CS 5334/4390 Spring 2015
Shirley Moore, Instructor
Final Project Assignment

The final project will consist of:

1. an implementation and/or analysis of a significant parallel algorithm not included in a homework assignment
2. a report describing a) the background for the algorithm and design decisions for the implementation and/or b) the performance analysis, as appropriate
3. a presentation during the final exam period describing and demonstrating your project

The specific topic can be one of those below or of your choosing, but you must have your topic pre-approved by the instructor. You may work individually on the final project or in teams of up to three people. In the case of group work, you must clearly document the contributions of each team member and carry out the amount and difficulty of work proportional to the size of your team. For example, a team project might implement and compare a set of related algorithms, or compare various parallelization strategies for a given algorithm.

Suggested topics:

1. Task-based parallel implementation of LU decomposition.
Design an efficient task-based algorithm for solving the LU decomposition problem in parallel. Implement your algorithm in both Cilk and OpenMP, tune the performance of your implementations, and compare the performance of the two implementations. (This could easily be a two-person project with each person doing one of the implementations). You can find some OpenMP code and an explanation here: http://crd-legacy.lbl.gov/~xiaoye/G2S3/Hands-On-Exercises/denseLU-OpenMP/TP_lu/

2. Parallel random number generator
Random numbers are used to simulate random processes or to sample large parameter spaces. Well-known sequential techniques exist for generating, in a deterministic manner, number sequences that are large indistinguishable from true random sequences. On a parallel computer, random number generation becomes more complicated because many concurrently executing tasks may require access to random numbers. Design and implement a general-purpose parallel random number generator in MPI with the following properties (and showing that these properties hold):
1) Preservation of pseudo-randomness
2) Each process draws from a distinct random number sequence
3) Repeatability between parallel runs when using the same random seed
4) Scalable – e.g., no centralized bottleneck

Do not use a replicated approach with multiple instances of the same generator, even if using a unique seed for each task. Sequences generated in this manner are not guaranteed to be independent and can suffer from serious correlation problems.
(This could become a two-person project by choosing two different strategies – e.g., random-tree method and leapfrog method – and comparing the results).

3. Parallel molecular dynamics
Molecular dynamics simulations are used to study the structure, dynamics, and thermodynamics of biological molecules. They are also used in the determination of structures from x-ray crystallography and from NMR experiments. The time evolution of a set of interacting atoms is followed by integrating their equations of motion. A sequential implementation of a simple molecular dynamics simulation is provided in the file md.c. Parallelize the code using one or more parallel programming paradigms (e.g., PThreads or Cilk, OpenMP, MPI) and calculate the speedup for up to at least eight processing elements. (This could be a group project if each person in the group did a different parallel implementation).

4. Parallel sorting
Implement using MPI and compare the performance of different parallel sorting algorithms. Possible parallel algorithms include parallel quicksort, sample sort, bitonic sort, and parallel radix sort. Note that one algorithm may perform best on smaller input sizes, while a different algorithm may perform the best on larger inputs. (This could be a group project if each person in the group does a different algorithm).

5. Parallel graph algorithms
Parallelizing graph algorithms is difficult because the classical algorithms are typically sequential in nature, for example processing vertices or edges in some sequential order in a manner that cannot be easily parallelized by just parallelizing the loop. Instead, we have to think "outside the box" and develop creative approaches that enable more parallelism. Implement and test a parallel minimum spanning tree algorithm or another parallel graph algorithm of your choosing.
Resources:

- Intel Graph Algorithm Examples, https://software.intel.com/en-us/courseware-graph-algorithm-examples
- Problem Based Benchmark Suite, http://www.cs.cmu.edu/~pbbs/index.html
- Galois, http://iss.ices.utexas.edu/?p=projects/galois

Teaching option:
An alternative to an implementation project is to learn about a parallel programming topic not covered in class and give the class an overview and demonstration of the topic. Possible topics include:
T1. MPI I/O
T2. One-sided communication operations in MPI 2
T3. Hybrid MPI+OpenMP programming
T4. MapReduce (see http://hadoop.apache.org/ )
T5. MPI for Python (see http://mpi4py.scipy.org/ )
T6. OpenACC for GPU programming