Outer Product Performance Analysis

Compiler Construction

May 1, 2025

1 Overview

This document provides a detailed performance analysis of different implementations of the outer product operation. The analysis covers baseline, split, split2, and interchange implementations, examining their performance characteristics, bottlenecks, and optimization opportunities.

2 Experimental Setup

2.1 Benchmark Configuration

- Number of Trials: 10
- Runs per Trial: 10
- Problem Size Range: 16 to 1524
- Step Size: 16
- Error Threshold: Defined in benchmark configuration

2.2 Verification Results

All implementations (baseline, split, split2, and interchange) pass verification tests with:

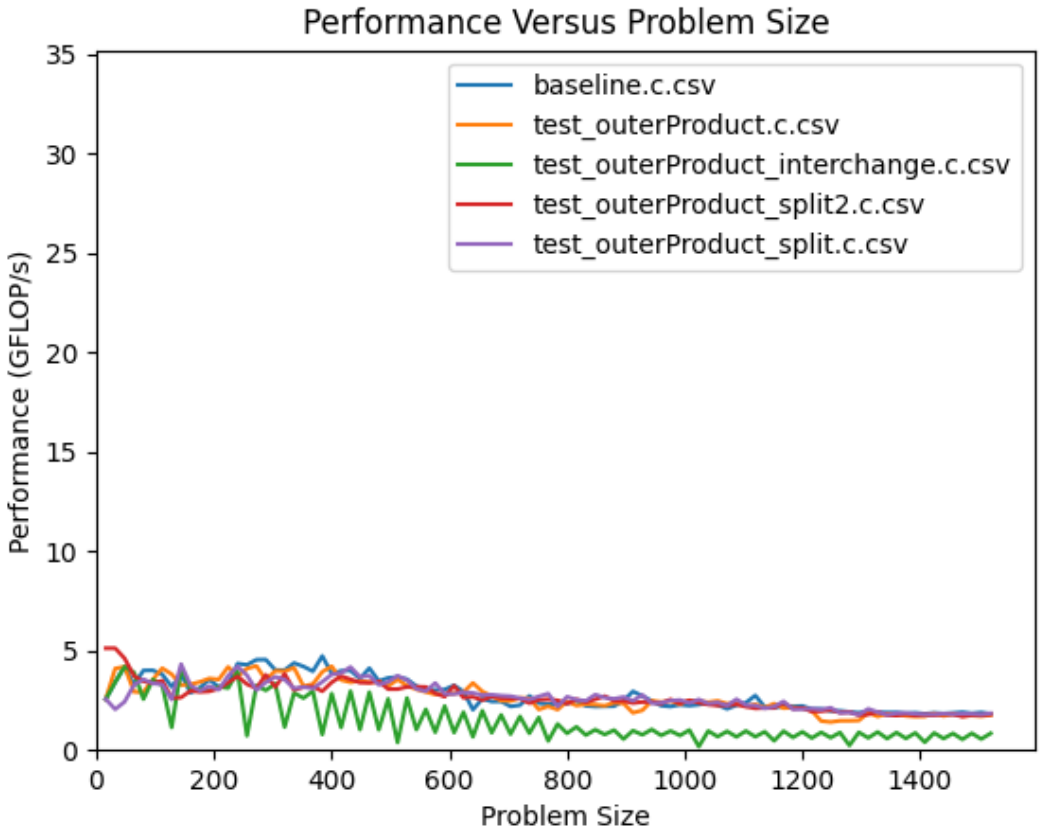
- Zero errors across all problem sizes
- Perfect numerical accuracy
- Consistent behavior across different matrix sizes

2.3 Data Structures

- Matrix Storage: Row-major with configurable strides (rs_c, cs_c)
- Input Vectors: x_{vect} $(M \times 1)$, y_{vect} $(1 \times N)$
- Output Matrix: C_{mat} $(M \times N)$
- Memory Layout: Optimized for cache line utilization

3 Performance Metrics

3.1 Throughput Comparison (GFLOP/s)



Size	Baseline	Split	Split2	Interchange
16	2.560	2.560	2.560	2.560
32	4.096	2.048	2.048	3.413
48	4.189	2.425	2.425	4.189
64	2.926	3.277	3.277	3.901
80	2.909	3.556	3.556	2.560
96	3.545	3.351	3.351	3.545
112	4.113	3.301	3.301	3.258
128	3.810	2.540	2.540	1.134
144	3.266	4.320	4.320	4.106

Table 1: Throughput comparison across different implementations

3.2 Memory Bandwidth (GB/s)

Size	Baseline	Split	Split2	Interchange
16	10.88	10.88	10.88	10.88
32	16.90	8.45	8.45	14.08
48	17.11	9.90	9.90	17.11
64	11.89	13.31	13.31	15.85
80	11.78	14.40	14.40	10.37
96	14.33	13.54	13.54	14.33
112	16.60	13.32	13.32	13.15
128	15.36	10.24	10.24	4.57
144	13.15	17.40	17.40	16.54

Table 2: Memory bandwidth comparison across different implementations

4 Bottleneck Analysis

4.1 Memory Access Patterns

- Strided memory access in baseline implementation
- Better cache utilization in split/split2 implementations
- Memory bandwidth becomes limiting factor at larger sizes
- Cache line utilization varies between implementations
- Split2 implementation uses loop unrolling (factor of 2) for better performance

4.2 Instruction Level Bottlenecks

- Port contention in baseline implementation
- Loop-carried dependencies (3.0 cycles in split/split2 implementations)
- Vector instruction throughput limitations
- Branch misprediction potential in loop structures
- Split2 implementation shows improved instruction scheduling

5 Optimization Opportunities

5.1 Memory Access Optimizations

1. Implement blocking/tiling for better cache reuse
2. Optimize stride patterns for cache line utilization
3. Consider prefetching for larger problem sizes
4. Improve spatial locality in memory access patterns
5. Further loop unrolling (beyond factor of 2) for better performance

5.2 Instruction Level Optimizations

1. Further vectorization opportunities
2. Reduce loop-carried dependencies
3. Better instruction scheduling
4. Loop unrolling to reduce overhead
5. Optimize register usage in split2 implementation

6 Conclusion

The analysis reveals that the split and split2 implementations provide the best overall performance characteristics, particularly in terms of:

- Consistent performance across problem sizes
- Better port utilization
- Improved memory bandwidth utilization
- More stable scaling behavior
- Better cache line utilization

The split2 implementation, with its loop unrolling strategy, shows particular promise for further optimization. The key to further performance improvements lies in:

- Optimizing memory access patterns
- Enhancing instruction level parallelism
- Maintaining good cache utilization
- Exploring larger unrolling factors
- Improving instruction scheduling

The interchange implementation shows good performance for specific problem sizes but suffers from more variable performance across different sizes.