Project 4: Vectorized Array Multiplication and Multiplication/Reduction Using SSE

By Caleb Knight

knightca@oregonstate.edu

Introduction

This is a technical write-up for observing the vectorization of Array multiplication and multiplication/reduction use SSE. SSE stands for Streaming SIMD Extensions. SIMD stands for Single Instruction Multiple Data. So, SIMD means that we can interact with multiple pieces of data simultaneously.
I ran these tests on my mid 2012 MacBook Pro. It has 16gb of RAM, with the 2.3 GHz Quad-Core Intel Core i7 processor.

Tables

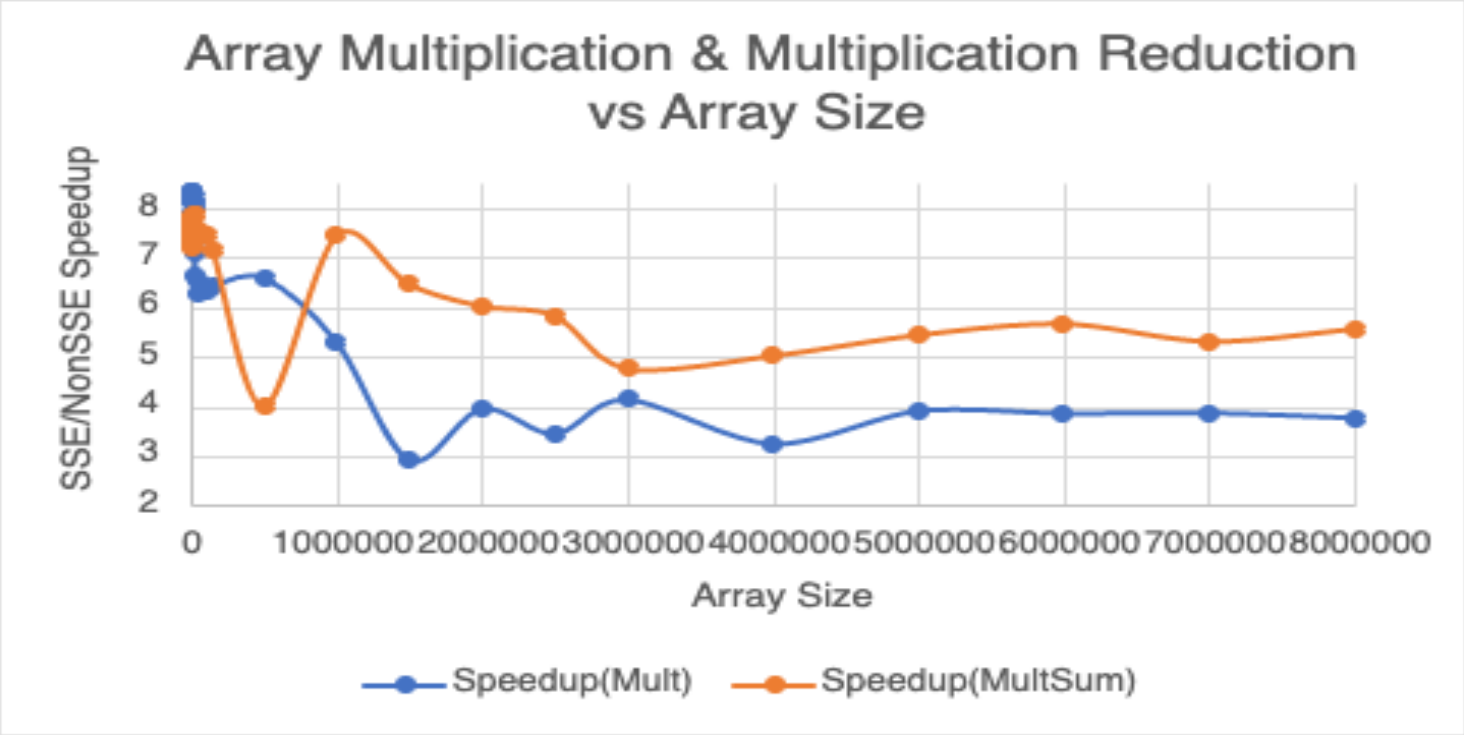
Regular Multiplication Both SSE and NonSSE

ARRAY SIZE	1k	2k	2.9k	3.9k	4k	4.3k	4.6k	4.9k	5k	5.5k	6k	6.5k	7.5k	10k	15k	15.5k	17.5k	18.5k	20k	22.5k	23.5k	25k	30k	50k	55k	75k	100k	150k	500k	1m	1.5m	2m	2.5m	3m	4m	5m	6m	7m	8m
Non SSE Mult	221.24	121.15	121.24	221.65	222.17	222.64	232.05	229.72	232.28	232.34	221.94	230.1	221.82	231.11	231.7	232.11	222.06	231.81	231.1	231.54	232.09	231.71	230.77	231.15	231.22	231.96	231	231.06	226.02	222.2	221.2	220.05	218.24	214.09	216.32	216.67	221.26	216.5	217.91
Mult	1815.29	886.57	995.87	1848.42	1855.28	1828.91	1831.88	1863.24	1831.54	1903.64	1845.58	1902.24	1837.34	1902.62	1899.48	1920.94	1790.29	1908.97	1847.93	1715.97	1651.78	169 8.47	1532.17	1445.34	1505.28	1490.14	1460.34	1484.37	1486.45	1176.19	644.71	869.3	748.65	884.15	699.04	844.85	851.98	835.7	819.67
Speed up (Mult)	8.2	7.32	8.21	8.34	8.35	8.21	7.89	8.11	7.89	8.19	8.32	8.27	8.28	8.23	8.2	8.28	8.06	8.24	8	7.41	7.12	7.33	6.64	6.25	6.51	6.42	6.32	6.42	6.58	5.29	2.91	3.9	3.43	4.13	3.2	3.9	3.85	3.86	3.76

Multiplication and Reduction Both SSE and NonSSE

ARRAY SIZE	1k	2k	2.9k	3.9k	4k	4.3k	4.6k	4.9k	5k	5.5k	6k	6.5k	7.5k	10k	15k	15.5k	17.5k	18.5k	22k	22.5k	23.5k	25k	30k	50k	55k	75k	100k	150k	500k	1m	1.5m	2m	2.5m	3m	4m	5m	6m	7m	8m
Non SSE Mult Sum	241.49	117.61	132.29	242.83	242.66	241.75	252.25	252.68	241.34	251.89	252.21	252.93	240.89	252.51	252.34	251.63	241.09	253.23	252.39	253.06	251.69	252.23	252.25	241.31	251.72	251.39	251.76	252.4	249.18	245.97	243.44	243.88	242.98	242.18	238.26	241.73	242.29	240.09	237.25
Mult Sum	1791.81	847.8	1030.56	1896.98	1877.99	1840.76	1870.56	1953.71	1784.51	1889.48	1826.48	1976.31	1816.41	1929.46	1884.42	1899.78	1819.9	1995.05	1919.2	1994.86	1899.43	1883.79	1833.52	1805.78	1884.27	1890.5	1875.15	1799.71	991.64	1830.25	1570.37	1465.35	1413.09	1149.48	1193.6	1311.66	1370.28	1272.36	1317.63
Speed up (Mult Sum)	7.42	7.21	7.79	7.81	7.74	7.61	7.42	7.73	7.39	7.5	7.24	7.81	7.54	7.64	7.47	7.55	7.55	7.88	7.6	7.88	7.55	7.47	7.27	7.48	7.49	7.52	7.45	7.13	3.98	7.44	6.45	6.01	5.82	4.75	5.01	5.43	5.66	5.3	5.55

Graph



Observations

As you can see, I had a lot of data to work with. I tried formatting these tables and my graph multiple different ways, and none of them seemed better than the others. With me having like 25 data points before I even hit 1 million on my X-axis, it's very hard to see that data. So below I include another graph that is 0 to 100k to look at that data a little closer.

From what I can see on the graph above, for both the multiplication and the multiplication and sum, their speedups are fairly high with small array sizes. The two speedups seem to mirror the opposite of one another. As we see near 3 million, they both converge between 4 and 5, and then go their separate ways. Not quite symmetrical, but a fairly close pattern.

For arrays smaller than 100k, we can look at the graph below and see that all of them fall between 6 and 8.5. For the most part, the smaller arrays do fluctuate quite a bit even on a small scaled graph. We see them start to flatten out after about 30k but then looking up to the graph above, we can see that there is a lot of fluctuation yet to come. At 500k, we see a large change. As we learned in the lecture, the set up time for smaller arrays is much faster, thus giving a

much better set up time. My guess is that at 500k, the set up time changes drastically from 150k (our previous data point). But then, after 500k, we see the multSum speedup increase and the mult speedup decrease more.

I think for the multSum, this is one of the optimal speedups. The array size is large which gives a lot of data, and can make up for the set up time. I think that after this point, we start to see it flatten out a bit because that is more of the true speedup time of this particular function. As for the mult speedup, it drops off pretty quickly here. My thoughts are that it could be due to memory issues, but also the instruction overhead is much greater here than it was at 150k or even 500k.

I think that with memory issues, as well as this overhead, we see a variety of speedups over the different array sizes. After 1 million, they still fluctuate, but much smoother. Like I mentioned before, I think this is the “true” speedup of the functions as we can see that quick set up times no longer help, and instruction overhead isn’t drastically killing us.