HW4 - Lock Performance

George Dill

October 24, 2018

1 Summary

This lab report explores the performance of lock types and implementation on a multi-threaded linked list. The implementation types considered were:

- A no-lock read only baseline
- A single Mutex lock over each of the linked list operations.
- Comparison of performance of a spin lock and mutex lock on a single core with multiple threads.
- Performance of a Readers-Writer lock
- A single spin lock per linked list node with a hand-over-hand locking strategy.

The single mutex lock is empirically the best option of the lock implementations tested for large lists. For small lists the single threaded versions performed better than all the lock strategies due to the overhead of the atomic operations required to obtain and release a lock. The readers writer lock provided a performance boost over the mutex lock in small lists because it allows concurrency of reads. The spin lock proved to be a poor solution when operating multiple threads on a single core because it puts lock holding threads to sleep to allow other threads to spin and wait for the sleeping thread. The hand over hand implementation has high overhead in atomic operations to change locks as well as using more memory.

Conclusion: To improve performance over a single entry lock to the linked list data structure consider devising an implementation that does not rely on locks.

For each performance metric the list was grown exponentially at a rate of 10 nodes for values between 1 and 1,000,001 nodes. The benchmark program was run for 3 seconds and the total number of operations recorded. The number of operations is divided by 3 to give an average count of operations per second.

2 Baseline

As a performance benchmark the linked list was run in read only mode with 1, 2, and 4 threads. The results of the baseline shown in 2 show that the number of operations scales roughly by the number of threads for lists of lengths 1:1e6.

3 Big Mutex Lock

The program tested in 3 is implemented by placing a mutex lock on the head of the linked list and locking the entire list for the duration of the operation.

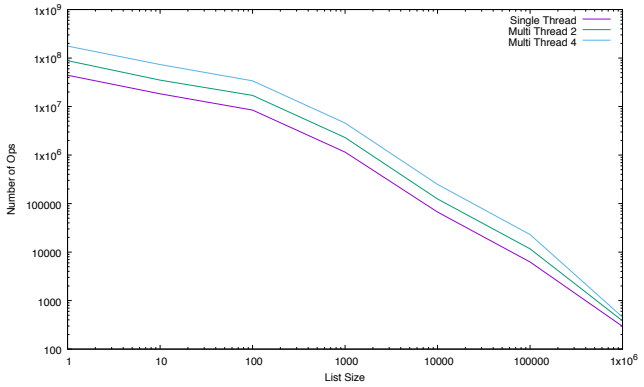

Figure 1: baseline reads/sec by list size logscale

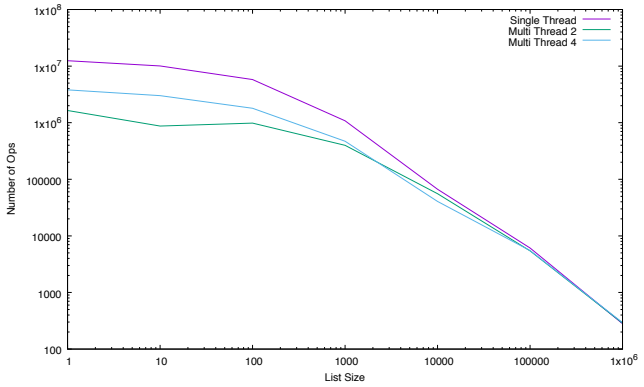

Figure 2: Mutex Lock ops/sec by list size logscale

The performance of the single threaded instance of the Mutex lock underperforms the baseline by a factor of 2. The atomic operations to obtain and release the lock overcome the time to access for small lists. As the list approaches size of 1e6 the time converges with the baseline.

The performance of the multi threaded instances over the baseline performs worse. For the 2 thread instance a 50x reduction for small lists and the 4 thread instance a 66x reduction. The mutex lock works by putting the threads to sleep while they wait for the lock to become available. The overhead of putting the threads to sleep overcomes the benefits of concurrency for small lists. As the list size grows we observe the threads converging to the same number of ops / sec. The single lock forces the program to run more or less sequentially. As the time to perform an operation increases with list size the difference in overhead time becomes proportionately smaller.

4 Mutex v Spinlock on Single Core

4 shows the single threaded spinlock outperforming the mutex lock for small lists with all the concurrent mutex lock performances converging with the single threaded spinlock instance for large lists. This observation corresponds with that of concurrency in section 3.

Its also observed that the 4-thread spinlock performs about half as many operations as the single threaded version and the 2-thread spinlock performs about 3/4 the operations of the single threaded version. This

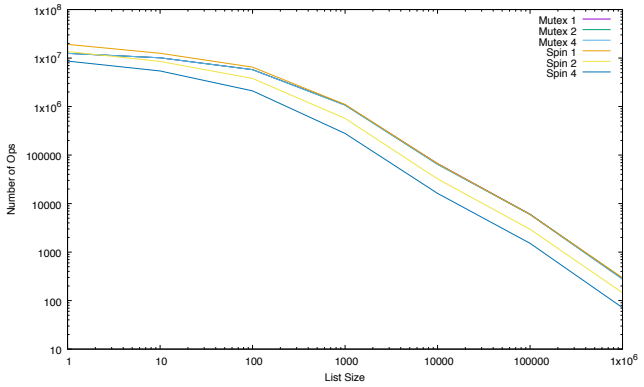

Figure 3: Mutex and Spin Lock ops/sec on single core logscale

is a result of context switching. When one thread holds the lock it is taken out of scope by the scheduler and the alternate thread will spin on the processor waiting for a lock that will not be released by a sleeping thread.

5 Readers Writer Lock

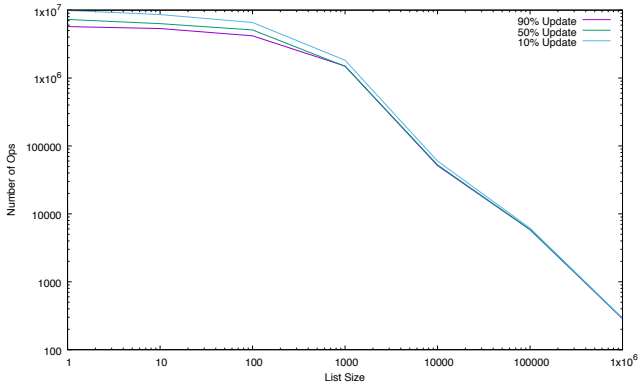

Figure 4: Reader-Writer Lock ops/sec

The readers-write lock defines a lock hierarchy where multiple reads can share a lock but writes cannot share with reads. 5 shows that for small lists fewer writes yields more operations / sec. This is an expected observation because reads can access a lock concurrently but writes cannot. In reference to 3 the readers writer lock performs as well as the single thread lock for small mutex lists and converges at the same time / op for large lists. This suggests the reader writer lock

6 Hand over Hand Nodewise Lock

Observations from 6 show that the Hand over Hand lock does not perform any better on large lists than the big Mutex lock in 3. For small lists the results are erratic, but this could be expected as the threads are more likely to conflict on smaller lists. As the lists become larger the 3 and 4 thread instances outperform the 1 and 2 thread instances by a factor of 2. Empirically the big lock seems to perform better than the tiny

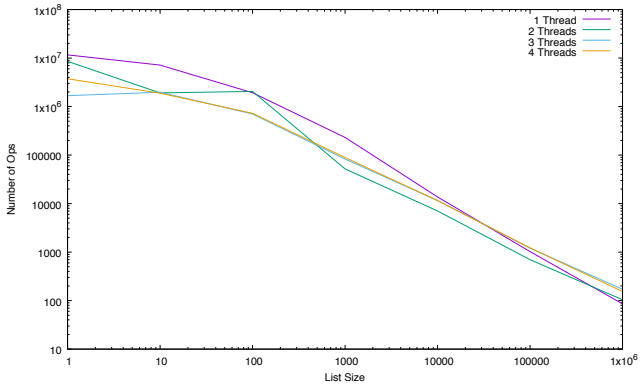

Figure 5: Spinlock on individual linked list nodes ops/sec logscale

locks. This is likely due to the overhead of performing the atomic lock and unlock operations for each step in the linked list.