Jessica Spokoyny<br>
May 22, 2017<br>
Project #5<br>
Vectorized Array Multiplication

1. My Machine:

I ran this program on flip in linux from a windows 10 laptop. My main function is contained in a file called project5.cpp

I wrote a script to automate the program in a file called runProj5.py . It can be compiled and executed by typing:

```
python3 runProj5.py
```

2. My performance results:

Array multiplication:

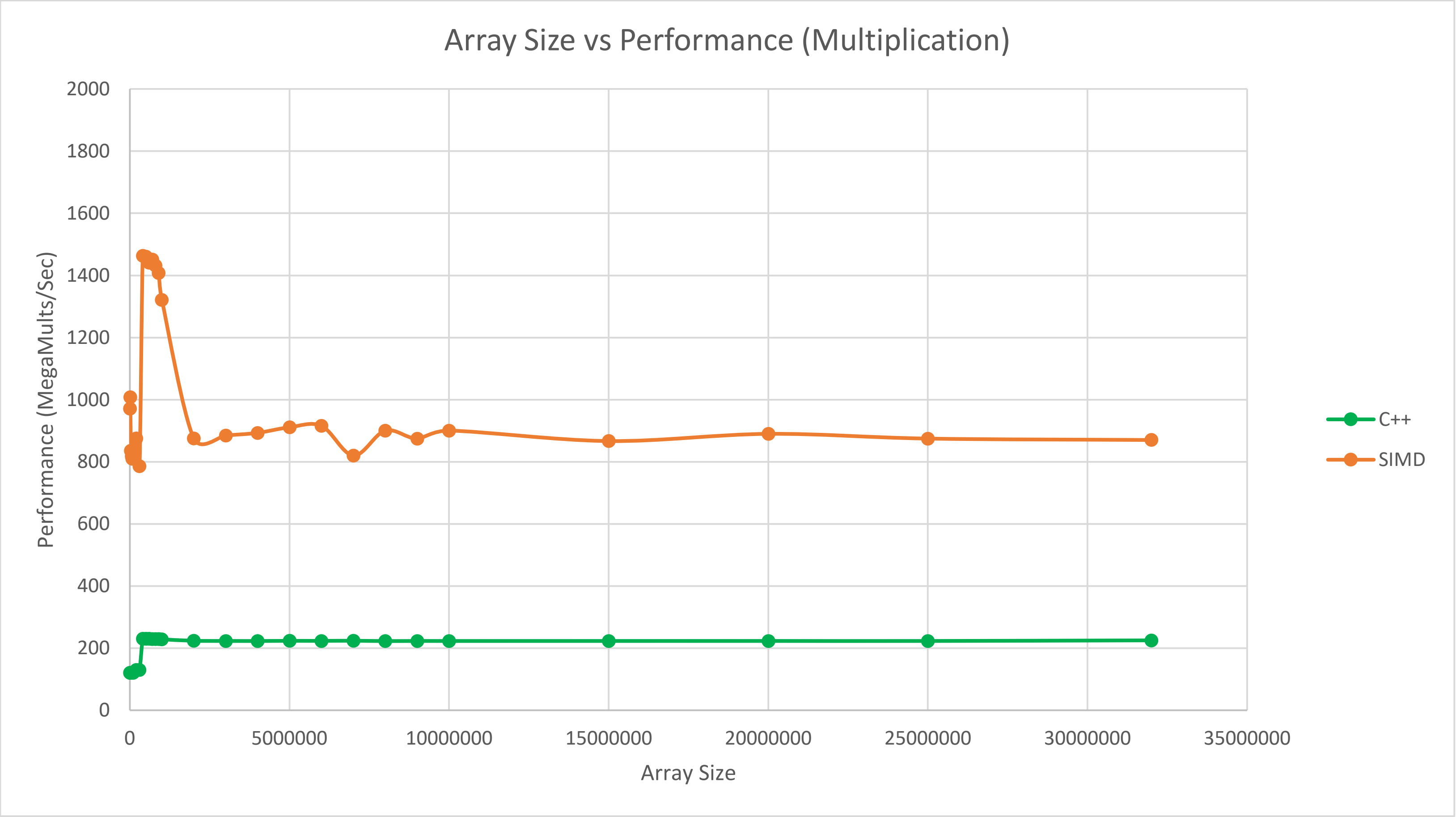
| ARRAYSIZE | C++ Timing (seconds) | | SIMD Timing (seconds) | | Performance (MegaMults/Sec) | | Speedup |
|---|---|---|---|---|---|---|---|
| | C++ Peak | C++ Average | SIMD Peak | SIMD Average | C++ Peak | SIMD Peak | |
| 1000 | 0.00000827 | 0.00000839 | 0.00000103 | 0.0000011 | 120.9189843 | 970.8738 | 8.029126214 |
| 10000 | 0.00008271 | 0.00008352 | 0.00000992 | 0.00001001 | 120.9043646 | 1008.065 | 8.337701613 |
| 30000 | 0.00024833 | 0.00025103 | 0.0000359 | 0.00003631 | 120.8069907 | 835.6546 | 6.917270195 |
| 50000 | 0.00041362 | 0.0004166 | 0.00006113 | 0.00006177 | 120.8839031 | 817.929 | 6.766235891 |
| 70000 | 0.0005782 | 0.00058277 | 0.00008646 | 0.00008724 | 121.0653753 | 809.6229 | 6.687485542 |
| 100000 | 0.00082887 | 0.00083428 | 0.00012251 | 0.00012349 | 120.6461809 | 816.2599 | 6.76573341 |
| 200000 | 0.00153594 | 0.00154082 | 0.00022853 | 0.00023855 | 130.2134198 | 875.1586 | 6.720955673 |
| 300000 | 0.00230479 | 0.0023821 | 0.00038184 | 0.00038431 | 130.1637025 | 785.6694 | 6.036009847 |
| 400000 | 0.00173617 | 0.00319755 | 0.00027334 | 0.00027502 | 230.3921851 | 1463.379 | 6.351686544 |
| 500000 | 0.00217172 | 0.00217961 | 0.0003424 | 0.00034451 | 230.2322583 | 1460.28 | 6.342640187 |
| 600000 | 0.00260117 | 0.0026217 | 0.00041629 | 0.00078643 | 230.6654313 | 1441.303 | 6.248456605 |
| 700000 | 0.0030495 | 0.00514608 | 0.0004824 | 0.00049053 | 229.5458272 | 1451.078 | 6.321517413 |
| 800000 | 0.00347935 | 0.00468217 | 0.00055887 | 0.00056433 | 229.9280038 | 1431.46 | 6.225687548 |
| 900000 | 0.00392048 | 0.00512529 | 0.00063933 | 0.00064882 | 229.5637269 | 1407.724 | 6.132169615 |
| 1000000 | 0.00437696 | 0.00581575 | 0.00075662 | 0.00079198 | 228.4690744 | 1321.667 | 5.784885411 |
| 2000000 | 0.0089358 | 0.01014341 | 0.00228457 | 0.00231108 | 223.8187963 | 875.4383 | 3.91137063 |
| 3000000 | 0.01343896 | 0.01472484 | 0.00339018 | 0.00341478 | 223.2315596 | 884.9088 | 3.964084503 |
| 4000000 | 0.01792593 | 0.01916757 | 0.00447789 | 0.00451504 | 223.1404452 | 893.2779 | 4.003209101 |
| 5000000 | 0.02235789 | 0.02651216 | 0.00548688 | 0.00665285 | 223.634699 | 911.2647 | 4.074791138 |
| 6000000 | 0.02684699 | 0.02942486 | 0.00655254 | 0.00658659 | 223.4887412 | 915.6754 | 4.097188266 |
| 7000000 | 0.03125881 | 0.03261286 | 0.00853321 | 0.00869485 | 223.9368677 | 820.3244 | 3.663194741 |

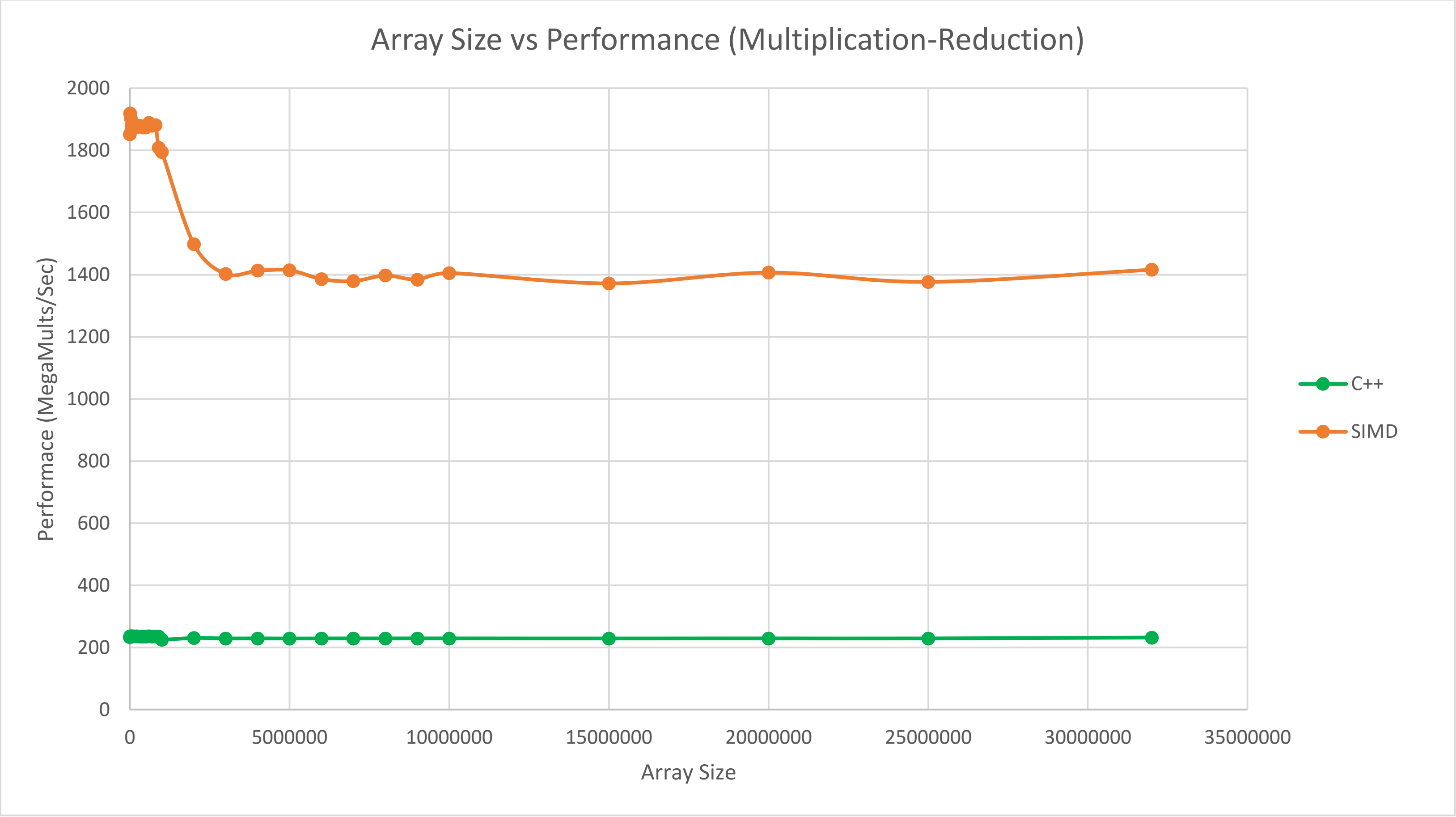
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 8000000 | 0.03589608 | 0.03609282 | 0.0088829 | 0.00953514 | 222.8655608 | 900.6068 | 4.041031645 |
| 9000000 | 0.04030456 | 0.04144555 | 0.01029528 | 0.0105207 | 223.2997954 | 874.187 | 3.914858071 |
| 10000000 | 0.0448177 | 0.04497463 | 0.01110521 | 0.0111834 | 223.126131 | 900.4782 | 4.03573638 |
| 15000000 | 0.06721246 | 0.0673972 | 0.01730253 | 0.01752695 | 223.1729057 | 866.9252 | 3.884545208 |
| 20000000 | 0.08958104 | 0.08985086 | 0.02246897 | 0.0225907 | 223.2615294 | 890.1165 | 3.986877903 |
| 25000000 | 0.11202863 | 0.11261517 | 0.02857529 | 0.02875641 | 223.1572411 | 874.8818 | 3.920472198 |
| 32000000 | 0.14209378 | 0.14286236 | 0.03676769 | 0.03817045 | 225.2033833 | 870.3294 | 3.864637131 |

Array multiplication-reduction:

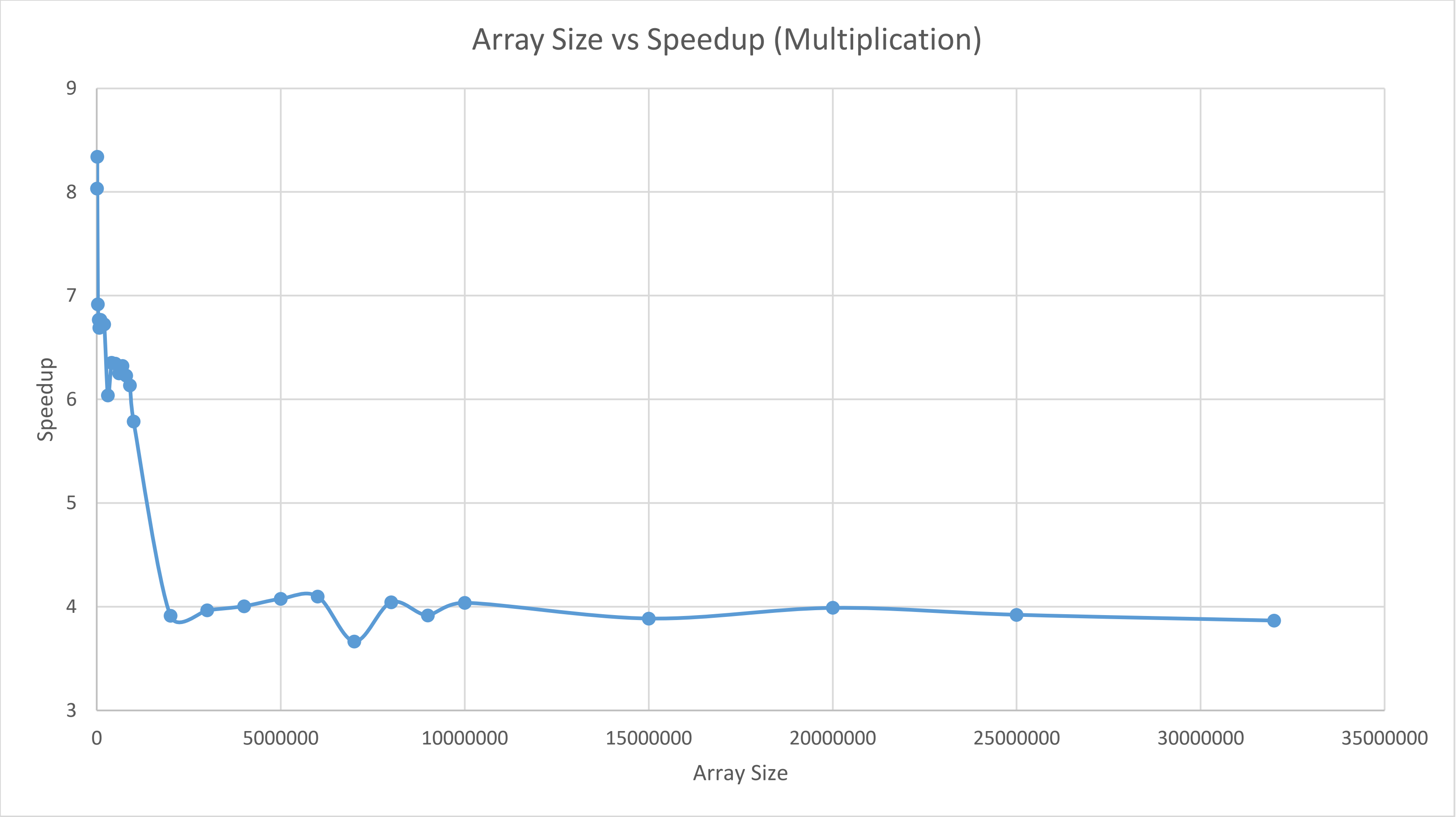
| ARRAYSIZE | C++ Timing (seconds) | | SIMD Timing (seconds) | | Performance (MegaMults/Sec) | | Speedup |
|---|---|---|---|---|---|---|---|
| | C++ Peak | C++ Average | SIMD Peak | SIMD Average | C++ Peak | SIMD Peak | |
| 1000 | 0.00000429 | 0.00000432 | 0.00000054 | 0.00000055 | 233.1002331 | 1851.851852 | 7.944444444 |
| 10000 | 0.00004237 | 0.0000427 | 0.00000521 | 0.00000523 | 236.0160491 | 1919.385797 | 8.13243762 |
| 30000 | 0.00012709 | 0.00012803 | 0.00001577 | 0.00001584 | 236.0531907 | 1902.346227 | 8.058972733 |
| 50000 | 0.00021216 | 0.00021436 | 0.00002661 | 0.00002679 | 235.6711916 | 1878.99286 | 7.972942503 |
| 70000 | 0.00029697 | 0.00029836 | 0.00003732 | 0.00003751 | 235.7140452 | 1875.669882 | 7.957395498 |
| 100000 | 0.00042421 | 0.00042628 | 0.00005334 | 0.00005358 | 235.7323024 | 1874.765654 | 7.952943382 |
| 200000 | 0.00084847 | 0.00087856 | 0.00010667 | 0.00010909 | 235.7184108 | 1874.941408 | 7.954157683 |
| 300000 | 0.00127578 | 0.00128505 | 0.0001596 | 0.0001602 | 235.150261 | 1879.699248 | 7.993609023 |
| 400000 | 0.00169797 | 0.00171308 | 0.00021346 | 0.00021451 | 235.5754224 | 1873.887379 | 7.954511384 |
| 500000 | 0.00212247 | 0.00212963 | 0.00026682 | 0.0002679 | 235.57459 | 1873.922495 | 7.954688554 |
| 600000 | 0.0025467 | 0.00255698 | 0.00031772 | 0.00031981 | 235.5990105 | 1888.455244 | 8.015548282 |
| 700000 | 0.00297323 | 0.00298359 | 0.00037238 | 0.00037498 | 235.4341911 | 1879.800204 | 7.984397658 |
| 800000 | 0.00339908 | 0.00341315 | 0.00042518 | 0.00042787 | 235.3578027 | 1881.556047 | 7.99444941 |
| 900000 | 0.00382337 | 0.00394036 | 0.00049773 | 0.00050041 | 235.3944295 | 1808.20927 | 7.68161453 |
| 1000000 | 0.00443653 | 0.00444426 | 0.00055705 | 0.00055896 | 225.4013835 | 1795.17099 | 7.964329952 |
| 2000000 | 0.00868328 | 0.00884171 | 0.00133465 | 0.0013507 | 230.3277103 | 1498.520211 | 6.50603529 |
| 3000000 | 0.01310586 | 0.01468074 | 0.00213912 | 0.00216112 | 228.9052378 | 1402.445866 | 6.126753057 |
| 4000000 | 0.01745863 | 0.01845483 | 0.00283087 | 0.00286002 | 229.1130518 | 1412.993179 | 6.167231275 |
| 5000000 | 0.0218606 | 0.02194306 | 0.00353476 | 0.00364812 | 228.721993 | 1414.523193 | 6.18446514 |
| 6000000 | 0.02620366 | 0.02760191 | 0.00432862 | 0.00436735 | 228.9756469 | 1386.12306 | 6.053582897 |
| 7000000 | 0.03056759 | 0.03066942 | 0.00507439 | 0.00514658 | 229.0007161 | 1379.476154 | 6.023894498 |
| 8000000 | 0.03493791 | 0.03610444 | 0.00572426 | 0.00620947 | 228.9776349 | 1397.560558 | 6.103480625 |
| 9000000 | 0.03927932 | 0.04156572 | 0.00650238 | 0.00672532 | 229.1282028 | 1384.108588 | 6.04076046 |
| 10000000 | 0.04365861 | 0.04464442 | 0.00711545 | 0.00716805 | 229.0498942 | 1405.392491 | 6.135748266 |
| 15000000 | 0.06554296 | 0.07048181 | 0.01093265 | 0.01145187 | 228.857531 | 1372.036972 | 5.995157624 |
| 20000000 | 0.08731087 | 0.08757948 | 0.01421706 | 0.01435095 | 229.0665527 | 1406.76061 | 6.141274638 |
| 25000000 | 0.10917367 | 0.1096608 | 0.01815522 | 0.01875734 | 228.9929431 | 1377.014434 | 6.013348778 |
| 32000000 | 0.1380887 | 0.13835394 | 0.02259804 | 0.02324053 | 231.7351094 | 1416.052012 | 6.110649419 |

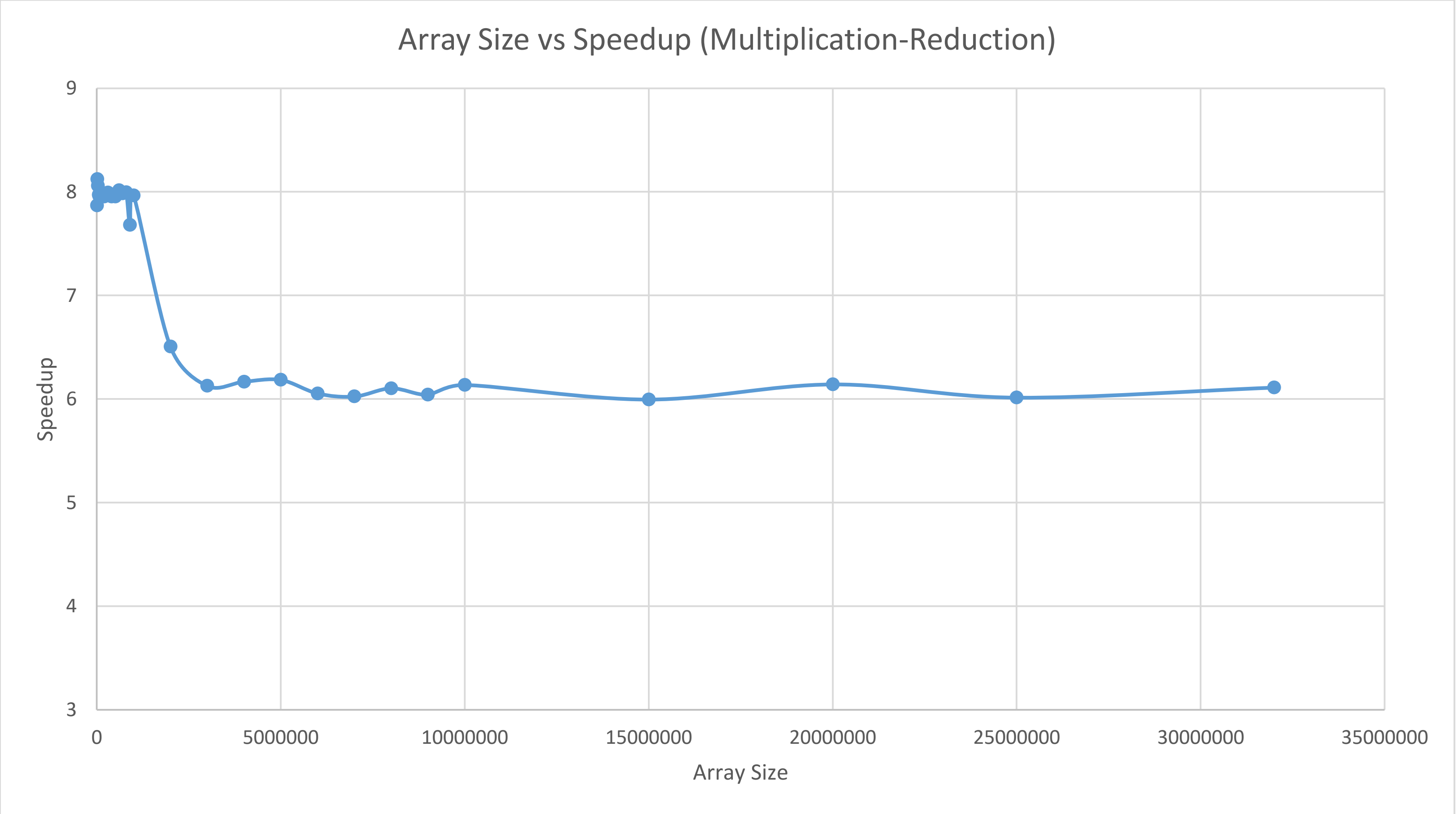
Graphs of Performance:





Graphs of Speedup:





## 3. Speedup Patterns:

For array multiplication, the speedup (of SIMD to non-SIMD) starts around 8 for the smallest array sizes, then there is a huge drop to around 4 and then it remains consistently around 4 as array size increases.

For array multiplication-reduction, the speedup (of SIMD to non-SIMD) also starts around 8 for the smallest array sizes and then there is a drop to around 6 and it remains consistently around 6 as array size increases.

## 4. Consistency

When the array size is between 1,000-200,000 for array multiplication (and 1,000-300,000 for array multiplication-reduction), we see a dramatic decrease in the speedup values (from 8 to 4, and 8 to 6 respectively). Within this range, as the array size increases, speedup drastically decreases.

Once the array size reaches 200000 for array multiplication (and 300000 for array multiplication-reduction), the speedup values are very consistent. For multiplication, the speedup stays around 4 and for multiplication-reduction, the speedup stays around 6.

Regardless of where we look at these graphs, we can see that the SIMD implementation is still a considerable improvement from the C++ implementation for any array size.

## 5. Explanation of Consistency

I believe that the initial dramatic drops in speedup are due to cache misses. SIMD requires 4 times the space in cache as non-SIMD and the CPU is trying to fetch data from the cache that isn’t being filled in time. Initially, the cache is filled with values we need which is why we have the most dramatic speedup initially. Then, as array size grows, we need to fill the cache line with more values but they are needed faster than they can be delivered.

It makes sense that the larger array sizes are consistent because the SIMD code is able to perform 4x the number of mathematical operations at a time as the non-SIMD code.

## 6. Array multiplication speedup != 4

Speedup for array multiplication could be less than 4 because of cache misses due to temporal coherence and the program requiring 3 arrays to store each of the 2 numbers to be multiplied as well as the product in a third array.

Speedup could be more than 4 if the values needed are already on the cache line and can be accessed very quickly.

7. Array multiplication-reduction speedup != 4

Speedup for array multiplication-reduction could be less than 4 because of cache misses due to temporal coherence and the program requiring 2 arrays to store each of the 2 numbers to be multiplied then added.

Speedup could be more than 4 because when performing the multiplication-reduction in C++, the sum is stored in a local variable, which is stored to memory via cache.  However, when performing the same operation with SIMD, the sum is stored in another xmm register which is much faster to access.