Semester Project Information
CMSC858N, Spring 2023, University of Maryland (Dhulipala)

Summary: One of the main parts of the course is completing a course project related to parallelism and parallel algorithms. Since many of you are graduate students pursuing your own research (and have limited time to spend on course projects), one option is to have your project study some aspect your ongoing research that is related to parallel algoirthms. If you have any questions about the suitability of a project idea, please chat with Kishen or myself after class, or over email.

Project requirements: Your project should pick a well-defined problem and develop parallel algorithms for it that can be run using fork-join parallelism and obtain good speedups on parallel machines. Ideally you would use the Parlay library that we gradually introduce in the homeworks.

Some example ideas are:

1. Implement a parallel algorithm to find the minimum spanning tree of a graph.
2. Implement an efficient version of the low-diameter decomposition algorithm, and apply it to some problems (e.g., Connectivity or Spanners).
3. Implement a parallel algorithm for text compression (e.g., LZ4 encoding/decoding).
4. Implement a parallel algorithm for computing a suffix array of text and run it on some large text data (e.g., genomic sequences).
5. Implement a parallel algorithm for some NP-hard optimization problems using local search or using parallelizable heuristics and study the tradeoffs between quality, work, and parallizability. E.g., you might consider graph coloring, finding large independent sets, etc.
6. Implement a parallel algorithm for solving max flow in a directed graph.

Some questions you should ideally explore include how your algorithm compares to a reasonbly optimized serial baseline (speedup), how much speedup it achieves over simply running your implementation on a single thread (self-speedup), analyzing the performance of your algorithm on some interesting datasets, including real-world datasets if applicable. Lastly, you should try to identify and fix performance bottlenecks in your algorithm, e.g., using perf or by using timers.

Your report should be about 3–5 pages long and should explain the algorithm you considered, any prior work on the topic (in theory and in practice), and details about your implementation. You should report running time results on any parallel machine of your choosing with at least 4–8 cores.

You should submit your code as either a compressed archive (e.g., zip or tarball) or as a link to a Github repository (your choice). Your code should be reasonably well documented, and have a README explaining how to run it, and what the input format is, if applicable.

Timeline: Our goal is not to have you do unnecessary work. That said, it often helps to write up a short progress report in advance of the project due date just to make sure you don't save all of the work for the last week. So in addition to the final project report, we will have two short reports due earlier in the semester. These short progress reports should describe what you have been thinking about regarding the project so far, and can be as short as a half page of text.

1. **Progress Report 1. (Due 4/11)** This report should briefly describe the problem you are looking at and the scope of your planned project. If you're working with multiple team-members, briefly write about who plans to do what.
2. **Progress Report 2. (Due 4/27)** This report should share any updates on what you have done so far, and any change of plans.
3. **Final Report. (Due 5/11)**.

Grading: We will look at your code and judge the quality of the code, any innovative ideas you explored in your implementation, and the depth of your experimental evaluation. For example, if you take some existing code and place a few parallel loops inside of it this would not be very innovative. If you simply run your algorithm on some toy datasets that fit in cache, this would not be very deep.