# CS301: HIGH PERFORMANCE COMPUTING
# ASSIGNMENT FIVE

**Submitted By:**
**Shaleen Kumar Gupta (201301429)**
**Visharad Bansal (201301438)**

## 1. PROBLEM STATEMENT

Calculate the value of pi using the trapezoid rule.

## 2. METHOD

$f(x)= 4/(1+x^2)$

The value of pi is calculated by integrating f(x) in the interval [0,1].

Integration is done using the Trapezoid Rule, where the area is divided into n small trapezoids and then summed.

## 3. COMPLEXITY

The serial code runs in O(n) time, where n is the number of integration steps.

MPI is used to divide the work among different number of processors.

## 4. OPTIMIZATION STRATEGY

The total number of steps are divided among the two nodes using MPI. Then each node divides up its share of work among its processors. It is a distributed memory system, thus each node stores its sum and a master node is used to sync all the sums in the end.

We executed the program on 4 cores, 8 cores and 16 cores respectively. We have compared our observations with the OpenMP version of the program running on 4 threads also.

Additionally, this particular code was also run on 2 independent nodes, using one as the master and one as the slave.

## 5. CHALLENGES FACED

Using parallelization introduces many race conditions. In lieu of the same, separate variables need to be used of local sums in each thread, and later a global variable is used to calculate the final sum. Hence, the data dependencies were removed.

Also, time is taken for data access, causing delays. Thus, as can be seen from the curve below, the parallel curve takes more time initially compared to the serial time, but as input size increases, the parallel code gives better efficiency.

Also, since we integrate only for a finite number of values, there is always an error with respect to the globally accepted value of pi. To account for the same, we have noted the error at each execution of the program against the globally accepted value of pi correct to 25 decimals, which is, 3.14159265358979323846264 3.

We observed that the error went down with growing number of integration steps, as could be inferred from Table 8.1.

## 6. HARDWARE DETAILS

### 6.1 HPC Cluster (10.100.71.158)

| | | |
|---|---|---|
| No. of Processors : | 23 | |
| vendor_id | : | GenuineIntel |
| cpu family | : | 6 |
| model | : | 62 |
| model name | : | Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz |
| CPU Frequency | : | 1200.000 MHz |
| Cache Size | : | 20480 KB |

### 6.1 Node 1:

| | | |
|---|---|---|
| No. of Processors : | 4 | |
| vendor_id | : | GenuineIntel |
| cpu family | : | 6 |
| model | : | 58 |
| model name | : | Intel(R) Core(TM) i5-3337U CPU @ 1.80GHz |
| CPU Frequency | : | 1800.968 MHz |
| Cache Size | : | 3072 KB |

### 6.2 Node 2:

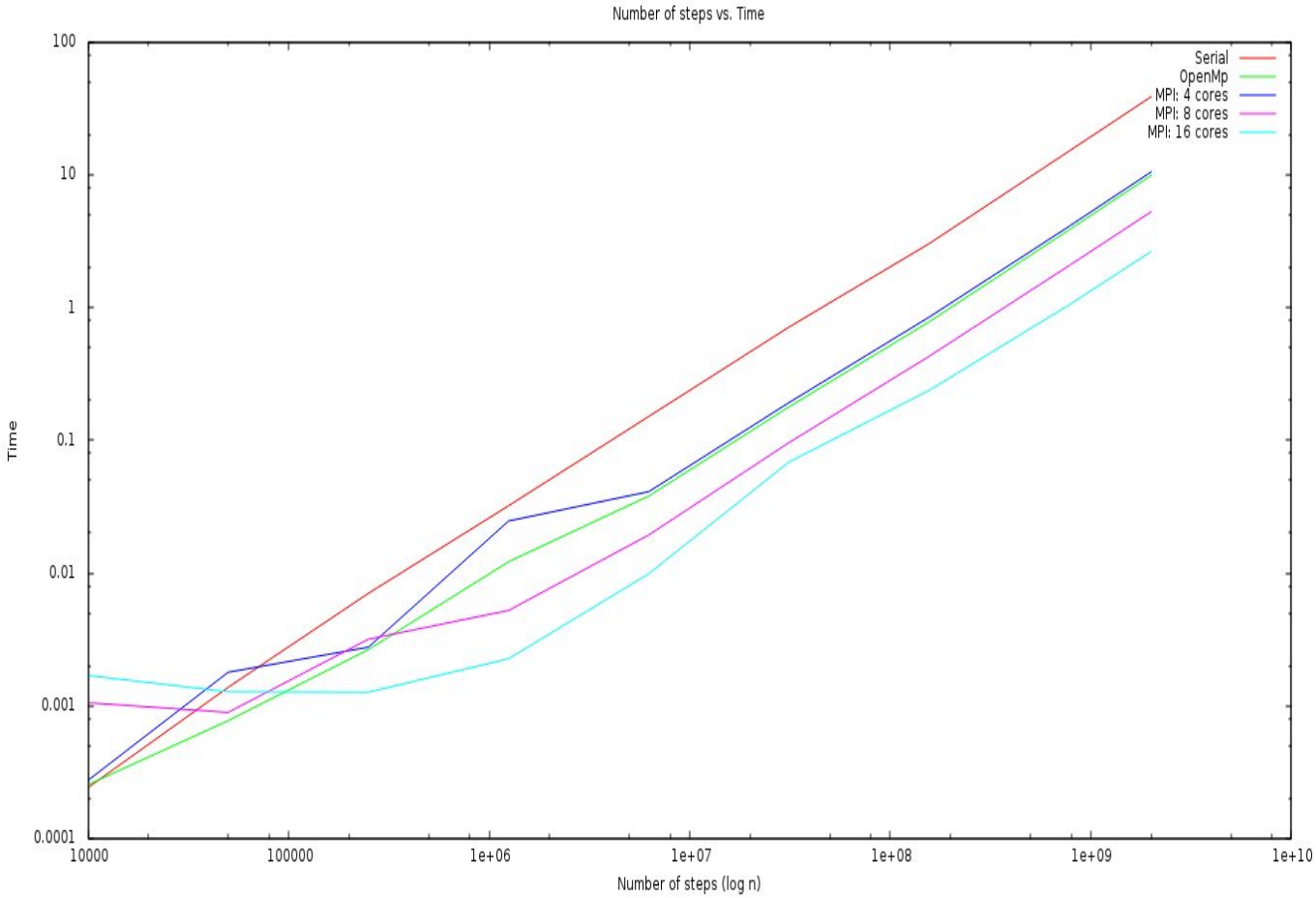
| | | |
|---|---|---|
| No. of Processors : | 4 | |
| vendor_id | : | GenuineIntel |
| CPU family | : | 6 |
| model | : | 60 |
| model name | : | Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz |
| CPU Frequency | : | 2500.000 MHz |
| Cache Size | : | 3072 KB |

## 7. INPUT PARAMETERS

Number of steps (n).

Range of n has been taken from 10000 to 2000000000 (2e9).

## 8. OBSERVATIONS



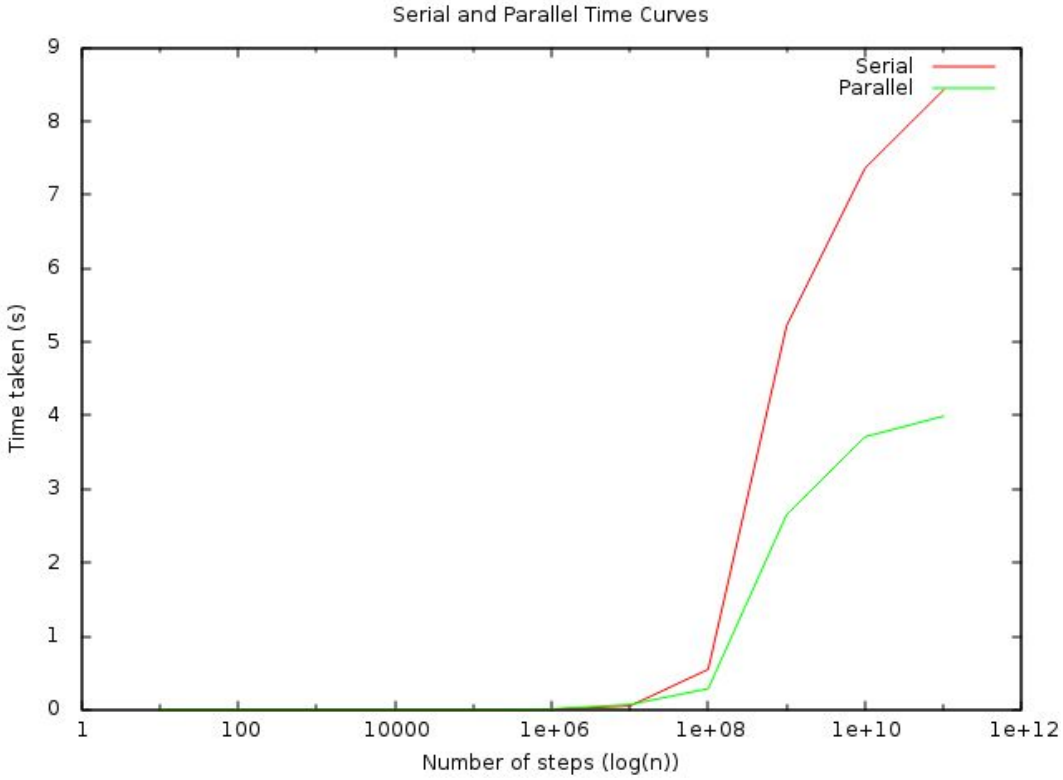

**Graph 8.1: Execution Time vs Problem Size for Serial Code, OpenMP Implementation, and MPI Implementation on 4 cores, 8 cores and 16 cores respectively**

We can clearly observe from the Graph 8.1 that the serial code took a lot more time to calculate the value of pi for number of integration steps in the order greater than 1e+6. However, the serial code was able to compute the same in a lesser time for very low values of number of integration steps.

The total time taken for execution continually decreases as the number of cores increase.

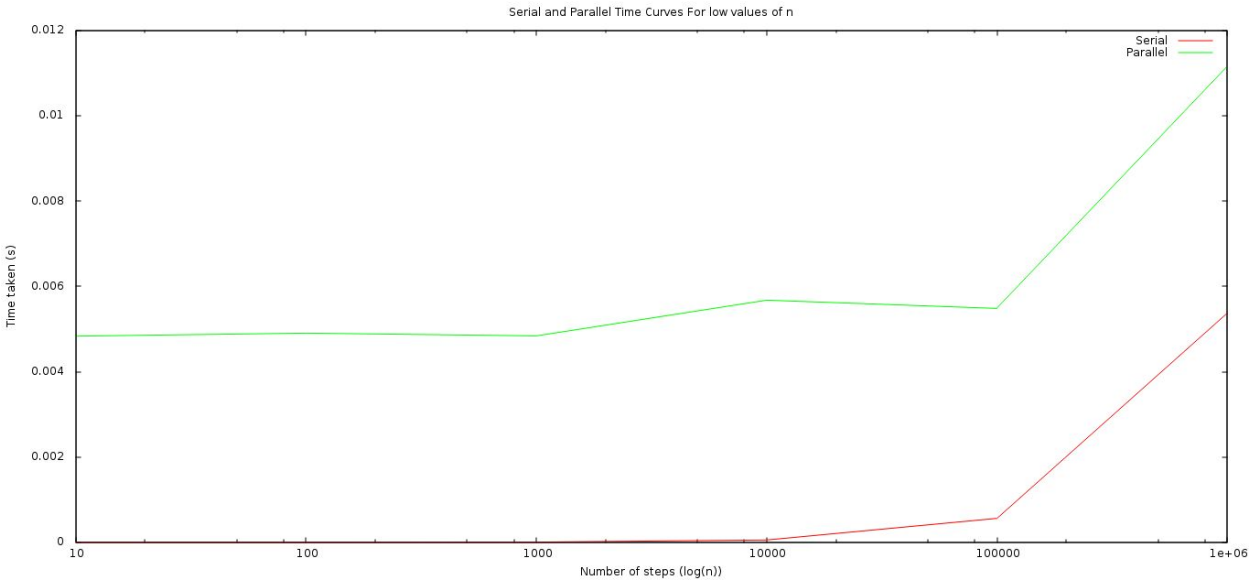
However, we observe that the OpenMP version and MPI version on 4 cores took a comparable amount of time, which is quite expected.

We also executed the program on a distributed memory cluster of 2 nodes having the specification as mentioned previously. Here are the results we got.




**Graph 8.2: Serial and Parallel Scaling with 2 Nodes**

We can clearly observe from the Graph 8.2 that the serial code took a lot more time to calculate the value of pi for number of integration steps in the order of 1e8 to 1e10. However, the serial code was able to compute the same in a lesser time for very low values of number of integration steps. If we zoom into the region where the problem size is less than 1e6 [see Graph 8.3], we can clearly infer that the serial program works faster when compared to the parallel programming running on given machines.




**Graph 8.3: Serial v/s Parallel Scaling for lower values of Problem Size (n) with 2 Nodes**

As mentioned previously, we have also noted the error of the calculated value of pi against the globally accepted value of pi correct to 25 decimal places, which is, 3.1415926535897932384626433.

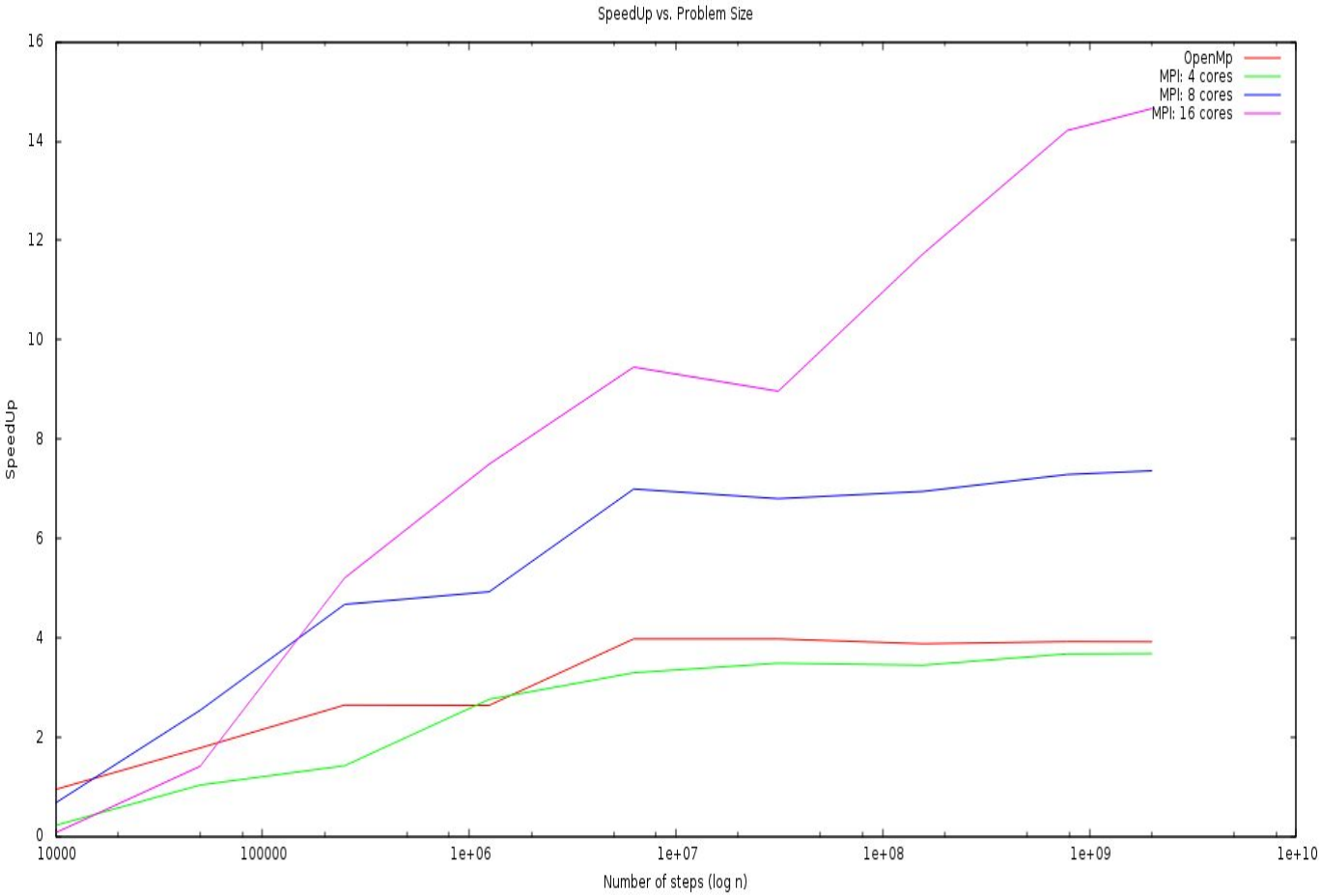
**Table 8.1: Putting Error in Perspective.**
**Problem Size vs Absolute Error**

| Problem Size | Absolute Error |
|---|---|
| 10 | 0.0008333314113051 |
| 100 | 0.0000083333333309 |
| 1000 | 0.0000000833333322 |
| 10000 | 0.0000000008333387 |
| 100000 | 0.0000000000083085 |
| 1000000 | 0.0000000000001457 |
| 10000000 | 0.00000000000001918 |
| 100000000 | 0.000000000000002292 |

| | |
|---|---|
| 1000000000 | 0.00000000000000002141 |
| 10000000000 | 0.00000000000000002194 |

We observe that to get error less than or in the order of 1e-12, one should use the parallel code against the serial code for faster computation with the given machines.

## 9. SPEED UP

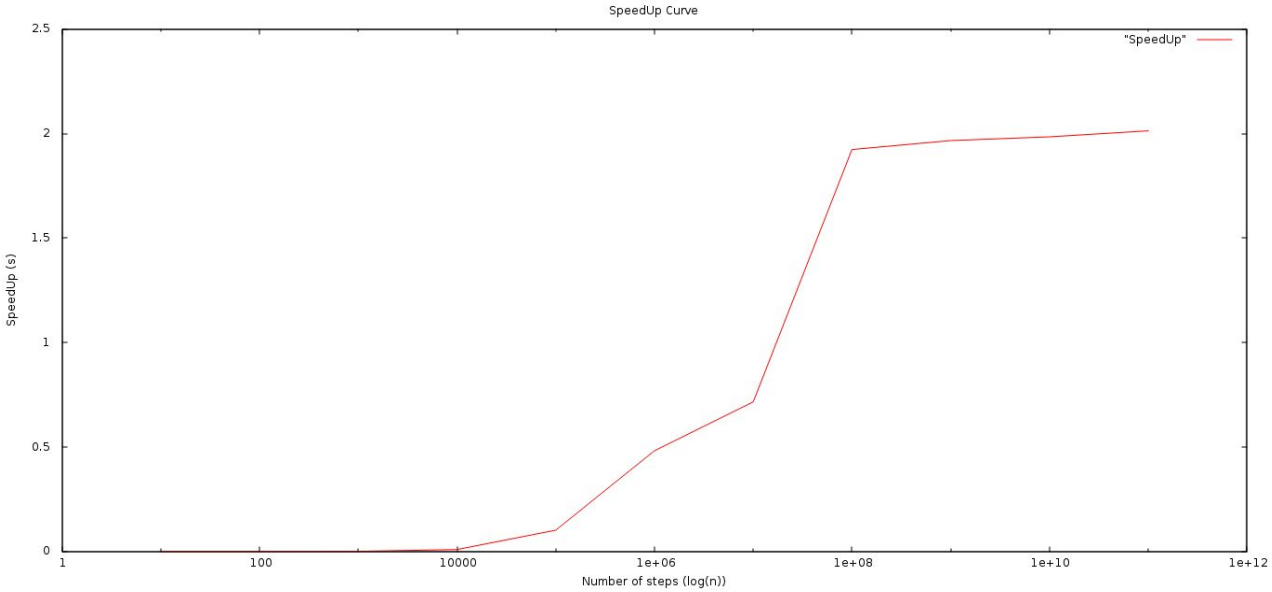



**Graph 9.1: SpeedUp Achieved vs Problem Size for Serial Code, OpenMP Implementation, and MPI Implementation on 4 cores, 8 cores and 16 cores respectively on HPC Cluster**

We observe from Graph 9.1 that the speedup achieved is much higher with the MPI implementation dividing the work on 16 cores compared to 4 cores.

This verifies the fact that the speedup increases with increasing number of cores. However, the efficiency, which is defined as the Speedup achieved per processor, decreases.

Also, the Speedup curve obtained when comparing the parallel MPI code running on 2 nodes against a serial code, is as shown in Graph 9.2. The speedup was less than one for problem size of the order of 1e7, but it was greater than one for problem sizes greater than that.




**Graph 8.3: Speed Up v/s Time Taken**