High Performance Computing

Assignment 5

Jahnavi Suthar(201301414) Deeksha Koul(201301435)

Description of the problem

Calculate the value of PI by integrating f(x) = $\frac{4.0}{1+x^2}$ over the interval [0, 1] using trapezoidal rule.

$$\int_0^1 \frac{4.0}{1+x^2} = \pi$$

Complexity of the Problem (Serial):

We integrate the function f(x) by dividing the interval [0, 1] into 'n' equal trapezoids and summing the area of each trapezoid.

h=(b-a)/n;

where a = 0 , b = 1 , h is step size.

The complexity of the problem is O (n), where n is number of trapezoids in which f(x) has been divided.

Possible Speedup (Theoretical)

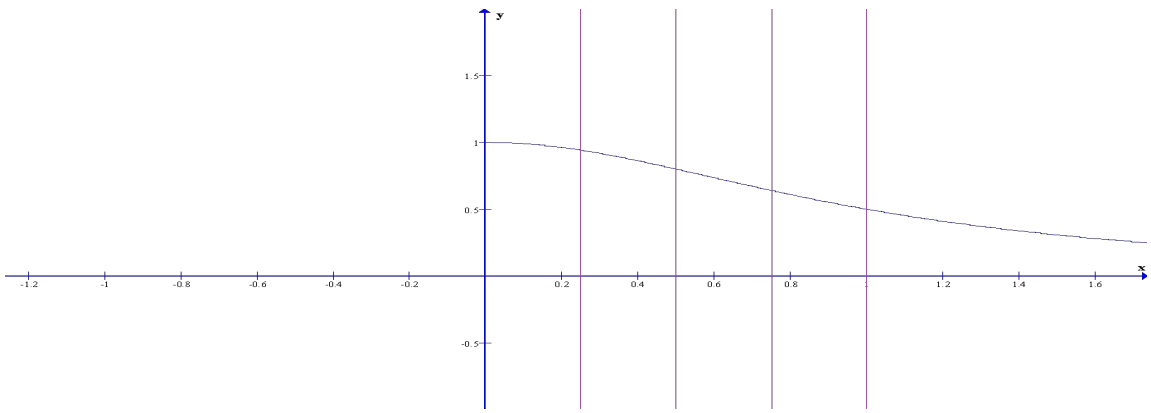
$$Speedup = \frac{Serial\ Time}{Parallel\ Time}$$

Serial Time = O(n), Parallel Time = O(n/p + p)

$$Speedup = \frac{np}{n+p^2} = p\ (n >> p)$$

where n is the number of steps, p is the number of cores.

Calculation of speedup does not include communication between CPUs.



f(x) = $\frac{4.0}{1+x^2}$

Optimization strategy

Assumption: Number of trapezoids n is evenly divisible across p processors

Each process has its own workload (interval to integrate)
• local number of trapezoids (local_n) = n/p
• local starting point (local_a) = a+(process_rank *local_n* h)
• local ending point (local_b) = (local_a + local_n * h)
Each process calculates its own integral for the local intervals
• For each of the local_n trapezoids calculate area
• Aggregate area for local_n trapezoids
If PROCESS_RANK == 0
• Receive messages (containing sub-interval area aggregates) from all processors
• Aggregate (ADD) all sub-interval areas
If PROCESS_RANK > 0
• Send sub-interval area to PROCESS_RANK(0)

Hardware Details

CPU Model: INTEL(R) Core i5-4590

No. of cores: 16

Memory: 7.6 GiB

Compiler: gcc

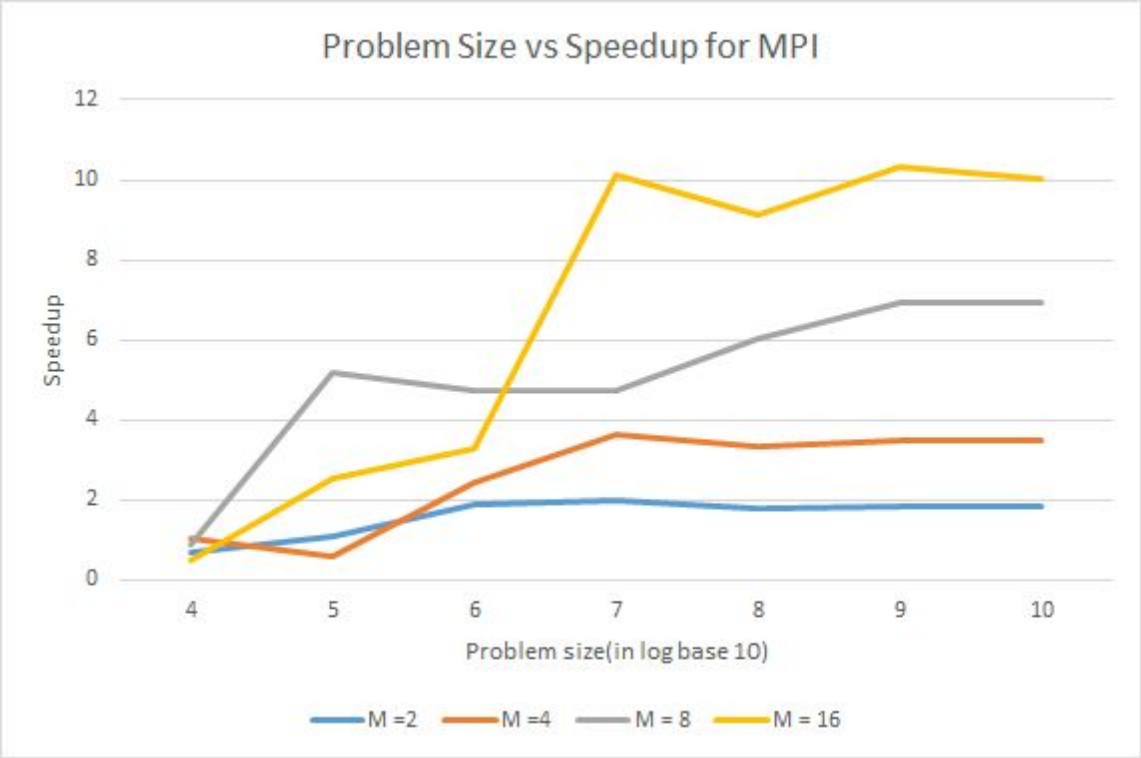
Optimization flags if used: None

Precision used: double point precision

Input Parameters: Number of steps, Number of threads or CPUs(for parallel code).

Output: Value of PI = 3.141593

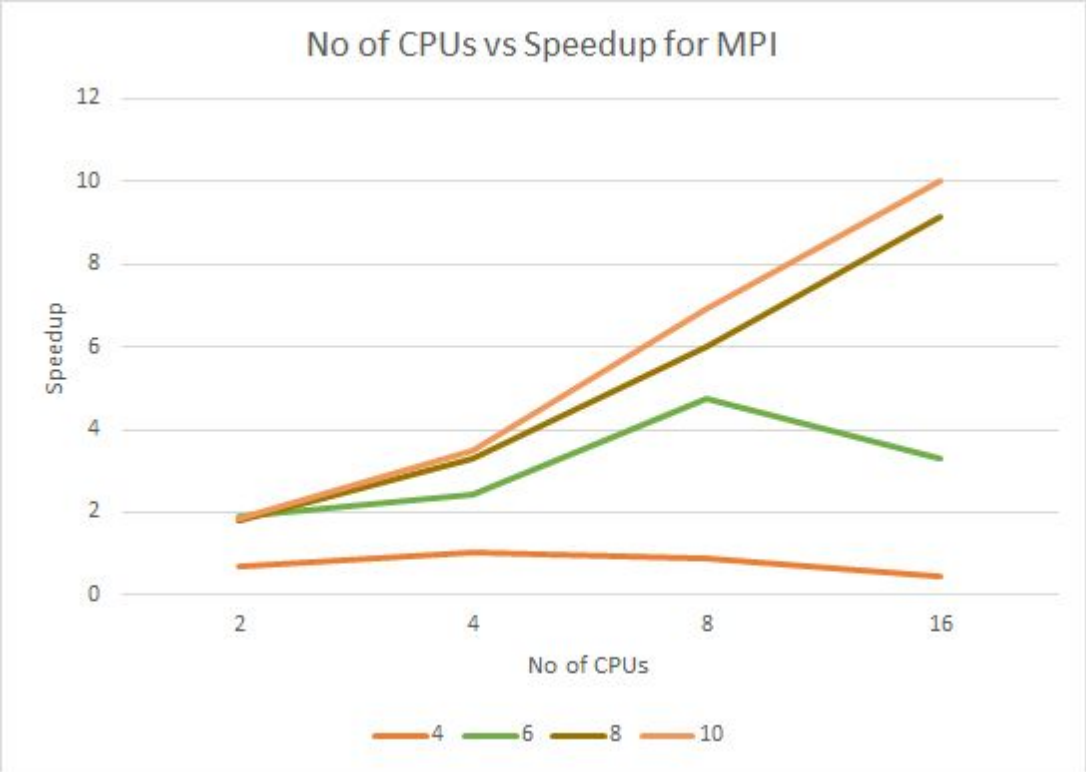
Problem Size vs speedup for different number of cores



Problem size vs speedup for 2, 4, 8 and 16 CPUs

Observation

As the problem size increases speedup increases, because the communication overhead is compensated by the computations for large problem sizes. For small problem sizes more number of processor shows lesser speedup than lower number processors, but it increases as the problem size increases.

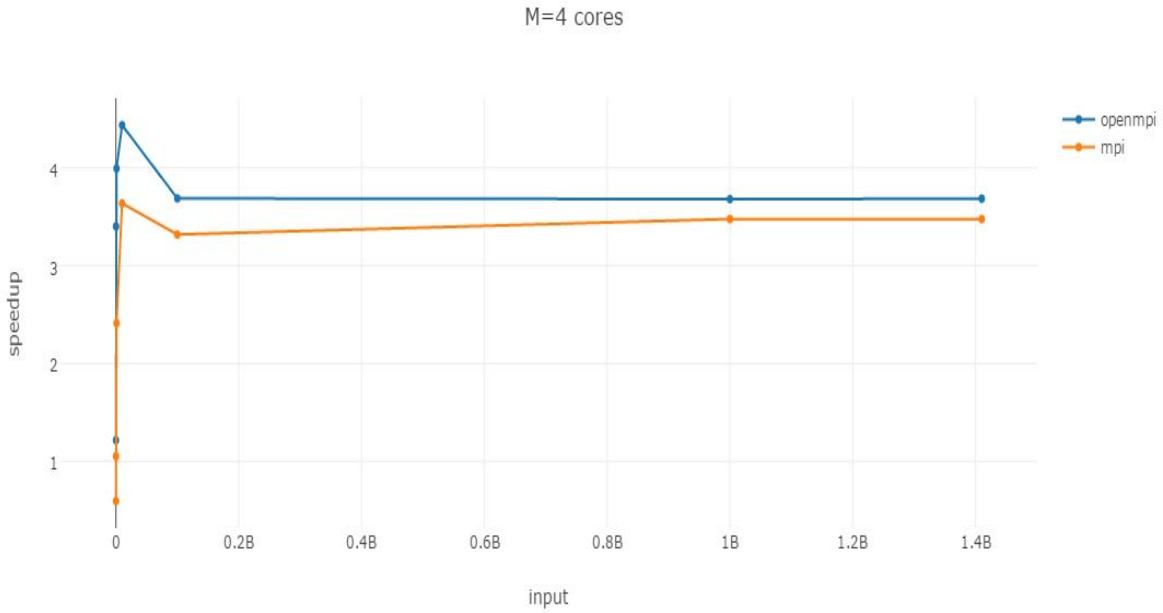


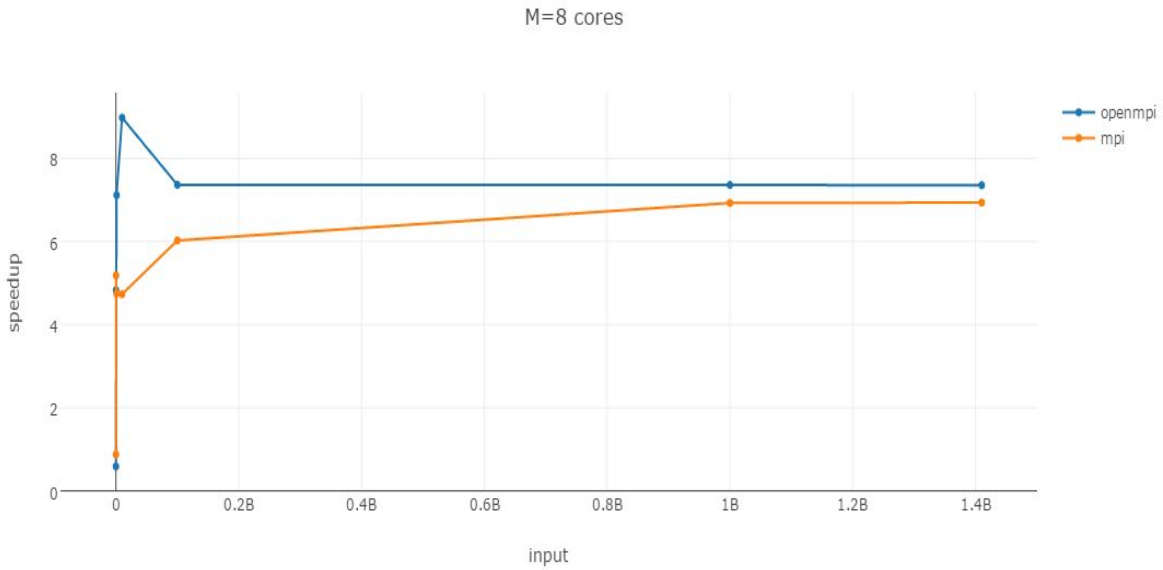
No. of CPUs vs speedup for problem sizes $10^4,\ 10^6,\ 10^8,\ 10^{10}$

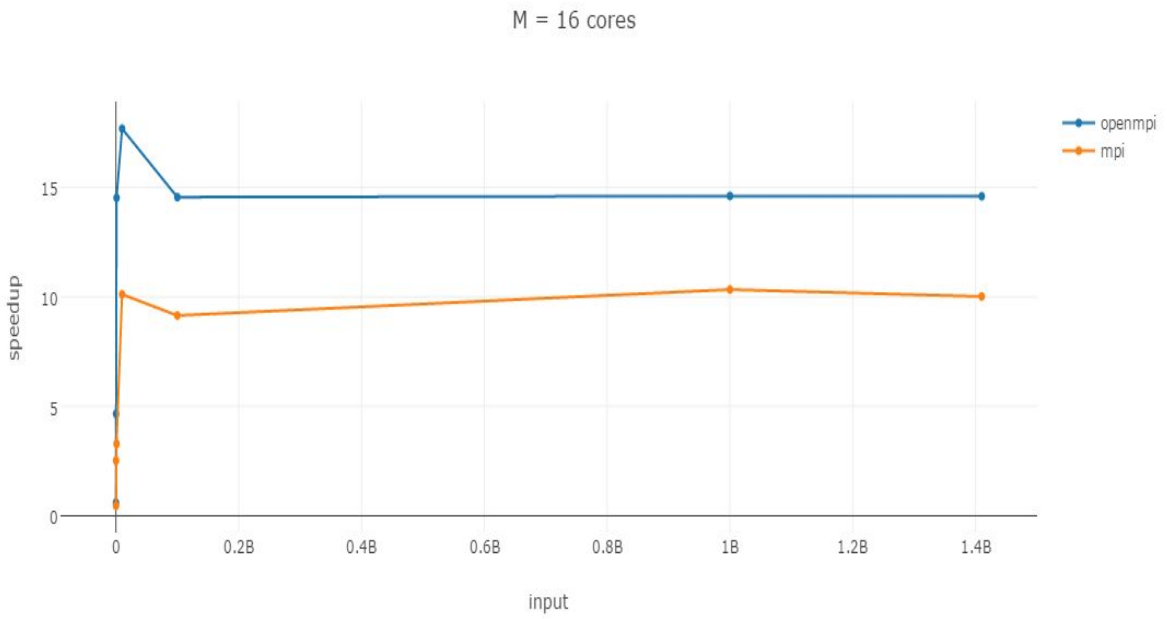
Observation

For the smaller problem sizes, speedup decreases for more number of CPUs. This is because of the overhead for communication between the processors is more than the decreased computation time due to parallelization. But as the problem sizes increases, more number of CPUs used for parallelization works better. We get the maximum speedup for problem size 10^{10} and number of CPUs equal to 16.

OPENMP AND MPI







Observation :

The Openmp system works more efficiently as compared to mpi system .As the number of cores increase the speedup tends to increase but in case of openmp the speedup is always more than speedup of mpi.Possible reasons may be that the communication overhead(synchronization) that is present in MPI is much larger than that of the synchronization(implicit barrier at the end of parallel construct of Openmp) of Openmp construct . This is because in MPI all the processor have their own distributed memory, whereas in OpenMP processors have shared memory. Therefore the speedup increases as processors(or threads in Openmp)increase but Openmp works better.

Problems faced in Parallelization and possible solutions

1. Synchronization problem is faced when we increase the number of processors as all the processor send their data to master process there is a possibility of synchronization overhead which reduces speedup
2. System overhead:the time spent when copying the message data from sender's message buffer to network and from network to the receiver's message buffer..
3. Instead of calling Send and Receive functions in processors we could also use the collective call "Reduce" function that would take calls from all other processes and pass it to a single(Rank = 0) process.