Project#4

Vectorized Array Multiplication

And

Multiplication/Reduction using SSE

Woonki Kim

kimwoon@oregonstate.edu

15 May 2024

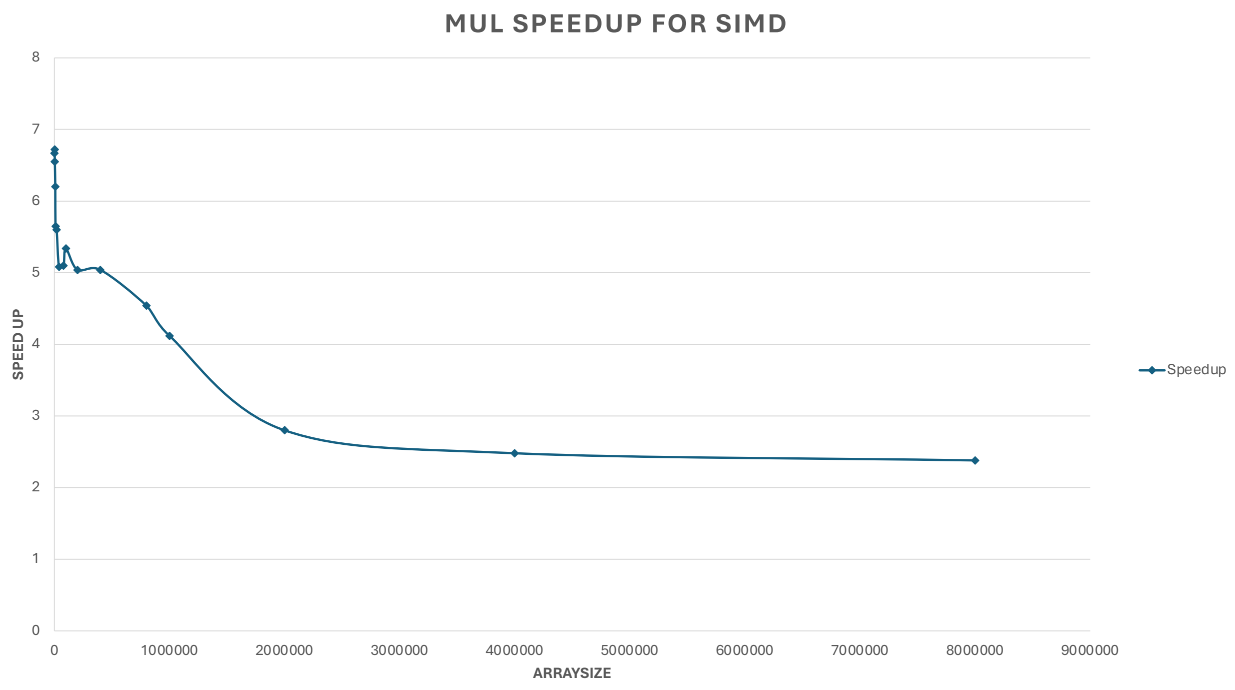
1. **Project Environment.**
    - Used FLIP2 server.
    - Ran on Linux OS.
    - Used g++ compiler.

2. **Speedup Data Table**

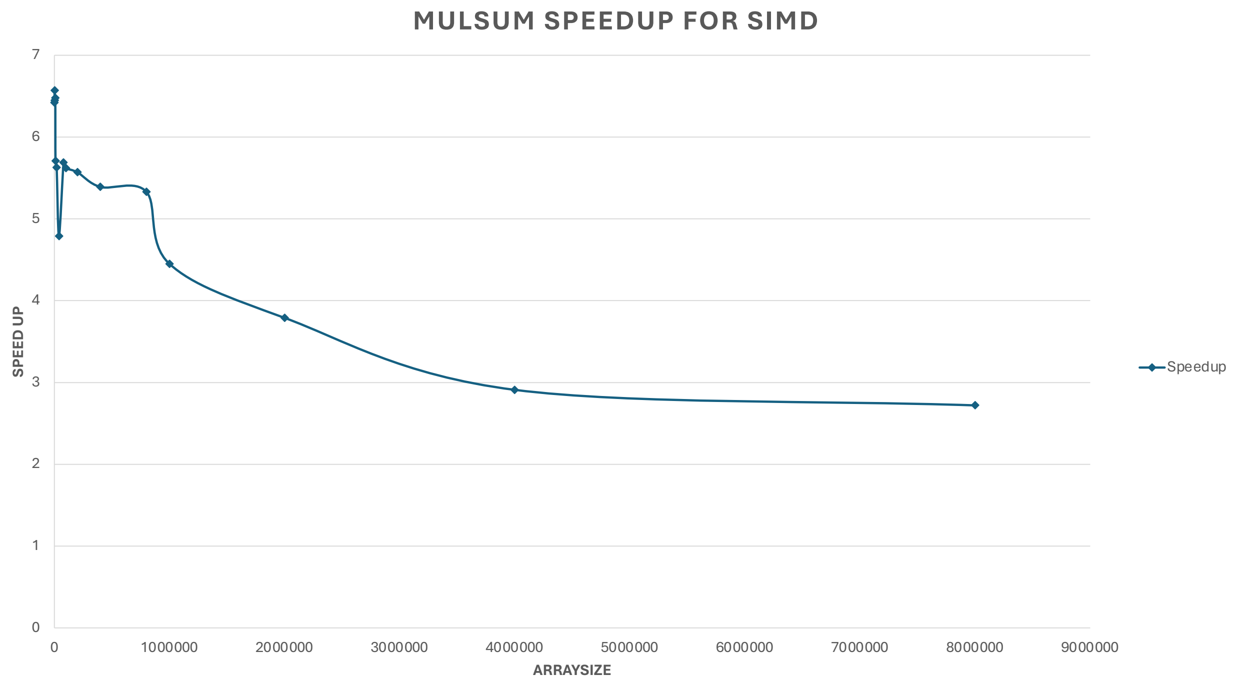
ArraySize	NonSimdMul	SimdMul	Speedup	NonSimdMulSum	SimdMulSum	Speedup
1000	298.76	1992.1	6.67	308.81	1981.07	6.42
2000	304.65	2047.17	6.72	313.68	2060.92	6.57
4000	305.04	1996.73	6.55	308.5	1988.41	6.45
8000	324.86	2013.11	6.2	313.2	2030.72	6.48
10000	359.8	2033.6	5.65	365.02	2084.13	5.71
20000	358.51	2009.25	5.6	365.06	2053.83	5.63
40000	359.1	1824.54	5.08	365.44	1751.48	4.79
80000	357.09	1819.48	5.1	365.54	2081.6	5.69
100000	357.08	1906.06	5.34	365.3	2052.69	5.62
200000	357.24	1800.49	5.04	365.04	2031.95	5.57
400000	354.66	1788.53	5.04	363.27	1958.49	5.39
800000	348.83	1584.93	4.54	361.65	1927.21	5.33
1000000	340.08	1399.43	4.12	357.58	1589.95	4.45
2000000	340.97	956.14	2.8	361.13	1369.8	3.79
4000000	320.83	794.35	2.48	347.74	1011.36	2.91
8000000	335.57	798.53	2.38	358.14	972.73	2.72

3. **Graphs**

- **Graph of Speed up for Multiplication using SIMD**



- **Graph of Speed up for Multiplication/Reduction using SIMD**

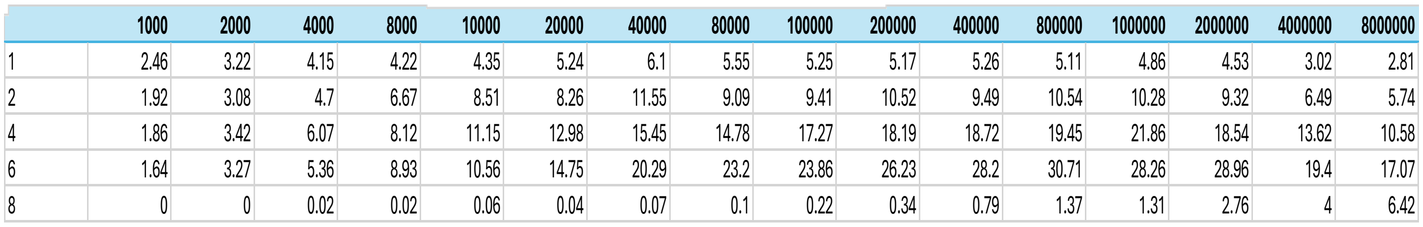


4. **Analysis of the Data & Graphs**

- **What patterns are you seeing in the speedups?**
  - The speedup patterns reveal that as the array size increases, the algorithm's ability to process array elements outpaces the rate at which the cache or memory bus can provide those elements. Up to an array size of 10,000 elements, the cache efficiently supplies elements for processing. However, between array sizes of 80,000 and 800,000 elements, there is only a slight decline in performance, with a speedup of around 3.5. Beyond 1,000,000 elements, the speedup drops below three, and as the array size approaches 8,000,000 elements, the remaining speedup becomes worthless.

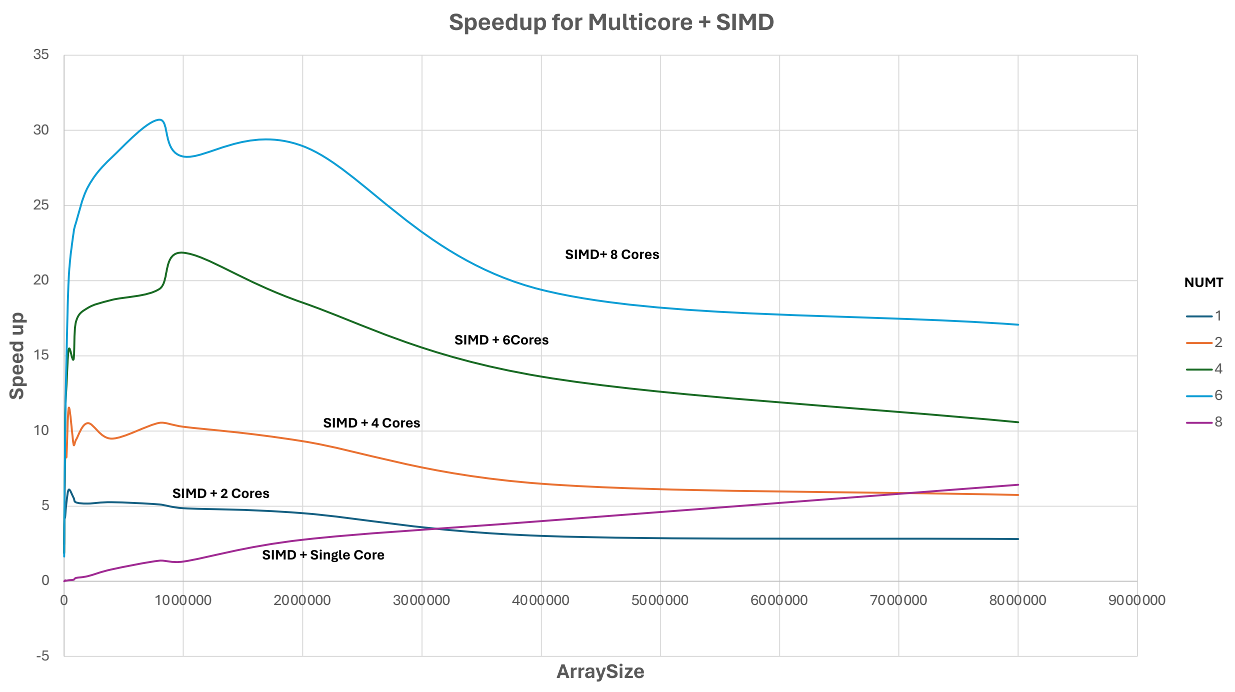
- **Are they consistent across a variety of array sizes?**
  - No, the speedups decline significantly as the array size exceeds 1,000,000 elements. While the algorithm's performance remains consistent for array sizes below 10,000 elements, there is a noticeable decline between 80,000 and 1,000,000 elements. Although the 4,000,000-element array still achieves a doubling of speed, by the time the array reaches 8,000,000 elements, there is almost no speedup left.
- **Why or why not, do you think?**
  - I believe that the algorithm's performance is hindered by the fact that it surpasses the cache's ability to provide elements for processing. As a result, the registers are forced to make round trips to RAM, negating the benefits of SIMD. To resolve this issue, the algorithm would greatly benefit from using prefetch, particularly if its primary purpose were to process arrays larger than 10,000 elements. Prefetching would likely enable the algorithm to maintain its 4x speedup throughout the entire range of array sizes tested.

5. **Extra credit.**
- **Speedup Data Table**



	1000	2000	4000	8000	10000	20000	40000	80000	100000	200000	400000	800000	1000000	2000000	4000000	8000000
1	2.46	3.22	4.15	4.22	4.35	5.24	6.1	5.55	5.25	5.17	5.26	5.11	4.86	4.53	3.02	2.81
2	1.92	3.08	4.7	6.67	8.51	8.26	11.55	9.09	9.41	10.52	9.49	10.54	10.28	9.32	6.49	5.74
4	1.86	3.42	6.07	8.12	11.15	12.98	15.45	14.78	17.27	18.19	18.72	19.45	21.86	18.54	13.62	10.58
6	1.64	3.27	5.36	8.93	10.56	14.75	20.29	23.2	23.86	26.23	28.2	30.71	28.26	28.96	19.4	17.07
8	0	0	0.02	0.02	0.06	0.04	0.07	0.1	0.22	0.34	0.79	1.37	1.31	2.76	4	6.42

- **Graph of Speed up for Multiplication/Reduction using SIMD + Multicore**



- **Analysis of the Graph**
  - The speedup curves indicate that in regions where the cache can sufficiently supply elements to keep the registers filled, it is possible to achieve close to a 16 times improvement by utilizing SSE SIMD with four-core multi-threading. However, both the single-core SIMD and multicore implementations suffer from diminishing returns as the array size exceeds 1,000,000 elements. This behavior persists because the cache fails to provide enough values for the algorithm to maintain efficiency.

  - The introduction of properly implemented prefetching would mitigate this issue, enabling the algorithm to maintain performance even with larger array sizes. Nevertheless, for array sizes below 1,000,000, this non-prefetch implementation remains highly efficient.