CX 4220 / CSE 6220 Introduction to High Performance Computing Spring 2019 Programming Assignment 1 (**Revised**) Due February 5, 2019

This is an introductory programming assignment meant for you to get your bearing in programming in C/C++ and some basic MPI commands. You are encouraged to discuss this in piazza in order to workout how to solve the problem.

Problem Description

In this programming assignment, you will calculate PI using a dartboard algorithm. While plenty of efficient algorithms exist to calculate PI upto hundreds of digits, dartboard algorithm is a simple method that is easy to parallelize. The original dartboard algorithm is as follows: you throw N darts randomly at a square board with a circular front (see Figure 1).

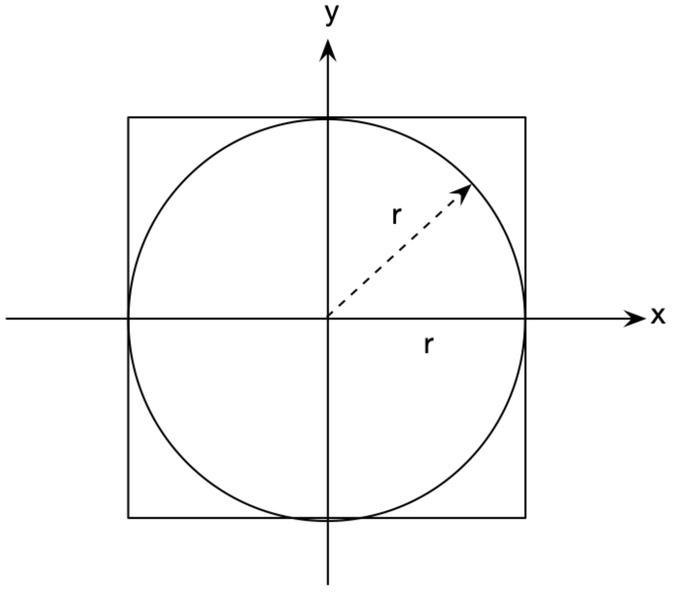

Figure 1: Original dartboard

Let M be the number of darts that land in the circular region. For large N, the ratio $\frac{M}{N}$ approximates to the ratio of area of the circle to the area of the square. In other words,

$$\begin{aligned} \frac{M}{N} &\approx \frac{\pi r^2}{4r^2} \\ \implies \pi &\approx 4 * \frac{M}{N} \end{aligned} \tag{1}$$

Therefore PI can be calculated by counting M and N and plugging the values in above equation.

However, in this project, we are using a slightly **modified** version of the original dartboard algorithm. Instead of a square board with an inscribed circular frontend, we have a circular board with an inscribed square frontend (as shown in Figure 2).

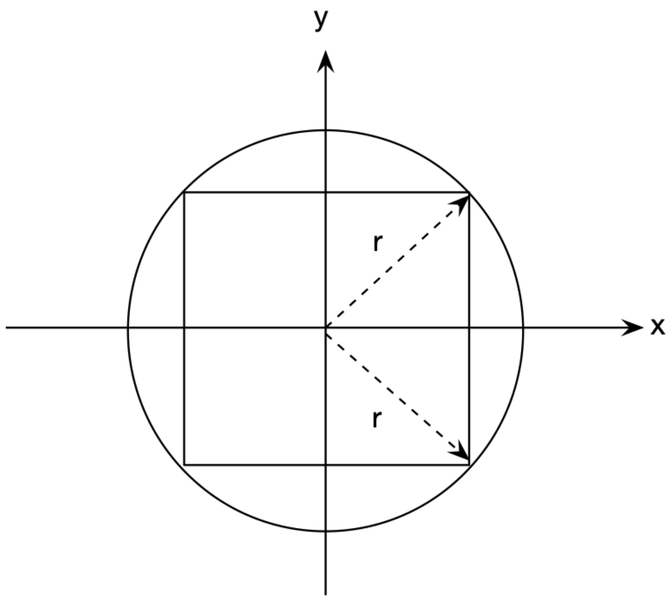

Figure 2: Modified dartboard

Suppose the radius of the circle is r. The inscribed square therefore has each side of length $\sqrt{2}r$, making its area $2r^2$. Let N darts be thrown at the circular board and M of them land inside the square. Following the same principle as the original dartboard method: for large N, we have:

$$\begin{aligned} \frac{M}{N} &\approx \frac{2r^2}{\pi r^2} \\ \implies \pi &\approx 2 * \frac{N}{M} \end{aligned} \tag{2}$$

The more darts you simulate throwing at the board, the better the approximation of PI would be. More specifically, you might consider throwing darts at board to be done by generating random points inside the circle. To compute M, we test how many of those random points "land" in the square.

A **uniformly distributed** random point (x, y) inside the circle with radius r is generated by generating random numbers ~~$a \in (0, r)$~~ $a \in (0, r^2)$ and $\theta \in (0, 360°)$, and then converting them to value for x and y as follows:

$$\cancel{x = a\cos\theta, y = a\sin\theta}$$

$$x = \sqrt{a}\cos\theta, y = \sqrt{a}\sin\theta$$

The point (x, y) lies inside the square if $|x| \leq r/\sqrt{2}$ and $|y| \leq r/\sqrt{2}$.

Your task is to write MPI PI Calculation using the **modified** dartboard algorithm. In parallel implementation, the master processor will take as input N. Each processor P_i will then simulate $\lfloor N/p \rfloor$ or $\lceil N/p \rceil$ darts inside the circular board as described above. It will then calculate m_i, the number of darts that fall inside the inscribed square. The master processor is responsible for computing PI by plugging in $M = \sum_{i=0}^{p-1} m_i$. The above algorithm is to be run R number of times, resulting in R estimates of PI, which are then to be averaged and reported. Note that changing the R and N will directly change the accuracy of the result.

Code Framework

We are not providing you with a code framework for this project. However, there are some rules that you MUST follow:

- Name your code as `prog1.cpp` (only one file of code is required for this project)
- Your code must contain the following functions:
  - A function named `main` in which you need to have:
    - **N**: Number of darts be thrown at dartboard as a **program argument** (an integer number at least 5000000)
    - **R**: Number of times the dartboard algorithm must run as a **program argument** (an integer number at most 100)
    - Broadcasting program argument N to all processors
    - For each round, call function `dboard(N)` to compute m_i. This function takes the *total* number of darts to be thrown as its input and computes the number of darts that land inside the square *for that processor*
    - For each round, Master rank must compute $M = \sum_{i=0}^{p-1} m_i$ and use it to estimate PI (a `double` number) (Set master rank as 0) (**Hint**: use `MPI_Reduce`)
    - Set the seed for random number generator equal to taskID (rank)
    - After **R** rounds of dartboard algorithm, average the **R** estimates of PI and report this average.

– Your code must also include a function named `dboard` with the following prototype:

```
int dboard(int N){
     // Compute number of darts to be thrown locally
     //Throw darts (Generate random coords (x,y) )
     //check whether dart lands in square
     return m;
}
```

Note regarding the parameter R: R indicates the number of times that each processor is required to run the dartboard algorithm. More specifically R refers to the number of times that the `dboard` function is called inside the main function per processor. As a result, master processor will calculate PI R number of times. For example, if R=100, the processor with rank say 1 must call the `dboard` function 100 times (inside the main function) and sends the results to the master processor 100 times. Then the master rank, say 0, must receive the information from rank 1 (from other ranks as well), and use MPI_reduce to do reduction on the received values. These **R** estimates are then averaged and reported. Check the pseudocode below (which has to be done inside the main function):

```
for (i = 0; i < R; i++) {
  m = dboard(N);

  M = sum(m) from all procs /*Obtained at master using MPI_reduce*/

  if (rank == MASTER) {
      /* compute PI*/
     }
}
/* Master computes average PI for all rounds and reports */
```

Input Format

Your code must have three inputs: N, R, and number of processors P, where N is a multiple of P and P is a power of 2.

```
$ mpirun -np P ./prog1 N R
Example:
$ mpirun -np 4 ./prog1 5000000 100
 //p =1, 2, 4, 8, 16, N >= 5000000, R <=100
```

Output Format

An output file named **output.txt** which contains ten lines such that for each number of processor (p =1, 2, 4, 8, 16) it has two lines as follows:

- N= #N, R=#R, P= #P, PI= #PIvalue
- Time = #Time

First line means after #N number of throws and #R number of iterations with #P number of processors, average value of PI is= #PIvalue, and the second line shows the total time to compute PI is= #Time.

Note that you must set N and R same while changing the number of processors.

You are also supposed to submit a **short report (not to exceed 2 pages)** named **report.pdf** showing speedup as you are changing N and P. Your report must contain two plots one should show the runtime while changing the number of processors from 1 upto 16, and the other plot should show the runtime as you are changing the value of N (at least you must have 5 values for N). You must **analyze your results** and write your **interesting observations** from the results.

Note that runtime must be measured as total time to compute dboard (maximum time among all processors) plus time needs to do the reduction on the master processor (time to broadcasting N must not include for the runtime of this project).

You are also required to submit a **team.txt** file containing the names of all team members.

Expected output format (taken care of by the code framework) should be strictly adhered and additional output (eg: debug print statements you've forgotten to remove) will prevent you from getting full points. **Any additional output from your implementation should be avoided by the time of your submission.**

Compile and Run

You may implement, run and test your solutions in your local machine, but you MUST test your code in `Pace-ICE` cluster. Following commands are used to compile and run your code.

```
$ mpicc prog1.cpp -o prog1
$ mpirun -np P ./prog1 N R
```

Where P, N, and R are program arguments that must be set by user.

Teams

You must form teams up to three members. Per group only one person should submit four files in a zip format `output.txt`, `report.pdf`,`team.txt`, and `prog1.cpp`.

Grading [10pt]

Your submission will be graded based on successful compilation and the accuracy of the output.

Submission Guideline

The submission to the Canvas should be in zip format. The source code should be self contained such that in Pace-ICE cluster, once we extract your code we should be able to compile and execute it without any additional parameters. You must name your zip file as `prog1.zip` (which includes four files `output.txt`, `report.pdf`,`team.txt`, and `prog1.cpp`). Your source code must be properly commented.

References and Resources

If you are new to programming in MPI and working with a cluster, `PaceIceCluster.pdf` and `GettingStartedwithMPIlocally.pdf` is a good place to start. We recommend you code your first MPI hello world program and execute it locally and in the `Pace-ICE` cluster before you start working on the assignment.