Project #4 – Vectorized Array Multiplication and Multiplication/Reduction using SSE

Hyun-Taek Oh
ohhyun@oregonstate.edu
15 May 2024

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

   The program was conducted on a Predator HELIOS 300 (2022).

CPU:	12th Gen Intel(R) Core™ i9-12900H
Motherboard:	Mainboard PH315-55 Intel Ci912900H GN20-E6
Memory:	16GB DDR5 (8GB * 2)
Server:	rabbit.engr.oregonstate.edu

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

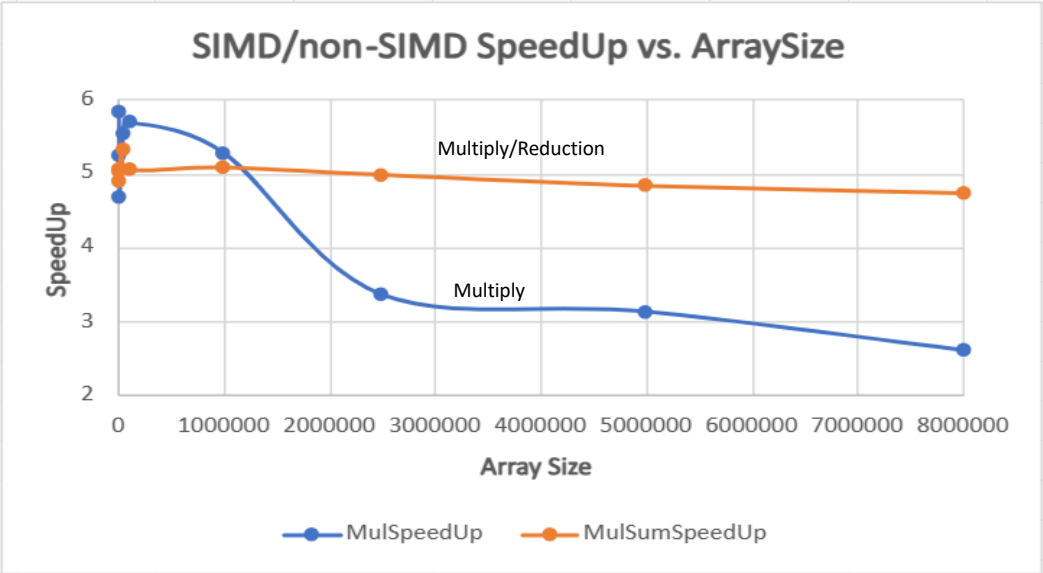
ArraySize	NonSimdMul	SimdMul	SpeedUp
1000	123.82	650.62	5.25
5000	123.88	723.8	5.84
10000	143.87	672.04	4.67
50000	181.25	1003.51	5.54
100000	205.56	1169.4	5.69
1000000	328.11	1733.81	5.28
2500000	323.64	1085.88	3.36
5000000	319.02	999.65	3.13
8000000	314.03	818.97	2.61

ArraySize	NonSimdMulSum	SimdMulSum	SpeedUp
1000	143.1	699.79	4.89
5000	143.83	725.48	5.04
10000	144.01	730.25	5.07
50000	221.64	1182.76	5.34
100000	354.62	1791.73	5.05
1000000	354.19	1800.42	5.08
2500000	353.73	1762.08	4.98
5000000	351.55	1696.41	4.83
8000000	348.58	1649.21	4.73

< **Figure 1.** The SpeedUp tables of Array Multiply and Multiply/Reduction >

The two tables above show the SpeedUp for Array Multiply and Multiply/Reduction. Unlike the table of Multiply/Reduction, when the array size is equal to 2.5M, 5M, and 8M, the SpeedUp for Array Multiply decreases rapidly. It may be affected by the condition of server, the time of calculation and searching array index, and so on.

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)**



< **Figure 2.** The graphs of the correlation between SpeedUp and Array Size >

Figure 2 displays the graph of the correlation between SpeedUp and Array Size. There are two lines which are the SpeedUp for Multiply and SpeedUp for Multiply/Reduction. As shown in the graph, the SpeedUp for Multiply is dramatically going down when the array size is more than 2.5M until being 2.61.

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

In the case of Multiply, when the array size is less than 1M, its SpeedUp is higher than one for Multiply/Reduction. However, when the array size grows up and reaches 8M, its SpeedUp decrease dramatically until being 2.61 whereas the SpeedUp, in the case for Multiply/Reduction, keeps steady state near 5, meaning that the case of Multiply/Reduction has better performance than the one of Multiply when processing big data set.

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

The SpeedUp for Multiply/Reduction is the only consistent case. Even the array size is various, its SpeedUp keeps steady state. However, the SpeedUp for Multiply reduces dramatically when the array size is larger than 1.5M, meaning that it has inconsistent SpeedUp.

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

In my opinion, using an array leads the program to search the index, which occurs cost of implement. On the other hand, using a variable can save the time and cost when the program does not have to search the index, just refer to the same address of that variable again.

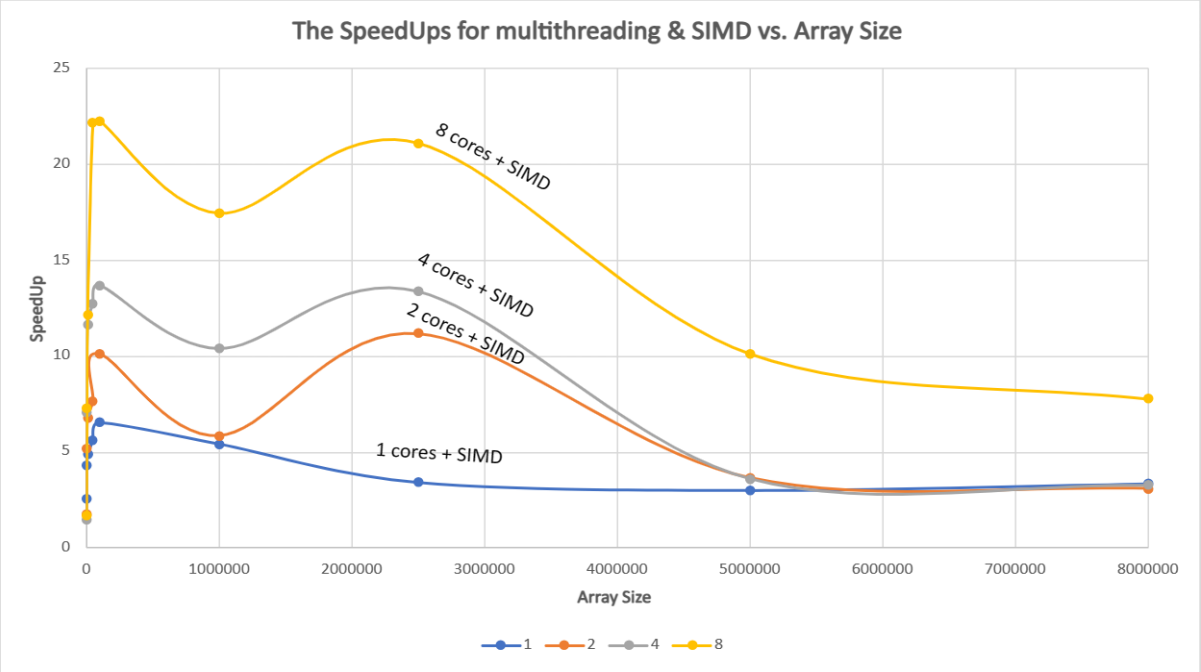
Likewise, in the Class Notes, prefetching can affect the SpeedUps indirectly since issuing the prefetch at the right time and right distance plays a significant role in hiding the latency of fetching from off-chip memory. If the cache lines uses Least Recently Used replacement policy, the latter case should be better than former case because former case should find next address of the array.

7. **[Extra Credit] Combining SIMD with OpenMP**

Combine multithreading and SIMD in a single test. In this case, you will vary both the array size and the number of threads (NUMT). Show your table of performances. Produce a graph similar to the one on Slied #21 of the SIMD Vector notes, using your numbers. Add a brief discussion of what your curves are showing and why you think it is working this way.

Array Size / NUMT	1000	5000	10000	50000	100000	1000000	2500000	5000000	8000000
1	2.54	4.31	4.83	5.58	6.5	5.39	3.38	2.96	3.3
2	1.75	5.15	6.74	7.64	10.09	5.82	11.15	3.63	3.07
4	1.42	7.03	11.59	12.72	13.68	10.37	13.35	3.58	3.23
8	1.63	7.29	12.12	22.15	22.26	17.43	21.09	10.09	7.73

< **Figure 3.** The SpeedUp table of multithreading and SIMD >



< **Figure 4.** The SpeedUp graphs of multithreading and SIMD >

As shown in Figure 3 and 4, the SpeedUps for multithreading and SIMD are larger than non-multithreading SIMD. The greatest value of multithreading and SIMD is 22.26 whereas the largest value of non-multithreading is less than 6. It means that mixing multithreading and SIMD is powerful tool to calculate and increase work performance.

However, when the array size become over 2.5M, all the graphs gradually decrease and keep the lower steady state. Especially, 8-cores case is even larger than non-multithreading SpeedUps.