Siddharth Kannan, Jayanth Bharadwaj
Andrew ID: siddhark, jbharadw

Parallel Computer Architecture and Programming (15-418/618)

Project Proposal

TITLE:-
Comparative study of Concurrent Hash Table implementations

URL:- https://jayanth86.github.io/418FinalProject/

SUMMARY:-
Our goal is to design a performant concurrent hash table using three different approaches, namely:- fine-grained locking, lock-free programming and transactional memory. Through this project we seek to understand the benefits and drawbacks of these approaches under various types of workloads.

BACKGROUND:-
Hash tables are data structures that enable constant-time access to their contents by mapping a set of keys in the universe to a fixed number of hash values. It is possible that many keys map to the same hash value and this results in what is called collision. The most common way to accommodate this is to maintain a list of keys for each hash value. Mapping functions are typically designed to minimize the number of collisions as, in the worst case, we can have can many elements mapping to the same hash value resulting in longer access times. There are three basic operations that can be performed on a hash table: add, remove, and get. While add and remove alter the hash table, get only reads the contents of the hash table. We have now described the basic structure and operations of the hash table that we are going to implement.

THE CHALLENGE:-
Depending on how many threads or processes are accessing this hash table, the nature of operations they are going to perform on it and the access patterns, we will have to take measures in our implementation that ensure data integrity without compromising on performance. Fine-grained locks and lockless data structures can cause subtle data races. There is also the potential to introduce deadlocks or livelocks if not careful. Even after these are overcome, there remains the possibility that the overhead incurred in overcoming these can offset the performance gains that we hope for by using lock-free data structures or fine-grained locks. Our code must also be efficient for varying workloads that may have different proportions of the above-mentioned operations. Additionally, our project considers the use of transactional memory for ease of synchronization without losing out much on performance. Since there are

many variants of transactional memory implementations, we will look to examine what kind of access patterns have an adversarial effect on the variant we use in our project.

RESOURCES:-
We will be writing our code from scratch but using reference implementations found here[1][2] as guidance. In order to compare the performance of our code on different workloads we will use the results from the paper 'High Performance Dynamic Lock-Free Hash Tables and List-Based Sets' by Michael M. Maged as our benchmark. We will also need access to transactional memory capable machines.

GOALS AND DELIVERABLES:-
We believe that a comprehensive study of fine-grained locks, lock-free implementation and transactional memory based implementation will make for a complete project requiring 6 weeks of coding and testing. If we do get ahead schedule, we will consider incorporating the 'expand' functionality in our hash table which we are presently not planning to add due to complications in getting it correct within the given time-constraints.

In our final presentation, we hope to share visual representations of the comparative performance of the various implementations we have planned on different workloads and some insights we gained in the process. We will particularly look to assess the impact on performance when the ratio of read/write operations on the hash table is varied and also when the level of concurrency is stepped up.

PLATFORM:-
We will be using C++ to code this project as we wish to have control over the code at a lower level and due to better familiarity with concurrent programming in C++. We plan to use the RMC machines to implement this project as recommended in the lecture.

[1] https://github.com/huxia1124/ParallelContainers

[2] https://github.com/ssteinberg

SCHEDULE:-

Week	Goal
1 Nov - 7 Nov	Sequential, Literature Review
8 Nov - 14 Nov	Fine-grained locking, Lock-less
15 Nov - 21 Nov	Lock-less, Transactional Memory (For checkpoint on 19, 125% goal is to complete transactional memory, 100% goal is to finish lock-less)
22 Nov - 28 Nov	Transactional memory, Testing
29 Nov - 5 Dec	Testing, incorporate expand capability
6 Dec - 12 Dec	Report and Analysis charts
13 Dec - 15 Dec	Finalize report