Title
Lock-Free Hashing

Members
Danny Balter

Summary
The goal of the project is to implement 3 hash tables, one that is lock free, one that using fine-grained locking, and one that uses coarse-grained locking, in order to compare their performance across different applications and learn about the strengths and weaknesses of each.

Background
Besides implementing the hash table structures, and functions that go with them, I will implement three distinct applications, with which to test my hash table implementations.

The first application is Union-Find, which is almost always implemented using hash tables. By running union find on some of the large graphs that were generated by project 2, I should get a good idea of how this application is accessing my hash tables, and how they respond to the parallel requests on a massive graph.

The second application is a cache. By getting access to a website trace (hopefully the 418 website trace), I will be able to build out a cache using my hashtable implementation and study the performance based on a constant set of hits and misses. This application particularly lends itself to parallelism, as there are obviously many website accesses at the same time, and they are not done sequentially.

The final application is undetermined (will be visiting office hours for assistance choosing one).

The Challenge
Implementing lock free parallel data structures is inherently difficult due to the likelihood of race conditions.

Resources
At this point I have no plans to start with any code base, with the exception of the project 2 code for generating graphs, and hopefully the 418 website trace. I intend to write the project in C++.

Goals and Deliverables
The project goal is to implement the three hash tables described above, and test them to compare relative performance on the three applications described above. By doing this, I hope to gain an understanding of the differences between lock free structures and the two common varieties of locks used. Additionally, I hope to learn in which situations, one of the hash table implementations would be more useful than another.

For the demo, the primary result shown will be performance graphs, analyzing the factors described above.

Platform Choice
C++, this makes the most sense as it is a language I'm comfortable in, and the language in which we have been working all course. Additionally, some of the code I hope to use is written in it.

Schedule
4/10/17 -- Proposal
4/11/17 - 4/18/17 Implement 3 hash tables as described above, beginning with the locking ones.
4/19/17 - 4/26/17 Implement the 3 applications described above.
4/27/17 - 5/4/17 Implement testing suite and begin testing my implementations.
5/4/17 - 5/11/17 Make presentation, write up results.

Checkpoint
As of 4/25/17, the date of the project checkpoint, the focus has changed a little bit. After receiving project feedback, I have decided to pivot away from the complicated applications to test my tables, instead opting for a more simple approach where I populate them based on certain conditions (low contention, high contention, size cap ,etc.). By doing this, I will have more time to work on my implementations, in order to optimize them and add more. In addition, it was recommended that I change my project to include multiple implementations of lock - free - hash tables, so I will be implementing one of my own design, as well as my own, optimized version, of both tables shown in the papers recommended by the prof (Harris 2001, Fomitchev 2004).

As of today, I have implemented my own tables that use 2 level of fine grained, and a coarse grained lock. Additionally, I have implemented my own lock free hash table. I have begun testing for performance in order to optimize. From here out, I need to implement the other two tables, discussed above, and perform testing / optimization on all of them. Once they are all optimized as much as possible, I will run them through a few sets of tests in order to gauge their performance. An important note is that they are all built on top of the same basic hash table structure, the only major difference is that they use different implementations to maintain thread safety. By doing this I believe my comparisons in runtime will be incredibly accurate, and thus my results will be significant.