# CS 575 Parallel Programming

# Project #4

# Vectorized Array Multiplication and Multiplication/Reduction using SSE

**Name**: Rachel Xing (#934229189)

**Email Address**: xingru@oregonstate.edu

**Description**: This project tests array multiplication and multiplication/reduction using non-SIMD, SIMD, non-SIMD with multithreading, and SIMD with multithreading. Two non-SIMD functions written in C/C++ perform element-wise multiplication of two arrays of equal length. One stores the results in an output array (array multiplication), and the other computes the sum of the products (array multiplication/reduction). The SIMD versions of these two functions are written in assembly to fully achieve the performance gain by SIMD. Multithreading is handled using OpenMP. The experiments test array sizes from 4 up to 8 million, and multithreading is tested with 1, 2, and 4 threads.

## Commentary:

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

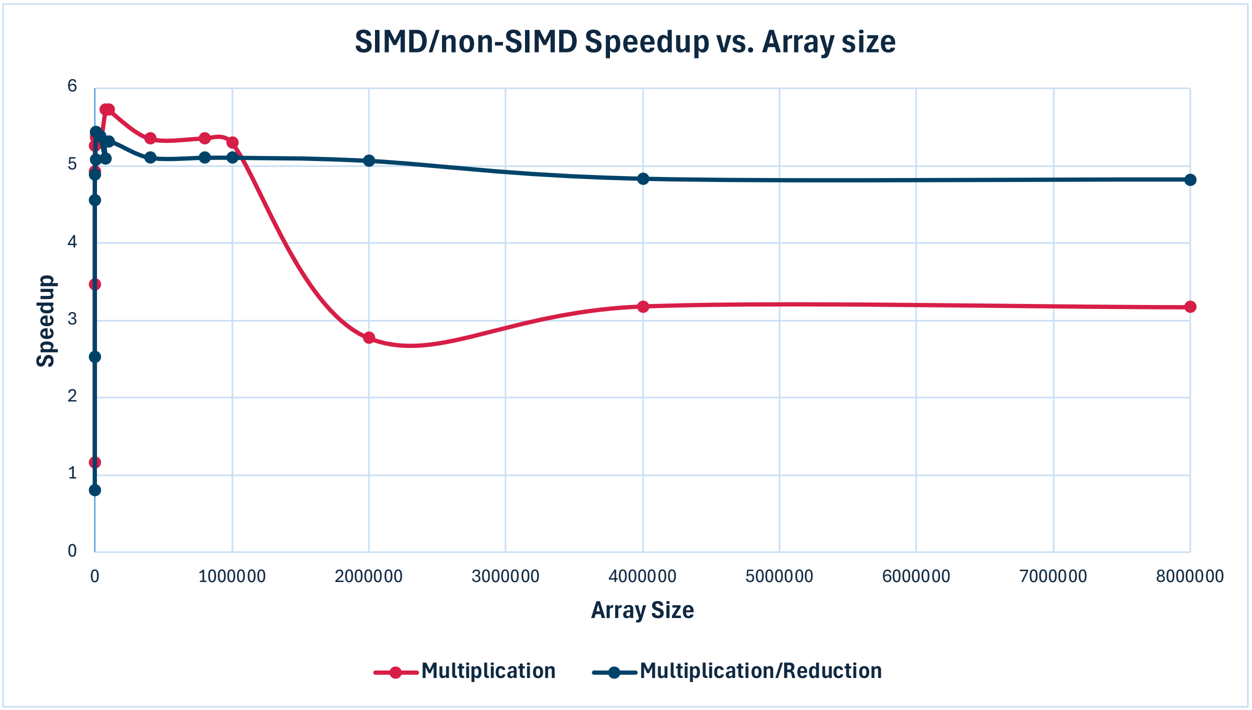
   The machine I ran the program on is the **OSU Rabbit server**, which is equipped with Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz and runs **CentOS Linux 7 (Core)**. The compiler I used is the **GNU C++ Compiler** (OpenMP enabled).

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

| **Array Multiplication C[i] = A[i] * B[i]** | | | |
|---|---|---|---|
| Array Size | Non-SIMD Performance (megamults/sec) | SIMD Performance (megamults/sec) | SIMD SpeedUp |
| 4 | 41.7 | 48.26 | 1.16 |
| 40 | 102.63 | 354.96 | 3.46 |
| 400 | 122.33 | 603.23 | 4.93 |
| 1000 | 123.76 | 649.38 | 5.25 |
| 4000 | 134.42 | 720.87 | 5.36 |
| 10000 | 132.77 | 719.69 | 5.42 |
| 40000 | 154.36 | 816.53 | 5.29 |
| **80000** | **174.91** | **1001.87** | **5.73** |
| 100000 | 184.92 | 1058.53 | 5.72 |
| 400000 | 328.17 | 1755.64 | 5.35 |
| 800000 | 327.97 | 1755.55 | 5.35 |
| 1000000 | 327.85 | 1736.44 | 5.3 |
| 2000000 | 325.62 | 900.86 | 2.77 |
| 4000000 | 320.82 | 1021.55 | 3.18 |
| 8000000 | 319.67 | 1012.26 | 3.17 |

| **Array Multiplication/Reduction sum = ΣA[i]*B[i]** | | | |
|---|---|---|---|
| Array Size | Non-SIMD Performance (megamults/sec) | SIMD Performance (megamults/sec) | SIMD SpeedUp |
| 4 | 40.14 | 32.54 | 0.81 |
| 40 | 105.27 | 266.77 | 2.53 |
| 400 | 130.11 | 592 | 4.55 |
| 1000 | 131.88 | 643.92 | 4.88 |
| 4000 | 143.8 | 729.94 | 5.08 |
| **10000** | **143.9** | **781.13** | **5.43** |
| 40000 | 188.37 | 1013.59 | 5.38 |
| 80000 | 354.68 | 1803.7 | 5.09 |
| 100000 | 254.93 | 1353.33 | 5.31 |
| 400000 | 354.12 | 1806.51 | 5.1 |
| 800000 | 353.9 | 1806.59 | 5.1 |
| 1000000 | 354.02 | 1806.35 | 5.1 |
| 2000000 | 353.76 | 1791.33 | 5.06 |
| 4000000 | 353.04 | 1704.35 | 4.83 |
| 8000000 | 352.51 | 1698.36 | 4.82 |

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?**

   From both experiments for array multiplication and array multiplication/reduction, we can observe an initial rapid growth in speedup as the array size grows from small to moderate. After reaching the peak speedup, it stays at a plateau for a while, and then gradually declines and stabilizes as the array size becomes very large.

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

   No, the speedups are not consistent across the range of array sizes from 4 to 8,000,000 for either multiplication or multiplication/reduction. Instead, they show distinct growth behaviors for small, moderate, and large array sizes.

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

   - *For small array sizes from 4 to 400, we can observe a rapid logarithmic-like growth of SIMD/non-SIMD speedup from ~1.0x to ~5.0x for both multiplication and combined multiplication/reduction operations.* It aligns with our expectations for SIMD, which can achieve a speedup of 4.0x by processing four groups of FP values in parallel compared to ordinary C code that handles one group of FP values at a time. As the array size increases, the overhead of SIMD becomes amortized, and the 4.0x speedup becomes more apparent and eventually dominant.

- *SIMD can achieve more than the expected 4.0x speedup because the assembly code is used*. C code stores local variables on the stack, while assembly allows us to manually store them in CPU registers, thus avoiding the overhead of accessing the stack. Such efficient allocation of registers can give additional performance gains for SIMD.
- *After reaching peak speedups for certain moderate array sizes (5.73x at an array size of 80,000 for multiplication and 5.43x at 10,000 for multiplication/reduction), the SIMD/non-SIMD speedup will stay at a plateau above 5.0x for a range*. This plateau occurs because caches and memory bandwidth become saturated as the array size increases. Thus, SIMD will not provide further performance gains.
- *As the array size becomes very large (around 1,000,000 for array multiplication and 2,000,000 for multiplication/reduction), memory bottlenecks become dominant, which further limits performance gains from SIMD or even diminishes the speedup*. This occurs for two reasons: (1) Violation of temporal coherence becomes more significant, leading to increased cache misses and slower memory access (2) The memory bandwidth further limits the rate of transferring data between the CPU and the main memory. However, this speedup degradation can be reduced by pre-fetching.
- *The speedup for array multiplication stabilizes around ~3.0x at an array size of 2,000,000, while the speedup for array multiplication/reduction stabilizes around ~4.8x at an array size of 4,000,000*. Such difference in final speedups occurs because array multiplication involves additional overhead from writing results to the third array, while array multiplication/reduction only accumulates results into a single SIMD register for the final sum. The latter has much less memory traffic by avoiding the additional fetches from and writes to memory when the array size exceeds the cache. Also, the array size to stabilize the speedup for array multiplication is about half that for array multiplication/reduction. That is because the former uses three arrays of equal size while the latter uses only two. Thus, the array multiplication fills the cache more quickly as the array size becomes very large.

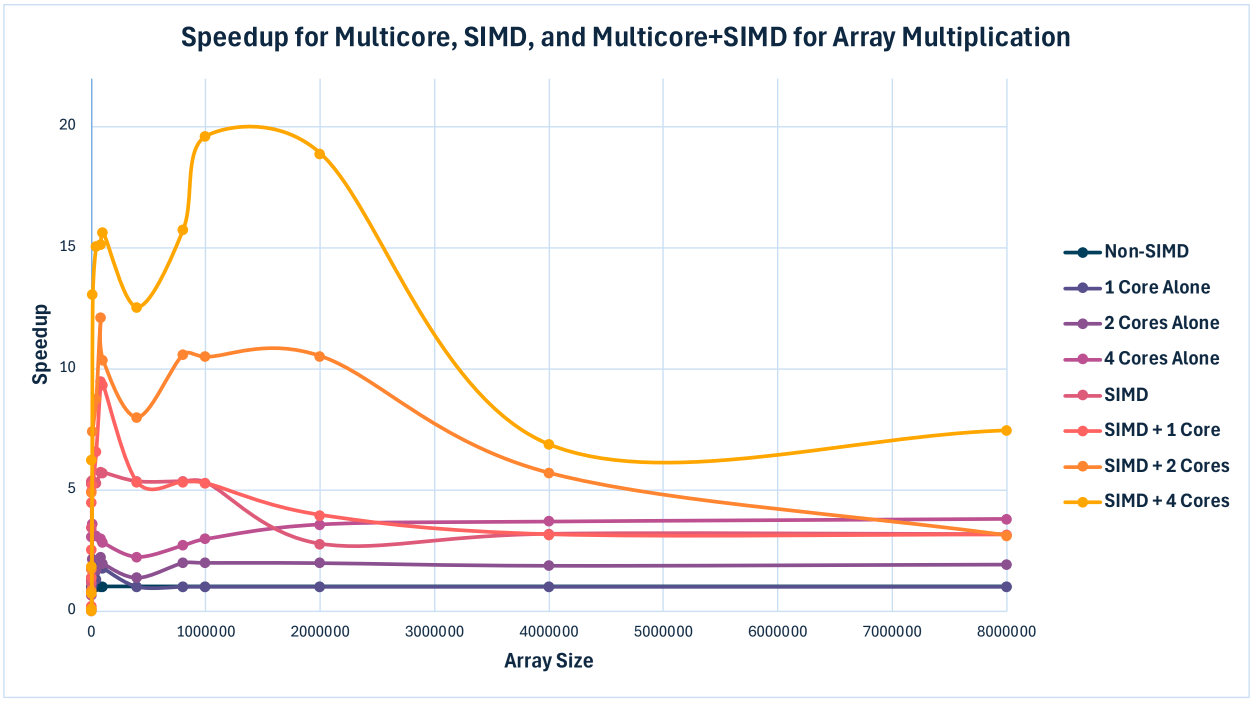
7. **(Extra Credit) Combining SIMD with OpenMP**

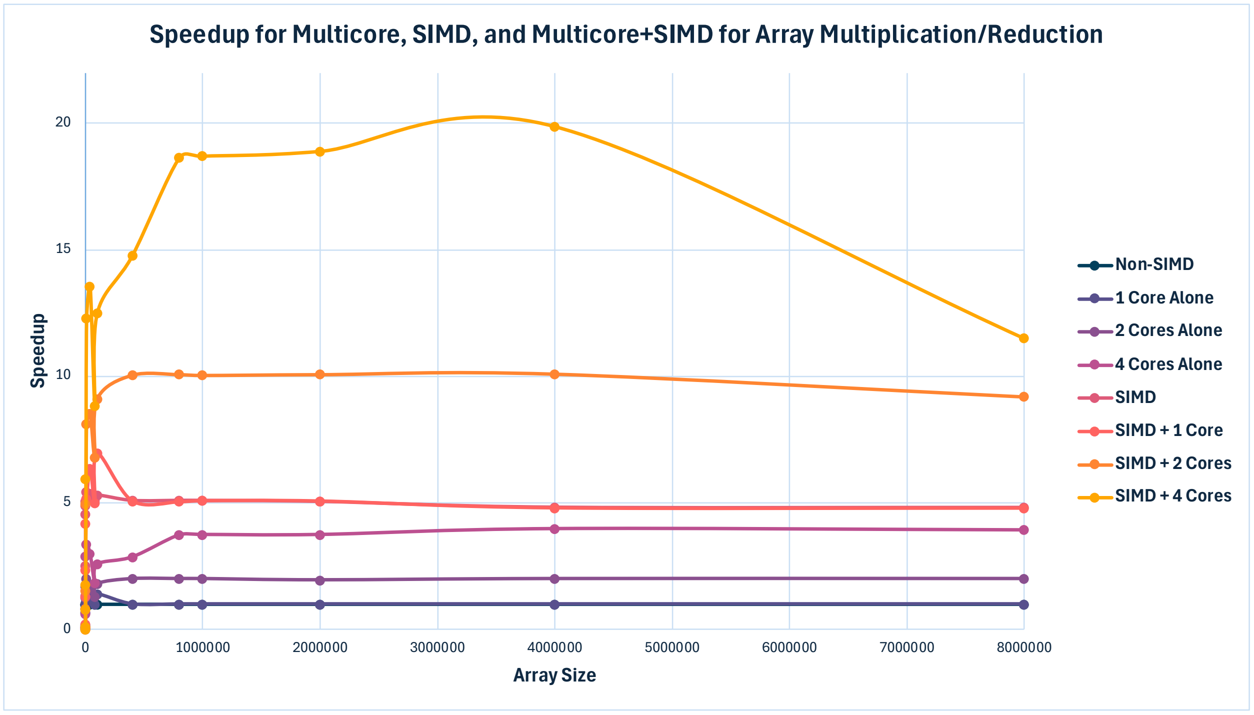
   (1) **Combine multithreading and SIMD in a single test. In this case, you will vary *both* the array size and the number of threads (NUMT). Show your table of performances.**

| **Array Multiplication C+F1:T1[i] = A[i] * B[i]** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ARRAYSIZE | Non-SIMD Performance (megamults/sec) | Non-SIMD SpeedUp | 1 Core alone Performance (megamults/sec) | 1 Core Alone SpeedUp | 2 Cores Alone Performance (megamults/sec) | 2 Cores Alone Speedup | 4 Cores Alone Performance (megamults/sec) | 4 Cores Alone SpeedUp |
| 4 | 41.7 | 1 | 2.31 | 0.06 | 1.19 | 0.03 | 0.95 | 0.02 |
| 40 | 102.63 | 1 | 19.67 | 0.19 | 9.82 | 0.1 | 9.92 | 0.1 |
| 400 | 122.33 | 1 | 81.63 | 0.67 | 88.69 | 0.73 | 81.35 | 0.67 |
| 1000 | 123.76 | 1 | 102.84 | 0.83 | 158.91 | 1.28 | 173.02 | 1.4 |
| 4000 | 134.42 | 1 | 136.76 | 1.02 | 231.37 | 1.72 | 411.4 | 3.06 |
| 10000 | 132.77 | 1 | 140.39 | 1.06 | 285.71 | 2.15 | 479.72 | 3.61 |
| 40000 | 154.36 | 1 | 204.66 | 1.33 | 261.85 | 1.7 | 478.15 | 3.1 |
| 80000 | 174.91 | 1 | **328.2** | **1.88** | **389.79** | **2.23** | 521.12 | 2.98 |
| 100000 | 184.92 | 1 | 327.87 | 1.77 | 364.58 | 1.97 | 524.11 | 2.83 |
| 400000 | 328.17 | 1 | 327.77 | 1 | 451.36 | 1.38 | 734.92 | 2.24 |
| 800000 | 327.97 | 1 | 327.78 | 1 | 655 | 2 | 891.67 | 2.72 |
| 1000000 | 327.85 | 1 | 327.76 | 1 | 655.52 | 2 | 981.36 | 2.99 |
| 2000000 | 325.62 | 1 | 326.12 | 1 | 646.55 | 1.99 | 1169.77 | 3.59 |
| 4000000 | 320.82 | 1 | 320.31 | 1 | 603.72 | 1.88 | 1191.85 | 3.71 |
| 8000000 | 319.67 | 1 | 319.8 | 1 | 618.17 | 1.93 | **1220.53** | **3.82** |
| ARRAYSIZE | SIMD Performance (megamults/sec) | SIMD SpeedUp | SIMD + 1 Core Performance (megamults/sec) | SIMD + 1 Core SpeedUp | SIMD + 2 Cores Performance (megamults/sec) | SIMD + 2 Cores SpeedUp | SIMD + 4 Cores Performance (megamults/sec) | SIMD + 4 Cores SpeedUp |
| 4 | 48.26 | 1.16 | 2.34 | 0.06 | 0.96 | 0.02 | 0.95 | 0.02 |
| 40 | 354.96 | 3.46 | 22.91 | 0.22 | 9.94 | 0.1 | 9.66 | 0.09 |
| 400 | 603.23 | 4.93 | 172.84 | 1.41 | 106.27 | 0.87 | 92.59 | 0.76 |
| 1000 | 649.38 | 5.25 | 312.5 | 2.53 | 212.64 | 1.72 | 227.83 | 1.84 |
| 4000 | 720.87 | 5.36 | 602.42 | 4.48 | 658.74 | 4.9 | 837.23 | 6.23 |
| 10000 | 719.69 | 5.42 | 694.69 | 5.23 | 984.72 | 7.42 | 1737.31 | 13.09 |
| 40000 | 816.53 | 5.29 | 1019.09 | 6.6 | 1349.9 | 8.75 | 2326.82 | 15.07 |
| 80000 | **1001.87** | **5.73** | **1658.1** | **9.48** | **2122.6** | **12.14** | 2650.76 | 15.15 |
| 100000 | 1058.53 | 5.72 | 1726.52 | 9.34 | 1918.17 | 10.37 | 2889.67 | 15.63 |
| 400000 | 1755.64 | 5.35 | 1743.78 | 5.31 | 2628.06 | 8.01 | 4118.31 | 12.55 |
| 800000 | 1755.55 | 5.35 | 1750.13 | 5.34 | 3480.64 | 10.61 | 5170 | 15.76 |
| 1000000 | 1736.44 | 5.3 | 1730.94 | 5.28 | 3444.6 | 10.51 | **6433.85** | **19.62** |
| 2000000 | 900.86 | 2.77 | 1291.92 | 3.97 | 3431.04 | 10.54 | 6154.02 | 18.9 |
| 4000000 | 1021.55 | 3.18 | 1018.19 | 3.17 | 1830.66 | 5.71 | 2206.93 | 6.88 |
| 8000000 | 1012.26 | 3.17 | 1009.42 | 3.16 | 1000.02 | 3.13 | 2385.14 | 7.46 |

| **Array Multiplication/Reduction sum = ΣA[i]*B[i]** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ARRAYSIZE | Non-SIMD Performance (megamults/sec) | Non-SIMD SpeedUp | 1 Core alone Performance (megamults/sec) | 1 Core Alone SpeedUp | 2 Cores Alone Performance (megamults/sec) | 2 Cores Alone Speedup | 4 Cores Alone Performance (megamults/sec) | 4 Cores Alone SpeedUp |
| 4 | 40.14 | 1 | 2.25 | 0.06 | 1.24 | 0.03 | 0.95 | 0.02 |
| 40 | 105.27 | 1 | 19.55 | 0.19 | 9.8 | 0.09 | 9.99 | 0.09 |
| 400 | 130.11 | 1 | 84.21 | 0.65 | 89.89 | 0.69 | 80.45 | 0.62 |
| 1000 | 131.88 | 1 | 107.96 | 0.82 | 166.3 | 1.26 | 174.37 | 1.32 |
| 4000 | 143.8 | 1 | 146.17 | 1.02 | 243.9 | 1.7 | 417.53 | 2.9 |
| 10000 | **143.9** | 1 | 162.17 | 1.13 | **288.84** | **2.01** | 484.82 | 3.37 |
| 40000 | 188.37 | 1 | 242.1 | 1.29 | 304.09 | 1.61 | 560.53 | 2.98 |
| 80000 | 354.68 | 1 | 353.29 | 1 | 463.23 | 1.31 | 650.19 | 1.83 |
| 100000 | 254.93 | 1 | **353.56** | **1.39** | 459.18 | 1.8 | 657.06 | 2.58 |
| 400000 | 354.12 | 1 | 353.51 | 1 | 707.87 | 2 | 1014.59 | 2.87 |
| 800000 | 353.9 | 1 | 353.65 | 1 | 706.76 | 2 | 1325.19 | 3.74 |
| 1000000 | 354.02 | 1 | 353.86 | 1 | 707.48 | 2 | 1326.19 | 3.75 |
| 2000000 | 353.76 | 1 | 353.69 | 1 | 690.4 | 1.95 | 1325.6 | 3.75 |
| 4000000 | 353.04 | 1 | 352.8 | 1 | 707.46 | 2 | **1403.86** | **3.98** |
| 8000000 | 352.51 | 1 | 352.4 | 1 | 703.35 | 2 | 1384.31 | 3.93 |
| ARRAYSIZE | SIMD Performance (megamults/sec) | SIMD SpeedUp | SIMD + 1 Core Performance (megamults/sec) | SIMD + 1 Core SpeedUp | SIMD + 2 Cores Performance (megamults/sec) | SIMD + 2 Cores SpeedUp | SIMD + 4 Cores Performance (megamults/sec) | SIMD + 4 Cores SpeedUp |
| 4 | 32.54 | 0.81 | 2.26 | 0.06 | 0.94 | 0.02 | 0.96 | 0.02 |
| 40 | 266.77 | 2.53 | 22.4 | 0.21 | 12.89 | 0.12 | 10.1 | 0.1 |
| 400 | 592 | 4.55 | 170 | 1.31 | 105.37 | 0.81 | 101.73 | 0.78 |
| 1000 | 643.92 | 4.88 | 309.88 | 2.35 | 204.66 | 1.55 | 232.24 | 1.76 |
| 4000 | 729.94 | 5.08 | 603.06 | 4.19 | 711.74 | 4.95 | 853.45 | 5.94 |
| 10000 | **781.13** | **5.43** | 745.42 | 5.18 | 1171.25 | 8.14 | 1770.83 | 12.31 |
| 40000 | 1013.59 | 5.38 | 1197.07 | 6.35 | 1608.15 | 8.54 | 2553.79 | 13.56 |
| 80000 | 1803.7 | 5.09 | 1770.39 | 4.99 | 2409.15 | 6.79 | 3134.33 | 8.84 |
| 100000 | 1353.33 | 5.31 | **1777.59** | **6.97** | 2326 | 9.12 | 3186.84 | 12.5 |
| 400000 | 1806.51 | 5.1 | 1798.88 | 5.08 | 3561.07 | 10.06 | 5235.66 | 14.78 |
| 800000 | 1806.59 | 5.1 | 1787.74 | 5.05 | 3566.2 | 10.08 | 6596.66 | 18.64 |
| 1000000 | 1806.35 | 5.1 | 1802.86 | 5.09 | 3557.95 | 10.05 | 6624.89 | 18.71 |
| 2000000 | 1791.33 | 5.06 | 1796.48 | 5.08 | 3565.41 | 10.08 | 6686.01 | 18.9 |
| 4000000 | 1704.35 | 4.83 | 1697.98 | 4.81 | **3566.37** | **10.1** | **7017.08** | **19.88** |
| 8000000 | 1698.36 | 4.82 | 1691.89 | 4.8 | 3244.58 | 9.2 | 4058.41 | 11.51 |

(2) **Produce a graph similar to the one on Slide #21 of the *SIMD Vector* notes, using your numbers.**





**(3) Add a brief discussion of what your curves are showing and why you think it is working this way.**

- *For both array multiplication and multiplication/reduction, using 2 and 4 cores can result in noticeable speedup as the array size increases for both SIMD and non-SIMD. The speedup from using 2 cores and 4 cores can reach nearly 2x and 4x respectively for array sizes >= 80,000 compared to SIMD-alone and non-SIMD-alone*. This aligns with Gustafson's Law that as problem size (i.e., array size) increases, the parallelizable portion grows, thus a greater performance gain from parallelization. Once the workload is large enough to fully saturate the CPU cores, a near-ideal speedup can be achieved. However, the speedup will never go beyond 2x or 4x with 2 or 4 cores due to the fact that the program will always have a non-parallelizable portion.
- *When SIMD and multithreading are combined, the pattern of speedups is similar to SIMD-alone cases but with significantly improved performance.* The maximum achievable SIMD multicore/non-SIMD speedup can reach ~10x with 2 cores and ~20x with 4 cores. Multithreading adds an extra layer of performance gain, which helps the speedup plateau at its peak for larger array sizes even SIMD speedup starts to taper off due to increased cache misses.
- *However, for very large array sizes, the SIMD multicore/non-SIMD speedup can continue to decline and may even drop below the SIMD-alone speedup*. This is because the overhead for managing multiple threads becomes more and more significant at very large array sizes and even outweighs the performance gain from SIMD and parallelization.