1 Overview

For this project, I did some of my favorite work of all term. I created a program that will take in a formatted string denoting which type of operation to use in a kernel. For example, you can run:

```
./main a=b*c 1024 128
```

and the program will generate the OpenCL code for a kernel and create host and device memory accordingly to handle that operation.

You can also specify reduction operations like:

```
./main a:=b*c 1024 128
```

This is extremely useful to specify an arbitrary kernel at runtime! It would be useful if you are doing a complex operation with a number of adds and multiplies such as:

```
./main a=b*c+d*e+f*g 1024 128
```

Anyway, I am super proud of this program and I hope you like it, too!

2 Machine Info

I submitted the code I wrote to the DGX server as a batch job. The DGX server ran the program with:

```
Platform #0:
        Name    = 'NVIDIA CUDA'
        Vendor  = 'NVIDIA Corporation'
        Version = 'OpenCL 1.2 CUDA 10.1.351'
        Profile = 'FULL_PROFILE'
        Number of Devices = 1
        Device #0:
                Type = 0x0004 = CL_DEVICE_TYPE_GPU
                Device Vendor ID = 0x10de (NVIDIA)
                Device Maximum Compute Units = 80
                Device Maximum Work Item Dimensions = 3
                Device Maximum Work Item Sizes = 1024 x 1024 x 64
                Device Maximum Work Group Size = 1024
                Device Maximum Clock Frequency = 1530 MHz
```

3 Patterns Seen and Discussion: Multiply and Multiply-Add

As a general trend, the performance increased as the global data size was doubled. There are some outliers, though. Figure 1 shows the GigaMults/sec (performance) vs. the `Global Size` of that run. Figure 3 plots the same parameters but for a `Multiply then Add` run. Additionally, we also see the performance really increase as the `Local Size` is increased. We see the same behavior when we run a `Multiply, Add` test, as well. However, the `Multiply then Add` runs are about 15 GigaOperations slower than just `Multiply` at peak performance. Another thing to note is that in 2, we start to see the performance level off after we hit a `Local Size` of 128. At this point, there is probably no benefit to packing more work-items into a work-group.

Why do the patterns seem this way?
This is more or less the behavior that I was expecting from `Project 5`. We see an increase in performance when we throw more and more numbers (increasing `Global Size`) at the GPU and see an even bigger increase when we increase the number of work-items per group.

Performance Difference Between Multiply and Multiply-Add?
There is some performance difference between `Multiply` and `Multiply−Add`, but not too bad, actually. If you think about it, we are adding in a whole new operation and get 75 percent of the performance. This is due to the Fused Multiply-Add instructions, or FMA for short. The NVIDIA OpenCL Best Practices Guide says to `group add and mul instructions into a single FMAD instruction whenever possible` and that it can `lead to large performance gains.`

What Does That Mean For Proper Use of GPU Parallel Computing?
I think the most important thing that I learned from this is to make sure you research different ways to optimize your code. For example, if you never knew to enable the `−cl−mad−enable` build option, you would end up losing out on potential performance.

4 Patterns Seen and Discussion: Reduction Multiply

The trend of the `Reduction Multiply` kernel follow the same pattern as the previous section, as the performance increased as the global data size was doubled. Figure **??** shows the GigaReductionMults/sec (performance) vs. the `Global Size` of that run. The performance really increases once we start to increase the `Local Size` is increased, which is increasing the amount of work-items that are packed into a work-group.

Why do the patterns seem this way?
This pattern makes sense, because as we pass more and more data (`Global Size`) into the

kernels, we will see the operations per seconds increase, since the GPU has more operations to do. We see a performance increase as the work size is increased because the GPU is amazing at handling these types of operations and can just eat it up. Additionally, we don't pass the work size max listed in the device info (6 x 1024 x 1024), so we shouldn't see any performance dips until then.

What Does That Mean For Proper Use of GPU Parallel Computing?

We also see a huge boost in performance here. I'm not sure how much optimization is done behind the scenes, because this `Reduction Multiply` kernel actually runs more per second than just a plain `Multiply` kernel, even though there is more technical operations handled behind the scenes. I *think* that this performance increase is due to the fact that we are performing less writes into global memory (which is costly) and putting them in local memory (which is much faster). In the end, we really only make `Local Size` number of writes to global memory, which is orders of magnitude less than the amount of writes we use when doing `Multiply` or `Multiply−Add` (we need to write as many times as `Global Size`). This means that when using a GPU, always keep track of certain penalties you might accrue if you use global memory when there could be an algorithm you use that benefits from the use of local memory instead.

5 Results: Graphs

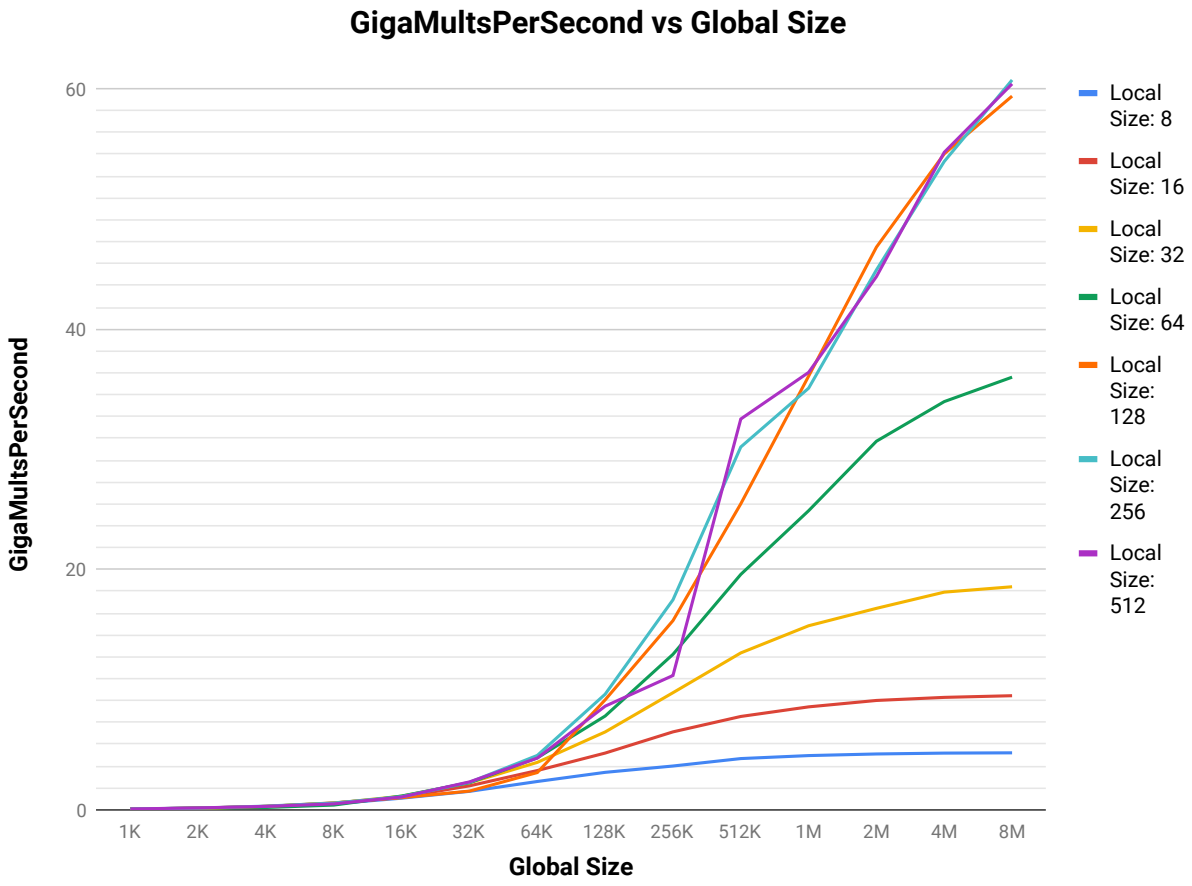

Figure 1: Multiply Performance vs. Global Size with multiple curves of Local Size.

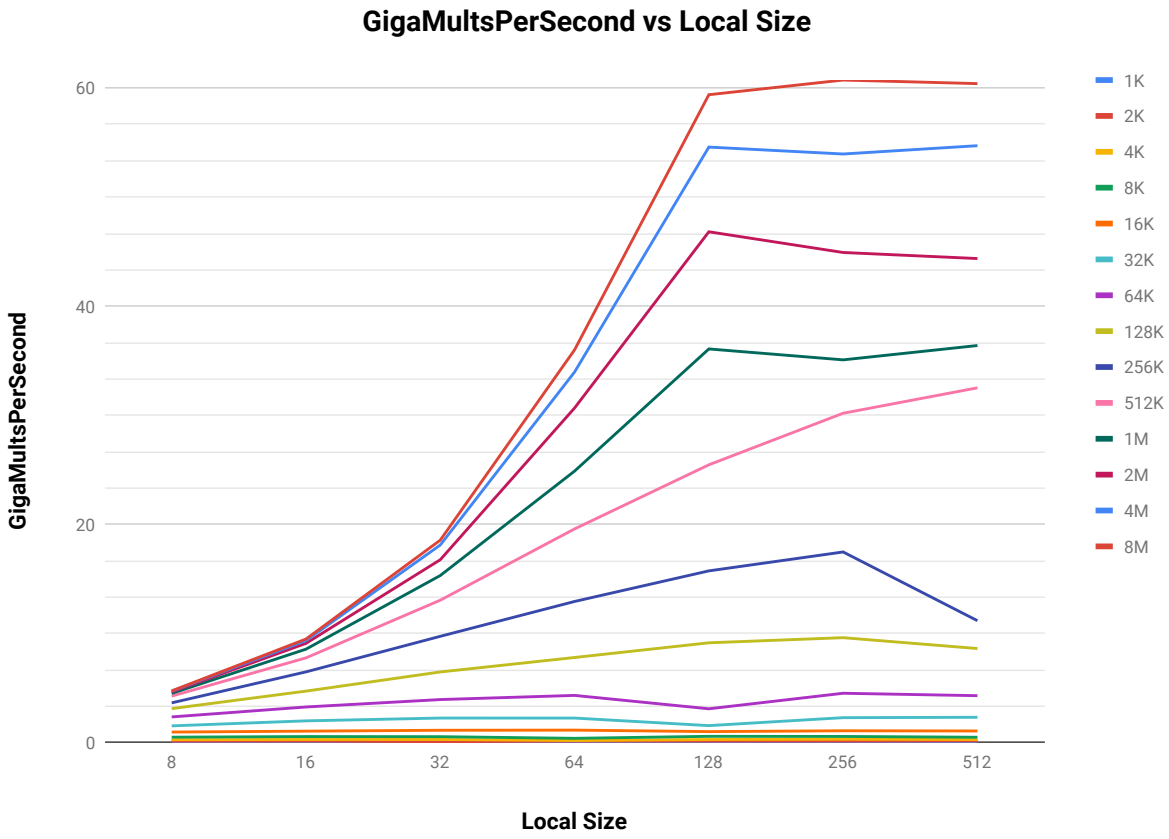

Figure 2: Multiply Performance vs. Local Size with multiple curves of Global Size.

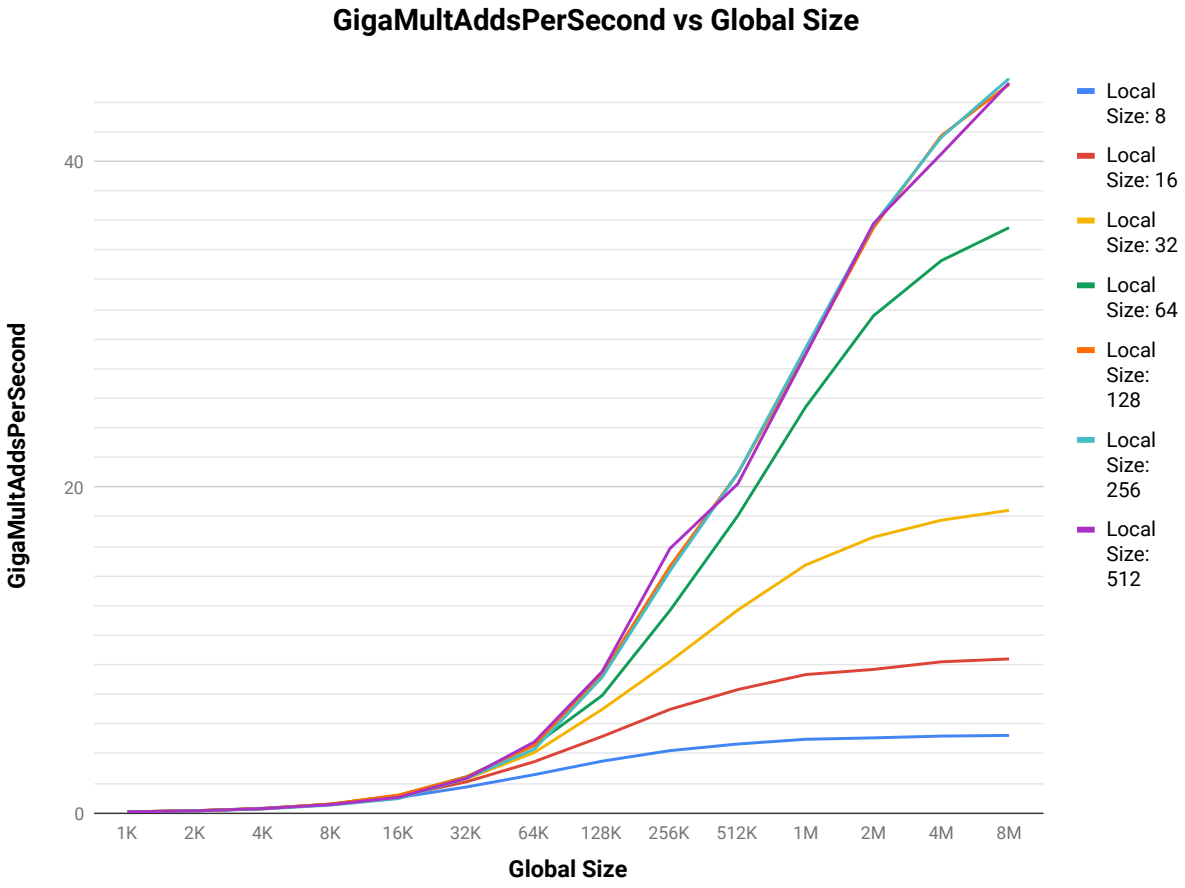

Figure 3: Multiply-Add Performance vs. Global Size with multiple curves of Local Size.

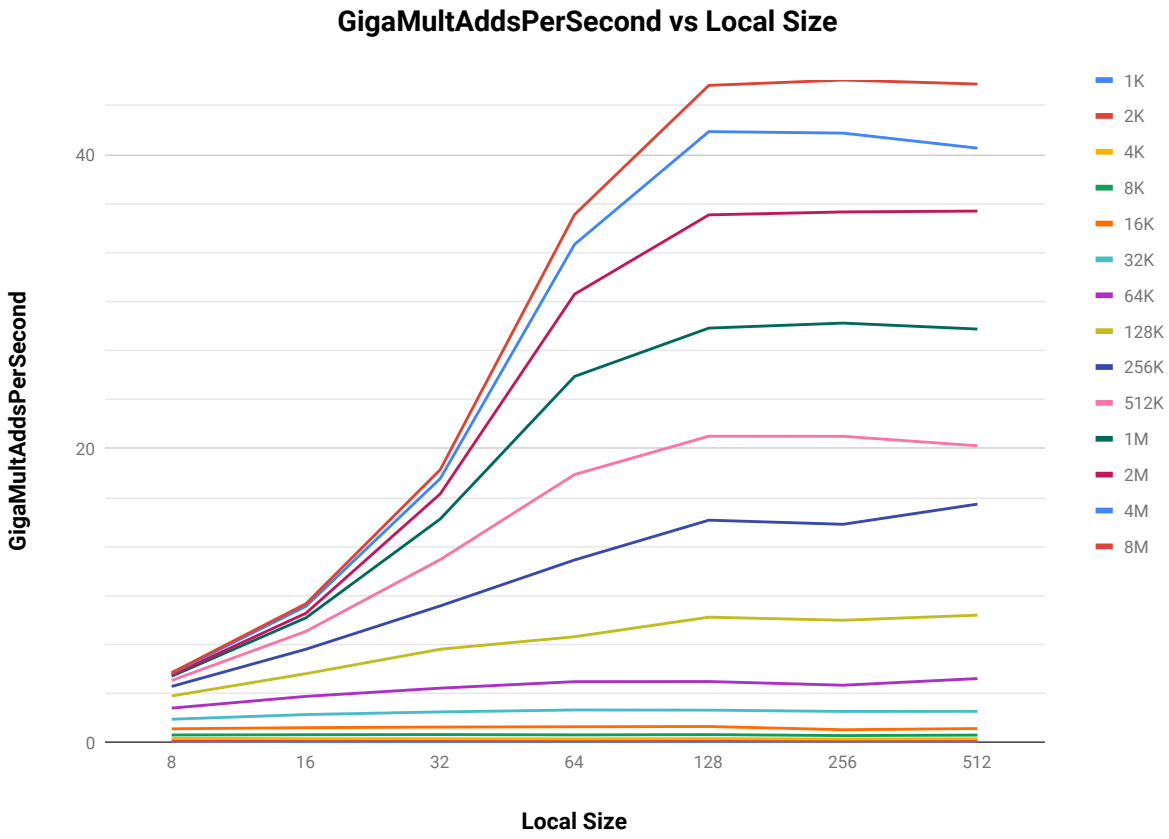

Figure 4: Multiply-Add Performance vs. Local Size with multiple curves of Global Size.

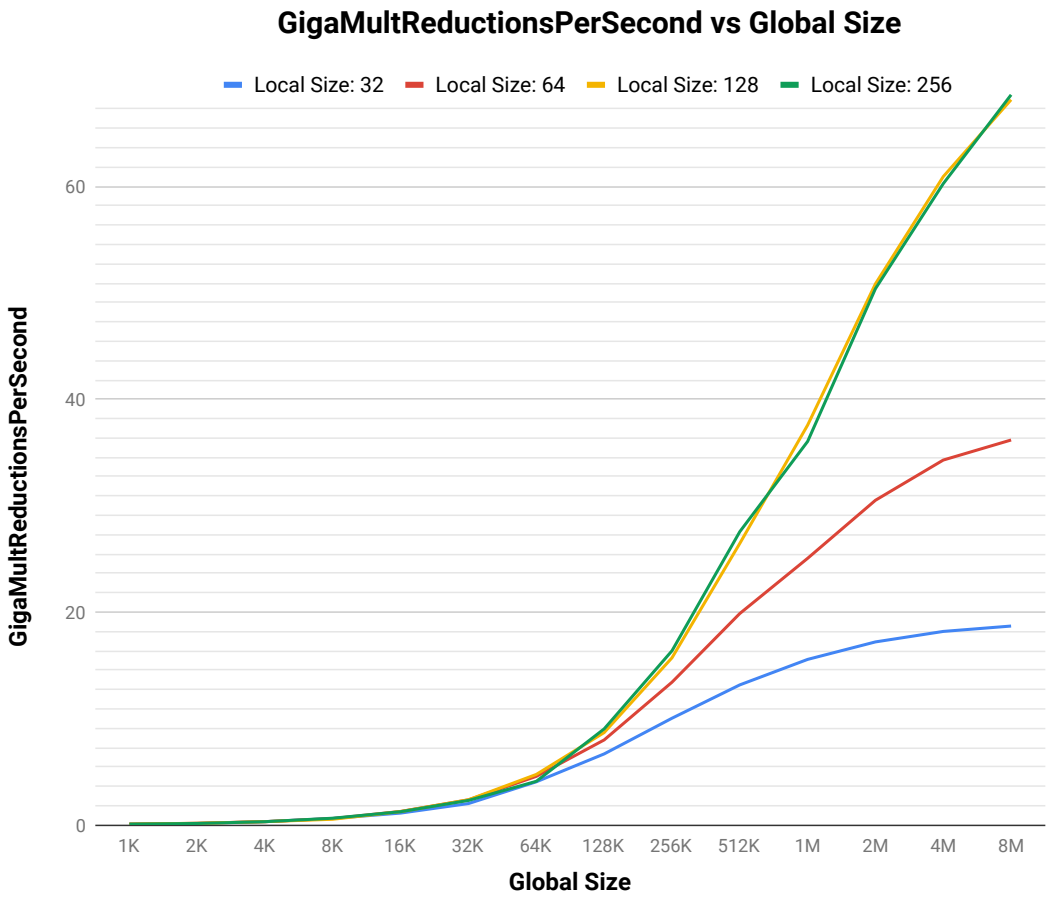

Figure 5: Reduction-Multiply Performance vs. Local Size with multiple curves of Global Size.

6 Results: Tables

Table 2 shows the results of the OpenCL Multiply Kernel and Table 2 shows the Multiply-Add Kernel. Finally, Table 3 shows the results for the Reduction Multiply Kernel.

Global Size	Local Size	# of Work Groups	GigaMults/sec
1024	8	128	0.066
2048	8	256	0.14
4096	8	512	0.269
8192	8	1024	0.5
16384	8	2048	0.968
32768	8	4096	1.536

65536	8	8192	2.356
131072	8	16384	3.117
262144	8	32768	3.644
524288	8	65536	4.263
1048576	8	131072	4.512
2097152	8	262144	4.646
4194304	8	524288	4.718
8388608	8	1048576	4.741
1024	16	64	0.068
2048	16	128	0.14
4096	16	256	0.285
8192	16	512	0.555
16384	16	1024	1.052
32768	16	2048	1.995
65536	16	4096	3.267
131072	16	8192	4.721
262144	16	16384	6.485
524288	16	32768	7.767
1048576	16	65536	8.566
2097152	16	131072	9.099
4194304	16	262144	9.355
8388608	16	524288	9.494
1024	32	32	0.069
2048	32	64	0.088
4096	32	128	0.288
8192	32	256	0.544
16384	32	512	1.133
32768	32	1024	2.252
65536	32	2048	3.947
131072	32	4096	6.478
262144	32	8192	9.744
524288	32	16384	13.055
1048576	32	32768	15.315
2097152	32	65536	16.764
4194304	32	131072	18.112
8388608	32	262144	18.56
1024	64	16	0.068
2048	64	32	0.142
4096	64	64	0.193
8192	64	128	0.394

16384	64	256	1.146
32768	64	512	2.251
65536	64	1024	4.328
131072	64	2048	7.803
262144	64	4096	12.938
524288	64	8192	19.588
1048576	64	16384	24.887
2097152	64	32768	30.671
4194304	64	65536	33.968
8388608	64	131072	36
1024	128	8	0.059
2048	128	16	0.141
4096	128	32	0.285
8192	128	64	0.577
16384	128	128	1.01
32768	128	256	1.559
65536	128	512	3.101
131072	128	1024	9.15
262144	128	2048	15.742
524288	128	4096	25.464
1048576	128	8192	36.084
2097152	128	16384	46.823
4194304	128	32768	54.586
8388608	128	65536	59.387
1024	256	4	0.078
2048	256	8	0.138
4096	256	16	0.308
8192	256	32	0.565
16384	256	64	1.087
32768	256	128	2.288
65536	256	256	4.525
131072	256	512	9.62
262144	256	1024	17.475
524288	256	2048	30.191
1048576	256	4096	35.084
2097152	256	8192	44.925
4194304	256	16384	53.952
8388608	256	32768	60.73
1024	512	2	0.07
2048	512	4	0.157

4096	512	8	0.286
8192	512	16	0.492
16384	512	32	1.068
32768	512	64	2.322
65536	512	128	4.303
131072	512	256	8.623
262144	512	512	11.176
524288	512	1024	32.52
1048576	512	2048	36.397
2097152	512	4096	44.37
4194304	512	8192	54.714
8388608	512	16384	60.404
8388608	512	16384	44.809

Table 1: Results for Multiply Kernel

Global Size	Local Size	# of Work Groups	GigaMultAdds/sec
1024	8	128	0.059
2048	8	256	0.121
4096	8	512	0.29
8192	8	1024	0.521
16384	8	2048	0.941
32768	8	4096	1.592
65536	8	8192	2.349
131072	8	16384	3.177
262144	8	32768	3.823
524288	8	65536	4.23
1048576	8	131072	4.521
2097152	8	262144	4.607
4194304	8	524288	4.716
8388608	8	1048576	4.758
1024	16	64	0.065
2048	16	128	0.138
4096	16	256	0.268
8192	16	512	0.536
16384	16	1024	1.018
32768	16	2048	1.911
65536	16	4096	3.147
131072	16	8192	4.693
262144	16	16384	6.356

524288	16	32768	7.571
1048576	16	65536	8.492
2097152	16	131072	8.808
4194304	16	262144	9.276
8388608	16	524288	9.453
1024	32	32	0.066
2048	32	64	0.136
4096	32	128	0.266
8192	32	256	0.548
16384	32	512	1.058
32768	32	1024	2.09
65536	32	2048	3.707
131072	32	4096	6.354
262144	32	8192	9.303
524288	32	16384	12.454
1048576	32	32768	15.219
2097152	32	65536	16.929
4194304	32	131072	17.967
8388608	32	262144	18.575
1024	64	16	0.067
2048	64	32	0.13
4096	64	64	0.255
8192	64	128	0.526
16384	64	256	1.08
32768	64	512	2.222
65536	64	1024	4.148
131072	64	2048	7.205
262144	64	4096	12.428
524288	64	8192	18.241
1048576	64	16384	24.916
2097152	64	32768	30.512
4194304	64	65536	33.9
8388608	64	131072	35.927
1024	128	8	0.065
2048	128	16	0.133
4096	128	32	0.272
8192	128	64	0.545
16384	128	128	1.097
32768	128	256	2.21
65536	128	512	4.157

131072	128	1024	8.536
262144	128	2048	15.136
524288	128	4096	20.848
1048576	128	8192	28.205
2097152	128	16384	35.91
4194304	128	32768	41.575
8388608	128	65536	44.72
1024	256	4	0.061
2048	256	8	0.127
4096	256	16	0.245
8192	256	32	0.48
16384	256	64	0.878
32768	256	128	2.122
65536	256	256	3.906
131072	256	512	8.326
262144	256	1024	14.856
524288	256	2048	20.843
1048576	256	4096	28.546
2097152	256	8192	36.109
4194304	256	16384	41.472
8388608	256	32768	45.082
1024	512	2	0.066
2048	512	4	0.126
4096	512	8	0.264
8192	512	16	0.513
16384	512	32	0.958
32768	512	64	2.13
65536	512	128	4.359
131072	512	256	8.675
262144	512	512	16.229
524288	512	1024	20.196
1048576	512	2048	28.14
2097152	512	4096	36.167
4194304	512	8192	40.451
8388608	512	16384	44.809

Table 2: Results for Multiply-Add Kernel

Global Size	Local Size	# of Work Groups	GigaReductionMults/sec
1024	32	32	0.073

2048	32	64	0.147
4096	32	128	0.309
8192	32	256	0.61
16384	32	512	1.114
32768	32	1024	2.01
65536	32	2048	4.055
131072	32	4096	6.674
262144	32	8192	10.028
524288	32	16384	13.161
1048576	32	32768	15.556
2097152	32	65536	17.2
4194304	32	131072	18.185
8388608	32	262144	18.691
1024	64	16	0.078
2048	64	32	0.152
4096	64	64	0.308
8192	64	128	0.601
16384	64	256	1.268
32768	64	512	2.36
65536	64	1024	4.566
131072	64	2048	7.988
262144	64	4096	13.413
524288	64	8192	19.869
1048576	64	16384	25.056
2097152	64	32768	30.511
4194304	64	65536	34.29
8388608	64	131072	36.175
1024	128	8	0.078
2048	128	16	0.156
4096	128	32	0.309
8192	128	64	0.535
16384	128	128	1.257
32768	128	256	2.366
65536	128	512	4.747
131072	128	1024	8.681
262144	128	2048	15.706
524288	128	4096	26.457
1048576	128	8192	37.573
2097152	128	16384	50.851
4194304	128	32768	60.929

8388608	128	65536	68.152
1024	256	4	0.078
2048	256	8	0.137
4096	256	16	0.306
8192	256	32	0.632
16384	256	64	1.246
32768	256	128	2.306
65536	256	256	4.1
131072	256	512	9.025
262144	256	1024	16.366
524288	256	2048	27.559
1048576	256	4096	36.039
2097152	256	8192	50.397
4194304	256	16384	60.267
8388608	256	32768	68.614

Table 3: Results for Reduction Multiply Kernel.