Project 5 Write Up

Andrew Johnson

CS475

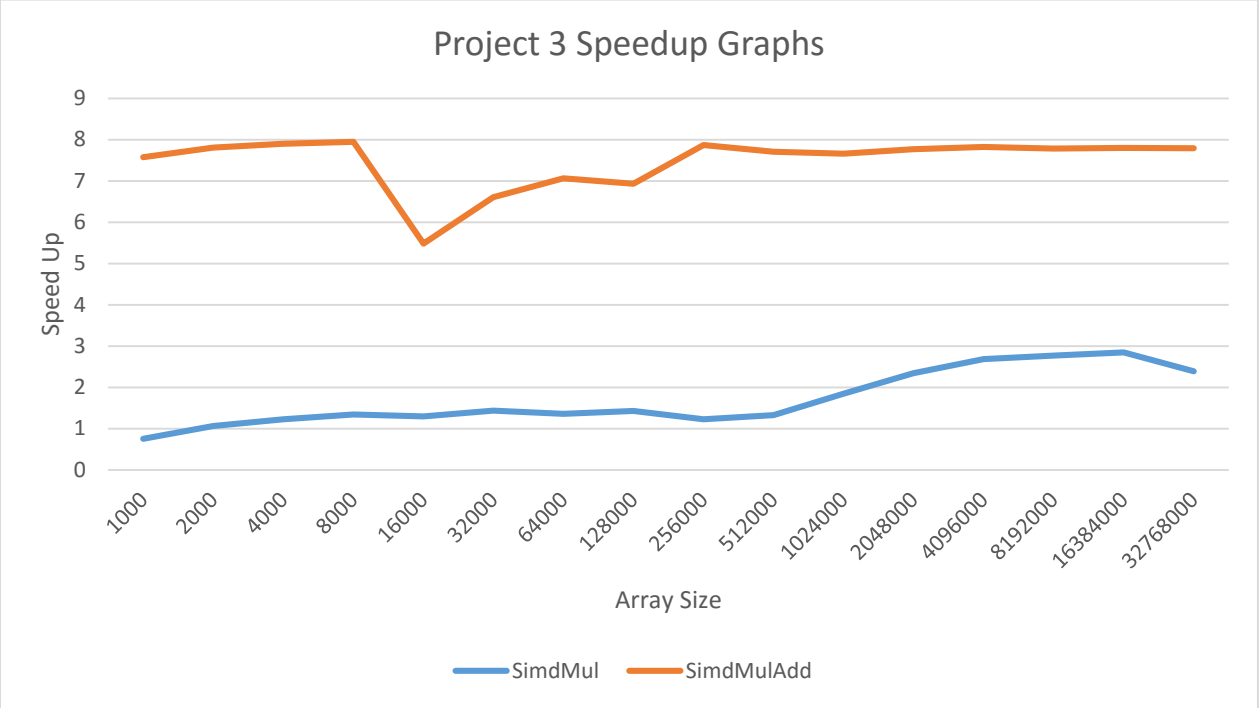
Commentary

I was starting to do this project on the terminal with the Kelly Engineering lab computers, but I found it better and easier to just use flip. Interesting enough I found you have to use double when using omp_get_wtime(), I was instead using float and it wasn't giving me proper numbers. Note taken for that. I also found it very annoying waiting for the program when using larger array numbers.

Table of Data from Project 5

NUM	SimdMul	SimdMulSum	1 / SpeedUp	
1000	1.325653	0.132049	0.754345	7.572946
2000	0.941841	0.128078	1.06175	7.807742
4000	0.817339	0.126536	1.223483	7.902889
8000	0.747497	0.125878	1.337798	7.9442
16000	0.774238	0.182394	1.291593	5.482636
32000	0.697965	0.151415	1.432737	6.604365
64000	0.73889	0.141559	1.353381	7.064192
128000	0.699687	0.144226	1.42921	6.933563
256000	0.815593	0.127035	1.226102	7.871846
512000	0.755118	0.129819	1.324296	7.703033
1024000	0.540862	0.130501	1.8489	7.662777
2048000	0.42739	0.128744	2.339783	7.767352
4096000	0.373125	0.127884	2.680067	7.819587
8192000	0.36118	0.128455	2.768703	7.784827
16384000	0.35147	0.128223	2.845193	7.798913
32768000	0.418579	0.128299	2.389035	7.794293

Graph of Data Shown Above



So I can see a that when using SimdMulAdd you get a significant more speed up of about 8, where for SimdMul, it doesn't get a good speedup until around an arraysize of a million. I can see a good constant line of what should be there for the speedups. For SimdMul, it stays around a speed up of 1 but gradually gets up to a speed up of around 3 when using very large numbers. I believe this increase of speed up is the program can benefit from setting up and the computation gets much faster. For SimdMulAdd, there is an obvious canyon that I don't believe should be on the graph. Although, I did further test with those arraysizes and it stayed constant. I can assume there should be around a speed up of 8 for almost all arraysizes. Even at 16000-128000 gets a very beneficial speed up. It seems like this would get a good speedup because the NonSimdMulAdd has a nested for loop so when we use SSE SIMD to split it up it creates smaller for loops thus cutting the runtime way down. That's where this high speed up comes in. For SimdMul it seems that set up hinders the speedup a lot, since SSE SIMD does the 4-floats at a time the set up to get to the for loops is almost as long as just going through the loop, so that's how it probably stays < 4.0. But with the reduction it cuts the time down making it get to computation faster, those getting its speed up > 4.0.

It seems that vectorizing arrays for computation can't hurt at all, it can only benefit. Especially when taking advantage of the reduction ability.