AVL Tree Parallelism Analysis: Project Milestone Report

I. Completed tasks

1. Current progress by week

Week	Task	In-charge	Progress
Week 0 (3/28-3/31)	Create sequential implementation and research coarse-grained and fine-grained approaches	Matt	Finished
Week 1 (4/1-4/7)	Create correctness and performance tests Finish coarse-grained lock implementation	Matt	Finished
Week 2 (4/8-4/14)	Finish fine-grained lock implementation	Matt, Tra	On-going

2. Details of accomplished tasks

- During the first three weeks, we implemented the coarse-grained lock implementation of AVL trees using mutexes, tested its correctness against the sequential version and obtained a speed up graph (see Part IV). For correctness, we created different edge test cases with multiple AVL tree instructions and utilized AVL tree invariant-checking functions. We optimized the coarse-grained implementation by using separate read and write mutexes to allow for concurrent reading (searching). We also swapped from using OpenMP to the pthread library; this enables more specific allocation of work for testing and performance analysis. With OpenMP, the task assignment to threads was more variable, which made it harder to test certain edge cases and workloads.
- We are currently working on the fine-grained lock implementation using hand-over-hand locking. Again, we are using the pthread library to create parallelism,

and we are building our implementation upon the coarse-grained implementation that we have already created.

II. Our plan for the remaining tasks

1. Original plan for after the milestone

Week	Task	In-charge	Progress
Week 4 (4/21)	Enforce correctness, optimize performance, and write milestone report	Both	Finished
Week 5 (4/28)	Finish lock-free implementation	Both	Not started
Week 6 (5/5)	Finish performance analysis	Both	On-going

2. Changes compared to the original schedule

- The fine-grained implementation (originally assigned to Tra) is behind schedule. Tra experienced time issues due to Spring Carnival. We expect to have more intensive progress in the coming weeks and to finish the fine-grained implementation by the end of this week.
- We focused heavily on the correctness and performance of the coarse-grained locking implementation, which resulted in the creation of a strong series of tests and workloads that can be translated to the other implementations. This will make it easier for us to conduct our final performance analysis.

3. Breakdown of steps for coming weeks

Interval	Deadline	Task	In-charge
Half-week 1	April 18th	Finish correct fine-grained implementation	Tra
	April 20th	Create starter code for lock-free implementation with data structures and locks	Matt
	April 20th	Generate performance graphs	Tra

		for fine-grained implementation and optimize	
Half-week 2	April 22nd	Avoid ABA problem: Implement basic operations using counter	Matt
	April 22nd	Avoid ABA problem: Implement basic operations using hazard pointers	Tra
	April 24rd	Refine lock-free implementation and test for correctness	Tra and Matt
Half-week 3	April 28th	Finish a correct lock-free implementation	Tra and Matt
Half-week 4	April 30th	Performance analysis and optimization of lock-free implementation	Matt
	April 30th	Diversify workload set for performance analysis using real-world use cases	Tra
Half-week 5	May 3rd	Generate all graphs and results of performance analysis for final report	Tra and Matt
	May 4th	Work on final report: find explanation of the results using course concepts; create compelling visuals	Tra and Matt
	May 5th	Final report write-up	Tra and Matt

4. Concerns

- Our main concern at the moment is Tra's time issues, as she did not allocate enough time to work on the project due to a heavy workload during Carnival week. However, she is planning to commit more time to the project now that she has completed much of her other coursework.
- We also still have not started coding the lock-free implementation, which is the main technical challenge of this project. However, we have researched a

compare-and-swap approach, which we are planning on implementing. To ensure that we have enough time, Matt will be starting early on the lock-free approach while Tra wraps up the fine-grained locking implementation.

III. Poster session plan

- Speed-up graphs of all implementations on various workloads
- Visuals and graphs explaining the different workloads' results

IV. Preliminary results

Coarse-grained implementation running 100,000 operations. All operations perform worse with multiple threads. This makes sense, as only one thread can modify the tree at a time in the coarse-grained lock implementation. As the number of threads increases, contention for the lock increases, causing threads to spend more time waiting for access. However, beyond a certain point, adding more threads does not significantly degrade performance further because the bottleneck is the lock itself rather than the number of threads. The parallel performance of the search function undergoes the least decrease, which is likely due to the implementation of concurrent reading.: