Program Design

Objective : Have a program that can:

1. Generate Points in unit square randomly.
2. Check if these points lie in unit circle
3. Use threads to perform in parallel the above computations
4. Generate time needed to perform the above
5. Logs the results from computations and final results

For 1. and 2. , The class **Point** is defined as these operations are related.

- The Point class's default constructor is overridden to initialise the x-coordinate, y-coordinate with randomly generated values. Reference for Random Number Generation
- The constructor also calculates the squared distance of the point from the origin
- The method *is_point_in_circle* returns a boolean result, checking if the distance calculated above is within 1 or not.
- The method *to_string* returns the string representation of the point

For 3., the **struct thread_data** (typedef-ed to thread_data) is used to to pass relevant information from the main threads to created threads and back. This stores :

- **thread_id** : decimal enumeration of threads different from the actual thread-ids
- **num_points** : The number of **Point** instances to be generated by the threads
- **result** : The count of **Point** instances that are inside the circle
- **point_array** : Reference to an array of **Point** instances (generated in the created threads) to be used by the main thread for Logging purposes
- **point_status_array** : Reference to a boolean array which stores the boolean result of the **Point** being inside circle (This wasn't necessary but is used to prevent re-computation of the check at the time of logging)

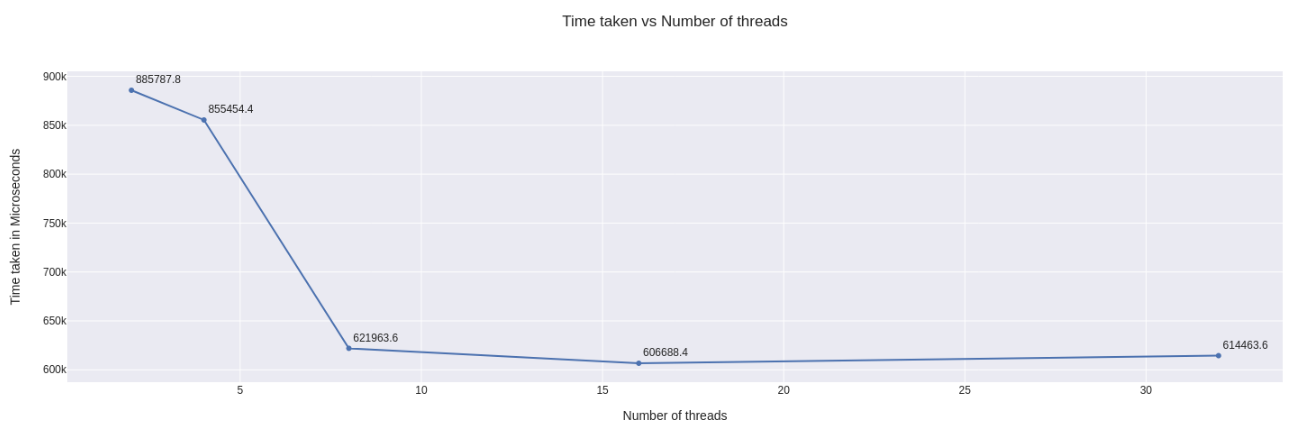
For 4., The features provided by the chrono library is used. Reference for using chrono library for timing code

For 5., Simple File I/O is used.

The main thread orchestrates the above 5 objectives. It consists of Initial Conditions I/O, Creation of threads and passing appropriate parameters. Joining the threads and Computing the final Pi value using Monte Carlo method formula. Logging of results obtained back from the threads stored in the structs.

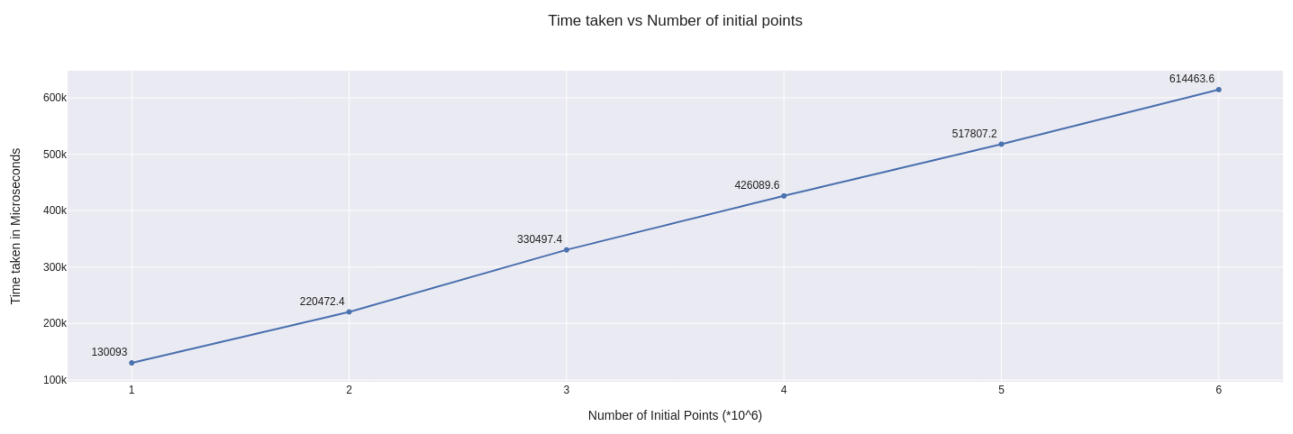
Results

Plot for time taken vs Number of Threads. Generated from ***threadvar.csv***



We can see that increasing the threads reduces the time taken for execution, in a manner that resembles the plots of Amdhal's law. Jump from 4 to 8 threads shows the maximum change in execution times, possibly resulting from the use of all 8 cores present on local machine. Increasing cores from 8 to 16 or 32 does not show appreciable decrease, possibly because the parallel component in code is not enough to take advantage of the hardware.

Plot for time taken vs Number of Points. Generated from pointvar.csv



We can see that keeping the number of threads constant, the time taken increases nearly linearly with the number of points provided. This is easy to conclude since workload on each thead is increasingly linearly as we are evenly distributing points to the threads.

Submitted By Divyansh Kharbanda, EP19BTECH11002