Oregon State University

CS 475 - Spring 2021

Professor

Mike Bailey

mjb@cs.oregonstate.edu

Project #4

Vectorized Array Multiplication/Reduction using SSE

by

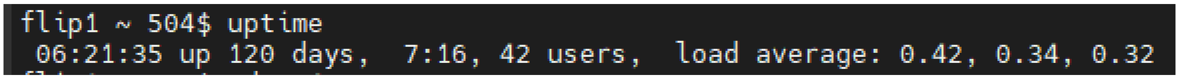
Abdullah Saydemir

saydemia@oregonstate.edu

May 6, 2021

Setup:

This experiment is run on *flip* servers. Uptime command outputted,



```
flip1 ~ 504$ uptime
 06:21:35 up 120 days,  7:16, 42 users,  load average: 0.42, 0.34, 0.32
```

ARRAYSIZE = [1024, 4096, 10240, 32768, 48000, 64000, 128000, 250000,
500000, 1000000]

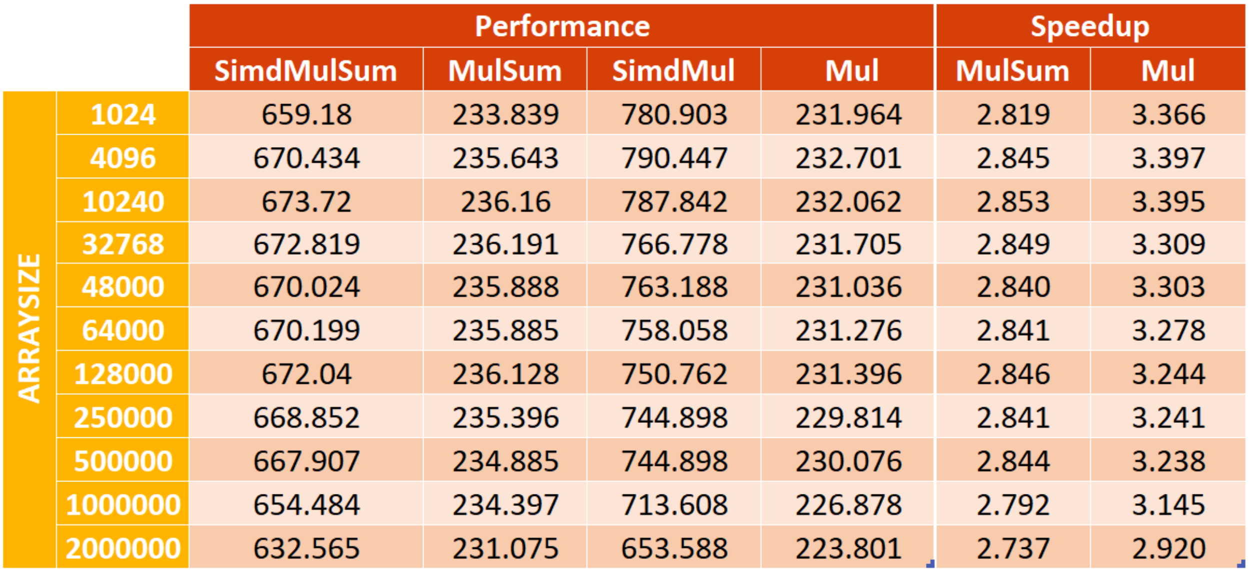
NUMTRIES = 10,000

Because I was getting 2 to 2.5 speedup at max, I used a large number of NUMTRIES and it was helpful getting 0.5 to 1 increase in the speedups. I was not expecting a higher speedup just by using a really high NUMTRIES.

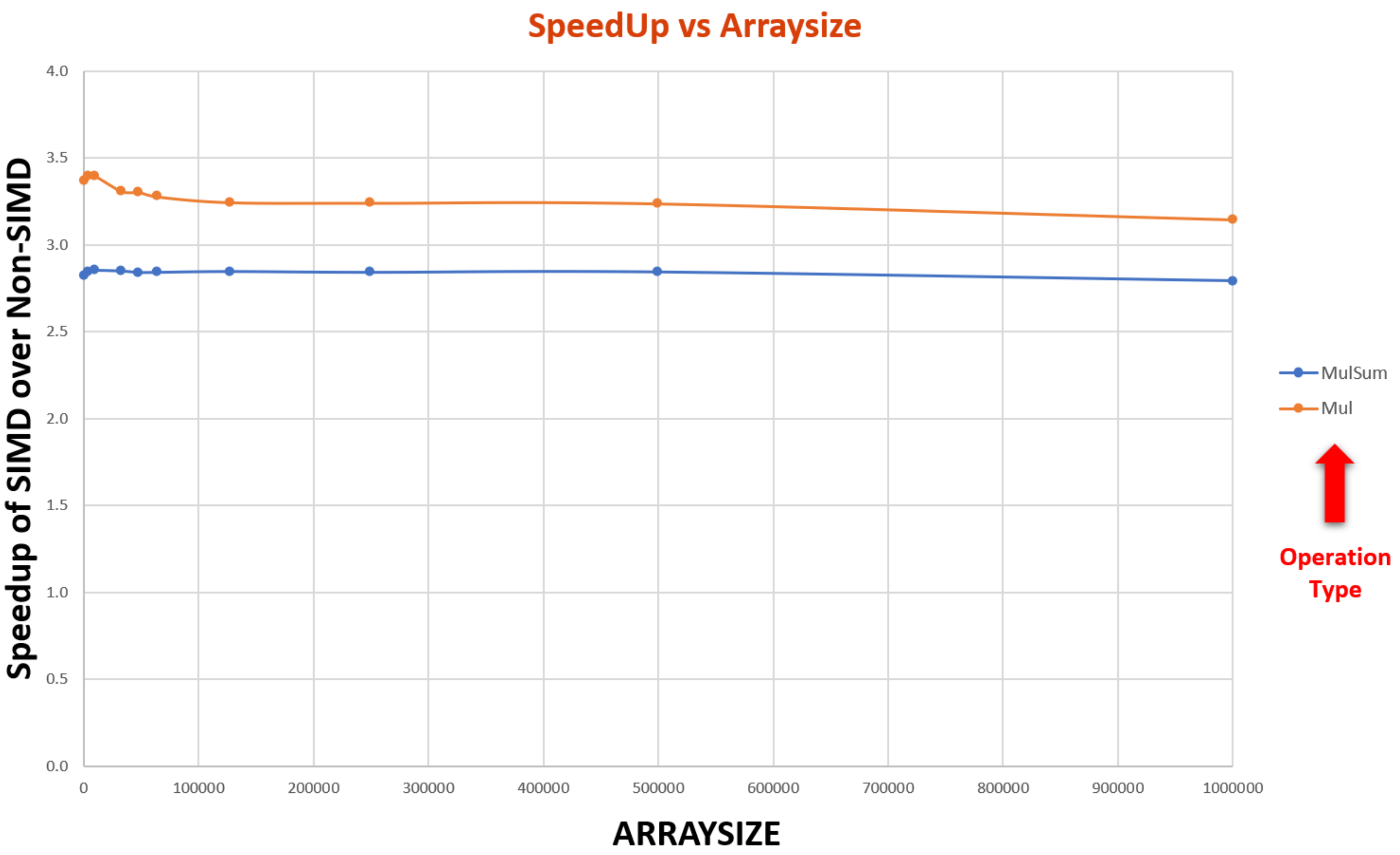
Results:

I used Intel intrinsics SIMD code. Expected speedup value is around 3, not 4.

Performance and Speedup table of each operation is as follows.



		Performance				Speedup	
		SimdMulSum	MulSum	SimdMul	Mul	MulSum	Mul
ARRAYSIZE	1024	659.18	233.839	780.903	231.964	2.819	3.366
	4096	670.434	235.643	790.447	232.701	2.845	3.397
	10240	673.72	236.16	787.842	232.062	2.853	3.395
	32768	672.819	236.191	766.778	231.705	2.849	3.309
	48000	670.024	235.888	763.188	231.036	2.840	3.303
	64000	670.199	235.885	758.058	231.276	2.841	3.278
	128000	672.04	236.128	750.762	231.396	2.846	3.244
	250000	668.852	235.396	744.898	229.814	2.841	3.241
	500000	667.907	234.885	744.898	230.076	2.844	3.238
	1000000	654.484	234.397	713.608	226.878	2.792	3.145
	2000000	632.565	231.075	653.588	223.801	2.737	2.920



What patterns are you seeing in the speedups?

For smaller array sizes the speedups are slightly higher than the larger array sizes. As the array size increases the speedup decreases. Speedup is around 2.8 for multiplication and 3.2 for multiplication & sum. Mul operation is always better in terms of speedup.

Are they consistent across a variety of array sizes?

For smaller sizes, the values are not really consistent. They go up and down, because the array size is small and there is a larger error margin for the smaller array sizes. I tried to minimize the error by having really big NUMTRIES and it helped me very well. That’s why the graph is pretty “linear” except for the small array sizes.

Why or why not, do you think?

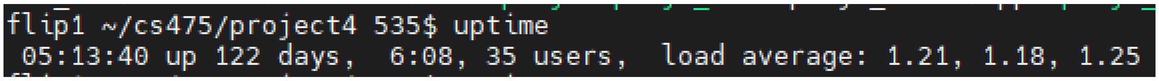
First, I used Intel intrinsics. That’s why the speedup is around 3 and not 4. Other than that, speedup is really consistent and as expected. I used really big NUMTRIES and after some point the array size is also enough not to have a wonky graph.

I also made sure that smaller values are multiples of 64 so that if something happens about cache, it will affect the performance in a good way. That may have a little effect on the consistency of the speedup as well.

EXTRA

Setup:

This experiment is run on *flip* servers. Uptime command outputted,



```
flip1 ~/cs475/project4 535$ uptime
 05:13:40 up 122 days,  6:08, 35 users,  load average: 1.21, 1.18, 1.25
```

ARRAYSIZE = [1024, 4096, 10240, 32768, 48000, 64000, 128000, 250000, 500000, 1000000]

NUMTRIES = 10,000

Because I was getting 2 to 2.5 speedup at max, I used a large number of NUMTRIES as before and it was again helpful getting 0.5 to 1 increase in the speedups.

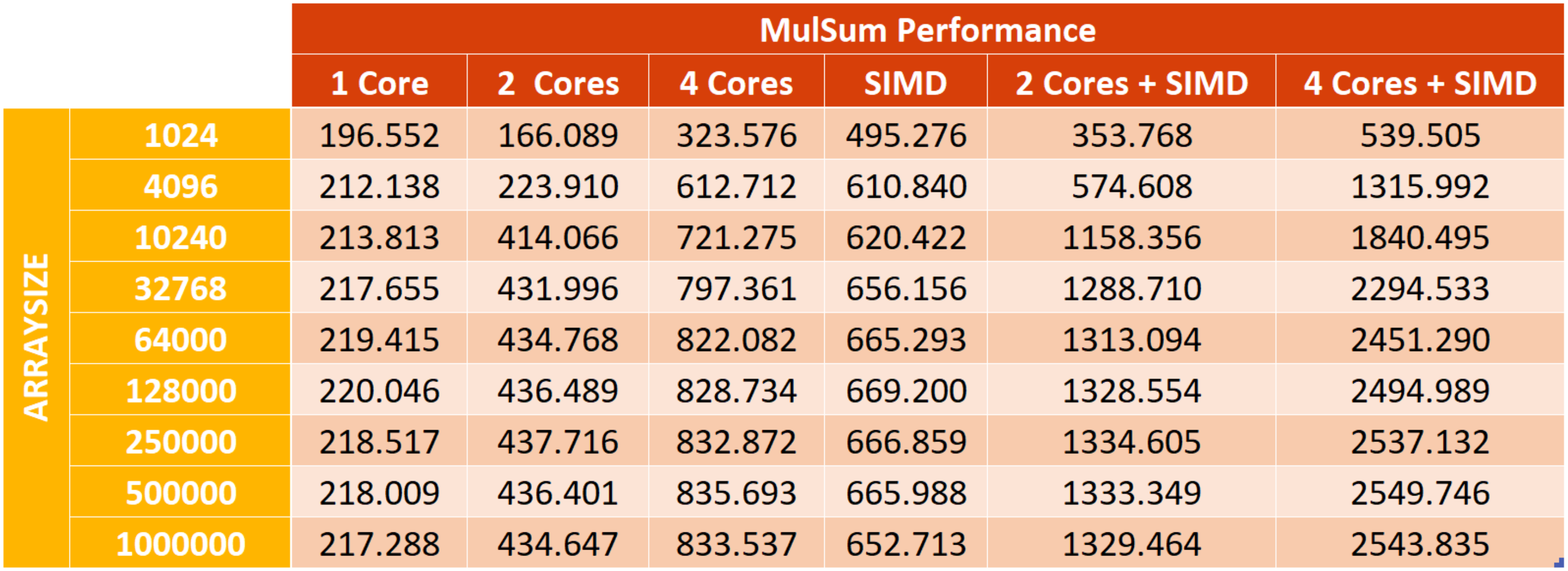
Results:

I used Intel intrinsics SIMD code. Expected speedup value is around 3, not 4. So, combined graphs do not fully utilize the speedup possible.

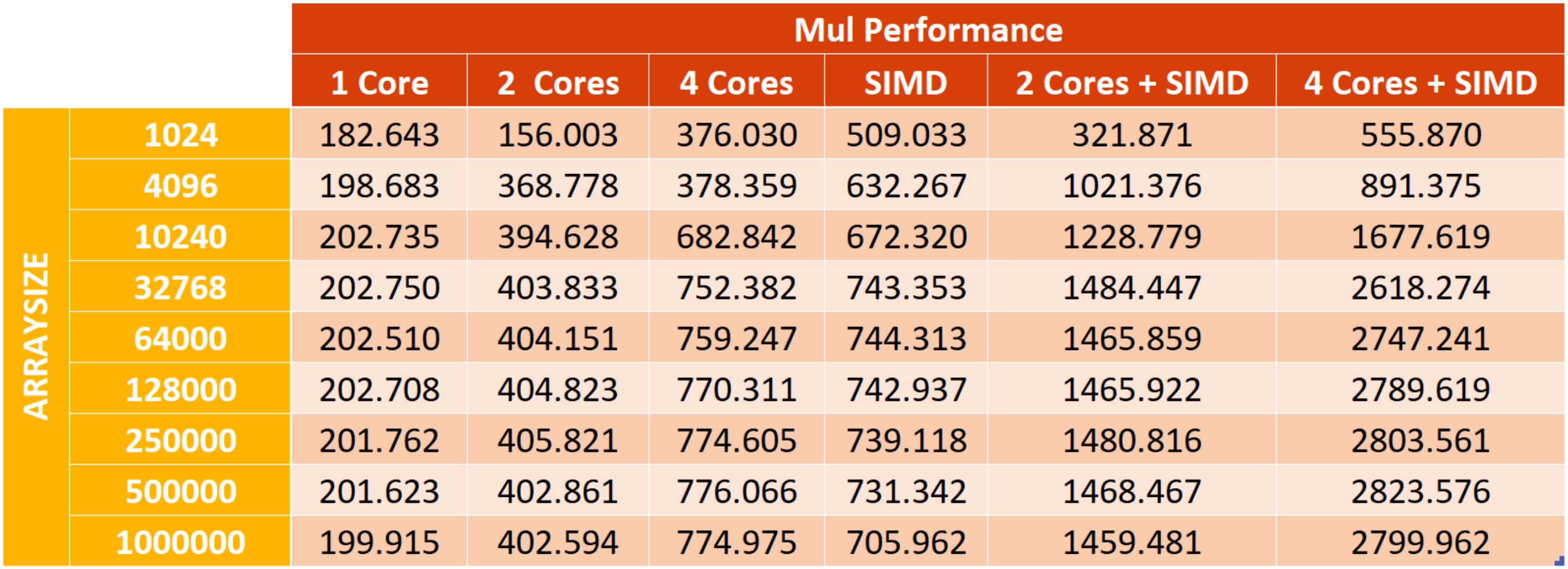
Speedup of Mul and MulSum were pretty different. Speedups for MulSum were better than Mul as opposed to the main experiment. Only difference between the main experiment is that I used arrays inside the main function.

So, I decided that separating the performance values of operations would be better. I would like to point out that the speedup values I collected by running SIMD were different from the values I got in the main experiment. I do not know why intrinsics provide different speedup even with 10,000 NUMTRIES.

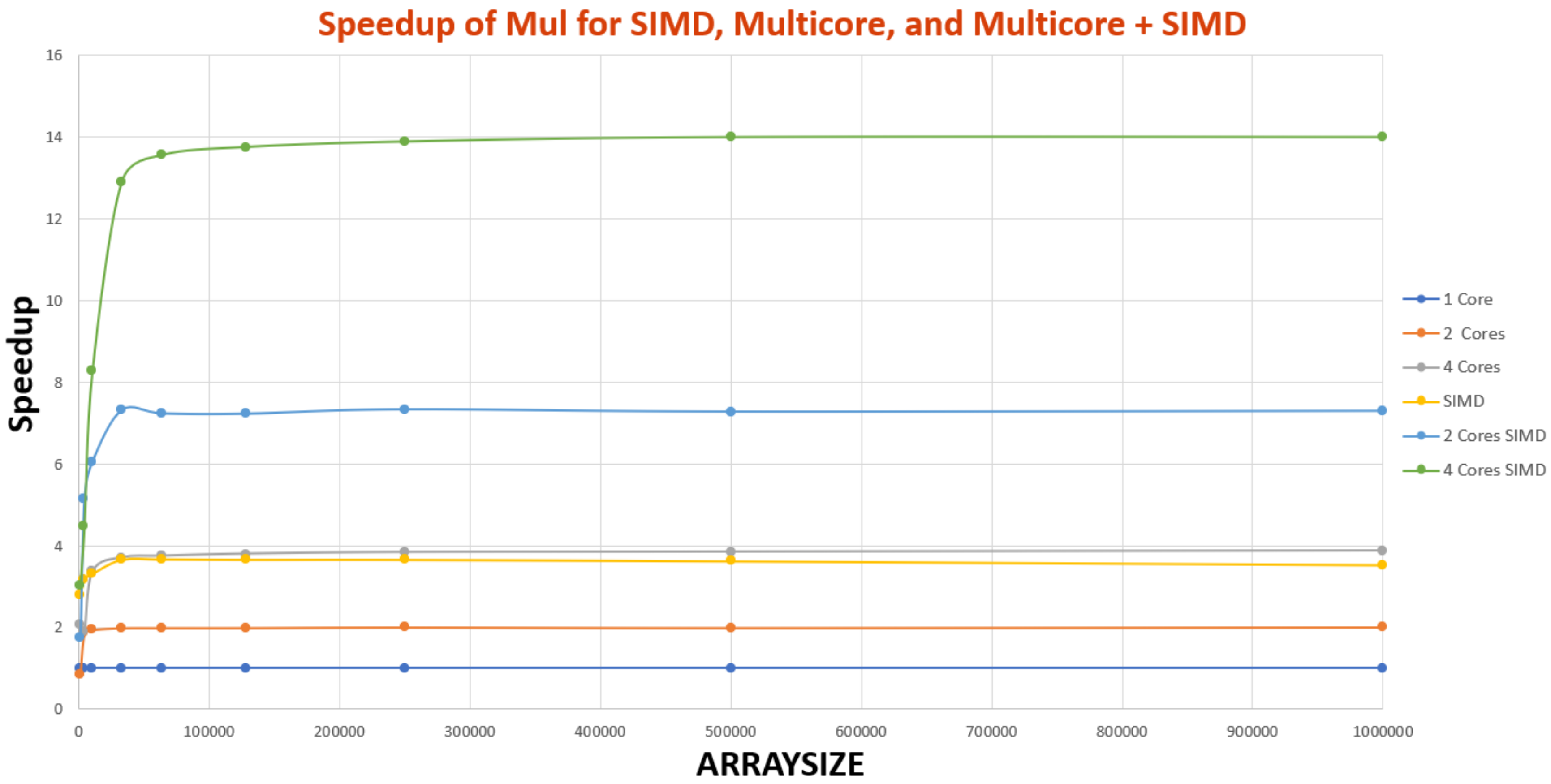
Performance table of each operation is as follows.



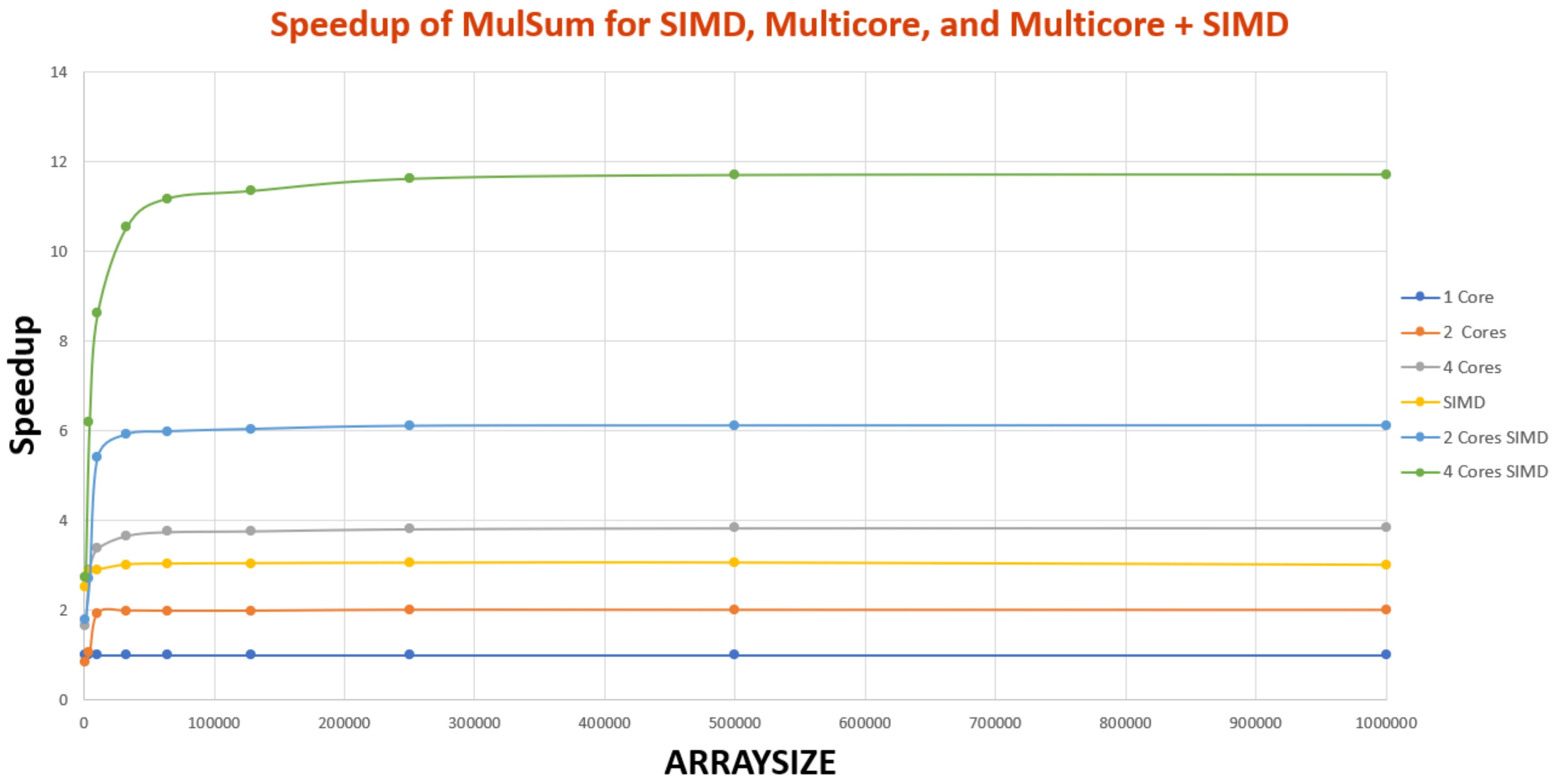
		MulSum Performance					
		1 Core	2 Cores	4 Cores	SIMD	2 Cores + SIMD	4 Cores + SIMD
ARRAYSIZE	1024	196.552	166.089	323.576	495.276	353.768	539.505
	4096	212.138	223.910	612.712	610.840	574.608	1315.992
	10240	213.813	414.066	721.275	620.422	1158.356	1840.495
	32768	217.655	431.996	797.361	656.156	1288.710	2294.533
	64000	219.415	434.768	822.082	665.293	1313.094	2451.290
	128000	220.046	436.489	828.734	669.200	1328.554	2494.989
	250000	218.517	437.716	832.872	666.859	1334.605	2537.132
	500000	218.009	436.401	835.693	665.988	1333.349	2549.746
	1000000	217.288	434.647	833.537	652.713	1329.464	2543.835



		Mul Performance					
		1 Core	2 Cores	4 Cores	SIMD	2 Cores + SIMD	4 Cores + SIMD
ARRAYSIZE	1024	182.643	156.003	376.030	509.033	321.871	555.870
	4096	198.683	368.778	378.359	632.267	1021.376	891.375
	10240	202.735	394.628	682.842	672.320	1228.779	1677.619
	32768	202.750	403.833	752.382	743.353	1484.447	2618.274
	64000	202.510	404.151	759.247	744.313	1465.859	2747.241
	128000	202.708	404.823	770.311	742.937	1465.922	2789.619
	250000	201.762	405.821	774.605	739.118	1480.816	2803.561
	500000	201.623	402.861	776.066	731.342	1468.467	2823.576
	1000000	199.915	402.594	774.975	705.962	1459.481	2799.962



SIMD speedup was to provide 4x speedup but intrinsics gave a little shy values. So, speedup of multicore + SIMD is also less than expected values. 4 Cores + SIMD gives 14x rather than 16x, 2 Cores + SIMD gives around 7 rather than 8 and SIMD gives around 3.5 rather than 4. However, the combination of multicore and SIMD reflects the results of both aspects consistently. For example, 4 Cores provide around 4x, SIMD provides 3.5x speedup and their combination is 14x speedup which is the multiplication of the two numbers.



Speedup of MulSum operation is more inefficient than Mul operation. As you can see, SIMD provides 3x speedup for MulSum operation rather than 3.5 as it was before. This also pulls down the SIMD + Multicore speedup values. 4 Cores + Multicore provides 12x speedup at most. On the other hand, multicore only speedups are the same as it was before. 2 cores provide around 2x and 4 cores provide around 4x speedup. So, something is really messed up with the intrinsics code.