Name: Vishal Vijay Devadiga

Roll Number: CS21BTECH11061

Code Flow

- Opens input file in main and gets values of n,k
- Create k threads that execute the threadsolve function. Parent stores the threadID of the threads in an array.
  - Child Thread opens 2 log file to print the output of its calculations(circle points in circ.log and 'not in circle' points in notcirc.log)
  - The thread then prints the number of circle points and number of 'not in circle' points found to a heap array and returns the pointer to the array.
- While the threads are executing the threadsolve function at their pace, parent thread is in a loop that waits for a specific thread(identified by the threadID stored)
- If that thread finishes executing(exits), then parent thread accesses the array created by the child thread and then reads the number of circle points and number of 'not in circle' produced by the child. It accesses the log files created by that child thread and prints it to the main log file, and repeats the procedure for all of the children threads
- After all threads finishe executing(exits), the parent thread prints the value of pi calculated and time taken for the calculations above to the output file.

Plots

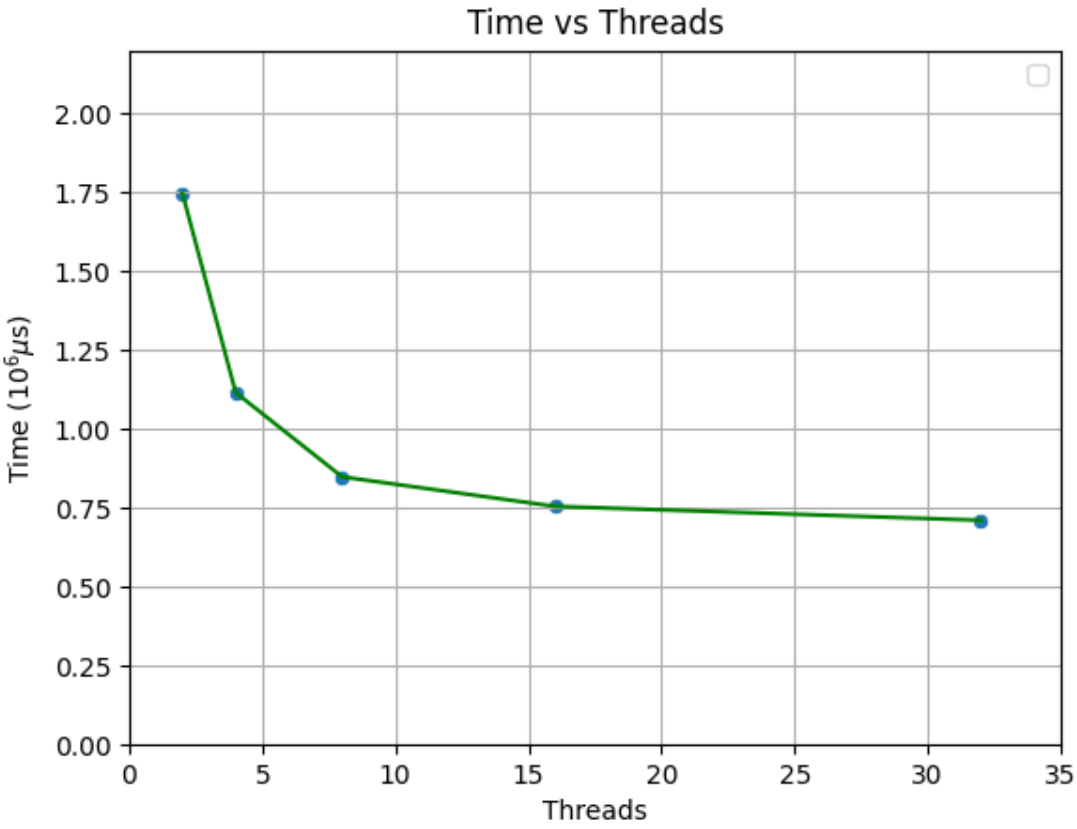
My system has the processor AMD Ryzen 7 5800u, which has 8 cores and 16 threads. Note that **all measurements of time are in microseconds(μs)**

Some Observations

Printing the log files takes the most amount of time in the program. The actual program without printing the log files takes a lot less time. Also, note that the log file is large and might crash the text editor being used(Using 5000000 points generates a file of size around 220 MB)

Increasing the number of threads after a certain amount actually increased the execution time of the program. Example: 512 threads vs 16 threads on my machine. One would expect the 512 threads program to execute in less time, but actually 16 threads program executes in less time.

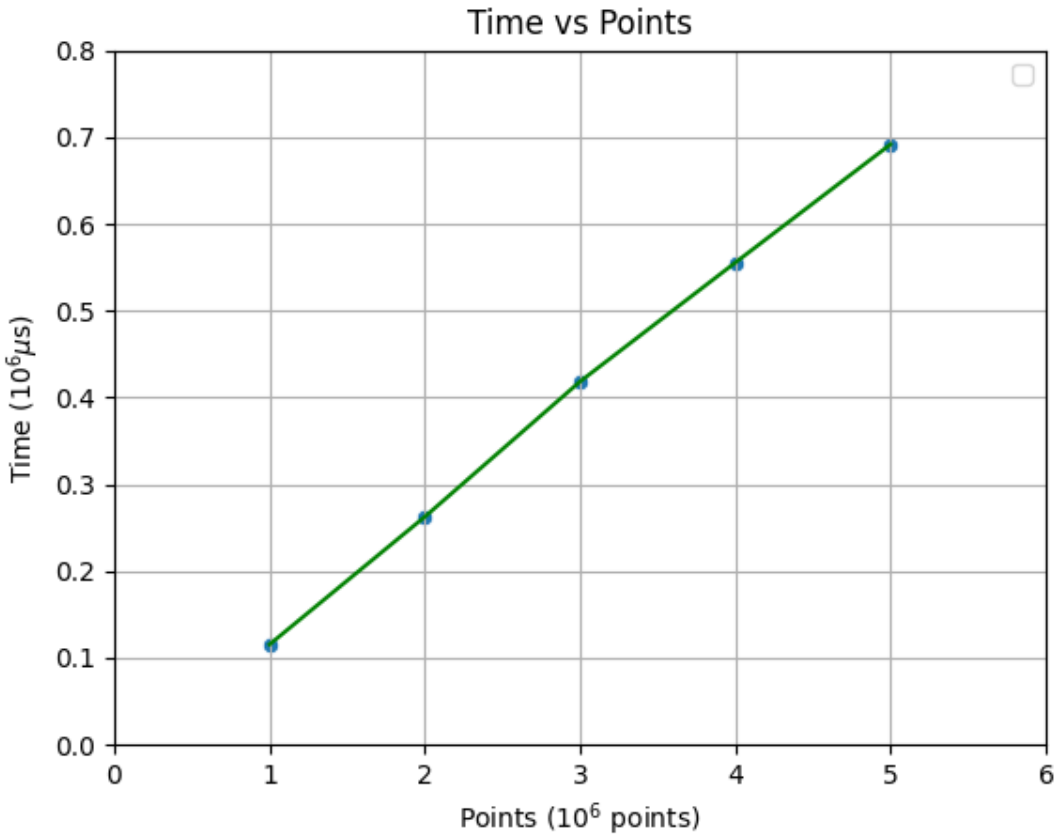
TIme vs Threads



Threads	Points	T_1	T_2	T_3	T_4	T_5	Average Time
2	6000000	1769247	1695148	1718875	1797328	1751171	1746353.8
4	6000000	1123782	1078404	1085019	1170165	1120873	1115648.6
8	6000000	818444	968035	823912	827325	805424	848628
16	6000000	765142	773924	745919	751284	739799	755213.6
32	6000000	742197	700290	706655	703048	703224	711082.8

By the graph, it is indicative that increasing the number of threads does increase the performance of the program, but not in a linear way. After a certain limit, increasing the number of threads does not yield increased performance and might even degrade performance. Both Amadahl's law(Sequential code in the parent thread limits the performance increase) and the number of cores in my system(8 cores) is the reason why the performance does not increase after using more than 16 threads for program.

Time vs Points



Points	Threads	T_1	T_2	T_3	T_4	T_5	Average Time
1000000	32	115319	106882	111131	117327	126840	115499.8
2000000	32	258127	265276	261909	261720	265739	262554.2
3000000	32	413150	417198	431268	413596	416522	418346.8
4000000	32	557417	561411	535880	584706	537325	555347.8
5000000	32	735659	676524	705314	677760	662852	691621.8

Points vs Time is a linear graph, since the points is equally distributed between the threads and threads is constant between the tests.