Benjamin Flin, Jay Karp
brf2117, jlk2225
11/22/20
Parallel Functional Programming

Parallel Sorting Algorithms

1 Introduction

In this project, we will be exploring and comparing different optimization techniques for three different sorting algorithms. We will attempt to build an implementation of six main algorithms. These include compare-exchange [1] for merge sort, a naive implementation of Quicksort, Hyperquicksort, Quicksort by regular sampling, a common implementation of Bitonic sort and some hybrid sorting algorithm using the best features of the previous five. This project will be an exploration into the pros and cons of sorting in parallel based on different implementations. Our goal for each of these implementations is to analyze their features in order to make them as fast and efficient as possible. We will test each algorithm, and determine which ones perform the best under certain conditions, e.g. when the size of the list is small or when the list is nearly already sorted.

2 Expectations

Sorting algorithms are often heavily optimized, and matching that performance will most likely prove to be a challenging task both in sequential and in parallel Haskell. However, we are expecting to receive a moderate performance benefit from each of the different algorithms over their sequential counterparts. We will have to have a very clear understanding of Haskell's paradigms for parallel programming in order to maximize performance and efficiency.

3 Resources

[1] 11.4 Mergesort
[2] Lecture 12: Parallel quicksort algorithms
[3] Bitonic Sort: Overview