Lachlan Sinclair

sinclala@oregonstate.edu

5/28/2020

CS 475 Spring

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

System: For this project I used the DGX system. I generated a script that compiles and runs the three separate programs for the three different portions of the assignment. These sections are first.cpp for Array Multiply, second.cpp for Multiply-Add and third.cpp for Multiply-Reduce.

Array Multiply and Array Multiply-Add

Results:

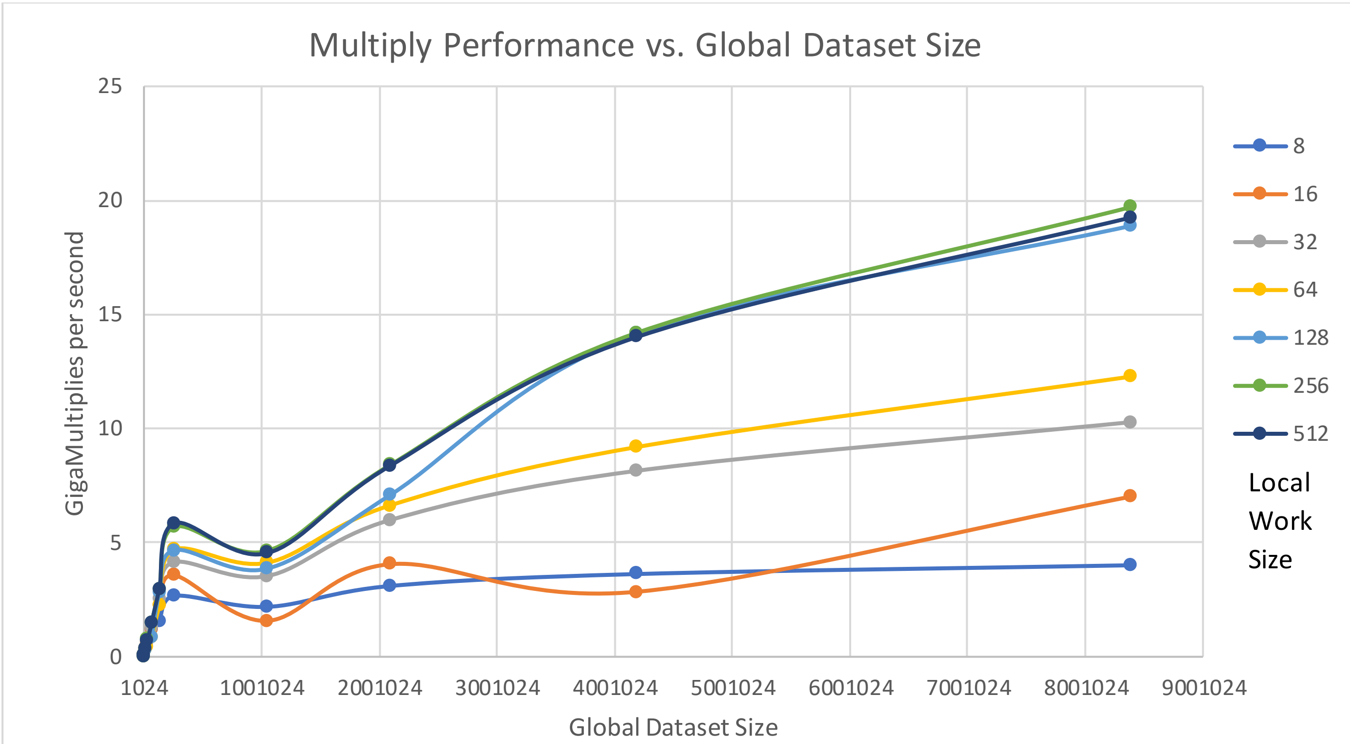
Multiply			
Global Dataset Size	Local Work Size	Number of work groups	GigaMultiplies per second
1024	8	128	0.019
2048	8	256	0.045
4096	8	512	0.074
16384	8	2048	0.342
32768	8	4096	0.553
65536	8	8192	1.179
131072	8	16384	1.586
262144	8	32768	2.669
1048576	8	131072	2.186
2097152	8	262144	3.103
4194304	8	524288	3.626
8388608	8	1048576	4.01
1024	16	64	0.019
2048	16	128	0.046
4096	16	256	0.088
16384	16	1024	0.351
32768	16	2048	0.616
65536	16	4096	1.197
131072	16	8192	2.287
262144	16	16384	3.565
1048576	16	65536	1.575
2097152	16	131072	4.059
4194304	16	262144	2.851

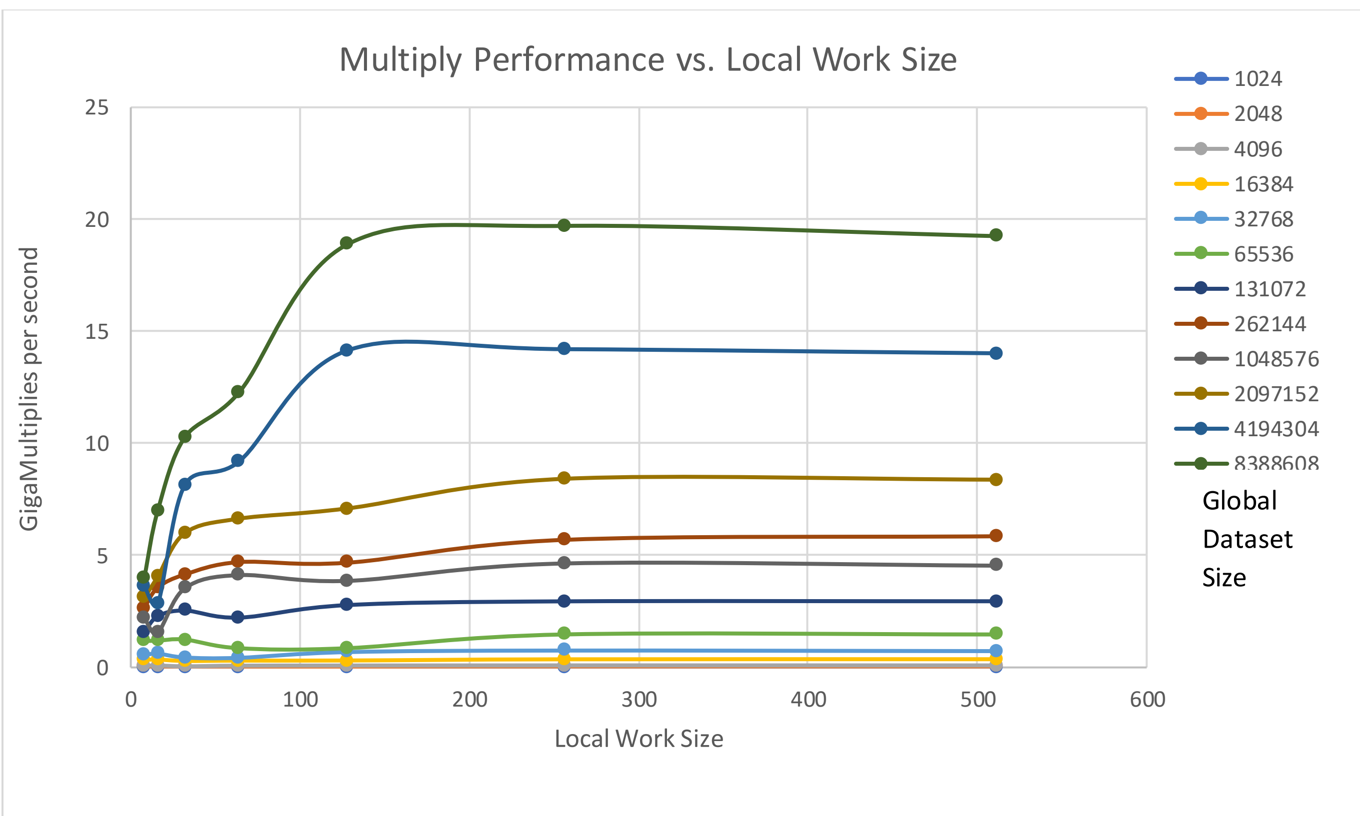
8388608	16	524288	7.019
1024	32	32	0.02
2048	32	64	0.042
4096	32	128	0.05
16384	32	512	0.29
32768	32	1024	0.443
65536	32	2048	1.243
131072	32	4096	2.539
262144	32	8192	4.152
1048576	32	32768	3.536
2097152	32	65536	5.999
4194304	32	131072	8.153
8388608	32	262144	10.282
1024	64	16	0.023
2048	64	32	0.043
4096	64	64	0.08
16384	64	256	0.305
32768	64	512	0.429
65536	64	1024	0.858
131072	64	2048	2.224
262144	64	4096	4.702
1048576	64	16384	4.121
2097152	64	32768	6.641
4194304	64	65536	9.195
8388608	64	131072	12.279
1024	128	8	0.012
2048	128	16	0.04
4096	128	32	0.086
16384	128	128	0.309
32768	128	256	0.686
65536	128	512	0.859
131072	128	1024	2.784
262144	128	2048	4.674
1048576	128	8192	3.863
2097152	128	16384	7.098
4194304	128	32768	14.15
8388608	128	65536	18.885
1024	256	4	0.023
2048	256	8	0.046

4096	256	16	0.092
16384	256	64	0.363
32768	256	128	0.753
65536	256	256	1.473
131072	256	512	2.95
262144	256	1024	5.695
1048576	256	4096	4.641
2097152	256	8192	8.42
4194304	256	16384	14.205
8388608	256	32768	19.708
1024	512	2	0.023
2048	512	4	0.046
4096	512	8	0.091
16384	512	32	0.367
32768	512	64	0.728
65536	512	128	1.472
131072	512	256	2.954
262144	512	512	5.855
1048576	512	2048	4.542
2097152	512	4096	8.382
4194304	512	8192	14.021
8388608	512	16384	19.25

Table used to plot the graphs:

	8	16	32	64	128	256	512
1024	0.019	0.019	0.02	0.023	0.012	0.023	0.023
2048	0.045	0.046	0.042	0.043	0.04	0.046	0.046
4096	0.074	0.088	0.05	0.08	0.086	0.092	0.091
16384	0.342	0.351	0.29	0.305	0.309	0.363	0.367
32768	0.553	0.616	0.443	0.429	0.686	0.753	0.728
65536	1.179	1.197	1.243	0.858	0.859	1.473	1.472
131072	1.586	2.287	2.539	2.224	2.784	2.95	2.954
262144	2.669	3.565	4.152	4.702	4.674	5.695	5.855
1048576	2.186	1.575	3.536	4.121	3.863	4.641	4.542
2097152	3.103	4.059	5.999	6.641	7.098	8.42	8.382
4194304	3.626	2.851	8.153	9.195	14.15	14.205	14.021
8388608	4.01	7.019	10.282	12.279	18.885	19.708	19.25


Multiply Performance vs. Global Dataset Size
GigaMultiplies per second
0
5
10
15
20
25
1024
1001024
2001024
3001024
4001024
5001024
6001024
7001024
8001024
9001024
Global Dataset Size
8
16
32
64
128
256
512
Local Work Size


Multiply Performance vs. Local Work Size
GigaMultiplies per second
0
5
10
15
20
25
0
100
200
300
400
500
600
Local Work Size
1024
2048
4096
16384
32768
65536
131072
262144
1048576
2097152
4194304
8388608
Global Dataset Size

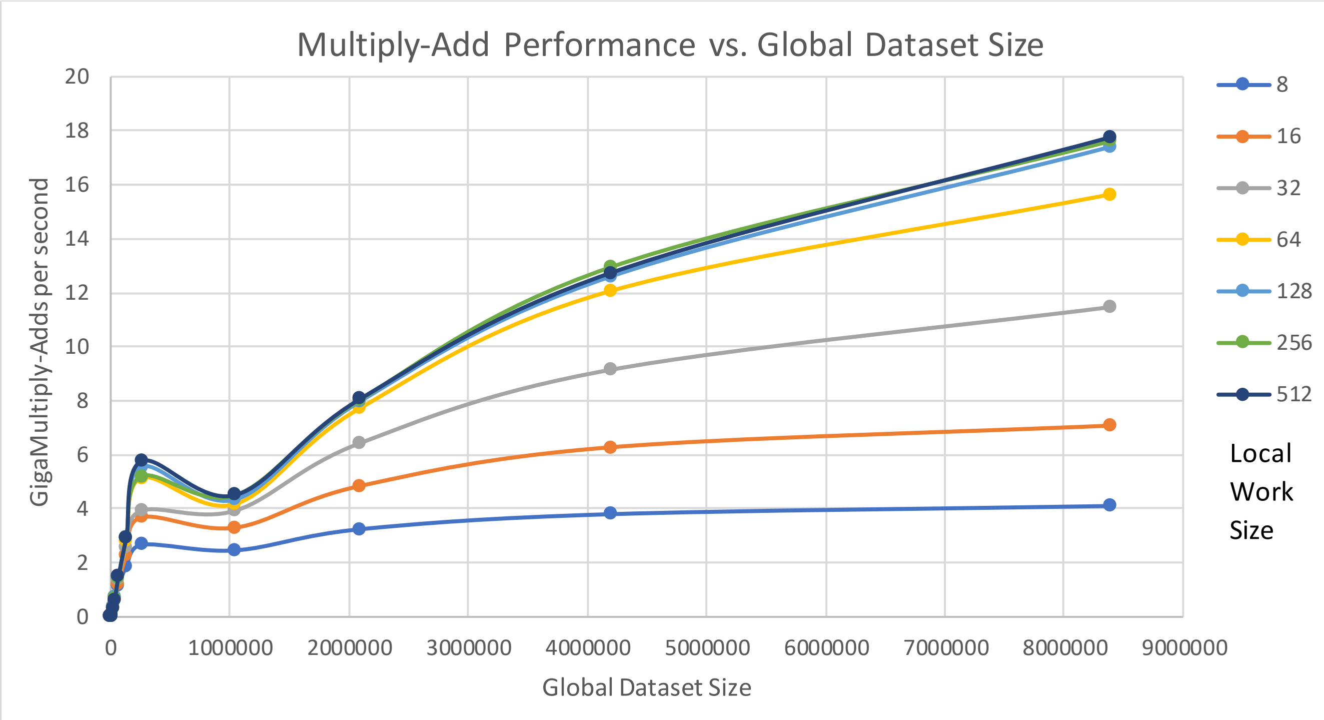
Multiply-Add			
Global Dataset Size	Local Work Size	Number of work groups	GigaMultiply-Adds per second
1024	8	128	0.022
2048	8	256	0.04
4096	8	512	0.09
16384	8	2048	0.337
32768	8	4096	0.645
65536	8	8192	1.127
131072	8	16384	1.871
262144	8	32768	2.67
1048576	8	131072	2.463
2097152	8	262144	3.247
4194304	8	524288	3.802
8388608	8	1048576	4.105
1024	16	64	0.023
2048	16	128	0.045
4096	16	256	0.09
16384	16	1024	0.347
32768	16	2048	0.692
65536	16	4096	1.242
131072	16	8192	2.284
262144	16	16384	3.688
1048576	16	65536	3.305
2097152	16	131072	4.852
4194304	16	262144	6.28
8388608	16	524288	7.082
1024	32	32	0.023
2048	32	64	0.045
4096	32	128	0.091
16384	32	512	0.35
32768	32	1024	0.723
65536	32	2048	1.406
131072	32	4096	2.586
262144	32	8192	3.936
1048576	32	32768	3.937
2097152	32	65536	6.441
4194304	32	131072	9.147
8388608	32	262144	11.47
1024	64	16	0.02

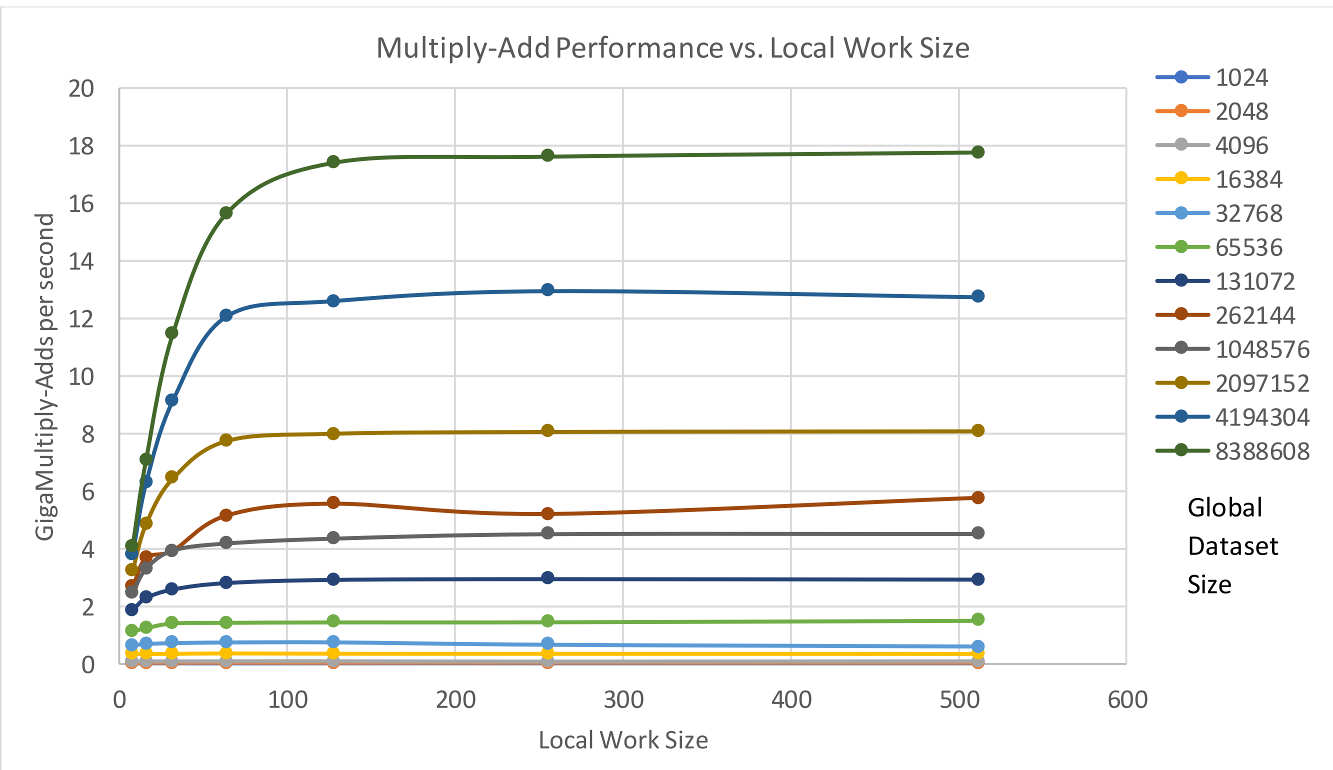
2048	64	32	0.046
4096	64	64	0.092
16384	64	256	0.365
32768	64	512	0.746
65536	64	1024	1.428
131072	64	2048	2.809
262144	64	4096	5.156
1048576	64	16384	4.183
2097152	64	32768	7.741
4194304	64	65536	12.066
8388608	64	131072	15.644
1024	128	8	0.023
2048	128	16	0.045
4096	128	32	0.094
16384	128	128	0.355
32768	128	256	0.75
65536	128	512	1.443
131072	128	1024	2.919
262144	128	2048	5.566
1048576	128	8192	4.352
2097152	128	16384	7.994
4194304	128	32768	12.601
8388608	128	65536	17.403
1024	256	4	0.023
2048	256	8	0.046
4096	256	16	0.083
16384	256	64	0.352
32768	256	128	0.667
65536	256	256	1.447
131072	256	512	2.945
262144	256	1024	5.21
1048576	256	4096	4.506
2097152	256	8192	8.056
4194304	256	16384	12.945
8388608	256	32768	17.614
1024	512	2	0.023
2048	512	4	0.046
4096	512	8	0.092
16384	512	32	0.35

32768	512	64	0.606
65536	512	128	1.498
131072	512	256	2.928
262144	512	512	5.771
1048576	512	2048	4.514
2097152	512	4096	8.079
4194304	512	8192	12.734
8388608	512	16384	17.76

Table used to generate the graphs:

	8	16	32	64	128	256	512
1024	0.022	0.023	0.023	0.02	0.023	0.023	0.023
2048	0.04	0.045	0.045	0.046	0.045	0.046	0.046
4096	0.09	0.09	0.091	0.092	0.094	0.083	0.092
16384	0.337	0.347	0.35	0.365	0.355	0.352	0.35
32768	0.645	0.692	0.723	0.746	0.75	0.667	0.606
65536	1.127	1.242	1.406	1.428	1.443	1.447	1.498
131072	1.871	2.284	2.586	2.809	2.919	2.945	2.928
262144	2.67	3.688	3.936	5.156	5.566	5.21	5.771
1048576	2.463	3.305	3.937	4.183	4.352	4.506	4.514
2097152	3.247	4.852	6.441	7.741	7.994	8.056	8.079
4194304	3.802	6.28	9.147	12.066	12.601	12.945	12.734
8388608	4.105	7.082	11.47	15.644	17.403	17.614	17.76


Multiply-Add Performance vs. Global Dataset Size
GigaMultiply-Adds per second
Global Dataset Size
0
2
4
6
8
10
12
14
16
18
20
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000
9000000
8
16
32
64
128
256
512
Local
Work
Size


Multiply-Add Performance vs. Local Work Size
GigaMultiply-Adds per second
Local Work Size
0
2
4
6
8
10
12
14
16
18
20
0
100
200
300
400
500
600
1024
2048
4096
16384
32768
65536
131072
262144
1048576
2097152
4194304
8388608
Global
Dataset
Size

Explanation:

To obtain the results for the array multiply and array multiply-add sections I created two separate programs that are nearly identical. They both utilize OpenCL to execute code found in their corresponding .cl files on the GPU. In these two programs all the computations required for multiplying and adding values from the arrays occurs on the GPU. The timing values and in turn the performance only reflect how long it took for the GPU to perform the calculations, the time required to create the arrays and pass data from the device to the host in either direction is not considered. For the code development I followed the hint provided in the project instructions. For both the array multiply and the multiply-add programs I created four arrays, passing the first three in and the returning the fourth.

Much like the CUDA project, the code executed on the GPU uses the gid to create an implied for loop that performs the multiply or fused multiply add on all indexes of the arrays. The passes of this implied for loop are done in parallel so we expect to see massive performance increases very similar to that of the CUDA project. The cl code for the multiply-add program is very similar to that of the cl file for exclusively multiplying, however as explained in the lectures we can expect this to be executed in assembly as a fused multiply-add which should allow that sequence of operations to occur nearly as quick as a simple multiply.

Analysis:

Overall, the results were as expected. In every graph the lower the local work size and lower the global dataset size the worse the performance was. It is interesting to note the local work size stopped increasing the performance after increasing past a size of around 128. This occurs for both types of computations. In the performance vs. local work size graphs it is apparent that all the lines plateau past a local work size of 128. This means that increasing the local size past 4 warps does not add much performance for this given problem. In these two programs the warps do not spend enough time blocked to get an increase performance by adding more than four warps.

The performance vs. global dataset size graphs both show that as the global dataset size increases the performance continually increases as well. This ties into the discussions about a more optimistic view on Amdahl's law (Gustafson's observation). As the dataset size increases the maximum speedup achievable increases. While I didn't calculate the parallel fraction of the code for these programs I think it is safe to assume it is quite high, even with a high initial parallel fraction I think this observation is one of the major contributors to the performance continually increasing. Unfortunately, due to hitting the time restrictions on the DGX server I was not able to run tests that had global dataset size much bigger than ~8 million.

The multiply performance at nearly every combination of the global dataset size and local work size outperformed the multiply-add performance. This is expected as it is running only a single operation compared to two operations on the data. However, as explained in the lectures, fused multiply add is taking place which is causing the performance of the multiply-add program only to be slightly worse than the multiply program. For example, the multiply performance for a global data set size of ~8 million and a local work group size of 512 was 19.25 gigamultiplies per second and the corresponding performance for multiply-add was 17.76 gigamultiply-adds per second. Technically, performing twice as many mathematical operations on the data only resulted in a .92 slowdown (17.76/19.25 = .92).

At a few points in the performance vs. global dataset size graphs the lines swerved around each other, but gradually stabilized as the dataset size grew larger. I think these occurrences are mainly due to other processes or other potential sources of "noise" on the hardware that were affecting the collected run times. Also due to the runtime limitations on the DGX server I did not run the program using the same values multiple times in order to take the best result as we have done in many other projects. This makes the results more susceptible to poor runs.

As with the Project 5, these two programs run a standard set of calculations on all elements in the given arrays. Data parallel programs such as these lend themselves to being parallelized on a GPU, this is clearly reflected by the amazing performances provided at the various combinations of global dataset sizes and local work sizes. This lends itself as proof that when GPU parallel computing is implemented properly massive performance increases can be achieved.

Array Multiply-Reduction

Results:

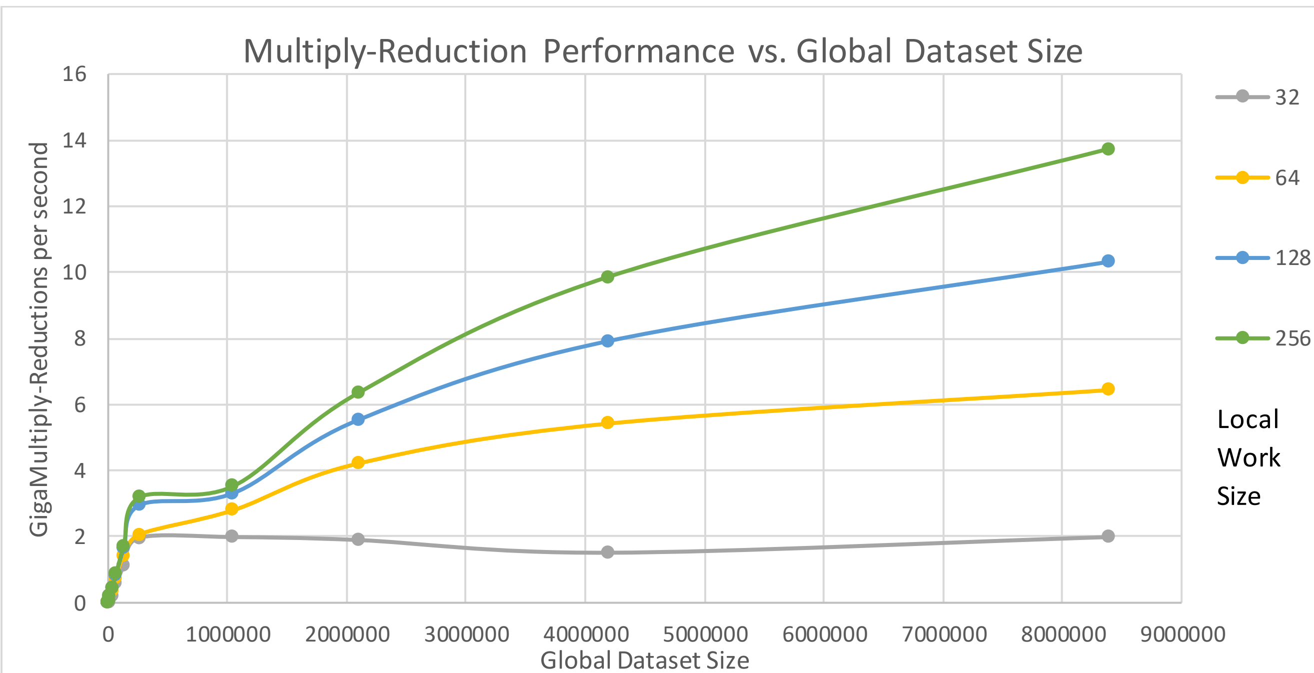
Multiply Reduce			
Global Dataset Size	Local Work Size	Number of work groups	GigaMultiply-Reductions per second
1024	32	32	0.013
2048	32	64	0.015
4096	32	128	0.032
16384	32	512	0.182
32768	32	1024	0.225
65536	32	2048	0.591
131072	32	4096	1.109
262144	32	8192	1.97
1048576	32	32768	1.988
2097152	32	65536	1.899
4194304	32	131072	1.515
8388608	32	262144	1.988
1024	64	16	0.012
2048	64	32	0.026

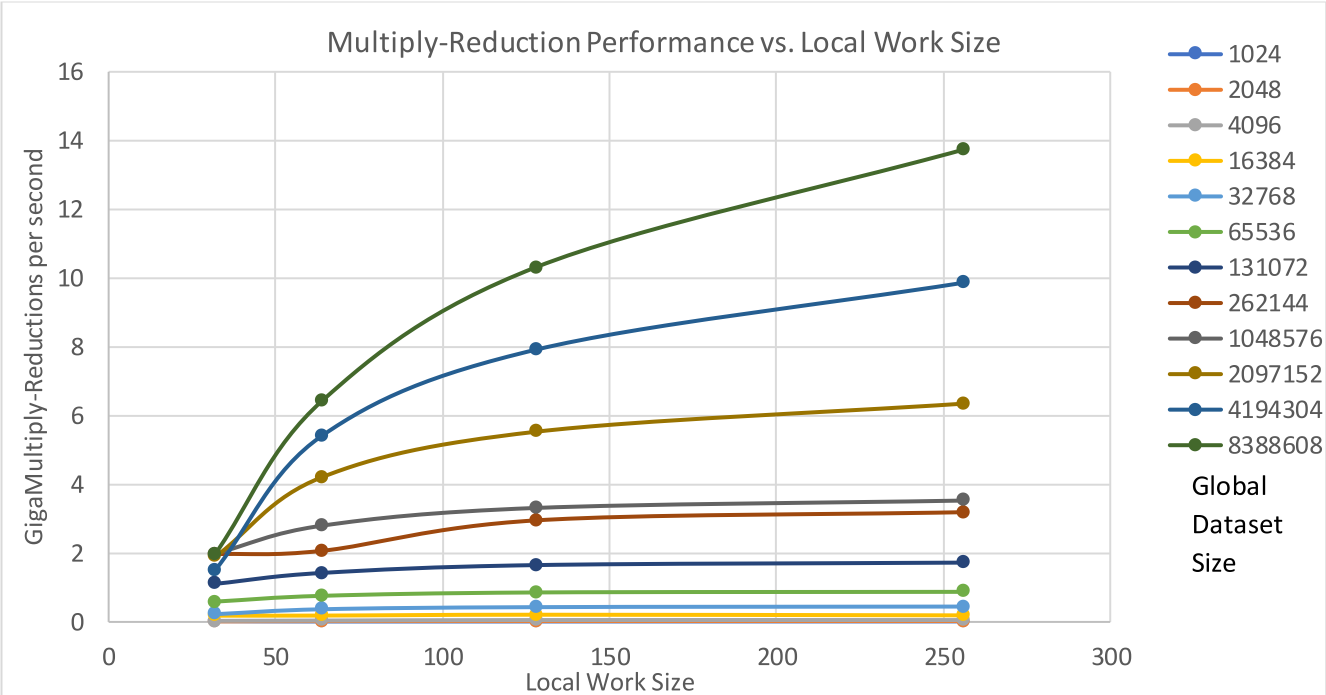
4096	64	64	0.041
16384	64	256	0.185
32768	64	512	0.369
65536	64	1024	0.76
131072	64	2048	1.426
262144	64	4096	2.069
1048576	64	16384	2.801
2097152	64	32768	4.214
4194304	64	65536	5.422
8388608	64	131072	6.436
1024	128	8	0.013
2048	128	16	0.027
4096	128	32	0.055
16384	128	128	0.21
32768	128	256	0.43
65536	128	512	0.857
131072	128	1024	1.653
262144	128	2048	2.954
1048576	128	8192	3.315
2097152	128	16384	5.532
4194304	128	32768	7.919
8388608	128	65536	10.317
1024	256	4	0.014
2048	256	8	0.027
4096	256	16	0.055
16384	256	64	0.191
32768	256	128	0.447
65536	256	256	0.875
131072	256	512	1.725
262144	256	1024	3.192
1048576	256	4096	3.53
2097152	256	8192	6.349
4194304	256	16384	9.865
8388608	256	32768	13.736

Table used to generate the graphs:

	32	64	128	256
1024	0.013	0.012	0.013	0.014

2048	0.015	0.026	0.027	0.027
4096	0.032	0.041	0.055	0.055
16384	0.182	0.185	0.21	0.191
32768	0.225	0.369	0.43	0.447
65536	0.591	0.76	0.857	0.875
131072	1.109	1.426	1.653	1.725
262144	1.97	2.069	2.954	3.192
1048576	1.988	2.801	3.315	3.53
2097152	1.899	4.214	5.532	6.349
4194304	1.515	5.422	7.919	9.865
8388608	1.988	6.436	10.317	13.736





Explanation:

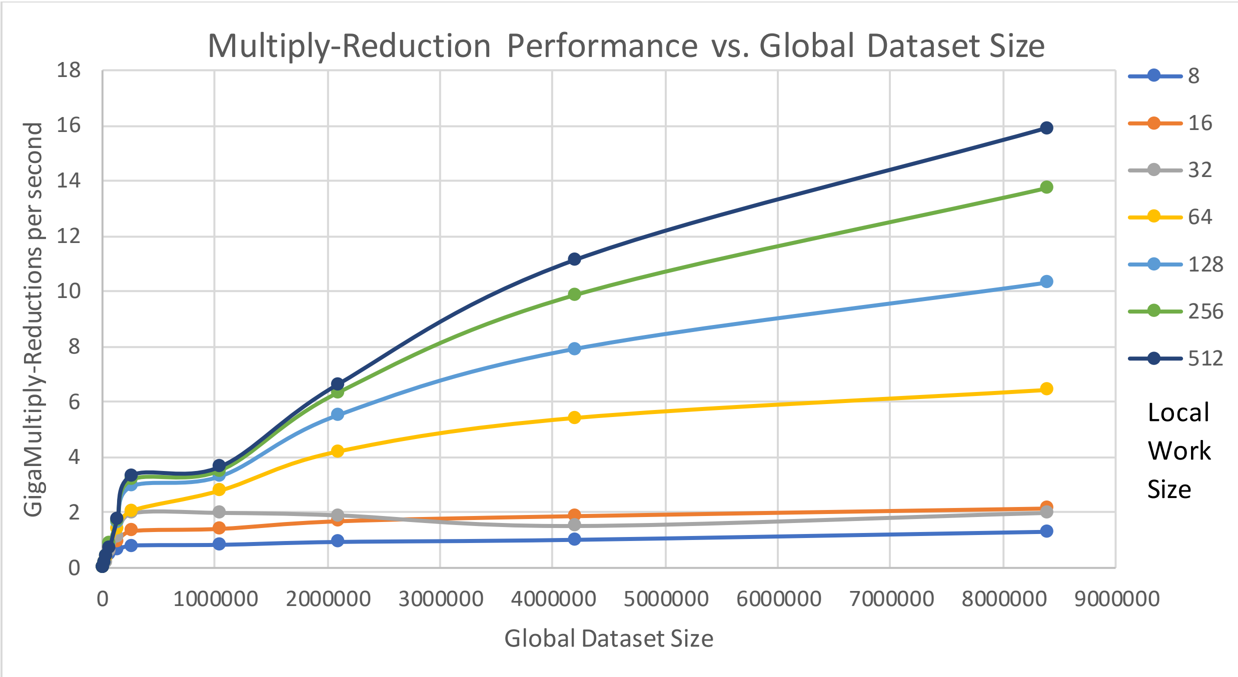
The multiply-reduce program has a very similar set up to the multiply and multiply-add programs. To develop the .cpp file and .cl file I followed the week 8 OpenCL Reduction slides. This program like the two previous does not include the amount of time required to create the arrays or pass data from the host to the device in its timing. It does however include the time required to read an array from the device to the host that has a size equal to the number of work groups being used, after that array has been read it also must sum all elements in that returned array, only once that is performed is the second time stamp generated. These steps on the host must be performed since the .cl file is only able to perform the reduction process on elements within the same workgroup. One other major differences in this .cl file is that this program utilizes a local array, this local array holds the initial results of the multiplication. These results are then used to perform the reduction steps for the workgroup. The gid, work group number, thread number and work group size are used to perform the reduction steps in an implied loop. Each work group produces a single value after performing the reductions steps which is written into the global array that has a size equal to the number of work groups.

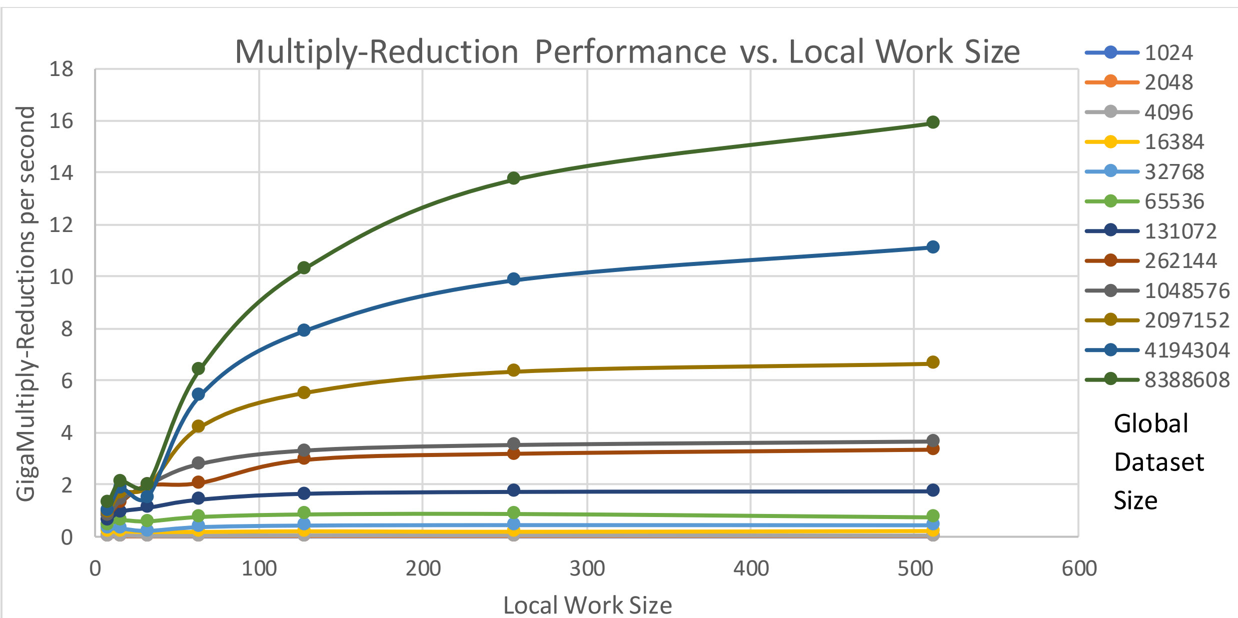
The graphs and tables above match the requirements in step 10 of the project assignment, however below I am also attaching two graphs I made that show results from running the program with local work sizes of 16-512. I found it quite interesting to compare the results at these two extremes to those of the previous two programs.

Analysis:

The results for this program follow the same general patterns of the previous two. As the local work size and global dataset size got smaller the performance decreased, as they increased the performance improved. One difference was the plateauing of the performance at certain local work sizes. In the previous two programs the local work size stopped increasing the performance after 128, however in this program the local work size did not stop improving the performance at any given value. I hypothesize this is because the .cl file for this program is far more complex and contains two barriers which allows for the addition of extra warps to be utilized since warps spend more time being blocked when compared to the previous two programs. However, the small global dataset sizes do plateau in performance at a low local work size, I think this is because the workgroups are able to step through the entire dataset very quickly regardless of how many warps each workgroup contains. One other thing to consider is that as the local work size increases the smaller the summation array will be on the host machine, since this part of the program runs in O(N) runtime, decreasing it size definitely helps the overall performance.

Similar to the previous discussion about a more optimistic view on Amdahl's law (Gustafson's observation), the performance of this program continually increases as the global dataset size increases. This trend also continues for the additional graphs I included below. The run times of this program were very impressive, it actually outperformed the multiply-add when looking at a global dataset size of ~8 million. However, it does still get out performed by the multiply program. This speaks to how efficient even somewhat more complex problems can be using GPU parallelism. As mentioned in the lectures, problems that simply stream data through an identical set of processes are ideal for GPU parallelism, and as we can see GPU parallelism provides absolutely stunning performance increases.


Multiply-Reduction Performance vs. Global Dataset Size
GigaMultiply-Reductions per second
Global Dataset Size
0
2
4
6
8
10
12
14
16
18
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000
9000000
8
16
32
64
128
256
512
Local Work Size


Multiply-Reduction Performance vs. Local Work Size
GigaMultiply-Reductions per second
Local Work Size
0
2
4
6
8
10
12
14
16
18
0
100
200
300
400
500
600
1024
2048
4096
16384
32768
65536
131072
262144
1048576
2097152
4194304
8388608
Global Dataset Size