Braden Ackles

CS 475

Project 5

Write Up

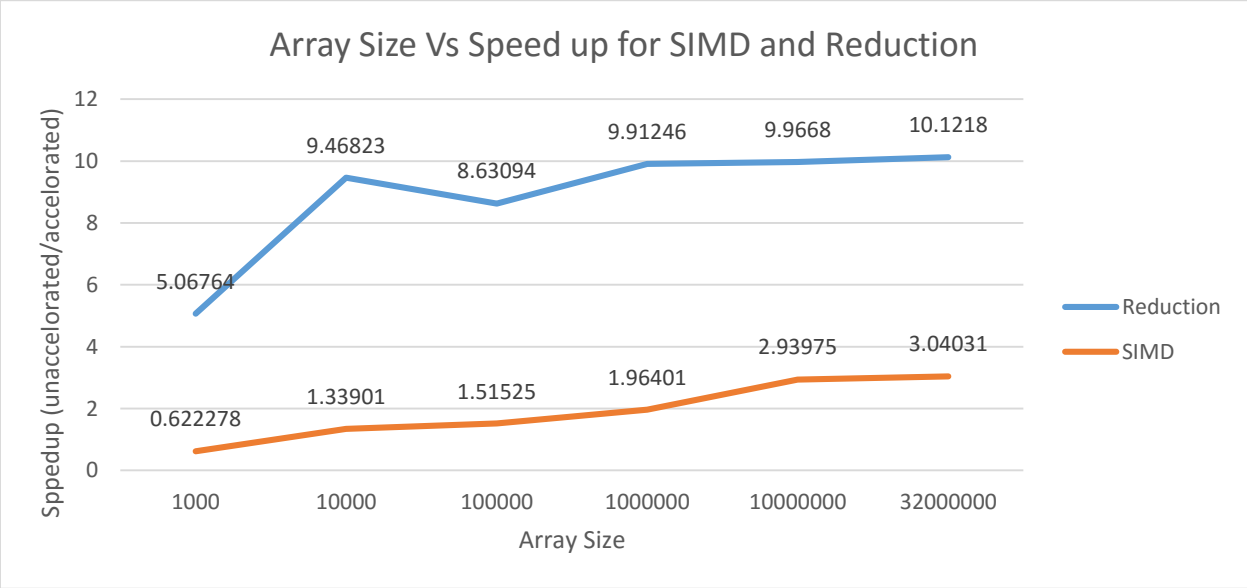
Machine

I ran this program on flip. At the time of running it I did a top on the server and there was less than 1% usage across all time frames.

Table:

	Speed Up	Array Size		Speed Up	Array Size
SIMD:	0.622278	1000	Reduction:	5.06764	1000
SIMD:	1.33901	10000	Reduction:	9.46823	10000
SIMD:	1.51525	100000	Reduction:	8.63094	100000
SIMD:	1.96401	1000000	Reduction:	9.91246	1000000
SIMD:	2.93975	10000000	Reduction:	9.9668	10000000
SIMD:	3.04031	32000000	Reduction:	10.1218	32000000

Graph:



Patterns:

Firstly, SIMD, we know that it has a maximum of 4 times speed up but we never expect to see that because that is theoretical. We see almost a constant speed up between the size of the array and the speedup times. Basically creating a linear line between array size and speed up meaning that each the assembly and personal code calculate data at seemingly constant speeds but the SIMD way has the

ability to simply do more at a time. With the array size getting larger allowing it to be faster than the c++ code for longer this shows its true speed up due to things like set up and moving things to and from memory. This is shown from the first data point being less than 1 meaning that it was actually slower than the c code. This is most likely due to the set up time for the assembly code.

Secondly, Reduction, this has a very weird speed up. It is always faster than the un-accelerated code by a factor of 5 times. Then jumping to 9.5 then basically having diminishing returns on the size of the array. This is most likely due to the fact that doing reduction in the assembly code was more efficient with how it added the values together, IE when and where. Also going to memory less times because of that as well. This meant that it would get the immediate speedup shown but then have diminishing returns after that because it is already that much faster that it can't be to many times faster.

These patterns are consistent across different array sizes. Except the one dip in the reduction line at 100,000 array size. The lines tend to follow a very set pattern compared to array size. And this was show based on the average and max speed up. This makes sense why they would be this way the only reason why it wouldn't be is if the floating points that are basically random in this code where more complex this could have been the issue in the array at 100,000 the floating points used could have been more complex and there for require more time. Other than this it would makes sense that the speed ups would be what they are for SIMD and reduction because of SIMD (4) and reduction the way it keeps track. The array size increases allowing the accelerated function to be able to be faster for longer (more calculations).

SEE SIMD

SEE SIMD does 4 floating points at a time. So we should see a maximum speed up of 4. This is because it can do 4 calculations in SIMD where without it only 1 could occur. This means that the maximum speed up is 4. However, we won't get that because that would be too easy. We have set up to do and moving things around which takes time and is constant in both sets meaning that we can't speed that up and we won't see the speedup on that part. Meaning that as array sizes get increasingly big we will see an asymptote line because we simply can't get any more speed up from the function due to constants in the time taken equation.

This is KIND of like threads in previous programs the more threads you use the faster your program gets. But at some point adding more threads only provided diminishing or no returns on speed up.