Project #6 OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce

Erick Branner, brannere@oregonstate.edu | 29 May 2022

First, work on the Array Multiply and the Array Multiply-Add portions:

What machine you ran this on

DGX System

Show the tables and graphs

Tables

Multiply

GLOBAL DATA SET SIZE (NUM_ELEMENTS)	LOCAL_SIZE	NUM_WORK_GROUPS	GigaMultsPerSecond
1024	8	128	0.02
1024	16	64	0.017
1024	32	32	0.022
1024	64	16	0.024
1024	128	8	0.023
1024	256	4	0.023
1024	512	2	0.023
524288	8	65536	1.639
524288	16	32768	1.943
524288	32	16384	2.23
524288	64	8192	0.347
524288	128	4096	2.425
524288	256	2048	2.47
524288	512	1024	2.55
1048576	8	131072	2.42
1048576	16	65536	3.21
1048576	32	32768	3.895
1048576	64	16384	4.42

1048576	128	8192	0.652
1048576	256	4096	4.421
1048576	512	2048	4.54
2097152	8	262144	3.152
2097152	16	131072	4.766
2097152	32	65536	6.483
2097152	64	32768	7.75
2097152	128	16384	8.337
2097152	256	8192	8.275
2097152	512	4096	8.022
4194304	8	524288	1.797
4194304	16	262144	6.124
4194304	32	131072	8.861
4194304	64	65536	3.127
4194304	128	32768	13.32
4194304	256	16384	13.193
4194304	512	8192	13.632
8388608	8	1048576	4.121
8388608	16	524288	7.29
8388608	32	262144	4.302
8388608	64	131072	16.249
8388608	128	65536	20.015
8388608	256	32768	18.122
8388608	512	16384	21.092

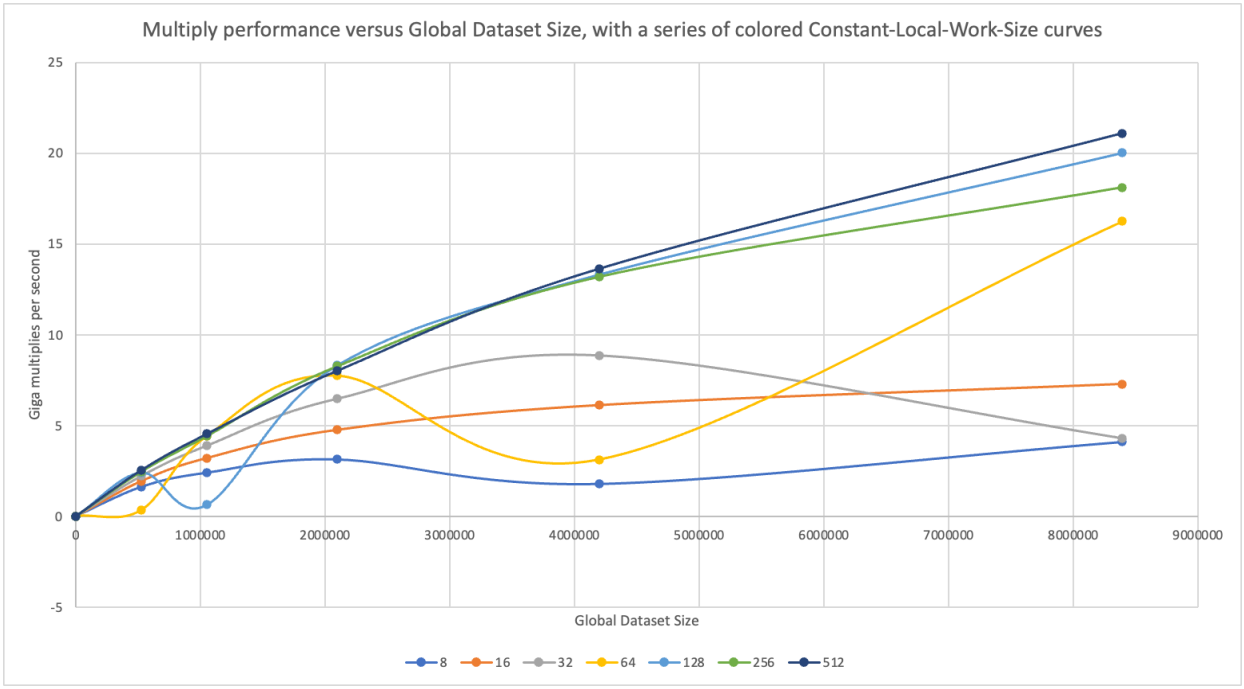
Multiply-Add

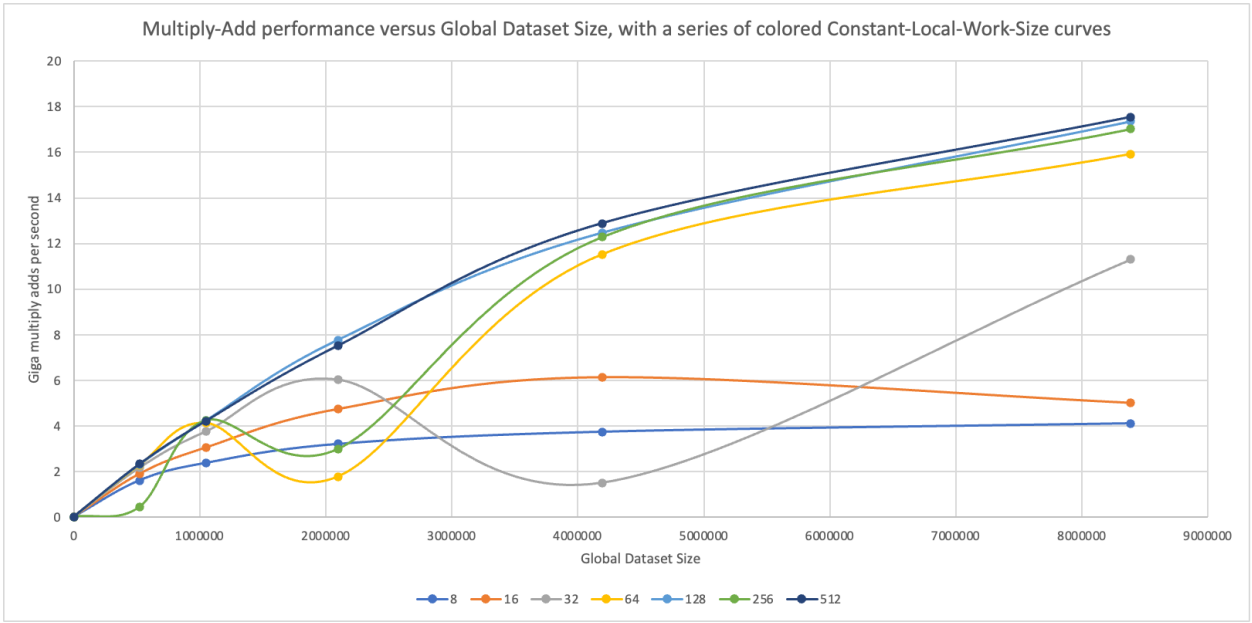
GLOBAL DATASET SIZE (NUM_ELEMENTS)	**LOCAL_SIZE**	**NUM_WORK_GROUPS**	**GigaMultAddsPerSecond**
1024	8	128	0.021
1024	16	64	0.023

1024	32	32	0.022
1024	64	16	0.022
1024	128	8	0.019
1024	256	4	0.023
1024	512	2	0.023
524288	8	65536	1.622
524288	16	32768	1.913
524288	32	16384	2.176
524288	64	8192	2.28
524288	128	4096	2.315
524288	256	2048	0.461
524288	512	1024	2.336
1048576	8	131072	2.384
1048576	16	65536	3.058
1048576	32	32768	3.758
1048576	64	16384	4.131
1048576	128	8192	4.228
1048576	256	4096	4.254
1048576	512	2048	4.212
2097152	8	262144	3.216
2097152	16	131072	4.748
2097152	32	65536	6.038
2097152	64	32768	1.783
2097152	128	16384	7.771
2097152	256	8192	2.994
2097152	512	4096	7.515
4194304	8	524288	3.752
4194304	16	262144	6.146
4194304	32	131072	1.514

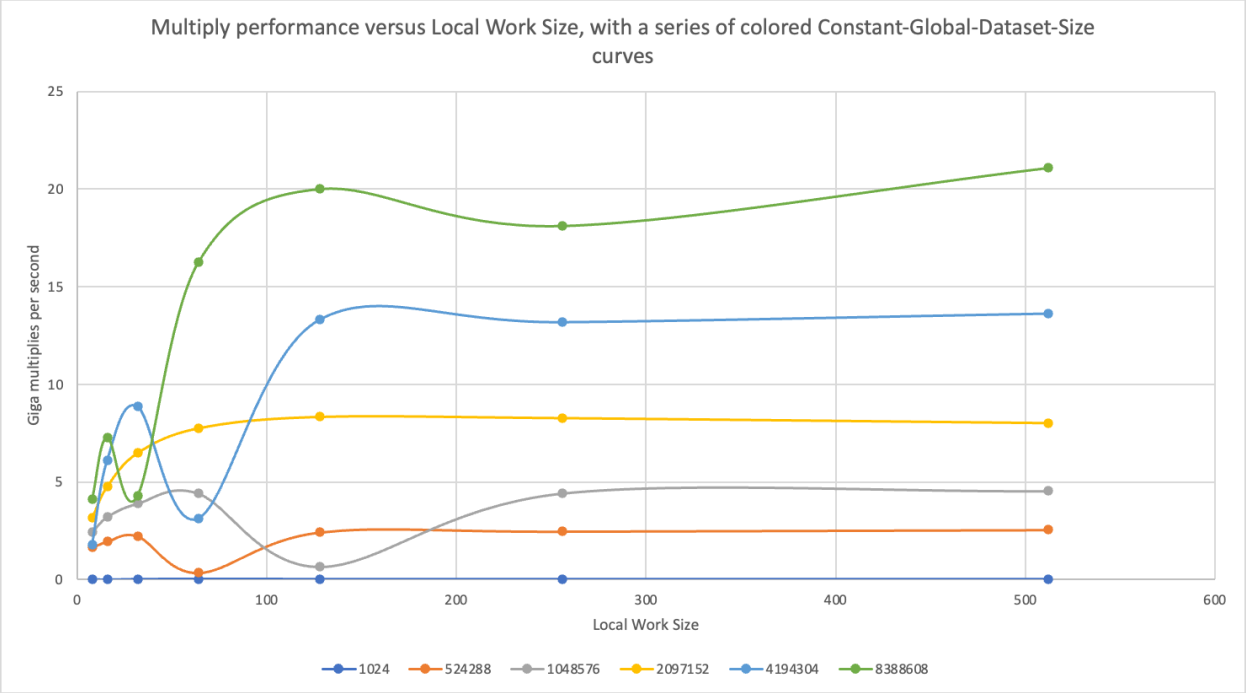
4194304	64	65536	11.521
4194304	128	32768	12.476
4194304	256	16384	12.28
4194304	512	8192	12.885
8388608	8	1048576	4.123
8388608	16	524288	5.02
8388608	32	262144	11.305
8388608	64	131072	15.921
8388608	128	65536	17.357
8388608	256	32768	17.015
8388608	512	16384	17.54

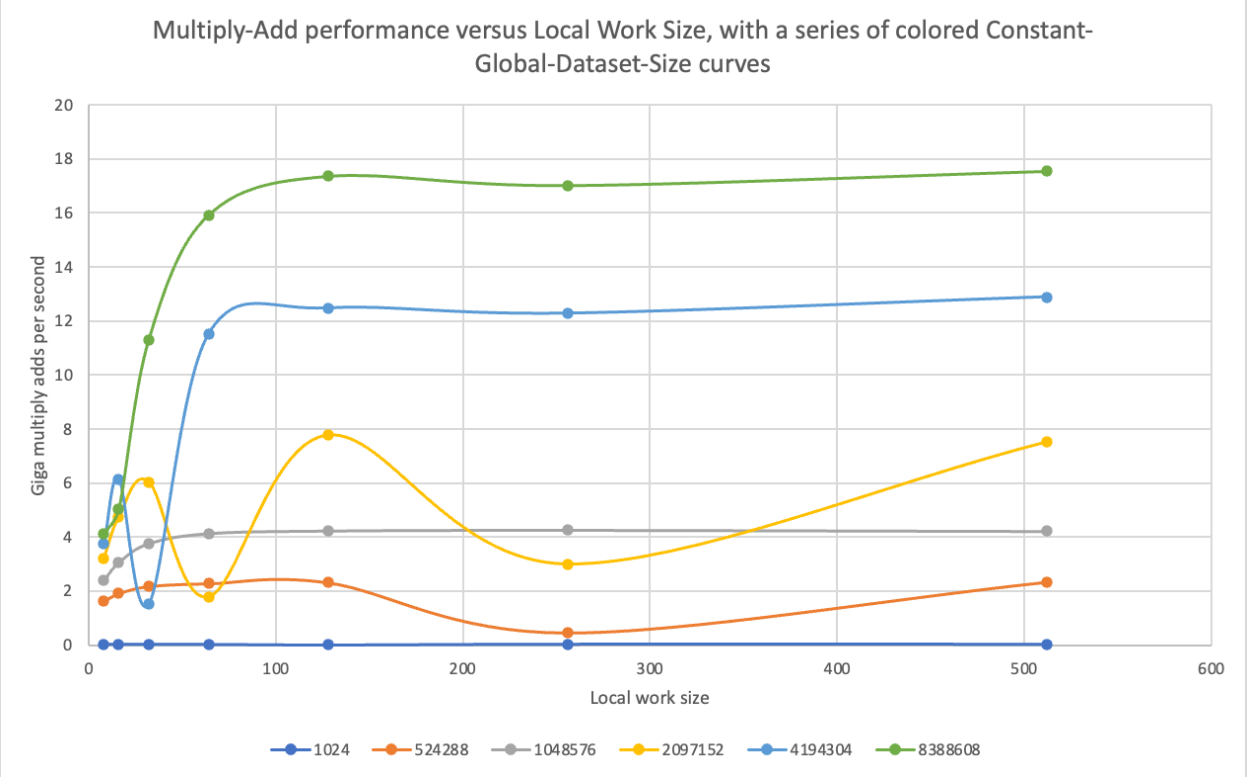
Multiply and Multiply-Add performance versus Global Dataset Size, with a series of colored Constant-Local-Work-Size curves





Multiply and Multiply-Add performance versus Local Work Size, with a series of colored Constant-Global-Dataset-Size curves





What patterns are you seeing in the performance curves?

Performance increases with local work size and larger global data set sizes.

Why do you think the patterns look this way?

With larger local work group sizes, more data can be worked on at the same time. It makes sense that the performance increases with larger data sizes.

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

Multiply-Add is a little bit slower than just Multiply. There is a fused multiply-add (for D = A + (B*C)) instruction that the compiler is taking advantage of – makes performance a little better.

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

If you have more instructions/work to be done, your performance will be slower. Larger local sizes with large data set size will help maximize your performance.

Then, write another version of the code that turns it into a Multiply+Reduce application.

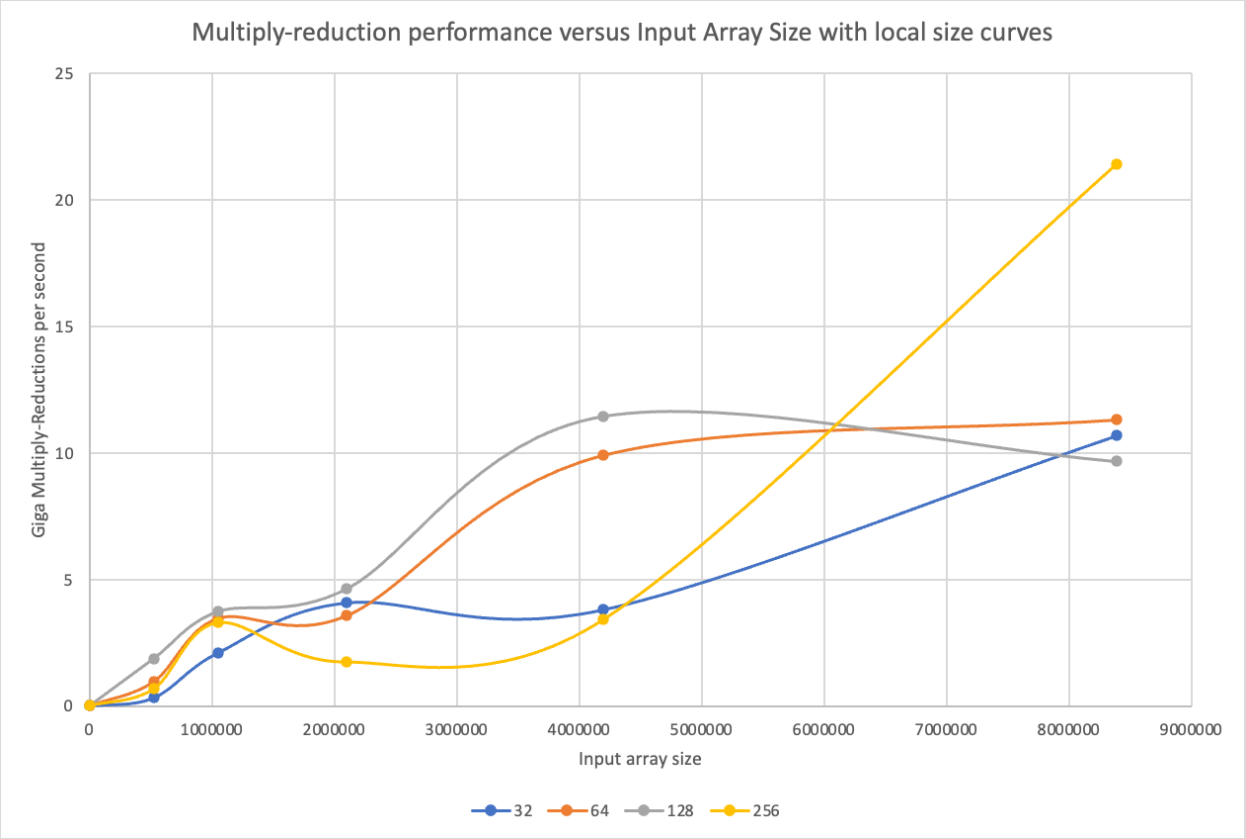
Show this table and graph

Table

GLOBAL DATA SET SIZE (NUM_ELEMENTS)	LOCAL_SIZE	NUM_WORK_GROUPS	GigaMultiply-Reductions Per Second
1024	32	32	0.017
1024	64	16	0.016
1024	128	8	0.02
1024	256	4	0.016
524288	32	16384	0.334
524288	64	8192	0.969
524288	128	4096	1.885
524288	256	2048	0.684
1048576	32	32768	2.107
1048576	64	16384	3.473
1048576	128	8192	3.743
1048576	256	4096	3.3
2097152	32	65536	4.09
2097152	64	32768	3.572
2097152	128	16384	4.633
2097152	256	8192	1.749
4194304	32	131072	3.815
4194304	64	65536	9.923

4194304	128	32768	11.457
4194304	256	16384	3.419
8388608	32	262144	10.69
8388608	64	131072	11.329
8388608	128	65536	9.67
8388608	256	32768	21.422

Graph



What pattern are you seeing in this performance curve?

The performance increases with larger array sizes using a local size of 256 – performance is much greater compared to other local sizes. With local size of 256 the performance is greater than the same array size for multiply and multiply-add (don't know why – tried re-running on dgx and got worse numbers). Performance is overall less than Multiply and MultiplyAdd.

Why do you think the pattern looks this way?

When using reduction, there are less steps needed to sum all the values ($\log_2$(num items)). So for a size of 256, there are only 8 steps needed to sum all values in the array across all the work groups – $\log_2(256)$. There's still extra work to be done which is why the permanence is a bit slower than the Multiply and MultiplyAdd

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

Maximize local work group size and data set size for better performance.