CSE 6010
Workshop 5

Due Date: 11:59pm on Thursday, November 9
Submit 'pi.c', 'contrib.txt', and 'howthingswent.txt" to Gradescope through Canvas
48-hour grace period applies to all deadlines

This workshop will combine **random numbers** and **OpenMP**. Thus, **we highly recommend that you go through the programming practices associated with OpenMP and with random numbers** before attempting this workshop so that you have some familiarity with both topics separately.

For this assignment, you may work in groups of 2-3 students. If you prefer, you also may choose to work independently.

If you choose to work as a team, you are expected to **work together for the full assignment**. A divide-and-conquer approach, where each team member works independently on some part of the assignment and the team only interacts when they combine their separate codes to submit, is not in the spirit of this assignment. If you don't want to work with anyone, you should submit independently.

The workshop is designed so that we expect most of you will be able to finish most of the assignment during the **75-minute class period**, possibly with a little extra time offline for cleanup and submission. Nevertheless, we know that different individuals and groups will have different levels of comfort/proficiency with C, and that programming does not always follow a timeline. Thus, it is not strictly required that every submission necessarily will include all topics. As part of your submission, please write a few sentences about how things went in a text file named **'howthingswent.txt'**. This is your opportunity to let us know if you got bogged down in a particular area and how you went about resolving any problems you may have encountered.

Contributions

We ask you to identify how each team member participated by filling in and including the contrib.txt file provided. For each problem, please assess the contribution level for each category according to the following scale.

- 3 = >50% (student did the majority of the work for that category – so there can only be a maximum of one "3" per category + problem combination)
- 2 = 20-50% (solid contribution)
- 1 = 1-20% (minor contribution)
- 0 = no contribution

The categories are:

- Ideas / planning
- Detailed code design
- Writing code / implementation
- Debugging / testing
- Documentation / comments

You can find the template 'contrib.txt' file on Canvas under Workshop 1. When submitting to Gradescope, make sure your filename is '**contrib.txt**'. **Submit this file even if you are a "group" with one member;** delete the space for contributions by other participants when not needed.

Your assignment: Using OpenMP, write a C code to approximate the value of π using a Monte Carlo method. Essentially, you will generate random points in the square bounded by (0,0), (1,0), (1,1), and (0,1). (That is, you will generate a random x value between 0 and 1 and a random y value between 0 and 1 for each point.) You will count the number of points within the quarter of a unit circle centered at the origin (that is, the number of points such that the sum of the squares of the x- and y-coordinates is less than 1). Finally, you will form the ratio of the number of points within the unit circle to the total number of points and multiply by 4: this value is an approximation of π! (See below for details and an illustration.)

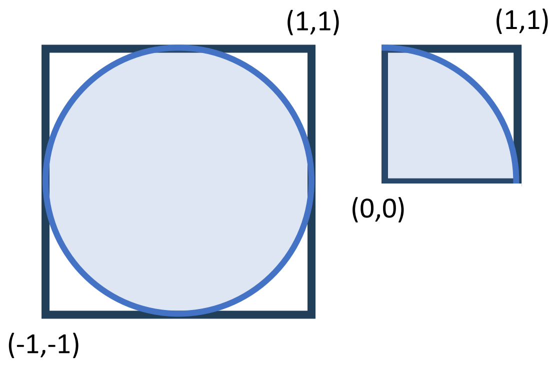
Note that although the specifications below may seem long, your program in the end should need only around 40 lines of code (excluding blank lines and commenting).

Specifications:

- Input and setup:
  - The program should read in the **number of random points to generate** within the square as the ***first command-line argument***. You do not need to perform any validation and may assume that the user will specify a valid integer number of points within the limits of integer precision. Don't forget to use `atoi()`.
  - The program should read in the **number of OpenMP threads** to use in the parallel portion of the code as the ***second command-line argument***. You do not need to perform any validation and may assume that the user will specify a valid positive integer number of threads. Don't forget to use `atoi()`.
- Monte Carlo algorithm:
  - Use the thread-safe pseudo-random number generator `rand_r(&seed)` to generate random numbers; the numbers will be positive integers up to the system constant `RAND_MAX`. To ensure they are between 0 and 1, divide the generated number by `RAND_MAX`. Note that because the randomly generated number and `RAND_MAX` are both integers, you will need to cast at least one to a double; otherwise, the result of the division will be an integer (basically, it will be 0). You can cast to a double by writing `(double)` (including the parentheses) before the value you wish to cast. You can cast just one of the operands to a double; C will promote the other operand to the same type. (E.g., the following are equivalent for integers a and b: `(double) a / b`, `a / (double) b`, and `(double) a / (double) b`.
  - Choose a seed that will vary each time you run the program by basing it on the current time. You can use `(unsigned int) time(NULL)` to generate an unsigned integer based on the current time. You will need to include the header file `time.h`.
  - Note that you should use a different seed for each thread; if you use the same thread, each thread will perform the exact same calculations, resulting in a

duplicative effort rather than accomplishing new work. You can set a thread-specific seed, e.g., by adding the thread number (`omp_get_thread_num()`) to the "base" seed derived from the current time.
  - Generate two random numbers to represent the x- and y- coordinates of a point in the unit square (whose four corners are listed above).
  - For each (x, y) pair of random numbers, determine whether the point lies within the quarter-circle by calculating whether $x^2 + y^2 < 1$. (In practice it should not matter whether you use $<$ or $\leq$). Keep count of the number of points that satisfy this condition.
- Parallelization with OpenMP:
  - Use the fork-join approach of OpenMP to distribute the workload of calculating random points and evaluating whether they fall within the quarter-circle. Add appropriate `#pragma` directives as needed. Hint: you likely will want to use a reduce operation to combine the results from the different threads.
  - Set the number of threads in the program to use the number specified as a command-line argument. You may do this using the function `omp_set_num_threads()` or by adding the `num_threads()` clause to your `#pragma`.
- Timing:
  - Use `omp_get_wtime()` to obtain time information while your program is running. Record the time immediately before the parallel section begins and immediately after the parallel section ends, then calculate the elapsed time as the difference in those times.
  - You should expect to see some speedup when more threads are used for many points (e.g., millions). If you are not seeing speedup on your local system, try the Gradescope environment and/or the COC-ICE cluster.
- Output and cleanup:
  - Calculate the approximation of π by taking the count of the number of points inside the quarter-circle, dividing by the total number of points, and multiplying by 4 (as explained below).
  - Your program should output two values, the approximation of π and the elapsed time, using a statement like the following (include a comma between the values and a newline character at the end):
    ```
    printf("%f, %f\n", pi_estimate, elapsed_time);
    ```
  - Any dynamically allocated memory should be freed (consider using valgrind to test for memory leaks).
- Compilation:
  - Don't forget to include `–fopenmp` when compiling your code (once you are using OpenMP)!
  - We also recommend you include the compiler flag `–O3`, which will allow some optimizations that will make your code run faster.
- Submission:
  - When submitting to Gradescope, make sure your code is in a single file named **'pi.c'**. For this assignment, please keep all your code including function prototypes and definitions in this one file.

More on how/why this approach works: see the figure for reference. For the full circle, we know that the area of the square is the square of its length (which is twice the radius of the circle), and that the area of the circle is the product of π and the square of its radius. Thus, for any radius r the area of the square is $A_{square} = (2r)^2 = 4r^2$, and for the corresponding circle the area is $A_{circle} = \pi r^2$. Therefore, if we knew the areas of the square and the circle, even without knowing the radius, we could calculate π as $\frac{A_{circle}}{r^2} = 4\frac{A_{circle}}{A_{square}}$.



The same logic applies to the quarter-circle and its surrounding square, except now the area of the square is r^2 and the area of the quarter-circle is $\pi r^2/4$. The expression for π remains the same because the ratio is the same: $\pi = \frac{A_{quarter-circle}}{r^2} = 4\frac{A_{quarter-circle}}{A_{small\ square}}$. We will use the quarter-circle version as it slightly simplifies the implementation and discussion.

Thus, if we know the areas of the quarter-circle and the surrounding square, we can calculate the value of π. We will work in a unit square so that the center of the quarter-circle is at the origin and the radius is 1. Here, we will *approximate* the area of the circle by randomly choosing points within the unit square and testing whether they are also within the quarter-circle. The ratio of the area of the quarter-circle to the area of the surrounding square will be proportional to the fraction of points in the square that are also in the quarter-circle. So, we can approximate the ratio of the areas, $\frac{A_{quarter-circle}}{A_{square}}$, as the number of points within the circle divided by the total number of points. Then we need only multiply this ratio by 4 to get an approximation for π!

Note that this Monte Carlo method for approximating π, while interesting, is highly inefficient. Nevertheless, it is a good way to get some more experience with both random numbers and OpenMP.

Group submissions

Submit your code file named **'pi.c'** along with **'contrib.txt'** and **'howthingswent.txt'** on **Gradescope** under **Workshop 5**. If you don't know how to submit on Gradescope as a group, you can follow this tutorial on YouTube.