CSE 603 **Programming Assignment 2**
Due Date: Oct 24, 2014
Weight: 9% of your grade

Problem: Simple Parallel Bucket Sorting using OpenMP and MPI: study of scalability.

Description: In this assignment you will optimize parallel programs using OpenMP in a shared-memory environment and using MPI in a distributed-memory environment. In particular, you will have to parallelize a simple bucket sorting algorithm.

Start with a uniprocessor (sequential) program written in C for bucket sorting. Assume a range within a bucket. Since the numbers are not uniformly distributed, your buckets may be imbalanced when you distribute the buckets to different processors. Make sure you balance load on each processor to get good performance.

- Your task is to parallelize the algorithm using the OpenMP API on the shared-memory nodes of CCR and MPI on the CCR cluster
- The program takes two main parameters, which are read in from the command line:
  P: the number of processors and N: the problem size.
- Use the provided programs to generate random numbers with a normal distribution.
- In addition, the program takes arguments that will tell whether to output the original and sorted list, either to a file or standard out. This should assist you in ensuring that the parallel version of the algorithm works correctly.

What you need to do:

- Your report should include the following:
  - Your rationale for parallelizing the program
  - Speedup plots for varying size of N and P.
  - Complete source codes in different directories – sequential, OpenMP, MPI, and OpenMP+MPI including makefiles for each embedded within the directories. Include scripts you may have used to run programs with different parameters and also scripts that you may have used to collate results and create graphs (highly recommended to do this to make it easier to finish your work in time).
  - Script (with complete description) to run your programs (including samples)
  - Discussion of results to include the following (graphs)
    - Impact on the performance of the algorithm based on
      - Increase in the size of the problem
      - Increase in the number of cores
      - Impact of load balance on performance
  - How does this relate to your theoretical evaluation of the scalability of this algorithm?

Extra Credit (2%): Incorporate a combination of OpenMP and MPI and compare the performance in terms of increase/decrease of speedup and scalability.

Specific Submission Guidelines:

1. Please submit your assignment as *username*.zip
2. Root directory should include OpenMP, MPI, and OpenMP_MPI (for extra point only) directories. Folder name should be as indicated above.
3. Each directory must contain a Makefile and the source files (e.g. OpenMP.c, MPI.c, and OpenMP_MPI.c)
4. Name of the executable file (after running make command), should be as same as *.c file (e.g. for OpenMP.c file, the executable -o file should be named as OpenMP)
5. For each program, when <-t problem size> arguments are passed, it should print out the results and execution time. For example, when running ./OpenMP -t 100 it should print out the sorted result for problem size 100 on standard output along with execution time.
6. Submit your report as *username*.pdf.

7. In your report, please include execution time for the following cases.

Program	Number of cores	Problem size
OpenMP	1 node 2 task-per-node	100,000
OpenMP	1 node 2 task-per-node	1,000,000
OpenMP	1 node 8 task-per-node	1,000,000
MPI	1 node 2 task-per-node	100,000
MPI	1 node 2 task-per-node	1,000,000
MPI	2 node 8 task-per-node	1,000,000
OpenMP_MPI	1 node 2 task-per-node	100,000
OpenMP_MPI	1 node 2 task-per-node	1,000,000
OpenMP_MPI	2 node 8 task-per-node	1,000,000

You should also include other experiment cases in your report.

Note: username in this context is your UB username, and the other file names should be followed exactly as stated.