CSC718 Operating Sys & Parallel Programming

Homework 3 MPI and OpenMP Programming Assignments

Total: 100 points

The homework assignment will be graded based on the following criteria:

- Accuracy: 1) the solution meets specific requirements in the problem description; 2) the solution produces correct results; 2) the procedures adopted in the solution are technically sound.
- Efficiency: efficiency will be one of the criteria when grading programing assignment. The solution should produce the desired results efficiently.
- Effort/neatness: the solution includes excellent effort, and all relate work is shown neatly and organized well.

Homework assignment feedback will be available through the DropBox folder on D2L.

For all the programming assignments, you can choose any operating systems you want. I will usually provide C/C++ samples for the programming assignments. If you prefer to use other languages, e.g., Java, they are accepted too. A README.txt is required to submit any programming assignments. In the README.txt, you need to provide the following information:

1) How to compile your program?
2) How to run your program?
3) What is the output and the results when I run your program?
4) Any descriptions which may help me to compile, run, and verify your answers. (FYI: I check each programming assignment turned in!)

Zip all you source code, project files, supporting files, and README.txt and submit the all-in-one zip file together to the D2L Dropbox.

(20 points) Programming Assignment 1: We have learned parallel sieve of Eratosthenes algorithm. To find all prime numbers up to some limit value N, this algorithm works as follows:

1. Create the sieve -- an array indexed from 2 to N. (This can be an array of bools or ints or anything else that supports random access and provides entries that are easy to "mark".)
2. Set k to 2.
3. Loop:
    1. In the sieve, mark all entries whose indices are multiples of k between k^2 and N, inclusive.
    2. Find the smallest index greater than k that is unmarked, and set k to this value.

    Until $k^2 > N$.

4. The indices of the unmarked sieve entries are prime numbers.

We have discussed how to convert the sequential program into parallel program in our class. We also identified a few ways to improve the algorithm. In this exercise, three versions of sieve program are given, i.e., sieve1.c, sieve2.c, sieve3.c. Go through the code and answer the following questions:

- (5 points) Read the code and briefly explain how each version works.
- (5 points) What are the differences between sieve1.c and seive2.c? How does sieve2.c improve on sieve1.c?
- (5 points) What are the differences between sieve1.c and seive3.c? How does sieve3.c improve on sieve1.c?
- (5 points) Benchmark the performance of the three versions on the Rushmore cluster using different number of processors and fill the table below. Does the Benchmark results meet your expectation? Why?

Problem	Np=1	Np=2	Np=3	Np=4
sieve1.c				
sieve2.c				
sieve3.c				

- (**Bonus 10 points**) If you are able to find another way to optimize the sieve program and perform better than the three visions provided in the exercise, you will have chance to get an extra 10 points for this homework. Proof needs to be submitted to demonstrate the improvement you have is better than the three programs provided. Examples of the proof includes, but are not limited to, fast execution time compared with the provided three sample programs.

(10 points) Assignment 2: Compare the differences of MPI and openMP programming. Which one do you like better and why?

(30 points) Assignment 3: gprof is a type of tool called a profiler. Profiling allows you to learn where your program spent its time and which functions called which other functions while it was executing. This information can show you which pieces of your program are slower than you expected and might be candidates for rewriting to make your program execute faster. gprof can be made available in the VMs in the Rushmore virtual cluster.

- (15 points) For linpack_bench.cpp, read the profile tool gprof user manual (see the html guide), run the profiling tool and report the percentage of running times for each function.
- (15 points) For linepack_bench.cpp, based on the percentage of running times from gprof, briefly describe your strategy to modify the program if openmp is chosen to optimize the program.

(40 points) Programming Assignment 4: For the 4 sequential c programs, p1.c, p2.c, p3.c, p4.c, using openmp to parallelize them as much as possible and do the following profiling.

- Add timestamp functions to the code to count the running time of the sequential programs and your openmp programs and show the result in the following table.
- Copy your openmp program running outputs to a report file running.txt. You need to make sure openmp programs should generate the same results as the sequential codes.

	P1	P2	P3	P4
Sequential code running time				
openmp running time				
Speed up				
No. of threads				

Note: speed up is the ration of the sequential code running time to the openmp running time.