

Set 4 - Monte Carlo, MPI

Issued: November 9, 2022

Hand in (optional): November 23, 2022 08:00

Question 1: Probabilities, Expectation, Variance (20 Points)

Probability Density Function: For a continuous random variable X with range $[a, b]$, the probability density function $p(x')$, is defined such that the probability P of the random variable X to take a value smaller than x is given by

$$P(X \leq x) = \int_{-\infty}^x p(x') dx' \quad (1)$$

Usually this is denoted by $F_X(x) = P(X \leq x)$ and called **cumulative distribution function**.

Expectation Value: For a continuous random variable X with range $[a, b]$ and probability density function $p(x)$, the expected value of $h(X)$ is defined as

$$\mathbb{E}[h(X)] = \int_a^b h(x)p(x)dx \quad (2)$$

A special case is $h(X) = X$, which gives the mean $\mu = \mathbb{E}[X]$.

Variance: For a continuous random variable X the variance of X is:

$$\text{Var}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \quad (3)$$

We realize that this is an expectation value with $h(X) = (X - \mu)^2$, i.e. the variance is the expected mean squared distance from the mean μ .

- a) You arrive in a building and are about to take an elevator to your floor. Once you call the elevator, it will take between 0 and 40 seconds to arrive. The duration until the elevator arrives can be seen as a random variable T . For the following we assume the time of arrival $t \in \mathbb{R}$ to be uniformly distributed between 0 and 40 seconds, i.e. $t \sim \mathcal{U}([0, 40])$.
- Write down the probability density function $p(t)$ for the random variable T .
 - Calculate the probability $P(T \leq t)$ that elevator takes up to 15 seconds to arrive.
 - Calculate the mean value $\mathbb{E}[T]$ that the elevator takes to arrive.
 - Calculate the variance $\text{Var}[T]$ the elevator takes to arrive.

Question 2: Parallel Monte Carlo using MPI (38 points)

Monte Carlo integration is a method to estimate the value of an integral over a domain Ω by taking samples $x_i \sim \mathcal{U}(\Omega)$ from a uniform distribution on the domain Ω . First, note that the integral can be expressed as an expectation value over the uniform distribution

$$\frac{1}{|\Omega|} \int_{\Omega} f(x) dx = \mathbb{E}_{x \sim \mathcal{U}(\Omega)}[f(x)], \quad (4)$$

where $|\Omega|$ is the volume of the domain Ω . The central limit theorem states that we can estimate the expectation value using the average

$$\mathbb{E}_{x \sim \mathcal{U}(\Omega)}[f(x)] \approx \frac{1}{N} \sum_{i=1}^N f(x_i), \quad \text{for } x_i \sim \mathcal{U}(\Omega) \quad (5)$$

Combining eq. (4) and eq. (5) we conclude that we can compute the integral as

$$\int_{\Omega} f(x) dx = \frac{|\Omega|}{N} \sum_{i=1}^N f(x_i), \quad \text{for } x_i \sim \mathcal{U}(\Omega) \quad (6)$$

In the skeleton code you are given a serial implementation of this algorithm estimating the intersection of the two circles shown in Fig. 1. The goal of the exercise is to parallelize the code using MPI.

- a) Implement all of the TODO_a) parts in the skeleton code to calculate the overlapping area of the two circles shown in Fig. 1 using Monte Carlo integration in serial mode!

Use the function `std::uniform_real_distribution<double>` which returns a uniformly distributed random number in the interval $[0, 1]$ and make use of it. Circle radius R_1 , R_2 and circle R_2 x-coordinate x_2 are changeable variables in the system. Make use of them when generating random samples and checking if you are inside the circles area. With $R_1 = 5$, $R_2 = 10$ and $x_2 = 12$ you should get an area of ≈ 17.01 .

You can ignore all the TODO_b parts and try only to recreate the same approximated area of ≈ 17.01 . When compiling, you will get warnings for unused variables, because you have not implemented the MPI part. You can ignore those.

Instructions on how to run your code:

- **module load open_mpi** to load the MPI libraries.
- **make** to compile the code.
- **bsub -n 1 -W 01:00 -ls bash** to have an interactive session.
- **./main** to execute your code on one core.

- b) Implement all of the TODO_b) parts in the skeleton code to parallelize it with MPI. You are given a fixed amount of MC samples $n = 1e9 + 1$. You should split the number of samples as equally as possible without leaving a sample out! One of the ranks will have more or less if the number of samples is not divisible by the number of processors. NOTE: if you are not able to do question a), you can still get full points in this question.

Instructions on how to run your code:

- **module load open_mpi** to load the MPI libraries.

- **make** to compile the code.
 - **sbatch launch.sh** to run the code on 48 cores. This should not take more than 5 minutes if correctly implemented! It will run the `run.sh` script. Which will do multiple MPI runs with different numbers of cores. It outputs them in a separate folder called `out/`. `launch.sh` will create two different files. `slurm_error.txt` which consists of any errors and `slurm_output.txt` which has all of the execution runs as well as the information of which CPU used. It can happen, that you get Warnings in the `slurm_error.txt` "WARNING: No preset parameters were found for the device that Open MPI detected:". You can ignore those. Do not run the script directly from an interactive node. Do it on the login node.
 - You are free to modify the number of cores and which type of CPU to use.
 - **python3 plot.py** to plot the results.png.
 - Note: You could also run the code directly on your computer. In this case, you have to install MPI on your own and adapt the number of cores in `run.sh`. You would also have to run the `run.sh` script directly using **source run.sh**.
- c) Plot the results with the given python script. Don't forget to state which type of cpu you used!
- d) What would you add to your code if you wanted to estimate the error of the Monte Carlo integration? Answer qualitatively, do not write any code.
- e) How does the error of the method change if you use 10 times more samples?

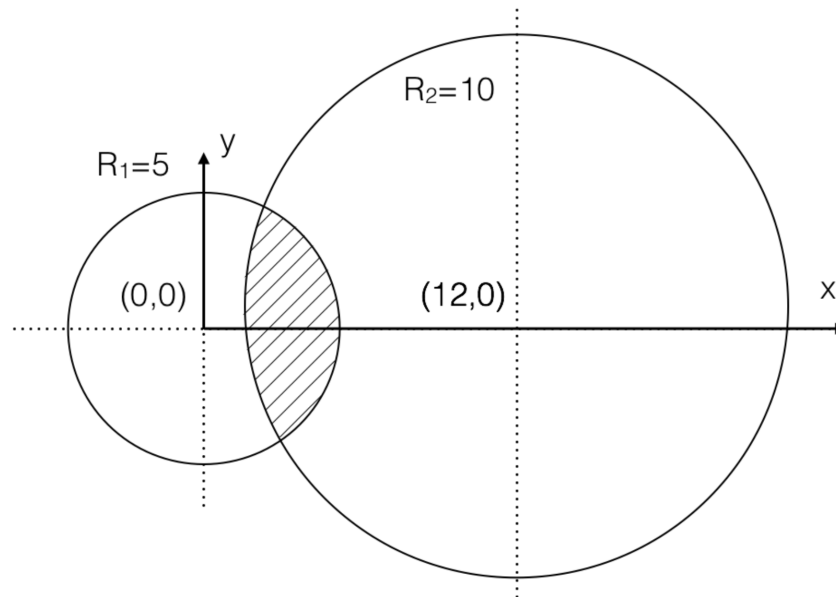


Figure 1: Sketch of two overlapping circles

Question 3: MPI Bug Hunt (32 points)

Find the bug(s) in the following MPI code snippets and find a way to fix the problem!

a)

```
1  const int N = 10000;
2  double* result = new double[N];
3  // do a very computationally expensive calculation
4  // ...
5
6  // write the result to a file
7  std::ofstream file("result.txt");
8
9  for(int i = 0; i <= N; ++i){
10     file << result[i] << std::endl;
11 }
12
13 delete[] result;
```

b)

```
1  // only 2 ranks: 0, 1
2  double important_value;
3
4  // obtain the important value
5  // ...
6
7  // exchange the value
8  if(rank == 0)
9      MPI_Send(&important_value, 1, MPI_DOUBLE, 1, 123,
10              MPI_COMM_WORLD);
11 else
12     MPI_Send(&important_value, 1, MPI_DOUBLE, 0, 123,
13             MPI_COMM_WORLD);
14
15 MPI_Recv(
16     &important_value, 1, MPI_INT, MPI_ANY_SOURCE,
17     MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE
18 );
19
20 // do other work
```

- c) What is the output of the following program when run with 1 rank? What if there are 2 ranks? Will the program complete for any number of ranks?

```
1 MPI_Init(&argc, &argv);
2
3 int rank, size;
4 MPI_Comm_size(MPI_COMM_WORLD, &size);
5 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
6
7 int bval;
8 if (0 == rank)
9 {
10     bval = rank;
11     MPI_Bcast(&bval, 1, MPI_INT, 0, MPI_COMM_WORLD);
12 }
13 else
14 {
15     MPI_Status stat;
16     MPI_Recv(&bval, 1, MPI_INT, 0, rank, MPI_COMM_WORLD,
17             &stat);
18 }
19 cout << "[" << rank << "]" " << bval << endl;
20
21 MPI_Finalize();
22 return 0;
```
