CS 475/575 -- Spring Quarter 2023

Project #4 - Vectorized Array Multiplication and Multiplication/Reduction using SSE.

Submitted by

Rahul Kumar Nalubandhu

Onid: nalubanr

Email : nalubanr@oregonstate.edu

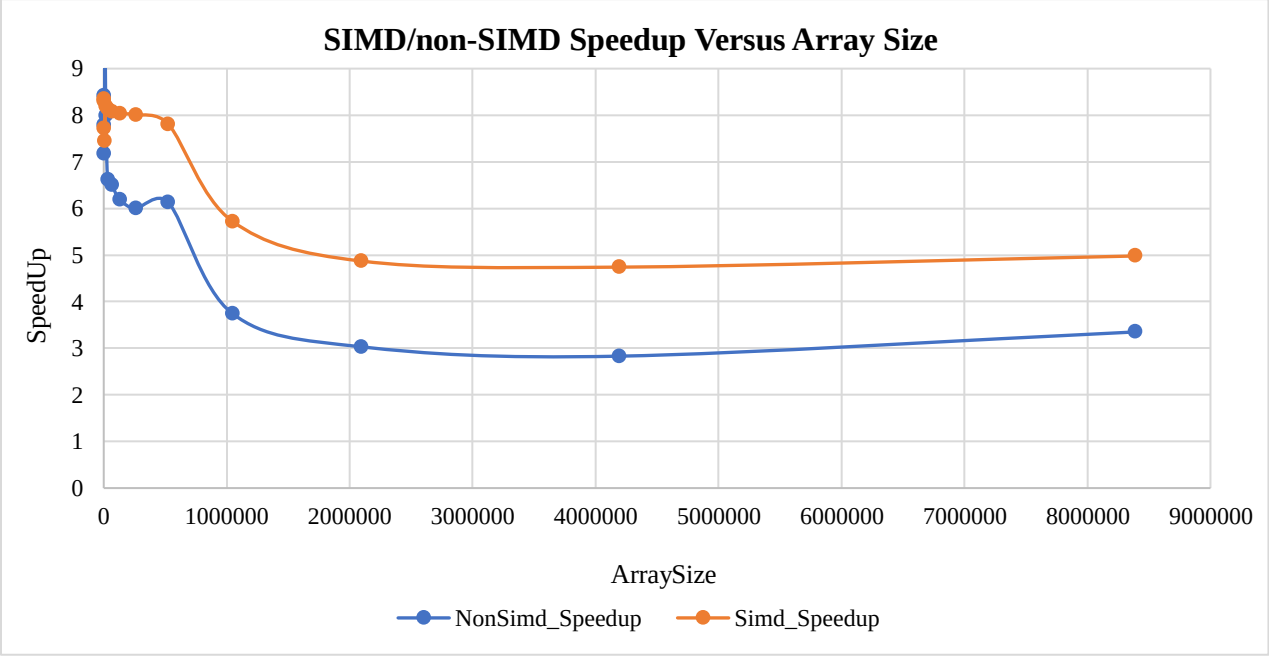
1. **What machine you ran this on?**

   I ran this project4 code on flip server.

2. **Show the 2 tables of performances for each array size and the corresponding speedups.**

ArraySize	NonSimd_megaMults	Simd_megaMults	NonSimd_Speedup	NonSimd_megaMultAdds	Simd_megaMultAdds	Simd_Speedup
1024	120.48	864.74	7.18	105.73	816.57	7.72
2048	121.24	1020.43	8.42	122.53	1017.95	8.31
4096	120.95	942.52	7.79	122.91	1026.56	8.35
8192	163.3	1741.54	10.66	171.23	1275.81	7.45
16384	120.85	965.41	7.99	123.26	1008.93	8.19
32768	121.01	800.96	6.62	123.24	1002.75	8.14
65536	120.94	785.75	6.5	123.19	994.02	8.07
131072	221.69	1371.5	6.19	225.84	1812.99	8.03
262144	219.9	1318.39	6	224.92	1801.71	8.01
524288	217.01	1330.97	6.13	224.44	1751.99	7.81
1048576	216.22	808.26	3.74	218.33	1248.92	5.72
2097152	212.39	643.62	3.03	219.81	1070.65	4.87
4194304	214.91	608.04	2.83	220.2	1044.68	4.74
8388608	214.86	720.15	3.35	220.52	1098.43	4.98

3. **Show the graphs (or graph) of SIMD/non-SIMD speedup versus array size (either one graph with two curves, or two graphs each with one curve).**



4. **What patterns are you seeing in the speedups?**

   For both SIMD and Non-SIMD speedups, the data starts with an array size of 1KB (1024) and goes up to a maximum size of 8MB (8388608).

   The maximum speedup value for SIMD is achieved at the array size of 1024 with a speedup of 7.72. For Non-SIMD, the highest speedup, 10.66, is seen at the array size of 8192.

   As the Array Size increases, the speedup of both SIMD and Non-SIMD generally decreases. However, the Non-SIMD speedup decreases at a faster rate than the SIMD speedup, indicating a higher efficiency of SIMD for larger arrays.

   The minimum speedup for SIMD is reached at the array size of 8388608 with a speedup of 4.98. For Non-SIMD, the lowest speedup of 2.83 occurs at the array size of 4194304. Beyond these points, up to the maximum array size of 8MB, the speedup for both SIMD and Non-SIMD appears to stabilize and remain at a relatively constant level. This suggests that while the efficiency of these operations decreases as array size increases, there is a limit to this decrease, after which the speedup remains relatively constant.

5. **Are they consistent across a variety of array sizes?**

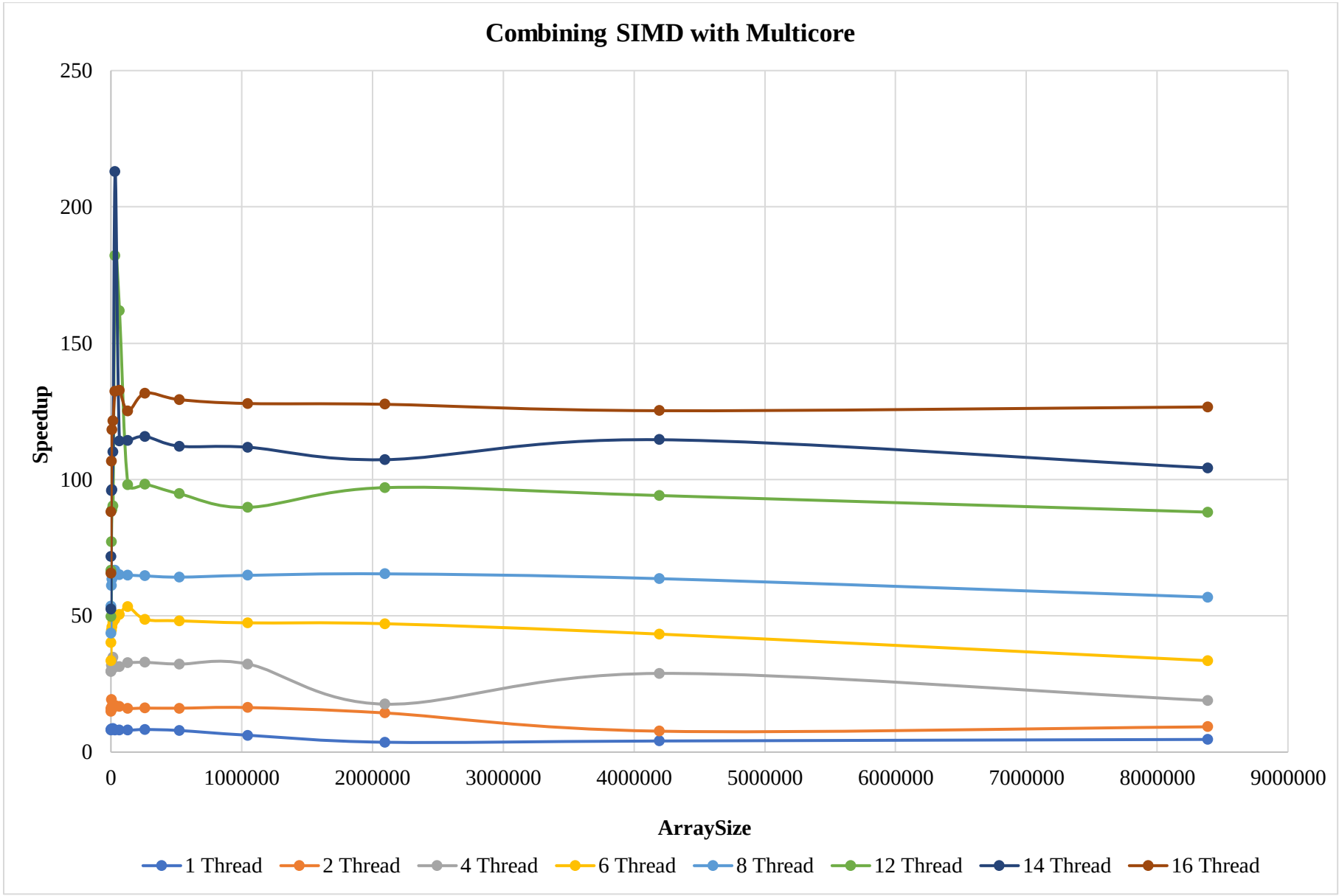
   From the provided data, we can see that the speedups for both SIMD and Non-SIMD are not exactly consistent across a variety of array sizes. As the array size increases, the speedups for both SIMD and Non-SIMD generally decrease. This indicates that the efficiency of both SIMD and Non-SIMD operations declines as the complexity of the task increases (i.e., as the array size gets larger). In the beginning of the graph, where the initial Array size is 1024, the speedup trend of both SIMD and non-SIMD is inconsistent. Later, it falls up to a level of 524288 and remains constant until the array's maximum size, 8388608, is reached.

6. **Why or why not, do you think?**

   SIMD is extremely fast, achieving a speedup of 8 or more when the array size is small. As the size grows, the value begins to fluctuate and decrease due to inefficiency. As a result of the inefficiency, memory fetching takes longer.

Extra Credit : I took the values of SIMD_ megaMultAdds_performance speed up values and the threads to generate the graph.

Array size/Threads	1 thread	2 threads	4 threads	6 threads	8 threads	12 threads	14 threads	16 threads
1024	8.03	14.92	29.57	33.48	43.63	49.8	52.53	65.7
2048	8.32	15.97	29.65	40.23	53.6	66.71	71.8	88.16
4096	8.38	19.21	31.87	44.65	61.09	77.18	95.96	106.77
8192	8.46	16.66	33.07	45.96	63.44	88.91	96.25	118.21
16384	8.56	16.2	34.86	47.56	66.12	90.3	110.15	121.47
32768	8.03	17.05	31.24	48.59	66.71	182.12	212.91	132.29
65536	8.09	16.7	31.38	50.55	65.15	162	114.16	132.77
131072	8.01	16.01	32.71	53.3	64.95	98.13	114.29	125.1
262144	8.28	16.14	32.91	48.71	64.63	98.27	115.76	131.67
524288	7.92	16.1	32.29	48.17	64.18	94.75	112.22	129.3
1048576	6.19	16.34	32.3	47.41	64.83	89.78	111.83	127.87
2097152	3.62	14.36	17.59	47.09	65.39	97.02	107.27	127.58
4194304	4.08	7.68	28.87	43.29	63.63	94.12	114.59	125.23
8388608	4.65	9.28	18.92	33.53	56.84	88.03	104.25	126.62


Combining SIMD with Multicore
250
200
150
100
50
0
Speedup
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000
9000000
ArraySize
1 Thread
2 Thread
4 Thread
6 Thread
8 Thread
12 Thread
14 Thread
16 Thread

The graphs show that combine SIMD and multicore can get large speedups which are better than the both the SIMD alone and multicore alone. When both the array size and the speed up for number of threads (NUMT) are varied, a few observations can be made:

As the number of threads increases up to a point, the performance improves, indicating efficient use of the CPU's multiple cores and SIMD capabilities. Each thread can handle its SIMD operation independently, allowing the system to compute multiple data points simultaneously.

Beyond a certain number of threads, the benefits start to plateau or even degrade. This could be due to the overhead of managing more threads and the contention for shared resources. Additionally, if the number of threads surpasses the total number of available SIMD lanes or CPU cores, the performance can start to decline due to necessary time-sharing of these hardware resources.

Up to a certain size, larger arrays appear to improve performance. This could be because larger arrays enable more data points to be processed simultaneously in SIMD operations. More threads can also be utilized more effectively with larger arrays.

As the array size becomes extremely large, there's a noticeable drop in performance. This could be because the data no longer fits into the CPU cache, leading to more frequent memory accesses which are slower than cache accesses. It could also be due to the limitations of the SIMD architecture for extremely large datasets.