**CSE 6220 High Performance Computing**
**Programming Assignemnt 1**
**Submitted Wednesday, September 18 2024**

**Name: (Group 13)** Srinath Suresh, Kaitlyn Tung, Dhruv Raj Bangad

# 1 Report

## 1.1 For n = 109, plot a graph of run-time of the program vs. the number of processors for values of p = 1, 2, 4, 8, 16, 24. This run-time should include all computations that contribute to the estimation, including any local computations and global reductions. Include your graph and observation regarding the speedup.

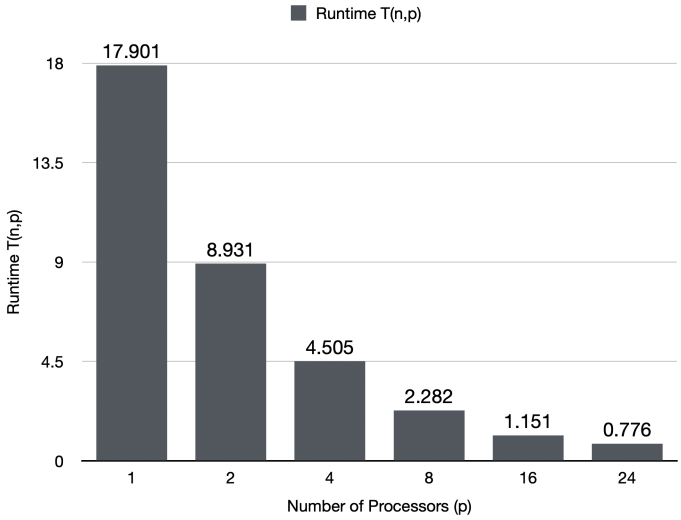


Figure 1: Runtime vs Number of Processors

| Number of Processors | $\pi$ Value | Runtime |
|---|---|---|
| 1 | 3.14159 | 17.901 |
| 2 | 3.14154 | 8.93106 |
| 4 | 3.14169 | 4.50544 |
| 8 | 3.14158 | 2.28186 |
| 16 | 3.1416 | 1.15132 |
| 24 | 3.14158 | 0.775531 |

## 1.2 Do the runtime analysis for T(n,1) (serial code), T(n,p) (parallel code) which includes both computation and communication cost and the speed-up in terms of n, p. This analysis will be on your entire code in "pi.h".

**Computation**:
$T(n, 1)$: the computation cost will be proportional to the problem size since we are looking at what happens when we have one processor. This would be $\Theta(n)$
$T(n, p)$: the computation cost will be $n/p$ since each processor is responsible for computing $n/p$ of

the total points needed

**Communication**:

$T(n, 1)$: there is no communication overhead since we are looking at the case where there is only one processor

$T(n, p)$: `MPI Reduce` combines our information from our processes into one. At each step, our process is communicating its data without another process. This data then communicates with another set of neighboring processes which have also communicated their data. This will take $\Theta(log(p))$ time to complete.

**Speedup**:

$S(p) = \frac{T(n,1)}{T(n,p)} = \frac{\Theta(n)}{\Theta(\frac{n}{p}+log(p))}$

# 2 Questions

## 2.1 Why is using `MPI Reduce` better than `MPI Gather`

Although both `MPI Reduce` and `MPI Gather` are used to combine data from all processes into a single process, they work in different ways. `MPI Reduce` performs a reduction operation (ex. in this case we are performing a sum operation for the points inside) to get the final result. This summation is directly applied across all processes. If we were to use `MPI Gather`, we would first need to gather the values from all processes into the root process. Then, we would have to manually sum the values in the root process, requiring a loop through the number of points inside. Therefore, `MPI Gather` requires more space and time than `MPI Reduce`, making `MPI Reduce` the better option.

## 2.2 Why is it necessary to have a random generator function for generating points? Why is necessary to have different seeds for the random generator in every processor?

The Monte Carlo method needs to have random points generated to create a proper estimation. Random generation means that the distribution of our points in the square is as unbiased as possible. We are assuming that this large sample of points is identically and independently distributed, which is important when we are using this statistical approximation method. Using different seeds ensures that we do not generate the exact same random points when we call `rand()` and we get duplicate random points across processes. If the same seed was used, there would be no purpose in parallelization since each processor would be running the same approximation using the same points. We get a better estimation of $\pi$ if each processor is performing operations on unique points.

## 2.3 Explain the functioning of every MPI instruction that you used in your program (`pi.h` and `pi.cpp`), also including the MPI setup instructions (eg. `MPI Init`, etc)

From `pi.h`

`MPI Comm size`: defines how many processes can be in a MPI communicator; in our case, we need to find the number of processors using `MPI_COMM_WORLD`

`MPI Wtime`: measures the time elapsed while a process has been running

`MPI Reduce`: combines data from the processes with a reduction operation (ex. sum in this case)

From `pi.cpp`

`MPI Init`: initializes the MPI environment

`MPI Comm size`: see above definition

`MPI Comm rank`: returns the rank, or identifier, for the process

`MPI Abort`: terminates all processes in the MPI communicator immediately

`MPI Finalize`: stops the MPI computation and also cleans up the resources used (ex. freeing allocated memory)

`MPI Wtime`: see definition above