Hayley Dodkins u21528790

Introduction

In a faraway land where the threads ran wild,
Three sync strategies lived, each with their own style.
They all had one job to keep the data just right,
But each took their turn to show who did it right.

First was Coarse-Grained, so big and so wide,
“Just one lock for everything!” it said with great pride.
But the threads were annoyed, stuck waiting in line,
"Come on, big guy, you’re taking your time!"

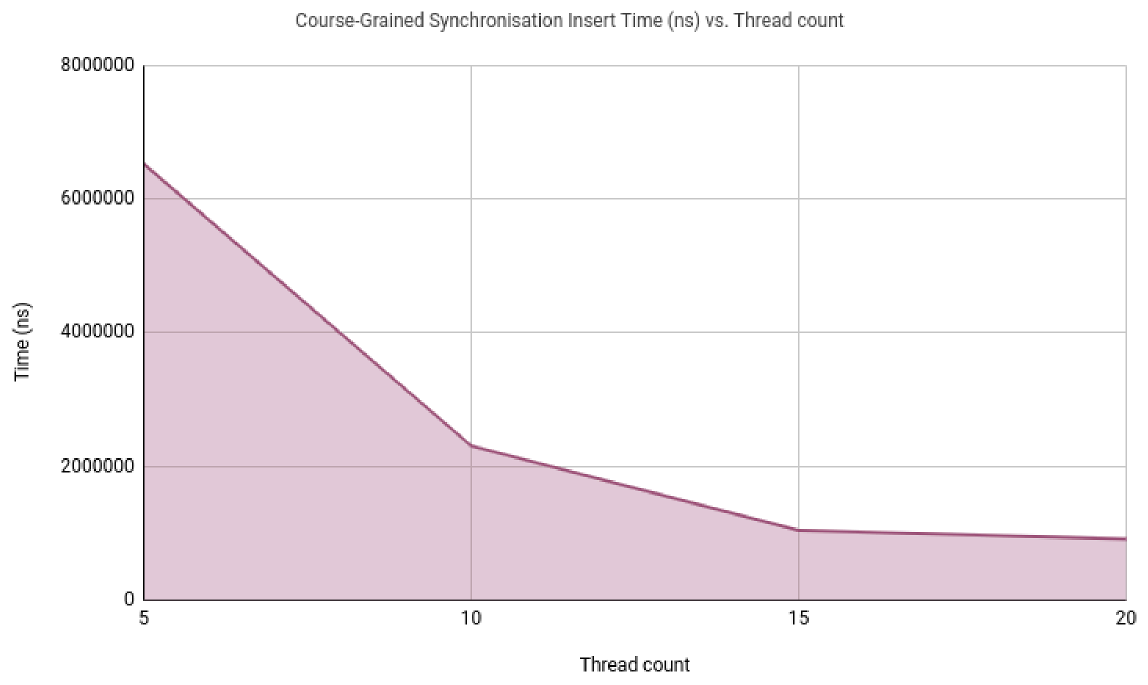
Then came Fine-Grained, with locks small and neat,
“I’ll lock little bits, so I’m light on my feet!”
But too many locks led to confusion and strife,
Now threads were all tangled in a locking life!

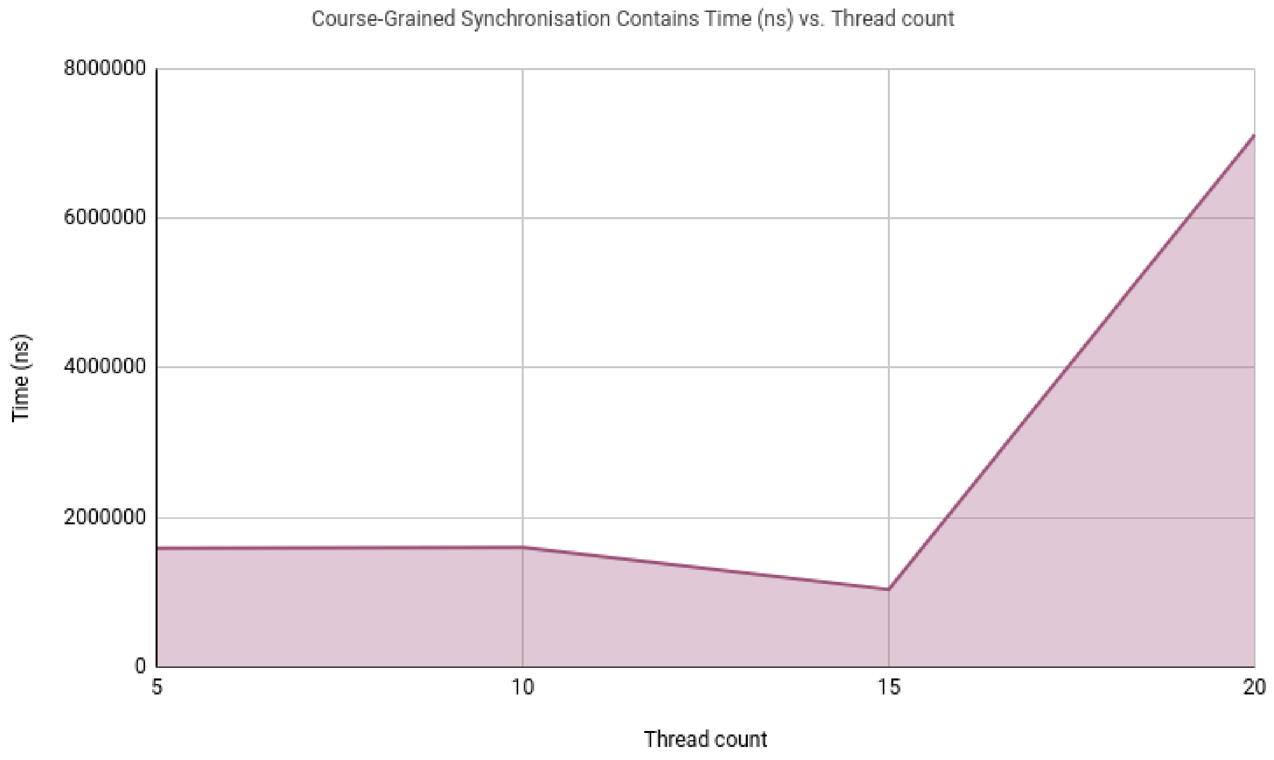
Then Optimistic strolled with a confident grin,
"I won’t lock right away, I’ll just dive in!"
It checked at the end if the coast was clear,
"Oops, retry!" But with speed, no fear!

So whether big, small, or just taking a chance,
Each lock had a role in this thread-driven dance.
But which one to choose? Oh, that’s up to you!
Just find the right lock for the task you must do!

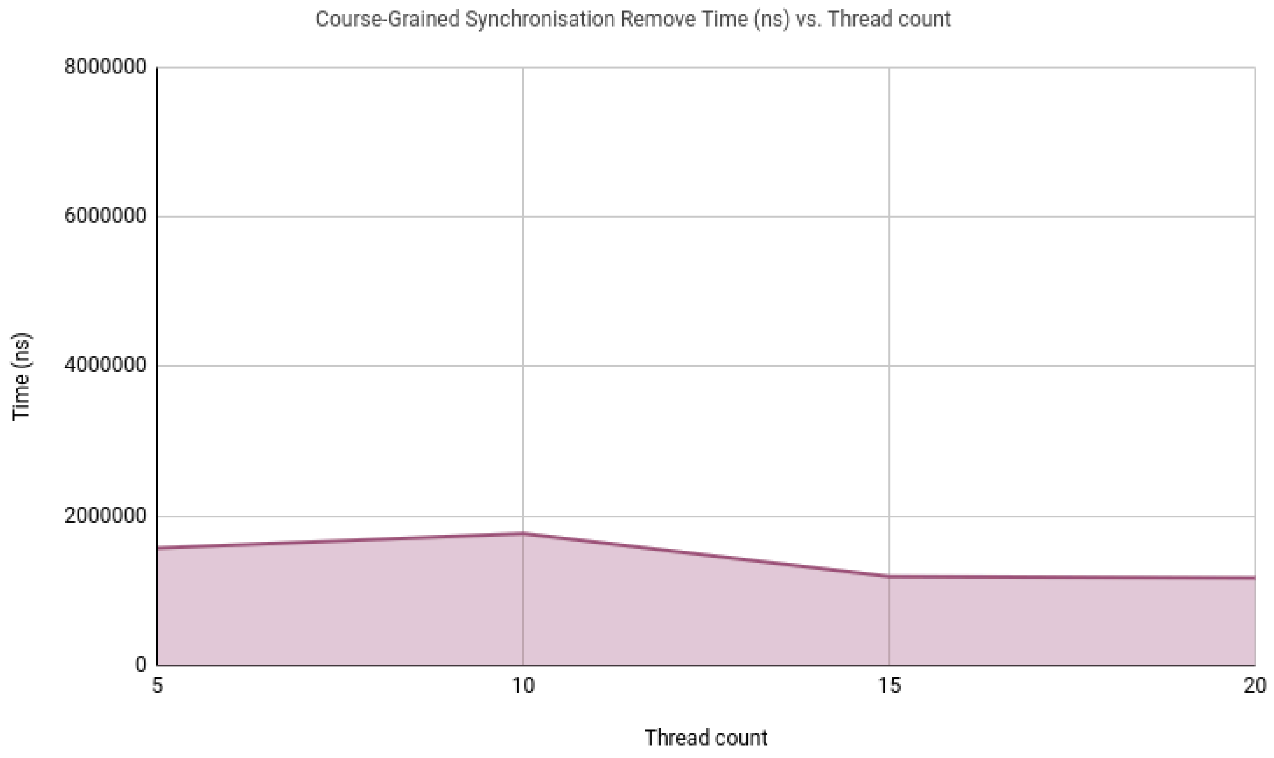
Hayley Dodkins u21528790

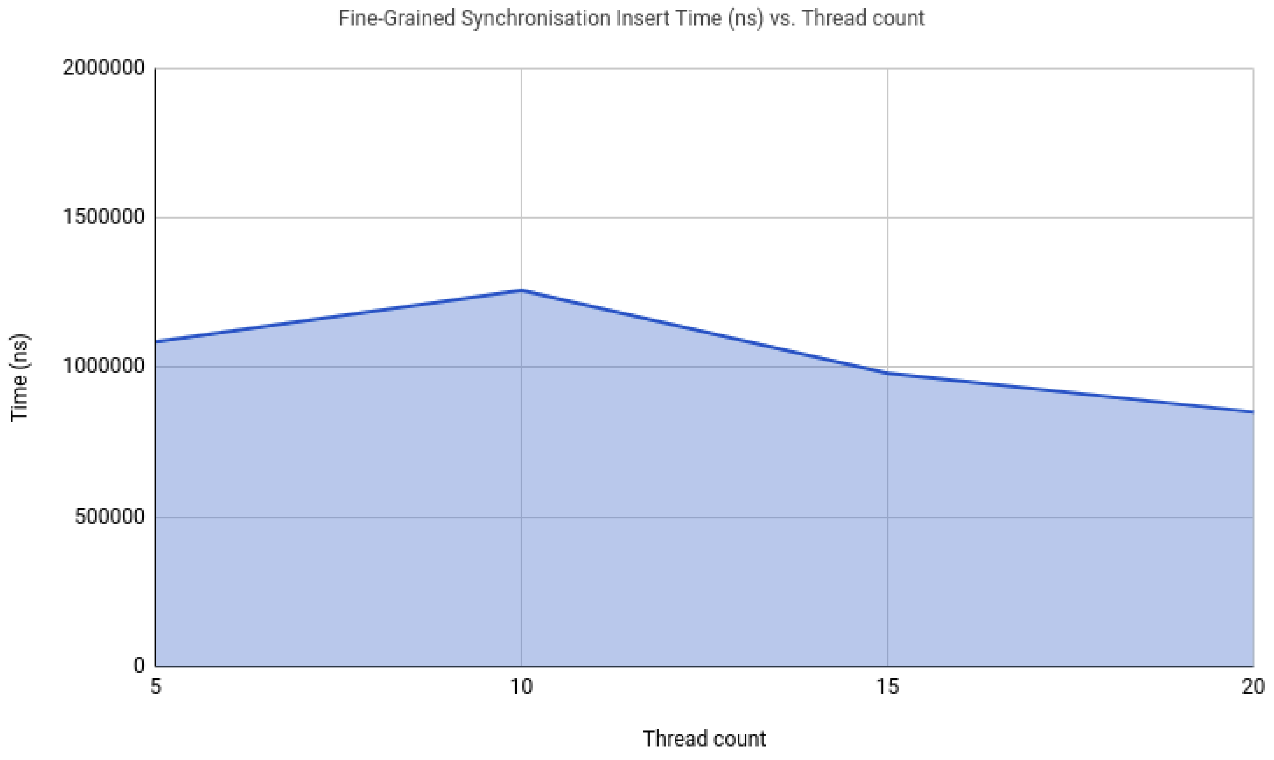
Results



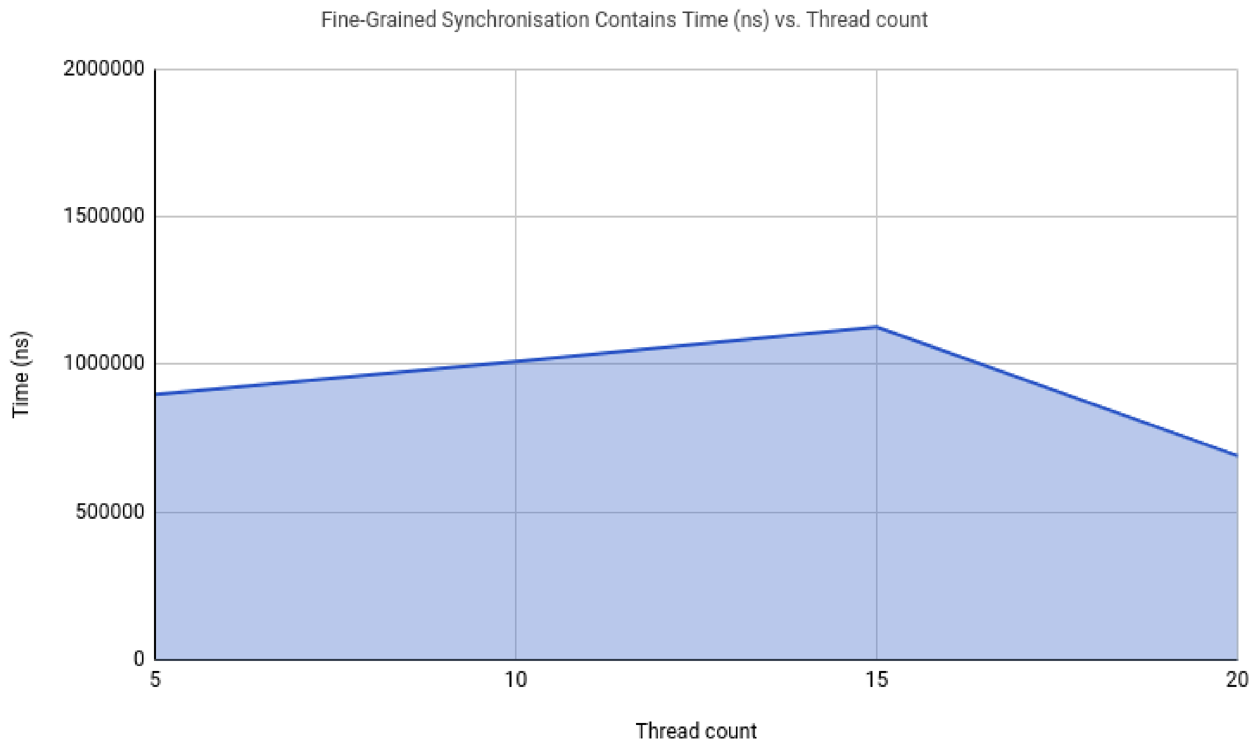


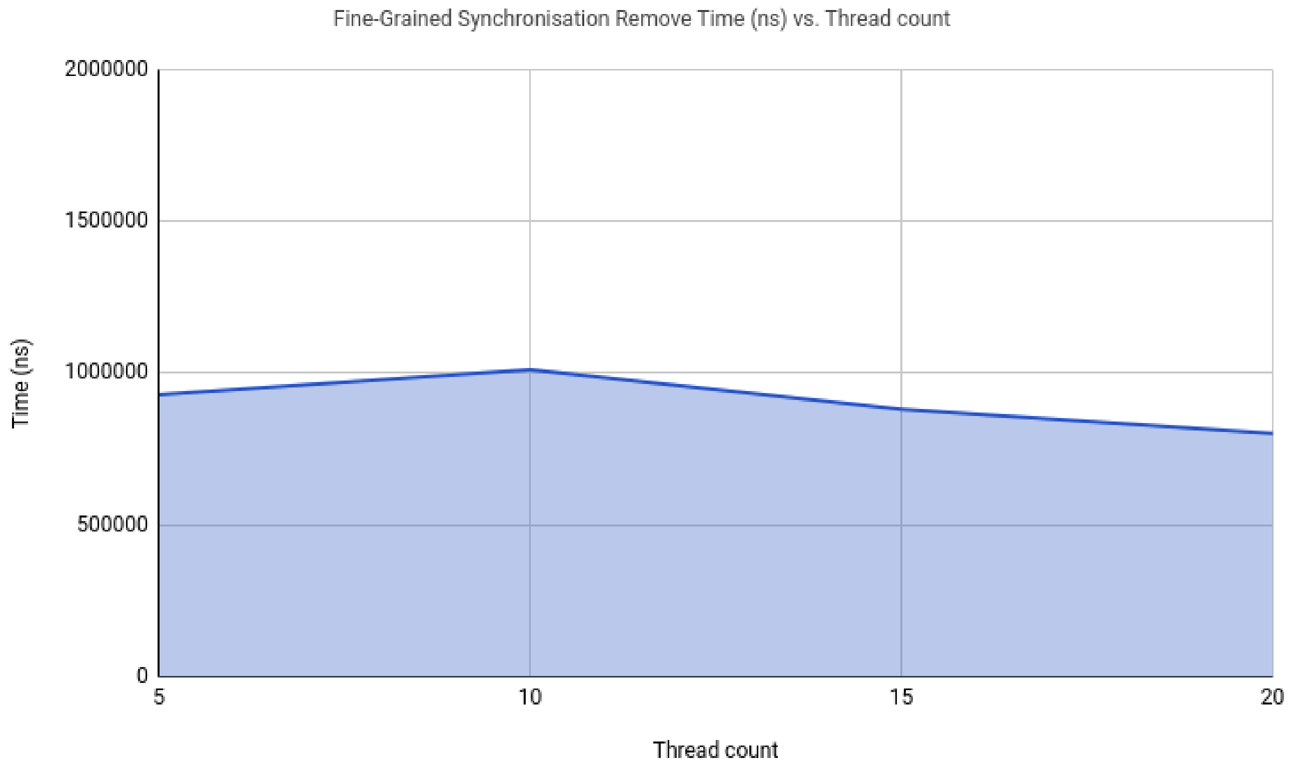
Hayley Dodkins u21528790



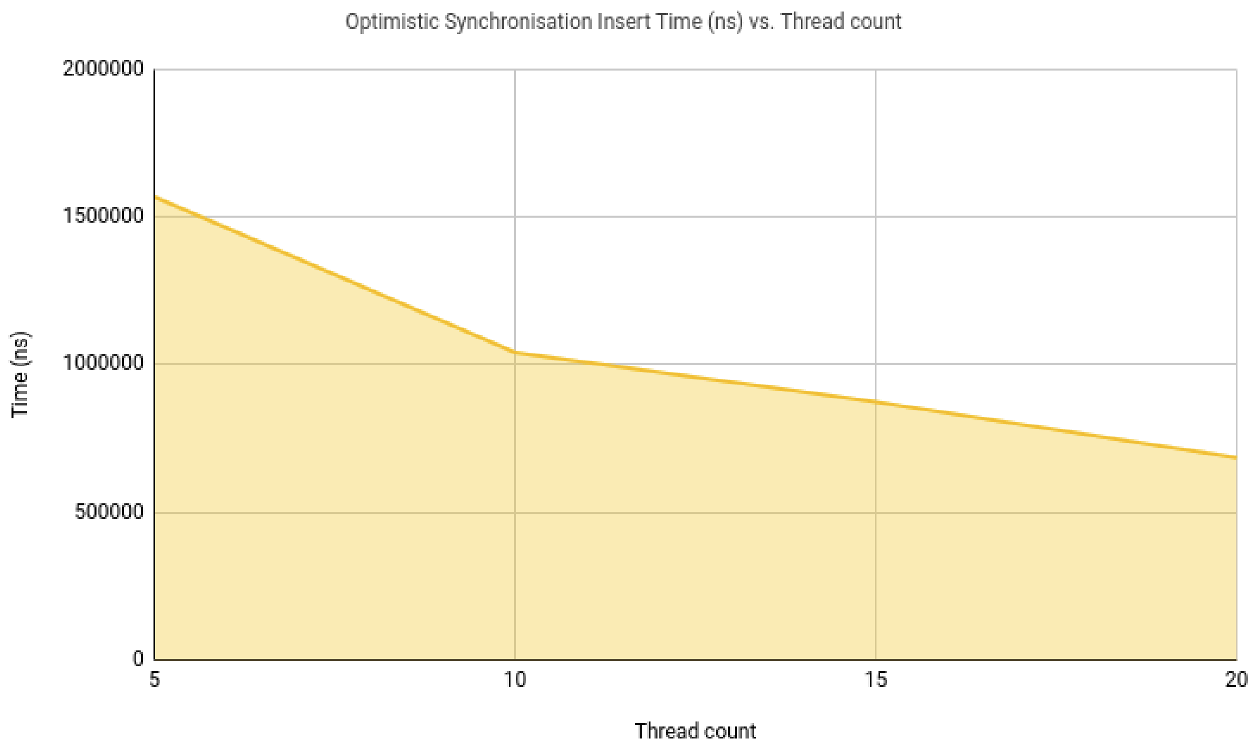


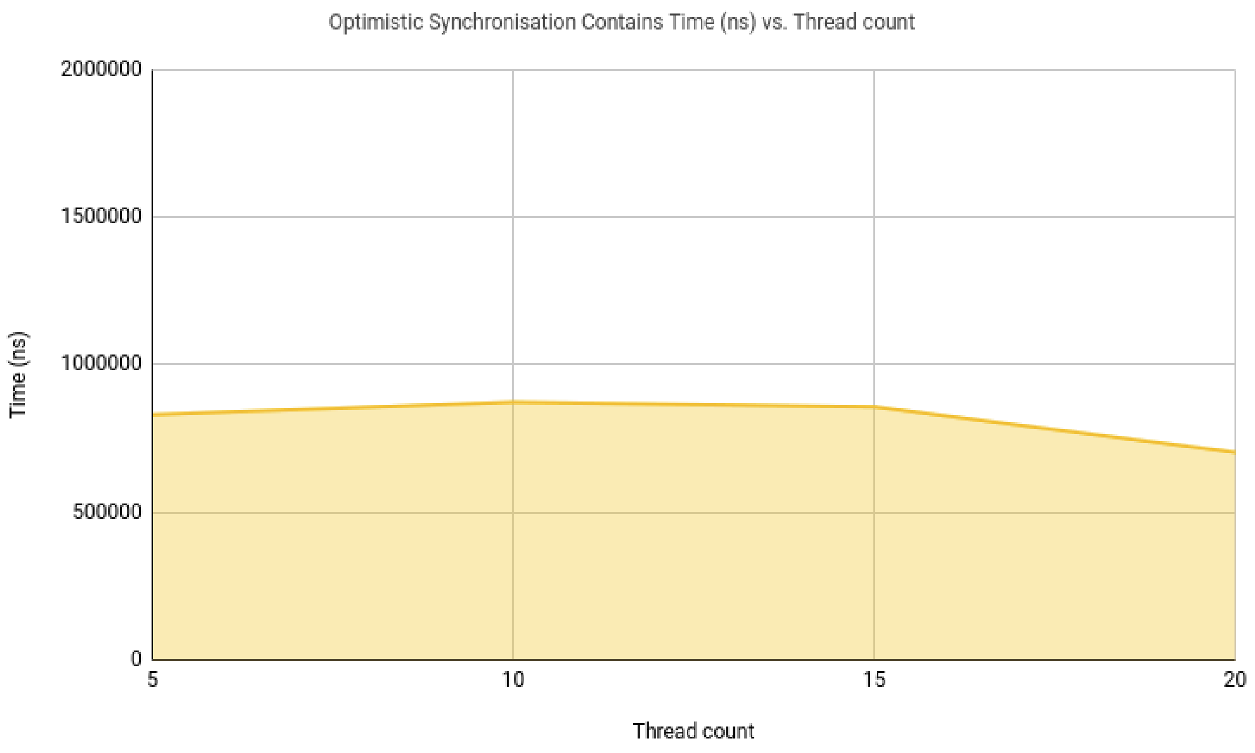
Hayley Dodkins u21528790



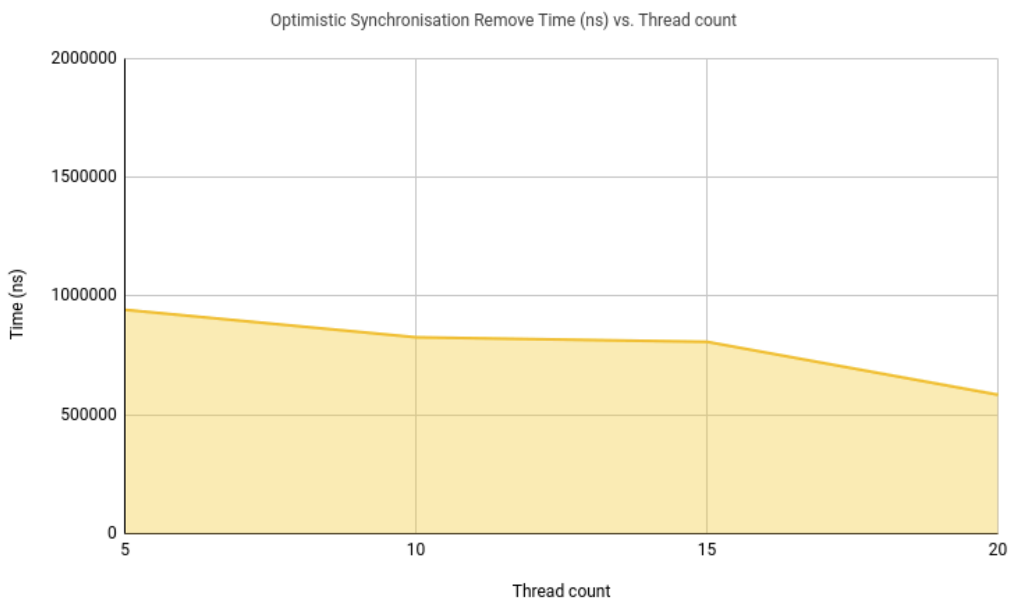


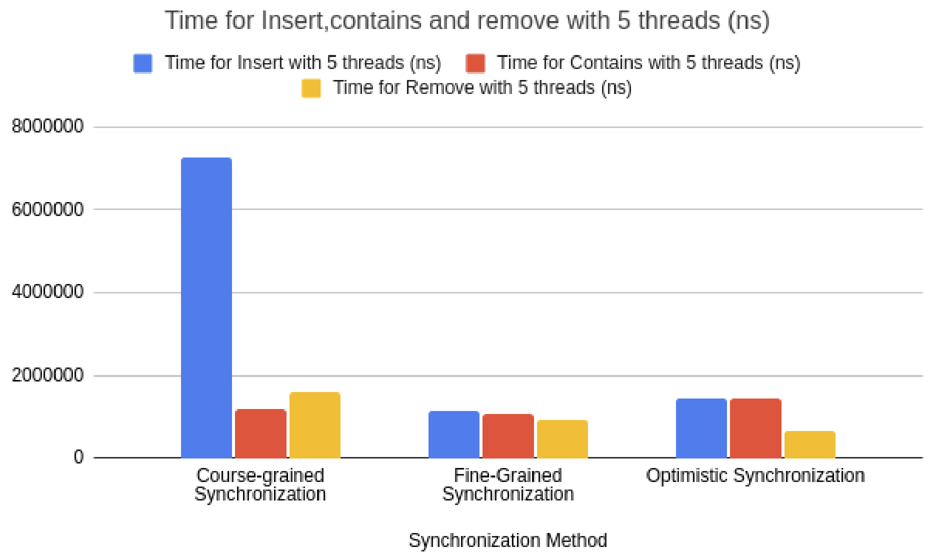
Hayley Dodkins u21528790

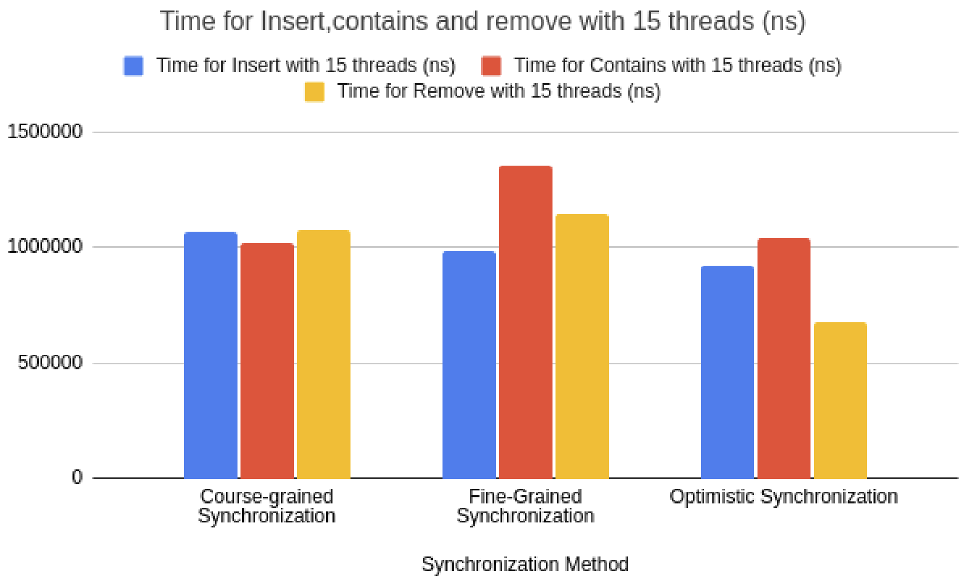




Hayley Dodkins u21528790







Hayley Dodkins u21528790

Report

Testing Environment

The performance of multi-threaded applications can vary significantly based on CPU architecture. The tests were performed using an Intel Core i9 CPU, which plays a crucial role in the observed results.

. The i9 features Hyper-Threading, allowing multiple threads to be processed simultaneously across its cores, which can reduce context-switching overhead and increase parallelism. As a result, higher thread counts generally lead to improved performance for tasks like tree insertion.

However, performance at lower thread counts (5 threads) may initially be lower due to inefficient resource utilization. Fewer threads can lead to idle time and suboptimal execution, as the CPU might not fully engage its capabilities. In contrast, with a higher number of threads (20 threads), the workload is better distributed, maximizing CPU efficiency and resulting in faster operation.

Additionally, the i9's advanced caching helps speed up multi-threaded workloads by improving data access times. As thread counts rise, the likelihood of accessing cached data increases, enhancing overall performance.

Coarse-GrainedSynchronization Analysis

This report presents the results of an evaluation of coarse-grained synchronization for binary tree insertion operations. Coarse-grained synchronization involves using a single global lock to protect access to an entire data structure, which simplifies implementation but often limits scalability due to increased contention as the number of threads grows. The experiment was conducted on a multi-core Linux system powered by an Intel Core i9 processor, which supports hyperthreading.

Hayley Dodkins u21528790

The tests were run on a system with an Intel Core i9 processor featuring hyperthreading (SMT). This architecture allows each physical core to handle two hardware threads (logical cores), thereby potentially increasing throughput in multi-threaded scenarios where context switching and CPU utilization are critical.

From 5 to 10 threads, there is a 73% decrease in execution time, from 7,270,252 ns to 1,985,641 ns. This increase in performance seems counterintuitive given the coarse-grained locking mechanism enforces sequential execution of critical sections. However the following factors can contribute to the performance improvement;

CPU Utilization and Thread Scheduling:
The CPU features hyperthreading, which allows each physical core to run two logical threads concurrently. With 5 threads, the system may not fully utilize all available CPU cores (both physical and logical), leading to periods of underutilization. The operating system's scheduler may not always keep all logical cores busy when there are fewer threads.Increasing the thread count to 10 allows the OS to better distribute the workload across available logical cores. Although contention exists due to the single lock, the increased number of threads ensures that more CPU resources are engaged when the lock is released. This reduces overall idle time as threads "take turns" faster, which keeps the CPU more active and increases throughput.

As the number of threads increases beyond 10, the reduction in execution time becomes less pronounced. In the tests with 15 and 20 threads, there is a higher degree of contention for the single lock. Although hyperthreading helps in reducing the impact of idle times to some degree, it cannot overcome the fundamental bottleneck caused by the lock itself. The additional overhead of managing more threads combined with the single lock becomes the dominant factor for why there is less of a decrease in time from 15 threads to 20 threads.

This experiment demonstrated both the advantages and limitations of coarse-grained synchronization in a multi-threaded environment. Initially, increasing the thread count from 5 to 10 resulted in substantial performance improvements, largely due to better CPU utilization facilitated by hyperthreading and more efficient thread scheduling. However, as the thread count continued to increase, performance gains diminished due to increased contention for the single lock.

While coarse-grained synchronization offers simplicity, it severely limits the scalability of multi-threaded applications as contention grows.

Fine-Grained Synchronization Analysis

Fine-grained synchronization differs from coarse-grained synchronization in that it allows for more granular locking mechanisms, where each nodes of the data structure is protected by a distinct lock. This approach allows for increased concurrency and reduced contention compared to the coarse-grained synchronization.

Initially, the execution time increases when moving from 5 threads (1,152,036 ns) to 10 threads (1,275,813 ns), this performance degradation can be explained by the complexity of managing multiple locks. Fine-grained synchronization involves managing multiple locks across the data structure. With few threads, the overhead of acquiring and releasing multiple locks might outweigh the benefits of reduced contention. At this thread count, there may still be significant contention for tree sections near the root, where many threads initially interact before branching out. Increased threads can also create overhead related to cache coherence and memory management. The fine-grained approach leads to more frequent lock acquisitions and releases, which can lead to memory access patterns that affect performance. In addition, as threads access different parts of the tree, cache misses can become more frequent, reducing efficiency.

With more threads, the benefits of fine-grained locking become clearer. As threads distribute across different parts of the binary tree, the contention for locks decreases because each thread is more likely to interact with different parts of the tree. In this test, the fine-grained locking mechanism allows true concurrent access to different sections of the tree, improving concurrency and leading to a reduction in execution time.

When the thread count is increased to 20, the execution time rises slightly to 1,068,344 ns, indicating a performance plateau. As the number of threads grows, the complexity of managing multiple locks increases. Each thread must acquire and release multiple locks during its operations, which introduces more synchronization overhead, especially in deeper parts of the tree where threads are likely to interact with more nodes. At high thread counts, false sharing (when multiple threads access variables that reside in the same cache line) may begin to impact performance, causing unnecessary synchronization overhead. Additionally, more frequent cache misses and higher memory latency could contribute to the slight increase in execution time observed at 20 threads.

Optimistic Synchronization Analysis

Optimistic synchronization differs from both coarse- and fine-grained locking mechanisms by allowing threads to proceed with operations under the assumption that contention will be rare, only validating their operations at critical points. If a conflict is detected, the operation may roll back and retry.

The performance of optimistic synchronization improves significantly as thread count increases from 5 to 15, where the execution time drops from 1,452,671 ns to 918,900 ns. This improvement aligns with the expected behavior of optimistic concurrency control under conditions of low to moderate contention. With 5 threads, optimistic synchronization performs reasonably well, but not as efficiently as fine-grained synchronization. The relatively high initial time (1,452,671 ns) suggests that contention is not entirely absent, and the overhead of validating operations at critical points adds to the total time. If conflicts are detected, threads need to retry their operations, which introduces some delays. Increasing the thread count to 10 and 15 threads results in performance gains, dropping execution time to 947,096 ns and 918,900 ns, respectively. The concurrency enabled by optimistic synchronization, combined with fewer conflicts, allows for better CPU utilization. Since optimistic synchronization minimizes the need for strict locking, it allows threads to work concurrently on different parts of the tree with fewer interruptions.

At 20 threads, there is a slight increase in execution time to 976,669 ns, signaling a performance plateau. The marginal increase in time (from 918,900 ns to 976,669 ns) indicates that the system has reached a point where contention or resource overhead begins to offset the benefits of optimistic concurrency. With 20 threads, optimistic synchronization faces higher chances of conflicts when multiple threads attempt to modify overlapping parts of the binary tree. As a result, threads may have to roll back and retry their operations more frequently, leading to a slight increase in overall execution time. The higher thread count increases the likelihood of contention for the same tree nodes, especially near the root, where many insert operations converge initially.

The overhead of detecting conflicts and rolling back operations becomes more significant as thread count rises, leading to diminishing returns in terms of performance. Compared to fine-grained synchronization, optimistic synchronization showed similar scalability, especially at moderate thread counts. It performed better than coarse-grained synchronization by allowing more parallelism without the overhead of managing multiple locks. However, at very high thread counts, it faces challenges due to increased conflicts and rollbacks, which limits its scalability beyond a certain point.

Conclusion

Coarse-grained synchronization is simple and easy to implement, and showed significant performance improvements from 5 to 10 threads due to better CPU utilization. However, it bottlenecked at higher thread counts due to contention for a single global lock, leading to diminishing returns and scalability limitations. The fundamental issue with coarse-grained synchronization is the heavy contention that occurs as more threads attempt to acquire the single lock, preventing concurrency.

Fine-Grained Synchronization, initially, exhibited a slight performance degradation as managing multiple locks created overhead. However, as thread counts increased, this method outperformed coarse-grained synchronization, offering better concurrency and reduced contention. The more granular locking allowed threads to operate on different parts of the binary tree simultaneously, improving efficiency up to a point. However, at very high thread counts, the overhead of managing multiple locks and cache coherence issues started to limit further gains.

Optimistic synchronization showed the best performance scaling up to 15 threads, with fewer constraints from locking, allowing for more concurrency. However, at 20 threads, the increased likelihood of conflicts and retries led to a performance plateau. While optimistic synchronization is more scalable than coarse-grained locking, its reliance on assumptions of low contention makes it vulnerable to performance drops when conflicts occur more frequently.
In conclusion, fine-grained synchronization offers the best trade-off between scalability and concurrency, but optimistic synchronization can be highly effective under conditions of low to moderate contention. Coarse-grained synchronization, while simple, does not scale well in multi-threaded environments due to excessive contention for the single lock.

Bibliography

Intel Corporation (n.d.) *Intel® Hyper-Threading Technology*. Available at: https://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html (Accessed: 13 October 2024).

Stack Overflow (2015) *Thread concurrency in Linux*. Available at: https://stackoverflow.com/questions/30936100/thread-concurrency-in-linux (Accessed: 13 October 2024).

Hayley Dodkins u21528790

Recipe

Ingredients

- 1 cup of Neovim superiority
- 2 tablespoons of Linux sophistication
- A pinch of hyperthreading zest
- 3 heaping scoops of synchronization strategies
- A dash of debugging patience
- A splash of data analysis syrup
- A handful of thread contention crunch
- A sprinkle of i9 elegance

Method

1. In a mixing bowl, whisk together the Neovim superiority and Linux sophistication until it's smooth to create a flawless blend, this is the foundation for all good code.
2. Scoop in the synchronization strategies, one at a time, mixing thoroughly to create the perfect tests.
3. Fold in a handful of thread contention crunch to elevate the synchronization strategies and add some testing scenarios to the report.
4. Slowly add a dash of debugging patience to make sure that the code actually works.
5. Before you pop it in the fridge, sprinkle in a pinch of hyperthreading zest and the sprinkle of i9 elegance to influence the results.
6. Let it chill while you procrastinate.
7. Once chilled, drizzle the splash of data analysis syrup over the report to bring everything together beautifully.