Report

Base Idea :

Each thread will contain a N/K number of points and by multithreading we are to determine the number of points which lie inside the circle, and hence compute the value of pi which will be given as pi = 4*(no. of points inside the circle)/(no. of points inside the square).
But in order to be able to print the log, i,e the activity of each thread after the value of pi has been computed, we cannot print the points directly as soon as they are checked onto the output file, and thus they need to be stored. In order to store so many number of points in the order of 10^6, we shall dynamically allocate the memory, and in this method, the most efficient structure to use is struct.

Implementation :

Firstly, as in any code computing something, there are necessary libraries included to be able to use their functions at ease. Two quantities number of points and the number of threads will be used throughout the code through various functions and in main, hence they have been made as global variables. Now in order to store information, structs have been made.

1. Struct coord -> stores the x and y coordinates of a point. It also tells whether the point is inside the circle or not (by bool inside_thread)
2. Struct input -> contains all the info that has to be passed through the function routine.

Also array of objects namely points_square of type struct coord has been made global to make things simple.

Next we have the two functions.

1. Check_circle of boolean type helps us to know whether a point is within the circle or not.
2. Routine function passes all the points contained in a thread through check_circle and stores the information.

Now comes the main function. First, we read the input file and obtain the values of N and K respectively. Then we create an object of struct input named input_to_routine which serves the purpose as named. And then we allocate memory to both the object created and the global variable points_square.
Input_to_routine is technically an object which stores the thread id, the thread number and the N/K number of points. Hence we malloc it with K time the size of struct input.
Whereas point_square of type struct coord is supposed to store all the points to be generated in the square for our ease.

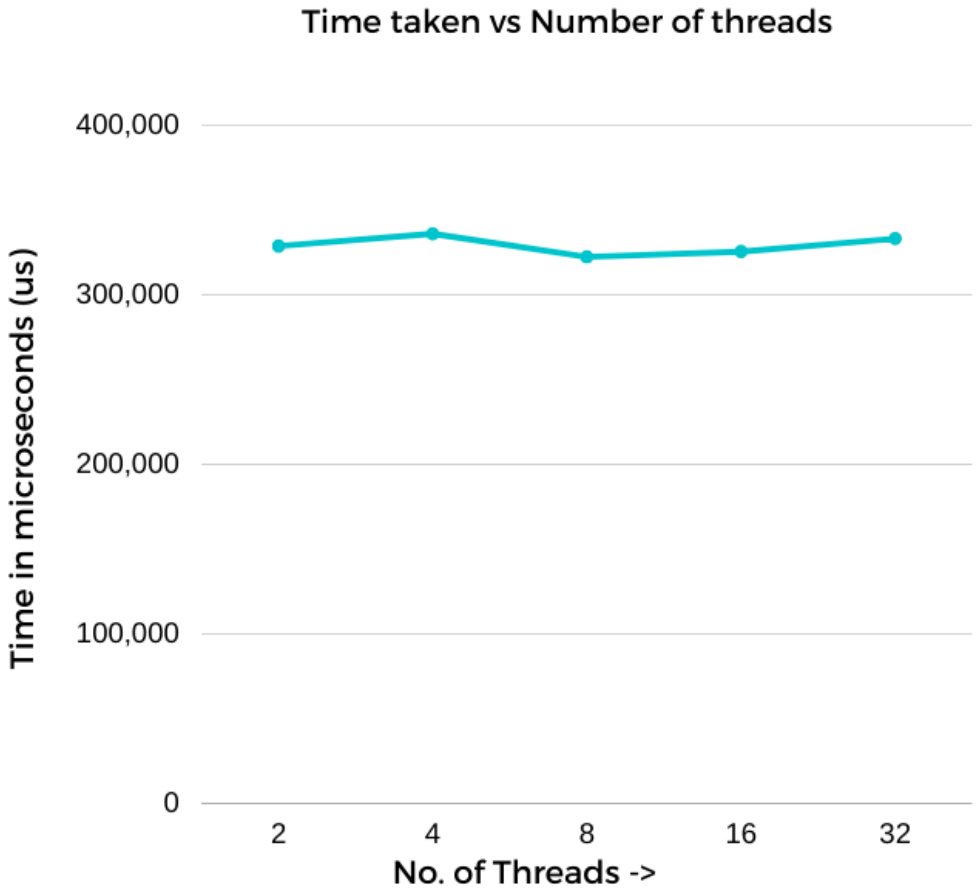
The next task is to generate random points by generating random numbers. Now in order for those points to be in the range of [-1, 1], they are divided by RAND_MAX (which the maximum number that can be generated by rand() function) and then multiply it by 2 and subtract with 1.

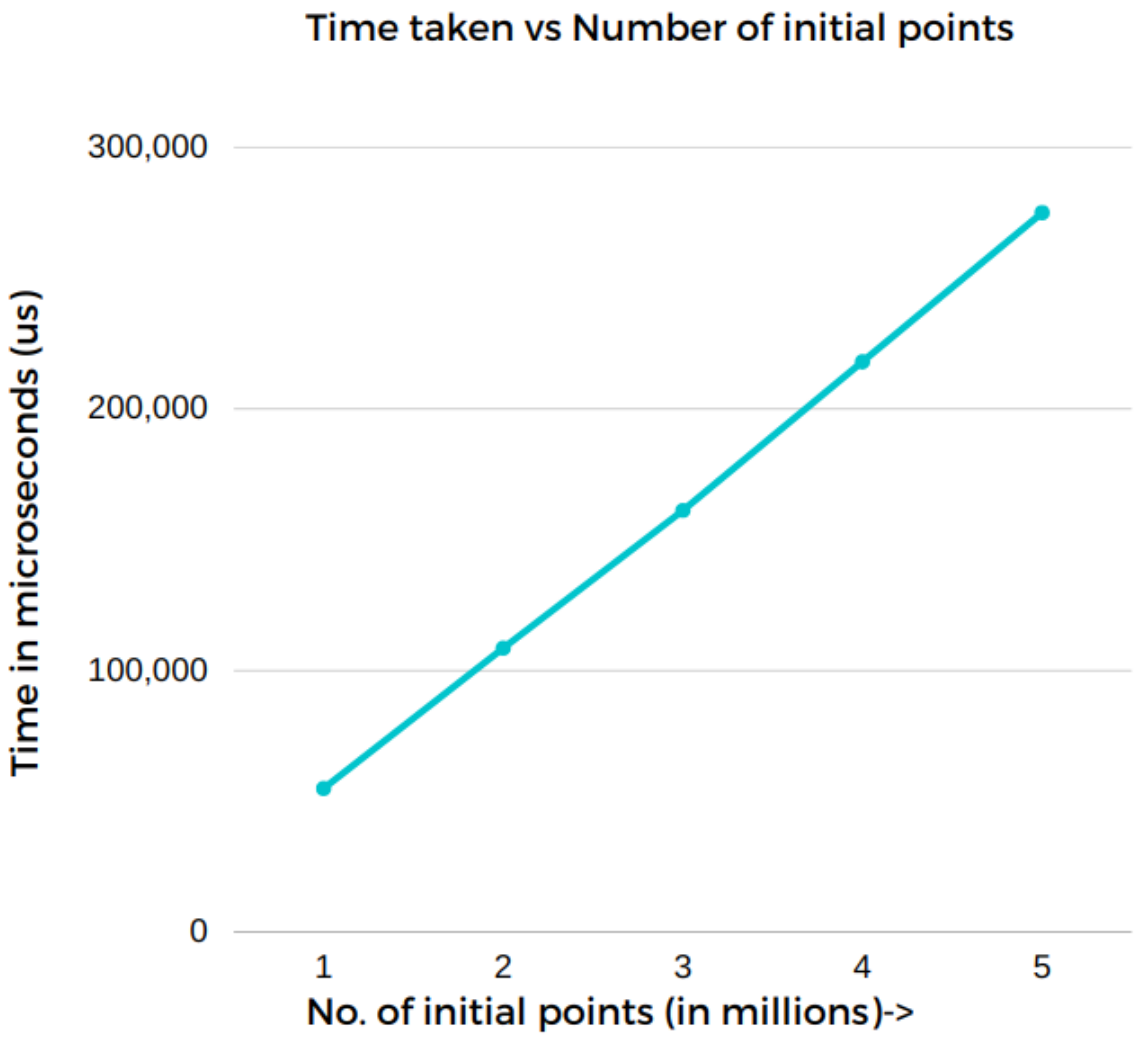
But since the main task has started, we note the time right before we start generating the random points.

Now we pay attention to each thread and since each thread will be passed to the function routine, we start assigning the information to the object input_to_routine. And since the same has to be done with all the K threads, we run a for loop K times. In each time, for each thread, we assign it a thread number and assign it's N/K points the values from points_square (which were generated earlier). And with this, we create threads which are then passed on to the function routine parallely.

As the threads are passed through the routine function, each point in the thread is being checked as to whether it lies in the circle or not and this information is then updated (in struct coord). And then by checking whether the value of inside_circle is 1 or not, we count the number of points present in the circle. And then we go on to calculate the value of pi. And as soon as the value of pi is calculated, we check the time and subtract from the previously noted time and we get the time taken in the whole process to calculate the value of pi.

Later, we just print out the results in the output file and and free the previously allocated memory. And hence the time taken to calculate the value of pi depends both on the number of points and the number of threads being given as input. For my program, the trend is as follows.




Time taken vs Number of initial points
300,000
200,000
100,000
0
Time in microseconds (us)
1
2
3
4
5
No. of initial points (in millions)->