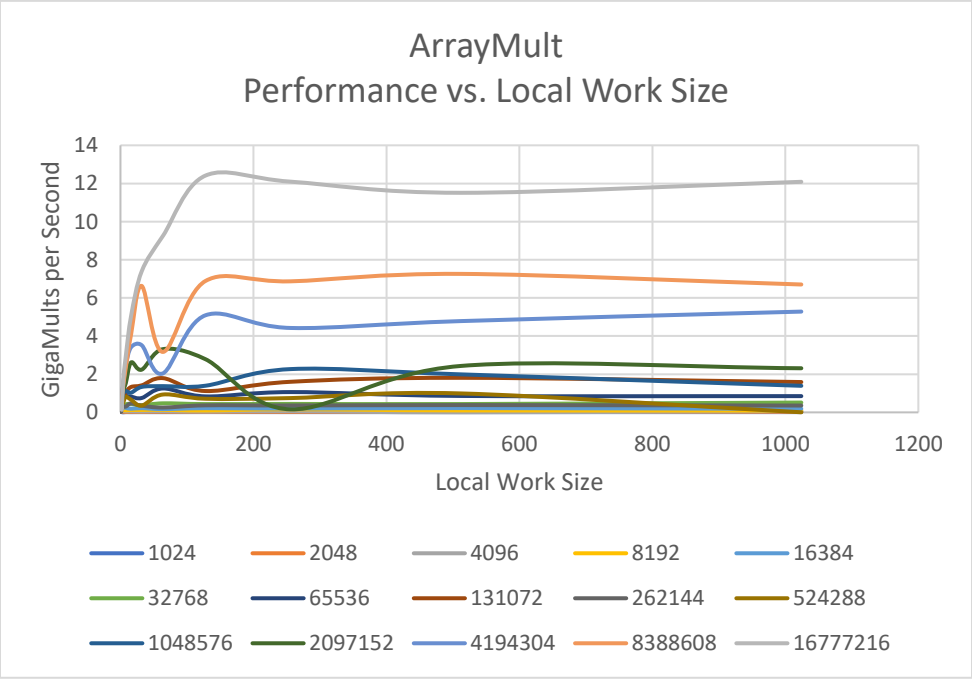
Erin Alltop
CS475 – Spring 2018
Project 6 Commentary

1. I ran my programs on Rabbit. Specifically –
Xeon system
rabbit.engr.oregonstate.edu
2 E5-2630 Xeon Processors
16 Cores total
64 GB of memory
2 TB of disk
Xeon Phi support: icc, icpc, libraries, drivers

2. **Table and Graphs for ArrayMult**

ArrayMult GigaMultsPerSecond with varying Local and Global Work Sizes															
	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216
1	0.014697	0.027653	0.048382	0.103757	0.151566	0.241472	0.266475	0.306041	0.019366	0.279453	0.316285	0.338994	0.352944	0.366732	0.322422
2	0.013391	0.024961	0.05299	0.109914	0.175234	0.359926	0.48185	0.53572	0.254198	0.429592	0.584917	0.630502	0.73907	0.69386	0.132893
4	0.000335	0.029279	0.067144	0.114556	0.263294	0.312997	0.667271	0.613191	0.30601	0.497777	0.955757	1.117463	1.435612	1.348239	1.584392
8	0.011933	0.02035	0.050955	0.105703	0.233397	0.418365	0.843569	1.042836	0.304695	0.813008	1.120268	1.526267	2.466959	2.337539	2.9469
16	0.010242	0.020601	0.053082	0.124227	0.193237	0.445314	0.822996	1.329965	0.46767	0.663219	1.030936	2.617553	3.415084	4.08948	5.057359
32	0.012194	0.021025	0.044879	0.124891	0.21961	0.370385	0.752967	1.416107	0.352334	0.357354	1.334985	2.240995	3.518714	6.627952	7.398438
64	0.01195	0.023985	0.055783	0.132044	0.174817	0.473054	1.251953	1.793517	0.251951	0.935602	1.375485	3.314586	2.046709	3.168929	9.257463
128	0.014388	0.021894	0.037047	0.098352	0.220606	0.434065	0.836804	1.117942	0.369161	0.714264	1.397917	2.783905	5.090657	6.890316	12.41584
256	0.013298	0.023453	0.059232	0.161127	0.201754	0.442315	1.064034	1.6037	0.361409	0.749621	2.268442	0.159556	4.423821	6.869551	12.0941
512	0.013704	0.024331	0.001785	0.103635	0.224365	0.435982	0.860538	1.812139	0.382139	0.991713	1.991249	2.433597	4.785074	7.259332	11.51024
1024	0.013254	0.024453	0.059282	0.095255	0.187531	0.514331	0.851626	1.589638	0.357912	0.003487	1.391629	2.311675	5.283417	6.702688	12.0863



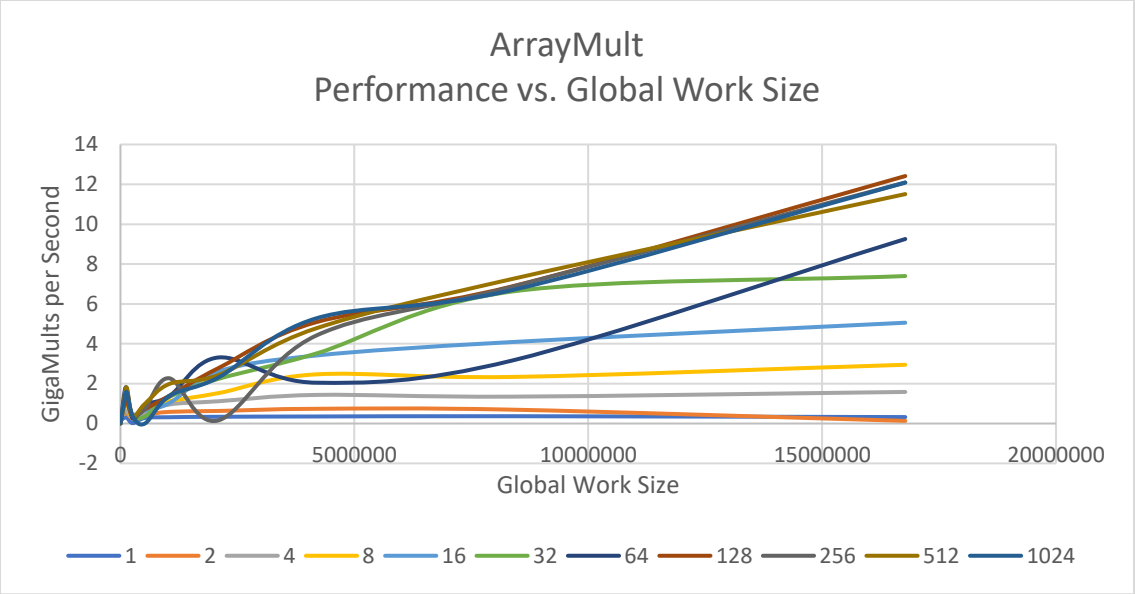
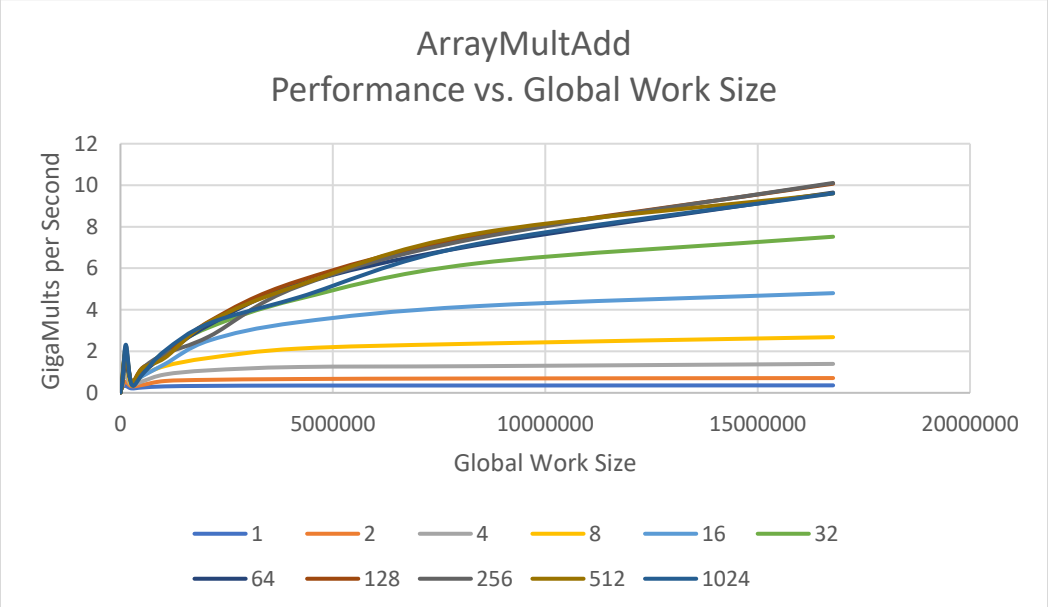
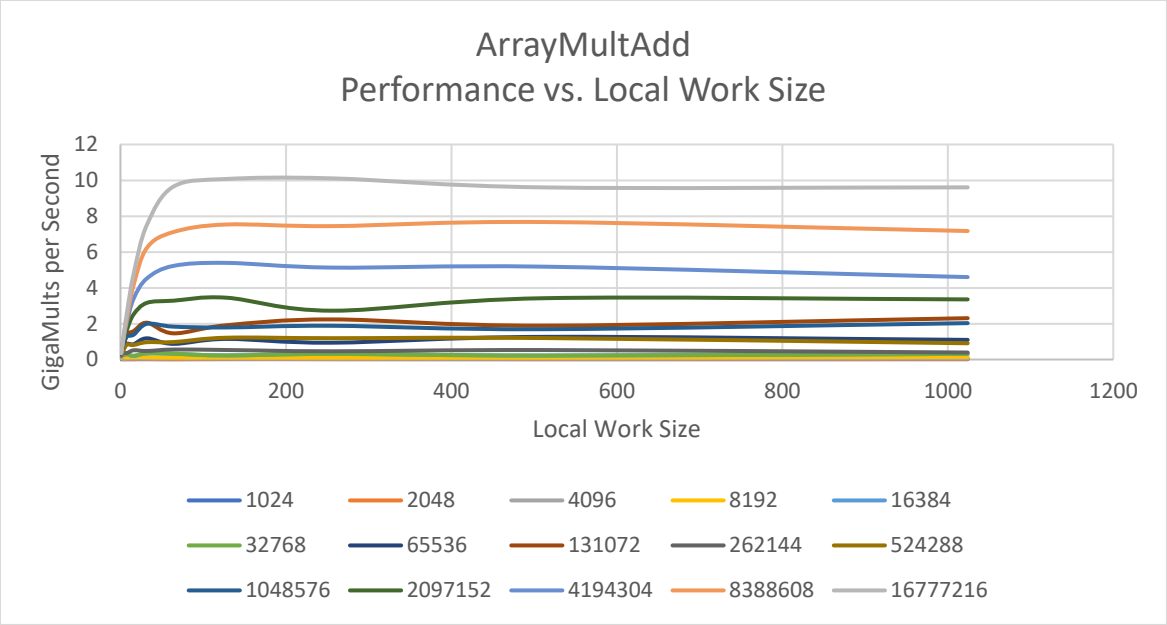


Table and Graphs for ArrayMultAdd

ArrayMultAdd GigaMultsPerSecond with varying Local and Global Work Sizes															
	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216
1	0.013402	0.038389	0.062939	0.115926	0.151853	0.225259	0.318888	0.326653	0.213841	0.258221	0.31019	0.33526	0.349824	0.353059	0.35881
2	0.011017	0.039804	0.066606	0.129518	0.223914	0.223914	0.416094	0.534615	0.317608	0.384597	0.569373	0.626272	0.665427	0.691021	0.711015
4	0.013191	0.024622	0.041154	0.138704	0.250946	0.250946	0.737993	0.793472	0.3794	0.558608	0.884742	1.085219	1.244085	1.284191	1.393552
8	0.013375	0.026011	0.053456	0.117578	0.295212	0.295212	0.883651	1.421097	0.365866	0.834484	1.29005	1.685701	2.132369	2.36909	2.68134
16	0.013204	0.027391	0.068227	0.137434	0.204516	0.204516	0.854892	1.585581	0.514331	0.814242	1.386739	2.526981	3.403829	4.175412	4.801377
32	0.018324	0.035218	0.059015	0.123418	0.341063	0.341063	1.183131	2.055258	0.475647	0.967775	1.991109	3.162624	4.526406	6.23559	7.51888
64	0.012933	0.021897	0.066912	0.105151	0.322348	0.322348	0.873976	1.461976	0.556068	0.972819	1.835424	3.283228	5.224327	7.114	9.647196
128	0.019592	0.036264	0.056549	0.10898	0.228243	0.228243	1.149876	1.910754	0.531152	1.216474	1.788958	3.449916	5.393264	7.533706	10.0741
256	0.019608	0.023454	0.051123	0.104209	0.3144	0.3144	0.939409	2.238672	0.460385	1.19053	1.882887	2.724986	5.129574	7.444252	10.10909
512	0.013936	0.032068	0.05428	0.104966	0.222054	0.222054	1.265614	1.889299	0.527778	1.198085	1.691587	3.421922	5.18098	7.673071	9.603773
1024	0.016104	0.033544	0.068213	0.143262	0.315168	0.315168	1.094858	2.307362	0.390155	0.905766	2.029681	3.354772	4.599461	7.171246	9.606688





3. For ArrayMult, the graphs are a bit all over the place. For performance vs. local work size, there seems to be an initial jump in performance at the lower work sizes, then several dips until there is an evening out of every series at about the 600 work size. When comparing performance with global work size, the graph is significantly different. You do see the same initial jump in performance at the lower work sizes, though much less pronounced. There are some curves, but less so, and then at about the 4M mark or so each series begins a stead incline, some steeper than others.

For the ArrayMultAdd portion, the graphs are more consistent. For performance vs. local work size, there is an initial jump in performance at the lower work sizes, and then a gradual evening out of every series at around the 200 mark. For performance vs. global work size, there is again an initial jump in performance at the beginning, though much less pronounced. Then there is a marked logarithmic increase in performance beginning almost immediately after the first jump in performance.

4. For both graphs when comparing against local work size, there is a gradual evening out of performance. This makes sense because the local group size is the number of processing elements that will be used per compute unit. So increasing the number indefinitely will not guarantee increased performance. In fact, the performance will only be able to increase until the number of processors that you have available to you is maxed out, and then the performance will even out (as we see). For both graphs we seeing the evening out around the 200 mark which makes sense.

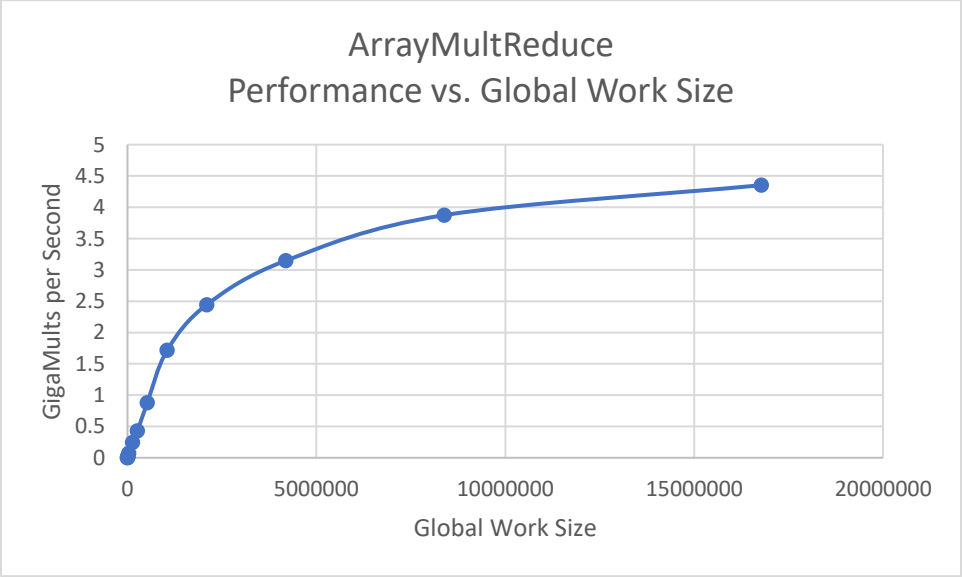
For the both graphs when comparing against global work size, there is a general logarithmic trend in performance. The ArrayMult is a bit all over the place but I think that might have more to do with Rabbit's performance when these results were run and less that it is completely accurate in perfect conditions. You can still see the hint of a logarithmic increase. I believe this has to do with the input sizes given being powers of two in nature. The global work size is not dependent on the number of processors like the local work sizes are, so the performance is able to increase without much trouble from hardware constraints.

5. Like I noted above, ArrayMultAdd and ArrayMult are generally in a “zoomed out” sense somewhat similar, however ArrayMult is a good deal more sporadic with more jumps and curves throughout and less consistent results. I think having the additional adding in ArrayMultAdd helps to stabilize the performance more than in ArrayMult.

6. For proper use of GPU parallel computing, you should make sure that you are tuning your local group sizes to an appropriate amount to be able to maximize the performance you see. You would not want to go with too little processors as your performance will be maxed out. Additionally, you would only want to use GPU parallel computing if you have a large enough data pool to see significant performance gains. All four of the graphs show performance all over the place in smaller data sizes. This would not be good for consistent results if you only have data in these smaller ranges. However, increased, consistent results can be expected with much higher amounts of data – making GPU parallel computing a powerful tool.

7. **Table and graph for ArrayMultReduce**

ArrayMultReduce	
GlobalSize	GigaMults
1024	0.001464
2048	0.0036
4096	0.007008
8192	0.01785
16384	0.027552
32768	0.070568
131072	0.244968
262144	0.427884
524288	0.880288
1048576	1.715265
2097152	2.443447
4194304	3.148662
8388608	3.874483
16777216	4.352488



8. The performance curve in the ArrayMultReduce graph is showing a clear logarithmic shape.
9. Similarly to the other graphs, I believe this graph looks this way in part because the input values selected were powers of two which allow for a consistent performance increase. Also, because the reduction occurs within the same work group with a constant local size, the performance is not contingent on the number of processors that exist in the computing unit.

10. When considering the proper use of GPU parallel computing and looking at the results of this test, you might consider that making calculations within the same workgroup is beneficial to your performance results. When using the same workgroup, the computing unit does not need to go outside to access any memory which would slow performance.