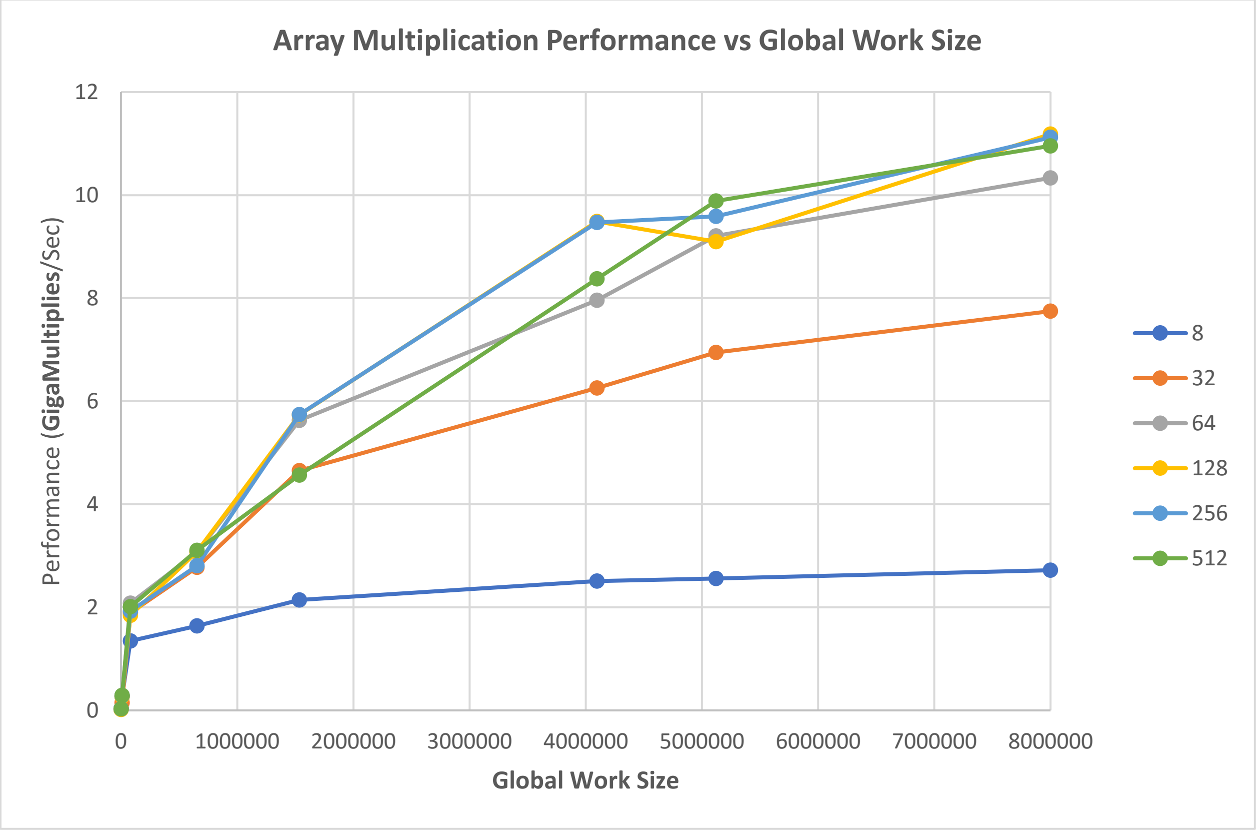
Amy Stockinger (stockina@oregonstate.edu)
CS475 – Project 05 – OpenCL Multiplication and Reduction
Due: 5/20/2019

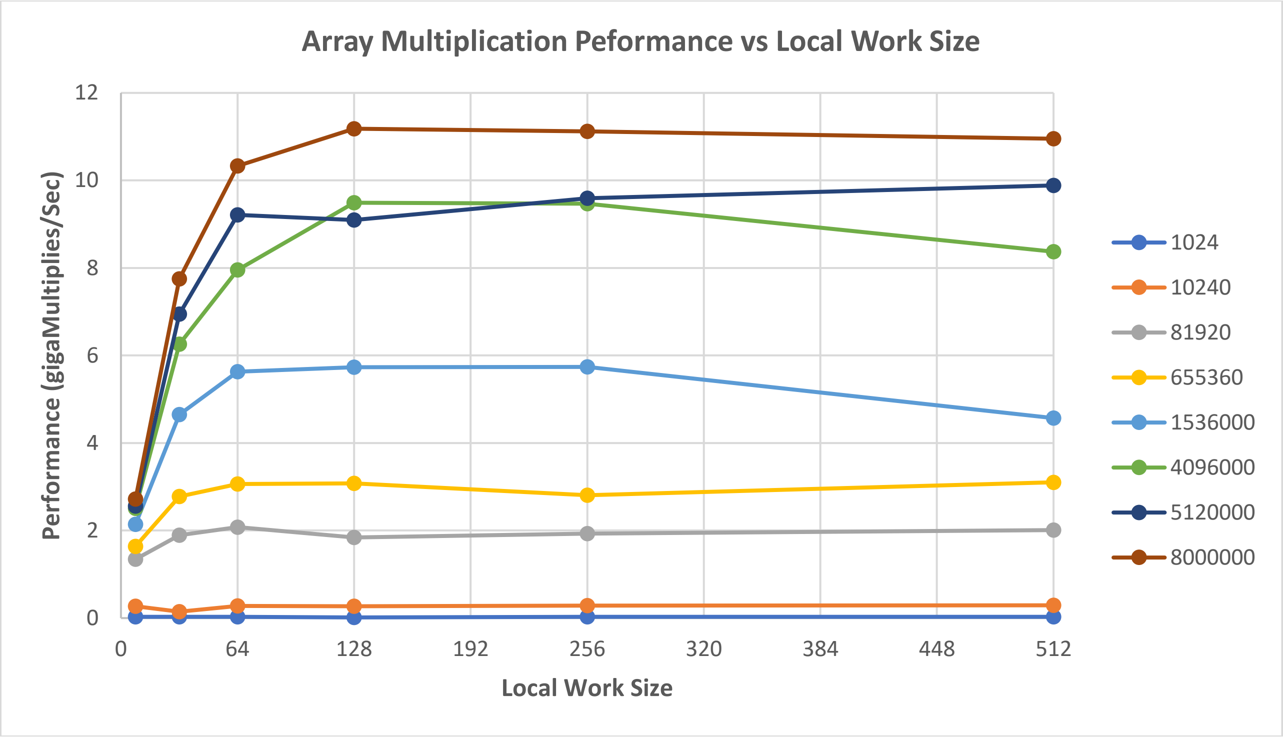
I ran these programs on *rabbit* using g++. Below are my results:

ArrayMult:

Multiply Performance (GigaMultiplies/Sec)

		Local Work Size					
		8	32	64	128	256	512
Global Work Size	1024	0.029	0.027	0.027	0.015	0.029	0.029
	10240	0.268	0.146	0.277	0.268	0.282	0.292
	81920	1.345	1.894	2.079	1.844	1.927	2.011
	655360	1.64	2.779	3.067	3.081	2.809	3.103
	1536000	2.144	4.652	5.631	5.735	5.739	4.569
	4096000	2.511	6.259	7.958	9.49	9.47	8.373
	5120000	2.557	6.944	9.209	9.093	9.59	9.886
	8000000	2.72	7.747	10.334	11.182	11.119	10.956





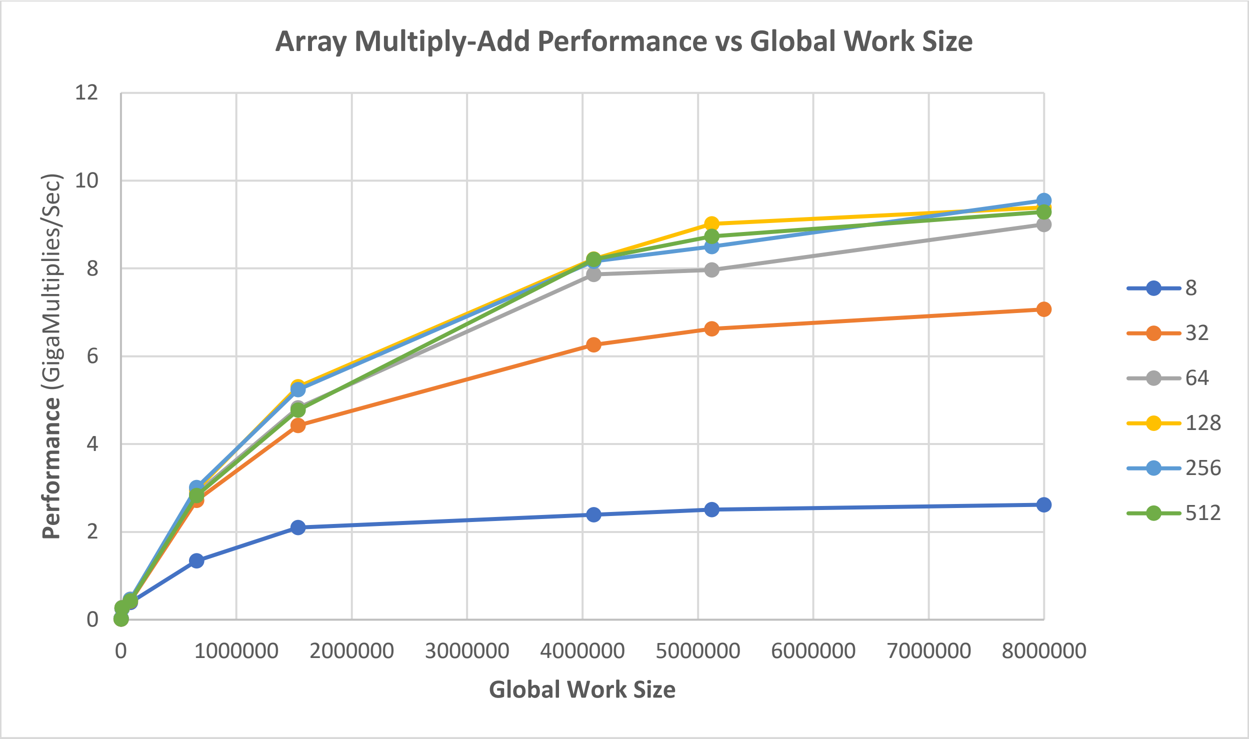
From these performance curves, we can see that performance steadies around a local size of 128. In the first graph, the performances begin to run together starting at the local size of 64, and this is also shown in the second graph as values begin to level off at local size 64 before steadying at 128.

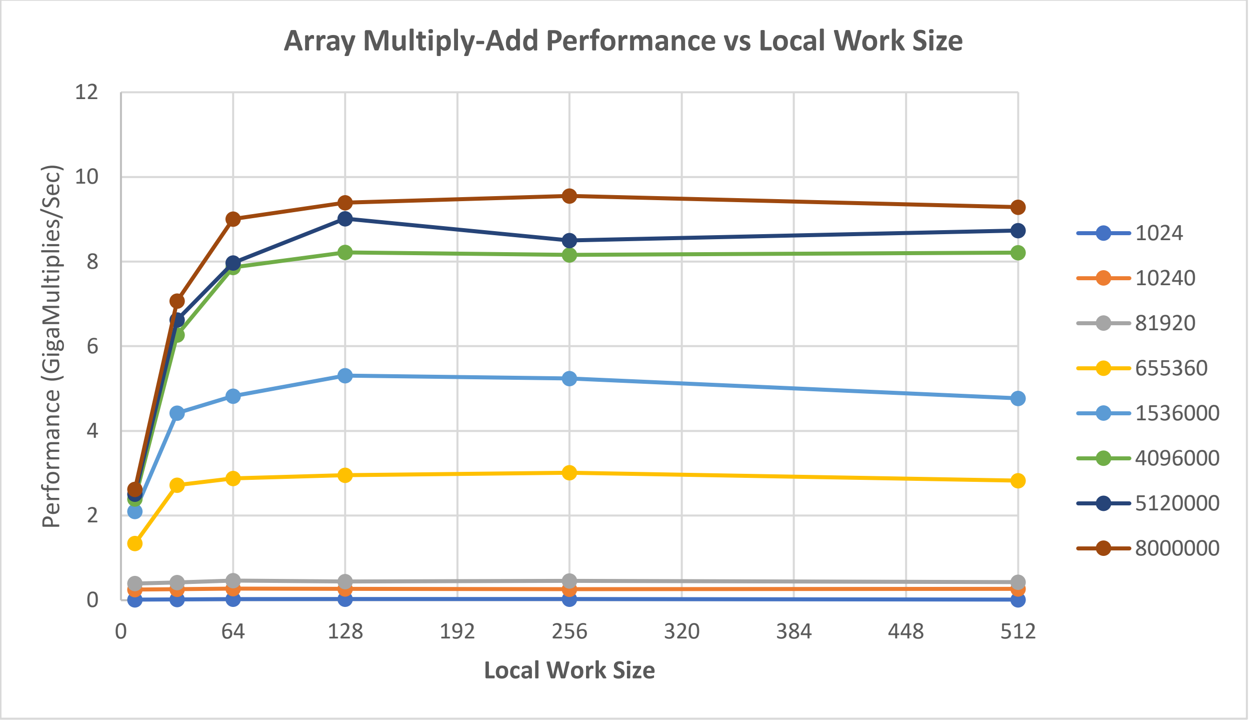
The first graph also shows that performance increases with global work size, but more dramatic performances occur with larger local work sizes. The local size of 8 shows a much slower performance increase than local sizes 256 or 512.

ArrayMultSum:

Multiply-Add Performance (GigaMultiplies/Sec)

		Local Work Size					
		8	32	64	128	256	512
Global Work Size	1024	0.014	0.015	0.027	0.028	0.028	0.014
	10240	0.252	0.257	0.274	0.265	0.257	0.268
	81920	0.394	0.42	0.464	0.442	0.459	0.43
	655360	1.34	2.719	2.877	2.957	3.011	2.827
	1536000	2.1	4.421	4.823	5.306	5.24	4.773
	4096000	2.394	6.265	7.865	8.216	8.162	8.211
	5120000	2.511	6.623	7.968	9.015	8.501	8.731
	8000000	2.619	7.068	9.005	9.391	9.551	9.289





These curves are a little neater than the Array-Mult, but they depict similar results where a larger global work size has higher performance. They show the same leveling-off of performance between the 64 and 128 local size mark. In the first graph, the lines run closer together at local size 64, and in the second graph the performance lines begin to level off at 64.

ArrayMult and ArrayMultSum Commentary:

As stated under the graphs, the graphs do show similar behaviors. The performances level off as local work sizes reach 128, but we do see performance increases as global work size increases. This means that performance is not dependent on local work size as much as global work size.

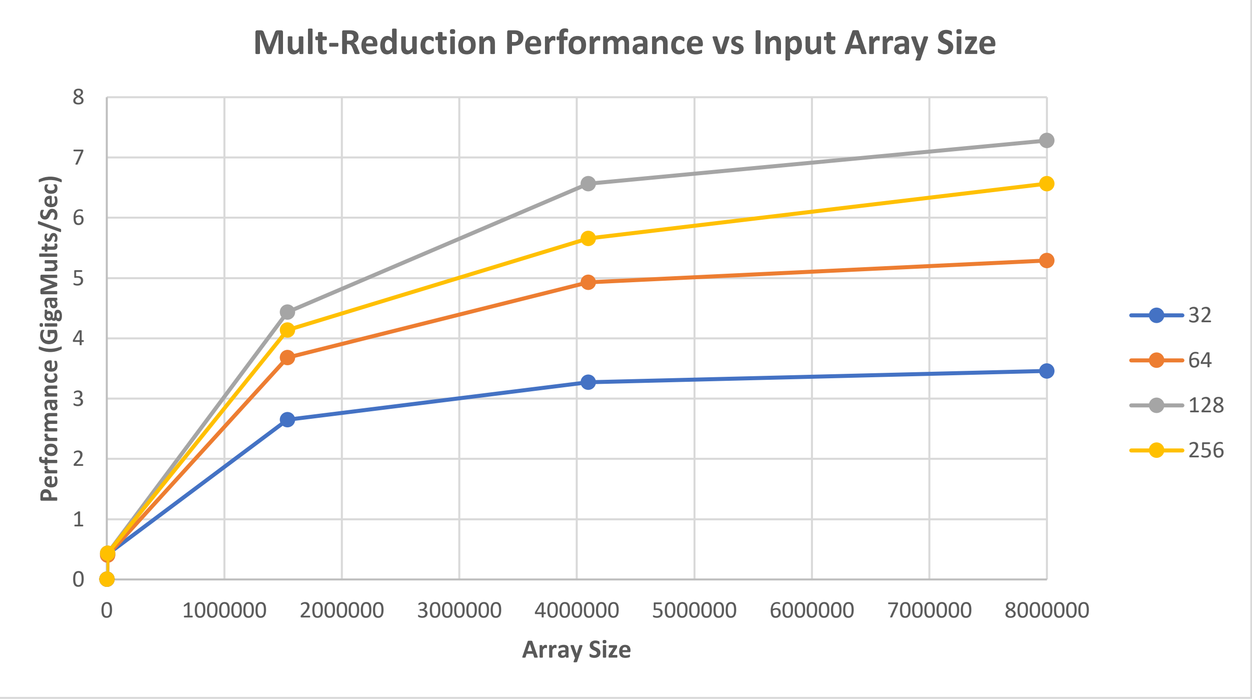
The performances differences are better for the ArrayMult (the first set) of graphs. I left the same scale on the ArrayMultSum graphs to show that it does not quite reach 10 GigaMults/sec with the same work sizes, unlike ArrayMult which reaches around 11.5 at the highest.

What does this mean for proper use of GPU parallel computing? It is better to have larger local work sizes, especially as global work sizes increase.

ArrayMultReduce:

Multiply-Reduce Performance (GigaMults/Sec)

		Local Work Size			
		32	64	128	256
Input Array Size	1024	0.006	0.006	0.006	0.006
	8192	0.416	0.403	0.437	0.437
	1536000	2.651	3.683	4.435	4.135
	4096000	3.269	4.931	6.564	5.656
	8000000	3.459	5.291	7.281	6.566



From this graph, we can see that performance increases with array size. The graph shown on the lecture slides would related to the blue line on my graph, which seems to fit given that the array sizes on that graph were much larger. Interestingly, the local work size of 128 clearly is the top performer, not 256. It appears that 128 is a sweet spot, and a local size of 256 puts too many threads on a single block, so they are all swapping against each other.

What does this mean for the proper use of GPU parallel computing? At a certain point, it is better to spread threads out onto more blocks or else performance will begin to drop.