Jeremy Udarbe
CS 475
Project 6

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

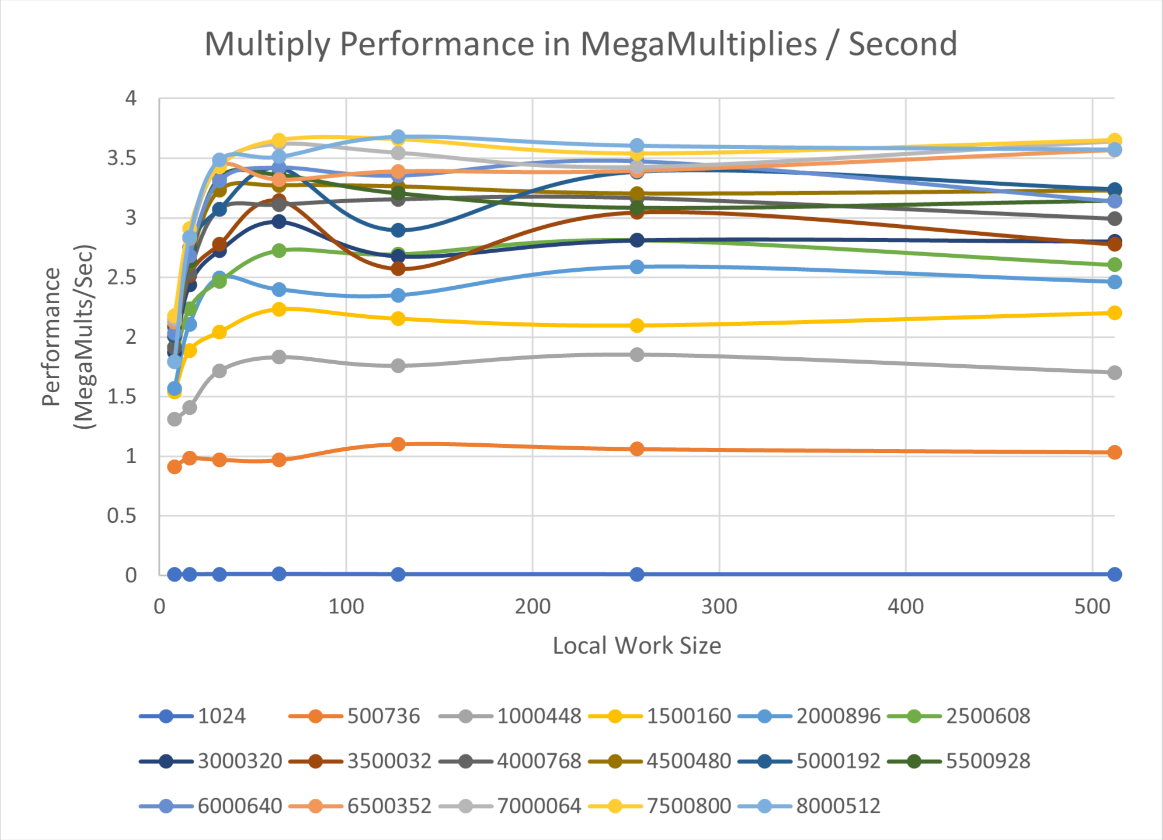
1. The program was compiled with Visual Studio 2019 on PC using a MSI GeForce GTX 1070 8G Graphics Card with 1920 CUDA cores.

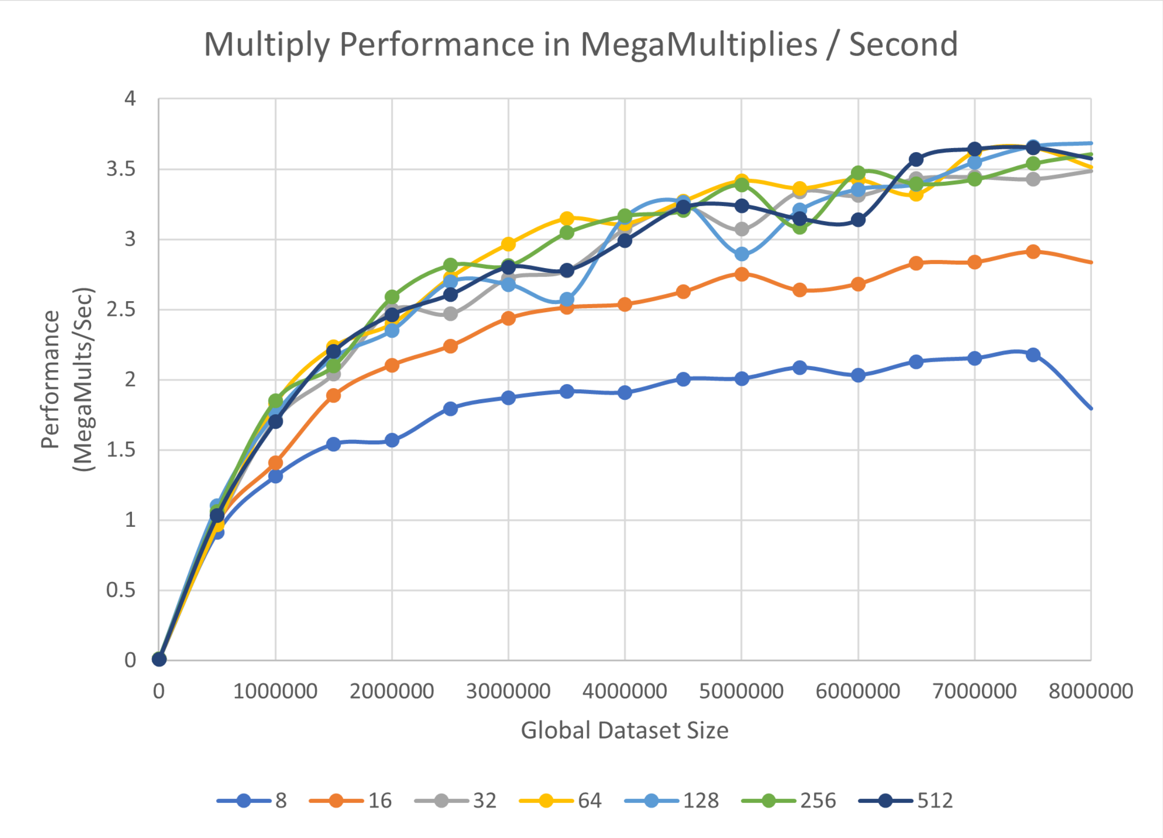
2.

Table of Performances in MegaMultiplies Per Second

Local / Global	8	16	32	64	128	256	512
1024	0.01	0.01	0.012	0.013	0.011	0.01	0.009
500736	0.912	0.983	0.972	0.968	1.102	1.061	1.033
1000448	1.311	1.41	1.715	1.831	1.759	1.852	1.703
1500160	1.541	1.887	2.042	2.234	2.155	2.098	2.202
2000896	1.568	2.104	2.497	2.399	2.35	2.588	2.463
2500608	1.794	2.24	2.469	2.727	2.696	2.815	2.606
3000320	1.872	2.437	2.723	2.967	2.677	2.811	2.802
3500032	1.916	2.514	2.777	3.146	2.572	3.046	2.778
4000768	1.909	2.537	3.072	3.11	3.156	3.166	2.992
4500480	2.004	2.626	3.23	3.271	3.262	3.203	3.228
5000192	2.008	2.752	3.071	3.415	2.894	3.385	3.238
5500928	2.086	2.637	3.338	3.362	3.207	3.085	3.145
6000640	2.033	2.682	3.311	3.422	3.355	3.475	3.138
6500352	2.128	2.827	3.432	3.32	3.39	3.394	3.568
7000064	2.153	2.837	3.443	3.62	3.546	3.426	3.642
7500800	2.178	2.911	3.429	3.651	3.659	3.539	3.651
8000512	1.794	2.835	3.487	3.513	3.682	3.607	3.574

Jeremy Udarbe
CS 475
Project 6



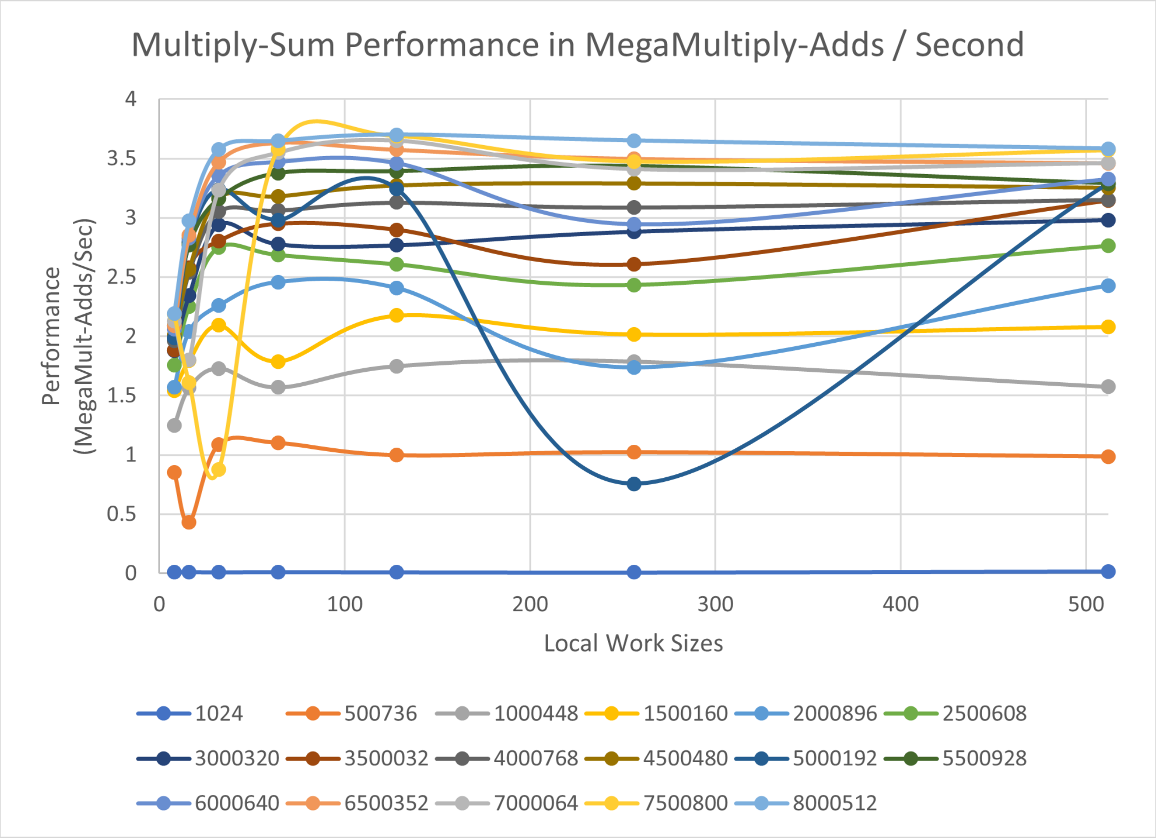


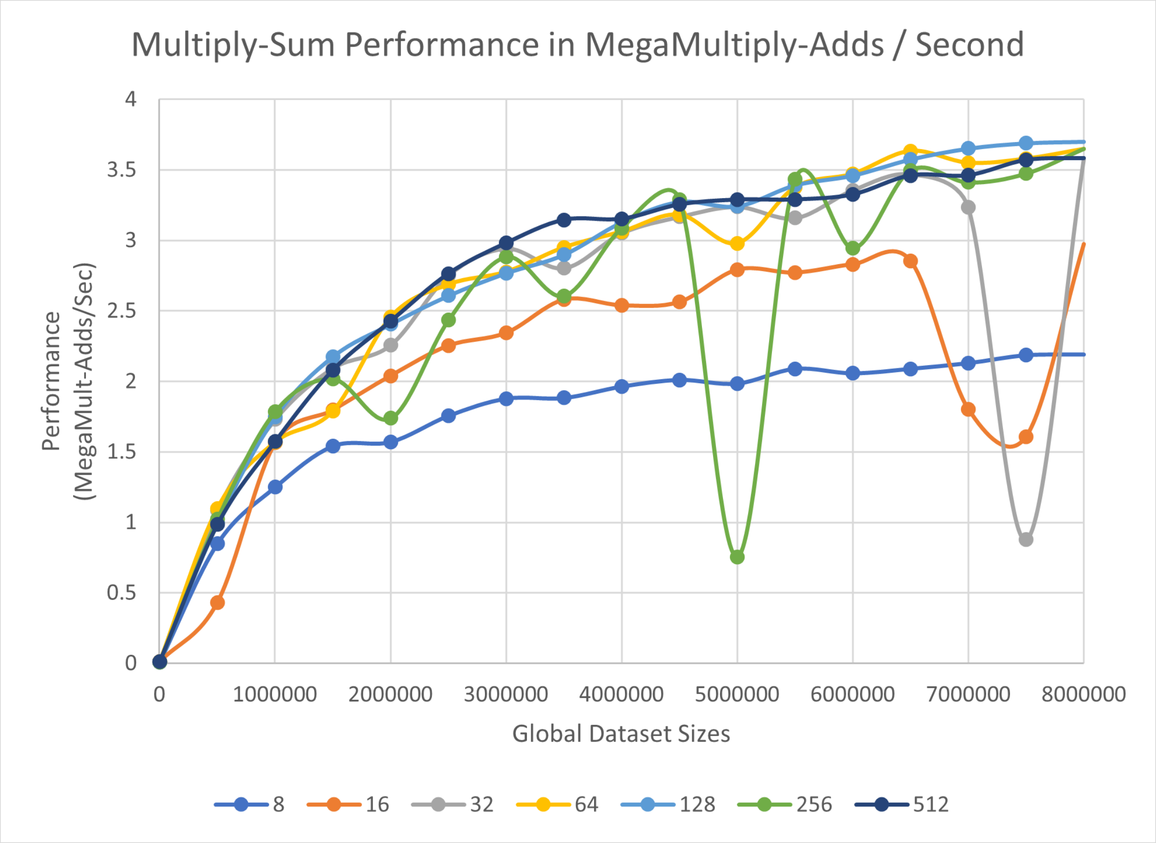
Jeremy Udarbe
CS 475
Project 6

Table of Performances in MegaMultiply-Adds Per Second

Local / Global	8	16	32	64	128	256	512
1024	0.01	0.011	0.01	0.011	0.01	0.009	0.014
500736	0.85	0.433	1.087	1.1	0.997	1.022	0.986
1000448	1.25	1.564	1.73	1.569	1.748	1.786	1.574
1500160	1.541	1.796	2.094	1.788	2.176	2.017	2.081
2000896	1.571	2.04	2.259	2.457	2.406	1.738	2.428
2500608	1.756	2.253	2.752	2.687	2.608	2.434	2.765
3000320	1.878	2.344	2.943	2.777	2.768	2.883	2.982
3500032	1.883	2.579	2.804	2.95	2.899	2.607	3.146
4000768	1.964	2.539	3.052	3.062	3.129	3.088	3.153
4500480	2.011	2.563	3.168	3.182	3.274	3.291	3.255
5000192	1.985	2.792	3.239	2.98	3.243	0.756	3.289
5500928	2.09	2.771	3.159	3.376	3.392	3.435	3.29
6000640	2.058	2.83	3.357	3.472	3.461	2.945	3.326
6500352	2.09	2.854	3.47	3.634	3.575	3.498	3.461
7000064	2.13	1.803	3.234	3.55	3.652	3.412	3.462
7500800	2.186	1.608	0.878	3.579	3.69	3.475	3.57
8000512	2.191	2.973	3.577	3.648	3.701	3.651	3.583

Jeremy Udarbe
CS 475
Project 6





Jeremy Udarbe
CS 475
Project 6

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

For this project, I chose Global dataset sizes in intervals of at least 500,000 and also divisible by 1024 so that it can fit a set of 32 cores evenly per group. As for the local work sizes, I used intervals of n^2 which all factor evenly into 1024 which is defined as the maximum work group size. Starting with the array multiplication performance graph displaying local work size curves, each curve generally shares a logarithmic distribution. There is no definite horizontal asymptote that each curve is reaching since it seems to continue growing past Global data size 8M. From the current pattern, we can only assume that all performance curves would eventually level as the values become larger once the speedup has reached max optimization. Specifically in this graph, as the local work size becomes larger, its curve becomes more erratic whereas it begins to oscillate to that resembling a trigonometric wave. In the same graph, at the 128 curve (light blue), the slope dips at global dataset sizes 3,500,032 and 5,000,192.

The Multiply-Sum Performance graph with local work size curves would share the same behavior in logarithmic distribution. Rather in this case, the performances per curve became more varied and erratic as the global dataset size became larger. Specifically at the 256 curve(lime green), where there is a large dip at global dataset size 5,000,192. This also occurs at curve 32 (grey) as well as curve 15 (orange) global size at global size 7,500,800 which share a significant decrease in performance around 0.75 MegaMult-Adds per second.

As for the Multiply Performance with Group dataset curves, there is still the positive logarithmic pattern. In addition the horizontal asymptote is more present which most curves level off at around local work size 64. The only anomaly is the dip in some curves such as with global values 3,500,032 (brown) and 3,000,320 (navy blue). In general, as the global dataset becomes larger, the steeper the curve towards leveling becomes steeper.

The Multiply-Sum performance with group dataset curves are more exaggerated in where the distribution becomes erratic, but share the same distribution behavior as the multiply only performance. The only significant alteration is at the 3,000,320 (navy blue) curve at local work size 256, where there is a large dip in performance from 3.243 to 0.756 MegaMult-Adds/sec.

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

Overall, the patterns of all the graphs are mostly normal in the sense that most curves share a similar rising pattern. This supports the fact that the speedup increases as the amount of memory available to process each work item is increased to the point where there is an optimal amount of work items being used per work group in the GPU’s workload. The only discrepancy worth pointing out are the large dips in data specifically in the Multiply-Sum Graphs which are probably caused by a hiccup in performance when OpenCL queues jobs for processing elements between the work groups in the GPU.

Jeremy Udarbe
CS 475
Project 6

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

When comparing the values between both methods, the Multiply-Add performances are somewhat larger and less consistent. This makes sense since there is an extra step when computing the value for the index dC[i]. In addition, the multiply-add graphs are more exaggerated in the oscillation for larger curve values where there would be significant decreases in performance which is probably due to how jobs are being split for processing as stated previously.

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

In previous projects, we observed how computing efficiency is undeniably affected by how GPU workloads are organized. This project displayed how the number of work items per block and the number of block work groups in a grid can be optimized for the GPU. In addition, dividing the GPU's workload into more manageable sub-groups of smaller tasks is one of the imperative issues in GPU parallel computing. Nevertheless, the size at which the work group must also be considered since in this case, the number of elements must be divisible by 1024 so that they may be divided into the local size evenly so that it can be processed through the register's fixed size.