COSC 420 - High-Performance Computing
Lab 1

Dr. Joe Anderson

Due: 10 September

1 Objectives

In this lab you will focus on the following objectives:

1. Develop familiarity with `C` programming language
2. Develop familiarity with parallel computing tool `OpenMPI`
3. Explore empirical tests for program efficiency in terms of parallel computation and resources

2 Tasks

1. You may work in groups of one or two to complete this lab. Be sure to document in comments and your `README` who the group members are.
2. Put your code in a folder called "Lab-1". This folder will be zipped and turned in at the end.
3. Make sure you have the `mpi` software available by using the command: `module load mpi/mpich-3.2-x86_64` (if needed). (Extra: Try `man module` for more on this awesome tool!)
4. Include a `Makefile` to build your code.
5. Write a first "hello world" program to:
   (a) Output the "rank" of the node, and the size of the current environment
   (b) Output the processor name of the node.
6. Write a second program to perform a "primality" test of a given number, N, an unsigned long integer:
   (a) Use brute force, but you need only test up to $\sqrt{N}$. Note you may need to use the compiler option `-lm` to link against the math libraries.
   (b) If the number is found to be composite, report one way to factor it.
   (c) Split the work across the available number of nodes, using the `mpi` interfaces.
   (d) Include some diagnostic messages to display which nodes are testing which ranges of factors, and other relevant information you find helpful to be sure the program is working appropriately.
   (e) For now, we won't worry about aggregating each node's result, but rather just have each node print its conclusion to standard output. o, if at least one node finds a valid decomposition, the number is not prime. If no nodes find a factor, it must be prime!

i. Optional: try using the `MPI_Abort` routine to terminate the whole thing. This is messy and not the best way, but it can prevent you from waiting on all processes to finish after you've already found proof of being composite!

7. Test the program on a variety of input sizes ($N = 200000$, 11, 19845, 349349124211, etc) and a variety of node parameters (1 through 8). You can use the `atoll` and/or `strtoull` functions to convert input strings to long integers (see manpages, these both require compiling as `c99`).
   (a) Run each input several times and use the `time` command to measure the time to complete each
   (b) Record the averages of each, report them in a clean tabular format

8. Include a `README` file to document your code, any interesting design choices you made, and answer the following questions:
   (a) What is the theoretical time complexity of your sorting algorithms (best and worst case), in terms of the input size?
   (b) For this specific task, the structure and flow of your program may drastically influence the runtime. What steps did/can you take to gain performance of the entire process?
   (c) According to the data, does adding more nodes perfectly divide the time taken by the program?
   (d) Justify/prove the fact that one needs only check up to $\sqrt{N}$ in the brute force primality test.
   (e) How could the code be improved in terms of usability, efficiency, and robustness?

3 Submission

All submitted labs must compile with `mpicc` and run on the COSC Linux environment.

Upload your project files to MyClasses in a single `.zip` file.

Turn in (stapled) printouts of your source code, properly commented and formatted with your name, course number, and complete description of the code.

Also turn in printouts reflecting several different runs of your program (you can copy/past from the terminal output window). Be sure to test different situations, show how the program handles erroneous input and different edge cases.