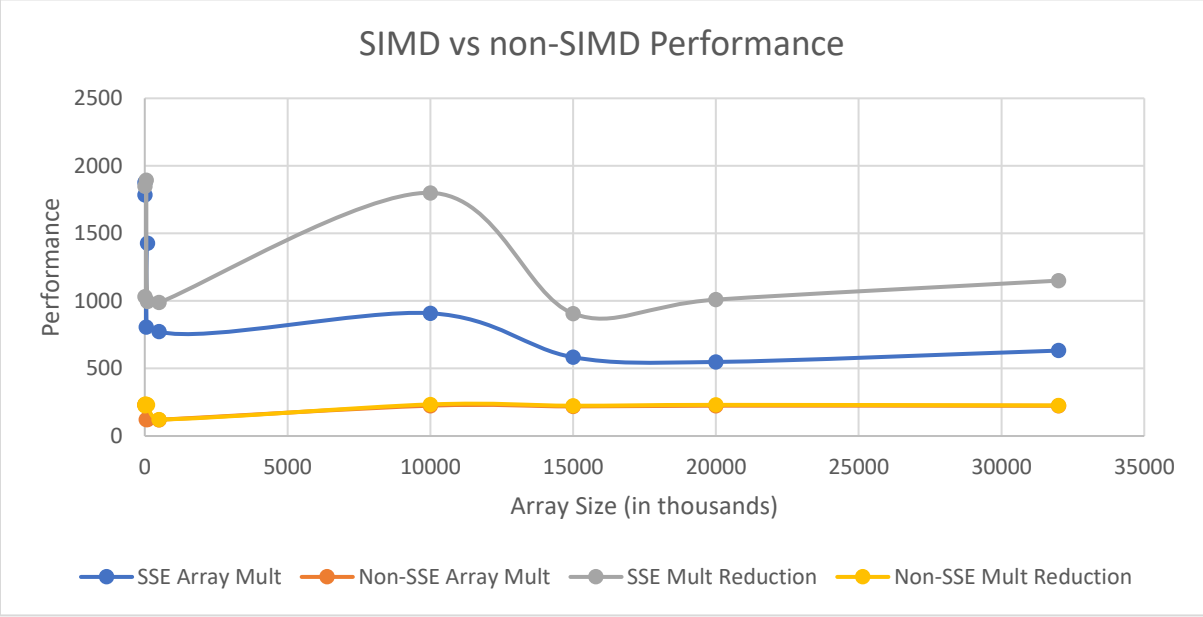
Erin Alltop
CS475 – Spring 2018
Project 5 Commentary

1. This was run on Flip1. More specifically:
flip1.engr.oregonstate.edu Dual Intel(R) Xeon(R) CPU X5650 (6 cores each with Hyperthreading, 24 total threads) 96GB RAM

2. I will show both the table and graphs for Performance and Speedup:

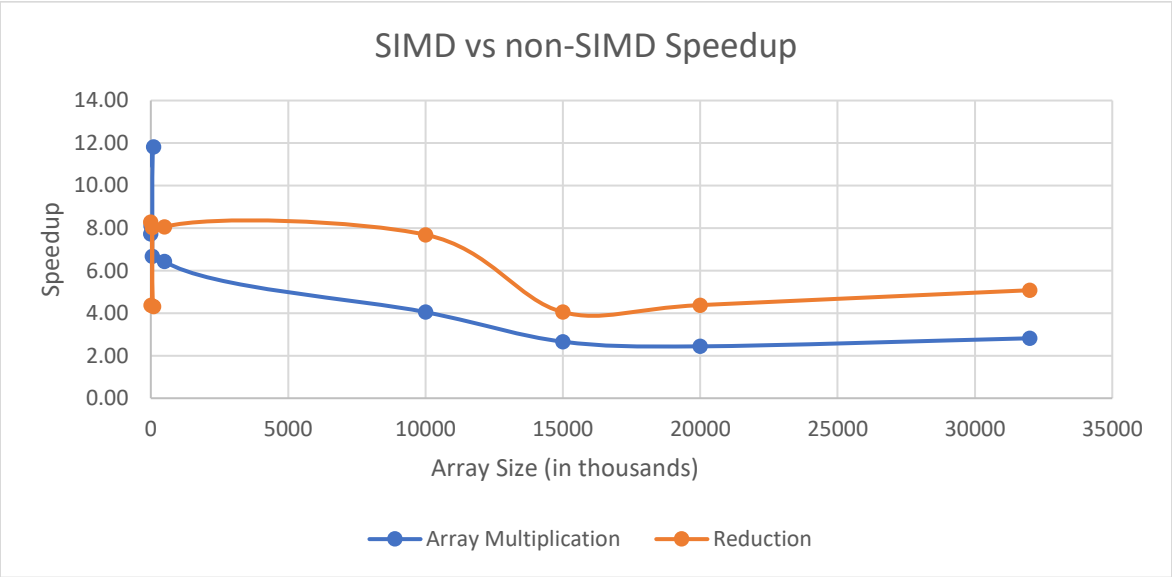
SIMD vs non-SIMD Performance

Size	SSE Array Mult	Non-SSE Array Mult	SSE Mult Reduction	Non-SSE Mult Reduction
1	1785.85	230.95	1848.89	222.96
10	1876.92	231.74	1030.49	236.08
50	806.1	120.88	1893.29	235.91
100	1427.37	120.74	996.97	230.73
500	772.38	120.03	987.69	122.59
10000	909.1	224.6	1800.27	233.96
15000	583.46	218.97	907.01	224.21
20000	548.15	224.45	1009.61	230.57
32000	632.33	223.94	1150.4	226.5



SIMD vs non-SIMD Speedup

SIZE	Array Multiplication	Reduction
1	7.73	8.29
10	8.10	4.37
50	6.67	8.03
100	11.82	4.32
500	6.43	8.06
10000	4.05	7.69
15000	2.66	4.05
20000	2.44	4.38
32000	2.82	5.08



3. The lower values vary quite a bit in terms of speedup. I ran several trials and they were somewhat unpredictable in the lower array sizes. As the array sizes increase however you can see the speedup increase and then dip for both Array Multiplication and Reduction around the 150000 mark. It then increases again slightly thereafter.

4. As stated above, the lower array sizes are all over the place in terms of speed up and there is very little consistency. Even with several trial runs, this inconsistency persisted. With higher array sizes the speedup becomes more consistent and after about the 150000 mark the increase is steady and consistent, though slight.

5. I think the speed up is inconsistent at lower array sizes because there just isn't enough data to have consistent results. The program flies through these arrays regardless of SIMD or non-SIMD so the speedup is not truly reflective of SIMD or non-SIMD. In larger array sizes it makes sense that the Speedup evens out because there is more data and time to do the performance efficiently and correctly.

The dip that is seen around 150000 also makes sense because the calculations are happening so quickly that temporal coherence is being violated (as we learned in lecture). Things are not being used multiple times so the cache line is reloading quickly and being reused.

6/7. I think both of these questions have similar answers. You could get a speed up of < 4.0 or > 4.0 in the array multiplication and the array multiplication-reduction because in the “regular” functions, memory might be stored in any given register which would influence how the data is accessed and what is cached or not. The data might be arbitrarily very easy to access and therefore the speedup is increased, or it might be farther/slower to access and therefore the speedup is decreased. SSE SIMD being 4-floats-at-a-time will give a more consistent speedup in comparison to the non-SSE functions. In the reduction function in particular, accessing an array that might be slower to access than the SSE SIMD function could slow down the speedup considerably. In my performance only data, I found that both the non-SSE functions performed almost exactly alike, while the SSE functions had a good amount of speedup.