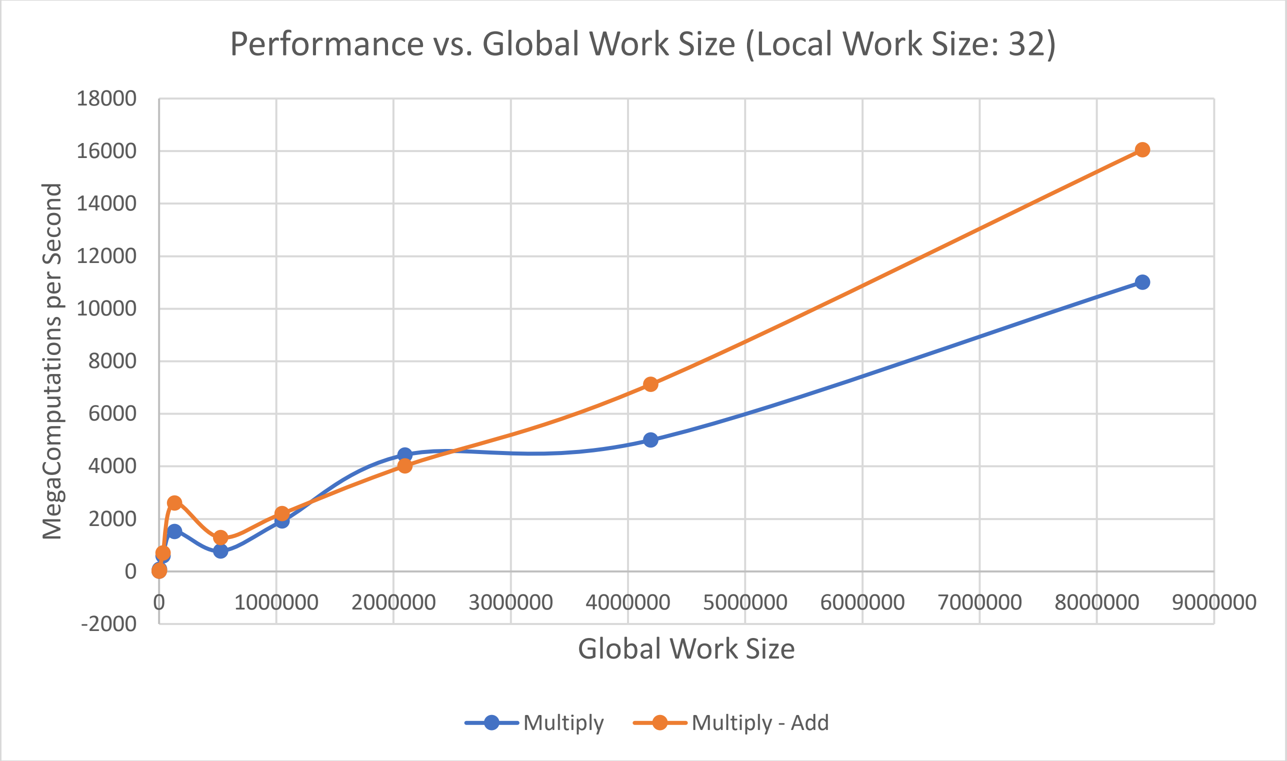
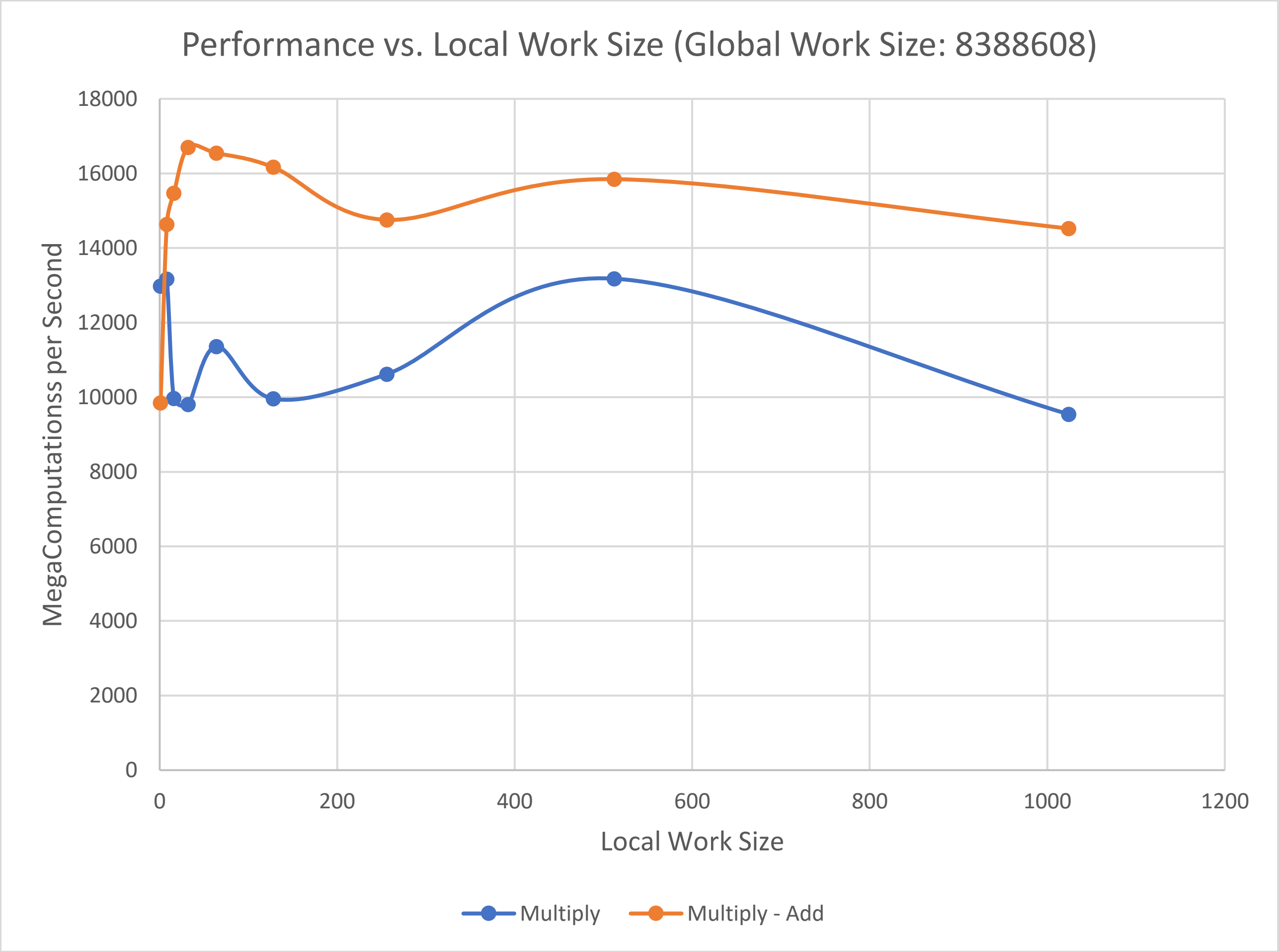
Project 6 Report CS 475 Jacob Leno

1. All of my data was collected from the rabbit server: rabbit.engr.oregonstate.edu

2.





Project 6 Report CS 475 Jacob Leno

Local Work Size: 32	Performance vs. Global Work Size									
Global	1024	2048	4096	32768	131072	524288	1048576	2097152	4194304	8388608
Multiply	13.18	40.42	81.01	595.27	1523.54	779.56	1916.93	4434.91	4997.18	11012.82
Multiply - Add	13.3	26.4	50.84	706.98	2608.71	1293.19	2201.05	4016.35	7120.6	16046.86

Global Work Size: 8388608	Performance vs. Local Work Size*										
Local	1	2	4	8	16	32	64	128	256	512	1024
Multiply	12979.77	10014.66	7984.53	13166.18	9967.03	9802.9	11360.5	9958.97	10618.37	13177.76	9543.31
Multiply - Add	9844.52	10049.14	16067.91	14635	15468.86	16699.06	16544.53	16166.6	14756.02	15845.95	14523.54

*To create more readable data the graph does not have points plotted for local work sizes 2 and 4

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

- When the global work size varies the performance goes up for higher sizes of the global work size (array). When local work size varies the performance increases until 32 where it maxes out and then is smaller for each value after that.

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

- When the size of the global array has smaller values there is a higher ratio of overhead to actual computation vs when the global array has larger values the ratio of overhead to computation is smaller with amount of computation being much larger.
- When the size of the local array has really small values the performance is lower because there are more processing elements in each of the compute units on the device than the local work size and thus we are wasting compute time because we are not taking advantage of all the processing elements the device has to offer and the setup/overhead is outweighing computation
- When the size of the local array has values at 32 and above the work size is the same or larger and since all of the processing elements of the device are being utilized the performance goes up. When the performance starts to go down for larger values it could be because there are so many elements in the local work size that we are now not utilizing the compute units on the device. The local work group size is becoming so large that there is plenty for the processing elements to do in each work group but not all of the array is being spread around to each processing element and thus we lose performance because we don't take advantage of everything the card can give.

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

- The multiply-add has better performance, possibly because there ration of setup to computation is larger and therefore, better performance.

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

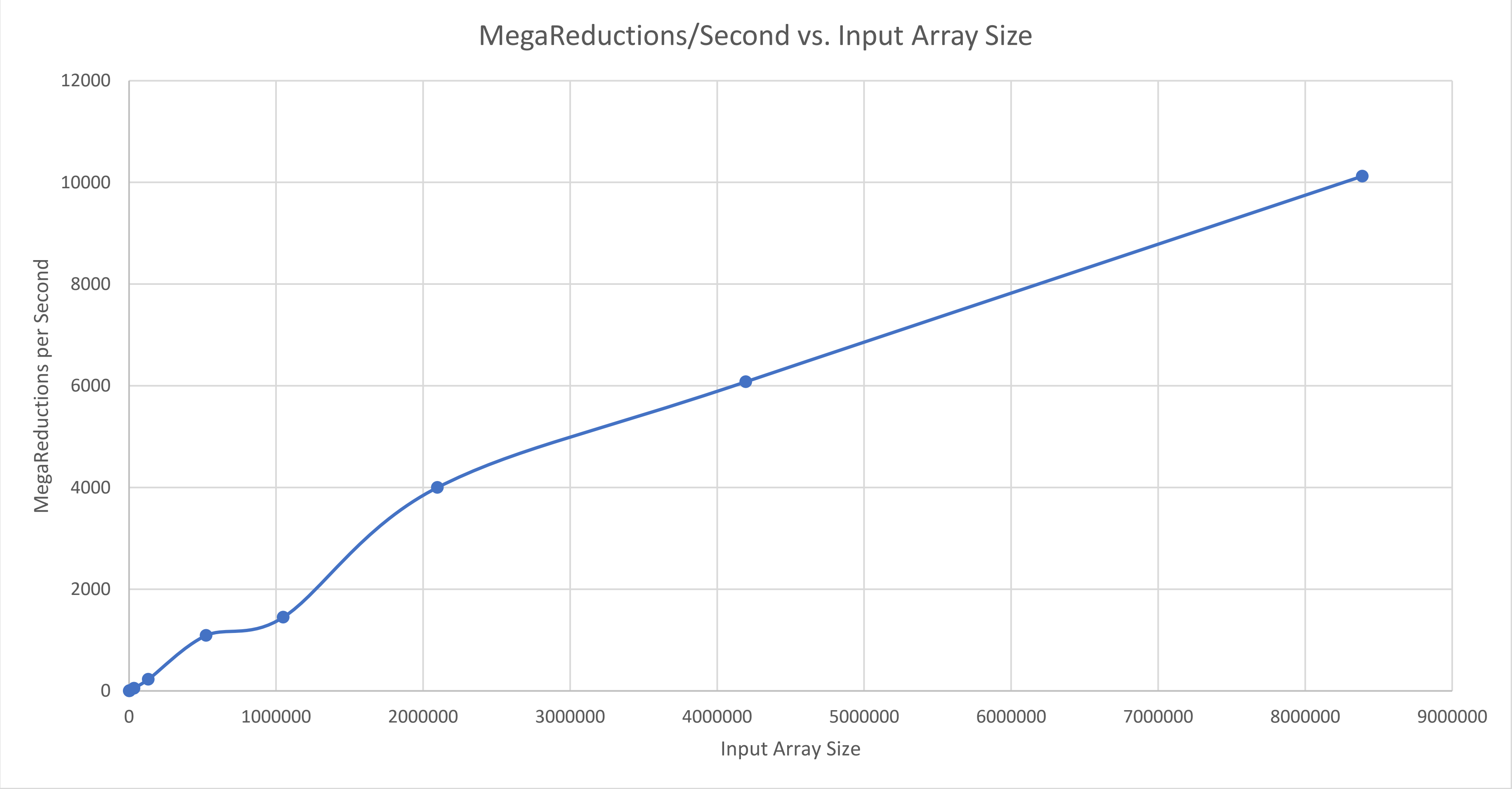
- We need to keep the GPU busy with large data sets and computation every time we enqueue something to take full advantage of its performance.

Project 6 Report CS 475 Jacob Leno

Multiply - Reduce Section:

1.

Local Work Size: 32	MegaReductions/Second vs. Input Array Size									
Global Work Size:	1024	2048	4096	32768	131072	524288	1048576	2097152	4194304	8388608
MegaReductions/Second	1.42	3.14	5.76	53.12	226.66	1089.98	1447.05	3998.99	6078.08	10122.2



2. What pattern are you seeing in this performance curve?

- As the size of the input array increases so does the performance.

3. Why do you think the pattern looks this way?

- Similar to answers to the previous questions in the Multiply and Multiply-Add section. When there are smaller values for the array the ratio of setup to computation is much smaller and for larger values the ratio is much larger. Because this ratio is much larger for larger values more of the devices time is invested into computation and not setup, thus, performance increases.

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

- Also a similar answer to the previous question in the Multiply and Multiply-Add section. This means that we need to keep the GPU busy with large data sets and computation every time we enqueue something to take full advantage of its performance.