**Performance Analysis of Concurrent Stack Implementations**

Arnaud Zander Strydom

u23536013

**Introduction**

This report analyzes the performance characteristics of three concurrent stack implementations: Sequential (Lock-based), Lock-free, and Elimination Backoff stacks. The analysis focuses on three key metrics: execution time, waiting time, and throughput, evaluated across varying thread counts (2, 4, 8, and 16 threads) with 100,000 operations per thread.

**Experimental Results and Analysis**

1. **Time Taken vs Number of Threads**

**A graph with a line graph and numbers

Description automatically generated**

Sequential Stack: 2→92ms, 4→38ms, 8→60ms, 16→73ms

Lock-Free Stack: 2→53ms, 4→107ms, 8→345ms, 16→1173ms

Elimination Stack: 2→54ms, 4→10ms, 8→52ms, 16→81ms

**Trend Analysis:**

* The Sequential Stack shows relatively stable performance with moderate scaling
* The Lock-Free Stack exhibits significant performance degradation as thread count increases
* The Elimination Stack maintains consistent performance across thread counts, with best performance at 4 threads

**Reasons:**

1. Sequential Stack's stable performance comes from its simple lock-based approach, which manages contention effectively but limits parallelism
2. Lock-Free Stack suffers from increasing contention on the shared top reference, leading to more CAS failures
3. Elimination Stack excels by allowing pairs of push/pop operations to cancel out without accessing the shared stack

**2. Average Waiting Time Analysis**

A graph with a line graph and numbers

Description automatically generated with medium confidence

Sequential Stack: 2→9.50ms, 4→7.25ms, 8→6.63ms, 16→7.50ms

Lock-Free Stack: 2→7.00ms, 4→6.75ms, 8→6.75ms, 16→8.31ms

Elimination Stack: 2→7.00ms, 4→7.25ms, 8→8.25ms, 16→7.25ms

**Trend Analysis:**

* All implementations show relatively consistent waiting times
* Slight increase in waiting time for Lock-Free Stack at 16 threads
* Elimination Stack maintains stable waiting times across thread counts

**Reasons:**

1. Sequential Stack's lock-based approach provides predictable waiting times
2. Lock-Free Stack's increased contention at higher thread counts leads to more retries
3. Elimination Stack's pairing mechanism helps maintain stable waiting times

**3. Throughput Analysis**

A graph with a line graph and numbers

Description automatically generated with medium confidence

Sequential Stack: 2→10.5M, 4→16.8M, 8→18.2M, 16→19.3M ops/sec

Lock-Free Stack: 2→11.2M, 4→4.3M, 8→2.4M, 16→1.3M ops/sec

Elimination Stack: 2→30.1M, 4→25.0M, 8→9.1M, 16→3.1M ops/sec

**Trend Analysis:**

* Sequential Stack shows steady throughput improvement with more threads
* Lock-Free Stack's throughput decreases significantly with thread count
* Elimination Stack shows best throughput at low thread counts but decreases with scaling

**Reasons:**

1. Sequential Stack's lock-based approach provides consistent scaling due to orderly access
2. Lock-Free Stack suffers from contention overhead, leading to decreased throughput
3. Elimination Stack excels at low thread counts due to successful operation pairing but faces coordination overhead at higher thread counts

**Conclusions**

1. **Sequential Stack:**
   * Most predictable performance
   * Good scaling up to 16 threads
   * Best choice for moderate concurrency scenarios
2. **Lock-Free Stack:**
   * Good performance with low thread counts
   * Poor scaling with increased concurrency
   * Not recommended for high-contention scenarios
3. **Elimination Stack:**
   * Best overall performance at low to moderate thread counts
   * Most efficient for balanced push/pop workloads
   * Optimal for scenarios with 2-4 threads

The results demonstrate that the choice of concurrent stack implementation should be based on the specific use case:

* For predictable performance: Sequential Stack
* For low contention scenarios: Lock-Free Stack
* For optimal performance with moderate threading: Elimination Stack

**References**

1. Hendler, D., Shavit, N., & Yerushalmi, L. (2004). A scalable lock-free stack algorithm. SPAA '04.
2. Michael, M. M., & Scott, M. L. (1996). Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. PODC '96.
3. Anthropic. (2024). Claude [Large Language Model]. Used for analysis and report generation.
4. Herlihy, M., & Shavit, N. (2012). The Art of Multiprocessor Programming. Morgan Kaufmann.