EECE5640
High Performance Computing
Homework 3
*Submit your work on Canvas in a single zip file.

1. (20) In this problem, you will utilize the IEEE 754 format and evaluate the performance implications of using floats versus doubles in a computation.

   a.) Compute f(x) = (1/x) + e^x using a Taylor series for e^x. To refresh your memory, $e^x = 1 + x/1! + x^2/2! + x^3/3! + \ldots$. You select how many terms you want to compute (but at least 10 terms). Compute f(x) for 4 different values, though be careful not to use too large a value. Generate two versions of your code, first defining x and f(x) to be floats (SP), and second, defining them as doubles (DP). Discuss any differences you find in your results for f(x). You should provide an in-depth discussion on the results you get and the reasons for any differences.

   b.) Explore the benefits of compiling on the Discovery cluster with floating point vector extensions (e.g., AVX). Report on the performance benefits of utilizing AVX. Additional information will be provided on AVX support on Discovery.

   c.) Provide both IEEE 754 single and double precision representations for the following numbers: 1.1, 6200, and -0.044.

2. (20) Using Linux semaphores and Posix pthreads, implement a C/C++ program that counts the number of numbers between 1-10,000 that are evenly divisible by 3 or 7, or by both. Run your code using 8 and 16 threads, distributing the work to the 8 and 16 threads to perform both checking. At the end, you will print out the number of numbers that were divisible. Also print out the number values that were found. Make sure you use pthreads for generating your 8 and 16 threads, and semaphores for synchronization. Discuss the performance of your implementations (don't time the printing) as we move from 8 to 16 threads.

3. (20) Revisit the Dining Philosophers Problem, but this time develop your solution in OpenMP. You will then answer the following questions:

   a.) Compare and contrast how difficult/easy it was for you to leverage OpenMP, as compared to pthreads, for your implementation.

   b.) Compare the performance of the two implementations – you will probably need to modify your program to get accurate performance results. Please do not just provide performance numbers, try to explain any differences you see your results.

4. (20) (Extra credit for undergraduates and PlusOne students, required for graduate students) Write an OpenMP program that performs a matrix-vector multiply on an 512x512 matrix of integers, multiplied by a 512x1 vector. Initialize the matrix in your program. Then have your program:
   a.) Print out the number of processors available to run the program on a multicore system of your choice and print out a unique threadID for each thread using the OpenMP built-in function.
   b.) Print out the resulting vector.
   c.) Use the OpenMP built-in reduction to compute each element of the output vector.

5. (15) (Extra credit for everyone)
   Find a published paper from an ACM or IEEE conference that discuss a novel sparse matrix format that was not covered in class. Discuss why the proposed format is superior to the CSR or CSC format. Make sure cite your sources.

* Written answers to the questions should be included in your homework 3 write-up in pdf format. You should include your C/C++ programs and the README file in the zip file submitted.