Performance and Schedule-Aware Bottleneck Analysis of `reduction.c`

Michael Crabb

May 2025

1 Overview

This report examines the effects of three sequential optimizations on `reduction.c`, incorporating schedule considerations alongside traditional bottleneck metrics

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

2 Baseline Performance

The untransformed single loop achieves:

- **IPC:** 3.28
- **Total Cycles:** 305
- **uOps/Cycle:** 3.61
- **Observed Bottlenecks:** Floating-point addition latency and loop-carried dependencies.

No scheduling adjustments are applied at this stage.

3 Case 1: Outer Loop Splitting

Splitting the original loop into two nested loops (indices `i0_o` and `i0_i`) yields:

- **IPC:** 3.30
- **Total Cycles:** 312
- **uOps/Cycle:** 3.59
- **Resource Pressure:** Slight increase in integer ALU usage due to extra index calculations
- **Bottleneck:** Floating-point dependency remains dominant; scheduling impact is minimal.

4 Case 2: Inner Loop Splitting

Further splitting the inner loop into two levels delivers:

- **IPC:** 3.45
- **Total Cycles:** 290
- **uOps/Cycle:** 3.70
- **Resource Pressure:** Improved cache prefetch behavior; slight rise in integer ALU pressure
- **Bottleneck:** Serial dependency still present, but control-flow scheduling overhead is reduced.

5 Case 3: Innermost Loop Unrolling

Unrolling the innermost loop produces the most significant gains:

- **IPC:** 4.60
- **Total Cycles:** 220
- **uOps/Cycle:** 5.10
- **Resource Pressure:** Higher floating-point unit utilization; reduced branch predictor pressure
- **Bottleneck:** Partial alleviation of loop-carried dependencies allows greater ILP; schedule becomes more regular with fewer branch mispredictions.

6 Schedule Considerations

Introducing explicit scheduling analysis highlights:

- **Control-Flow Regularity:** Unrolled loops exhibit straighter-line code paths, improving static schedule predictability.
- **Latency Hiding:** Inner loop splitting and unrolling help hide floating-point latencies by overlapping independent iterations.
- **Branch Reduction:** Fewer loop-control branches reduce pipeline flushes, enhancing effective scheduling of micro-operations.

7 Summary of Transformations

Transformation	IPC	Total Cycles	Primary Effect
Baseline (Single Loop)	3.28	305	Floating-point latency
Outer Loop Splitting	3.30	312	Minor overhead, similar performance
Inner Loop Splitting	3.45	290	Improved cache behavior
Innermost Loop Unrolling	4.60	220	Significant ILP and schedule regularity

8 Conclusion

While loop splitting yields modest gains through improved cache prefetching and reduced control overhead, the most substantial performance improvement—and the greatest scheduling benefit—comes from innermost loop unrolling. This transformation boosts IPC by over 40%, cuts cycles by nearly 28%, and produces a more predictable execution schedule.