Project 5: OpenCL Array Multiplication and Reduction

Joel Huffman huffmajo@oregonstate.edu

Tables and Graphs

Table 1: Multiply and Multiply-Add Performance Across Global and Local Work Sizes

Global Work Size	**Local Work Size**	**Number of Work Groups**	**Array Multiply Performance (GM/S)**	**Array Multiply-Add Performance (GM/S)**
1024	4	256	0.027	0.027
1024	16	64	0.014	0.014
1024	64	16	0.014	0.025
1024	256	4	0.027	0.026
1024	1024	1	0.027	0.025
16384	4	4096	0.347	0.329
16384	16	1024	0.18	0.41
16384	64	256	0.43	0.407
16384	256	64	0.414	0.43
16384	1024	16	0.427	0.226
262144	4	65536	0.707	0.683
262144	16	16384	1.123	1.155
262144	64	4096	1.251	1.26
262144	256	1024	1.259	0.491
262144	1024	256	1.266	1.264

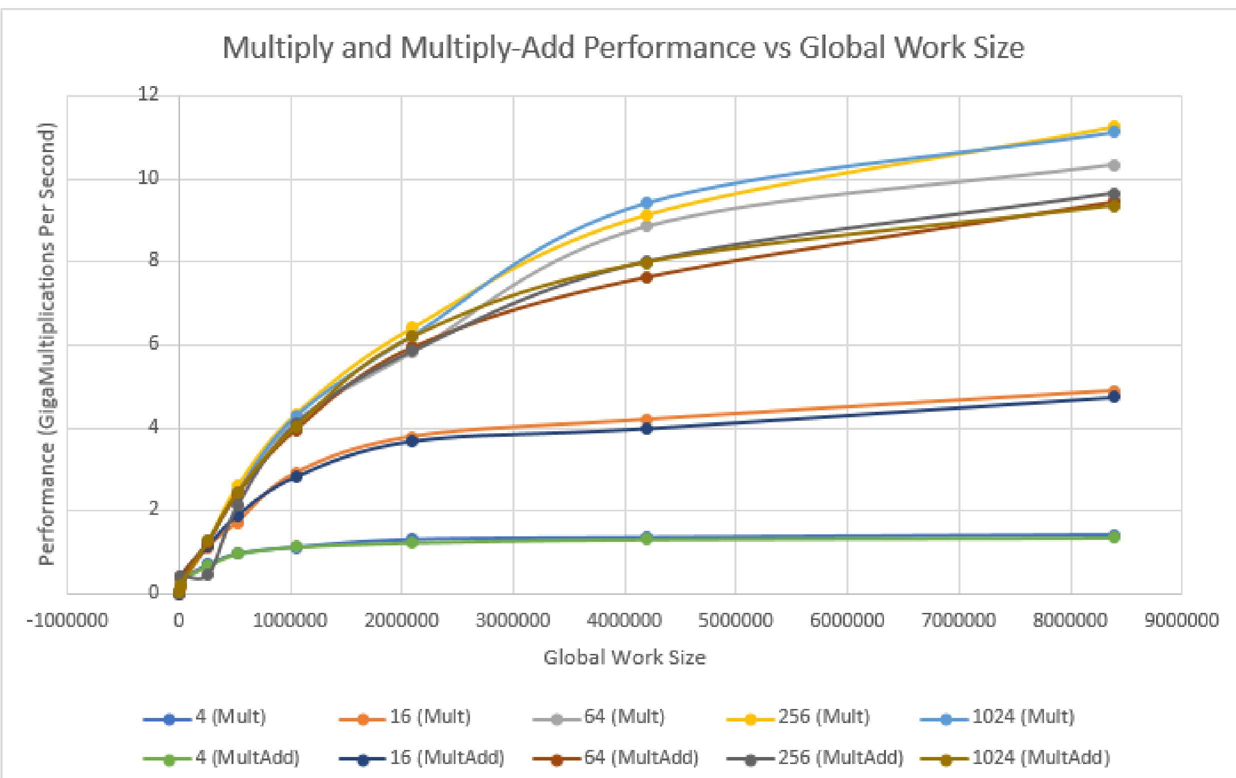
524288	4	131072	0.97	0.969
524288	16	32768	1.72	1.897
524288	64	8192	2.478	2.405
524288	256	2048	2.6	2.154
524288	1024	512	2.44	2.432
1048576	4	262144	1.129	1.139
1048576	16	65536	2.912	2.826
1048576	64	16384	4.165	3.96
1048576	256	4096	4.339	4.103
1048576	1024	1024	4.288	4.022
2097152	4	524288	1.31	1.247
2097152	16	131072	3.787	3.688
2097152	64	32768	5.822	5.949
2097152	256	8192	6.423	5.857
2097152	1024	2048	6.229	6.192
4194304	4	1048576	1.358	1.328
4194304	16	262144	4.197	3.992
4194304	64	65536	8.855	7.638
4194304	256	16384	9.145	8.004
4194304	1024	4096	9.425	7.985
8388608	4	2097152	1.425	1.365
8388608	16	524288	4.889	4.745
8388608	64	131072	10.348	9.456
8388608	256	32768	11.269	9.649
8388608	1024	8192	11.13	9.345

Table 2: Multiply-Reduce Performance across Global and Local Work Sizes

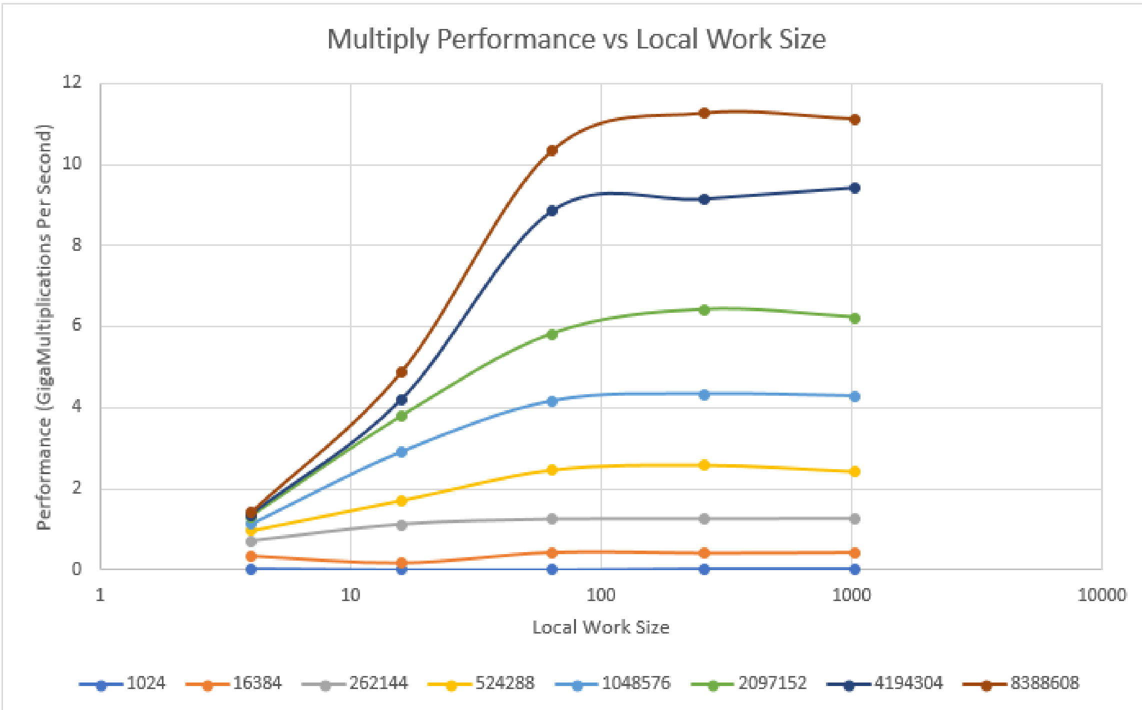
Global Work Size	**Local Work Size**	**Number of Work Groups**	**Performance (GM/S)**
1024	32	32	0.006
1024	64	16	0.006
1024	128	8	0.006
1024	256	4	0.005
16384	32	512	0.09
16384	64	256	0.091
16384	128	128	0.085
16384	256	64	0.092
262144	32	8192	1.075
262144	64	4096	1.152
262144	128	2048	0.961
262144	256	1024	1.11
524288	32	16384	1.68
524288	64	8192	1.798
524288	128	4096	2.237
524288	256	2048	0.808
1048576	32	32768	2.322
1048576	64	16384	3.006
1048576	128	8192	3.539
1048576	256	4096	2.775
2097152	32	65536	2.869
2097152	64	32768	4.027

2097152	128	16384	5.198
2097152	256	8192	4.717
4194304	32	131072	3.088
4194304	64	65536	4.778
4194304	128	32768	6.473
4194304	256	16384	5.413
8388608	32	262144	3.489
8388608	64	131072	5.416
8388608	128	65536	6.962
8388608	256	32768	6.662

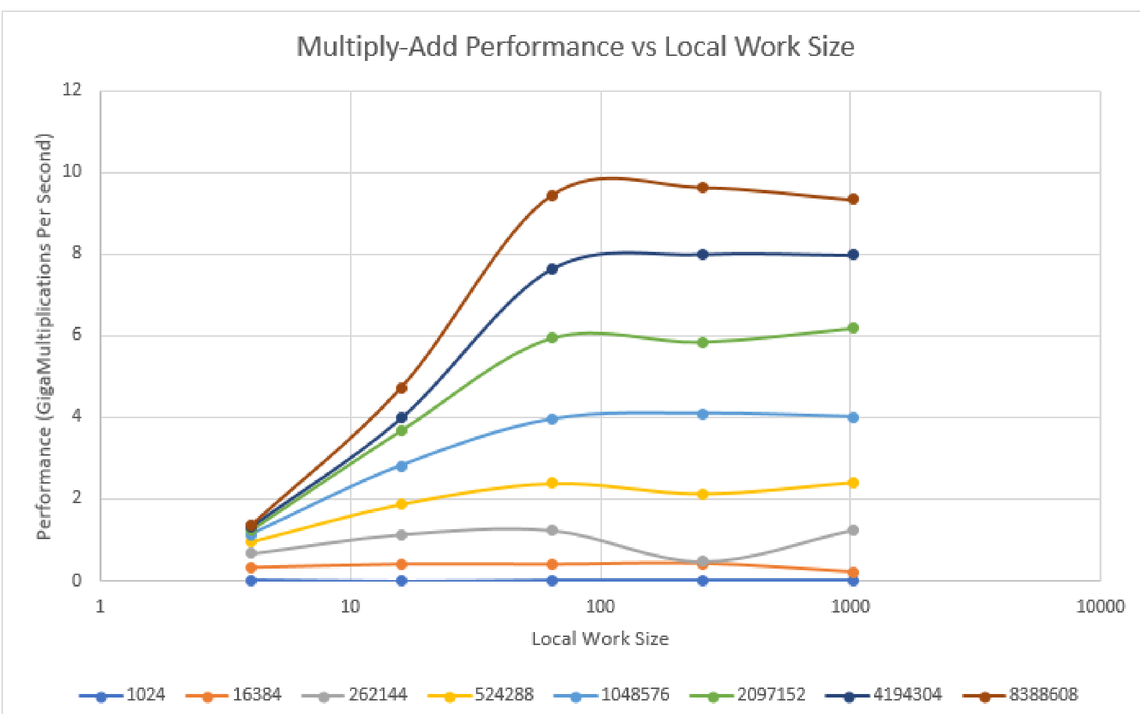
Graph 1:



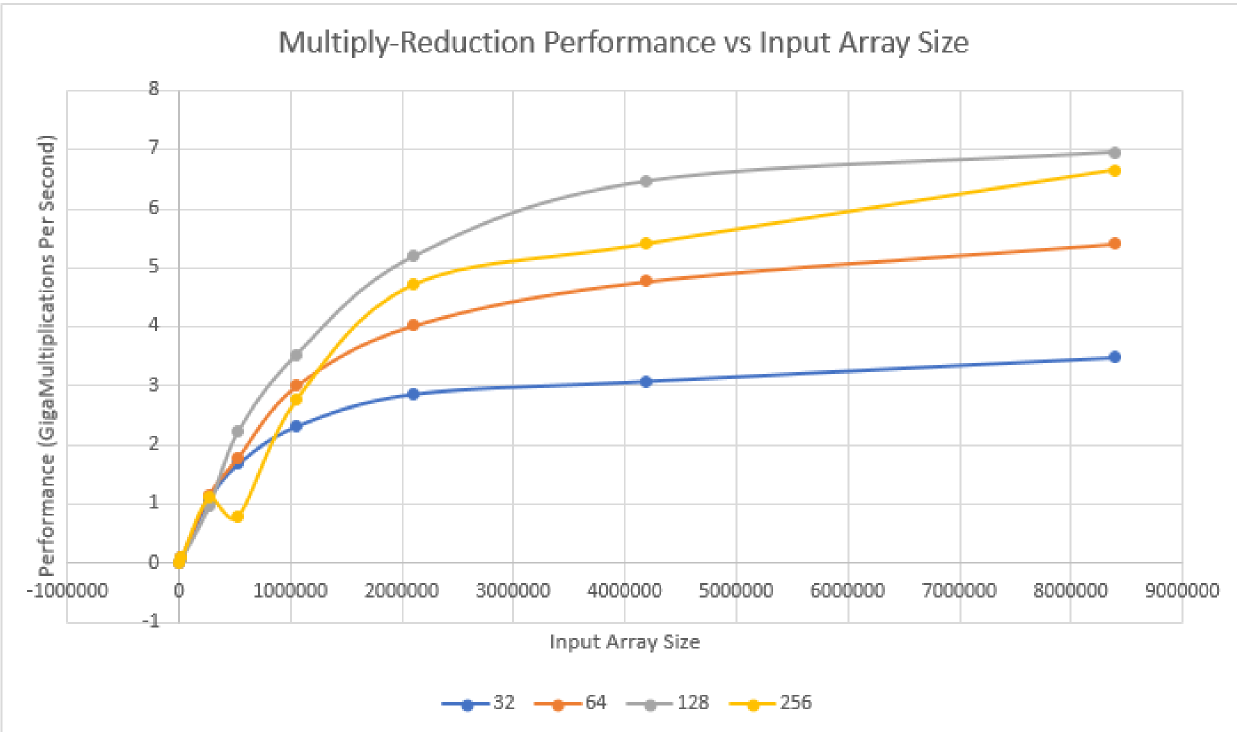
Graph 2:



Graph 3:



Graph 4:



Explanation and Analysis

Array Multiply and Array Multiple-Add Analysis

1. **What machine you ran this on**

All benchmarking was performed on OSU rabbit server.

2. **Show the tables and graphs**

See table 1 and graphs 1 - 3 above. I split the Multiply and Multiply-Add Performance vs Local Work Size into two graphs (2 and 3) so it was easier to distinguish the Multiply and Multiply-Add curves from one-another.

3. **What patterns are you seeing in the performance curves?**

From Graph1, performance increases as global work size increases across all local work sizes, but the increase is much greater for larger local work sizes. From Graph 2 and 3, performance levels out once the appropriate value of local work size is hit for each global work size. The appropriate local work sizes increases for larger global work sizes. These trends are seen for both the Multiply and Multiply-Add curves.

4. **Why do you think the patterns look this way?**

The curves increase as global work size increases because we are allocating more work items to tackle the calculations. The more resources we have to utilize, the more computations we can crunch in the same amount of time.

The curves are leveling out after an optimal local work size (as seen in Graph 2 and 3) due to the GPU's hardware only being able to efficiently utilize a limited local work size. Using larger local work sizes doesn't yield better performance and in some cases hampers performance. The Titan Black used for this project has 192 CUDA cores per SM. This is right around the number of local work size where we see optimal performance.

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

Multiply performance is higher than Multiply-Add performance across the board. This is most noticeable with larger local and global work sizes where the difference is millions, if not billions, of computations per second.

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

The difference in performance between the Multiply and Multiply-Add curves shows that while GPUs can perform ridiculous feats of parallel computation, optimization of these calculations is crucial to maximize performance. We could utilize c++'s fma functions to optimize our Multiply-Add calculations or directly use assembly as presented in week 7's video on OpenCL assembly.

Multiply-Reduction Analysis

7. **Show this table and graph**

See all table 2 and graph 4 above.

8. **What pattern are you seeing in this performance curve?**

Performance increases as input array size increases. The 128 local work size looks to be performing the best out of all the local work sizes.

9. **Why do you think the pattern looks this way?**

As input array size increases, so does the amount of data we have to crunch. This means the GPU has more data it can utilize for parallelization. Thus the higher performance with higher input array size. We see a local work size of 128 performing better than higher or lower values because that is the sweet spot for this GPU and dataset. A smaller local work size is overused and can't keep up with the amount of data and a larger local work size is underutilized.

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

GPU parallelization benefits the most when the cores can act independently although they can utilize local shared memory to provide data to one another in a still efficient manner (like with their reduction). While GPUs can yield ridiculous performance for specific tasks, they do not perform all tasks equally spectactually. There are several cases where CPU parallelization can and will perform better than GPU parallelization.