CS575(Introduction To Parallel Programming)
Project4

Project Title : Vectorized Array Multiplication/Reduction using SSE(Project 4)
Name : Si Thu Lin
ID : 933-957-884
Email : linsi@oregonstate.edu

The code was run on the flip.

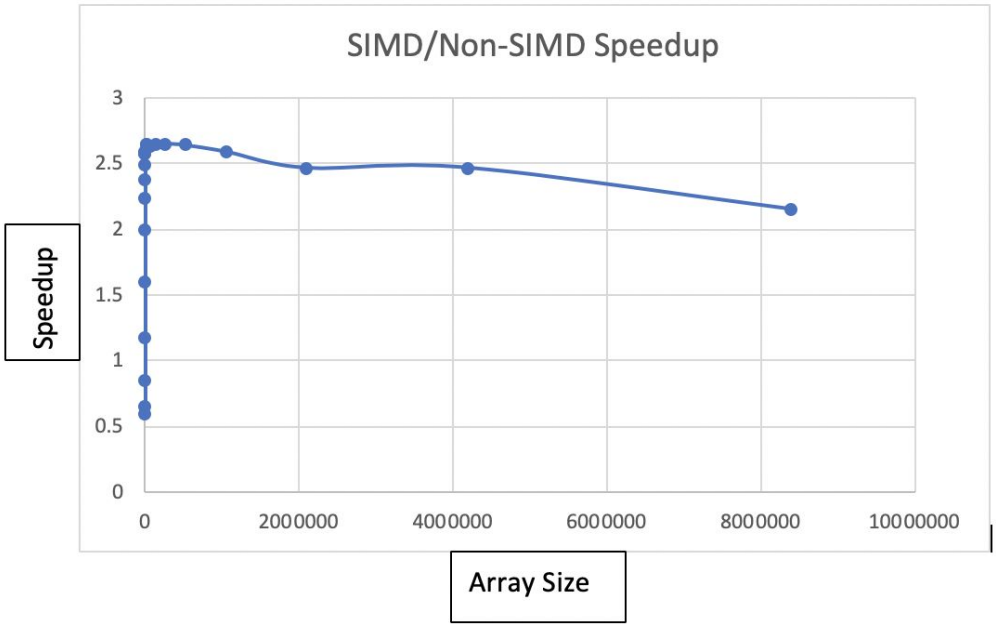
The Table of Performance is

	1 Thread	SIMD Alone	SIMD+2 Threads	SIMD+4 Threads	SIMD+8 Threads
2	54.36667463	32.29298719	1.53611134	0.93368854	0.67203369
4	99.88296037	65.57201979	1.98016012	1.67216947	1.18694688
8	148.1023206	125.400505	4.46114495	3.36596183	2.2161854
16	179.8939182	210.7959409	7.37650029	6.78777921	5.11001463
32	210.7959409	336.8601801	19.14724902	15.60742147	9.95645852
64	221.6757314	441.9258954	51.82464309	28.94670461	20.31617938
128	236.555858	527.5967504	62.74318807	60.58582917	38.95109918
256	242.5036674	576.86864	184.4199309	130.6143535	79.32984327
512	245.4267026	610.8397932	327.3330241	238.4023477	162.8664832
1024	243.875264	627.3960786	428.659504	407.4528915	350.6654849
2048	248.0427788	637.6752952	630.5443027	780.2104863	690.4311634
4096	247.150689	639.0187448	824.4533717	1088.760122	1141.164118
8192	243.5916096	643.5537769	1067.681377	1632.23103	1577.208718
16384	243.7620608	644.1782546	1167.945961	1969.237818	2082.161918
32768	244.1528027	643.8717557	1211.83344	2255.986925	2388.052539
65536	244.1439083	641.4389946	1232.485229	2373.353486	2601.047689
131072	243.5962474	643.0171647	1262.531013	2463.438765	2722.274115
262144	241.8869172	641.3198846	1270.368786	2515.123147	2788.703278
524288	240.5018234	634.6459008	1272.568519	2536.581859	2829.100379
1048576	237.9008001	615.9275737	1174.624908	2530.99856	2852.085368
2097152	236.9974442	584.5472195	1154.940599	2028.882442	2522.225237
4194304	237.0957124	585.1166592	1043.578278	1769.287301	2213.99257
8388608	233.5712838	503.0232528	989.6344762	1623.885655	1901.947105

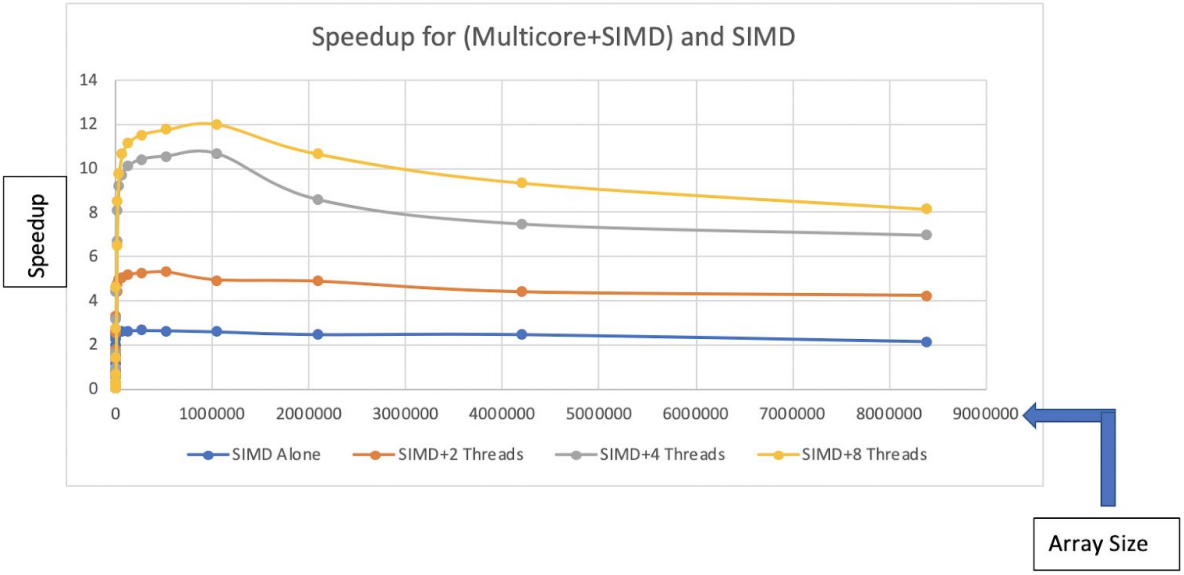
The Table of Speedup is

	SIMD Alone	SIMD+2 Threads	SIMD+4 Threads	SIMD+8 Threads
2	0.59398496	0.02825465	0.01717391	0.01236113
4	0.65648855	0.0198248	0.01674129	0.01188338
8	0.84671533	0.03012205	0.02272727	0.01496388
16	1.17177914	0.04100472	0.03773212	0.02840571
32	1.59803922	0.0908331	0.07404043	0.04723269
64	1.99356913	0.23378582	0.1305813	0.09164819
128	2.2303263	0.26523625	0.25611638	0.1646592
256	2.37880378	0.76048306	0.53860775	0.32712843
512	2.48888889	1.33373028	0.97137901	0.66360539
1024	2.57261056	1.75769981	1.67074301	1.43788869
2048	2.5708279	2.54207885	3.14546745	2.78351648
4096	2.58554304	3.33583279	4.40524817	4.61728075
8192	2.64193737	4.38307944	6.70068658	6.47480724
16384	2.64265182	4.7913361	8.07852465	8.54178009
32768	2.63716717	4.96342219	9.24006155	9.78097533
65536	2.62729879	5.04819161	9.72112515	10.65374806
131072	2.6396842	5.18288367	10.11279439	11.17535325
262144	2.65132109	5.25191193	10.39792964	11.52895456
524288	2.63884029	5.29130508	10.54703795	11.76332195
1048576	2.5890101	4.93745674	10.63888208	11.98854887
2097152	2.46647056	4.87321964	8.56077773	10.64241535
4194304	2.46785002	4.40150633	7.4623336	9.33796966
8388608	2.15361771	4.2369698	6.95241996	8.14289785

The Graph for Standard Credit



The Graph for the Extra Credit



Explaination

The speedup for the small array size is low because there is overhead and that overhead is dominant when the speedup for the small array size is calculated. I opine that there is no solution for this. Another fact which is noticeable in the graph is that the speedups drop at the particular array size(around 1000000 in the graph). The cause of that is that the calculation is done so fast that the next data to be used for calculation cannot be fetched in time. Another cause is the lack of “Temporal Coherence” because the calculation uses every element in the fetched array only one time. These can be solved by using “Prefetching”;