Performance Report: Loop Splitting and Unrolling in Matrix-Vector

Overview

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

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

Baseline Performance

The original code consisted of a outer and inner loop:

- Instructions Per Cycle (IPC): 2.41
- Total Cycles: 166
- uOps Per Cycle: 4.22
- 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: 2.94

- Total Cycles: 156
- uOps Per Cycle: 4.87
- 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. Loop-carried dependency still present

Case 2: Inner Loop Splitting

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

- IPC: 3.06
- Total Cycles: 171
- uOps Per Cycle: 4.79
- Resource Pressure: Improved cache prefetch behavior observed. Integer ALU pressure increased slightly.
- Bottleneck Analysis: Loop-carried dependency still present, but reduced control overhead improved execution efficiency.

Case 3: Innermost Loop Unrolling

The final optimization unrolled the innermost loop:

- IPC: 3.68
- Total Cycles: 122
- uOps Per Cycle: 5.32
- Resource Pressure: Higher floating-point unit utilization. Less index calculations
- Bottleneck Analysis: Removal of loop-carried dependencies allowed higher instruction-level parallelism

Conclusion

The schedule transformations that were applied to this program showed quite noticeable improvements compared to its baseline implementation. In the two cases of loop splitting, we see small gains in the number of operations performed per cycle due to the shorter index ranges for the split loops. We see the most improvement when unrolling the loop, as our main bottleneck in this program was the presence of a loop-carried dependency. This severely limited the performance of the program as each cycle was slowed down by this. By unrolling the loop, we are able to perform more operations in parallel since each loop iteration is now independent of its predecessor.