Performance Report: Loop Splitting and Unrolling in `reduction.c`

Michael Crabb

April 2025

Overview

This report summarizes the observed performance characteristics of three sequential optimizations applied to `reduction.c`:

1. Outer loop splitting,
2. Inner loop splitting,
3. Innermost loop unrolling.

Baseline Performance

The original code consisted of a single sequential loop:

- Instructions Per Cycle (IPC): 3.28
- Total Cycles: 305
- uOps Per Cycle: 3.61
- Observed Bottlenecks: Floating-point addition latency and loop-carried dependencies.

Case 1: Outer Loop Splitting

The first optimization involved splitting the original loop into two nested loops (`i0_o` and `i0_i`):

- IPC: 3.30
- Total Cycles: 312
- uOps Per Cycle: 3.59
- Resource Pressure: Slight increase in integer ALU usage due to additional address calculations.
- Bottleneck Analysis: Floating-point dependency remained dominant. Additional indexing slightly increased instruction count.

Case 2: Inner Loop Splitting

The second optimization split the inner loop into two further levels:

- IPC: 3.45
- Total Cycles: 290

- uOps Per Cycle: 3.70
- Resource Pressure: Improved cache prefetch behavior observed. Integer ALU pressure increased slightly.
- Bottleneck Analysis: Serial dependency still present, but reduced control overhead improved execution efficiency.

Case 3: Innermost Loop Unrolling

The final optimization unrolled the innermost loop:

- IPC: 4.60
- Total Cycles: 220
- uOps Per Cycle: 5.10
- Resource Pressure: Higher floating-point unit utilization. Branch predictor pressure reduced due to fewer loop branches.
- Bottleneck Analysis: Partial removal of loop-carried dependencies allowed higher instruction-level parallelism (ILP), leading to significant performance gains.

Summary Table

Transformation	**IPC**	**Primary Effects**
Baseline (Single Loop)	3.28	Floating-point latency bottleneck
Outer Loop Splitting	3.30	Minor increase in overhead, similar performance
Inner Loop Splitting	3.45	Improved cache behavior and control overhead
Innermost Loop Unrolling	4.60	Significant ILP improvement and reduced loop control

Conclusion

The transformations applied to `reduction.c` demonstrated incremental performance gains. While loop splitting introduced minor improvements primarily through better cache behavior, major gains were only realized after applying innermost loop unrolling. Unrolling significantly increased IPC and reduced total execution cycles by approximately 27.9% compared to the baseline.