EECE5640
High Performance Computing
Homework 2
*Submit your work on Canvas in a single zip file.

1. (30 points) In this problem you will develop two different implementations of the computation of pi (π) numerically using pthreads and OpenMP. Then you will compare them in terms of scalability and accuracy. Undergraduates/PLUS-One students only need to complete parts b and c of this problem for full credit, though can complete part a (i.e., the pthreads implementation) to receive 20 points of quiz extra credit.

   In this problem, you will develop a program that computes the value of pi. You can refer to the following video that suggests a way to compute this using Monte Carlo simulation:

   https://www.youtube.com/watch?v=M34TO71SKGk&ab_channel=PhysicsGirl

   This is not the most efficient way to compute pi, though will provide you with a baseline. There are better ways to compute the value. Select a more efficient method (e.g., Leibniz's formula) and compare it to the Monte Carlo method in terms of convergence rate, assessing the accuracy of the value of pi as a function of runtime.

   a. Evaluate the speedup that you achieve by using pthreads and multiple cores. You are free to use as many threads as you like. The program should take two input parameters, the number of threads and the number of "darts" thrown. Your program should print out the time required to compute pi and the final value of **pi**. Make sure to document the system you are running on and the number of hardware threads available.

   b. Now develop the same program using OpenMP. Repeat all of the steps requested in part a.

   c. Now compare the two implementations in terms of strong and weak scaling, where the number of Monte Carlo simulations (i.e., "darts" thrown) is used to assess weak scaling. Make sure you plot your results.

   * Written answers to the questions should be included in your homework 2 write-up in pdf format. You should include your C/C++ program and the README file in the zip file submitted.

2. (30 points) In 1965, Edsger W. Dijkstra described the following problem. Five philosophers sit at a round table with bowls of noodles. Forks are placed between each pair of adjacent philosophers. Each philosopher must alternately think or eat. However, a philosopher can only eat noodles when she has both left and right forks. Each fork can be held by only one philosopher, and each fork is picked up sequentially.

A philosopher can use the fork only if it is not being used by another philosopher. Eating takes a random amount of time for each philosopher. After she finishes eating, the philosopher needs to put down both forks, so they become available to others. A philosopher can take the fork on her right or the one on her left as they become available, though cannot start eating before getting both forks. Eating is not limited by the remaining amounts of noodles or stomach space; an infinite supply and an infinite demand are assumed.

Implement a solution for an unbounded odd number of philosophers, where each philosopher is implemented as a thread, and the forks are the synchronizations needed between them. Develop this threaded program in pthreads. The program takes as an input parameter the number of philosophers. The program needs to print out the state of the table (philosophers and forks) – the format is up to you.

Answer the following questions: you are not required to implement a working solution to the 3 questions below.

a.) What happens if only 3 forks are placed in the center of the table, but each philosopher still needs to acquire 2 forks to eat?
b.) What happens to your solution if we give one philosopher higher priority over the other philosophers?
c.) What happens to your solution if the philosophers change which fork is acquired first (i.e., the fork on the left or the right) on each pair of requests?

Provide clear directions on how you tested your pthreads code so that the TA can confirm that your implementation is working. Provide these directions in a README file which instructs how to run through at least 12 iterations of updating the state of the philosophers and forks around the table.

In your writeup, also discuss who was Edgar Dijkstra, and what is so important about this dining problem, as it relates to the real world. Make sure to discuss the algorithm that bears his name, Dijkstra's Algorithm. Cite your sources carefully.

*Written answers to the questions should be included in your homework 2 write-up in pdf format. You should include your C/C++ program and the README file in the zip file submitted.

3. (30 points) Graph coloring involves coloring a graph G (V, E) such that no two adjacent vertices have the same color. This algorithm is used in several important applications, including register allocation in compilers, sparse matrix ordering and VLSI routing. The goal is to use the least number of colors to color the graph. An exact solution to graph coloring is NP-hard. You don't need to obtain the optimal solution (i.e., you can use a greedy approach). Make sure to test your solution with different graphs that you will generate. Describe how you tested your implementation.

a.) There are a few approaches you can take to solve this problem. Develop your solution using OpenMP.
b.) Evaluate the strong scaling and weak scaling performance of your implementation. Compare the runtime taken to the algorithmic complexity (big-O) of the algorithm you have chosen for your implementation.

* Written answers to the questions should be included in your homework 2 write-up in pdf format. You should include your C/C++ program and the README file in the zip file submitted.

4. (10 points) Read the paper provided on Sparse Matrix-Vector (SpMV) reordering optimizations performed on multicore processors. (only parts a and b are required)
   a.) Select one of the reordering schemes described and provide pseudocode (i.e., detailed steps and describe the transformations) **for 2** of the schemes described in the paper.
   b.) **For 1** of the hardware platforms discussed in the paper, provide the details of the associated memory hierarchy on the system.
   c.) Completing this part is optional, but can earn extra credit on your quiz average.

   Develop your own SpMV (sparse matrix-vector multiplication) implementation that uses one of the memory reordering algorithms described in the paper (6 are provided). You will need to generate your own sparse matrices or use those available from MatrixMarket (https://math.nist.gov/MatrixMarket/). Report on the speedup achieved as compared to using the standard dense SpMV kernel approach.

   Students can earn an extra 10 points of extra credit for implementing and evaluating each memory reordering algorithm (60 maximum points).

   * Written answers to the questions should be included in your homework 2 write-up in pdf format. You should include your C/C++ program and the README file in the zip file submitted.