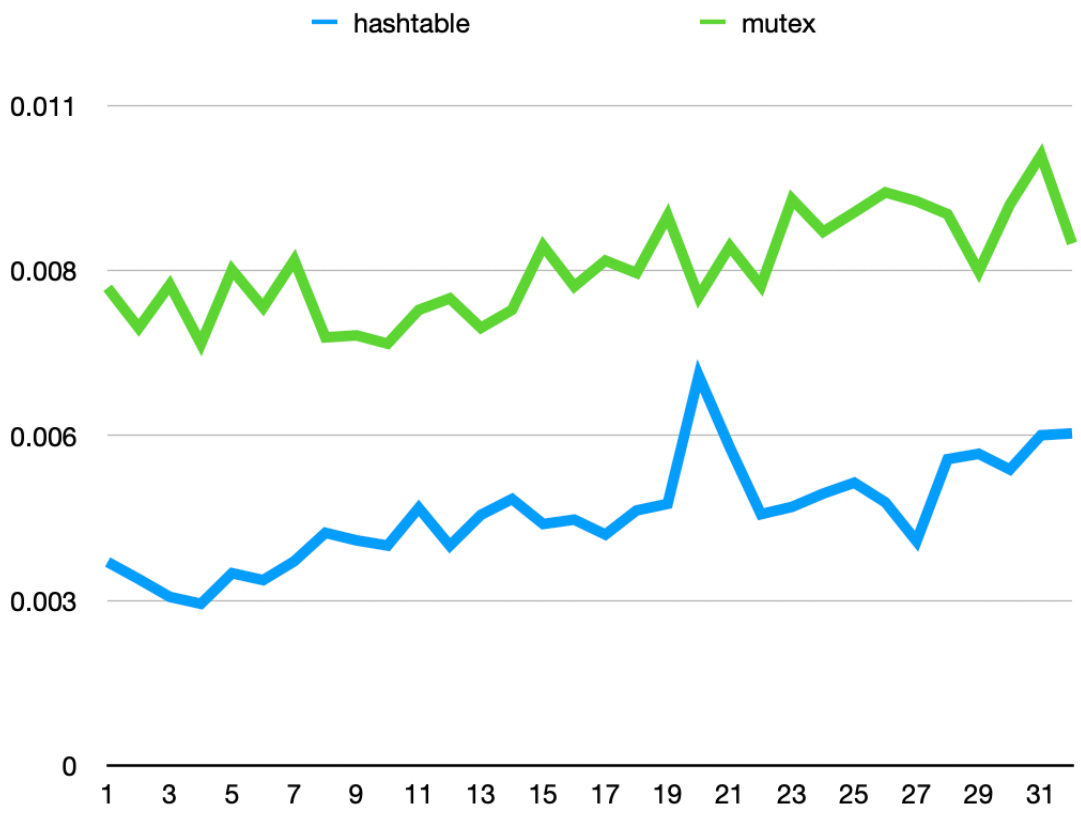
Part 2

• Analysis of prior code's unintended behavior

An entry is considered to be "lost" when it couldn't find a key in the hash table (retrieve(key[key]) == NULL). insert() is causing this unintended behavior when run with multiple threads. Since it doesn't use a lock, more than one thread could be performing malloc and write at the same time.

• A graph of prior running time vs. updated running time



• An estimate of the timing overhead

We are using 2 cores to produce all the results in the graphs. The execution time for mutex exceed the execution time for hashtable. We estimate that the timing overhead is roughly 2 times. Using a different number of cores or other settings may result in a different overhead result.

• An explanation of how you came up with that estimate

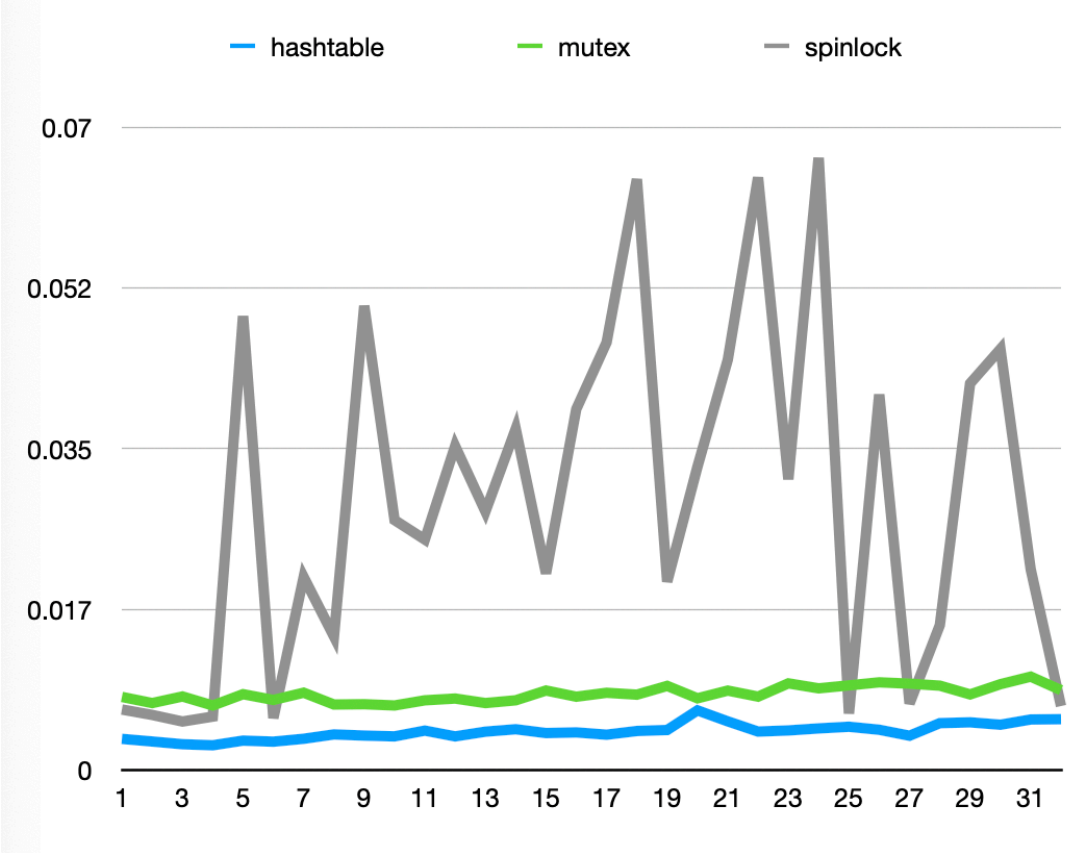
By running the hash table and mutex program with different many of threads, we were able to produce the graph of hash table vs. mutex. By comparing the execution times, we estimated that the timing overhead is roughly 2 times. The time difference in using mutex could be from threads waiting to get the mutex lock. As the number of threads increases, the execution time generally increases as well.

Part 3

• Short answer for what you expect to happen when replacing mutex with spinlock
The runtime for spin locks should be shorter than mutexes.

• Explanation for why you have that hypothesis
Spin locks constantly check if the lock became available. It avoids the overhead of putting thread to sleep and waking it up, which mutex does. So the running time for spin lock should be faster.

• A graph of prior running time vs. mutex running time vs. spin lock running time



• An estimate of the timing overhead of spinlocks
The timing overhead of spin locks seem to vary greatly depending on the number of threads. Spinlock generally has higher timing overhead than mutex.

• An explanation of how you came up with that estimate
By comparing the execution time of hashtable, mutex and spinlock with different number of threads, we noticed that compared to hashtable and mutex, spin lock's execution time varies greater. The reason could be from spin locks active waiting mechanism, where the threads are actively spinning while waiting for the lock to be available. We did not see a consistent increase in timing overhead with the increase of the number of threads.

Part 4

• Short answer for why we do or don't need a lock for retrieval

No we do not need a lock when retrieving an element from the hash table. We are only reading the elements when retrieving, instead of modifying the elements. Multiple threads should be allowed to retrieve the elements at the same time.

• An explanation of what you've changed to allow retrieval to be parallelizable

We did not change anything because multiple retrieve operations are already allowed to be performed at the same time.

Part 5

• Short answer for when insertions could be safely parallelized

Multiple insertions can happen safely when they are performing on different buckets.

• An explanation of what you've changed to allow insertion to be parallelizable

We changed to have as many mutexes as the number of buckets: pthread_mutex_t lock[NUM_BUCKETS];

We initialized the mutexes. When performing insertions, index i indicates the bucket it belongs. The lock for that bucket is then locked to perform insertion and unlocked afterwards. In this way, insertions can be performed to different buckets in parallel.