Taylor Kirkpatrick

CS 475 Project 6

1. I ran these tests on the rabbit server (rabbit.engr.oregonstate.edu), at 3:00pm during the time I scheduled in advance (then deleted the reservation after I had finished). Rabbit is a two processor Xeon system with a total of 16 cores. Rabbit has 64 gigabytes of memory and 2 terabytes of storage. More importantly for this project, rabbit has an NVIDIA Titan Black GPU card with 15 streaming multiprocessors, 2880 CUDA cores, and 6 gigabytes of memory. The system also has a 31S1P Xeon Phi processor which boasts enormous multiprocessing capabilities but was not used for this assignment.

   It is worth noting that the rabbit server was under significant load during testing for these programs. Uptime run during a particularly noted slow time found an average load of around 5 processes, which is reasonably slow and may explain the number of outliers and messy data in this report. I at least can say that I was running my code at the scheduled time, but I believe the cause of the various outliers and data variance visible in my results are the cause of server load.
2. Table for Array Multiplication:

Number of Elements	Local Work Size	Global Work Size	Speed (GigaMults per Second)
1024	1	1024	0.014
2048	1	2048	0.047
4096	1	4096	0.079
8192	1	8192	0.142
16384	1	16384	0.288
32768	1	32768	0.302
65536	1	65536	0.88
131072	1	131072	0.036
262144	1	262144	0.452
524288	1	524288	0.035
1048576	1	1048576	1.866
2097152	1	2097152	4.437
4194304	1	4194304	0.676
8388608	1	8388608	1.17
1024	2	512	0.015
2048	2	1024	0.043
4096	2	2048	0.062
8192	2	4096	0.099
16384	2	8192	0.172
32768	2	16384	0.328
65536	2	32768	1.522
131072	2	65536	0.35
262144	2	131072	0.532
524288	2	262144	1.674
1048576	2	524288	2.814
2097152	2	1048576	1.213
4194304	2	2097152	5.478
8388608	2	4194304	1.38
1024	4	256	0.01
2048	4	512	0.029
4096	4	1024	0.072
8192	4	2048	0.11
16384	4	4096	0.118
32768	4	8192	0.516
65536	4	16384	0.819
131072	4	32768	0.147

262144	4	65536	0.171
524288	4	131072	1.231
1048576	4	262144	0.164
2097152	4	524288	0.404
4194304	4	1048576	2.076
8388608	4	2097152	4.024
1024	8	128	0.014
2048	8	256	0.028
4096	8	512	0.058
8192	8	1024	0.135
16384	8	2048	0.37
32768	8	4096	0.172
65536	8	8192	0.928
131072	8	16384	0.171
262144	8	32768	0.298
524288	8	65536	0.906
1048576	8	131072	1.572
2097152	8	262144	0.231
4194304	8	524288	8.447
8388608	8	1048576	11.892
1024	16	64	0.013
2048	16	128	0.04
4096	16	256	0.06
8192	16	512	0.094
16384	16	1024	0.359
32768	16	2048	0.595
65536	16	4096	1.298
131072	16	8192	0.132
262144	16	16384	0.752
524288	16	32768	1.395
1048576	16	65536	1.092
2097152	16	131072	6.712
4194304	16	262144	6.894
8388608	16	524288	13.936
1024	32	32	0.015
2048	32	64	0.044
4096	32	128	0.062
8192	32	256	0.104
16384	32	512	0.179
32768	32	1024	0.588

65536	32	2048	0.867
131072	32	4096	0.065
262144	32	8192	0.288
524288	32	16384	0.1
1048576	32	32768	2.252
2097152	32	65536	0.969
4194304	32	131072	5.576
8388608	32	262144	1.578
1024	64	16	0.013
2048	64	32	0.029
4096	64	64	0.052
8192	64	128	0.122
16384	64	256	0.219
32768	64	512	0.69
65536	64	1024	1.054
131072	64	2048	0.269
262144	64	4096	0.701
524288	64	8192	1.239
1048576	64	16384	2.021
2097152	64	32768	3.628
4194304	64	65536	10.048
8388608	64	131072	18.106
1024	128	8	0.009
2048	128	16	0.033
4096	128	32	0.039
8192	128	64	0.146
16384	128	128	0.197
32768	128	256	0.541
65536	128	512	1.647
131072	128	1024	0.164
262144	128	2048	0.456
524288	128	4096	1.284
1048576	128	8192	0.841
2097152	128	16384	5.11
4194304	128	32768	11.024
8388608	128	65536	9.187
1024	256	4	0.017
2048	256	8	0.038
4096	256	16	0.059
8192	256	32	0.086

16384	256	64	0.2
32768	256	128	0.384
65536	256	256	1.057
131072	256	512	0.141
262144	256	1024	0.051
524288	256	2048	0.97
1048576	256	4096	2.385
2097152	256	8192	0.311
4194304	256	16384	10.161
8388608	256	32768	2.474
1024	512	2	0.016
2048	512	4	0.049
4096	512	8	0.082
8192	512	16	0.094
16384	512	32	0.313
32768	512	64	0.619
65536	512	128	0.821
131072	512	256	0.371
262144	512	512	0.387
524288	512	1024	0.7
1048576	512	2048	1.982
2097152	512	4096	5.509
4194304	512	8192	0.014
8388608	512	16384	15.217
1024	1024	1	0.016
2048	1024	2	0.029
4096	1024	4	0.097
8192	1024	8	0.189
16384	1024	16	0.29
32768	1024	32	0.572
65536	1024	64	1.467
131072	1024	128	0.73
262144	1024	256	0.781
524288	1024	512	1.825
1048576	1024	1024	2.474
2097152	1024	2048	4.499
4194304	1024	4096	6.287
8388608	1024	8192	22.368

Table for Array Multiplication-Summation:

Number of Elements	Local Work Size	Global Work Size	Speed (GigaMults/Sums per Second)
1024	1	1024	0.014
2048	1	2048	0.023
4096	1	4096	0.05
8192	1	8192	0.154
16384	1	16384	0.298
32768	1	32768	0.593
65536	1	65536	1.576
131072	1	131072	1.671
262144	1	262144	0.214
524288	1	524288	0.947
1048576	1	1048576	0.349
2097152	1	2097152	0.709
4194304	1	4194304	1.066
8388608	1	8388608	0.705
1024	2	512	0.014
2048	2	1024	0.026
4096	2	2048	0.074
8192	2	4096	0.08
16384	2	8192	0.418
32768	2	16384	0.512
65536	2	32768	1.226
131072	2	65536	2.68
262144	2	131072	0.544
524288	2	262144	0.58
1048576	2	524288	0.147
2097152	2	1048576	0.441
4194304	2	2097152	1.787
8388608	2	4194304	0.459
1024	4	256	0.012
2048	4	512	0.025
4096	4	1024	0.057
8192	4	2048	0.098
16384	4	4096	0.198
32768	4	8192	0.54

65536	4	16384	0.487
131072	4	32768	1.783
262144	4	65536	0.034
524288	4	131072	0.414
1048576	4	262144	0.569
2097152	4	524288	0.165
4194304	4	1048576	3.72
8388608	4	2097152	0.785
1024	8	128	0.02
2048	8	256	0.03
4096	8	512	0.035
8192	8	1024	0.113
16384	8	2048	0.223
32768	8	4096	0.45
65536	8	8192	0.644
131072	8	16384	1.628
262144	8	32768	0.25
524288	8	65536	0.756
1048576	8	131072	2.063
2097152	8	262144	0.699
4194304	8	524288	1.373
8388608	8	1048576	1.246
1024	16	64	0.012
2048	16	128	0.048
4096	16	256	0.038
8192	16	512	0.138
16384	16	1024	0.277
32768	16	2048	0.753
65536	16	4096	0.864
131072	16	8192	2.166
262144	16	16384	0.517
524288	16	32768	1.158
1048576	16	65536	0.114
2097152	16	131072	5.912
4194304	16	262144	3.143
8388608	16	524288	14.358
1024	32	32	0.013
2048	32	64	0.029
4096	32	128	0.046
8192	32	256	0.109

16384	32	512	0.187
32768	32	1024	0.433
65536	32	2048	1.139
131072	32	4096	1.718
262144	32	8192	0.023
524288	32	16384	0.907
1048576	32	32768	2.281
2097152	32	65536	3.317
4194304	32	131072	9.613
8388608	32	262144	13.235
1024	64	16	0.012
2048	64	32	0.027
4096	64	64	0.057
8192	64	128	0.094
16384	64	256	0.301
32768	64	512	0.311
65536	64	1024	0.661
131072	64	2048	1.572
262144	64	4096	0.222
524288	64	8192	1.148
1048576	64	16384	2.947
2097152	64	32768	2.259
4194304	64	65536	0.413
8388608	64	131072	9.247
1024	128	8	0.012
2048	128	16	0.034
4096	128	32	0.053
8192	128	64	0.122
16384	128	128	0.213
32768	128	256	0.507
65536	128	512	0.802
131072	128	1024	1.63
262144	128	2048	0.478
524288	128	4096	0.936
1048576	128	8192	2.453
2097152	128	16384	4.063
4194304	128	32768	4.609
8388608	128	65536	22.071
1024	256	4	0.023
2048	256	8	0.028

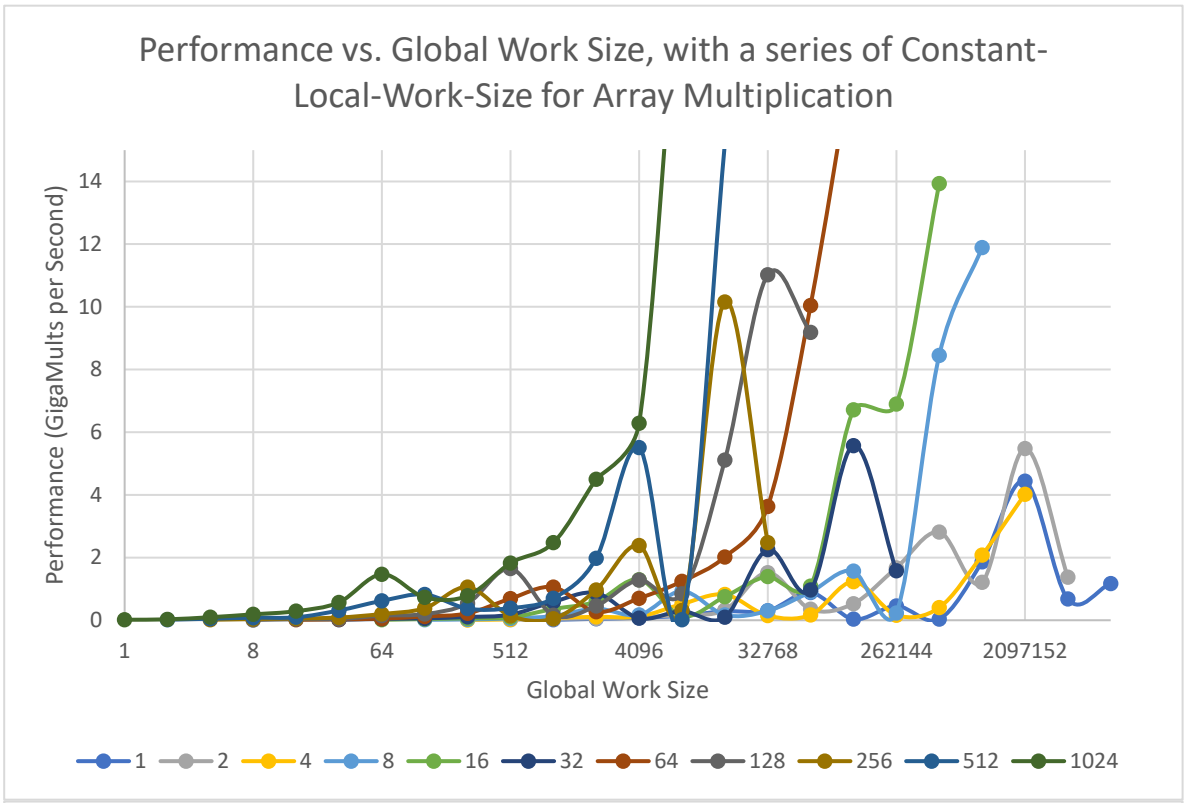
4096	256	16	0.046
8192	256	32	0.108
16384	256	64	0.242
32768	256	128	0.396
65536	256	256	1.097
131072	256	512	1.74
262144	256	1024	0.751
524288	256	2048	1.288
1048576	256	4096	1.424
2097152	256	8192	3.599
4194304	256	16384	8.397
8388608	256	32768	0.984
1024	512	2	0.019
2048	512	4	0.031
4096	512	8	0.075
8192	512	16	0.162
16384	512	32	0.337
32768	512	64	0.516
65536	512	128	1.076
131072	512	256	0.148
262144	512	512	0.477
524288	512	1024	0.4
1048576	512	2048	2.599
2097152	512	4096	1.43
4194304	512	8192	7.297
8388608	512	16384	0.651
1024	1024	1	0.01
2048	1024	2	0.043
4096	1024	4	0.041
8192	1024	8	0.167
16384	1024	16	0.193
32768	1024	32	0.378
65536	1024	64	0.795
131072	1024	128	1.482
262144	1024	256	0.598
524288	1024	512	0.984
1048576	1024	1024	1.958
2097152	1024	2048	4.747
4194304	1024	4096	0.372
8388608	1024	8192	19.003

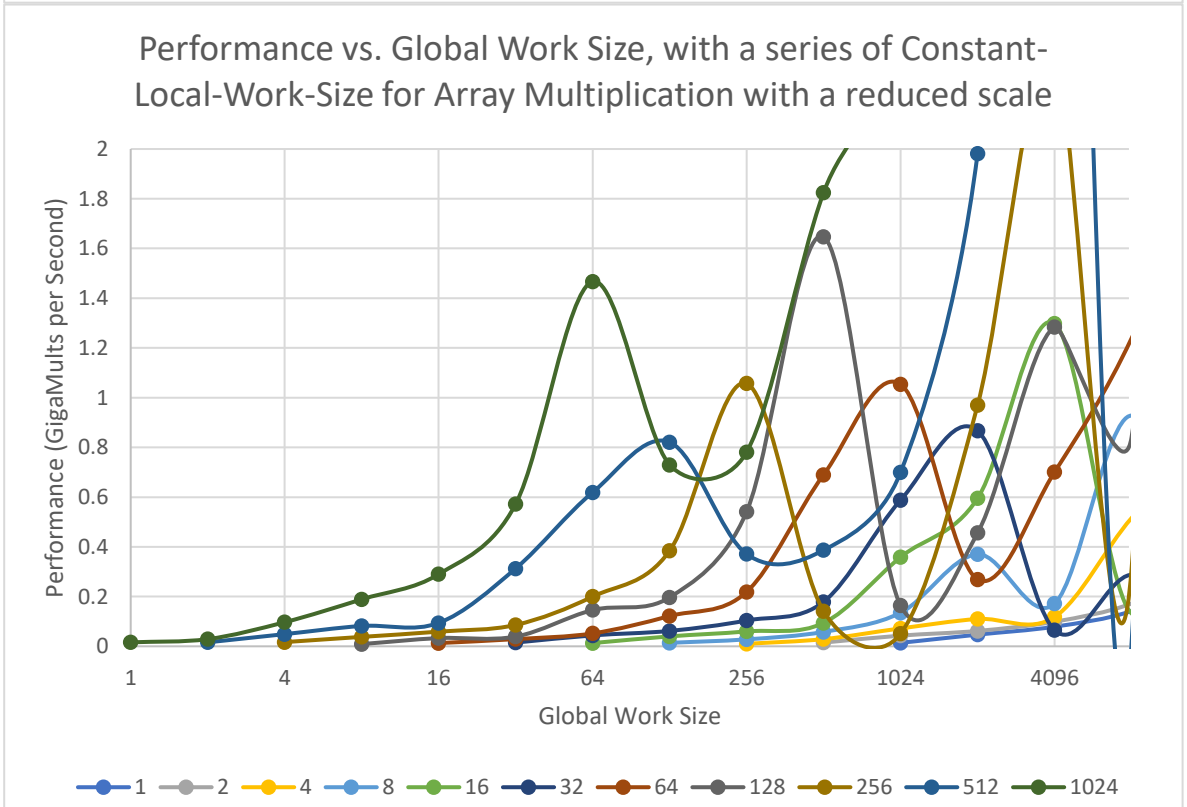
Table for Array Multiplication-Reduction:

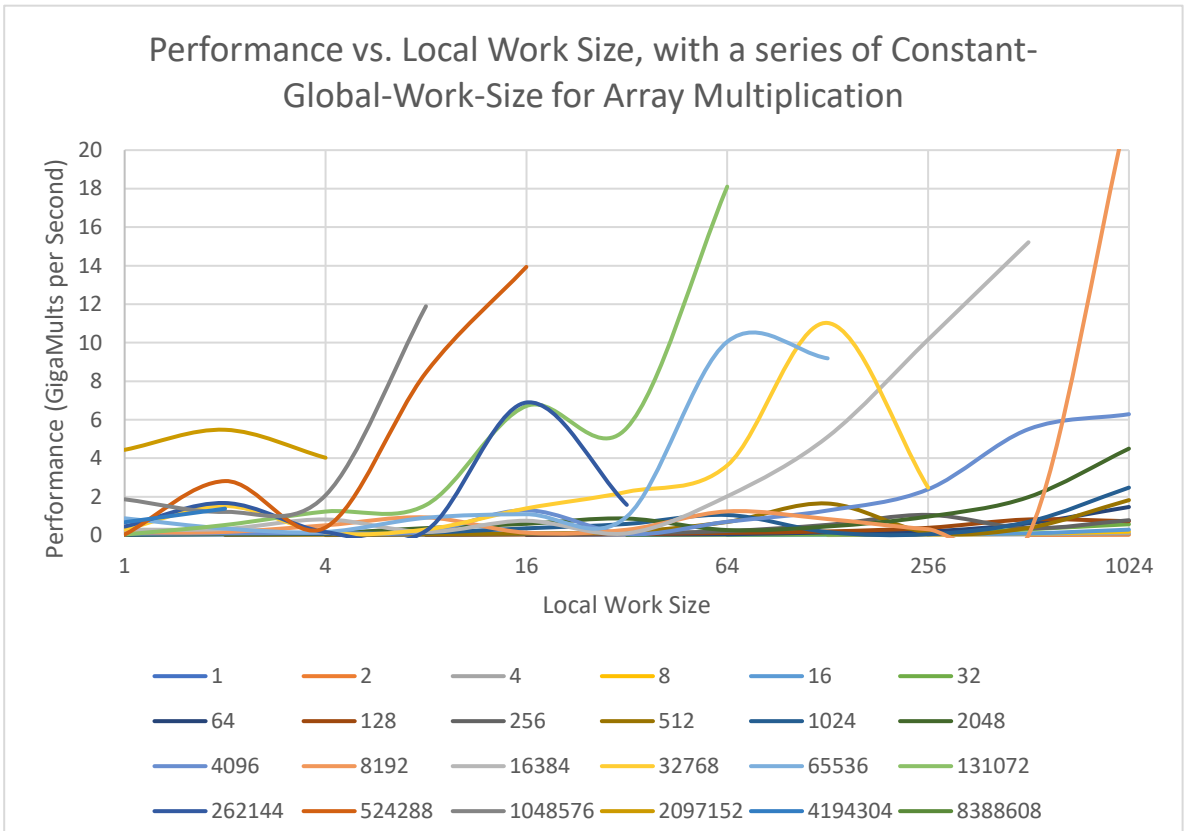
Number of Elements	Local Work Size	Global Work Size	Speed (GigaMults/Reductions per Second)
1024	32	32	0.001
2048	32	64	0.001
4096	32	128	0.006
8192	32	256	0.011
16384	32	512	0.017
32768	32	1024	0.01
65536	32	2048	0.032
131072	32	4096	0.062
262144	32	8192	0.055
524288	32	16384	0.578
1048576	32	32768	1.627
2097152	32	65536	0.36
4194304	32	131072	0.829
8388608	32	262144	16.607

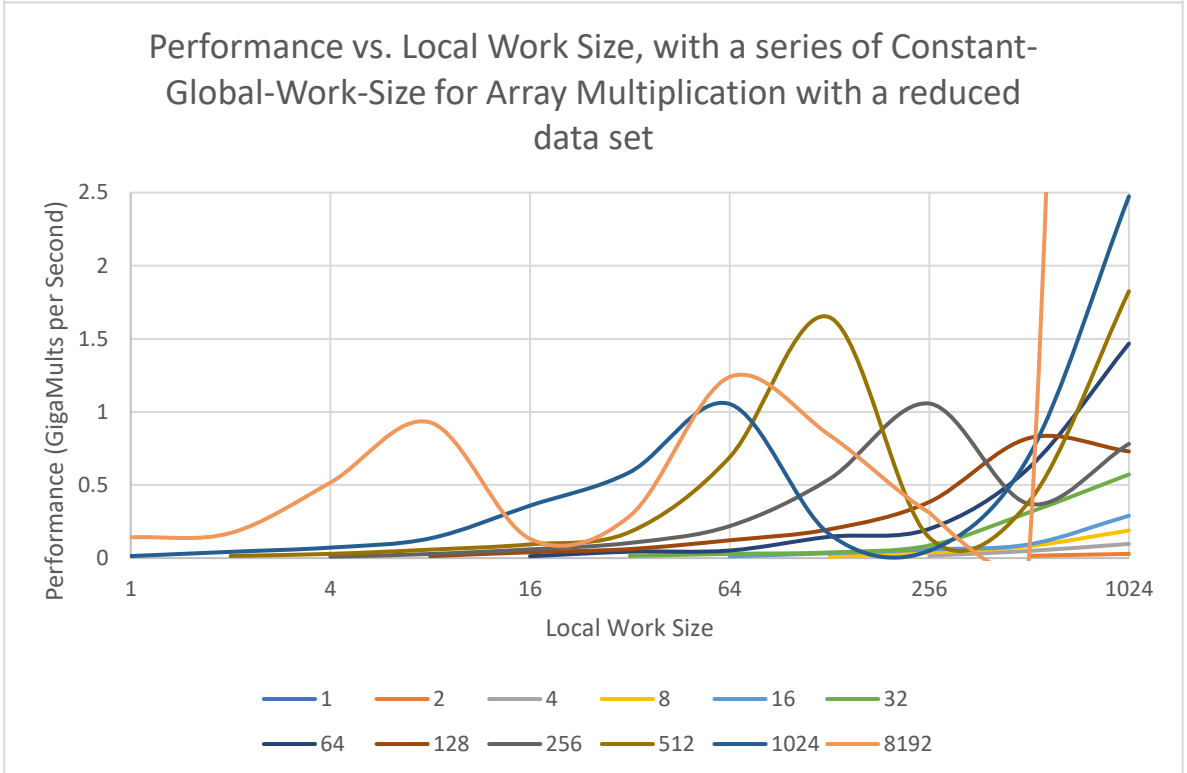
Graphs:

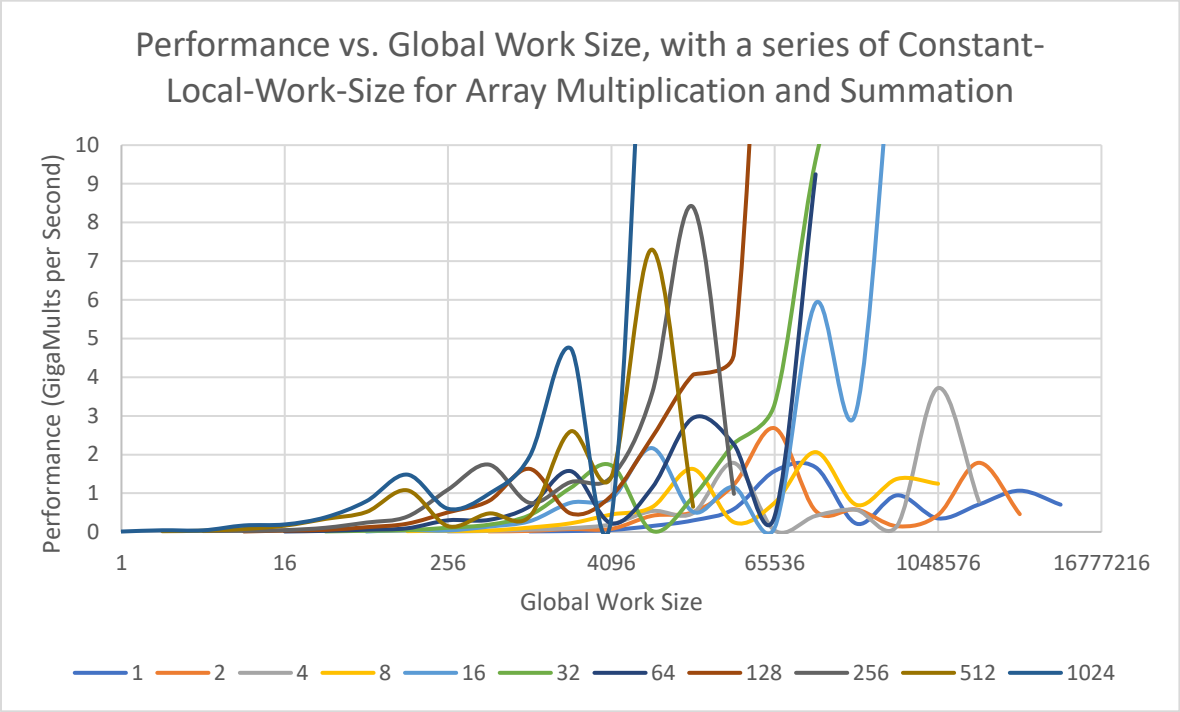
Due to the size of the data sets, the outliers, the exponential increase of the work sizes (which on second thought was not the wisest decision, though my second attempt didn't really show patterns that were any more meaningful than these), and the various outliers, many of the initial graphs do not look very good or correctly representative of the concept they were meant to show. Thus, I have included many more graphs than was required. The graphs as requested are included and may be followed by graphs with a reduced data set or scale to more properly show trends or particularly interesting patterns. Also included in the .xlsx file are line graphs that are inaccurate for making judgments but do show a reasonably accurate trend of speed up as the arrays grow. These extra graphs are marked accordingly in their titles.

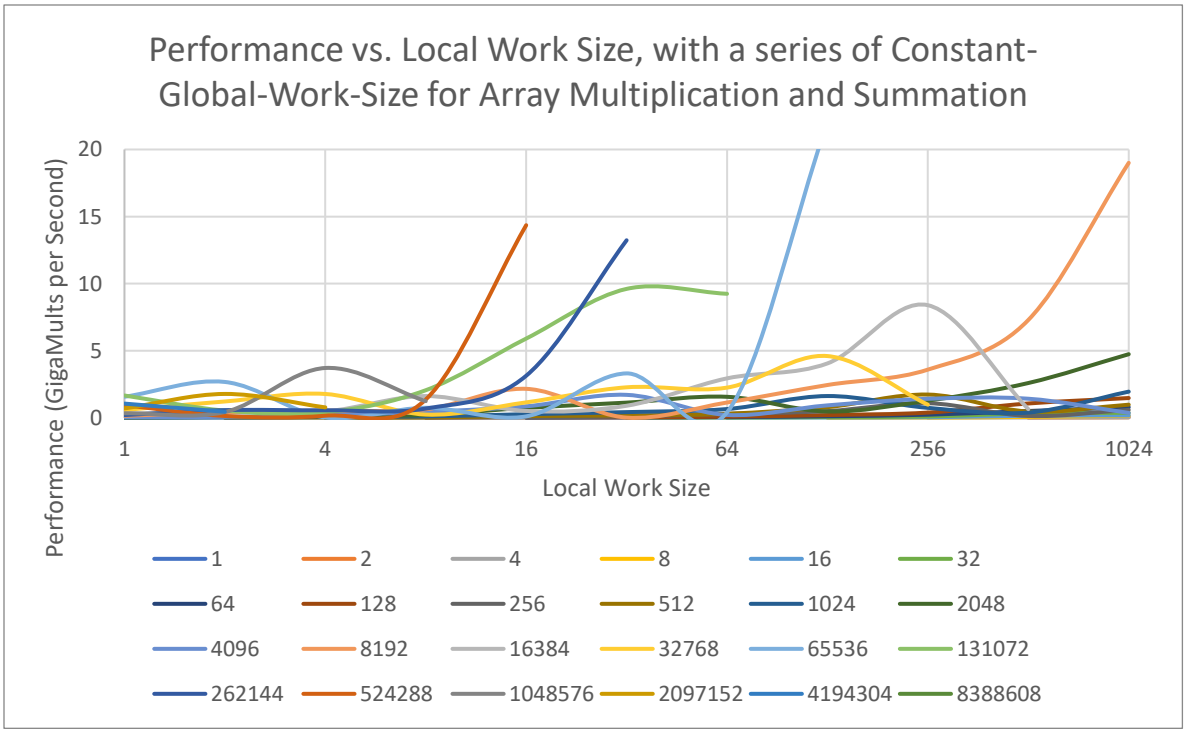

Performance vs. Global Work Size, with a series of Constant-Local-Work-Size for Array Multiplication
Performance (GigaMults per Second)
0
2
4
6
8
10
12
14
1
8
64
512
4096
32768
262144
2097152
Global Work Size
1
2
4
8
16
32
64
128
256
512
1024

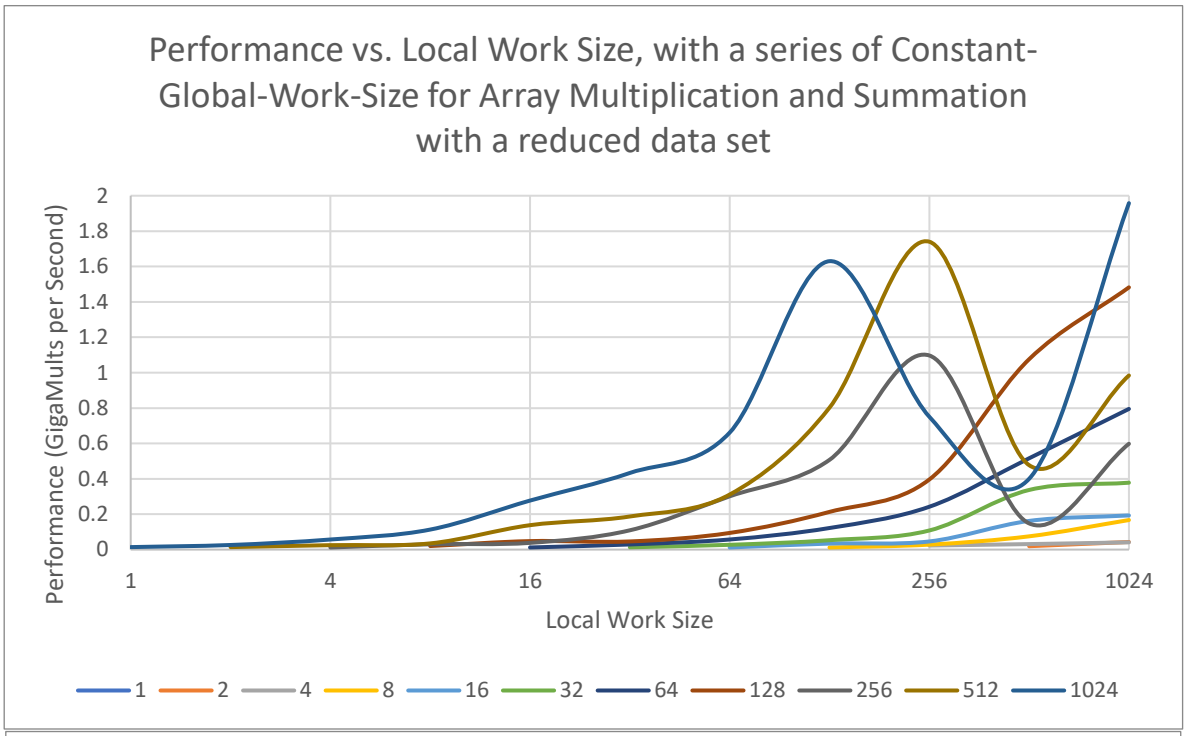

Performance vs. Global Work Size, with a series of Constant-Local-Work-Size for Array Multiplication with a reduced scale
Performance (GigaMults per Second)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
1
4
16
64
256
1024
4096
Global Work Size
1
2
4
8
16
32
64
128
256
512
1024

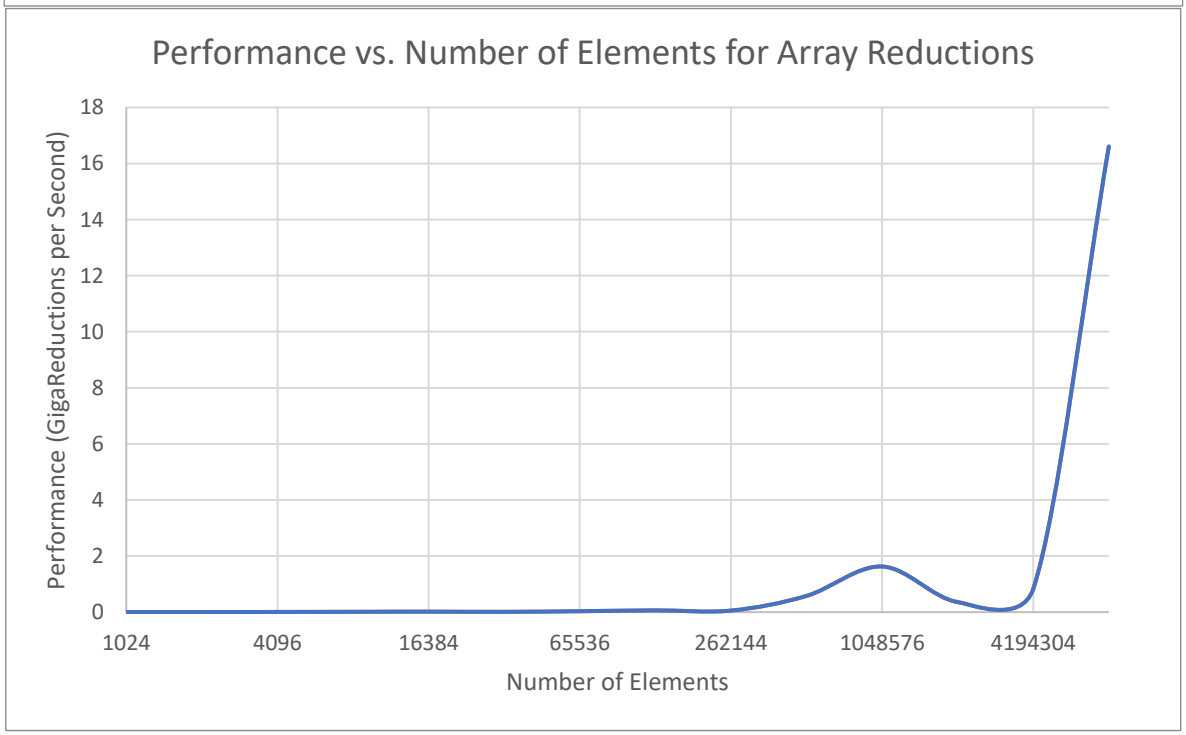

Performance vs. Local Work Size, with a series of Constant-Global-Work-Size for Array Multiplication
Performance (GigaMults per Second)
20
18
16
14
12
10
8
6
4
2
0
1
4
16
64
256
1024
Local Work Size
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608

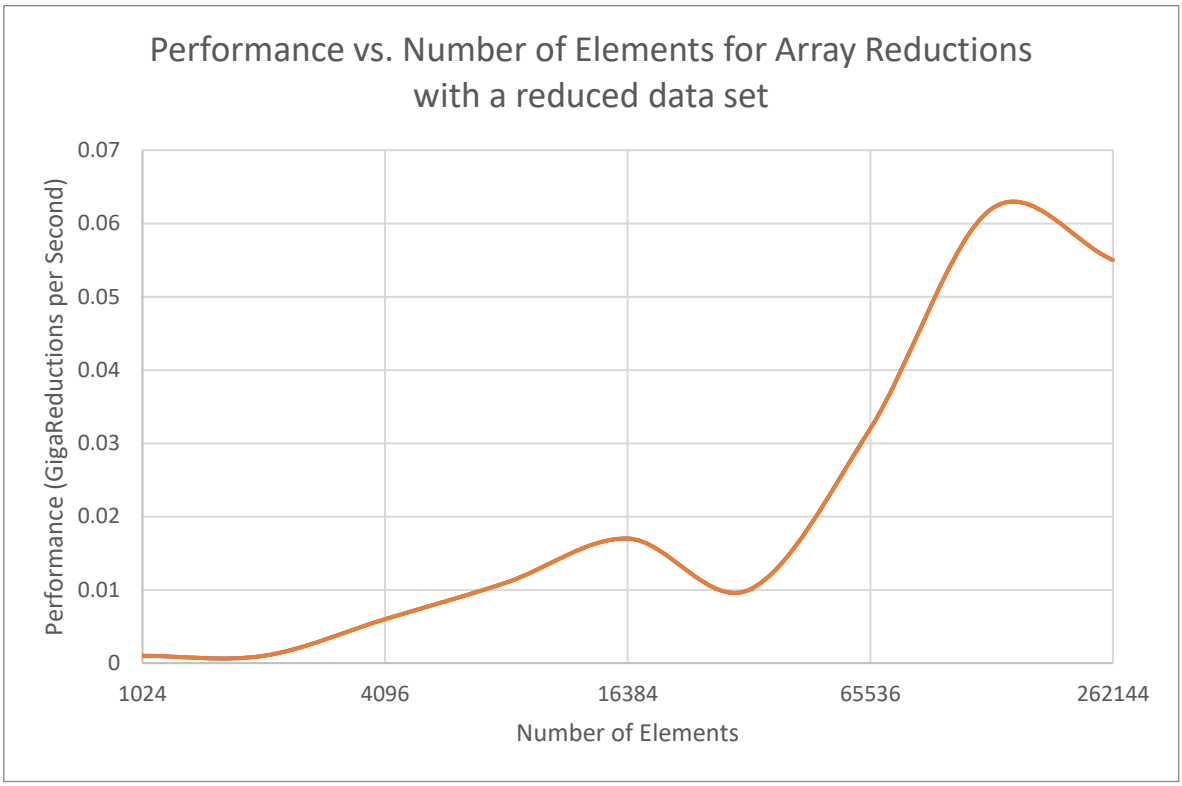

Performance vs. Local Work Size, with a series of Constant-Global-Work-Size for Array Multiplication with a reduced data set
Performance (GigaMults per Second)
2.5
2
1.5
1
0.5
0
1
4
16
64
256
1024
Local Work Size
1
2
4
8
16
32
64
128
256
512
1024
8192


Performance vs. Global Work Size, with a series of Constant-Local-Work-Size for Array Multiplication and Summation
Performance (GigaMults per Second)
10
9
8
7
6
5
4
3
2
1
0
1
16
256
4096
65536
1048576
16777216
Global Work Size
1
2
4
8
16
32
64
128
256
512
1024


Performance vs. Local Work Size, with a series of Constant-Global-Work-Size for Array Multiplication and Summation
Performance (GigaMults per Second)
20
15
10
5
0
1
4
16
64
256
1024
Local Work Size
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608

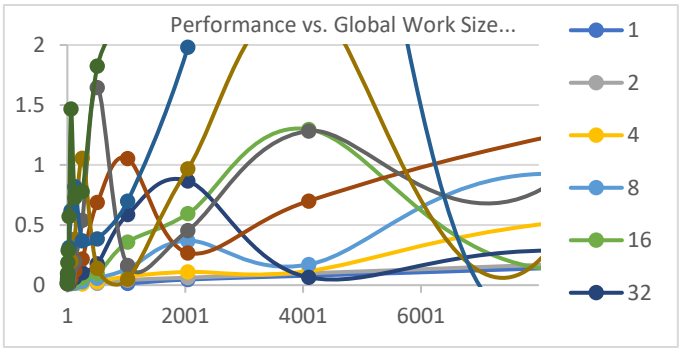

Performance vs. Local Work Size, with a series of Constant-Global-Work-Size for Array Multiplication and Summation with a reduced data set
Performance (GigaMults per Second)
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1
4
16
64
256
1024
Local Work Size
1
2
4
8
16
32
64
128
256
512
1024


Performance vs. Number of Elements for Array Reductions
Performance (GigaReductions per Second)
18
16
14
12
10
8
6
4
2
0
1024
4096
16384
65536
262144
1048576
4194304
Number of Elements



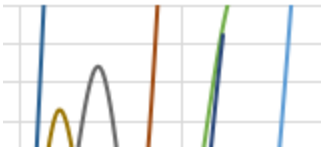
3. Patterns in Part 1 (Multiply and Multiply-Sum)

To start with the Performance vs Global Work Size Array Multiplication, it can be seen that most and probably all curves begin with a steady increase in performance as the global work size increases, then drop suddenly, then resume their previous slope as it was before the drop. The lines that extend to the larger values seem to rise even higher than is reasonable to display on the graph, but this is almost certainly because of the improper decision to include an exponentially increasing work size rather than a specific emergent quality from this project. The pattern likely intended to emerge from this project is a tad more easily displayed in the reduced data set version of this graph, displayed to the left without axis labels to save room. Particularly in the light blue line which represents a local work group size of 8, one can see this trend as the gradual increase is interrupted by a drop when the global work size is 4096, followed by the prior trend continuing. While some lines appear to be establishing a horizontal asymptote, one can see on the full-sized graph that no line appears to be establishing a max for speedup.



As for the Performance vs Local Work Size Array Multiplication, the graphs are in as much disarray as the others. These curves seem to be continuing the pattern of ups and downs, but otherwise appear like they would continue growing faster if given a larger data set. It is worth noting that the last datapoint for the orange line, a global work size of 8, does not even fit the messy patterns shown by the other lines, and that the very dramatic rise for that point is quite likely an outlier.

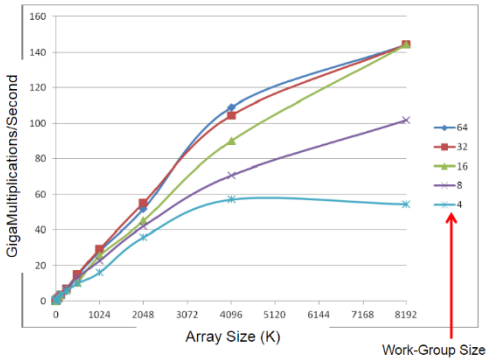
I will speak to both of the Multiplication-Summation graphs at the same time, as they are mostly similar. These graphs are a tad more consistent, showing the constant up and down pattern for most lines starting earliest at a work group size of 128 for both local and global. It is worth noting that the five lines in the Performance vs Global Work Size graph (as pictured to the right) that continue to rise without coming back down are the lines that represent the local work sizes of 1024, 128, 64, 32, and 16. The interesting thing about this list is twofold: first, that these are not consecutive. 512 and 256 are both unable to reach the same height, but are pictured in the same image as the yellow and grey line respecitvely. The other interesting thing about this list is that those are the same five points on the Performance vs Local graph that do the same, rise without coming back down.



It is also worth noting that in general, performance seems to maximize in the lines that have a closer global and local group.

4. Why are these patterns here?

Initially I thought that the patterns I found in most of my examples were simply incorrect due to what I thought was an improper choice of element increments. I was expecting something more like the graph the professor provides in the week 6 slides, pictured to the left. Because of this, I attempted a second test for the Array Multiplication part of the assignment using a constant element increase of 524288. However, this ended up reporting data that was even less helpful. The table and graph of which are included in the .xlsx file included and at the bottom of this document. Thus, I came to the conclusion that my data was not, in fact incorrect aside from the clear outlier points, but rather that my data sizes were not sculpted in such a way to show the same result as the graph produced by Professor Bailey. In addition, while the patterns of increasing and decreasing can't be caused by load unless it was very methodical, the various pattern breakers like those detailed in the Multiplication-Summation section are likely to be outliers. While this volume of outliers is rare, the intense server load from many people rushing to get their code run on rabbit is sure to cause many more outliers than usual. The large amount of variation in the results is also likely the cause of this.



As for why the pattern of up and down, I believe this is caused by the efficiency of splitting jobs up between the processing elements, or work-items as OpenCL calls them. As OpenCL works by giving

work-items jobs in a queue, the average performance oscillates up and down based on how efficient the queue is at handing out these jobs to the various work-groups and work-items.

And as for why the fastest runs seem to be in the middle of the data, this is again a result of the efficiency of different work-items and groups, but this time it is caused by what they do when finished. When the work-items are divided in such a way as to do very small tasks (that is, where the local work size is high), they quickly finish their tasks and must wait to be given new tasks by their group and by the GPU, not to mention the overhead that is likely caused by data transfer and setting the work-items to work with new data. Meanwhile, very work-items with very large tasks is hardly doing it any different than doing it without parallelization at all.

5. What does this mean for GPU parallelization?

This means that properly splitting up the tasks one hopes to parallelize into more manageable chunks is paramount. This has, however, always been necessary when properly parallelizing a program. However, thinking more about the size of the data sets in relation to how they will be broken up by the GPU is important. In our programs, we needed to set both the global and local size, meaning the consideration of things like register use and data size are for the developer to figure out before maximizing the performance of their code. The cause for the oscillating speed increase is likely the most important applicable results to draw from this data. Identifying why this is and how to avoid the low speed occurrences would be of vital interest to a developer looking to maximize their GPU parallelization.

6. Patterns in Part 1 (Multiply and Multiply-Sum) w/ explanation

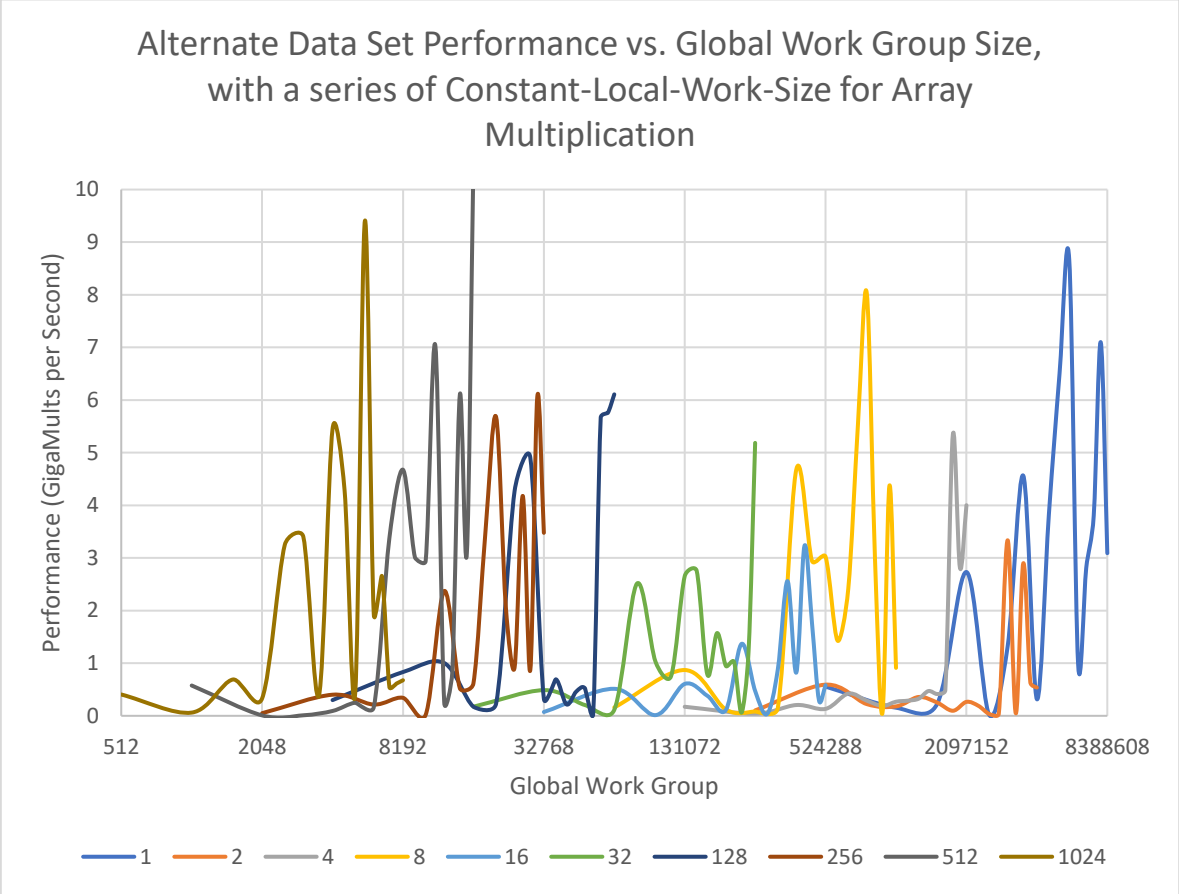
There are fewer lines on the graph here meaning the patterns are much easier to spot. The bulge at point 1048576 is prominent and zooming in to the smaller part of the graph dwarfed by the jump to 8388608, we can see two other bumps in the line, and odd jumps in the data. While my first instinct was to believe that these were caused by the OpenCL barrier statement used in this part of the project, after reviewing the data, these bumps appear at the same points as bumps appear in part one's data as well. This further confirms that these bumps are not just outliers but are indeed changes in efficiency based on how the OpenCL code runs as described above in question 4.

7. What does this mean for GPU parallelization?

Most of what goes here is already covered in section 5 above, but I will note that while the barriers in the reduction OpenCL code had a negligible impact in this case, that does not mean they are not a dangerous thing to use if put in the wrong place or told to wait for incorrect conditions. While the barriers did not cause any significant changes in the results for varying data sets, the speed of the reduction did not climb nearly as fast as did the speed of the programs in part one. This is likely either the cause of the barriers or the need to send data back into global memory, causing a flat slowdown for every work-item.

8. Alternative data set

The graph and table following were an attempt to correct my data, which I initially assumed was wrong. However, after seeing results from the Array Multiplication program, I chose to continue with the exponentially increasing increments, as I was running out of time to write this report and because the data seemed to be even less helpful than the data presented elsewhere in this report. While this data is surely also full of outliers as it was run at a time that was likely even busier than before, it also shows an interesting pattern of the program being seemingly more efficient at the higher and lower data set sizes.



Number of Elements	Local Work Size	Global Work Size	Speed (GigaMults per Second)
524288	1	524288	0.555
1048576	1	1048576	0.154
1572864	1	1572864	0.227
2097152	1	2097152	2.732
2621440	1	2621440	0.02

3145728	1	3145728	1.32
3670016	1	3670016	4.562
4194304	1	4194304	0.316
4718592	1	4718592	3.818
5242880	1	5242880	6.504
5767168	1	5767168	8.672
6291456	1	6291456	0.998
6815744	1	6815744	2.807
7340032	1	7340032	3.802
7864320	1	7864320	7.098
8388608	1	8388608	3.09
524288	2	262144	0.103
1048576	2	524288	0.592
1572864	2	786432	0.223
2097152	2	1048576	0.179
2621440	2	1310720	0.358
3145728	2	1572864	0.251
3670016	2	1835008	0.096
4194304	2	2097152	0.265
4718592	2	2359296	0.19
5242880	2	2621440	0.037
5767168	2	2883584	0.04
6291456	2	3145728	3.336
6815744	2	3407872	0.048
7340032	2	3670016	2.898
7864320	2	3932160	0.638
8388608	2	4194304	0.537
524288	4	131072	0.172
1048576	4	262144	0.042
1572864	4	393216	0.205
2097152	4	524288	0.132
2621440	4	655360	0.422
3145728	4	786432	0.288
3670016	4	917504	0.189
4194304	4	1048576	0.268
4718592	4	1179648	0.288
5242880	4	1310720	0.325
5767168	4	1441792	0.47
6291456	4	1572864	0.415
6815744	4	1703936	0.488

7340032	4	1835008	5.347
7864320	4	1966080	2.818
8388608	4	2097152	4.002
524288	8	65536	0.159
1048576	8	131072	0.87
1572864	8	196608	0.12
2097152	8	262144	0.082
2621440	8	327680	0.149
3145728	8	393216	4.683
3670016	8	458752	2.944
4194304	8	524288	3.022
4718592	8	589824	1.428
5242880	8	655360	2.412
5767168	8	720896	5.535
6291456	8	786432	8.031
6815744	8	851968	3.029
7340032	8	917504	0.052
7864320	8	983040	4.375
8388608	8	1048576	0.911
524288	16	32768	0.073
1048576	16	65536	0.513
1572864	16	98304	0.013
2097152	16	131072	0.606
2621440	16	163840	0.38
3145728	16	196608	0.113
3670016	16	229376	1.367
4194304	16	262144	0.473
4718592	16	294912	0.035
5242880	16	327680	0.881
5767168	16	360448	2.56
6291456	16	393216	0.823
6815744	16	425984	3.23
7340032	16	458752	1.761
7864320	16	491520	0.29
8388608	16	524288	0.583
524288	32	16384	0.173
1048576	32	32768	0.488
1572864	32	49152	0.204
2097152	32	65536	0.114
2621440	32	81920	2.512

3145728	32	98304	1.029
3670016	32	114688	0.757
4194304	32	131072	2.653
4718592	32	147456	2.755
5242880	32	163840	0.783
5767168	32	180224	1.572
6291456	32	196608	0.947
6815744	32	212992	1.024
7340032	32	229376	0.06
7864320	32	245760	1.231
8388608	32	262144	5.186
524288	128	4096	0.299
1048576	128	8192	0.832
1572864	128	12288	1.003
2097152	128	16384	0.174
2621440	128	20480	0.232
3145728	128	24576	4.275
3670016	128	28672	4.907
4194304	128	32768	0.326
4718592	128	36864	0.694
5242880	128	40960	0.216
5767168	128	45056	0.46
6291456	128	49152	0.52
6815744	128	53248	0.045
7340032	128	57344	5.666
7864320	128	61440	5.753
8388608	128	65536	6.108
524288	256	2048	0.05
1048576	256	4096	0.401
1572864	256	6144	0.214
2097152	256	8192	0.342
2621440	256	10240	0.018
3145728	256	12288	2.371
3670016	256	14336	0.521
4194304	256	16384	0.615
4718592	256	18432	3.479
5242880	256	20480	5.678
5767168	256	22528	2.079
6291456	256	24576	0.921
6815744	256	26624	4.179

7340032	256	28672	0.869
7864320	256	30720	6.063
8388608	256	32768	3.477
524288	512	1024	0.576
1048576	512	2048	0.008
1572864	512	3072	0.01
2097152	512	4096	0.092
2621440	512	5120	0.248
3145728	512	6144	0.162
3670016	512	7168	3.381
4194304	512	8192	4.676
4718592	512	9216	3.01
5242880	512	10240	2.941
5767168	512	11264	7.031
6291456	512	12288	0.266
6815744	512	13312	0.832
7340032	512	14336	6.102
7864320	512	15360	3.09
8388608	512	16384	10.52
524288	1024	512	0.405
1048576	1024	1024	0.061
1572864	1024	1536	0.69
2097152	1024	2048	0.331
2621440	1024	2560	3.25
3145728	1024	3072	3.402
3670016	1024	3584	0.389
4194304	1024	4096	5.451
4718592	1024	4608	4.296
5242880	1024	5120	0.437
5767168	1024	5632	9.406
6291456	1024	6144	1.941
6815744	1024	6656	2.655
7340032	1024	7168	0.545
7864320	1024	7680	0.612
8388608	1024	8192	0.676