Assignment #1

Monte Carlo Method for π

π is the area of the disk with the radius r equals to one. The approximating of π can be performed in the following steps:

- Inscribe a circle with radius r in a square with side length of $2r$
- The area of the circle is πr^2 and the area of the square is $4r^2$
- The ratio of the area of the circle to the area of the square is $\frac{\pi r^2}{4r^2} = \frac{\pi}{4}$
- A simple way to compute the fraction is to generate the points in the square and count the number of points which lie in the inside of the circle
- Therefore, π is approximated as $\pi = 4 * \frac{number\ of\ points\ inside\ the\ ciricle}{number\ of\ all\ points}$

The Monte Carlo methods are a class of algorithms that rely on repeated random sampling to calculate the result. To compute the value of π, it generates points in the square [-1, 1] * [-1, 1] and counts the number of points in the inside of the unit circle. The result is four times the fraction between the number of points which lie in the inside of the circle and the number of all generated points.

Given the sequential Monte Carlo Method for π program below, parallelize it with OpenMP.

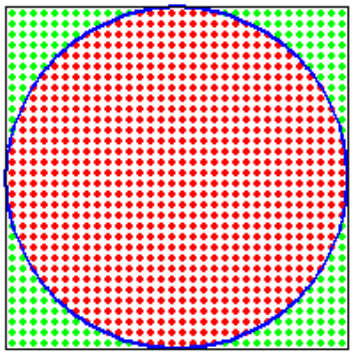
```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

main(int argc, char *argv[])
{
  int i, count;  // points inside the unit quarter circle
  double x, y; // coordinates of points
  int samples; // samples number of points to generate
  double pi; // estimate of pi

  samples = atoi(argv[1]);

  double start = omp_get_wtime();
  for (i = 0; i < samples; i++) {
    x = (double) rand() / RAND_MAX;
    y = (double) rand() / RAND_MAX;
    if (x*x + y*y <= 1)
      count++;
  }
  double end= omp_get_wtime();
  printf("elapsed time: %.16g\n", end - start);

  pi = 4.0 * (double)count/(double)samples;
  printf("Count = %d, Samples = %d, Estimate of pi = %7.5f\n", count, samples, pi);
}
```



Requirements:

1. Write the parallelized OpenMP code in either C or Fortran; try to maximize the performance with what you learned in class;
2. Take a second command argument as the number of threads variable, such as
   int nthreads = atoi(argv[2]);
3. Set the number of threads in OpenMP using nthreads.
4. Fill in the table below with varying sets of threads and samples for your parallel program (top to down in the row direction: increasing number of threads in use; left to right in the column direction: increasing number of samples of points of the problem)

sec	100	100,000	10,000,000
1			
2			
4			
8			
16			

5. Turn in your code and table through canvas.