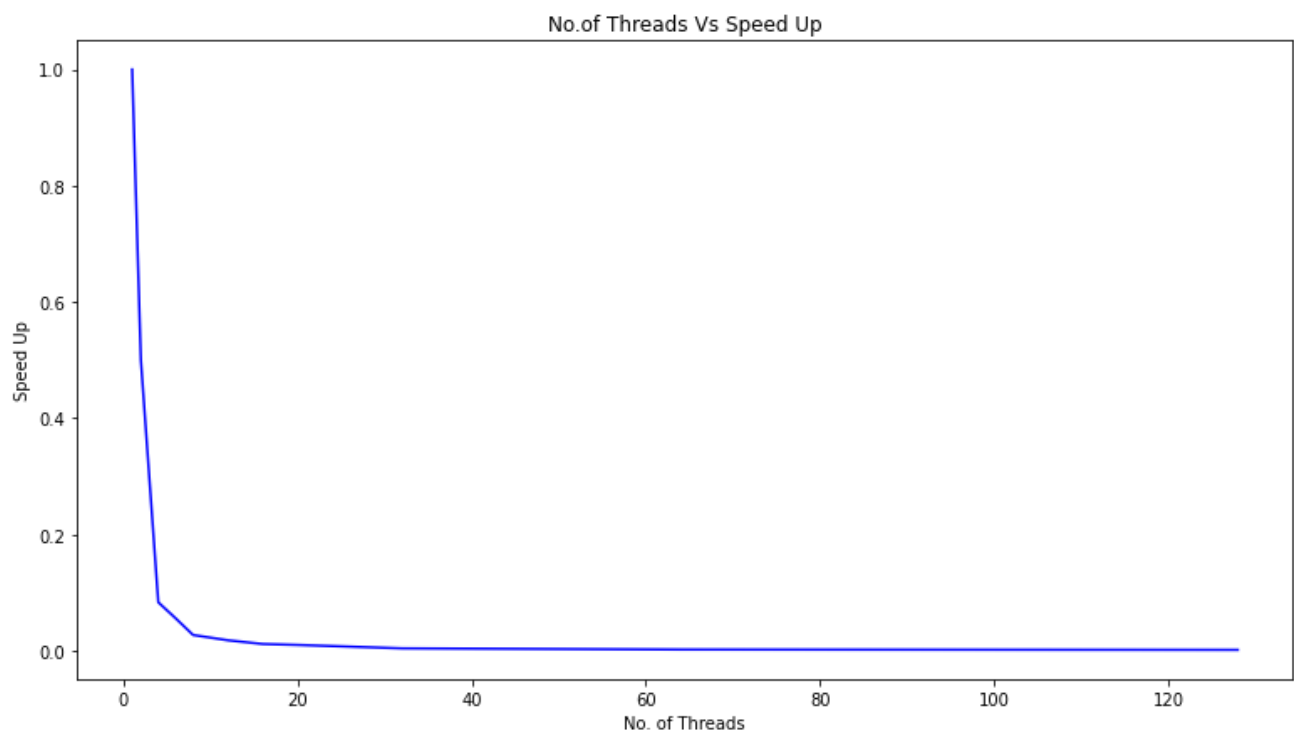
n = Number of threads

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.