15-418 Project Proposal

Aidan Smith

November 15, 2023

1 Title

Concurrent Hash Map

2 URL

`https://github.com/aidan-smith/chm/`

3 Summary

I will implement a concurrent hash map with a focus on latency sensitivity. My main goal is build lock-free implementation using CAS operations and hazard pointers for memory reclamation. If time permits, I would then like to experiment by relaxing my lock-free stipulation and testing different granularities of locking and compare their performance with the lock-free implementation. With additional time, I intend to further work on general and workload-specific latency optimizations.

4 Background

Hash maps are a very basic data structure in computer science that see extremely wide-spread use. They provide constant time insertion and lookup for generic objects, which is something almost no other data structure can provide. These features makes them particularly useful for tasks like database indexing, caching mechanisms, and symbol table implementations. Essentially every standard library contains at least a non-thread safe version, and as programs become increasingly more parallel, programmers are

more often reaching for concurrent versions of their favorite algorithms and data structures, with hash maps being no exception.

There are several popular algorithms for implementing hash maps such as open addressing or separate-chaining, each with many different flavors. Many of the popular algorithms have their own trade-offs and implementation challenges, while yielding their own unique benefits. In this project I intend to research these different options and then make an informed decision on what underlying algorithm will be the most beneficial for my goals.

5 The Challenge

Implementing lock-free concurrent hash maps, and lock-free programming in general, is tough due to the complexity of maintaining thread safety without having access to the traditional sledgehammer of locks. There are locality considerations where we would like our nodes to be in as contiguous of memory as possible to reduce latency. Moreover, using atomic primitives reduces performance, so managing that while ensuring correctness is another hurdle.

The main challenge occurs when multiple writers get involved and we have to manage updates occurring simultaneously, while trying to manage contention. More than just adding and deleting sections, depending on the implementation, we may have to manage resizes/reallocations of the table. I do believe different workloads (read-heavy vs write-heavy) will also have significant effect on the performance of a given implementation, so what I am trying to target will have to be something I have to take in consideration when making design decisions.

Memory reclamation strategies are also crucial to manage obsolete objects during ongoing operations. Preventing issues like ABA and ensuring correctness add to the complexity. Such a project will involve very precise low-level programming to prevent race-conditions and is an opportunity to employ advanced techniques like hazard pointers. Understanding the memory-model well will be very important for this project, and is something that I hope I get to learn very well throughout.

6 Resources

I intend to start my implementation from scratch and base it off of Java's ConcurrentHashMap[1]. In the research phase of my project, I hope to read some more papers regarding concurrent hash maps and may expand my resources if I think an idea can be incorporated well. For memory reclamation I intend to implement hazard pointers following the Michael 2004 paper[2]. For benchmarking, I would like to test my code using both pthreads and OpenMP, on the GHC and latedays clusters.

7 Goals and Deliverables

7.1 Plan to Achieve

- Implement a correct lock-free concurrent hash map using CAS primitives
- Implement hazard pointers for safe memory reclamation in lock-free map
- Compare and benchmark my implementation with existing library implementation e.g., Boost, Folly, oneTBB

7.2 Hope to Achieve

- Implement a fine and course-grained locking version and compare with the lock-free implementation
- Optimize performance to be somewhat competitive with library implementations
- Explore mixing lock-free and locking techniques for performance benefits

8 Platform Choice

I am intending on doing this project in C++, to be uniform with the comparison libraries and that is the most likely route. However, I have been briefly considering using Rust.

[1] `https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html`

[2] `https://ieeexplore.ieee.org/document/1291819`

9 Schedule

- Nov 20 - Nov 26: Research lock-free hash map techniques and implementations, begin scaffolding
- Nov 27 - Dec 3: Implement lock-free hash map with CAS
- Dec 4 - Dec 10: Experiment with other locking schemes and begin benchmarking against libraries
- Dec 11 - Dec 14: Finalize benchmarking, compile report, and construct poster board