Hsiang Lo

Project #5

Goal - Learn about Array Multiply, Multiply-Add and Multiply-Reduce using OpenCL

Procedures

1. Multiply two arrays together using OpenCL (benchmark against both input array size and the local work size).
2. Multiply two arrays together (benchmark against both input array size and the local work size).
3. Make two graphs – 1) Multiply and Multiply-Add performance versus Global Work Size with a series of colored Constant-Local-Work-Size-curves. 2) Multiply and Multiply-Add performance versus Local Work Size, with a series of colored Cosntant-Global-Work-Size curves
4. Commentary PDF questions 1- 6
5. **Then write another version of the code that turns it into a Multiply+Reduce application.** (Using same code as 1.) (Note: this will compute a single floating point)
6. Plot another graph showing Multiply-reduction performance versus Input Array
7. Add to the PDF the additional 1-4 quesitons on step 14.

Notes

- It looks like I won't be able to use FLIP server for this assignment, ill be using Rabbit Machine instead.
- Array Multiply and the Array Multiply-Add can really be the same program. Write one program that creates the 4 arrays. Pass A,B and C into OpenCL and return D. Tehn all you have to do between the Multiply and the Multiply-Add tests is change one line in the .cl file.
- Global work sizes – 1K to 8M
- Local work sizes – 8 to 512
- Performance unit: MegaMultiplies Per Second and MegaMultiply-Adds Per Second

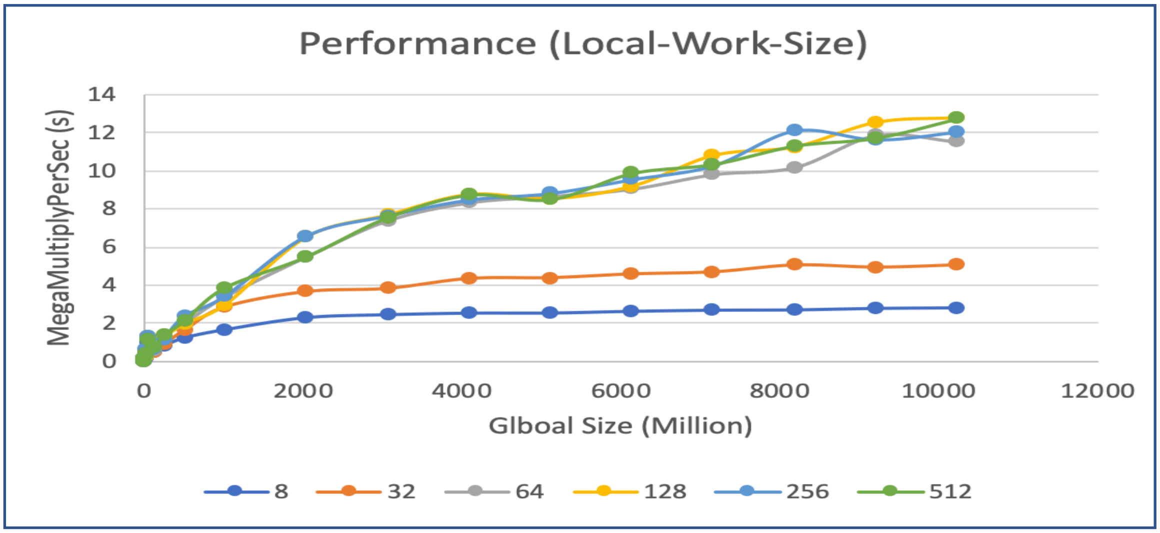
Questions

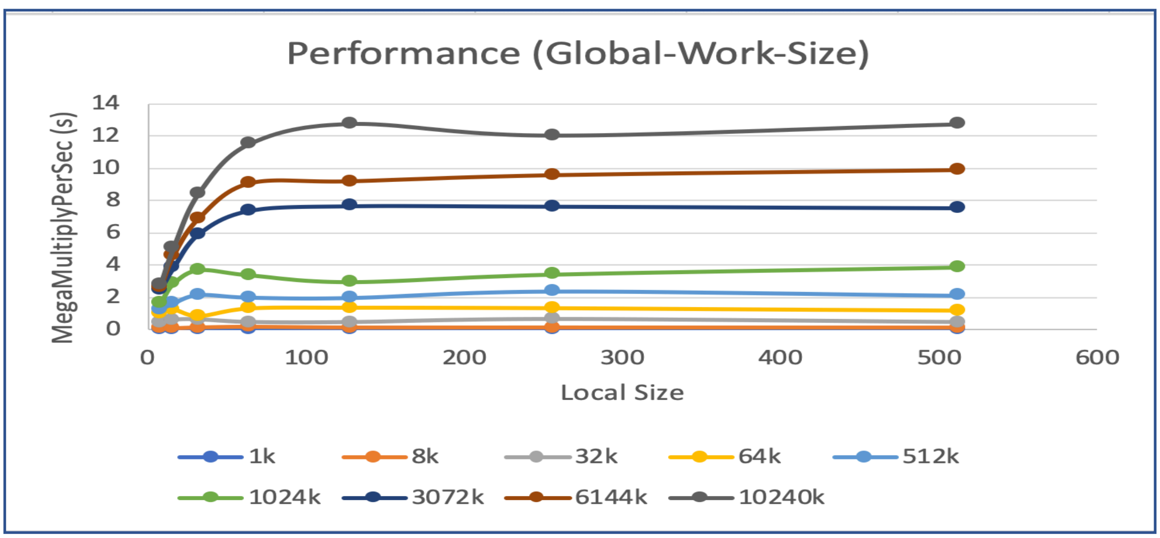
1. What machine you ran this on

I ran this assignment/homework using my MacBook Pro 2018 as I did with my previous assignments. Due to the fact that Flip is not capable of running this assignment, and the fact that I did not want to install any additional softwares, I ran this assignment on Rabbit (rabbit.engr.oregonstate). As requested, I scheduled a session and ran during my session.\

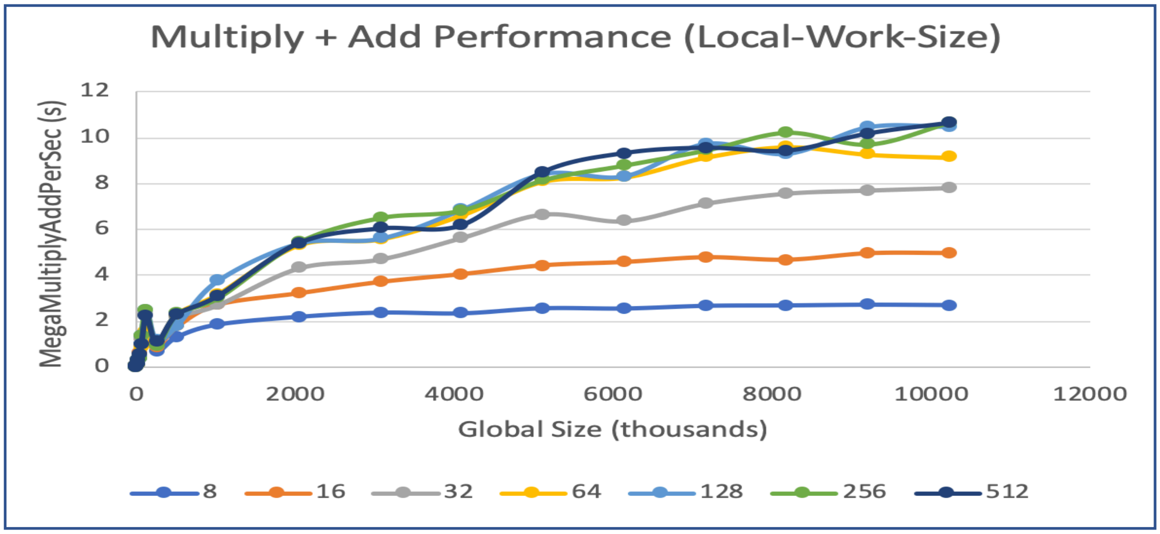
2. Show the tables and graphsMake two graphs:

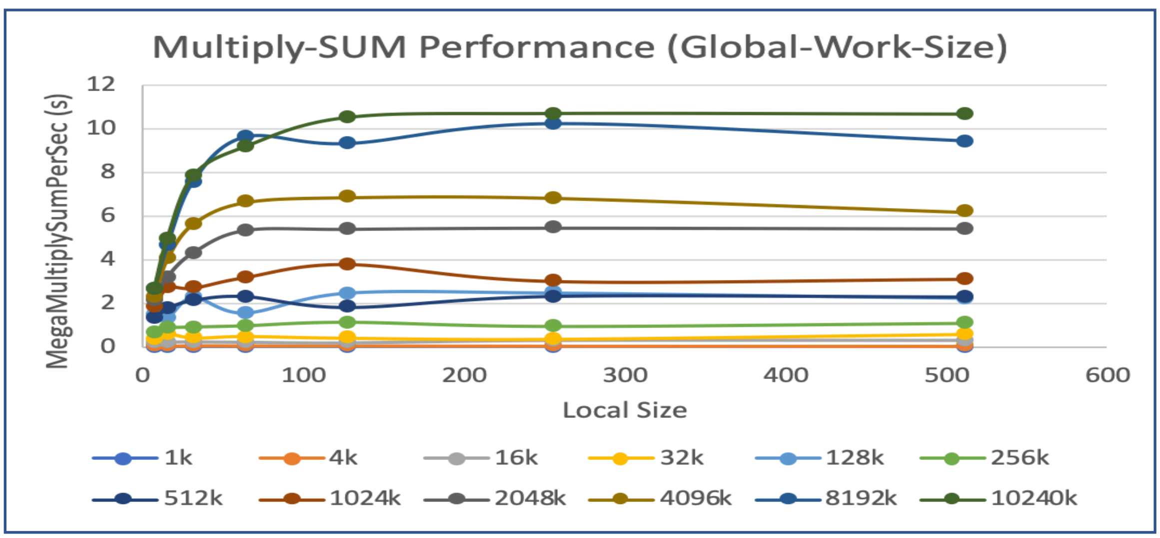
- Multiply and Multiply-Add performance versus Global Work Size, with a series of colored Constant-Local-Work-Size curves





- Multiply and Multiply-Add performance versus Local Work Size, with a series of colored Constant-Global-Work-Size curve





Data for both Multiply and Multiply + Sum

Global Size	Local Size	Work Groups	MegaMultiplyPerSecond
1	8	128	0.014
4	8	512	0.079
8	8	1024	0.131
16	8	2048	0.153
32	8	4096	0.356
64	8	8192	0.894
128	8	16384	0.363
256	8	32768	0.645
512	8	65536	1.188
1024	8	131072	1.572
2048	8	262144	2.184
3072	8	393216	2.449
4096	8	524288	2.53
5120	8	655360	2.538
6144	8	786432	2.626
7168	8	917504	2.685
8192	8	1048576	2.703
9216	8	1179648	2.78
10240	8	1310720	2.809
1	16	64	0.016
4	16	256	0.08
8	16	512	0.082
16	16	1024	0.217
32	16	2048	0.585
64	16	4096	1.202
128	16	8192	0.437
256	16	16384	0.877
512	16	32768	1.617
1024	16	65536	2.84
2048	16	131072	3.636
3072	16	196608	3.806
4096	16	262144	4.32
5120	16	327680	4.364
6144	16	393216	4.558
7168	16	458752	4.669
8192	16	524288	5.024
9216	16	589824	4.913
10240	16	655360	5.031
1	32	32	0.013
4	32	128	0.055
8	32	256	0.112
16	32	512	0.17
32	32	1024	0.609
64	32	2048	0.8
128	32	4096	0.44
256	32	8192	0.955
512	32	16384	2.139
1024	32	32768	3.68
2048	32	65536	5.19
3072	32	98304	5.92
4096	32	131072	6.635
5120	32	163840	6.681
6144	32	196608	6.876
7168	32	229376	6.971
8192	32	262144	7.752
9216	32	294912	8.272
10240	32	327680	8.446
1	64	16	0.017
4	64	64	0.062

Gloval Size	Local Size	Work Groups	MultiplySumPerSecond
1	8	128	0.013
4	8	512	0.082
8	8	1024	0.133
16	8	2048	0.214
32	8	4096	0.378
64	8	8192	1.004
128	8	16384	1.482
256	8	32768	0.687
512	8	65536	1.311
1024	8	131072	1.842
2048	8	262144	2.176
3072	8	393216	2.355
4096	8	524288	2.325
5120	8	655360	2.542
6144	8	786432	2.533
7168	8	917504	2.647
8192	8	1048576	2.665
9216	8	1179648	2.695
10240	8	1310720	2.679
1	16	64	0.018
4	16	256	0.047
8	16	512	0.137
16	16	1024	0.223
32	16	2048	0.596
64	16	4096	1.21
128	16	8192	1.348
256	16	16384	0.875
512	16	32768	1.767
1024	16	65536	2.716
2048	16	131072	3.212
3072	16	196608	3.699
4096	16	262144	4.038
5120	16	327680	4.418
6144	16	393216	4.563
7168	16	458752	4.766
8192	16	524288	4.66
9216	16	589824	4.94
10240	16	655360	4.949
1	32	32	0.018
4	32	128	0.056
8	32	256	0.143
16	32	512	0.233
32	32	1024	0.42
64	32	2048	1.261
128	32	4096	2.271
256	32	8192	0.921
512	32	16384	2.142
1024	32	32768	2.705
2048	32	65536	4.317
3072	32	98304	4.707
4096	32	131072	5.641
5120	32	163840	6.658
6144	32	196608	6.37
7168	32	229376	7.134
8192	32	262144	7.576
9216	32	294912	7.709
10240	32	327680	7.818

8192	64	131072	10.177
9216	64	147456	11.852
10240	64	163840	11.512
1	128	8	0.017
4	128	32	0.067
8	128	64	0.11
16	128	128	0.221
32	128	256	0.432
64	128	512	1.317
128	128	1024	0.674
256	128	2048	1.229
512	128	4096	1.956
1024	128	8192	2.953
2048	128	16384	6.504
3072	128	24576	7.664
4096	128	32768	8.73
5120	128	40960	8.512
6144	128	49152	9.169
7168	128	57344	10.777
8192	128	65536	11.231
9216	128	73728	12.512
10240	128	81920	12.757
1	256	4	0.017
4	256	16	0.073
8	256	32	0.11
16	256	64	0.335
32	256	128	0.629
64	256	256	1.285
128	256	512	0.622
256	256	1024	1.145
512	256	2048	2.357
1024	256	4096	3.412
2048	256	8192	6.569
3072	256	12288	7.638
4096	256	16384	8.472
5120	256	20480	8.82
6144	256	24576	9.544
7168	256	28672	10.28
8192	256	32768	12.108
9216	256	36864	11.637
10240	256	40960	12.042
1	512	2	0.017
4	512	8	0.057
8	512	16	0.107
16	512	32	0.326
32	512	64	0.433
64	512	128	1.135
128	512	256	0.685
256	512	512	1.346
512	512	1024	2.094
1024	512	2048	3.851
2048	512	4096	5.48
3072	512	6144	7.542
4096	512	8192	8.724
5120	512	10240	8.495
6144	512	12288	9.863
7168	512	14336	10.344
8192	512	16384	11.307
9216	512	18432	11.725
10240	512	20480	12.74

4	128	32	0.053
8	128	64	0.161
16	128	128	0.194
32	128	256	0.412
64	128	512	1.385
128	128	1024	2.479
256	128	2048	1.147
512	128	4096	1.825
1024	128	8192	3.763
2048	128	16384	5.393
3072	128	24576	5.61
4096	128	32768	6.866
5120	128	40960	8.421
6144	128	49152	8.327
7168	128	57344	9.739
8192	128	65536	9.306
9216	128	73728	10.464
10240	128	81920	10.481
1	256	4	0.014
4	256	16	0.047
8	256	32	0.103
16	256	64	0.298
32	256	128	0.37
64	256	256	1.356
128	256	512	2.481
256	256	1024	0.947
512	256	2048	2.322
1024	256	4096	3.007
2048	256	8192	5.444
3072	256	12288	6.504
4096	256	16384	6.835
5120	256	20480	8.148
6144	256	24576	8.791
7168	256	28672	9.449
8192	256	32768	10.219
9216	256	36864	9.708
10240	256	40960	10.661
1	512	2	0.015
4	512	8	0.044
8	512	16	0.102
16	512	32	0.3
32	512	64	0.564
64	512	128	0.996
128	512	256	2.245
256	512	512	1.095
512	512	1024	2.296
1024	512	2048	3.103
2048	512	4096	5.411
3072	512	6144	6.041
4096	512	8192	6.187
5120	512	10240	8.502
6144	512	12288	9.319
7168	512	14336	9.551
8192	512	16384	9.427
9216	512	18432	10.174
10240	512	20480	10.634

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

In the case of both Array Multiply and Array Multiply Add, the performance seems to increase as local size increases but only for a short while. Once it hits the a certain point, the performance becomes more steady and the slope in terms of increases in performance slows down.

On the other hand, as the size of the global size increases, it can be observed that the performance increases as well. However it can be noticed that reaching around some size, the performance again slows down just like the local sizes.

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

    It would appear that when the size input is too small, we are not fully utilizing the resource available to us, in this case, the processing elements. As the processing elements reach a desirable size, which can be observed to be around 128, we are fully utilizing the resource, and therefore the performance reached its height.

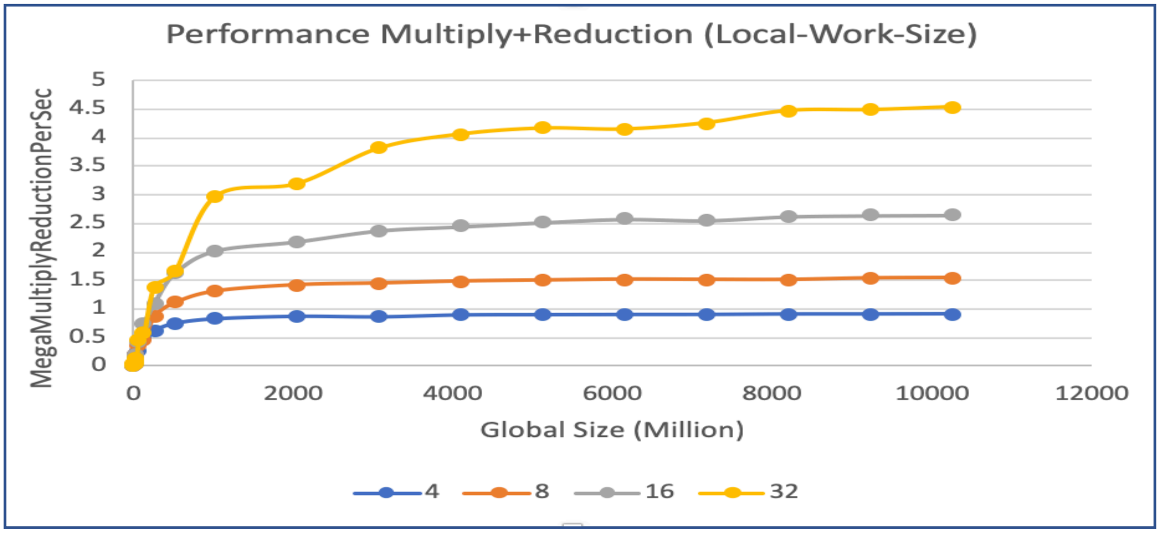
5. What is the performance difference between doing a Multiply and doing a Multiply-Add?

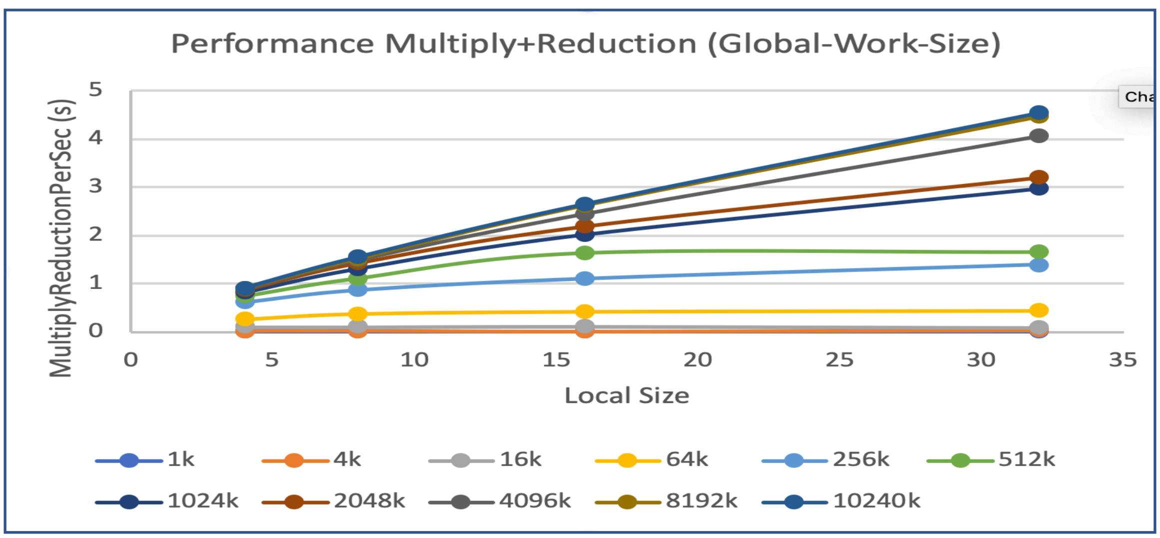
    Performance difference between doing a Multiply and doing a Multiply-Add can be observed in that the performance for Multiply-Add is lower than just Multiply. This can be categorized as a result for doing more work for the processors, which led to lower performance. Furthermore, the performance across Multiply-Add can be seen to be a step or two below Multiply at each variable sizes.

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

    In order to fully utilize the benefit of GPU parallel computing, the more data parallelism the better. In the case of Multiply vs Multiply + Add, it can be argued that Multiply is a more straight forward than Multiply + Add in that there is less commands and more data processing. Regardless, processing an adequate amount of data sizes is ideal In using GPU parallel computing.

7. Show this table and graph - Multiply-reduction performance versus Input Array Size.





Data for Multiply + Reduction

Global Size	Local Size	Work Groups	MegaMultiplyReductionPerSec
1	4	256	0.007
4	4	1024	0.027
8	4	2048	0.057
16	4	4096	0.11
32	4	8192	0.197
64	4	16384	0.265
128	4	32768	0.505
256	4	65536	0.625
512	4	131072	0.748
1024	4	262144	0.826
2048	4	524288	0.868
3072	4	786432	0.859
4096	4	1048576	0.892
5120	4	1310720	0.897
6144	4	1572864	0.9
7168	4	1835008	0.901
8192	4	2097152	0.909
9216	4	2359296	0.91
10240	4	2621440	0.913
1	8	128	0.007
4	8	512	0.028
8	8	1024	0.059
16	8	2048	0.111
32	8	4096	0.194
64	8	8192	0.376
128	8	16384	0.462
256	8	32768	0.873
512	8	65536	1.114
1024	8	131072	1.317
2048	8	262144	1.426
3072	8	393216	1.456
4096	8	524288	1.49
5120	8	655360	1.507
6144	8	786432	1.522
7168	8	917504	1.518
8192	8	1048576	1.517
9216	8	1179648	1.544
10240	8	1310720	1.551
1	16	64	0.007
4	16	256	0.02
8	16	512	0.033
16	16	1024	0.117
32	16	2048	0.214
64	16	4096	0.423
128	16	8192	0.75
256	16	16384	1.107
512	16	32768	1.635
1024	16	65536	2.026
2048	16	131072	2.187
3072	16	196608	2.377
4096	16	262144	2.449
5120	16	327680	2.521
6144	16	393216	2.578
7168	16	458752	2.556
8192	16	524288	2.624
9216	16	589824	2.642
10240	16	655360	2.649
1	32	32	0.007
4	32	128	0.031
8	32	256	0.06
16	32	512	0.103
32	32	1024	0.137
64	32	2048	0.442
128	32	4096	0.57
256	32	8192	1.394
512	32	16384	1.655
1024	32	32768	2.978
2048	32	65536	3.2
3072	32	98304	3.817
4096	32	131072	4.063
5120	32	163840	4.167
6144	32	196608	4.151
7168	32	229376	4.26
8192	32	262144	4.473
9216	32	294912	4.489
10240	32	327680	4.539
1	32	32	0.002
4	32	128	0.03
8	32	256	0.04
16	32	512	0.075
32	32	1024	0.197
64	32	2048	0.21

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

In comparison to Multiply and Multiply + Add, Multiply + Reduction shows a graph of constant increases in performance when both the global and local sizes reaches an adequate level. This is truer for global sizes. For local sizes, signs of a slowdown for even large value of local size can be observed

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

    The most crucial difference between Multiply + Reduction and Multiply (And Multiply + Add) is that in the graph, the slope of the performance with a larger global size remains more relatively constant. Furthermore when the sample size is too small, GPU parallel programming isn’t the ideal tool for the job. But with a large dataset, it is a good tool.

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

    GPU parallel programming truly is a data-focused method. With an adequate amount of data, GPU parallel computing is evident of a good tool.