Concurrent Wait-Free Red Black Trees

Group 16

David Ferguson, Jacob Spigle

March 20, 2018

Abstract

This wait-free implementation of the Red Black tree data structure boasts search(), insert(), update(), and delete() functions, all executed utilizing single-word compare-and-swap instructions. The data structure's concurrent implementation employs the use of "windows", overlapping snapshots of the current state of the Red Black tree within the scope of the windows' root node. Each of these windows is a balanced Red Black tree itself, and pushing a modified window into the windows' origin will result in a correct, linearizable solution.

This solution also strives for optimal concurrency by introducing an array that holds pending instructions (*announce*) and decides whether or not a thread will assist by checking for conflicts with it's own update operation (using *gate*).

An modify operation may also help during a search operation to ensure that the search operation eventually terminates [1]. This is necessary because this implementation avoids copying windows unnecessarily, and instead traversing to the next root when such a transaction would occur.

These additions to the traditional sequential Red Black tree allow for an efficient algorithm that has outperformed other attempts at this implementation of the concurrent wait-free Red Black tree data structure.

1 Introduction

References

[1] Aravind Natarajan, Lee H. Savoie, and Neeraj Mittal. *Concurrent Wait-Free Red Black Trees*. The University of Texas at Dallas, Richardson, TX 75080, USA, 2013.