Project 5

Brandon Lee

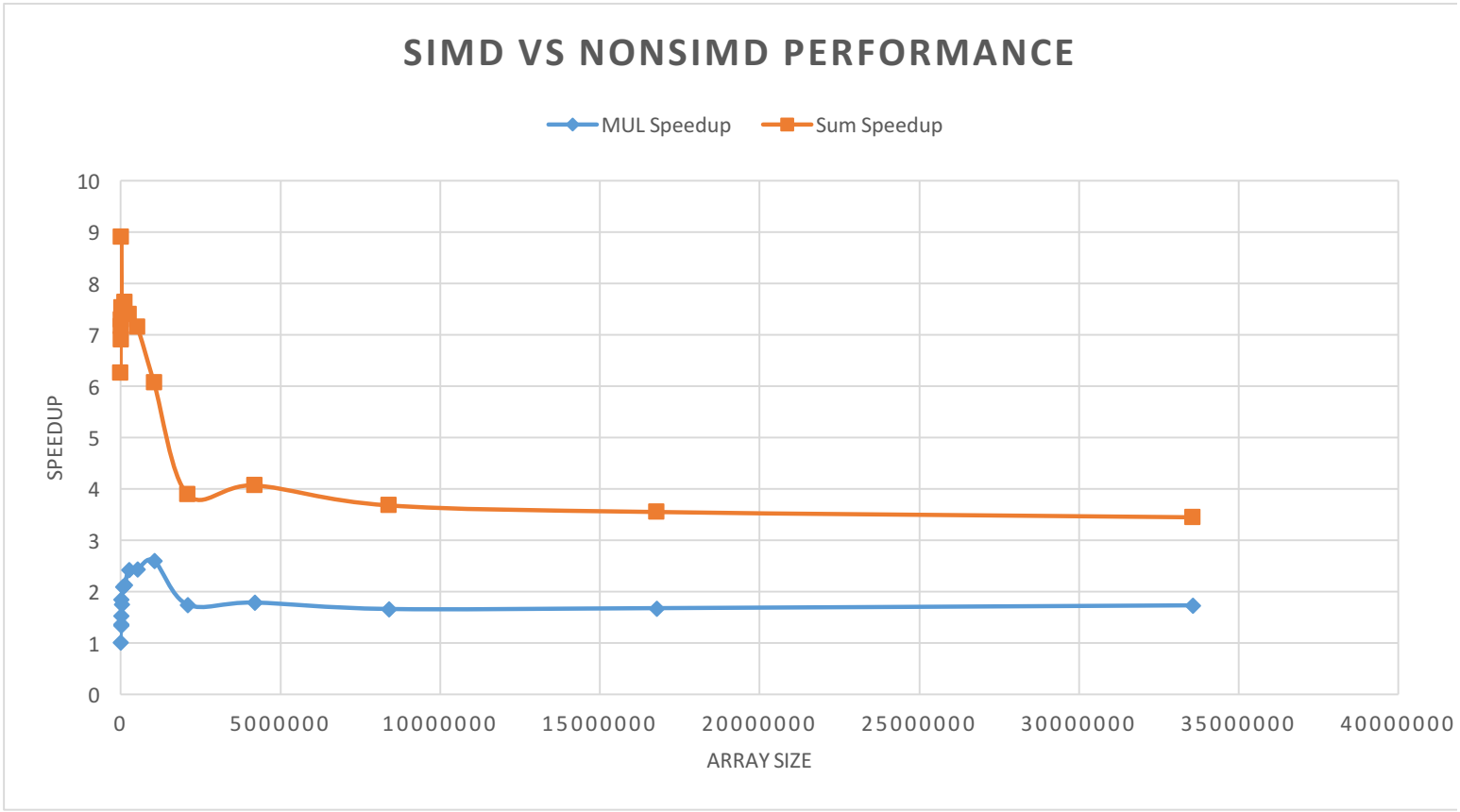
Machine Specifications

I ran this project on OSU's flip server through my 2015 MacBook Pro.

A table and graph

Array Size	MUL Speedup	Sum Speedup
1024	1.01161	6.26089
2048	1.36741	6.91392
4096	1.34289	7.17268
8192	1.84907	7.29495
16384	1.53532	8.91132
32768	1.75802	7.53479
65536	2.09899	7.5136
131072	2.13759	7.64434
262144	2.42937	7.40893
524288	2.43856	7.15618
1048576	2.60632	6.06921
2097152	1.75085	3.90027
4194304	1.79009	4.07515
8388608	1.66463	3.68771
16777216	1.68078	3.55941
33554432	1.73557	3.45639

*** 18:23:27 up 23 days, 26 min, 97 users, load average: 0.17, 0.47, 0.71



What patterns are you seeing in the speedups?

We observe that in both speedups, in the initial few data points, there's a steep ascent and decent in the speedups. This is followed by a rapid drop in speedup as we move towards the array size of 32K. Afterwards, the speedups appear to increase a bit, followed by a slow and steady plateau.

Are they consistent across a variety of array sizes? Why or why not, do you think?

From the graph above, we observe that the speedups are mostly consistent throughout a variety of graphs. However, towards the beginning of the graph, we can observe that the data is quite more inconsistent. I believe that this is due to the smallness of the arrays towards the beginning of the evaluation. Once we obtain larger arrays, we observe speedup becomes more consistent.

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?

Assembly code is very tight and it essentially wastes nothing. As a result, array multiplication code can perform can perform at least perfect 4.0 speedup or maybe even more.

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?

Summation could even be better than the array multiplication (see answer above for context). When this is done in C++, the sum is in a local variable and stored into memory, while the sum in assembly is stored into another xmm register, which is faster to store into.