Parallel Functional Programming Final Project Report: WordLadder Solver

Daisy Wang(uni: yw3753), Yiqu Liu(uni: yl4617)

1. Introduction + Background

Word Ladder, also known as Doublets, or word-links, is a well-studied word game problem invented by Lewis Carroll. The target of this problem is to find the shortest transformation sequence from one word to a target word based on a given set of words, in which any two adjacent words differ by one character and each word must be a proper English word.

Our final project will consist of making improvements to the word ladder problem we implemented in homework 4 by allowing starting and ending words to be of different lengths, every adjacent word should still differ by only one letter. In this case, we are allowing adding an extra character or removing a character from the previous word. If the ending word can not be reached within 20 steps, the algorithms will terminate and assume the word ladder could not be built based on the current dictionary. For example, with starting word head, and ending word eat, the shortest transformation could look like below:

head
heat
eat

2. Basic Implementation

We will use a breadth-first search to generate all the valid words within 20 steps.

3. Serial Optimization

- **Bi-directional BFS**
  Bi-directional breadth-first searching runs two BFS simultaneously - one forward from the source word and the other backward from the target word. When both searches meet the same word, the search terminates and the shortest path is found. Bi-directional BFS greatly improves the performance by reducing the exponential growth of the nodes.
- **Pruning**
  In order to avoid duplicate nodes, a hash set is maintained to eliminate the nodes that have been calculated in the upper levels.
- **Using ByteString instead of String**

A ByteString stores one byte per character. It is much faster than using a string in Haskell, which uses 12-byte per character and is non-contiguously allocated. In order to achieve faster speed, ByteString is a better choice.

4. **Parallel Optimization**

The following parts of our solution can be parallelized:

a. Bi-directional BFS
b. Finding next valid hop
c. Doing BFS for every node on a single level

5. **Expectations**

We expect to experiment with parallelizing, exploring different strategies to receive benefits in performance, we will be using thread scope for runtime analyzing, and considering the balance between empirical results and the number of cores.