Hayley Dodkins u21528790

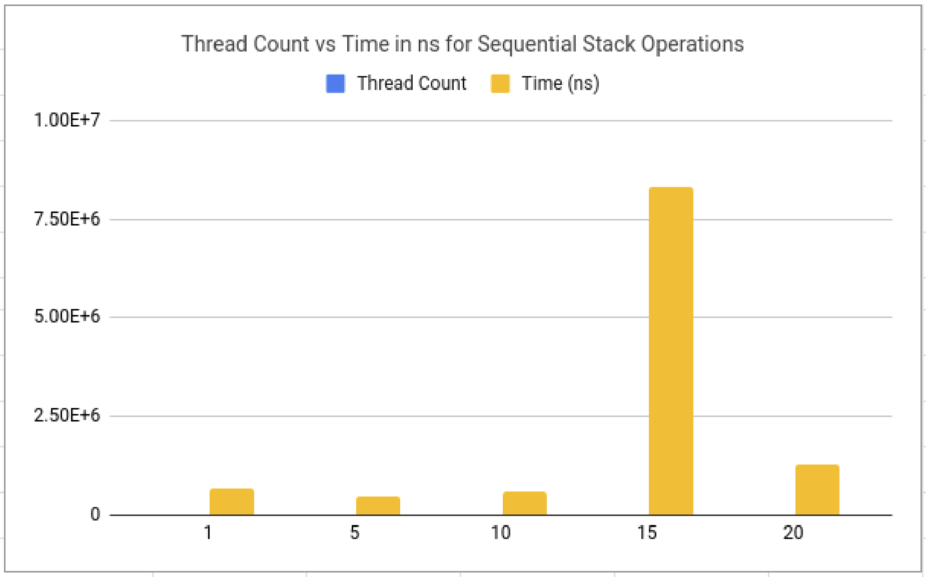
Introduction

The sequential stack is the simplest form of stack implementation, employing a linear approach to perform, push and pop operations. While this works well in single threaded environments it becomes a bottleneck in a multi-threaded environment as it is unable to be used by multiple threads.
Lock-free stack implementation provides a stack that can be used with multiple-threads. While it works well with lower thread counts the stack is not scalable. The elimination backoff stack builds on the lock-free stack with an elimination backoff technique to improve performance under high contention. Threads attempting to push and pop are the same time can cancel each other out rather than changing the state of the stack. In this report I will analyze and compare each stack variant.

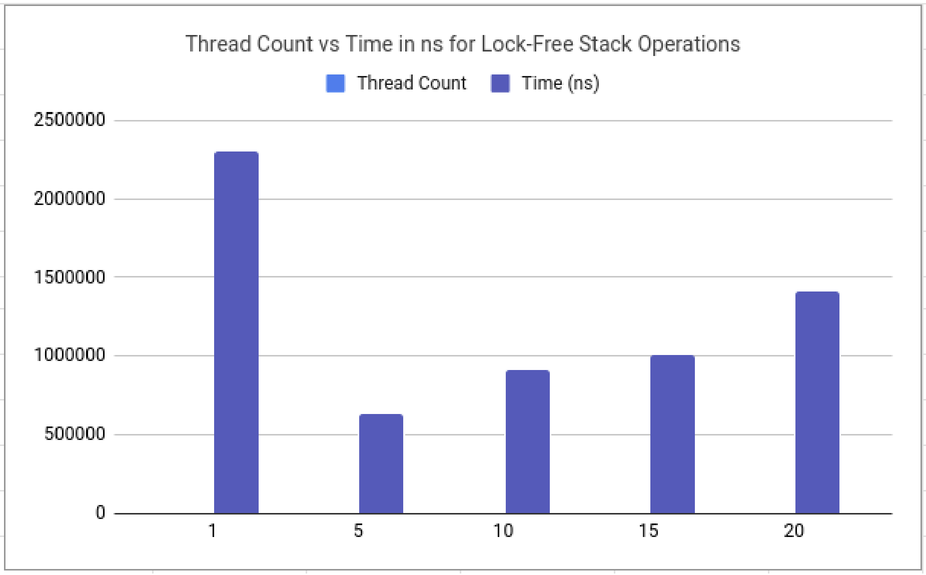
Hayley Dodkins u21528790

Results

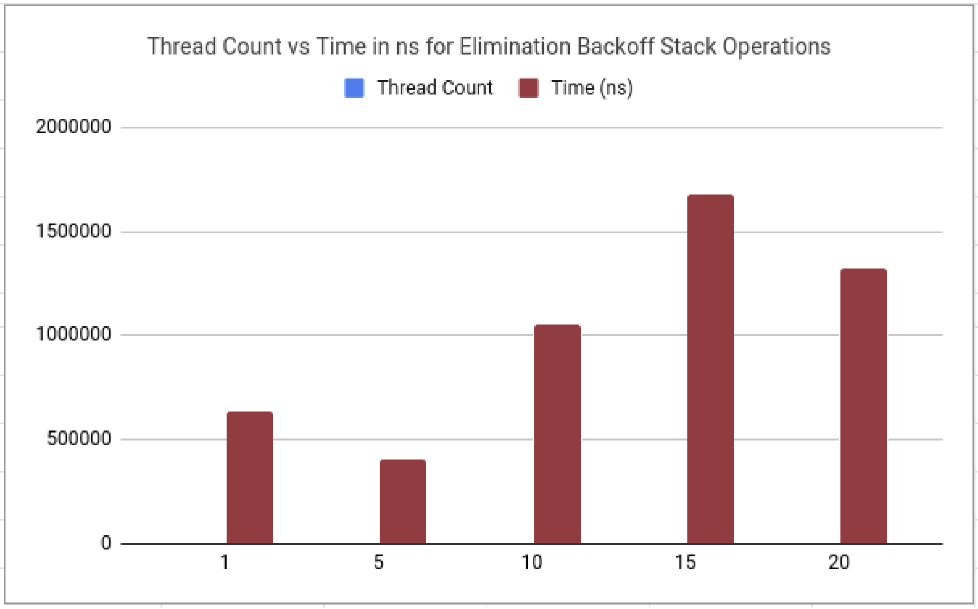
Thread Cou	Time (ns)
1	658555
5	461401
10	577738
15	8319138
20	1285893



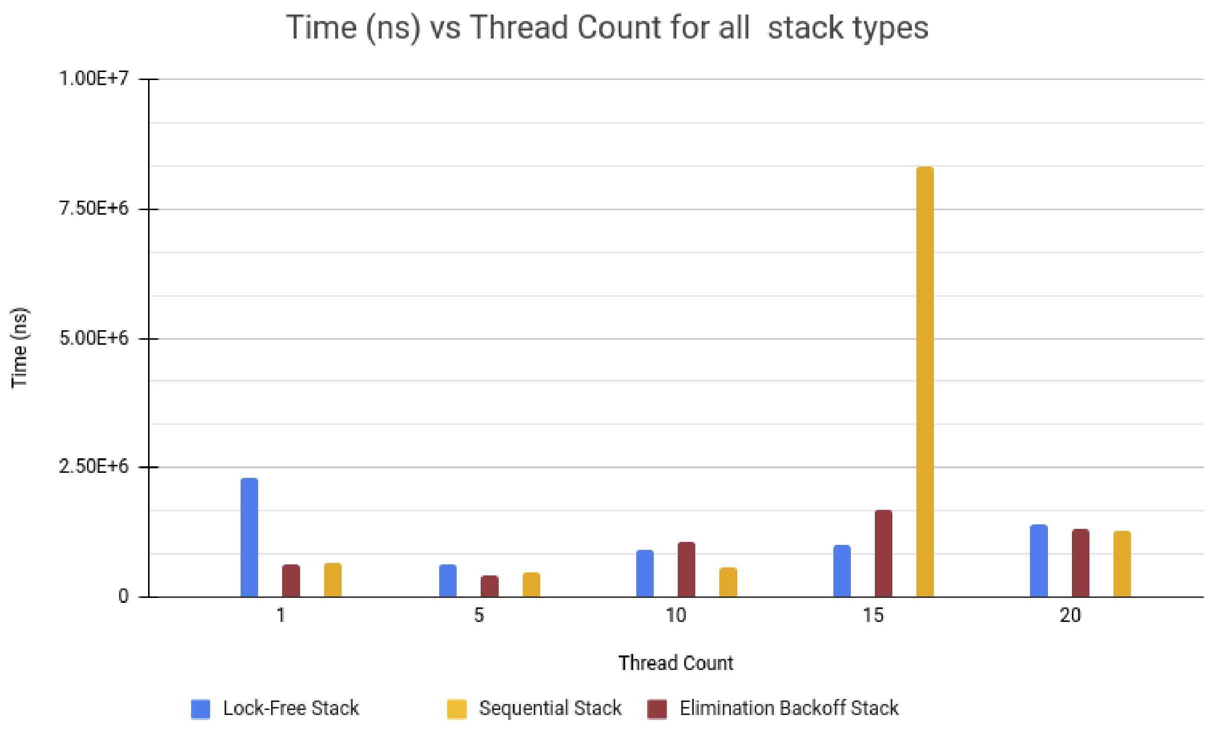
Thread Cou	Time (ns)
1	2301923
5	630542
10	911451
15	1004183
20	1410062

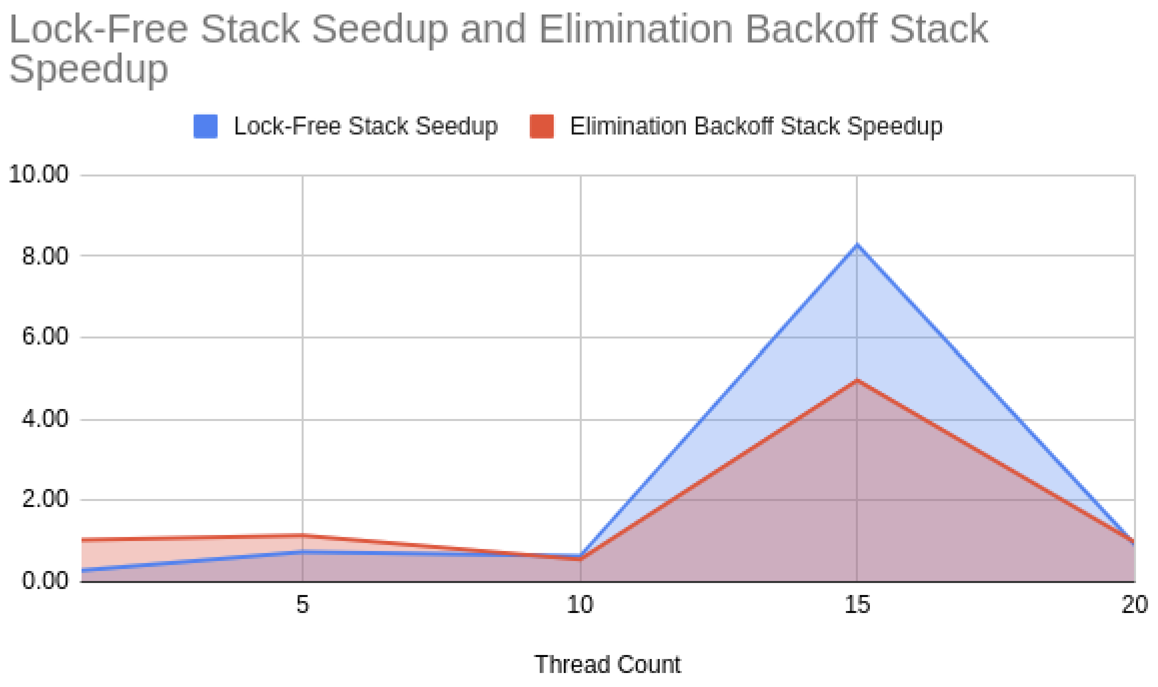


Thread Cou	Time (ns)
1	635263
5	405410
10	1052303
15	1678211
20	1323591



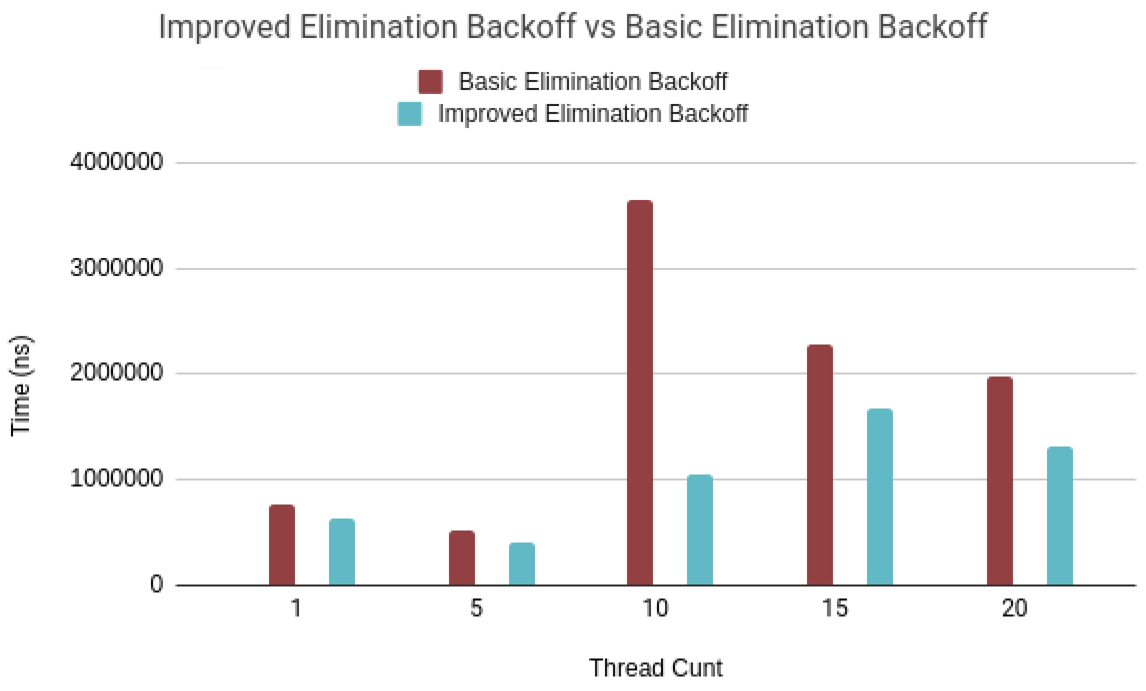
Hayley Dodkins u21528790





Speedup = sequential time/lock-free time or sequential/elimination time

Hayley Dodkins u21528790



Hayley Dodkins u21528790

Report

Lock-Free Analysis

In the single thread tests the Lock-free stack performed worse than the sequential stack with a speedup factor of 0.29. This performance could be due to the additional overhead of using atomic operations that are not beneficial when in a single threaded environment. The performance of the lock-free stack improves as the thread count increases. At 5 threads this stack implementation achieved a speedup of 0.73. This shows that the stack starts to benefit from concurrency but the overhead of atomic operations still affects performance. At 10 threads the speedup drops to 0.63 indicating that contention has increased and multiple threads try to access the stack concurrently. At 15 threads with a speedup of 8.28 this is a significant improvement from the previous tests. At this level represents where the stack gets the ideal balance between thread count and being able to use the cpu efficiently without excessive overhead from atomic operations. At 20 threads the speedup reduces back down to 0.91 indicating that the thread count exceeds favorable levels. The increased thread count led to more contention where the atomic operations increased and the additional overhead of context switching reduced the benefits. This aligns with the assumption that the lock-free stack is not scalable.

Elimination Backoff Stack

In a single threaded environment the Elimination backoff stack performed almost identical to the sequential stack achieving a speedup of 1.04. At low thread counts the elimination backoff strategy does not cause an overhead allowing it to operate similarly to the sequential stack. At 5 threads the elimination backoff stack performs better than the sequential stack with a speedup of 1.14 indicating that the backoff mechanism is beneficial at a moderate thread count where it is able to reduce contention and enables better concurrent stack access without overhead. The performance at 10 threads drops to a speedup of 0.55 suggesting that the elimination mechanism experiences some overhead. At this point the backoff mechanism may introduce some latency as more threads compete for access. The elimination array's overhead is more noticeable resulting in a slower overall performance.When working with 15 threads the speed up increased to 4,96 indicating that the backoff mechanism allows threads under higher contention to work more efficiently concurrently. This appears to be the most optimal thread count. At 20 threads the speedup declines to 0.96 indicating that it is close to the sequential stack's performance at this level. At this level the mechanism may cause too much latency and leads to increased overhead and the overhead from the context switching may impact performance drastically.

At 10 threads the improved elimination backoff strategy shows significant performance improvement at a speedup of 1.01 indicating the effects of the length of the elimination array and how that can effect the overall performance with the additional improvement allowing threads to wait in the array longer causes a better performance at this level and a mostly consistent performance at lover levels. At 15 threads we see similar results to the initial elimination backoff stack while at 20 threads we see a decrease in performance where the speedup dropped to 0.6 indicating the longer waiting time and decreased array length may contribute to the slower times as there is more contention in the array.

The sequential stack acts as the baseline for this report where it uses locks to provide sequential access to the stack using locks.

The best performance for both lock-free and elimination stack was seen at 15 threads. This indicates the idea balance between threads and overhead allowing both stack implementations to perform optimally. This also suggests that 15 threads may align with the number of available cpu cores also leading to better utilization of resources.

At 20 threads both stacks see a decline in performance which may be due to the increased overhead of context switching as the number of threads exceeds the core count. The higher thread count also leads to more contention. But from the results of the improved elimination backoff stack it suggests that by getting the right combination of backoff time and array length that the elimination stack is overall better at scaling.

Hayley Dodkins u21528790

Hayley Dodkins u21528790