John Olgin
olginj@oregonstate.edu
CS 475 – Spring 2019
Project #4
Vectorized Array Multiplication and Reduction using SSE

Description

The information in this report is an analysis of the effect of SSE on speedup when performing array multiplication and array multiplication with reduction. My program was developed on macOS Mojave version 10.14 and tested on the flip server provided by Oregon State University.

Since speedup is what we'll be measuring, it will be placed on the Y-axis. There is no specific unit of measurement for speedup. It is simply a multiplier to demonstrate the increased efficiency of SSE for array multiplication and array multiplication with reduction. For this report, speedup will be calculated by the following equation for both experiments:

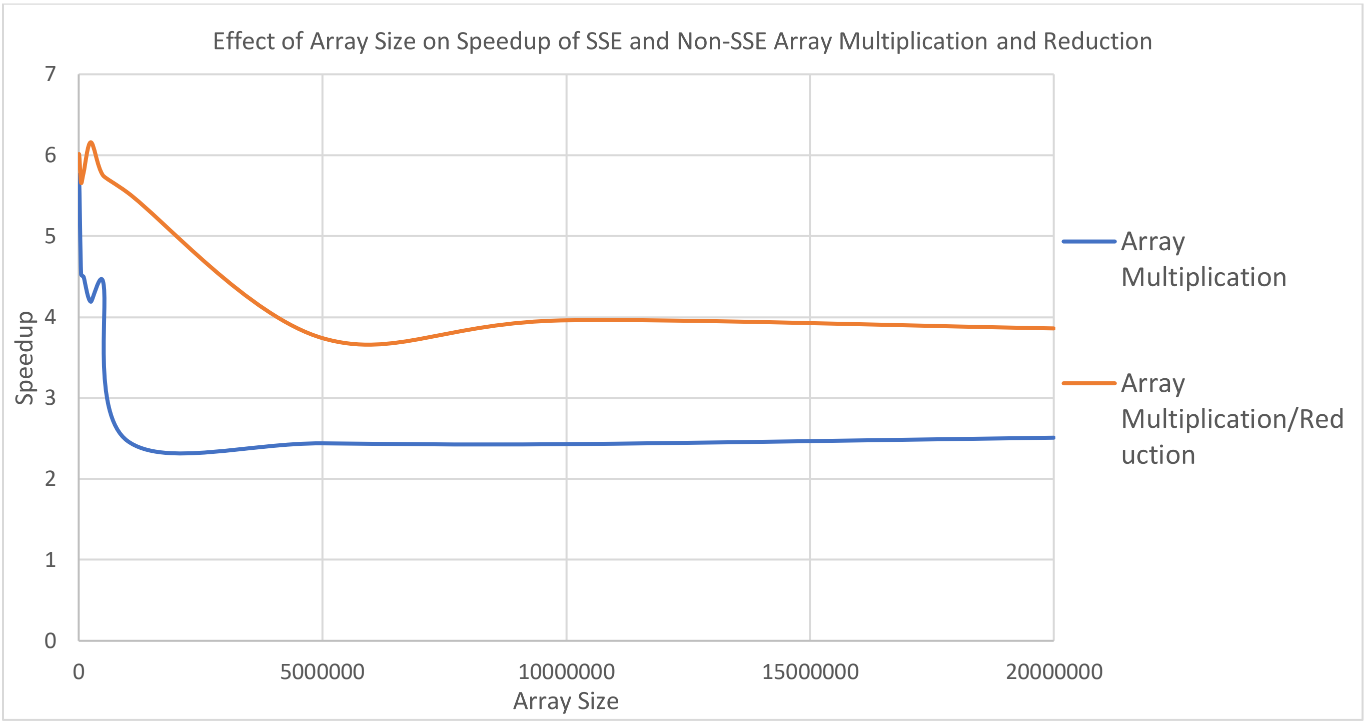
$$\textbf{Speedup} = \mathbf{T_{Non\text{-}SSE} / T_{SSE}}$$

Theoretically, since the vectorization (SSE) will allow four floating-point operations to happen simultaneously, speedup should be higher than one. We intuitively expect it to always be around four, but we'll soon see evidence to the contrary.

The first experiment will be the effect of SSE on array multiplication. In other words, we are going to determine how much faster vectorization can multiply all the corresponding elements from two different arrays and place them in another array. Second, we'll test vectorization's effects on multiplying the corresponding elements from each array and summing them all together.

Data and Graph

Array Size	Array Multiplication	Array Multiplication/Reduction
1000	5.58	5.84
10000	5.74	6.01
50000	4.53	5.66
100000	4.5	5.78
250000	4.19	6.16
500000	4.44	5.75
1000000	2.47	5.54
5000000	2.44	3.74
10000000	2.43	3.96
20000000	2.51	3.86



For array multiplication, speedup started around six. It experienced some inconsistency before taking a sharp dive around the 500000-element mark. It seemed to hit a valley around 2.5 and remained constant all the way through 20000000 elements. Therefore, it seems as if SSE has its largest effect on speedup for smaller array sizes. While it does offer increased efficiency in large arrays, the effect isn't as intense.

The plotted line for array multiplication with reduction was somewhat similar. It started with roughly the same speedup as the first experiment and also experienced some erratic behavior at the smaller element sizes. However, it decreased much slower and didn't hit a more consistent speedup until about the 5000000-element mark. Also, in the 10000000 to 20000000-element region the speedup seems to start slowly declining.

I believe the speedup's irregularity can be largely attributed to the cache issue discussed in the lecture. Cache is almost a non-issue for the smaller array sizes. However, as the sizes increase to infinity, cache can no longer keep up with the CPU and more cache lines are required. This is why (excluding one random spike) we see primarily decreases in speedup as array size increases. After the initial inconsistency, we don't see much of an increase.

Questions 6 and 7

Experiencing a speedup of less than four is most likely due to the increase in cache misses as array size increases. Since we’re constantly referencing arrays to perform multiplication and reduction, larger arrays are going to require more cache lines to obtain the necessary elements in the array. As mentioned above, this is why the plotted line for array multiplication and reduction begins decreasing steadily from four.

Achieving a speedup of four isn’t supposed to happen, but I think it’s possible given how we’re performing the experiment. We’re basically comparing C++ code with assembly language. Since assembly language is more efficient and more similar to machine code, it will perform better than C++ in general. Therefore, at least in the smaller arrays where cache isn’t an issue yet, speedup could possibly go above four. However, as the cache becomes busier the advantages of using assembly language become slightly counteracted.