ECE 4122/6122 Lab 6: Using OpenMPI to Estimate the Value of a Definite Integral using the Monte Carlo method

(100 pts)

Category: OpenMPI
Due: Tuesday December 3rd, 2024 by 11:59 PM

Objective:

Use OpenMPI to estimate the value of a definite integral using the Monte Carlo method.

How to use Monte Carlo simulation to estimate an integral:

[1]The Monte Carlo technique takes advantage of a theorem in probability that is whimsically called the Law of the Unconscious Statistician. The theorem is basically the chain rule for integrals. If X is a continuous random variable and $Y = g(X)$ is a random variable that is created by a continuous transformation (g) of X, then the expected value of Y is given by the following convolution:

$$E[Y] = E[g(X)] = \int g(x) f_X(x) dx$$

where f_X is the probability density function for X.

To apply the theorem, choose X to be a uniform random variable on the interval (a, b). The density function of X is therefore $f_X(x) = 1/(b-a)$

if x is in (a, b) and 0 otherwise. Rearranging the equation gives

$$\int_a^b g(x) dx = (b-a) \cdot E[g(X)]$$

Consequently, to estimate the integral of a continuous function g on the interval (a, b), you need to estimate the expected value $E[g(X)]$, where $X \sim U(a, b)$. To do this, generate a uniform random sample in (a, b), evaluate g on each point in the sample, and take the arithmetic mean of those values. In symbols,

$$\int_a^b g(x) dx \approx (b-a) \cdot \frac{1}{n} \sum_{i=1}^{n} g(x_i)$$

where the x_i are independent random uniform variates on (a, b).

[1] Estimate an integral by using Monte Carlo simulation

Method:

Here's a step-by-step example of how to estimate a definite integral using the Monte Carlo method:

1. **Define the Integral**: First, choose the integral you want to estimate. Let's say we want to estimate the integral of $f(x)$ over the interval $[a, b]$. For simplicity, let's take $f(x) = x^2$ over the interval $[0, 1]$, so we want to estimate $\int_0^1 x^2 dx$.
2. **Generate Random Points**: Next, generate a large number of random points within the domain of integration. In our case, since we're integrating over $[0, 1]$, we'll generate random points x_i where each x_i is uniformly distributed over $[0, 1]$.
3. **Compute Function Values**: For each random point x_i, compute the value of the function, $f(x_i)$. In our example, this means computing x_i^2 for each x_i.
4. **Average the Function Values**: Average the computed function values. This average will approximate the average value of the function over the interval $[a, b]$.
5. **Estimate the Integral**: Multiply the average value by the width of the integration interval. In our example, the interval width is $b - a = 1 - 0 = 1$, so the estimated integral is simply the average of the function values.
6. **Increase Precision**: To increase the accuracy of the estimate, increase the number of random points. The more points you use, the closer the estimate will be to the true value of the integral.

Description:

Write a C++ application that uses OpenMPI to estimate the following two definite integrals

1) $\int_0^1 x^2 dx$ which should be equal to 1/3 to test your results
2) $\int_0^1 e^{-x^2} dx$

You application should take two command line arguments (-P and -N):

-P 1 this can be 1 or 2, and indicates which integral listed above to estimate.
-N 1000000 this is the number of random samples to generate in **total** and needs to be distributed across the available processors.

Your program should use **MPI's broadcast communication** methods to distribute random runs among processors and then gather the results for the final calculation of the integral.

Sample Run:

- srun ./a.out -P 1 -N 10000000
- The estimate for integral 1 is 0.33333321
- Bye!

Turn-In Instructions

Zip up your file(s) into **Lab6.zip** and upload this zip file on the assignment section of Canvas.

Grading Rubric:

If a student's program runs correctly and produces the desired output, the student has the potential to get a 100 on his or her homework; however, TA's will look through your code for other elements needed to meet the lab requirements. The table below shows typical deductions that could occur.

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

Element	Percentage Deduction	Details
Does Not Compile	40%	Code does not compile on PACE-ICE!
Does Not Match Output	Up to 90%	The code compiles but does not produce correct outputs.
Clear Self-Documenting Coding Styles	Up to 25%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score – 0.5 * H	H = number of hours (ceiling function) passed deadline

Appendix A: Coding Standards

***Indentation*:**

When using *if/for/while* statements, make sure you indent **4** spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

Camel Case:

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird).

This applies for functions and member functions as well!

The main exception to this is class names, where the first letter should also be capitalized.

***Variable and Function Names*:**

Your variable and function names should be clear about what that variable or function represents. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of “imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

***File Headers*:**

Every file should have the following header at the top

```
/*
Author: your name
Class: ECE4122 or ECE6122 (section)
Last Date Modified: date

Description:
```

What is the purpose of this file?

*/

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.