Title ParlayLib: Fast and Scalable Parallelism for Everyone

Abstract Writing correct and efficient parallel code is challenging, even for parallel programming experts. Parallel programming is permeated with pitfalls, such as task granularity, which can make theoretically efficient code run extremely slow if not handled correctly, and race conditions, which can cause subtle and hard to reproduce bugs. To help make writing correct and efficient code easier, our research group has put together ParlayLib, a C++ library of useful, efficient, and easy-to-use parallel primitives. We have recently released version 2 of the library.

The ultimate goal of ParlayLib is to allow novice and intermediate programmers to write parallel algorithms that come close to or even match the performance of expert-written code. ParlayLib V2 comes with a set of over 50 example applications that demonstrate that this goal is feasible. Each implements an algorithm using a combination of ParlayLib's simple primitives and exhibits strong performance, sometimes matching the best available code for the problem at hand. In this tutorial, we will cover some of the most important features of ParlayLib, and demonstrate several example applications which we hope will illustrate that highly efficient parallel algorithms can have simple and concise implementations.

Preferred duration 1.5 hours

Outline of tutorial content and objectives with enough details for both the scope and the depth
The tutorial will touch on three main points:

1. Setting up and getting started with ParlayLib
   - **Objective** To understand how to start working with ParlayLib
   - **Content** We will demonstrate how to set up a project using ParlayLib
2. Implementing parallel algorithms using ParlayLib
   - **Objective** To understand the range of algorithms and tools available in ParlayLib and demonstrate how they can be effectively applied to implement parallel algorithms
   - **Content** We will demonstrate some example implementations of algorithms using ParlayLib. The goal is to emphasize the relative simplicity of the implementations and demonstrate that expertise in parallel programming is not a necessity to implement fast parallel code.
3. Benchmarking parallel algorithms
   - **Objective** To understand how to benchmark parallel algorithms and measure their performance
   - **Content** We will benchmark the algorithms that we implement to illustrate how to compare the performance of various implementations.

Prerequisite knowledge Familiarity with programming in C++ and basic algorithms

Very brief biography of the tutorial organizers and relevant information

- **Daniel Anderson** PhD student in the Computer Science Department of Carnegie Mellon University
- **Guy Blelloch** Professor in the Computer Science Department of Carnegie Mellon University