CS 575 Project 6

Jeremy Dao

May 2020

1. The following results were gathered using a machine with a Nvidia Quadro 4000 GPU.

2. The following table displays the array multiply performance data for varying number of global array sizes and local work sizes. Note that all performance values are in units of GigaMutiples per Second.

Array Size \ Work Size	8	16	64	128	256	512
1024	0.05	0.05	0.05	0.05	0.05	0.05
16384	0.35	0.49	0.69	0.73	0.74	0.74
65536	0.43	0.76	1.67	2.03	2.11	2.06
102400	0.29	0.40	0.59	0.63	0.65	0.65
256000	0.39	0.65	1.20	1.37	1.42	1.42
512000	0.44	0.79	1.81	2.21	2.37	2.33
1048576	0.47	0.88	2.37	3.13	3.36	3.36
2097152	0.48	0.91	2.70	3.71	4.12	4.03
4194304	0.49	0.95	3.01	4.24	4.75	4.75
6291456	0.49	0.97	3.28	4.93	5.55	5.40
8388608	0.49	0.97	3.27	4.95	5.62	5.54

The following table displays the array multiply add performance data for varying number of global array sizes and local work sizes. Note that all performance values are in units of GigaMutiply Adds per Second.

Array Size \ Work Size	8	16	64	128	256	512
1024	0.05	0.05	0.05	0.05	0.05	0.05
16384	0.34	0.47	0.68	0.70	0.70	0.71
65536	0.42	0.74	1.58	1.86	1.87	1.90
102400	0.44	0.80	1.95	2.33	2.39	2.39
256000	0.38	0.63	1.14	1.28	1.29	1.30
512000	0.42	0.75	1.68	2.00	2.07	2.09
1048576	0.45	0.84	2.20	2.75	2.86	2.85
2097152	0.46	0.88	2.54	3.27	3.44	3.42
4194304	0.47	0.92	2.84	3.90	4.11	4.12
6291456	0.47	0.93	2.95	4.04	4.25	4.25
8388608	0.47	0.93	2.98	4.11	4.32	4.36

The following graphs displays the performance data from the tables above. The first graph shows global array size versus performance for various number of local work sizes. The second graphs is the inverse; it shows local work size versus performance for various global array sizes.

3. We can see from both figures that a larger local work size results in greater performance. This makes sense since a larger work size means that more things are happening in parallel (provided that the

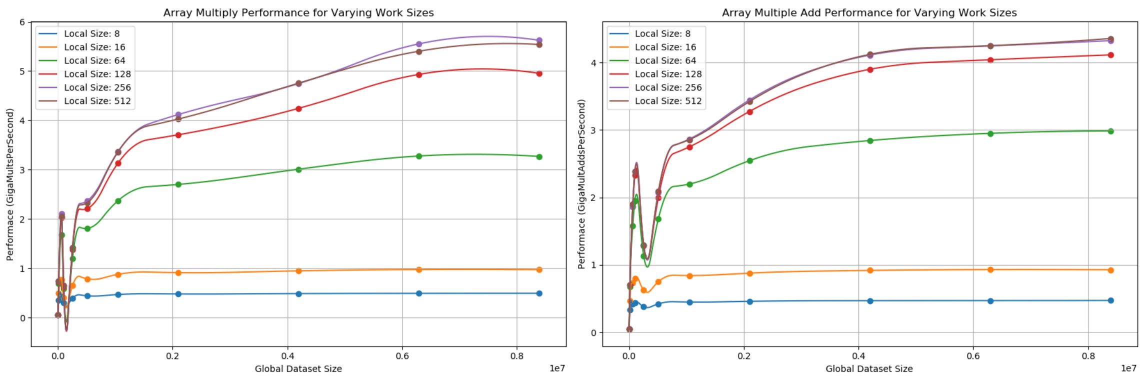

Figure 1: Dataset size vs. Performance for various local work sizes

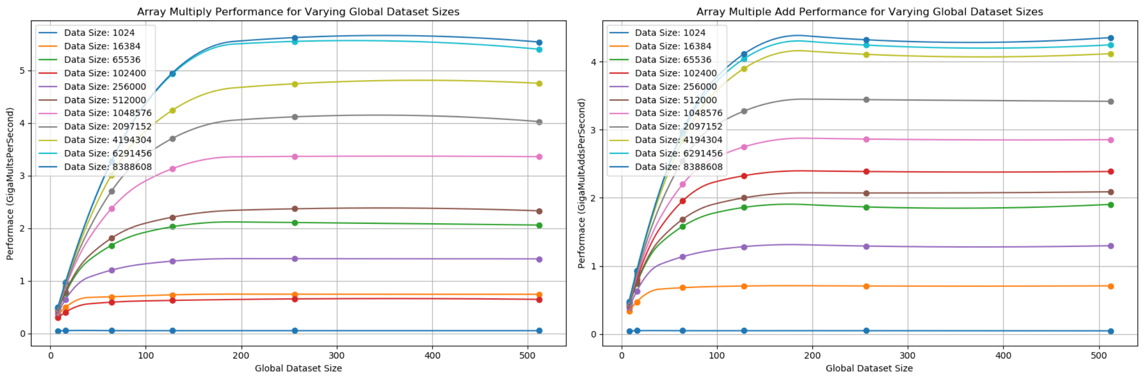

Figure 2: Local work size vs. Performance for various global dataset sizes

GPU is large enough to do so). There does seem to a be a limit to this however, as noted by the similar performance numbers between work sizes of 256 and 512. As with most parallel computation, we can also see that for each work size, there is a certain amount of data that is required before peak performance is reached. This makes sense since a certain amount of computation is necessary to make the overhead of parallelism worth it.

4. Looking at the raw performance numbers, it may seem like the Multiply performs better than the Multiply-Add. However, we need to take into account that Multiply-Add is doing more work than the Multiply; it performs more floating point operations and operates on more data, specifically 200% and 150% more specifically. This contributes to why the raw performance numbers for Multiply-Add is lower than Multiply (i.e. Multiply-Add takes longer than Multiply for the same array size), because Multiply-Add is doing more computation and has to handle more memory. Basically, a GigaMultPerSecond is not equivalent to a GigaMultAddPerSecond. Thus, multiplying all the MultipyAdd values by 2 (MultiplyAdd does double the work of a Multiply assuming floating point multiplication and addition are equivalent computation-wise) we find that MultiplyAdd actually has higher performance, which makes sense since it is doing more work in parallel.

5. For proper use of GPU parallel computing this means, much like with all parallel computing, one should try to do as much computation as possible on the GPU in order to do as much computation in parallel as possible. Furthermore, there seems to be a direct relation between array size and optimal work size. The data gathered here indicated that larger dataset sizes require a larger local work size for optimal performance.

6. The following table displays the array multiply reduce performance for varying number of global array sizes and local work sizes. Note that all performance values are in units of GigaMultiply-Reductions

per Second.

Array Size \ Work Size	32	64	128	256
1024	0.01	0.01	0.01	0.01
16384	0.10	0.11	0.11	0.11
65536	0.30	0.35	0.37	0.36
102400	0.39	0.49	0.52	0.50
256000	0.58	0.81	0.90	0.86
512000	0.69	1.05	1.20	1.13
1048576	0.76	1.22	1.45	1.35
2097152	0.80	1.33	1.60	1.48
4194304	0.82	1.39	1.69	1.55
6291456	0.83	1.42	1.72	1.59
8388608	0.83	1.42	1.74	1.60

The following graph displays the performance from the data above

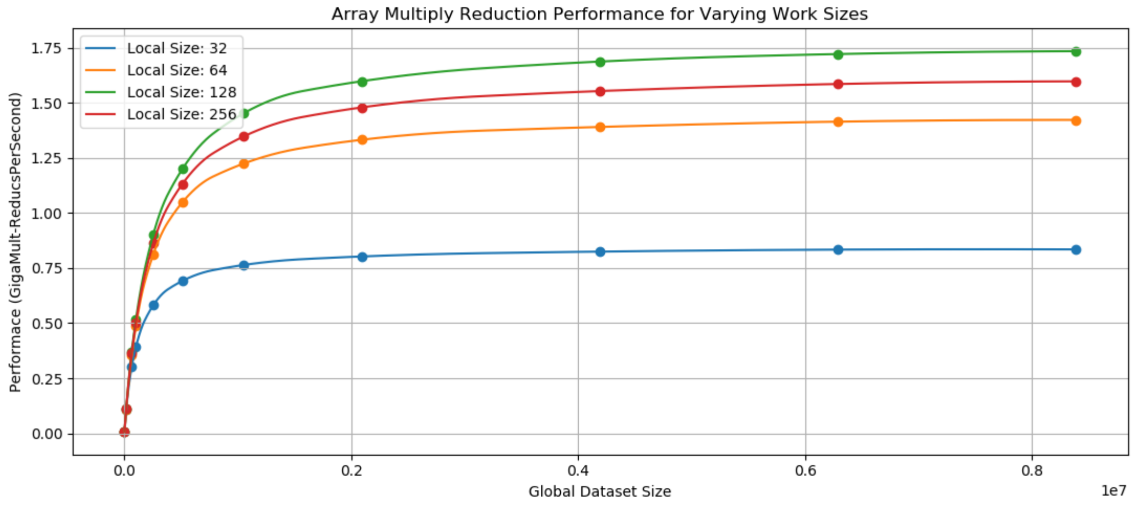

Figure 3: Global dataset size vs. Performance for various local work sizes for array Multiply-Reduce operation

7. We see very similar trends to the graphs for Multiply and Multiply-Reduce, however here we do see that there is a "sweet spot" for the local work size. Now that each thread is doing more computation, it seems like having too many threads can bog down the GPU and result in a decrease in performance. In this case, when each thread and thus work-group takes longer to process, having more, smaller work groups is better so that there are more work-groups to swap in and out to keep the GPU running when a process blocks. For our specific function case and with my GPU, a work group size of 128 seems to be this "sweet spot".

8. Thus for proper GPU parallel computing, thorough testing should be done to determine the proper work group size for the user's specific use case and hardware. For more practical applications, the dataset size if fixed and known before hand. Thus different work group sizes should be tested to find optimal performance. Contrary to our expectations, the largest group size possible may not be what leads to the best possible performance.