Cache-Conscious Concurrent HAMT in Rust

Sponsored by Prof. Ding

Chi-Chun Chen, Shaojie Wang

March 22, 2018

Introduction

In this project, we are going to implement a cache-conscious concurrent HAMT in Rust. For concurrency, we will try atomic and locks. For cache-consciousness, we will try clustering and compressing [1].

Hash array mapped trie (HAMT) is an optimization of hash table which reduces memory usage significantly with the cost of a little bit higher time complexity ($O(1) \rightarrow O(log_{32}n)$) [2].

To make HAMT concurrent, we need to maintain **correctness**, **progression** and **scalability**. To maintain safety, we need to ensure that inserting and deleting an element in a (HAMT) have an total order from the view of every thread, also, data races could not happen. To maintain progression, HAMT should allow different threads to insert or delete an element with different prefix of hash value. To maintain scalability, we only use scalable lock or atomic read modify write while inserting or deleting elements in the same place.

While the HAMT decrease memory footprint significantly, it treats all of the memory block as the same and therefore can be improved by adding the concept of cache-conscious. Cache-conscious in HAMT could increase the cache hit rate so we could reasonably expect it to outperform the original HAMT. To achieve cache-conscious, we will apply the clustering and compression principles from [1].

Correctness

Write unit test for the four principle methods in HAMT, namely `search`, `contains`, `insert`, and `delete`, and compare the results with those given by Rust `std::collections::HashMap`. The unit tests contain different magnitude of data.

Evaluation

We will make four pairs of comparisons, from the perspective of memory usage and performance.

1. Concurrent HAMT v.s. serial HAMT.
2. Concurrent HAMT without cache-conscious design v.s. cache-conscious concurrent HAMT.
3. Concurrent HAMT without cache-conscious design v.s. Rust `std::collections::HashMap`.
4. Cache-conscious concurrent HAMT v.s. Rust `HashMap`.

We mainly compare with respect of four methods of HAMT, namely `search`, `contains`, `insert`, and `delete`.

Relevant topics

1. Parallel data structures
2. Scalable locks
3. Parallel memory system

References

[1] Trishul M. Chilimbi, Mark D. Hill, and James R. Larus. *Cache-Conscious Data Structure.* Microsoft.

[2] Phil Bagwell. *Ideal Hash Trees.* Infoscience Department, École Polytechnique Fédérale de Lausanne, 2000.