Assignment 1 Report

Tanmay Goyal
AI20BTECH11021

Program Design:

The program begins with reading from the input file ‘input.txt’ and getting the values of the number of random points to be generated and the number of threads we wish to create. If the file cannot be opened, we return an error.

We then define a class to store our Cartesian points, which consist of the following attributes: x-coordinate, y-coordinate, a thread_number indicating the thread the point has been allotted to and a Boolean of whether the point is within the circle or not. The thread_number helps in making sure that each thread knows which part of the global array it is responsible for.

One of the global arrays used is a starts array consisting of start_points for each thread. It is calculated as 1 + the last point seen by the previous thread. We keep this global so that our thread can also access it whenever required, instead of passing it down multiple times.

The other global array is the points_within_circle array where each thread returns its own values of the number of points within the circle it encountered.

Once we have the starts array, we create a 2D array, with the number of rows equal to the number of threads, and the number of columns equal to the number of points assigned to each thread as per the starts array. The advantage of the 2D array is we can directly pass a row of the empty array to the thread function, instead of having to return the set of randomly generated points from each thread, compiling them in the correct order along with the thread number and iterating through them. Each row consists of one more point than the required number; this extra point is the first point in each row, consisting of a dummy point, meant to convey the thread number to each thread.

We have used the srand() function to make sure every time the program is run, we get a different result. Each thread then generates the random points using two random floats between 0 and 1 using the rand() function and use the random floats to generate our random x and y coordinates. Whilst generating the points, it also checks if each point is within the circle, i.e the distance between the point and the origin is less than 1, since we assume a unit circle centered at the origin. It accordingly updates the within_circle attribute of every point as well. Finally, it also creates the point with the x and y coordinate, as well as the thread number and whether it is within the circle, so that the main thread can access it easily at a later stage. It also updates the global array with the number of points it encountered within the circle.

We start the timer to find the time required for creation of threads and time required for all threads to finish executing their tasks. We also wait for all the threads to finish, after which we join them and proceed. We end the time after that.

Once control returns to the main function, it simply adds all the values in the global array and calculates the value of pi according to the following formula:

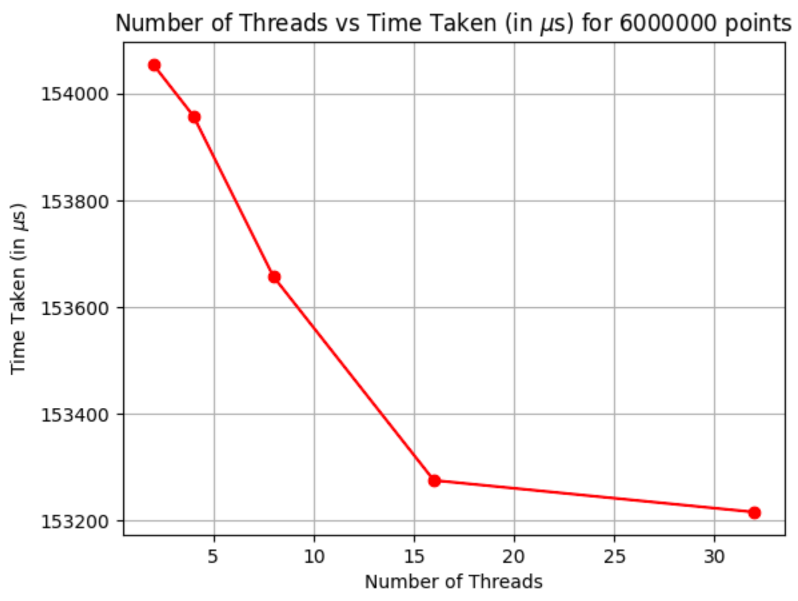
$$\pi = 4 \times \frac{\text{Number of points within circle}}{\text{Number of points within square}}$$

The main function then writes the values to the output file and starts compiling the values from all threads within the file.

Analysis:

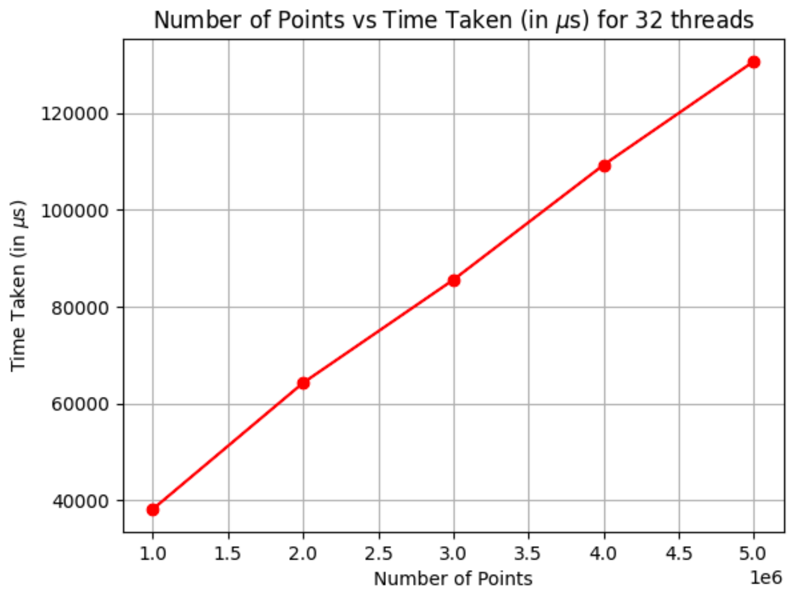
1. Number of Threads vs Time taken for 6000000 points

Number of Threads	Runtime 1 (in µs)	Runtime 2 (in µs)	Runtime 3 (in µs)	Runtime 4 (in µs)	Runtime 5 (in µs)	**Average time (in µs)**
2	153263	152893	157784	151913	154420	**154054.6**
4	156977	155073	153570	149598	154573	**153958.2**
8	155134	154705	152622	151604	154218	**153656.6**
16	151600	154106	155561	153575	151531	**153274.6**
32	152231	156282	153267	150044	154251	**153215**



2. Number of points vs Time Taken for 32 threads

Number of points	Runtime 1 (in µs)	Runtime 2 (in µs)	Runtime 3 (in µs)	Runtime 4 (in µs)	Runtime 5 (in µs)	**Average time (in µs)**
1000000	40441	35153	35476	38890	40970	38186
2000000	68685	66012	49506	70262	66850	64263
3000000	87176	88310	84680	90602	76995	85552.6
4000000	111288	110120	113457	112475	99083	109284.6
5000000	132174	133136	121852	131615	134616	130678.6


Number of Points vs Time Taken (in μs) for 32 threads
Time Taken (in μs)
Number of Points
1e6