TITLE: Lock-free Extensible Hash Tables by Ashley Xing and William Wang

URL: https://github.com/Wang-oss-tech/15418-lock-free-hashing

SUMMARY:

This project aims to develop a lock-free extensible hash table optimized for concurrent programming environments, utilizing the principles of lock-free data structures, particularly focusing on implementing a recursive split-ordering technique for resizing operations. By leveraging basic compare-and-swap (CAS) atomic operations, the project intends to create a scalable, concurrent hash table that can handle increasing loads of parallel access without the complexity and overhead associated with lock-based schemes. The system will primarily be tested and deployed in multi-threaded programming scenarios, where efficient and concurrent access to shared data structures is crucial.

BACKGROUND:

Extensible hash tables have several crucial real world applications, including in database management systems (DBMS). Thus, the focus of this project is the implementation of an extensible hash table designed for concurrent environments. The hash table's key feature is its lock-free operation that facilitates the rapid access and modification of data by multiple threads without the performance bottlenecks associated with lock-based synchronization mechanisms.

In a DBMS, hash tables are often used to index data, allowing for fast retrieval of records based on specific keys. As the volume of data grows, the hash table must dynamically resize to maintain efficient access times. This resizing is particularly challenging in a concurrent environment where multiple threads may be reading from and writing to the table simultaneously.

The proposed hash table leverages a lock-free design, which inherently supports parallel access. Key to this design is the use of recursive split-ordering for item organization and a resizing mechanism that does not physically relocate items but rather adjusts pointers to bucket lists.

<ins>Implementation Sketch:</ins>

Initialization: Create a shared, lock-free linked list to hold all the items. Initialize with a small number of buckets, each pointing to segments of the shared list.

Insert/Delete: Use (compare-and-swap) CAS to add or remove items from the list. Determine the correct bucket using the hash of the key, and modify the list directly at the bucket's pointer location. We must order keys by using their binary reversal, which is key to achieving recursive split ordering.

Resize: When a resize threshold is met, double the number of buckets. New buckets point to appropriate segments of the shared list without data movement, using recursive split-ordering of keys for distribution.

Benefits from parallelism:

- Multiple threads can insert, delete, or find items in parallel, leveraging CPU cores efficiently.
- The lock-free design scales with the number of threads, offering consistent performance under high concurrent load.
- Resizing is done in a way that minimizes disruption to ongoing operations, allowing the system to adapt to growing data sizes without significant performance penalties.

THE CHALLENGE:

Challenges

- Managing dependencies that arise during insertions, deletions, and especially resizing. Ensuring that operations on the hash table are completed in a logically consistent order without locks requires sophisticated control over the sequence of operations.
- The use of atomic operations like compare-and-swap (CAS) is fundamental to lock-free designs. However, these operations can suffer from performance issues like excessive retries in highly contentious environments. Designing around these limitations without resorting to locks is a significant challenge.
- As the number of threads increases, the overhead of coordinating these threads—ensuring they do not interfere with each other's operations—can grow. Finding the sweet spot where the overhead does not outweigh the benefits of additional parallelism is important.

Learning Objectives:

- Developing a deeper understanding of how to design algorithms that are both lock-free and efficient, particularly in the face of complex operations like resizing a hash table.
- Learning how to effectively balance the trade-offs between maximizing parallelism and minimizing synchronization overhead and contention among threads.

RESOURCES: Describe the resources (type of computers, starter code, etc.) you will use. What code base will you start from? Are you starting from scratch or using an existing piece of code? Is there a book or paper that you are using as a reference (if so, provide a citation)? Are there any other resources you need, but haven't figured out how to obtain yet? Could you benefit from access to any special machines?

We will be basing our code from the research paper linked here regarding lock-free extensible hash tables. (https://ldhulipala.github.io/readings/split_ordered_lists.pdf).

GOALS AND DELIVERABLE:

Plan to achieve:

1) Develop a fully functional lock-free hash table using C/C++ and OpenMP, incorporating atomic operations for all modifications to ensure thread safety without locks. This includes basic operations: insert, delete, and find.
2) Ensure that the hash table supports concurrent operations, allowing multiple threads to perform read and write operations simultaneously without significant performance degradation.
3) Implement an efficient resizing mechanism that allows the hash table to expand and contract dynamically based on load, without hindering concurrent access.
4) Establish a set of benchmarks to measure the performance of the hash table, including throughput (operations per second) and scalability as the number of concurrent threads increases.
5) Documentation of our performances compared with the lock-based hash table implementation

Hope to Achieve:

1) Achieve a significant speedup (e.g., 2x-3x) over a baseline lock-based hash table implementation under high concurrency scenarios, justified by the efficient use of atomic operations and the reduction of contention points.
2) Explore and implement advanced dynamic resizing strategies that minimize disruption to ongoing operations and improve the efficiency of memory usage.

Demo Flow

Create an interactive demo that showcases the hash table's performance in real-time, highlighting its efficiency and scalability in handling high-volume, concurrent operations. Hope to show our stats with our visualized results.

PLATFORM CHOICE:

We believe that C++ and OpenMP, allow us to assign tasks that can be divided into parallel independent segments of work. This will allow us to tackle low-level operations associated with the hash table.

SCHEDULE:

Week 1: Proposal, Brainstorming Idea
Week 2: Start implementation with coarse-grained and fine-grained implementation
Week 3: Finish implementation, Lock free implementation
Week 4: Finish lock-free implementation
Week 5: Compare characteristics and Prepare final report
Week 6: Present