Project 3: Vectorized Array Multiplication and Reduction using SSE

Joel Huffman huffmajo@oregonstate.edu

Tables and Graphs

Table 1: SSE and Non-SSE Performance Across Array Size

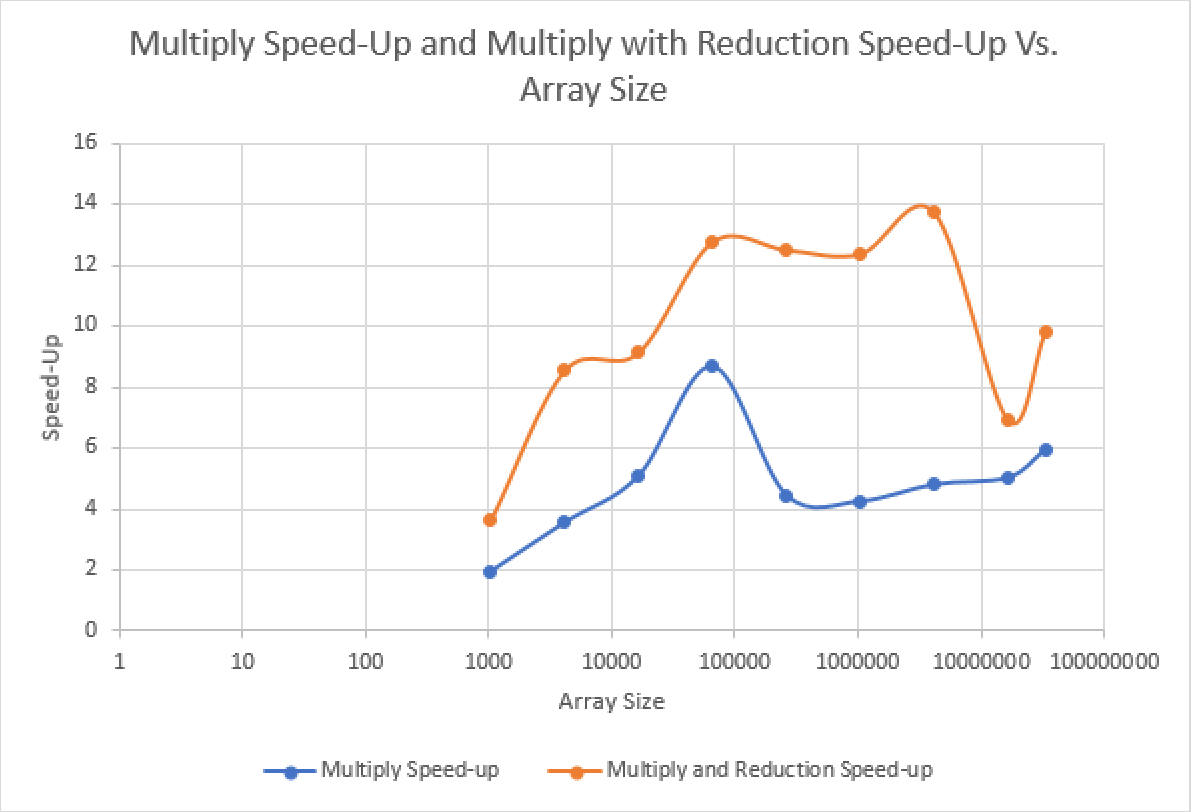
Array Size	SSE Multiply (MegaMults/sec)	Multiply (MegaMults/sec)	SSE Multiply and Reduction (MegaMults/sec)	Multiply and Reduction (MegaMults/sec)
1024	120.037187	61.568335	203.230674	55.874929
4096	328.93667	92.283621	652.836177	76.328708
16384	537.746017	105.321166	710.408877	77.689208
65536	965.276248	110.749716	1023.130948	80.108893
262144	930.24439	208.746773	1004.436099	80.296699
1048576	896.238444	210.414914	994.180628	80.332198
4194304	811.379615	168.487184	1743.844511	126.917169
16777216	1022.505782	202.808163	1001.375443	144.521349
33554432	1227.694931	206.018021	1463.317087	149.580162

Table 2: Speed-up Across Array Size

Array Size	Multiply Speed-up	Multiply and Reduction Speed-up
1024	1.94965784	3.637242635
4096	3.564410092	8.552957257
16384	5.105773487	9.144241463
65536	8.715834973	12.77175242

262144	4.45632944	12.50905842
1048576	4.2593865	12.37586737
4194304	4.815675565	13.74002056
16777216	5.041738788	6.928910157
33554432	5.95916282	9.782828601

Graph 1:



Explanation and Analysis

1. **What machine you ran this on**

All development, testing and benchmarking took place on OSU's flip3 server. I ran multiple benchmarks to check for consistently, but loads remained fairly high (>30) for the duration of my testing period.

2. **Show the table and graph**

Please see Table 1, Table 2 and Graph 1 above.

3. **What patterns are you seeing in the speedups?**

The speed-up for just array multiplication is mostly level around 4 with the exception of a single spike of 8.7. The multiplication with reduction speed-up remained pretty high and fluctuated between 7 to almost 14. Both curves had lower speed-ups at lower array sizes.

4. **Are they consistent across a variety of array sizes?**

The multiply speed-up was fairly concentrated around 4 with the exception of a spike near the middle. The multiply with reduction curve was less consistent, only remaining in a somewhat specified area (12-14) for four sequential array sizes.

5. **Why or why not, do you think?**

It makes sense that the multiply speed-up would remain around 4 as the SIMD function is utilizing 4 floats at a time with it’s SSE width. Our outlier value of 8.7 can probably be attributed to our heavy load while we were running the benchmark tests. The dip in speed-up of the multiply and reduction curve once we get into much larger datasets could be attributed to caching issues with that large of arrays, but our high-load runs could also be the cause of that unstable data.

6. **Knowing that SSE SIMD is 4-floats-at-a-time, why could you get a speed-up of < 4.0 or > 4.0 in the array-multiplication?**

It’s logical that SIMD’s 4-floats-at-a-time functionality would yield us a speed-up of right around 4 over a similar 1-float-at-a-time function. The SIMD function is able to process 4 calculations in the same time our non-SIMD function is able to process 1. Our speed-up values above 4 may be attributed to the inefficiency of our non-SIMD function and the high-load value of the flip server we were testing on.

7. **Knowing that SSE SIMD is 4-floats-at-a-time, why could you get a speed-up of < 4.0 or > 4.0 in the array-multiplication-reduction?**

A speed-up higher than 4 that we got for our multiply and reduction functions is completely reasonable as the difference in efficiency between our SIMD and non-SIMD functions is large. The SIMD function is significantly faster because it utilizes assembly code speed advantages (mostly the xmm register addition) over the easier usability of C. This coupled with our less efficient C code (due to it requiring more language compiling/conversion) causes a larger speed-up.