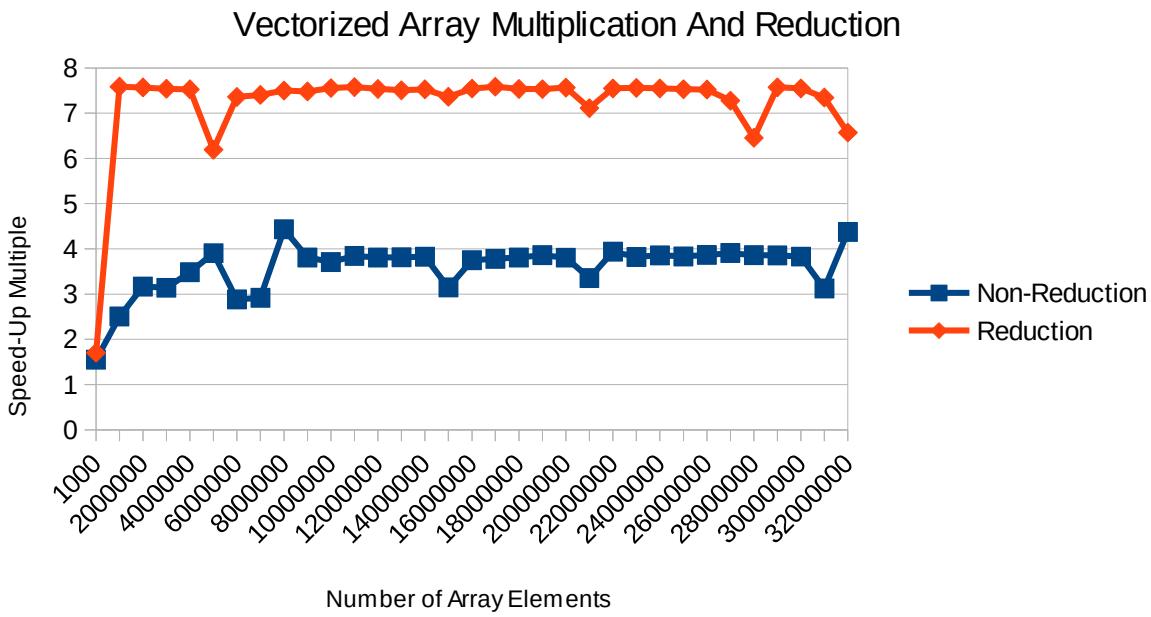
Travis Robinson
CS475
Spring 2016
Project 5
Vectorized Array Multiplication and Reduction

For project 5, I used OSUs Flip server, at access.engr.oregonstate.edu.

Table And Chart

Vectorized Array Multiplcation and Reduction		
Length	Non-Reduction	Reduction
1000	1.549482	1.703754
1000000	2.505125	7.585893
2000000	3.165735	7.568983
3000000	3.139292	7.53775
4000000	3.483168	7.528567
5000000	3.906835	6.19092
6000000	2.882271	7.35998
7000000	2.921315	7.405128
8000000	4.432095	7.499025
9000000	3.809526	7.479907
10000000	3.708605	7.555506
11000000	3.842442	7.57633
12000000	3.809073	7.535654
13000000	3.81506	7.506073
14000000	3.828529	7.527486
15000000	3.146829	7.36315
16000000	3.749459	7.543507
17000000	3.780014	7.585005
18000000	3.809894	7.534399
19000000	3.860751	7.531435
20000000	3.806974	7.566143
21000000	3.354155	7.111096
22000000	3.938343	7.545819
23000000	3.821408	7.559042
24000000	3.856352	7.548422
25000000	3.83341	7.529081
26000000	3.865509	7.520887
27000000	3.907911	7.275089
28000000	3.86169	6.4535
29000000	3.856776	7.570984
30000000	3.829537	7.547186
31000000	3.123247	7.344575
32000000	4.375939	6.571261



Commentary

The pattern that we see in these graphs is that both non-reduction and reduction have relatively constant speedups across array sizes, with the exception of the occasional dip or peak. (At least when the array size isn't really small). This is because when the number of array elements exceeds the number of processing units, there's no further speedups to make, because each one is only able to perform a limited number of operations on a limited number of elements in a specified time period; adding more array elements won't change the amount of processing that can be done, so we hit an improvement cap when the number of elements has exceeded the processing units.

We also see from the graph that the speedup for non reduction is around four, while the speedup for reduction is a little less than eight. The reason for this is that with SIMD, four floats can be done at a time, while with regular processing, we can only do one, so we would expect to get an improvement of around four times as much. With reduction though, we do get around eight times the improvement, and that is because of the way that reductions work; with a reduction, once a calculation is completed, it sends it's results up to the next level, which then compiles all the results it received, adds it's results, and sends that up to the next level. This is repeated until the last level is reached where all results have been put together. Sort of like a pyrimid. This gives means that with out reduction we were getting O(N) times, but with it O(log N). (In other words, for each element we add, the increase in needed time to solve gets halved). This is why we get closer to eight times the speed for reduction calculations: four floating point values, and results being combined at the next level instead of one-at-a-time.