Milestone Report

Group members:
Jiachun Li <jiachunl>
Jianan Ji <jji2>

Summary of Work Completed So Far

We have built the core infrastructure of the library, including parts of the domain-specific language (DSL) interface and a working backend for dynamic programming problems. We implemented both the Longest Increasing Subsequence (LIS) and Longest Common Subsequence (LCS) in the DSL, and verified correctness against standard test cases.

We also completed the implementation of the Cordon Algorithm as the primary parallelization strategy and applied it to LIS and LCS. Preliminary testing has been done on CPU, and we are currently analyzing how well the parallelism scales with increasing core count and input size.

Status of Original Goals and Deliverables

PLAN TO ACHIEVE	STATUS
Develop a C++ library with a DSL that allows users to define common DP problems (LIS, LCS, and convex GLWS) in a declarative manner	Partly Finished
Implement the Cordon Algorithm as the primary parallelization strategy	Finished
Support multi-core CPU execution with good scalability	Ongoing
Demonstrate speedup on GHC and PSC for representative problems	Ongoing
Create comprehensive documentation and examples	Ongoing

HOPE TO ACHIEVE	STATUS
Support for more complex DP problems (optimal alphabetic tree, matrix chain multiplication)	Not Start

HOPE TO ACHIEVE	STATUS
Automatic detection of problem properties (e.g., decision monotonicity)	Not Start
GPU backend implementation	Not Start
Performance comparable to hand-optimized parallel implementations	Not Start
Support for irregular DP problems with non-uniform state spaces	Not Start

At this point, we still believe we can complete all the “plan to achieve” deliverables. Some of the “nice to have” goals, such as automatic detection of problem properties and GPU backend, may be delayed depending on how much tuning and debugging the CPU backend requires.

Original Schedule Status

Week 1 (March 25 - March 31)

We have finished all the schedules.

- Design DSL interface and core abstractions
- Implement sequential DP solver for LIS and LCS
- Set up testing infrastructure

Week 2 (April 1 - April 7)

We haven’t implemented prefix-doubling optimization as it is related to convex GLWS problem which we haven’t done yet.

- Implement basic Cordon Algorithm
- Add parallel execution for LIS

Week 3 (April 8 - April 14)

We haven’t implemented convex GLWS as it is another large problem and we want to first scale LIS and LCS problem first.

- Complete parallel LCS implementation
- Begin optimization and benchmarking

Updated Project Schedule

Week 4-1 (April 15 - April 17)

- LIS and LCS performance analysis (Jiachun Li, Jianan Ji)
- Implement convex GLWS problem (Jiachun Li)
- Add prefix-doubling optimization (Jianan Ji)

Week 4-2 (April 18 - April 21)

- Test GLWS performance (Jiachun Li)
- Further develop frontend DSL (Jianan Ji)
- Optimize memory usage and task granularity (Jiachun Li)

Week 5-1 (April 22 - April 24)

- Performance analysis and optimization (Jiachun Li)
- Complete documentation and examples (Jiachun Li, Jianan Ji)
- Complete frontend DSL (Jianan Ji)
- Initiate final report and poster (Jiachun Li, Jianan Ji)

Week 5-2 (April 25 - April 28)

- Performance analysis and optimization (Jianan Ji)
- Complete documentation and examples (Jiachun Li, Jianan Ji)
- Finish final report and poster (Jiachun Li, Jianan Ji)

Poster Session Plan

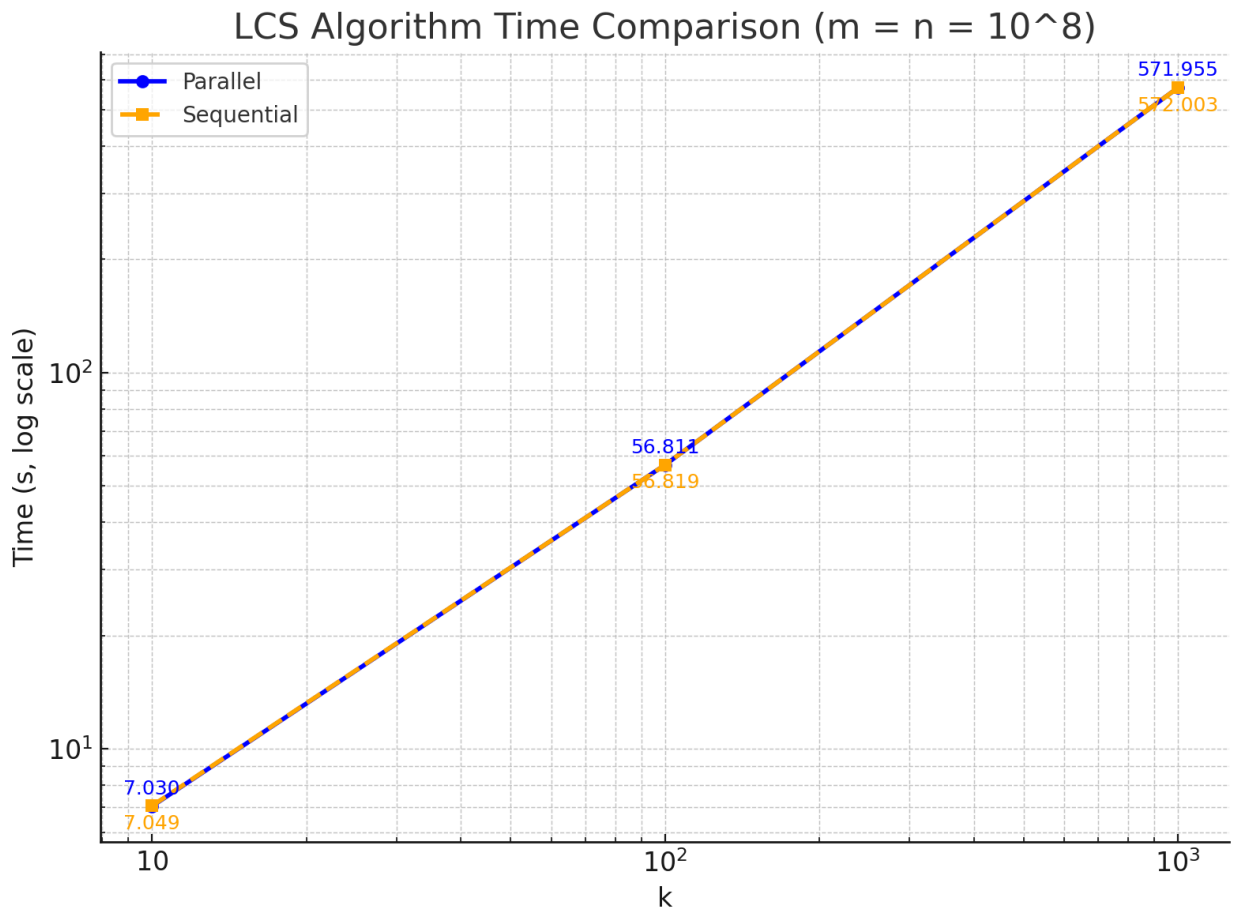
We plan to present the overall structure of our project, explain the methods we are using, and showcase performance acceleration graphs from various experiments. We will also bring a laptop to allow attendees to run tests on the spot and observe the results in real time.

Preliminary Results

Current result indicates there are still serious problems in our implementation since it's too slow than our expectation and the paper's result.

What's more, there is nearly no difference between sequential version and parallel version of our implementation.

We are doing this on PSC with two nodes(128 threads) and the main parallel operation is traverse from root to leaves on a segment tree to update values in parallel.



Current Issues and Concerns

Compare to the result from the paper, our implementation is too slow. The main differences are that we are using OpenMP with tasks pool to do parallel recursive traverse on the segment tree instead of Parlay, and we have different strategy of generating data. We will further dig out what is the reason.

Another concerns is the sheer diversity and complexity of subproblems within the realm of dynamic programming. Each subtype often has its own unique structure and optimization strategies. Parallel algorithms, in particular, require these highly specialized techniques to ensure that the total work remains efficient (i.e., comparable to or better than sequential solutions), rather than ending up with excessive overhead that negates any potential speedup. As a result, it is extremely difficult to find a unified solution that handles all DP problems efficiently under a single abstraction.

This leads to two major challenges:

1. The DSL needs to be highly flexible to support a wide range of problem patterns.
2. The backend solvers must be developed incrementally, gradually adding support for each new problem type.

Therefore, our design follows this pipeline:

Front-end DSL → Identify the algorithm type → Use the corresponding solver to solve the problem

Our primary focus will be on the solver component for different types of problems, as this is the critical part where parallel algorithms play the most important role.

For the scope of this project, we plan to focus on common problems like LIS (Longest Increasing Subsequence), LCS (Longest Common Subsequence), and GLWS (a generalized local window-based sequence problem). However, for the scope of a real open-source project, our design aims to be extensible. With time and contributions, we believe we can accumulate enough supported patterns and solvers to transform this into a fully usable DSL for dynamic programming.