Jesse Thoren

May 23, 2017

CS 475 – Parallel Programming

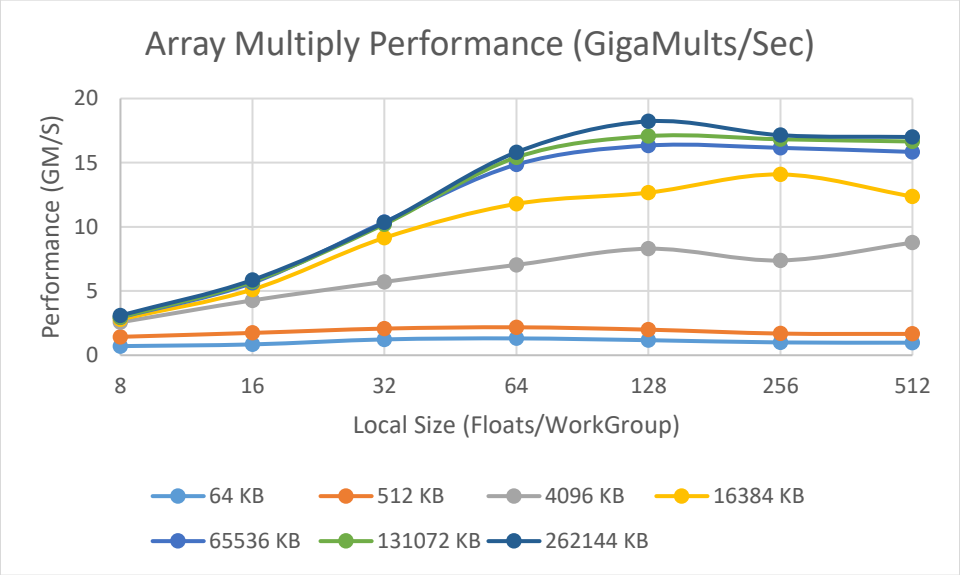
Project 6 – OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce

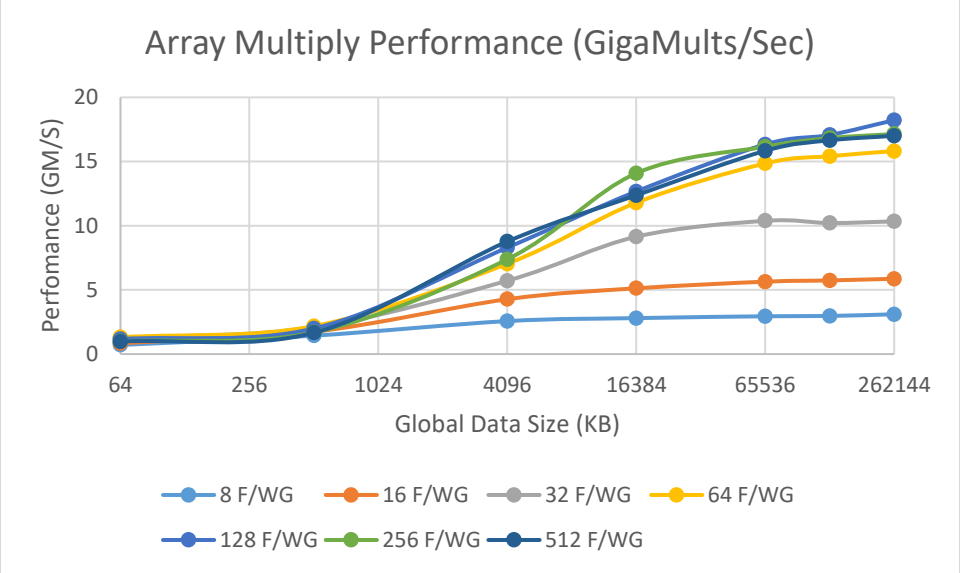
Section 1 – Array Multiply and Array Multiply-Add

1. I ran this report on Rabbit at some ungodly hour of the morning. (Uptime load around .02)
2. Data for Array Multiply:

	8	16	32	64	128	256	512	
64	0.709	0.844	1.233	1.31	1.171	0.997	0.977	64 KB
512	1.433	1.738	2.076	2.177	1.991	1.687	1.66	512 KB
4096	2.569	4.273	5.716	7.039	8.295	7.383	8.769	4096 KB
16384	2.8	5.122	9.145	11.792	12.669	14.087	12.369	16384 KB
65536	2.944	5.639	10.381	14.855	16.336	16.156	15.84	65536 KB
131072	2.97	5.738	10.208	15.417	17.07	16.825	16.653	131072 KB
262144	3.1	5.866	10.339	15.816	18.223	17.146	17.004	262144 KB
	8 F/WG	16 F/WG	32 F/WG	64 F/WG	128 F/WG	256 F/WG	512 F/WG	

Tables for Array Multiply (logarithmic in horizontal-axis):

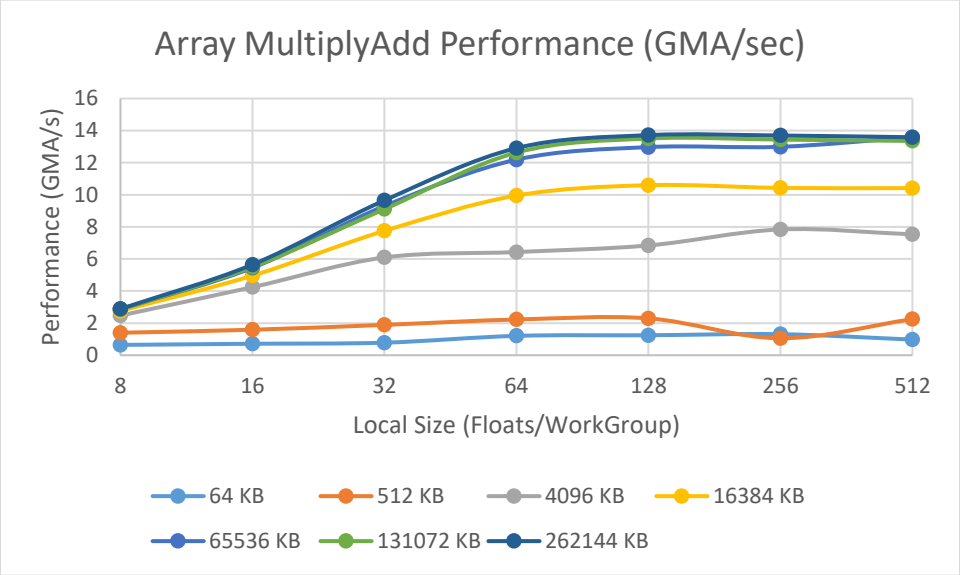


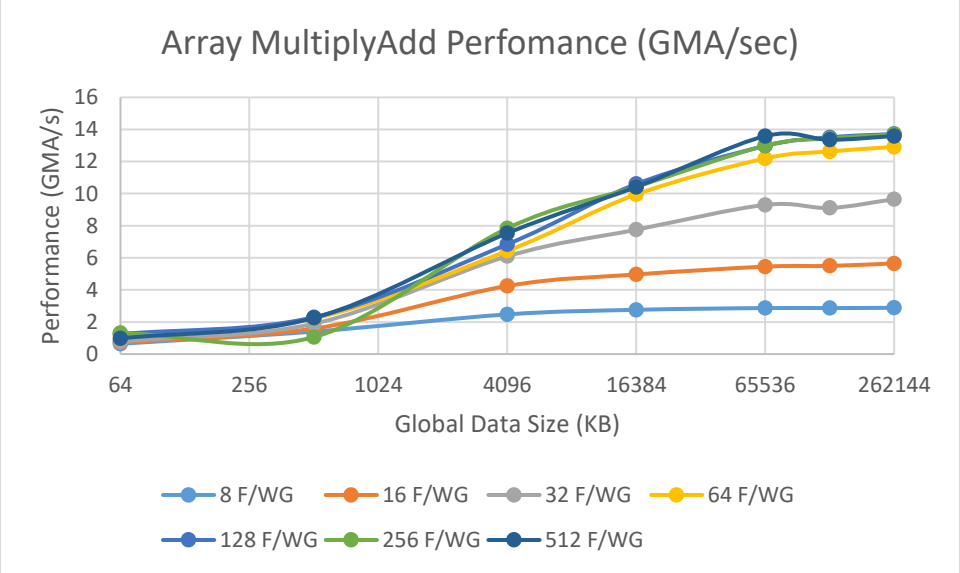


Data for Array Multiply-Add:

	8	16	32	64	128	256	512	
64	0.643	0.715	0.783	1.207	1.245	1.311	0.976	64 KB
512	1.402	1.589	1.893	2.226	2.298	1.058	2.249	512 KB
4096	2.473	4.243	6.095	6.432	6.843	7.84	7.535	4096 KB
16384	2.758	4.96	7.752	9.952	10.599	10.428	10.418	16384 KB
65536	2.868	5.449	9.3	12.192	12.973	12.992	13.588	65536 KB
131072	2.865	5.496	9.107	12.631	13.509	13.44	13.356	131072 KB
262144	2.885	5.65	9.653	12.91	13.719	13.694	13.588	262144 KB
	8 F/WG	16 F/WG	32 F/WG	64 F/WG	128 F/WG	256 F/WG	512 F/WG	

Graphs for Array Multiply-Add (Logarithmic in horizontal-axis):





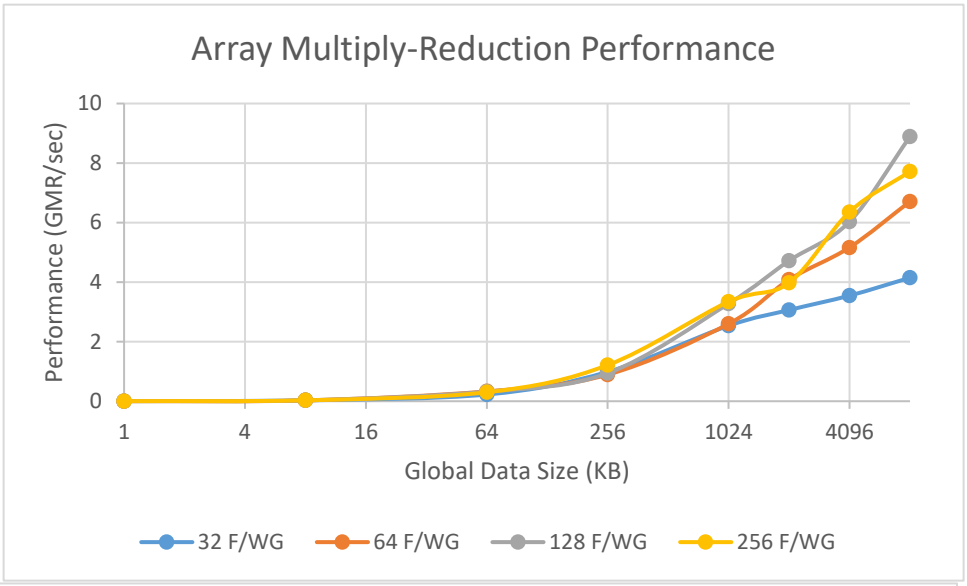
3. The performance curves suggest that OpenCL will produce far better results as you increase your local workgroup size, and your global data size, but both show diminishing returns around a certain point. Optimal performance is achieved around a global data size of 64 Megabytes and a workgroup data size of about 128-256 floats (or .5-1 KB local data size).
4. This performance is reflective of the general idea that GPUs are really good at processing large groups of data, streaming quickly. If the workgroup is too small, the GPU is underutilized because there are compute cores not being used, and if there is too little data to compute, there is a lot of overhead wasted in setting up the gpu for computation, only to use it a couple times before shutting it off.
5. The performance for only Multiply is greater than the performance for Multiply-Add. This is because there is a different operation being used in conjunction with the multiply, which requires the GPU to switch gears between multiplying and adding, and vice-versa, over the time it computes a large set of data.
6. This infers that GPUs are best used a cranking out single operations for a sufficiently large dataset, given sizeable chunks of data at a time.

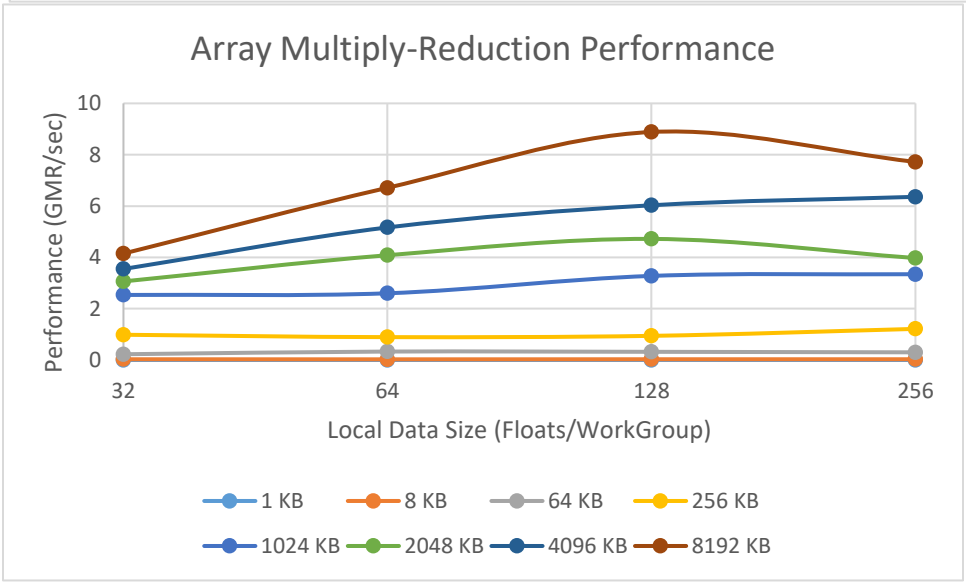
Section 2: Multiply-Reduction

1. Data for Multiply-Reduction:

	32	64	128	256	
1	0.004	0.004	0.004	0.003	1 KB
8	0.031	0.028	0.031	0.031	8 KB
64	0.226	0.331	0.321	0.301	64 KB
256	0.986	0.893	0.943	1.214	256 KB
1024	2.537	2.601	3.279	3.344	1024 KB
2048	3.064	4.084	4.72	3.975	2048 KB
4096	3.548	5.166	6.03	6.359	4096 KB
8192	4.148	6.711	8.886	7.72	8192 KB
	32 F/WG	64 F/WG	128 F/WG	256 F/WG	

Graphs for Multiply-Reduction:





2. Similar to the Array Multiply and Array Multiply-Add graphs above, we are seeing a general growth in performance as the Global Data size increases, and a general improvement in speed until we include 128 floats in a workgroup.
3. We should expect that the performance will increase as global data size increases, at least up until a certain point. Furthermore, we can expect that the performance will increase as the data size increases for the workgroup data size, but we see a slight decrease as we hit 256 floats/workgroup. This is probably happening because we have included a for loop inside our gpu instructions, so the benefit from parallelization on a gpu is being overcome by the performance penalty of doing “CPU types of things” – like looping structures – where we do a lot of different kinds of actions… and a GPU isn’t designed to handle that gracefully.
4. GPUs are best for cranking out single operations on large data sets in parallel. However, there may be some increase in performance (to a point) in doing CPU kinds of operations if we have the data in local memory and don’t want to wait for a lot of memory shuffling between a data computation and a reduction.