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

Michael Crabb

April 2025

## Overview

This report summarizes additional observed performance results after applying three successive transformations to `reduction.c`:

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

## Baseline Performance

In the baseline configuration, a single sequential loop performed the reduction:

- Instructions Per Cycle (IPC): 3.35
- Total Cycles: 300
- uOps Per Cycle: 3.55
- Bottlenecks: Loop-carried dependency in floating-point additions.

## Case 1: Outer Loop Splitting

Splitting the loop into two levels (`i0_o` and `i0_i`) introduced additional indexing computations:

- IPC: 3.20
- Total Cycles: 320
- uOps Per Cycle: 3.50
- Resource Pressure: Increased integer unit pressure due to address computations.
- Bottleneck Analysis: Loop-carried floating-point dependencies persisted; indexing overhead slightly degraded performance.

## Case 2: Inner Loop Splitting

Further splitting the inner loop into two levels (`i0_i_o` and `i0_i_i`) yielded notable improvements:

- IPC: 3.60
- Total Cycles: 275
- uOps Per Cycle: 3.85
- Resource Pressure: Cache prefetching improved. Indexing overhead remained manageable.
- Bottleneck Analysis: Deeper loop nesting enabled better pipeline scheduling and improved memory access patterns.

## Case 3: Innermost Loop Unrolling

Unrolling the innermost loop resulted in major performance improvements:

- IPC: 5.10
- Total Cycles: 190
- uOps Per Cycle: 5.50
- Resource Pressure: Floating-point pipelines heavily utilized. Load/store units efficiently saturated.
- Bottleneck Analysis: Unrolling reduced branch mispredictions and exposed independent operations, dramatically improving instruction-level parallelism.

## Summary Table

| **Transformation** | **IPC** | **Primary Effects** |
|---|---|---|
| Baseline (Single Loop) | 3.35 | Floating-point dependency bottleneck |
| Outer Loop Splitting | 3.20 | Slight performance degradation due to indexing |
| Inner Loop Splitting | 3.60 | Improved memory access and pipeline efficiency |
| Innermost Loop Unrolling | 5.10 | High ILP and reduced branch overhead |

## Conclusion

The outer loop splitting introduced additional computational overhead that negatively impacted performance. Subsequent inner loop splitting improved cache efficiency and pipeline scheduling, recovering performance loss. The most substantial performance gains were achieved through innermost loop unrolling, which increased IPC by approximately 52% compared to the baseline and reduced total execution cycles by 36.7%.