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 MPI.

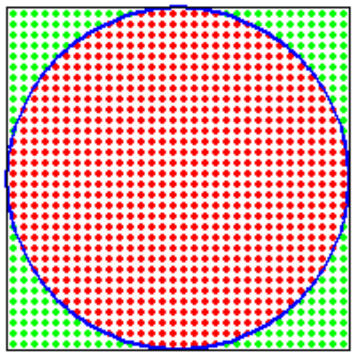
```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <math.h>
#include <time.h>

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
  int rank, numtasks;

  samples = atoi(argv[1]);

  count = 0;
  for (i = 0; i < samples; i++) {
    x = (double) rand() / RAND_MAX;
    y = (double) rand() / RAND_MAX;
    if (x*x + y*y <= 1)
      count++;
  }

  pi = 4.0 * (double)count/(double)samples;
  printf("Count = %d, Samples = %d, Estimate of pi = %7.5f\n", count, samples, pi);
  return 0;
}
```



Requirements:

1. Write the parallelized MPI code in C;
2. Distribute the samples to multiple processes evenly, i.e, if you have *np* processes, the number of samples per process is *samples/np*;
3. Calculate the partial count on each process and reduce the final count to process 0;
4. Measure elapsed time by using MPI_Wtime() and print it on process 0;

5. Print the estimated pi on process 0;
6. Request 16 mpiprocs on palmetto in an interactive mode with:
   qsub -I -l select=1:ncpus=16:mpiprocs=16,walltime=00:10:00
   load modules: module add openmpi/1.10.3-gcc/5.4.0-cuda9-2
   compile your code with mpicc
7. Fill in the table below with varying sets of samples and mpi processes (e.g., “mpirun -np 4 --mca mpi_cuda_support 0 *yourexecutable* 100000” will compute the pi with 100,000 samples on 4 processes) and record the elapsed time.

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

8. Provide a plot of the runtime of your parallel implementation as a function of increasing processes (1 to 16) for 10,000,000 samples based on your results in table 7.
9. Turn in your source code, table, and plot through Canvas.