John Olgin
olginj@oregonstate.edu
CS 475 – Spring 2019
Project #5
OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce

Description

In this project I'll be analyzing the effects of parallelism on array multiplication, array multiplication and addition, and array multiplication and reduction. I'll be using OpenCL to measure the effect on performance as we manipulate both the global and local dataset sizes while holding the other constant. My programs were developed on macOS Mojave Version 10.14 using Sublime Text. However, OpenCL functionality has been severely limited in 10.14 since Apple is promoting the use of Metal over OpenCL and OpenGL. Therefore, all my testing was done on rabbit provided by Oregon State University. Unfortunately, it was experiencing significant load for all my tests since everyone was rushing to complete their projects.

To measure performance, I'll keep track of calculations per second performed by the program for different global work sizes while holding local work size constant, and vice versa. Thus, performance will be measured in GigaMults/s, GigaMultsAndAdds/s and GigaMultsAndReductions/s. These will correspond to the programs for array multiplication, array multiplication and addition, and array multiplication and reduction, respectively. The general equation for each of these will be:

(Global Data Size) / (Duration) / 1000000000

When setting my array sizes, I ensured that every single local work size is a factor of every global work size. This forced every iteration of the program to have a perfect, even number of work groups to perform the computations. To make it simple, I used sizes that were powers of 2 so I knew the global sizes were evenly divisible by the local sizes. Any OpenCL errors relating to evenly distributed work groups were avoided by doing this.

Raw Data for Multiply and Multiply with Addition

Global Work Size

Local Work Size	1024	32768	131072	262144	524288	1048576	4194304	8388608
8	0.027	0.666	0.575	0.914	1.457	1.618	2.51	2.724
32	0.028	0.778	0.631	1.15	2.235	3.681	6.295	7.88
64	0.027	0.876	0.667	1.245	2.049	3.9	8.065	10.804
128	0.027	0.851	0.629	1.281	2.059	4.123	8.473	10.63
256	0.028	0.859	0.637	1.27	2.301	4.284	8.338	11.062
512	0.012	0.866	0.654	1.231	2.269	4.196	7.649	11.189

8add	0.027	0.59	1.617	0.756	1.249	1.837	2.354	2.624
32add	0.028	0.806	2.649	1.168	1.668	3.403	5.773	7.189
64add	0.026	0.836	2.946	1.107	1.654	3.147	6.828	9.241
128add	0.027	0.415	2.99	0.799	1.556	3.629	6.394	9.642
256add	0.027	0.853	2.98	1.268	2.332	3.857	7.147	9.759
512add	0.028	0.764	1.778	1.244	1.875	3.689	7.072	9.701

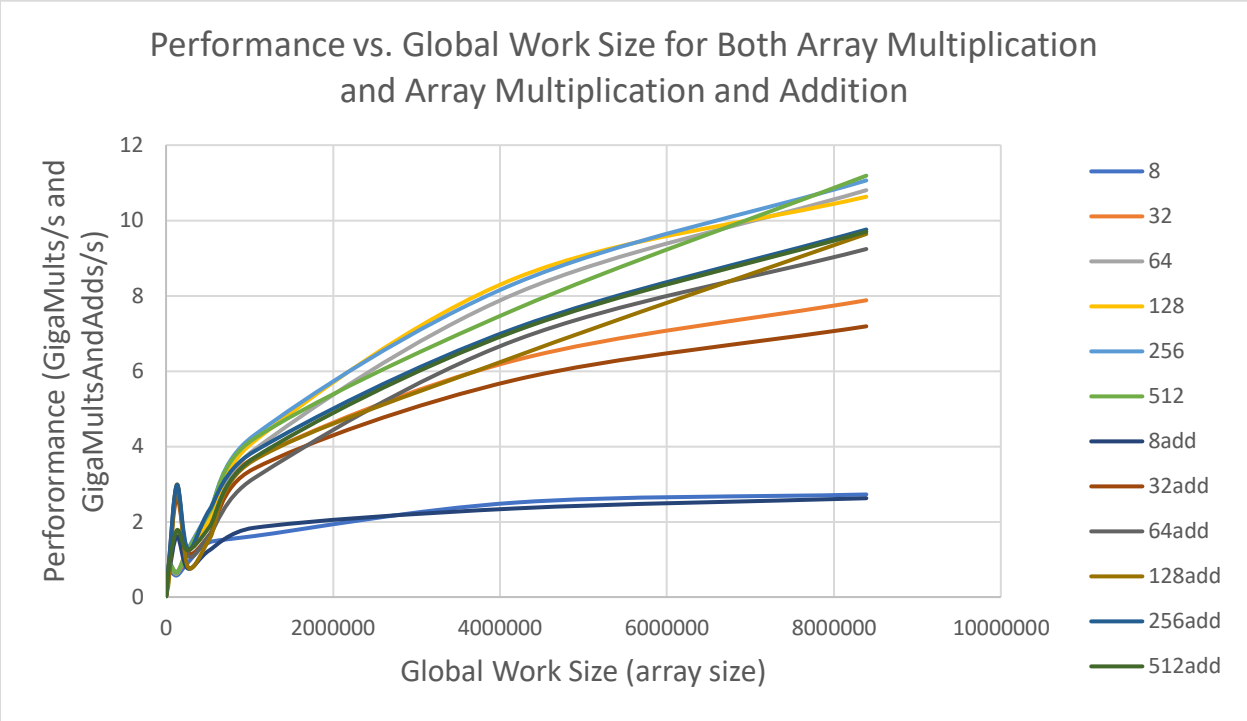
Local Work Size

Global Work Size	8	32	64	128	256	512
1024	0.027	0.028	0.027	0.027	0.028	0.012
1024add	0.027	0.028	0.026	0.027	0.027	0.028
32768	0.666	0.778	0.876	0.851	0.859	0.866
32768add	0.59	0.806	0.836	0.415	0.853	0.764
131072	0.575	0.631	0.667	0.629	0.637	0.654
131072add	1.617	2.649	2.946	2.99	2.98	1.778
262144	0.914	1.15	1.245	1.281	1.27	1.231
262144add	0.756	1.168	1.107	0.799	1.268	1.244
524288	1.457	2.235	2.049	2.059	2.301	2.269
524288add	1.249	1.668	1.654	1.556	2.332	1.875
1048576	1.618	3.681	3.9	4.123	4.284	4.196
1048576add	1.837	3.403	3.147	3.629	3.857	3.689
4194304	2.51	6.295	8.065	8.473	8.338	7.649
4194304add	2.354	5.773	6.828	6.394	7.147	7.072
8388608	2.724	7.88	10.804	10.63	11.062	11.189
8388608add	2.624	7.189	9.241	9.642	9.759	9.701

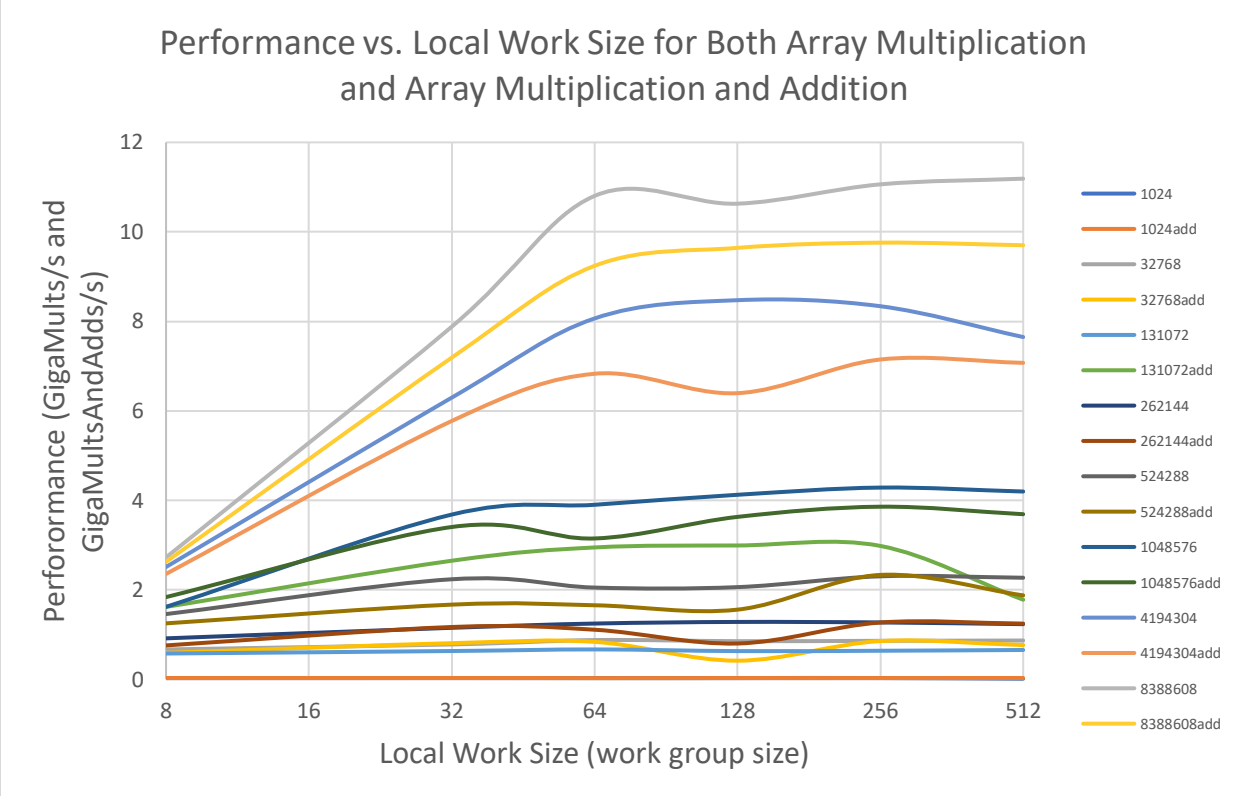
The two tables here present the exact same data, just organized differently to accommodate creating the graphs. You can see that both tables have numbers with “add” appended to the end. This corresponds to the performance measurements collected for the “Array multiplication and addition” part of the project. Thus, the numbers without “add” at the end correspond to just the “array multiplication” section. Since they are required to be graphed together, they are included in the same table, using “add” to differentiate.

Graphs for Multiply and Multiply with Addition

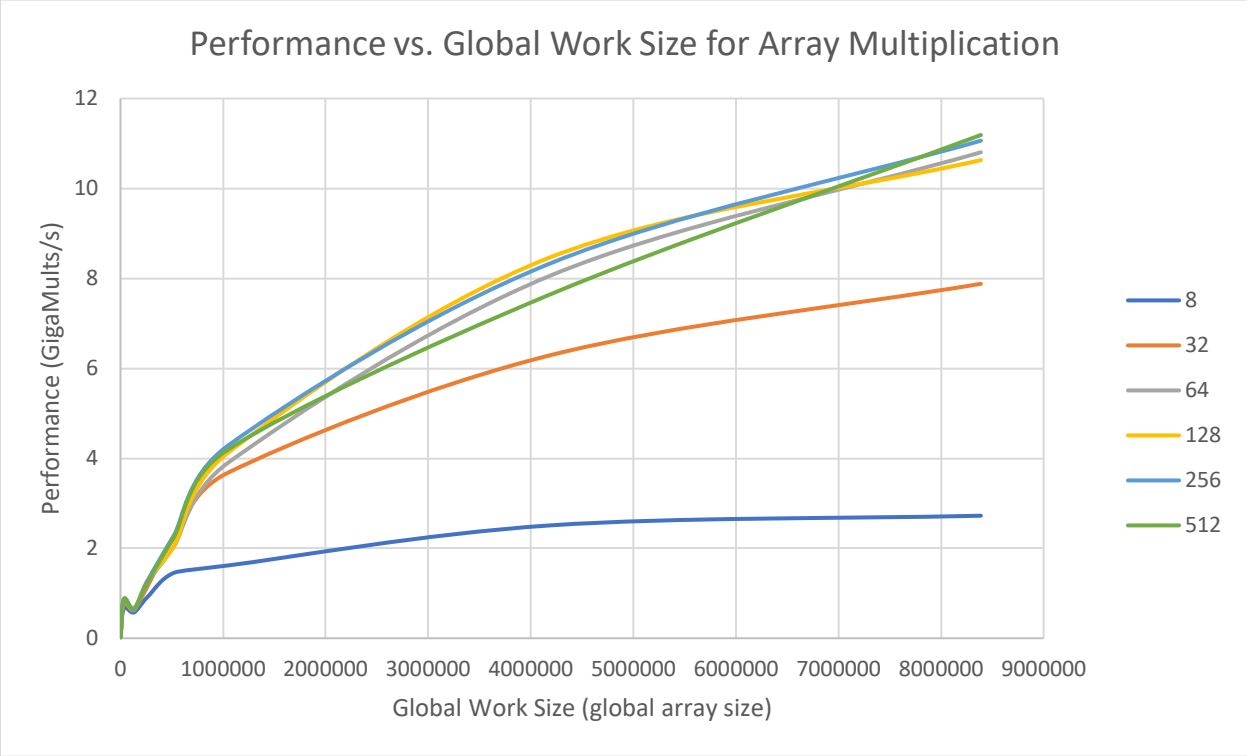
Graph 1



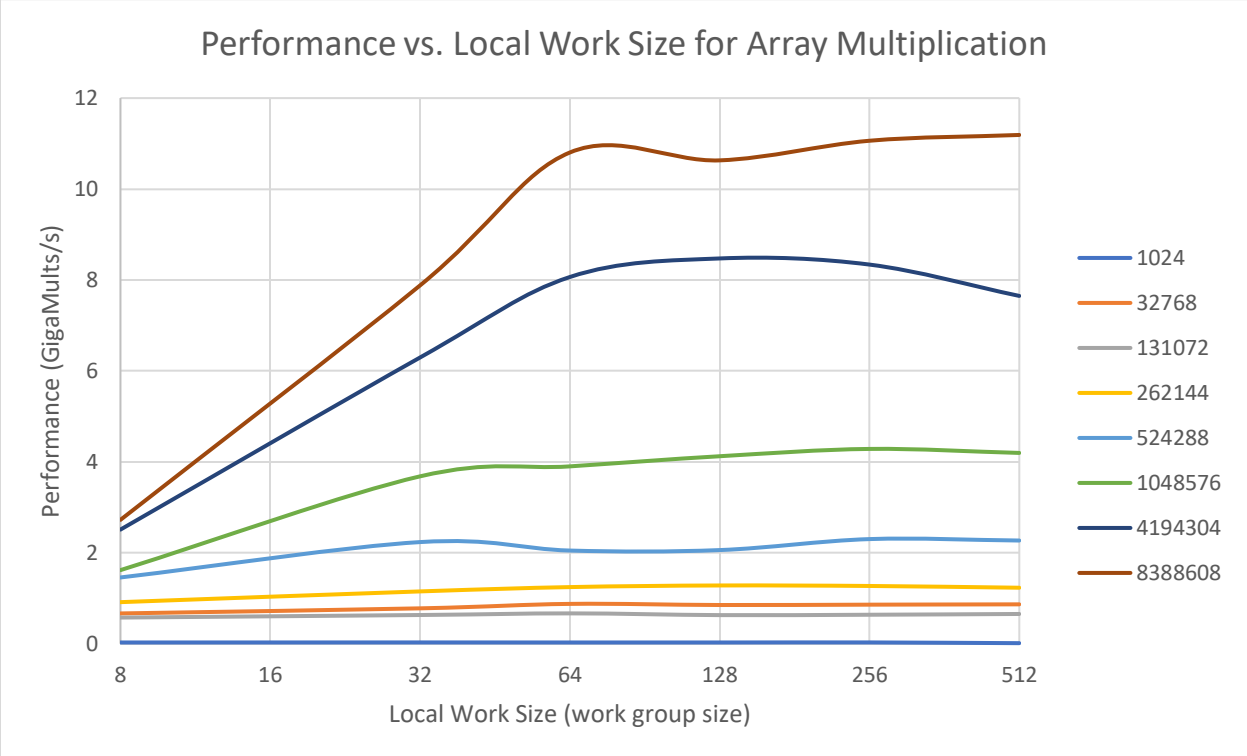
Graph 2



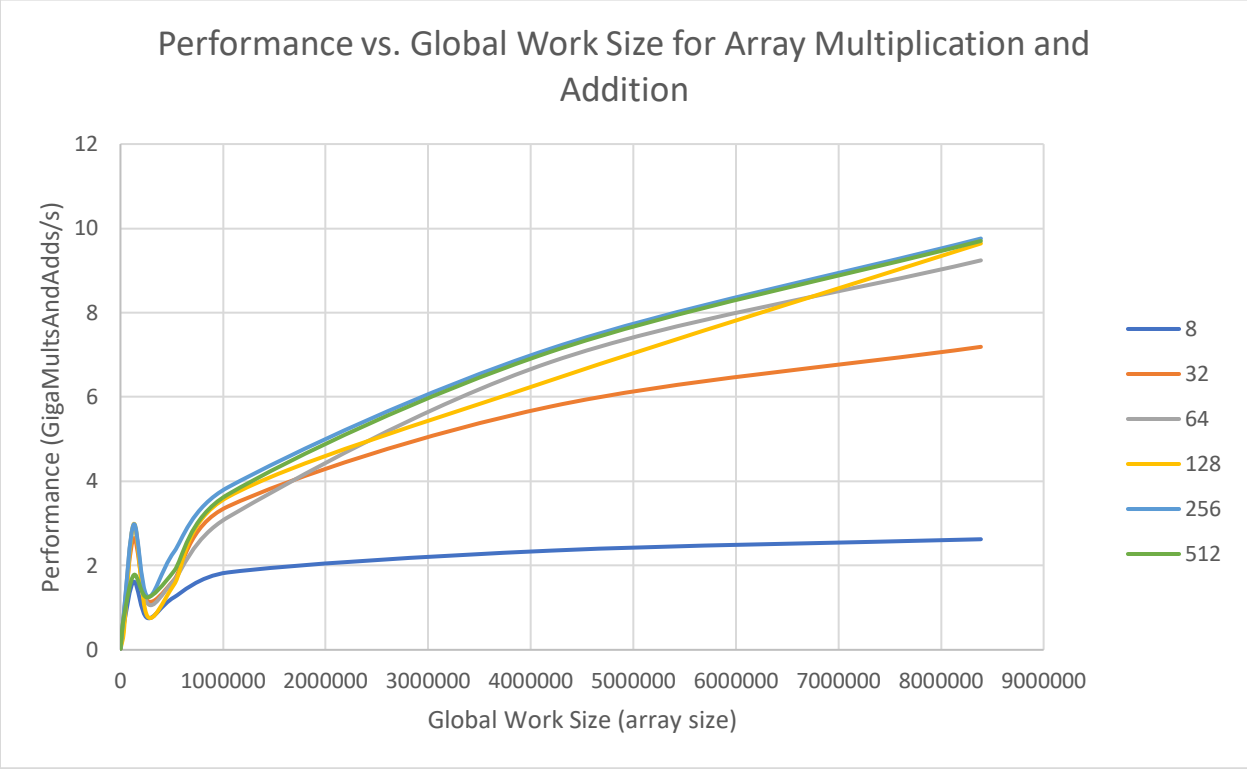
Graph 3



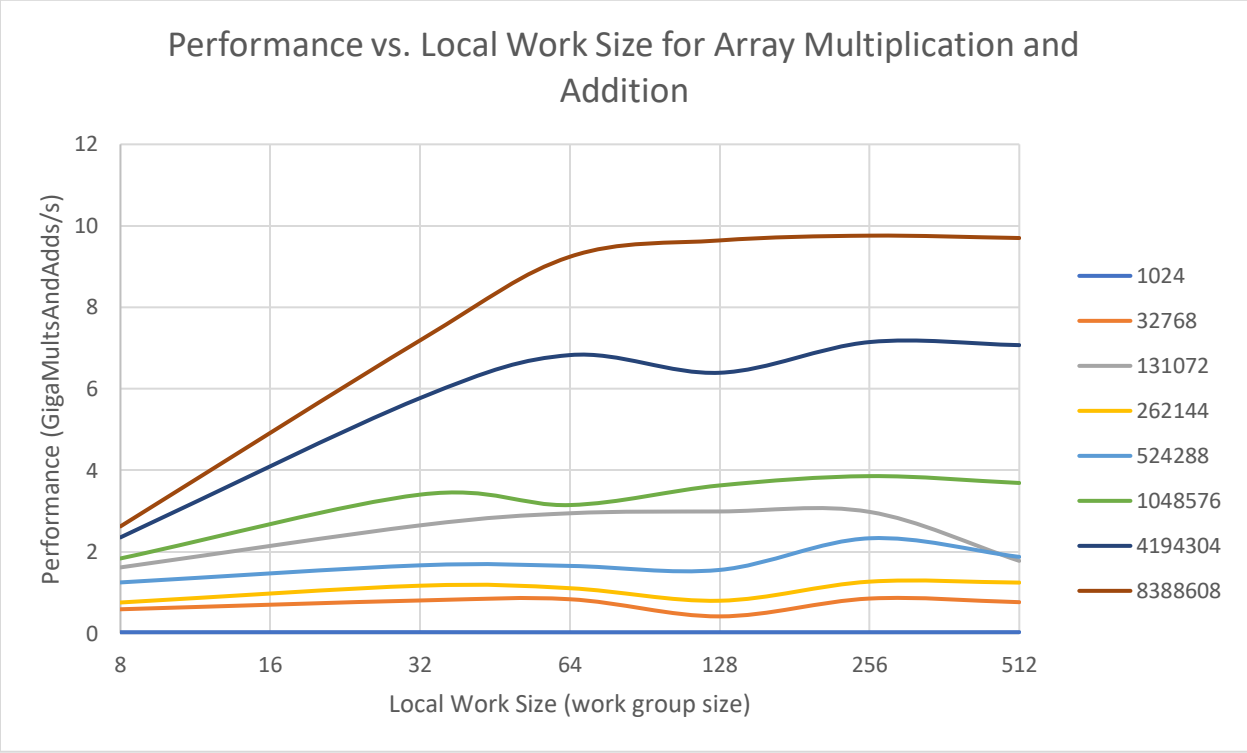
Graph 4



Graph 5



Graph 6



Graphs 1 and 2 have numbers with the word “add” appended to them. As mentioned before, these correspond to a set of performance measurements collected for the “multiply and add” section of the project. Again, the numbers without “add” appended correspond to just the “multiply” part of the project. For easier comparison, I also included separate graphs for both the Multiply and the Multiply-Add programs. Each program has a graph comparing performance vs. global work size and performance vs. local work size.

For Graph 1, I plotted performance based on changes in global work size. The curves represent performance measurements where local work size (work group size) is held constant while global work size is varied. By looking at these curves, we can tell that performance is generally increasing for each local work size. However, the rate of increase slowly decreases as global work size reaches multiple millions of elements. That is, the curves tend to flatten out more and more. Considering this trend, I would expect almost every curve to become nearly flat (constant) as global work size approaches infinity. The curves representing 8-element local sizes for both parts (array multiplication and array multiplication with addition) already start to demonstrate this within the global work size range. They flatten out and never show signs of increasing above 3 billion computations per second. For both parts, as we look at curves with higher and higher local work sizes, they generally take longer to flatten out with increases in global array size. In other words, it seems like the higher the local work size the longer it can maintain its rate of performance increase before becoming more constant.

Graph 2 shows similar trends in performance as local work size increases and global work size is held constant. However, increases in performance seem to taper off much quicker than when varying global work size. Additionally, the smaller the local work size the less effect increasing global work size has on performance. For example, the curves for 1024, 32768 and 262144 have very little, if any, increases in performance. But the curves for 4194304 and 8388608 increase almost 4-fold before eventually becoming constant. Therefore, considering the two graphs, it appears the global work size may have a little more effect on performance than local work size.

I believe the trends in these curves are directly caused by rapidly increasing global array size. Since we approached the maximum work-group size allowed by the system, more and more work groups were required as global array size became extremely large. Therefore, the system has to perform increasingly more computations with the same work-group sizes. This leads to performance tapering off as the immense size of the global array starts to counteract the growth in local work size. At the point where local work size no longer yields better performance, the only way to increase computations per second is to use a system that allows larger work-group sizes.

When comparing the Multiply and Multiply-Add performances on separate graphs, I noticed the Multiply curves perform better at almost every global work size increment. One explanation for this could be the number of mathematical procedures required for each part. The Multiply section only requires a multiplication of two numbers to complete a full computation. However, the Multiply-Add section requires the same multiplication and then an addition of a third number to the product to complete a full computation. This could easily explain why the Multiply program is able to perform more computations per second.

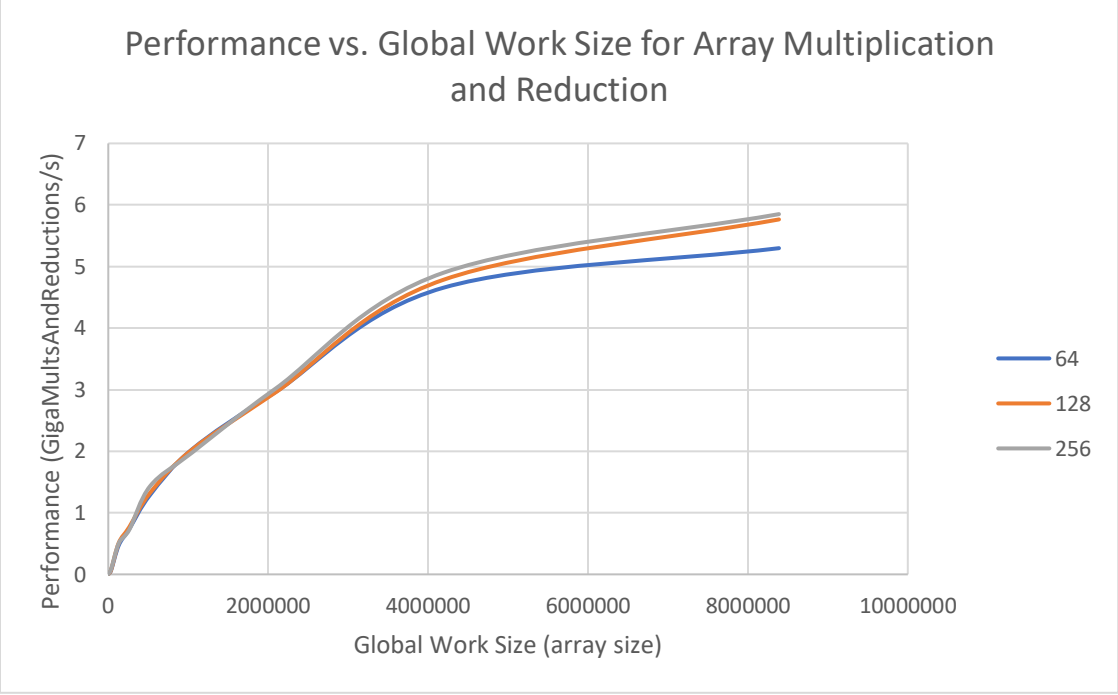
The performance difference between the two programs has some implications for the proper use of GPU parallel computing. It indicates that it is best used for simpler tasks and operations. The simpler the task, the better performance the program can yield. As tasks become more complex, performance increases will deplete because the total number of operations per computation increase. This is why the Multiply program performed better at every global work size. Furthermore, this intuitively means the best use of GPU parallel computing is completing simple tasks that have to be done thousands, or even millions, of times. Programs with these characteristics have the most efficiency to gain from utilizing GPU parallelism.

Raw Data for Multiply and Reduction

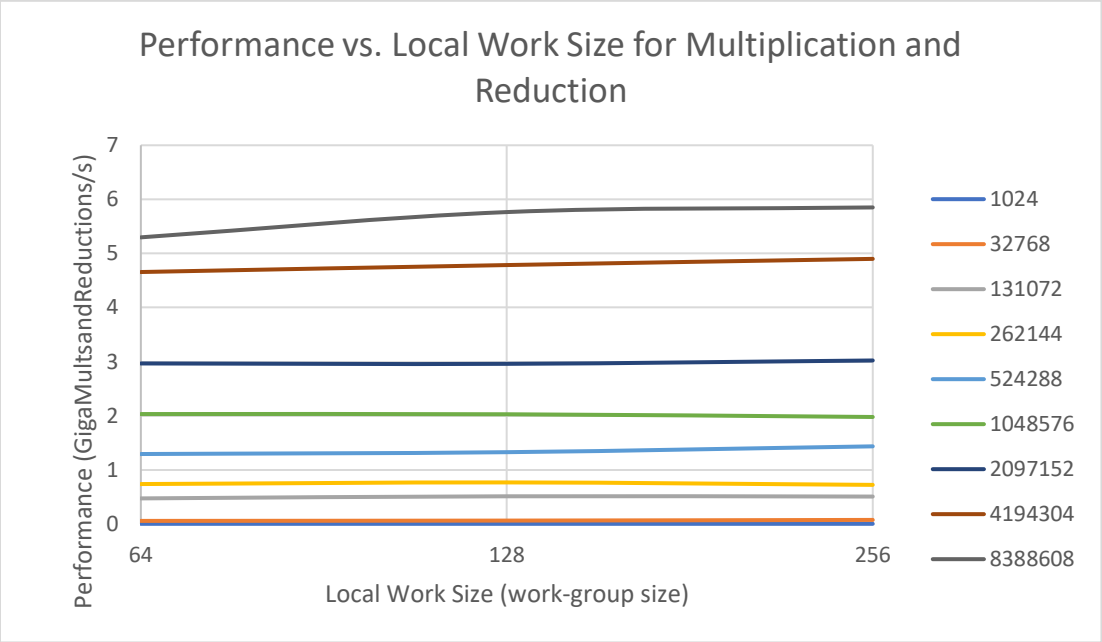
Local Work Size \ Global Work Size	1024	32768	131072	262144	524288	1048576	2097152	4194304	8388608
64	0.003	0.057	0.473	0.738	1.293	2.032	2.967	4.657	5.298
128	0.003	0.063	0.512	0.768	1.327	2.028	2.96	4.786	5.765
256	0.003	0.075	0.507	0.723	1.435	1.978	3.022	4.9	5.851

Graphs for Multiply and Reduction

Graph 7



Graph 8



In Graph 7, the trend seen in the curves is very similar to the trend in Graphs 1, 3 and 5. In general, the curves begin with aggressive increases in performance as global work size increases. Around the 1000000-element mark, the rate of increase begins to slow down and the curves begin to flatten out. The curves maintain this pace until about the 4000000-element size, where the rate of performance increase tapers again. At this point the curves are only experiencing marginal gains in performance as global work size increases. I would expect the curves to become nearly constant and flat as global size grows closer to infinity. This would depict a realistic maximum on performance regardless of any further increases in the size of the global dataset.

Again, I attribute the curves' trend directly to the quick growth of the global work size. I believe the rate of performance increase fades because the work-group size is held constant, causing the program to create increasingly more work groups. As global work size gets extremely large, the advantages of large work groups is counteracted, thus making performance increases unattainable. Therefore, the only way to be reap more performance benefits, we would have to find a way to further increase work-group size.

In addition to what I mentioned earlier about the proper use of GPU parallelization, there are a few other aspects to consider. Dataset size and how the data will be grouped and organized are important components in effective parallelization. By looking at Graph 7, we can see that despite increasing local work size, dataset size eventually overshadows it and performance plateaus. Therefore, the developer should always be aware of how large the dataset he's dealing with is. Doing this will allow him to choose an appropriate work group size considering the system he's using. Some other things to consider the use of registers and actually remembering to set the global and local work sizes before including GPU parallelization into a program.