Oregon State University

CS 475 - Spring 2021

Professor

Mike Bailey

mjb@cs.oregonstate.edu

Project #6

OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce

by

Abdullah Saydemir

saydemia@oregonstate.edu

May 27, 2021

Setup:

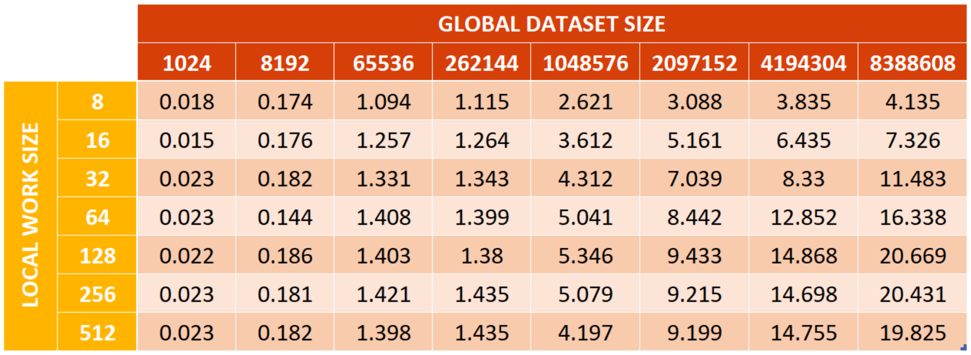
I used the *DGX* servers for this experiment. I don't think we need an uptime command output since *DGX* allocates resources dedicated to me only during the run.

GLOBAL DATASET SIZE = [1024, 8192, 65536, 262144, 1048576, 2097152, 4194304, 8388608]
LOCAL WORK SIZE = [8, 16, 32, 64, 128, 256, 512]

GigaMultiplies / Second = How many billions of multiplication operations are executed in a second.

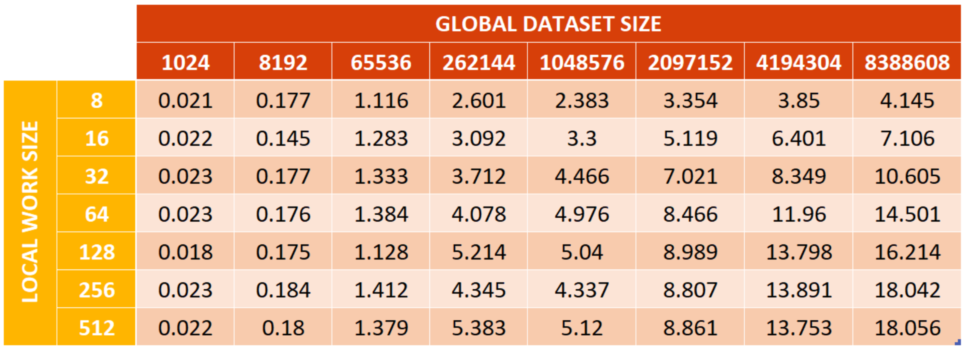
GigaMultiply-Adds / Second = How many billions of multiplication and addition operations are executed in a second.

Multiply Performance

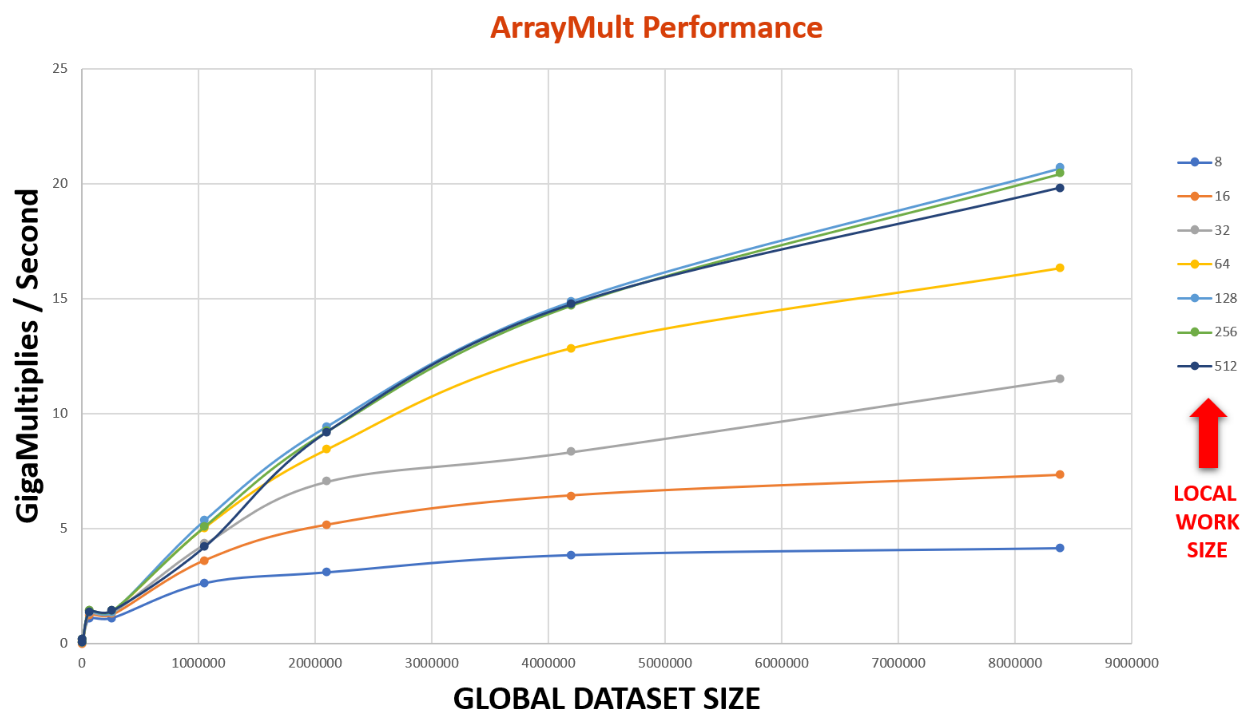


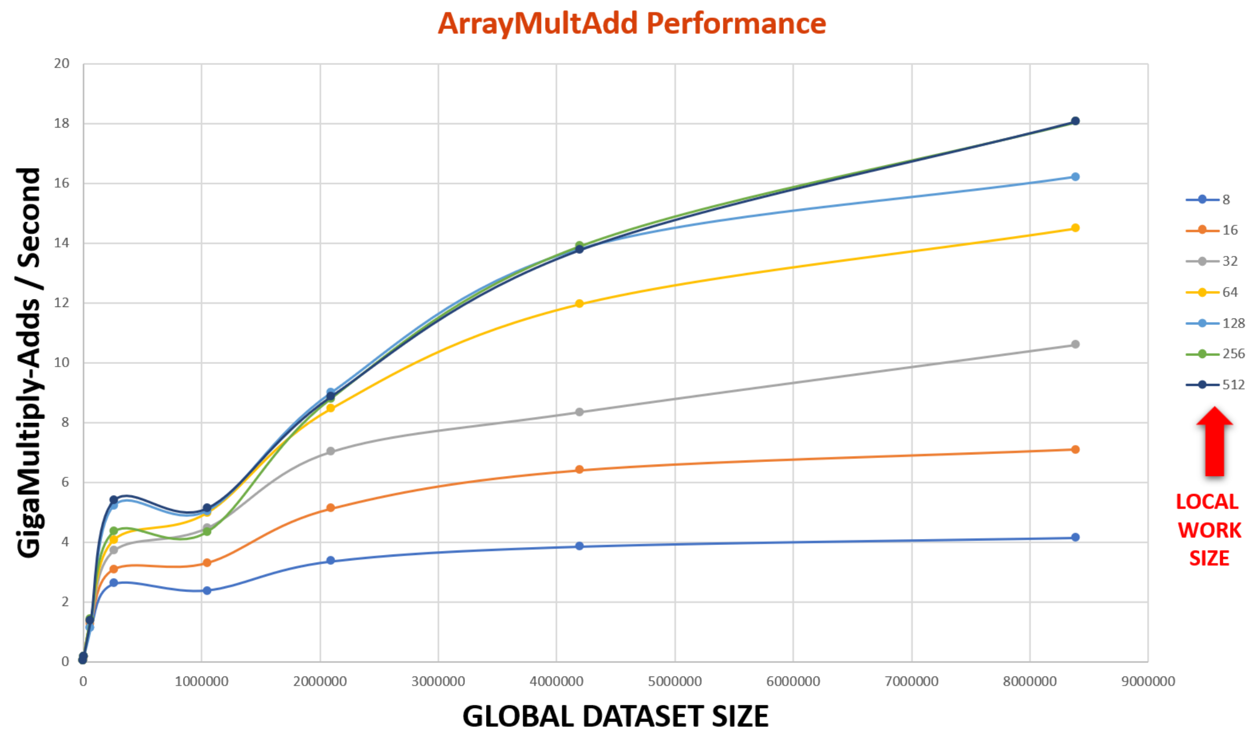
		GLOBAL DATASET SIZE							
		1024	8192	65536	262144	1048576	2097152	4194304	8388608
LOCAL WORK SIZE	8	0.018	0.174	1.094	1.115	2.621	3.088	3.835	4.135
	16	0.015	0.176	1.257	1.264	3.612	5.161	6.435	7.326
	32	0.023	0.182	1.331	1.343	4.312	7.039	8.33	11.483
	64	0.023	0.144	1.408	1.399	5.041	8.442	12.852	16.338
	128	0.022	0.186	1.403	1.38	5.346	9.433	14.868	20.669
	256	0.023	0.181	1.421	1.435	5.079	9.215	14.698	20.431
	512	0.023	0.182	1.398	1.435	4.197	9.199	14.755	19.825

Multiply-Add Performance



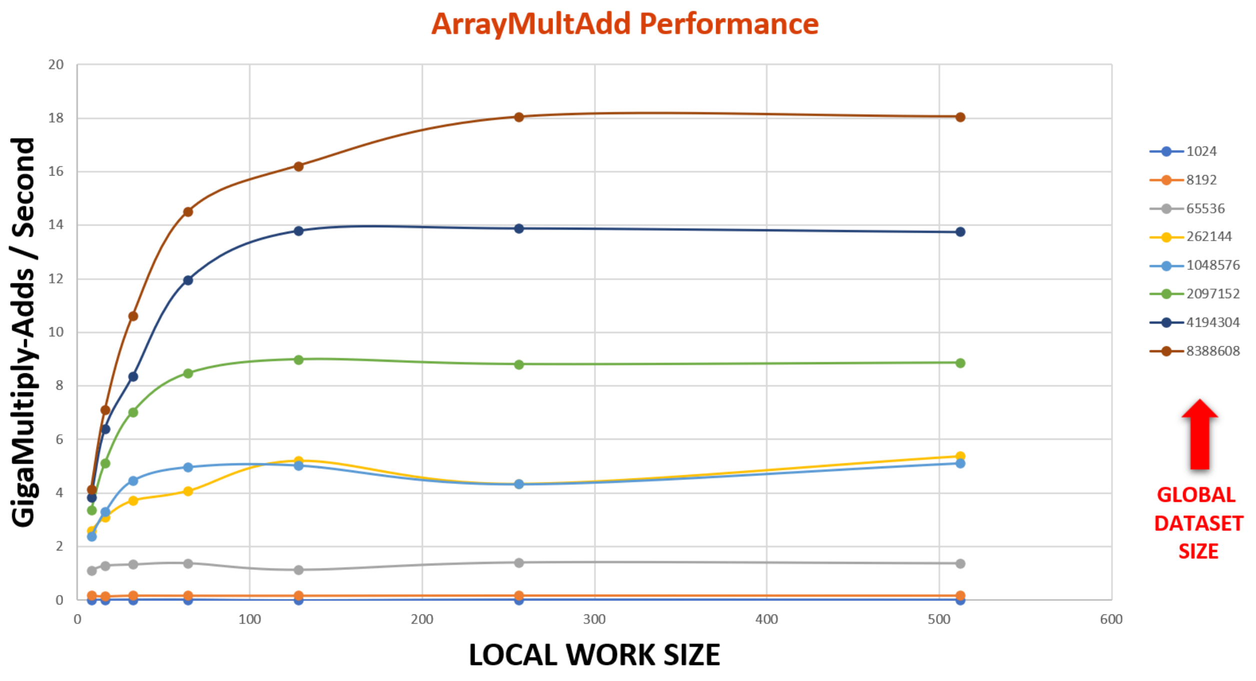
		GLOBAL DATASET SIZE							
		1024	8192	65536	262144	1048576	2097152	4194304	8388608
LOCAL WORK SIZE	8	0.021	0.177	1.116	2.601	2.383	3.354	3.85	4.145
	16	0.022	0.145	1.283	3.092	3.3	5.119	6.401	7.106
	32	0.023	0.177	1.333	3.712	4.466	7.021	8.349	10.605
	64	0.023	0.176	1.384	4.078	4.976	8.466	11.96	14.501
	128	0.018	0.175	1.128	5.214	5.04	8.989	13.798	16.214
	256	0.023	0.184	1.412	4.345	4.337	8.807	13.891	18.042
	512	0.022	0.18	1.379	5.383	5.12	8.861	13.753	18.056

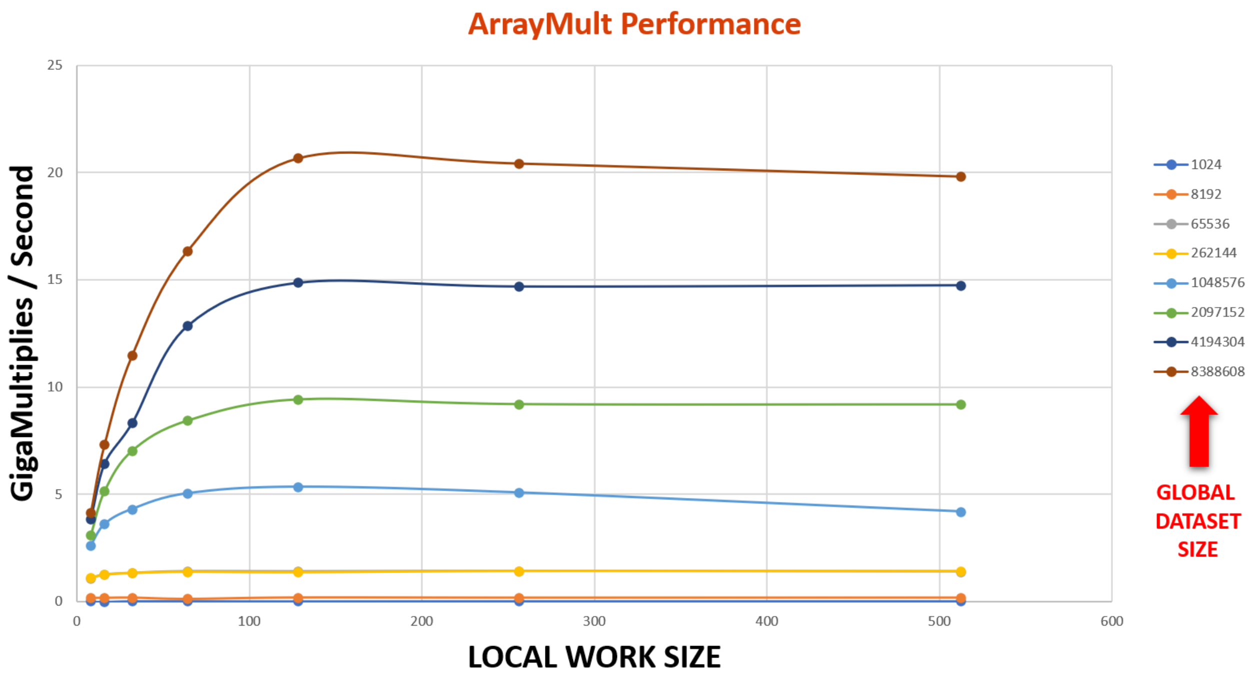




Performance graphs for the two functions are similar. That's why I put them back-to-back.

For both graphs, the performance is very low for smaller dataset sizes. Especially for 1K there is really no difference between different local work sizes. However, as the dataset size increases, first, the performance increases and second, the performance between different work sizes increases. After some point, the performance flattens out because the warps are saturated. There is a dip on the left side of both graphs but I don't have any reasoning for that. I think it should not be there.





The first thing I noticed is for small global dataset sizes, providing more local work size does not increase the performance. Because, when there is not much work to do, some cores just sit around and do nothing. For larger global dataset sizes, performance increases until local work size is 128 and after that it continues full flat because the cores are saturated. Also, the performance for ArrayMultAdd is little lower than the performance for ArrayMult, since we are doing additional work.

What is the performance difference between doing a Multiply and doing a Multiply-Add?

Multiply-Add has 10% lower performance than just Multiply. Because, we are doing additional work. It should have been lower but probably there is some optimization going on in the background.

What does that mean for the proper use of GPU parallel computing?

Yeah. So, parallel computing is good for large dataset sizes. If you have a 1K array to multiply just use SIMD and CPU. It is not worth passing the data to the GPU then calculating and putting it back. On the other hand, if you have a really big dataset size GPUs are experts on that.

Multiply-Reduce

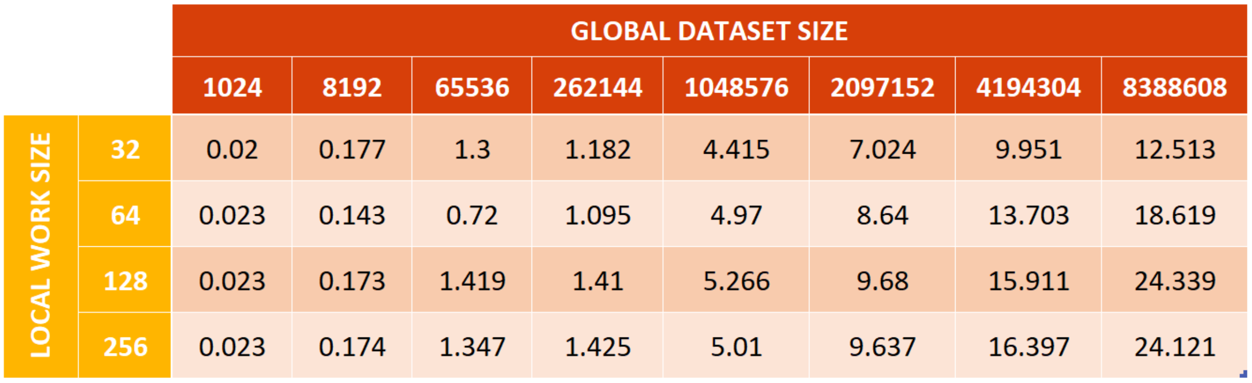
Setup:

I used the *DGX* servers for this experiment. I don't think we need an uptime command output since *DGX* allocates resources dedicated to me only during the run.

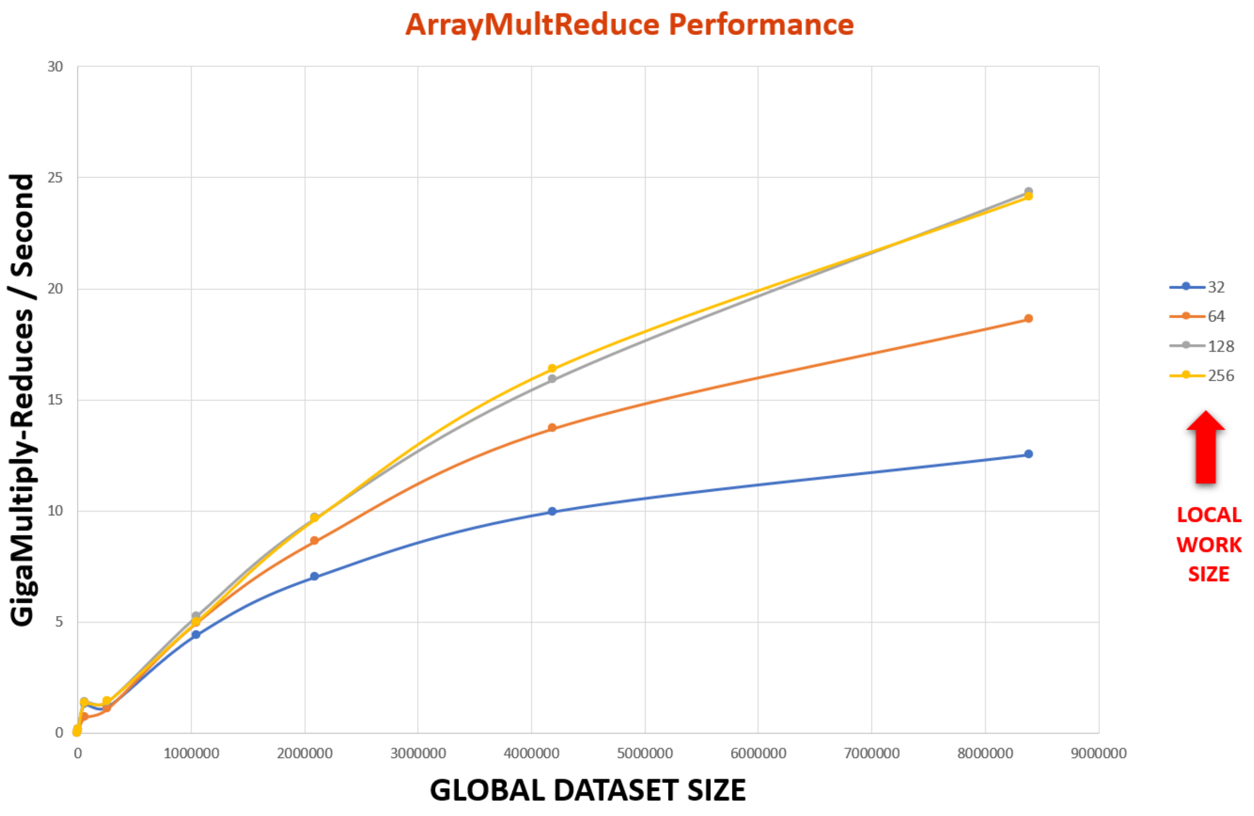
GLOBAL DATASET SIZE = [1024, 8192, 65536, 262144, 1048576, 2097152, 4194304, 8388608]
LOCAL WORK SIZE = [32, 64, 128, 256]

GigaMultiply-Reduces / Second = How many billions of multiplication and reduction operations are executed in a second.

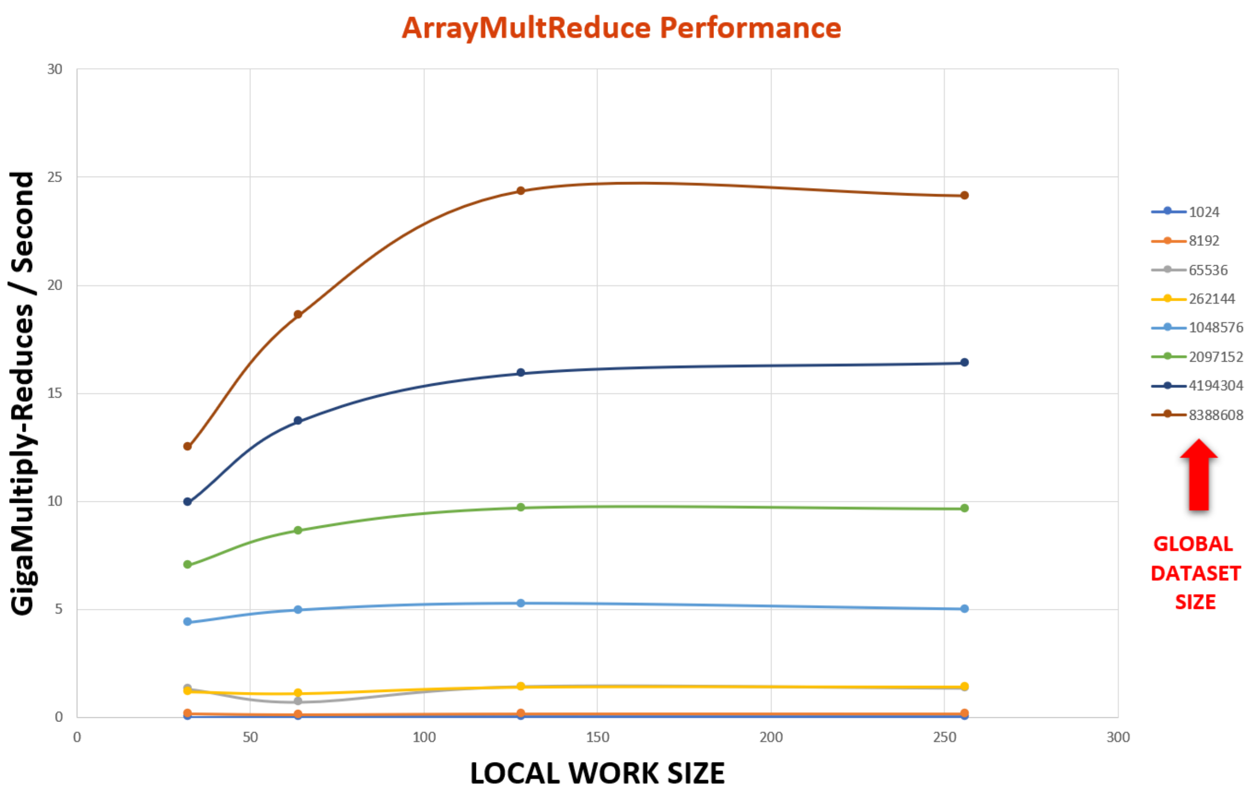
Multiply-Reduce Performance



		GLOBAL DATASET SIZE							
		1024	8192	65536	262144	1048576	2097152	4194304	8388608
LOCAL WORK SIZE	32	0.02	0.177	1.3	1.182	4.415	7.024	9.951	12.513
	64	0.023	0.143	0.72	1.095	4.97	8.64	13.703	18.619
	128	0.023	0.173	1.419	1.41	5.266	9.68	15.911	24.339
	256	0.023	0.174	1.347	1.425	5.01	9.637	16.397	24.121



This graph is really similar to Multiply and Multiply-Add except that the performance is most of the time better than both of them. This is weird because we are doing extra work. One thing is that reduce operation is done in the local array before data is passed back. This might have increased the performance. Other than that, performance is lower for small global dataset sizes and there is not much difference between different local work sizes. As the global dataset size increases both the performance difference between different local work sizes and the performance itself increase. For 128 and 256 work sizes, there is not any difference at all.



As before, for small global dataset sizes, providing more local work size does not increase the performance. Because, when there is not much work to do, some cores just sit around and do nothing. For larger global dataset sizes, performance increases until local work size is 128 and after that it continues full flat because the cores are saturated.

What does that mean for the proper use of GPU parallel computing?

Passing data from/to global memory takes time. That means, if you can do your work in local memory without accessing global memory, do it without accessing global memory.

Yet again, parallel computing is good for large dataset sizes. If you have a small number of elements just use SIMD and CPU. If you have a really big number of elements then use GPUs.