Brian Laccone

CS 475

5/20/2019

Project 5

1. I ran this on my own computer. I have a NVIDIA GeForce GTX 1080.
2. Multiply and Multiply-Add Tables and Graphs:

<u>Multiply Table</u>

	8	16	32	64	128	256	512
1024 (Mult)	0.001	0.021	0.023	0.023	0.024	0.02	0.024
2048 (Mult)	0.049	0.048	0.044	0.046	0.048	0.047	0.042
4096 (Mult)	0.096	0.097	0.097	0.097	0.092	0.096	0.097
8192 (Mult)	0.194	0.194	0.192	0.163	0.189	0.192	0.194
16384 (Mult)	0.38	0.391	0.343	0.341	0.325	0.333	0.356
32768 (Mult)	0.608	0.73	0.467	0.662	0.686	0.707	0.605
65536	1.034	0.87	0.798	0.812	0.82	0.832	1.256

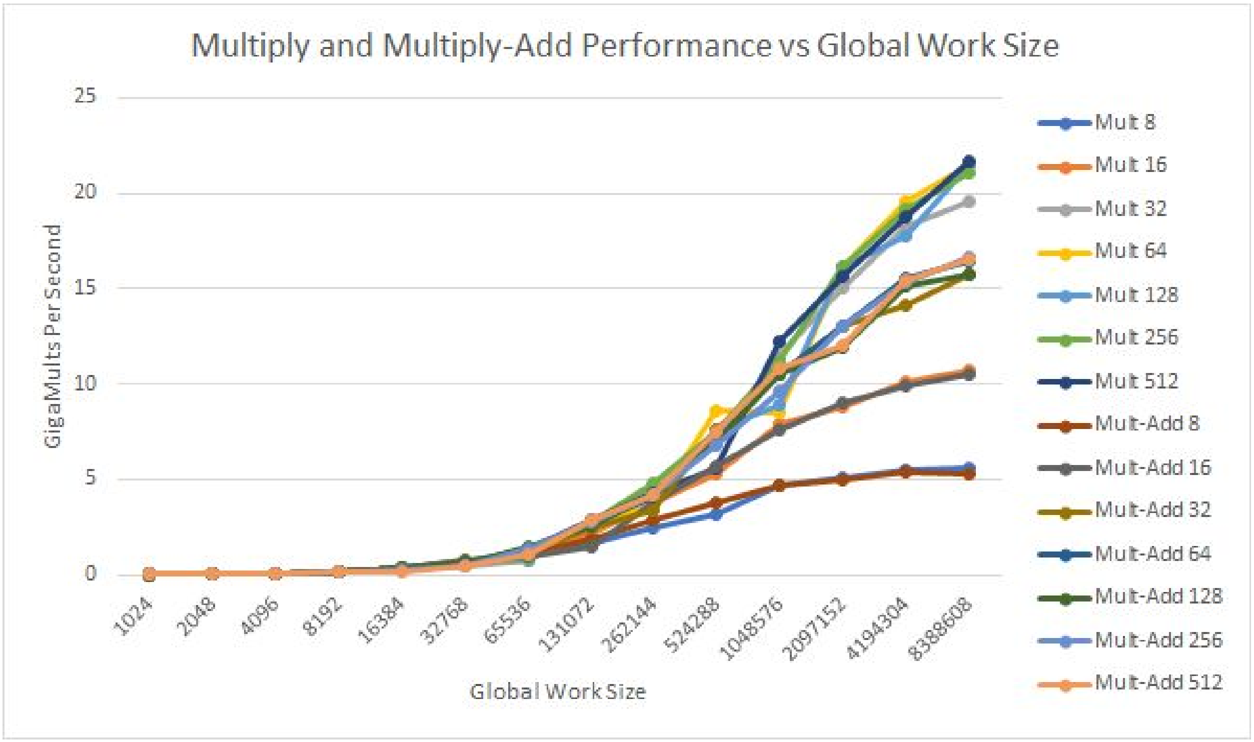
(Mult)							
131072 (Mult)	1.723	2.185	2.815	2.906	2.875	2.743	2.922
262144 (Mult)	2.512	3.826	4.443	3.584	3.968	4.758	4.267
524288 (Mult)	3.162	5.297	7.144	8.636	7.653	7.545	5.585
1048576 (Mult)	4.695	7.92	11.499	8.483	8.941	11.259	12.253
2097152 (Mult)	5.132	8.84	15.064	16.108	16.168	16.168	15.668
4194304 (Mult)	5.46	10.125	18.262	19.571	17.809	19.2	18.801
8388608 (Mult)	5.612	10.742	19.549	21.45	21.45	21.108	21.68

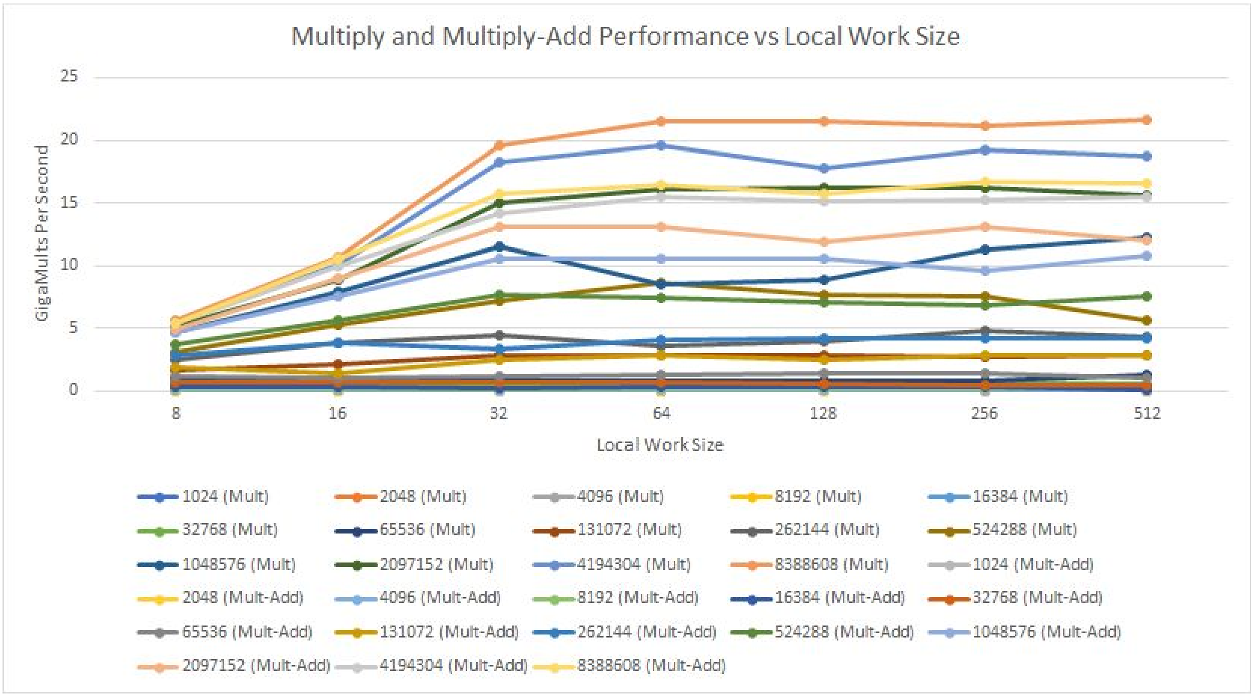
<u>Multiply-Add Table</u>

	8	16	32	64	128	256	512
1024 (Mult-Add)	0.001	0.023	0.024	0.024	0.016	0.024	0.024
2048 (Mult-Add)	0.044	0.047	0.048	0.047	0.032	0.048	0.048

4096 (Mult-Add)	0.097	0.097	0.096	0.088	0.085	0.088	0.096
8192 (Mult-Add)	0.192	0.193	0.188	0.19	0.192	0.193	0.192
16384 (Mult-Add)	0.382	0.386	0.233	0.322	0.363	0.31	0.123
32768 (Mult-Add)	0.715	0.723	0.723	0.7	0.549	0.465	0.509
65536 (Mult-Add)	1.154	1.022	1.179	1.358	1.43	1.386	1.034
131072 (Mult-Add)	1.893	1.437	2.5	2.875	2.536	2.829	2.906
262144 (Mult-Add)	2.86	3.799	3.392	4.135	4.216	4.167	4.233
524288 (Mult-Add)	3.753	5.674	7.653	7.467	7.05	6.784	7.545
1048576 (Mult-Add)	4.7	7.612	10.515	10.567	10.515	9.621	10.806
2097152 (Mult-Add)	4.963	9.054	13.053	13.053	11.881	13.053	12.03
4194304 (Mult-Add)	5.391	9.915	14.159	15.526	15.104	15.305	15.484
8388608 (Mult-Add)	5.346	10.564	15.79	16.494	15.79	16.678	16.565

Graphs


Multiply and Multiply-Add Performance vs Global Work Size
GigaMults Per Second
25
20
15
10
5
0
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
Global Work Size
Mult 8
Mult 16
Mult 32
Mult 64
Mult 128
Mult 256
Mult 512
Mult-Add 8
Mult-Add 16
Mult-Add 32
Mult-Add 64
Mult-Add 128
Mult-Add 256
Mult-Add 512


Multiply and Multiply-Add Performance vs Local Work Size
GigaMults Per Second
25
20
15
10
5
0
8
16
32
64
128
256
512
Local Work Size
1024 (Mult)
2048 (Mult)
4096 (Mult)
8192 (Mult)
16384 (Mult)
32768 (Mult)
65536 (Mult)
131072 (Mult)
262144 (Mult)
524288 (Mult)
1048576 (Mult)
2097152 (Mult)
4194304 (Mult)
8388608 (Mult)
1024 (Mult-Add)
2048 (Mult-Add)
4096 (Mult-Add)
8192 (Mult-Add)
16384 (Mult-Add)
32768 (Mult-Add)
65536 (Mult-Add)
131072 (Mult-Add)
262144 (Mult-Add)
524288 (Mult-Add)
1048576 (Mult-Add)
2097152 (Mult-Add)
4194304 (Mult-Add)
8388608 (Mult-Add)

3. I first pattern that I noticed when assessing the performance curves was that the lower global sizes had very poor performance regardless of the local work size. Anything below 65k was very poor. The differences between the local work sizes seemed to become more apparent the larger the global size grew. By the time the global work size grew to 8M, the local work size difference was maximized and showed that 8 and 16 local work sizes performed very poorly compared to the 32, 64, 128, 256, and 512 local work sizes. It is interesting that there was very little difference between the rest of the local work sizes though. I would have thought that 512 would have a lot better performance than 32. It's also worth mentioning that Multiply was a little faster than Multiply-Add.

4. I believe that the patterns look the way they do because of two reasons: global work sizes being too little and local work sizes being too little. If the global work sizes are too small such as less than 100,000 then the advantages of having work group division is not in full effect. The amount of work is too small to benefit from different local work sizes and splitting up the work among them. When the local work sizes were at 8 and 16 the performance was very low compared to the rest of the local work sizes. I believe that this is because it isn't dividing up the work enough. It's requiring that only 8 or 16 work groups handle 8M when that could have been spread out between more local work sizes and distribute the work more evenly. It's very interesting that 32 - 512 work sizes didn't have much of a difference between all the global work sizes. I would be curious to see if beyond 8M makes the difference increase.

5. The performance of Multiply was a little bit faster than Multiply-Add. At 8M global work size and 512 local work size, multiply was about 5 GigaMultsPerSecond faster. I'm not sure if this would qualify as a large increase or not but to me this seems like a small difference between the two. Multiply is faster at almost every global work size and local work size than Multiply-Add. This is to be expected though as Multiply-Add has to take an extra step for adding in addition to multiplying.

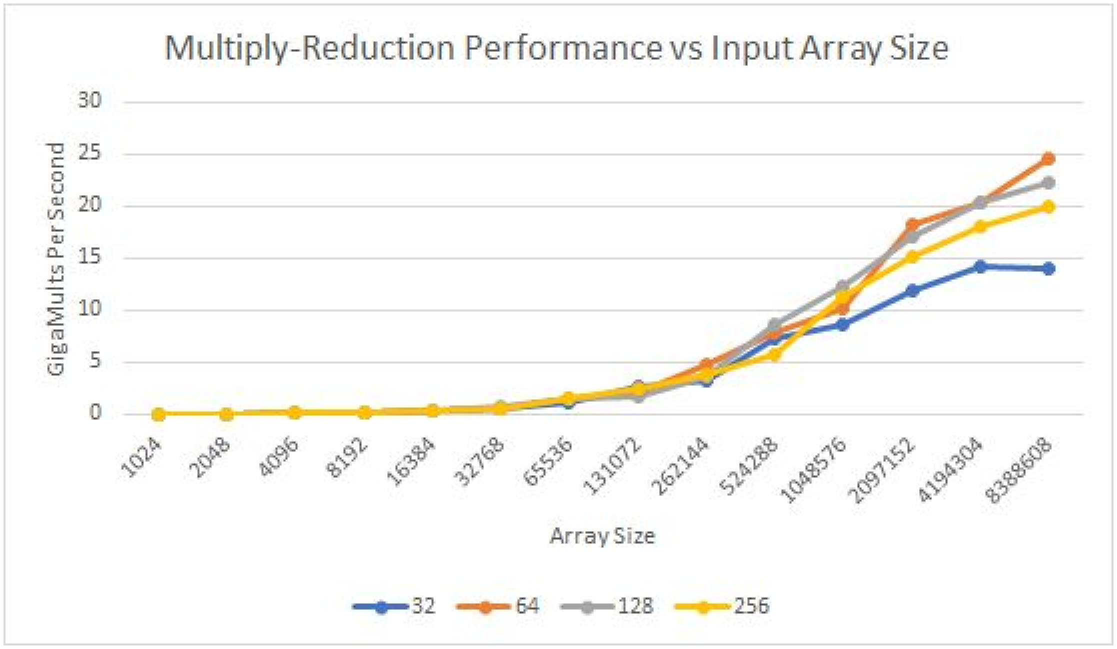
6. The results of comparing performances between the two show that the local work size does matter when trying to achieve high speeds with GPU parallel computing. If the local work sizes are too small or too large then speeds with drop because work is not being distributed efficiently to achieve the best possible speeds. The global work size also matters because based off of my

results, if the global work size is too small then performance will dip because there isn't enough work to even be distributed and have the local work size take advantage of its divisions.

1. Multiply-Reduce Table and Graph:

Multiply-Reduce

	32	64	128	256
1024	0.002	0.015	0.017	0.008
2048	0.044	0.047	0.047	0.047
4096	0.093	0.094	0.095	0.088
8192	0.188	0.19	0.192	0.191
16384	0.382	0.382	0.378	0.384
32768	0.533	0.509	0.704	0.631
65536	1.097	1.502	1.519	1.527
131072	2.572	2.036	1.712	2.433
262144	3.288	4.822	3.682	3.786
524288	7.339	7.848	8.602	5.734
1048576	8.533	10.072	12.288	11.378
2097152	11.947	18.301	17.169	15.17
4194304	14.182	20.287	20.456	18.109
8388608	14.107	24.576	22.255	19.911



2. The performance curve of Multiply-Reduce vs Input Array Size is very similar to that of the multiply graph. When the input array size is small then the performance is very poor regardless of the local work sizes. When the input array size grows then the performance also grows in proportion to the input array size. A pattern that differs in this graph compared to the Multiply and Multiply-Add graphs is that 64 and 128 work sizes out performed the 32 and 256 work sizes. In the previous graph there was very little difference from 32 to 512 but in this graph the performance did vary from 32 to 256. 64 local work size was the best performing size on almost every input array size. 32 local work size was very poor compared to the rest of the local work sizes.

3. I believe that the pattern caused by the local work size is more prominent in the Multiply-Reduction than the other two because the way reduction is being done. Reduction is a more efficient way of using the hardware which is why the differences show up more in the local work sizes. Cache problems could also be causing different local work sizes to completely change the speed of the program. I still don't know why exactly 64 local work size was the best performing local work size compared to the other 3 sizes. Maybe it was as simple as cache problems or that my computer wasn't working on as much things in the background as when it ran the 64 local sizes.Regardless of the answer, it is clear that 64 was the sweet spot for local

work sizes. The reason that the performance was so low during smaller input array sizes is because there is simply not enough work to take advantage of the work groups.

4. The multiply-reduction performance shows that local work size and input array sizes matter when dealing with certain programs. If this test would not have been run then I would not have known that a local work size of 64 performed the best out of the other work sizes. This can make a huge difference when trying to make programs run as fast as possible. This, of course, means more work will need to be done to determine which is the best combination for each machine that the program will be running on but at least it shows that you can increase performance by changing different work sizes. This also means that the input array size will also make a big difference for proper GPU parallel computing. Smaller input array sizes or global work sizes makes a big difference in terms of performance. It may not even be worth trying to improve performance when dealing with such small amount of sizes. Overall the results of this project show that GPU parallel computing can make a big difference in performance, the hard part is determining which work sizes to use.