**CSE-443/543: High Performance Computing**

**Homework #03**

Max Points: 42

**Objective**: The objective of this exercise is to:

* Build experience developing a non-trivial parallel code using OpenMP
* Build experience using thread-safe random numbers in parallel programs
* Learn more about Monte Carlo integration
* Gain experience using the CSE C++ style guidelines
* Build experience with benchmarking parallel programs
* Gain experience using spreadsheets for scientific data analysis

Fill in answers to all of the questions in this document and the corresponding Google spreadsheet. You may discuss the questions with your instructor. The timing runs for this assignment should take less than 30 minutes in total and your single-thread execution time should be less than 100 seconds per run.

## Background

In [Session 02](https://docs.google.com/presentation/d/1JTglnMueX_wzqtEzttfh2wxYHdvVfEo1upJDzDxfFCo/edit?usp=sharing) you saw an example of using [Monte Carlo integration](https://en.wikipedia.org/wiki/Monte_Carlo_integration) to estimate π by calculating the area under a quadrant of a circle. In this homework assignment you will write an OpenMP program to do this for yourself using the approach presented in that session. You may want to review the [Session 02](https://docs.google.com/presentation/d/1JTglnMueX_wzqtEzttfh2wxYHdvVfEo1upJDzDxfFCo/edit?usp=sharing) presentation and/or video for a refresher.

You will be parallelizing this code using OpenMP. The way I am asking you to write the code involves two nested loops, and you will write code that parallelizes the inner loop.

Some comments and hints:

1. Unlike previous labs and homeworks, I am **not** giving you step-by-step directions for the experiment. You’ve done enough of these that you should no longer need them to meet the requirements.
2. Your code must adhere to [CSE Programming Guidelines](https://miamioh.edu/cec/academics/departments/cse/academics/programming-style/index.html) for C++. You can test this by running your program through the [cpplint.py](http://cec.miamioh.edu/files/cpplint.py) style checker which is installed on all BEN002 lab machines. **To get full credit for this assignment your code must pass the following checks:**
   1. No errors when checked with cpplint.py
   2. No warning messages when compiled with these options

–O3 –std=c++17 -Wall -fopenmp

* 1. It must give the correct answer

1. **To get any credit at all, your program must compile without any errors and be able to run.**
2. You should write a program called monte\_omp.cpp. The inner loop of your code will “throw” 1,000,000 “darts” to determine an estimate of π as discussed in [Session 02](https://docs.google.com/presentation/d/1JTglnMueX_wzqtEzttfh2wxYHdvVfEo1upJDzDxfFCo/edit?usp=sharing). You should use appropriate OpenMP directives to correctly parallelize this inner loop.
3. Your outer loop will repeat this computation N times and your code should take N as a command line argument. You should use N = 10000 for the final version you submit for this homework.
   1. At the end of each iteration of the outer loop you should compute the estimate for π and accumulate them, returning their average value at the end.
4. Your program should also have a second command line argument, the random number generator seed to use. This seed should be declared as an unsigned int variable.
   1. This means that you should run it as, for example, ./monte\_omp 10000 1234567
5. If two arguments are not provided, your program should return a non-zero value and print a message to standard error.
6. You should use a different random number seed for every run. You may use the getSeed function from the Homework02-starter.bash shell script for this purpose.
7. As with Homework02, for your code to work correctly you will need to use a thread-safe random number generator. I am giving you starter code (monte\_omp\_stub.cpp) that implements the drand48\_r random number generator. It may be to your advantage to build your code around this file.
   1. If you run monte\_omp\_stub for OMP\_NUM\_THREADS=4 you should get the result

Sum = 19.4017706371

1. You should implement a function called monteCarloIntegration that has this signature

double monteCarloIntegration(int N)

The value it returns should be the estimate for π determined for the run.

1. You will need to write out the value of π, using the setprecision method to set the precision to 12.
2. If you run your monte\_omp program with 4 threads, for N = 40 blocks, and a seed of 1234567 your estimate for π should be 3.141775
3. Since some of you didn’t remember this: I stop answering questions about homework assignments 24 hours before they’re due. You have a week and a half to complete this homework so there should be no need to rush at the end unless you choose to wait.
4. You will have time in class during sessions 19 and 20 to work on the assignment. I encourage you to take advantage of that time and the ability to ask questions.

## Experiment

Develop the program requested and generate timing data for it. You should run the program for 1 through 8 threads and do 5 timing runs for each number of threads.

# Apparatus

The experiment documented in this report was conducted on the following platform (fill in the two lines of the Details column using information determined in your shell script):

|  |  |
| --- | --- |
| Component | Details |
| CPU Model | model name : Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz |
| Main Memory (RAM) size | MemTotal: 8056076 kB |

# Observations

Enter the raw timing data and the estimates for π from your runs into the Google spreadsheet provided.

# Analysis

Compute the requested quantities and enter them into the spreadsheet provided.

# Discussion

How do the average values of π you determined for each value of OMP\_NUM\_THREADS compare with the exact answer? Can you conclude that your computation of π via Monte Carlo integration is correct? Be quantitative, and use appropriate statistical measures to back up your answer.

|  |
| --- |
| From the results of my testing, running the program with 3 threads and with 7 threads had the closest estimate to the actual value of pi, both being around .000001 away. 5 threads had the worst estimate, being around .00008 away. It is worth noting that even though 7 threads was close, it had the highest standard error of all the trials, at .000009, although 3 threads was much lower, at .000002. All of the trials had relatively low standard errors, and all of the average estimates were close to the actual value of pi. All of them were correct to at least 3 decimals points. So, it seems as if my implementation of the Monte Carlo integration is correct. |

# Submit files to Canvas

When you complete the homework, download this document from Google Drive as a Microsoft Word (.docx) file with the naming convention Homework03 - MUid.docx (example: Homework03 - ferrenam.docx).  You should save the corresponding Google spreadsheet file as a Microsoft Excel (.xlsx) file with the naming convention Homework03 – Results MUid.xlsx (example: Homework03 - Results - ferrenam.xlsx)

Then, submit the following files to Canvas:

1. The Microsoft Word file you downloaded from Google Drive.
2. The Microsoft Excel file you downloaded from Google Drive.
3. The monte\_omp.cpp program you created for this homework.