CX 4220/CSE 6220 High Performance Computing (Spring 2026)
Programming Assignment 1
Due: February 11, 2026

1 Problem Statement

Write a parallel program in C/C++ to estimate the value of π using the Monte Carlo method specified below. The program should take n as input, which is the number of points to be used for the estimation. The program should then generate n random points in the unit square $[0, 1] \times [0, 1]$ and count the number of points that lie inside the first quarter of the unit circle.

Let n be a large integer; then the estimated value of π is:

$$\pi \approx 4 \times \frac{\text{number of points in the circle}}{n}$$

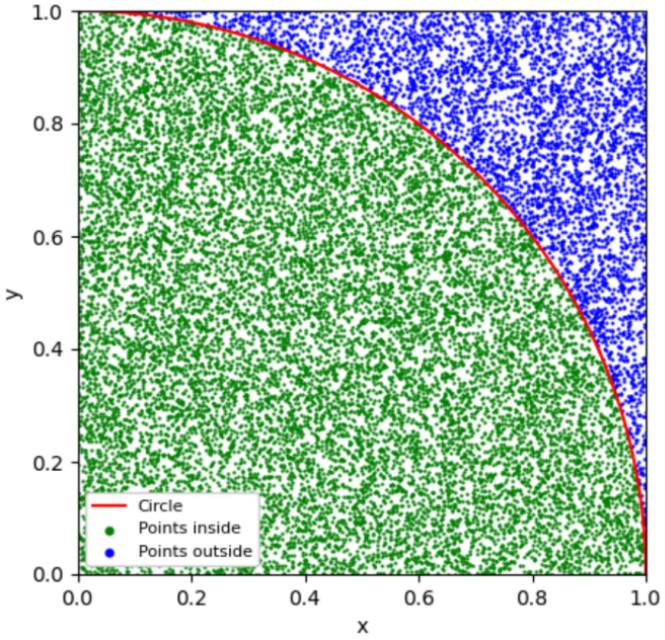

Figure 1: Estimating π using Monte Carlo method

2 Parallel Algorithm

The estimation can be easily parallelized by assigning each processor the responsibility for distinct $\frac{n}{p}$ points, where p is the total number of processors. You can make use of the following MPI functions:

- We pass the value of n (problem size) to the pi calculation function. You need to divide the problem size among the p processors. Make sure to handle the situation where p does not perfectly divide n.

- Use **MPI_Send, MPI_Recv** functions to sum the number of points in the circle across all processors after doing the local computation.

For the random number generator, you can use the `rand()` function in C/C++. However, you should **make sure that the random number generator is seeded differently for each processor**. You can use the processor rank for this purpose. For example, if the rank of the processor is i, then you can use `srand(time(NULL) + i)` to seed the random number generator on that processor.

3 Code Framework

3.1 Input & Output Format

Your program takes n as the input using command line arguments and outputs the estimated value of π and the time taken to compute this value. The output will be in the below given format:

```
Estimated Pi: 3.14152
Time: 0.0777975 seconds
```

Download Files

Download the PA1 files from this link or use `git clone` (also available in Canvas/Files):

- https://github.gatech.edu/cse6220-s26/PA1
- `git clone https://github.gatech.edu/cse6220-s26/PA1.git`

3.2 File Structure

1. **Makefile** - It is good to get yourself familiar with the makefiles. Check the resources section for a quick intro.
2. **pi.h** - The header file containing the function signature to be implemented.
3. **pi_calc.cpp** - Develop your code in this file. This is your **submission file**.
   Note: You should only submit the file pi_calc.cpp
4. **pi.cpp** - The driver file for your code. You can make changes in the driver file for your own testing and to generate data for plotting, but make sure the submitted `pi.h` runs with the original `pi.cpp` file.
5. **autograder.sh** - This file can be used to check if your code passes the test cases. This script will request the resources, run the tests and display the output in the command line.

3.3 Instructions

1. After downloading all the files, go through each of the files and try to understand the structure.
2. Write your implementation in the `pi_calc.cpp` file.
3. Use `make` to compile the program. An executable `pi` will be created if there are no compilation errors.
4. Request resources (interactive node) using `salloc` and use `module load openmpi` to load the MPI library.

5. Use `srun -n <num_processors> ./pi -n ⟨num_points⟩` to run the program, replacing `⟨num_processors⟩` and `⟨num_points⟩` with the value of p and n respectively.

6. After finalizing your code, you can run the test cases from the `autograder.sh` file. Use `sbatch autograder.sh` to run the autograding script. You can run it on the login node as it will request resources and run the autograding script. Results will be in `results.out` and any run-time/compile-time errors in `results.err`.

3.4 Deliverables

1. **pi_calc.cpp**: Make sure this file runs with the original `pi.cpp` file.
   NOTE: Ensure your final submission does not include any debugging print statements. If you use them for debugging, make sure to remove them on your final submission.
2. **Report**: A PDF file containing the following:
   - List names of all teammates at the beginning.
   - For $n = 10^9$, plot a graph of run-time vs. the number of processors for $p = 1, 6, 12, 24$. This run-time should include all computations that contribute to the estimation, including any local and global computations.
   - Do the theoretical run-time analysis for $T(n, 1)$ (serial) and $T(n, p)$ (parallel), including computation and communication costs, and the speedup in terms of n, p. This analysis will be on your entire code in `pi_calc.cpp`.
   - Answers to the questions below:
     (a) Explain the communication pattern you implemented to sum the results.
     (b) Why is it necessary to have a random generator function for generating points? Why is it necessary to have different seeds for the random generator in every processor?
     (c) Explain the functioning of every MPI instruction used in your program (`pi_calc.cpp` and `pi.cpp`), also including setup instructions like `MPI_Init`, etc).

3.5 Grading Scheme

- **Code (15 points)**
  We will be running your submissions on an Intel(R) Xeon(R) Gold 6226 CPU @ 2.70GHz (24 cores) node on PACE ICE Cluster.
  - Passing all code correctness and scalability tests - **10 points**
    - NOTE: You are required to pass the correctness test to be eligible for any points in this section.
    - **Correctness Test (5 points)** - The estimated value of π should lie between 3.13 and 3.15 for $n = 10^6$ and $p = 1, 16$.
    - **Runtime Test (2.5 points)** - Runtime should be less than 6.5 seconds for $n = 10^9$ for $p = 8$.
    - **Scalability Test (2.5 points)** - For $n = 10^9$, $\frac{T(p=1)}{T(p=16)}$ should be greater than 15.
  - Random Generator and different seeds for all processors - **5 points**
- **Report (10 points)**
  - Plot - **2 points**
  - Runtime analysis - **2 points**
  - Questions - **6 points**

4 Resources

- What is a Makefile and how does it work? https://opensource.com/article/18/8/what-how-makefile
- PACE ICE cluster guide https://gatech.service-now.com/home?id=kb_article_view&sysparm_article=KB0042102
