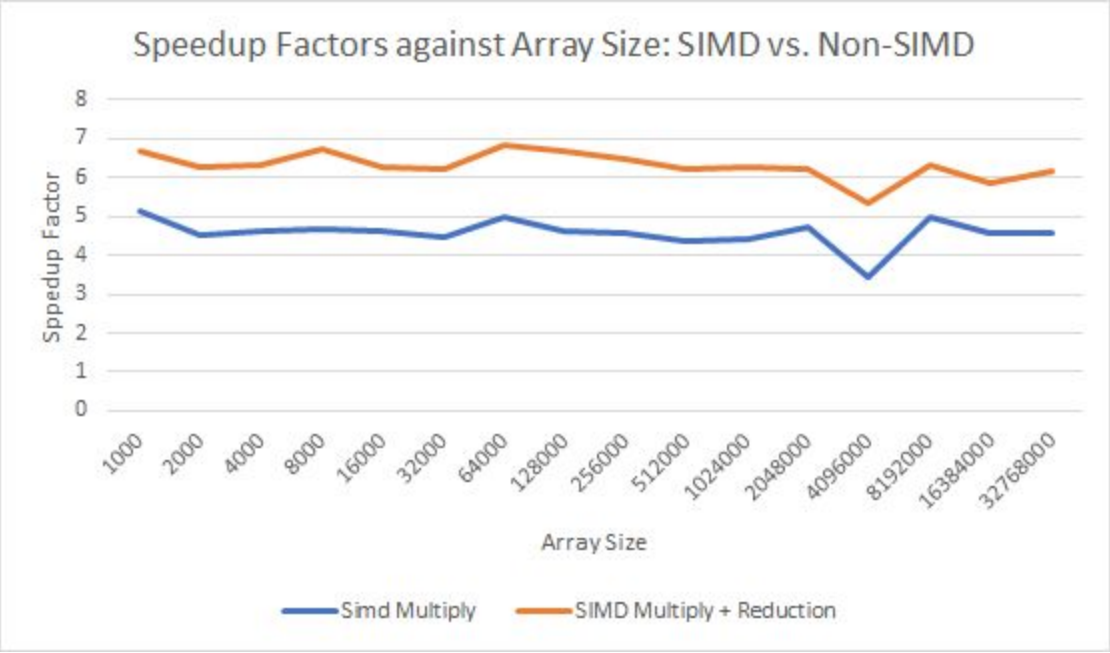
Brian Laccone
CS 475
5/13/2019

Project 4

1. This project was run on Flip 3.
2. Graph and Table:

Graph



Table

Size	SimdMul	Mul	SimdMul Sum	MulSum	Simd Multiply Speedup	SIMD Multiply + Reduction Speedup
1000	584.24	114.08	788.32	117.87	5.12	6.69
2000	518.24	114.24	735	117.27	4.54	6.27
4000	528.19	114.66	743.21	117.75	4.61	6.31

8000	530.85	114	788.75	117.02	4.66	6.74
16000	528.73	113.83	723.48	115.64	4.64	6.26
32000	508.59	114.29	725.77	116.36	4.45	6.24
64000	570.4	114.71	802.64	117.87	4.97	6.81
128000	528.9	114.49	790.08	118.01	4.62	6.7
256000	520.96	114.55	749.1	115.85	4.55	6.47
512000	525.23	120.74	745.65	119.92	4.35	6.22
1024000	532.77	120.15	769.08	122.83	4.43	6.26
2048000	523.87	110.87	721.57	116.39	4.72	6.2
4096000	668.74	194.54	1028.22	192.33	3.44	5.35
8192000	570.26	114.21	744.75	117.92	4.99	6.32
16384000	520.07	114.21	743.87	127.29	4.55	5.84
32768000	513.35	112.43	735.38	118.87	4.57	6.19

3. The biggest speedup pattern that I am observing in this project is that Multiply + reduction is always around 1-2 speedup factors higher than multiply. This speedup pattern is consistent from 1k to 32m array size. I believe that it is safe to say that pattern would continue into the hundreds of millions and beyond based on the consistency of this pattern. I also did multiple runs of this project on different server loads on Flip and this same pattern emerged throughout each run.

4. Yes, multiply was around 4 and multiply + reduction was around 5-7 for most of the array sizes. Multiply was at 5 when the array size was 1k but evened out to around 4 for almost the rest of the array sizes. Multiply took a hit at 4m array size when the speedup dropped to 3.44 but multiply + reduction also took a hit and went down to 5.35, so at least it stayed within the 1-2 speedup increase that I observed.

5. I believe that the data is relatively consistent because SIMD multiply is suppose to be faster that non SIMD and the multiply + reduction benefits from having the multiply speedup plus the

extra speedup from doing addition into the xmm2 register. This extra benefit is causing the multiply + reduction to be consistently higher than the multiply speedup.

6. The speedup of array multiplication could drop below 4 because the cache line could have to go get the memory. This could be solved with prefetching. The speedup of array multiplication could rise above 4 when the cache has the right amount of memory that the program needs to run the executions. Conversely, the speedups could be above or below 4 because of the way the non-SIMD code was created. If the non-SIMD code is slower than normal than a higher speed up could be achieved which could explain why my multiplication speedup is above 4.0 for most of the array sizes.

7. I didn't observe the array multiplication + reduction fall below 4 but the only thing I could think of the reason as to why is because a cache problem combined with very high numbers. I did however observe the array multiplication + reduction rise above 4 and I believe this is because it takes advantage of the addition into the xmm2 register instead of having to assign a value to a variable each iteration which is much more efficient that the non-SIMD code.