




# Final Project

## ParallelDP: A Generic Library for Efficient Parallel Dynamic Programming

GitHub - NorthmanPKU/ParallelDP

Contribute to NorthmanPKU/ParallelDP development by creating an account on GitHub.

 <https://github.com/NorthmanPKU/ParallelDP>

NorthmanPKU/  
**ParallelDP**



 1 Contributor  0 Issues  1 Star  0 Forks



Group members:

Jiachun Li <jiachunl>

Jianan Ji <jji2>

## SUMMARY

We are going to develop a high-level parallel dynamic programming library (ParallelDP) on multi-core CPU platform. The library enables users to focus only on expressing dynamic programming (DP) problems via a simple domain-specific

language (DSL) while the backend will automatically parallelize computations using techniques such as the Cordon Algorithm.

## BACKGROUND

Dynamic programming (DP) is a fundamental algorithmic technique that solves problems by breaking them down into overlapping subproblems. Despite its wide applicability, efficiently parallelizing DP algorithms remains challenging due to dependencies between states. Recent research has shown that many DP problems can be parallelized efficiently using techniques like the Cordon Algorithm, which identifies sets of states that can be computed concurrently.

Many common DP problems share underlying structure:

- **States** represented as one or more dimensions (e.g., indices, positions)
- **Transitions** defining how states depend on other states
- **Optimization functions** (e.g., min, max) to select the best result
- **Base cases** as initial conditions

Traditionally, developers must manually implement both the problem definition and parallelization strategy, requiring expertise in parallel systems and specific DP optimizations. Our library will bridge this gap by providing a unified interface for defining DP problems and automatically applying appropriate parallelization techniques.

## THE CHALLENGE

Parallelizing DP algorithms presents several challenges:

1. **Dependency Management:** DP problems inherently have dependencies that limit parallelism. Identifying the maximum available parallelism requires sophisticated analysis of the dependency DAG.
2. **Problem Diversity:** DP problems exhibit variety in their structure (dimensions, transition patterns, optimization criteria), making a unified approach difficult.
3. **Decision Monotonicity and Other Optimizations:** Many efficient sequential DP algorithms use optimizations like decision monotonicity to reduce work. Preserving these optimizations while enabling parallelism is challenging.

4. **Work Efficiency:** Naive parallelization often sacrifices work efficiency. Achieving both high parallelism and optimal work bounds is non-trivial.
5. **DSL Design:** Creating an expressive yet simple DSL that can capture diverse DP problems while exposing all necessary information for optimization.

## RESOURCES

We will build our system using:

- C++ for the core implementation
- OpenMP for multi-threading on CPU
- The Cordon Algorithm and related techniques from recent research

We will rely on the paper "Parallel and (Nearly) Work-Efficient Dynamic Programming" and more for theoretical foundations and algorithms. For testing, we'll use standard DP benchmarks (LIS, LCS, GLWS, etc.) and high-core count machines available in the lab.

## GOALS AND DELIVERABLES

### PLAN TO ACHIEVE

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

Example usage (conceptual):

```
// Define an LIS problem
auto lis = DPPProblem<int>()
    .setStateSpace({0, n-1})
    .setBaseCase(0, 1)
    .setTransition(
```

```

ExprBuilder::init(StateVar::Result, 1)
.forLoop(StateVar::J, 0, Expr::lt(StateVar::J, StateVar::I))
.ifCondition(Expr::lt(ArrayRef(ArrayVar::Nums, StateVar::J), ArrayRef(ArrayVar::Nums, StateVar::I)))
.update(StateVar::Result, Expr::max(StateVar::Result, Expr::add(StateRef(ArrayVar::DP, StateVar::J), 1)))
.returnValue(StateVar::Result)
)

// Solve in parallel
auto solver = ParallelSolver();
auto results = solver.solve(lis);

```

## HOPE TO ACHIEVE

1. Support for more complex DP problems (optimal alphabetic tree, matrix chain multiplication)
2. Automatic detection of problem properties (e.g., decision monotonicity)
3. GPU backend implementation
4. Performance comparable to hand-optimized parallel implementations
5. Support for irregular DP problems with non-uniform state spaces

## PLATFORM CHOICE

Our library will be developed in C++ due to its high-level features and strong performance.

We will primarily target multi-core CPUs using C++ with OpenMP because:

1. Dynamic programming problems often have complex memory access patterns and dependencies that benefit from CPU's flexible cache hierarchy
2. C++ provides the necessary performance while allowing high-level abstractions
3. OpenMP offer simple yet powerful primitives for task-based parallelism
4. The Cordon Algorithm can be efficiently implemented using these frameworks

If time permits, we will extend to GPUs for problems with high arithmetic intensity and regular structure.

## **SCHEDULE**

### **Week 1 (March 25 - March 31)**

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

### **Week 2 (April 1 - April 7)**

- Implement basic Cordon Algorithm
- Add parallel execution for LIS
- Begin implementing prefix-doubling optimization

### **Week 3 (April 8 - April 14)**

- Complete parallel LCS implementation
- Implement convex GLWS
- Begin optimization and benchmarking

### **Week 4 (April 15 - April 21)**

- Add support for decision monotonicity detection
- Optimize memory usage and task granularity
- Implement more complex test cases

### **Week 5 (April 22 - April 28)**

- Performance analysis and optimization
- Complete documentation and examples
- Prepare final report and poster
- (Stretch) Begin GPU implementation if time permits

## **Additional Notes**

This project aims to bridge the gap between theoretical advances in parallel DP algorithms and practical systems that developers can use. By creating a high-level interface for defining DP problems while automatically handling parallelization, we hope to make efficient parallel DP accessible to a wider audience.