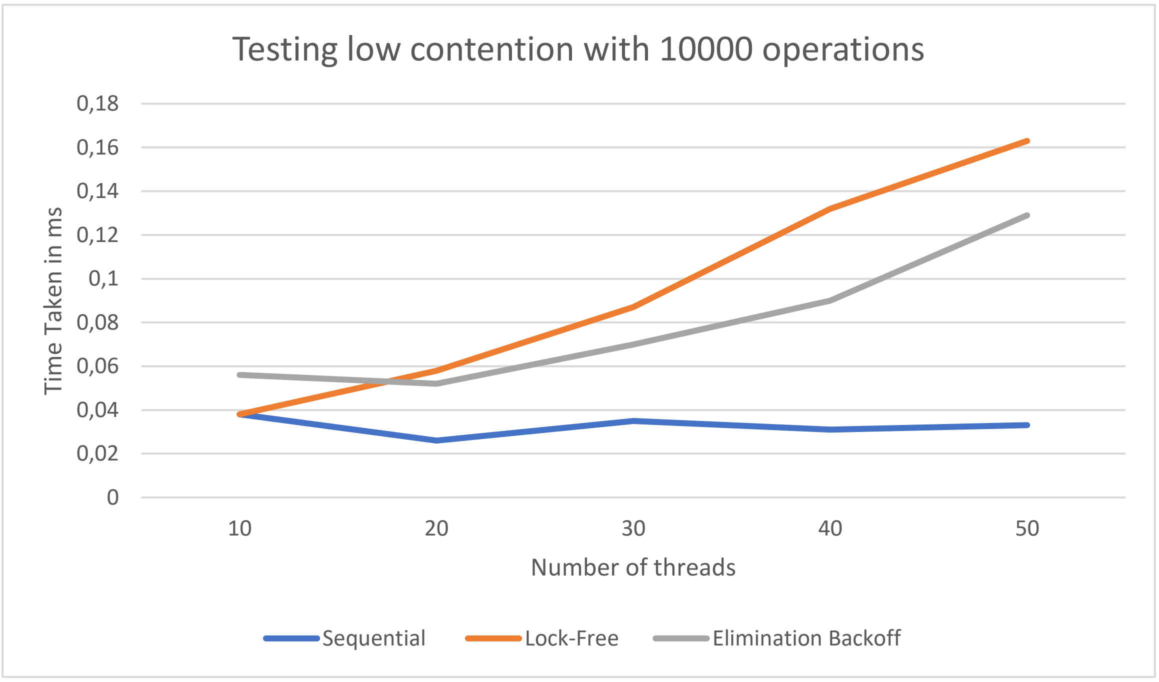
U22598546

Bahiya Hoosen

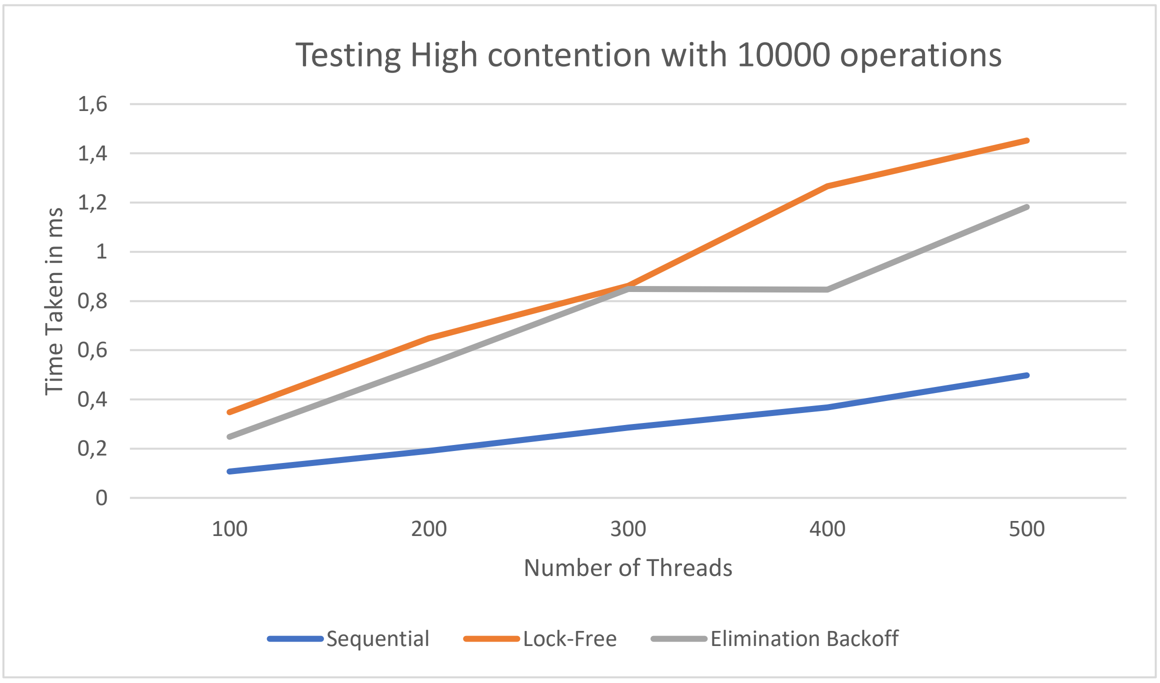
226 PRACTICAL 7

INTRODUCTION

In the wild world of concurrent programming, stacks are like traffic jams waiting to happen—especially when a bunch of threads want in on the action! We've got the **Sequential Stack**: the line-waiter, slow but steady. Then there's the **Lock-Free Stack**, the impatient rule-breaker that dodges locks and cuts in line with atomic moves. Finally, the **Elimination Backoff Stack** is the clever matchmaker—pairing threads up to swap values and skip the stack altogether. It's like a speed-dating service for threads under pressure, keeping things fast, fun, and free of pileups!



1. **Sequential/Blocking Stack Performance**
   The **Sequential Stack** performance remains stable and slightly improves as thread count increases, with execution times staying consistently low under low contention. **Sequential is better than Lock-Free because of its simple design**. The Sequential Stack doesn't attempt complex operations to handle contention; it just uses locks to allow one thread at a time to access the stack.
2. **Lock-Free Stack**
   As thread count increases, the **Lock-Free Stack** experiences higher execution times. **Elimination Backoff is better than Lock-Free because of contention management**. While the Lock-Free Stack is designed to prevent complete blocking, it still suffers as contention increases due to frequent retries when threads collide on stack operations. These retries lead to an increase in execution time, especially when many threads are accessing it simultaneously.
3. **Elimination Backoff Stack**
   The **Elimination Backoff Stack** maintains relatively low execution times even as thread count increases, and successful eliminations increase with more threads. **Elimination Backoff is better than both Sequential and Lock-Free because of its elimination strategy under contention**. This stack allows threads to "eliminate" each other's operations, meaning they don't all access the central stack



1. **Sequential/Blocking Stack Performance**
   The **Sequential Stack** shows a consistent increase in execution time as the number of threads increases, ranging from 0.113 seconds at 100 threads to 0.465 seconds at 500 threads.**Sequential is better than Lock-Free because it handles low contention scenarios more effectively**. In high contention, however, it becomes less efficient because multiple threads must wait for their turn to access the stack, leading to increased wait times and contention delays. The sequential nature of this approach causes significant overhead as more threads are introduced, making it unsuitable for high contention environments.
1. **Lock-Free Stack**
   The **Lock-Free Stack** demonstrates a sharp increase in execution time as the number of threads increases, from 0.345 seconds at 100 threads to 1.692 seconds at 500 threads. �
   Sequential is better than Lock-Free under low contention because of its simplicity.
   However, as contention rises, Lock-Free stacks struggle with increased retries and collisions among threads trying to perform push/pop operations. The need for threads to frequently retry their operations when accessing shared data leads to longer execution times, highlighting that while Lock-Free mechanisms are designed to prevent blocking, they are not immune to performance degradation under high contention.
2. **Elimination Backoff Stack**
   The **Elimination Backoff Stack** maintains relatively better performance, with execution times increasing from 0.210 seconds at 100 threads to 1.201 seconds at 500 threads, while also reporting a substantial number of successful eliminations. **Elimination Backoff is better than both Sequential and Lock-Free because of its efficient contention management**. This stack's ability to allow threads to eliminate each other's operations significantly reduces contention on the central stack. As the number of threads increases, it effectively pairs threads to perform operations directly with each other, minimizing the need for all threads to compete for access to the stack. This reduces wait times and enhances throughput, resulting in better scalability and performance under high contention compared to both the Sequential and Lock-Free stacks.

   Conclusion

- In the low contention tests, both the **Elimination Backoff Stack** and **Lock-Free Stack** demonstrated similar performance trends. � This suggests that at low levels of contention, the performance of **Elimination Backoff Stack** can indeed be comparable to that of the **Lock-Free Stack**. The slight edge in time efficiency for the Elimination Backoff Stack indicates that it can perform efficiently in environments with fewer contention issues.

- At high levels of contention, the trends diverge significantly. The **Lock-Free Stack** showed a dramatic increase in execution time, from **0.345 seconds** at 100 threads to **1.692 seconds** at 500 threads. In contrast, the **Elimination Backoff Stack** managed to maintain better performance, with execution times ranging from **0.210 seconds** at 100 threads to **1.201 seconds** at 500 threads.

  Additionally, the number of successful eliminations for the Elimination Backoff Stack increased substantially, indicating that the stack was effectively allowing threads to pair up and complete operations without contending for the same resource. The recorded successful eliminations went from **27,6534** at 100 threads to **1,748,249** at 500 threads. This showcases the ability of the Elimination Backoff Stack to handle higher contention efficiently by reducing the overall wait times and allowing more operations to be completed in parallel.

REFERENCES

1. Java Stacks – The Basics
   https://www.youtube.com/watch?v=afaIN_hloB4
2. Lock Free Concurrency – CAS
   https://www.youtube.com/watch?v=DcWfs7YlXLc
3. Herlihy, M., & Shavit, N. (2012). *The Art of Multiprocessor Programming* (Revised First Edition). Elsevier.