Operating System-2

Programming Assignment 1: Parallel Monte Carlo technique for calculating π using Multi-threading in C++

Rajiv Shailesh Chitale
CS21BTECH11051

Program design:

Struct Point *is* used to store coordinates of a point.
Struct *RunnerThread* stores the tid and attributes of a thread.

Global variables used are:

```
long long k                 //number of threads
long long n                 //total number of points
long long* inCircleCount;   // array with count of points inside circle, for each thread
long long* inSquareCount;   // array with count of points inside square, for each thread
Point** circlePoints;       // array of k arrays: used for log of points inside circle
Point** squarePoints;       // array of k arrays: used for log of points inside square
```

Functions and Program Flow:

main
Parses optional arguments. It calls *input()* and *piFinder()* functions.

input
Reads n and k from an input file using ifstream, and updates respective global variables.

piFinder
It notes the start time using the chrono library.
It allocates memory to global arrays required for k threads, on the heap.
It creates k posix threads whose details are stored in a struct named *RunnerThread*.
They are made to run the function *runner().*
Each thread is fed a single argument called *tindex*, a unique index in 0 to k-1
Note: the argument needs void* so it is typecasted

It joins the k threads.
It sums up the number of points inside the circle, over each thread.
It computes π = 4 * total points inside circle/ total points inside square.
It notes the end time. It subtracts the start time to find the elapsed time.
It calls the *output()* function to display these results.
It deallocates memory from the global arrays.

runner
The thread determines '*total'*, the number of points it has to generate.
Suppose n points have labels p from [0, n-1], then each thread covers the set {p | p%k = tindex}
It allocates memory for the array of points inside the square and points inside the circle.

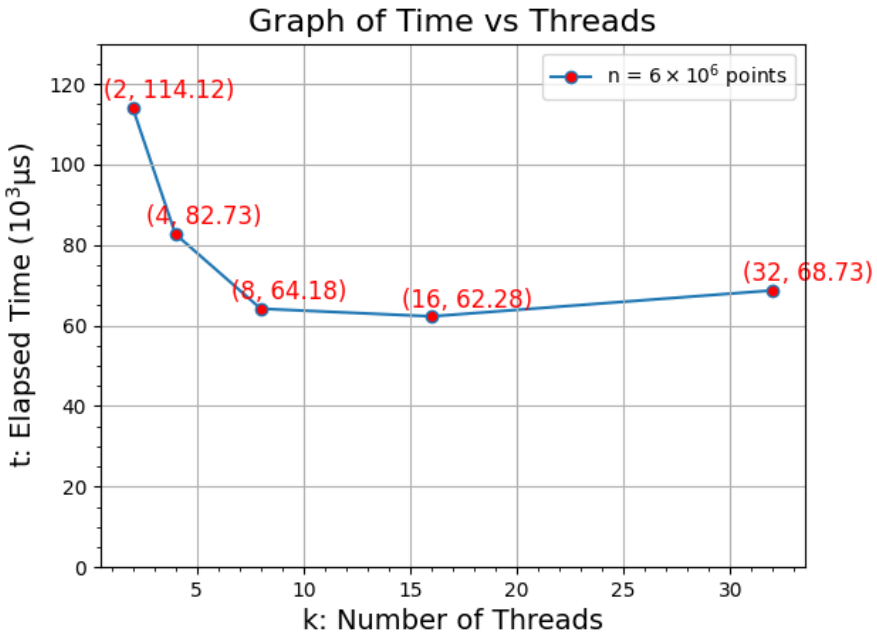
It uses time() and *tindex* to generate a fresh seed for the random number generator.
In a for loop, it generates two random coordinates. The value rand_r() / RAND_MAX lies in [0,1] and is transformed to lie in [-1, 1], the boundary of the square.

A point is deemed to be in the circle if its distance from the centre (and square of it) is less than 1. This check is implemented using pythagoras theorem. A counter is maintained for such points.
This function stores the number of points inside the square and points inside the circle in global arrays. The coordinates of points are stored in respective global arrays, used for logging.

output
Takes the elapsed time and computed value of pi as parameters. Displays them followed by a log. The log contains count and coordinates of points inside the square and points inside the circle.

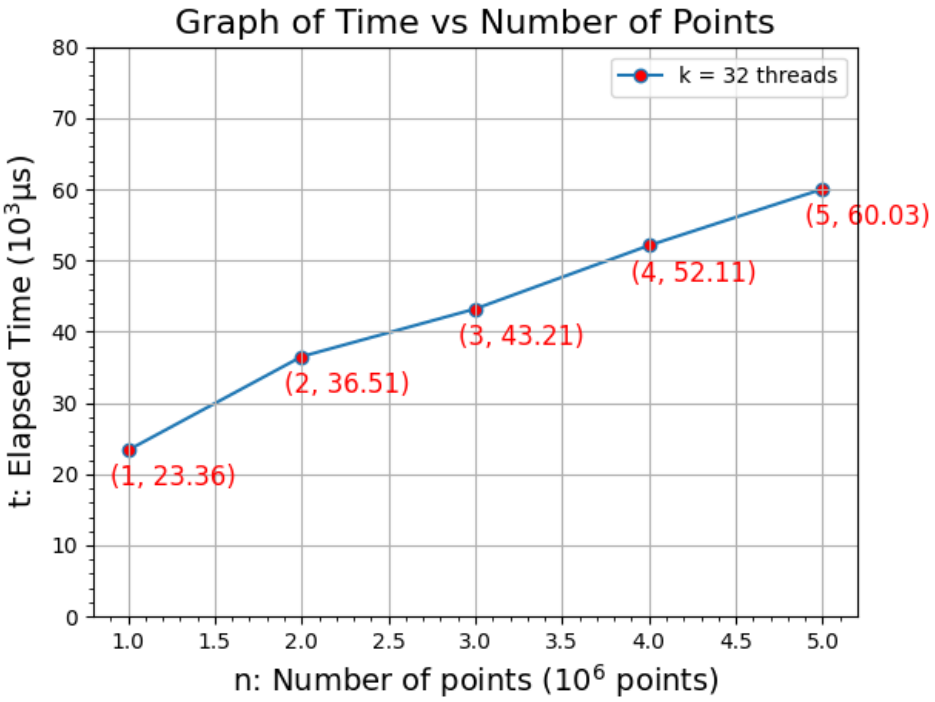
Plot1 - Time taken vs Number of threads:
Taking n = 6×10^6 as constant,

Number of Threads (k)	**$Time_1$**	**$Time_2$**	**$Time_3$**	**$Time_4$**	**$Time_5$**	**Avg. Time (t)** (in μs)
2	109,747	122,279	112,889	115,881	109,823	114,123.8
4	85,262	83,238	81,693	83,633	79,812	82,727.6
8	67,500	62,482	61,811	64,252	64,846	64,178.2
16	60,085	63,147	68,588	61,903	57,666	62,277.8
32	66,153	76,424	76,230	61,350	63,511	68,733.6



Plot2 - Time taken vs Number of initial points:
Taking k=32 as constant,

Number of Points (n)	**$Time_1$**	**$Time_2$**	**$Time_3$**	**$Time_4$**	**$Time_5$**	**Avg. Time (t)** (in µs)
1	24,833	18,237	23,710	28,620	21,394	23,358.8
2	36,321	34,907	36,346	35,835	39,159	36,513.6
3	44,145	41,056	47,414	41,399	42,033	43,209.4
4	46,862	53,209	56,467	47,136	56,880	52,110.8
5	59,850	61,236	59,281	56,528	63,238	60,026.6



Output analysis:
For n=6 million, k=8
Estimated π = 3.14195
This is accurate to 4 digits.

Plot 1
The elapsed time initially decreases as the number of threads (k) increases. After increasing the number of threads to over 8, there is little change. By 32 threads the runtime begins to increase slowly.

A speedup is only possible if threads run concurrently. This implies that my computer can only support execution of up to 8 threads concurrently. This is correct as my hardware has 4 cores with 2 hardware threads each making a total of 8 CPUs.

On a further increase in the number of threads, the threads compete for the CPU time. There is context switching between threads which begins to take a toll.

Plot 2
The elapsed time increases almost linearly with the number of points (n). Each thread generates approximately n/k points. As n increases, this phase of computation increases.