PROGRAMMING ASSIGNMENT 2

ABHAY SHANKAR K : CS21BTECH11001

Program flow:

- The main thread reads the values of n and k from the specfied input file, in the function `io()`.
- The `main()` function then calls `montecarlo()`, which calculates the number of random numbers that each thread must generate, and calls the `spawn()` function.
- The `spawn()` function calls `pthread_create()` k times, creating k threads which begin in the function `thread_init()`.
- `thread_init()` calls `gen_points()`, which generates the required number of random numbers.
- The random numbers generated by the string and data about the points (i.e. # points inside circle, # points total, etc.) are stored as member fields of `struct wdata`. Furthermore, they are classified into two `std::vector`s based on whether or not they are inside the circle.
- `gen_points()` returns the number of points inside the circle. A semaphore is opened, and this value is added to the global variable `in_circle` which combines all the threads' results.
- Within the same synchronised block, the `struct wdata` is pushed onto the global variable `std::vector rgen`.
- The function `join_all()` joins all threads to the main thread, taking a `std::vector` of pthread_id as an argument.
- Within `montecarlo()` again, the estimated value of pi and the time taken for the execution of the parallel portion of the code is calculated and printed into the output file.
- `spawn_io()` is called, and k new threads are formed, beginning in the function `thread_io()`. Each io thread writes the random numbers produced by one thread (stored as a single instance of `struct wdata` within `rgen`) into a separate file.
- The function `join_all()` is called again, and all the io threads are joined.
- `montecarlo()` returns, and within `main()`, `combine_files()` is called, and this appends the multiple logfiles produced by the io threads into the output file. It also deletes the intermediate files.

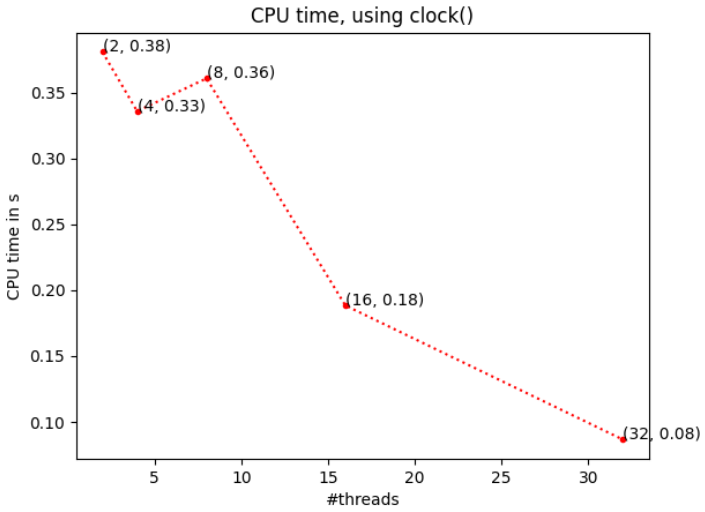
Benchmarking:

- Two functions are used: `clock()` and `clock_gettime()`.

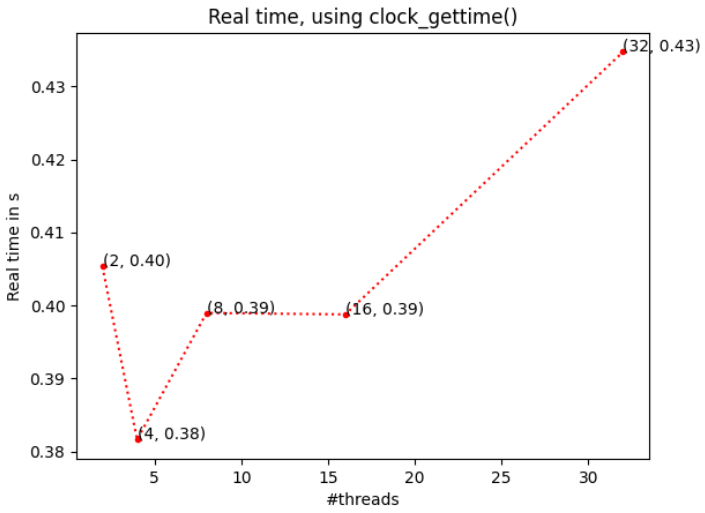
- `clock()` gives the CPU time utilised by the process since it began in μs - measuring before and after `spawn()` and dividing the difference by k yields an upper bound on the CPU time utilised by each computation thread.
- `clock_gettime()` gives the time elapsed since the Epoch. Measuring twice and subtracting yields an upper bound on the actual time interval during which the MonteCarlo estimation occurs.
- Both these times are plotted in the graphs.

Graphs:

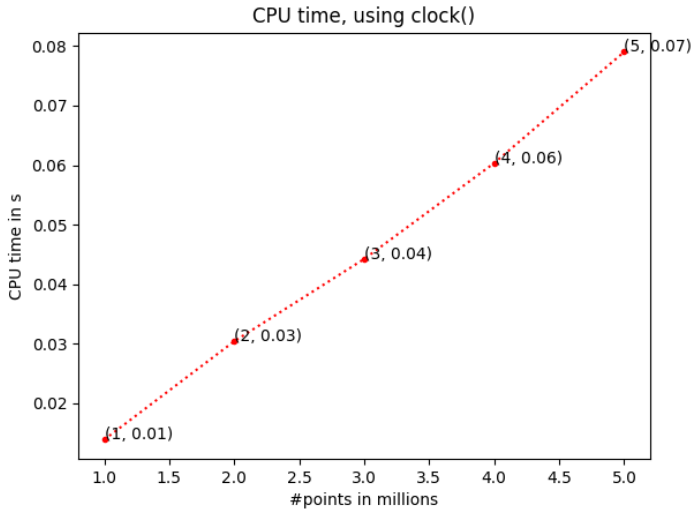
- k variation:
  - CPU



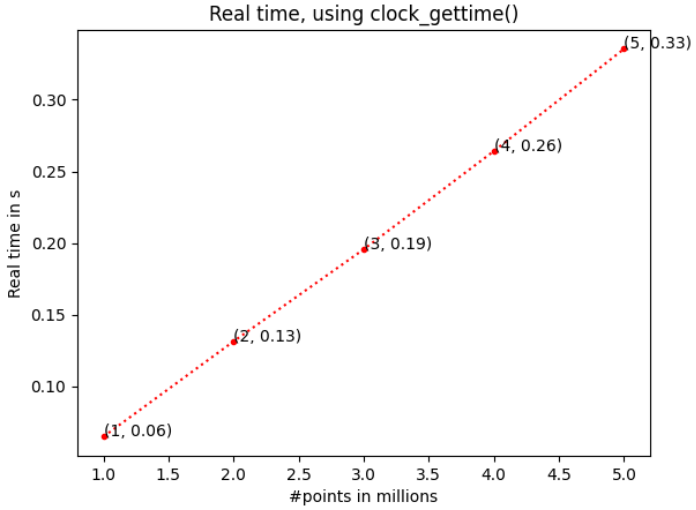
  - Realtime



- n variation:



– CPU



– Realtime

Formats:

- Input: NA
- Input file: `./inp.txt`
- Input Format: `n k`
- Output: Value of pi, time taken to compute the value
- Output file: `./output.txt`
- Output Format:
  `Value: <pi>`
  `Ticks: <CPU time in μs> Time: <Realtime in μs>`
  `Thread <num>: <#Inside circle> <#Outside circle>`
  `Points inside square: [points]`
  `Points inside circle: [points]`
  ⋮

Output Analysis:

- The output is an estimated value of pi.
- This estimation is predicated on the fact that computer RNG yields a uniform distribution.
- We generate two independent random variables, and due to the ature of uniform distributions, upon plotting these random numbers on orthogonal axes, they are uniformly distributed over a unit square,
- Scaling and translating, we get a uniform distribution of points over a square of side length 2 centered at the origin. Because the distribution is uniform, the ratio of areas of this square and the unit circle ($\frac{\pi}{4}$) is equal to the fraction of points within unit distance from origin.
- Performing the appropriate arithmetic, we can estimate pi.

Structs:

- point: Just a pair of doubles.
- wdata: Contains the entire result of a single thread, including random points classified by their Euclidean norms $\gtrless 1$, their lengths, and the thread number.

Functions:

- `main()`: Minimal functionality. Calls other functions which estimate and print the value of pi. Also prints the time taken by the serial part of the function
- `io()`: Reads the values of n and k from the designated input file and returns them as a `std::pair`.
- `montecarlo()`: Calls the functions to spawn and join all threads. It also performs the necessary arithmetic, given the number of points within unit distance of origin, to compute pi. Prints the benchmark times as well as the estimate for pi into the output file.

- `spawn()`: Spawns the required number of threads (taken as argument). Stores the pthread_t id's of the threads in a `std::vector` (taken as argument). Returns the value of `clock()` called immediately before the threads are spawned.
- `join_all()`: Joins all the threads whose pthread_id is stored in a `std::vector` taken as argument. Returns the value of `clock()` called immediately after the threads are joined.
- `thread_init()`: Calls `gen_points()`, stored the generated numbers and the extracted metadata in a `struct wdata`, and pushes it onto the global `std::vector`.
- `gen_points()`: RNG occurs here. Their Euclidean norms are evaluated and, depending on the value, they are pushed onto one of two possible `std::vector`s.
- `inside()`: Evaluates the Euclidean norm of a pair of reals.
- `spawn_io()`: Spawns k threads, storing the pthread_t id's of the threads in a `std::vector` (taken as argument).
- `thread_io()`: Writes the points from the global `std::vector` into multiple files using multiple threads.
- `combine_files()`: Concatenates multiple files produced by `spawn_io()`. Cleans up the various intermediate files.