Kevin Harvell
harvellk@oregonstate.edu

Project 5 - OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce

Multiply and Multiply-Add

1. **What machine you ran this on**
   Rabbit
2. **Show the tables and graphs**

Multiply

# of Elements	Local Size	# of Work Groups	GigaMultsPerSecond
1024	8	128	0.0284966
2048	8	256	0.0595006
4096	8	512	0.119169
8192	8	1024	0.239063
16384	8	2048	0.231696
32768	8	4096	0.904297
65536	8	8192	0.783896
131072	8	16384	3.67826
262144	8	32768	1.40423
524288	8	65536	2.31552
1048576	8	131072	4.88003
2097152	8	262144	9.02098
4194304	8	524288	16.4147
8388608	8	1048576	28.9842
1024	16	64	0.0302762
2048	16	128	0.0330452

4096	**16**	**256**	**0.122283**
8192	**16**	**512**	**0.114072**
16384	**16**	**1024**	**0.233752**
32768	**16**	**2048**	**0.427826**
65536	**16**	**4096**	**1.65449**
131072	**16**	**8192**	**3.87195**
262144	**16**	**16384**	**0.696689**
524288	**16**	**32768**	**1.73423**
1048576	**16**	**65536**	**5.43194**
2097152	**16**	**131072**	**6.63195**
4194304	**16**	**262144**	**15.2991**
8388608	**16**	**524288**	**28.5838**
1024	**32**	**32**	**0.0147372**
2048	**32**	**64**	**0.0248978**
4096	**32**	**128**	**0.114312**
8192	**32**	**256**	**0.240239**
16384	**32**	**512**	**0.467653**
32768	**32**	**1024**	**0.945156**
65536	**32**	**2048**	**1.97743**
131072	**32**	**4096**	**1.78442**
262144	**32**	**8192**	**0.796306**
524288	**32**	**16384**	**2.63189**
1048576	**32**	**32768**	**5.50223**
2097152	**32**	**65536**	**8.84905**

4194304	**32**	**131072**	**17.503**
8388608	**32**	**262144**	**27.8406**
1024	**64**	**16**	**0.0290922**
2048	**64**	**32**	**0.0588352**
4096	**64**	**64**	**0.0572409**
8192	**64**	**128**	**0.233665**
16384	**64**	**256**	**0.473647**
32768	**64**	**512**	**0.452765**
65536	**64**	**1024**	**1.8754**
131072	**64**	**2048**	**3.78123**
262144	**64**	**4096**	**1.33288**
524288	**64**	**8192**	**1.90687**
1048576	**64**	**16384**	**5.35592**
2097152	**64**	**32768**	**9.95156**
4194304	**64**	**65536**	**16.0752**
8388608	**64**	**131072**	**28.2262**
1024	**128**	**8**	**0.0158044**
2048	**128**	**16**	**0.0583977**
4096	**128**	**32**	**0.117281**
8192	**128**	**64**	**0.232972**
16384	**128**	**128**	**0.233665**
32768	**128**	**256**	**0.882787**
65536	**128**	**512**	**1.91667**
131072	**128**	**1024**	**3.82418**

262144	**128**	**2048**	**1.36941**
524288	**128**	**4096**	**2.36188**
1048576	**128**	**8192**	**5.32718**
2097152	**128**	**16384**	**9.65683**
4194304	**128**	**32768**	**17.0868**
8388608	**128**	**65536**	**28.3201**
1024	**256**	**4**	**0.0299741**
2048	**256**	**8**	**0.0594395**
4096	**256**	**16**	**0.117256**
8192	**256**	**32**	**0.240646**
16384	**256**	**64**	**0.23082**
32768	**256**	**128**	**0.607234**
65536	**256**	**256**	**1.85122**
131072	**256**	**512**	**3.33849**
262144	**256**	**1024**	**1.33655**
524288	**256**	**2048**	**2.3961**
1048576	**256**	**4096**	**5.56027**
2097152	**256**	**8192**	**9.55253**
4194304	**256**	**16384**	**16.3068**
8388608	**256**	**32768**	**28.6226**
1024	**512**	**2**	**0.0286719**
2048	**512**	**4**	**0.0574187**
4096	**512**	**8**	**0.120839**
8192	**512**	**16**	**0.115805**

16384	512	32	0.452963
32768	512	64	0.72369
65536	512	128	0.866546
131072	512	256	3.7562
262144	512	512	1.32606
524288	512	1024	2.34756
1048576	512	2048	4.46959
2097152	512	4096	10.8745
4194304	512	8192	13.907
8388608	512	16384	25.9464

Multiply-Add

# of Elements	Local Size	# of Work Groups	GigaMultsPerSecond
1024	8	128	0.0299709
2048	8	256	0.05881
4096	8	512	0.0590564
8192	8	1024	0.112126
16384	8	2048	0.476237
32768	8	4096	0.455614
65536	8	8192	1.91251
131072	8	16384	3.78408
262144	8	32768	1.0397
524288	8	65536	2.7177

1048576	**8**	**131072**	**4.13129**
2097152	**8**	**262144**	**10.5679**
4194304	**8**	**524288**	**13.7916**
8388608	**8**	**1048576**	**27.9647**
1024	**16**	**64**	**0.0290953**
2048	**16**	**128**	**0.0578995**
4096	**16**	**256**	**0.123333**
8192	**16**	**512**	**0.238519**
16384	**16**	**1024**	**0.46305**
32768	**16**	**2048**	**0.919036**
65536	**16**	**4096**	**0.927613**
131072	**16**	**8192**	**2.36351**
262144	**16**	**16384**	**0.94113**
524288	**16**	**32768**	**2.7624**
1048576	**16**	**65536**	**5.61277**
2097152	**16**	**131072**	**9.68191**
4194304	**16**	**262144**	**12.1143**
8388608	**16**	**524288**	**27.4854**
1024	**32**	**32**	**0.0302963**
2048	**32**	**64**	**0.0604559**
4096	**32**	**128**	**0.116166**
8192	**32**	**256**	**0.23754**
16384	**32**	**512**	**0.23296**
32768	**32**	**1024**	**0.936801**

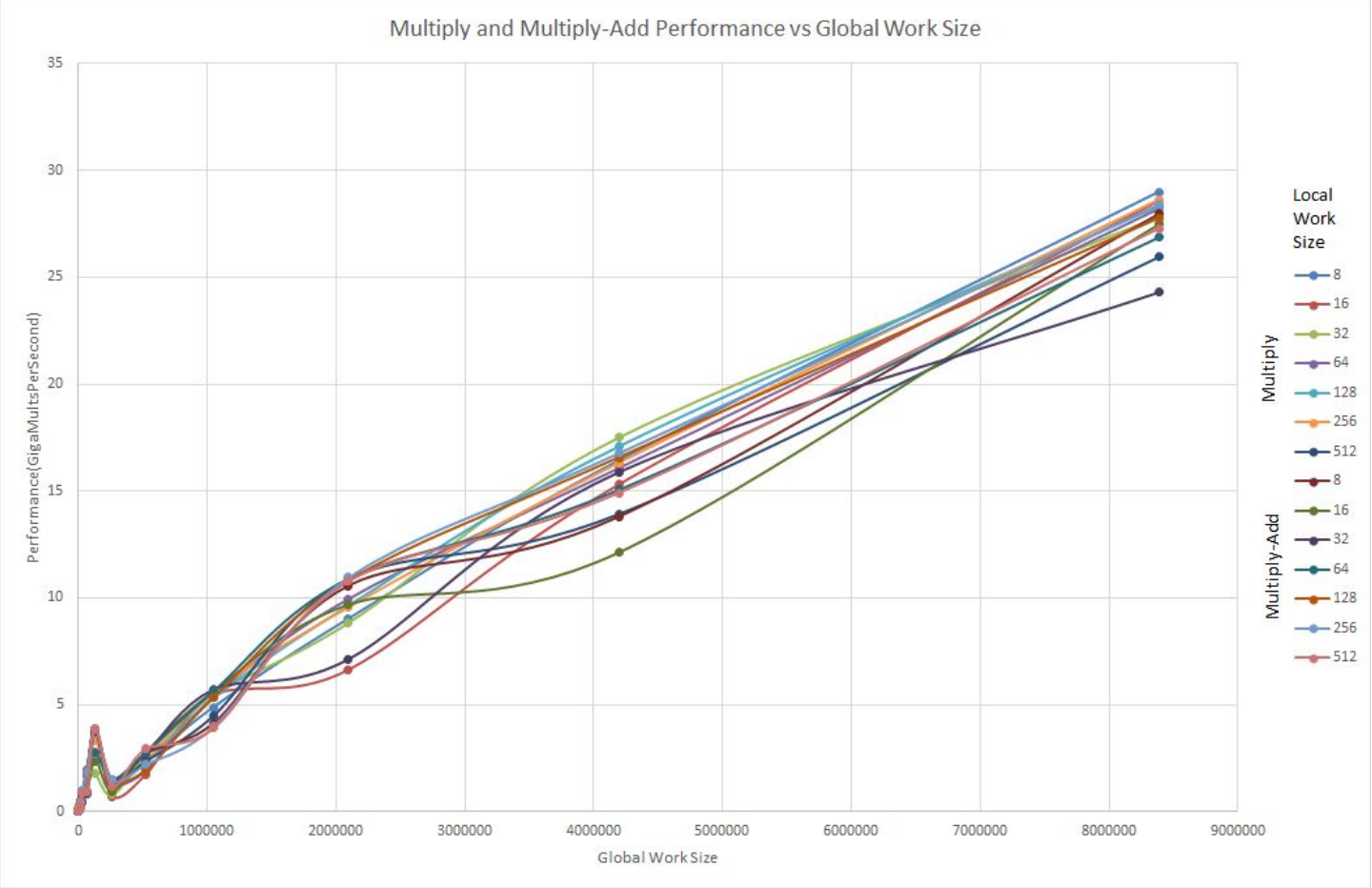
65536	**32**	**2048**	**1.87022**
131072	**32**	**4096**	**3.7242**
262144	**32**	**8192**	**1.49066**
524288	**32**	**16384**	**2.73224**
1048576	**32**	**32768**	**5.71071**
2097152	**32**	**65536**	**7.1273**
4194304	**32**	**131072**	**15.8852**
8388608	**32**	**262144**	**24.3155**
1024	**64**	**16**	**0.0291385**
2048	**64**	**32**	**0.0569607**
4096	**64**	**64**	**0.120972**
8192	**64**	**128**	**0.116246**
16384	**64**	**256**	**0.480346**
32768	**64**	**512**	**0.932629**
65536	**64**	**1024**	**1.91971**
131072	**64**	**2048**	**2.74899**
262144	**64**	**4096**	**1.5012**
524288	**64**	**8192**	**2.75632**
1048576	**64**	**16384**	**5.62792**
2097152	**64**	**32768**	**10.8939**
4194304	**64**	**65536**	**15.0494**
8388608	**64**	**131072**	**26.8743**
1024	**128**	**8**	**0.0299872**
2048	**128**	**16**	**0.053209**

4096	**128**	**32**	**0.0575164**
8192	**128**	**64**	**0.232197**
16384	**128**	**128**	**0.479874**
32768	**128**	**256**	**0.946987**
65536	**128**	**512**	**1.91064**
131072	**128**	**1024**	**3.87067**
262144	**128**	**2048**	**1.47747**
524288	**128**	**4096**	**1.94436**
1048576	**128**	**8192**	**5.3228**
2097152	**128**	**16384**	**10.8088**
4194304	**128**	**32768**	**16.5574**
8388608	**128**	**65536**	**27.7519**
1024	**256**	**4**	**0.0145801**
2048	**256**	**8**	**0.0579911**
4096	**256**	**16**	**0.0583249**
8192	**256**	**32**	**0.239336**
16384	**256**	**64**	**0.443732**
32768	**256**	**128**	**0.967667**
65536	**256**	**256**	**1.88061**
131072	**256**	**512**	**3.88692**
262144	**256**	**1024**	**1.45989**
524288	**256**	**2048**	**2.22684**
1048576	**256**	**4096**	**3.98984**
2097152	**256**	**8192**	**10.9621**

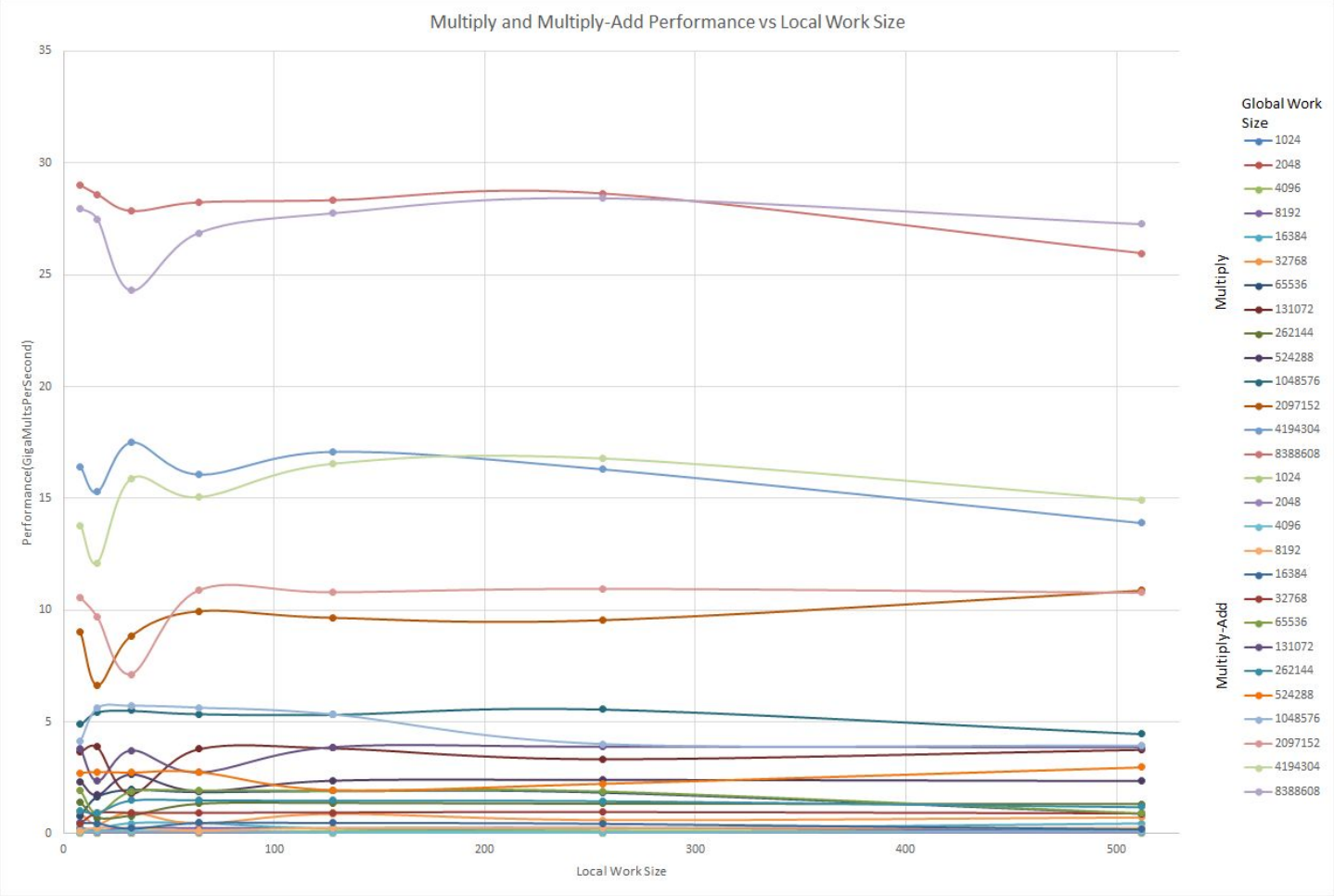
Kevin Harvell
harvellk@oregonstate.edu

4194304	**256**	**16384**	**16.7891**
8388608	**256**	**32768**	**28.4169**
1024	**512**	**2**	**0.0146488**
2048	**512**	**4**	**0.0599026**
4096	**512**	**8**	**0.117269**
8192	**512**	**16**	**0.240488**
16384	**512**	**32**	**0.198428**
32768	**512**	**64**	**0.904762**
65536	**512**	**128**	**0.924032**
131072	**512**	**256**	**3.86069**
262144	**512**	**512**	**1.19362**
524288	**512**	**1024**	**2.96442**
1048576	**512**	**2048**	**3.92127**
2097152	**512**	**4096**	**10.7882**
4194304	**512**	**8192**	**14.9133**
8388608	**512**	**16384**	**27.2613**

Kevin Harvell
harvellk@oregonstate.edu
Project 5 - OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce



Kevin Harvell
harvellk@oregonstate.edu
Project 5 - OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce



3. **What patterns are you seeing in the performance curves?**
   In general, it appears that performance is directly dependent on global work size. It seems that the greater the global work size, the better the performance, with some minor variations due to local work size. Performance between multiply and multiply-add was close. There doesn't seem to be a clear trend indicating if multiply or multiply-add is more performant. Local work size doesn't seem to make a significant impact either. It seems as though smaller local work sizes produce some squirrely behavior, but performance levels out at around 64 for local work size. Performance starts to marginally decline at around a local work size of 256.
4. **Why do you think the patterns look this way?**
   I think the biggest, most conclusive pattern of note is that the performance is dependent on global work size. This makes sense because we are using the GPU to perform the same calculation over and over again. This is what GPU's are built for. The more data we give the GPU, the faster it seems to tear through the data. I suspect that this would only go to a certain point due to there being a limited number of cores on a GPU. I don't think I reached that point because rabbit has impressive specifications.
5. **What is the performance difference between doing a Multiply and doing a Multiply-Add?**
   Performance difference between multiply and multiply-add is inconclusive at best for my results. There are times when multiply is more performant and there seem to be equal times where multiply-add is

more performant. I think due to the nature of GPU being made for these kinds of operations, it rips through the operations on the data no matter if there is an extra step or not.

6. **What does that mean for the proper use of GPU parallel computing?**
   I suspect that I should have gotten that multiply is more performant than multiply-add because multiply-add has to store intermediate values, which storage is not the GPU's forte. It makes sense in theory that a GPU would perform a straight multiply faster than a multiply-add because of this. My results didn't seem to prove the theory.

Multiply-Reduction

1. **Show this table and graph**

# of Elements	Local Size	# of Work Groups	GigaMultsPerSecond
1024	8	128	0.00316574
2048	8	256	0.0112834
4096	8	512	0.0200224
8192	8	1024	0.0410028
16384	8	2048	0.0854786
32768	8	4096	0.149064
65536	8	8192	0.247674
131072	8	16384	0.49418
262144	8	32768	0.599329
524288	8	65536	0.846765
1048576	8	131072	0.692291
2097152	8	262144	0.669207
4194304	8	524288	1.18786
8388608	8	1048576	1.22161
1024	16	64	0.00574542
2048	16	128	0.0111002

4096	**16**	**256**	**0.0225206**
8192	**16**	**512**	**0.0405133**
16384	**16**	**1024**	**0.0865578**
32768	**16**	**2048**	**0.151582**
65536	**16**	**4096**	**0.251285**
131072	**16**	**8192**	**0.438104**
262144	**16**	**16384**	**0.889679**
524288	**16**	**32768**	**0.911911**
1048576	**16**	**65536**	**1.39817**
2097152	**16**	**131072**	**0.791572**
4194304	**16**	**262144**	**1.93612**
8388608	**16**	**524288**	**2.0737**
1024	**32**	**32**	**0.00297308**
2048	**32**	**64**	**0.0113291**
4096	**32**	**128**	**0.0229032**
8192	**32**	**256**	**0.0444314**
16384	**32**	**512**	**0.0900612**
32768	**32**	**1024**	**0.165751**
65536	**32**	**2048**	**0.247875**
131072	**32**	**4096**	**0.468691**
262144	**32**	**8192**	**0.887458**
524288	**32**	**16384**	**1.44201**
1048576	**32**	**32768**	**1.90108**
2097152	**32**	**65536**	**1.01379**

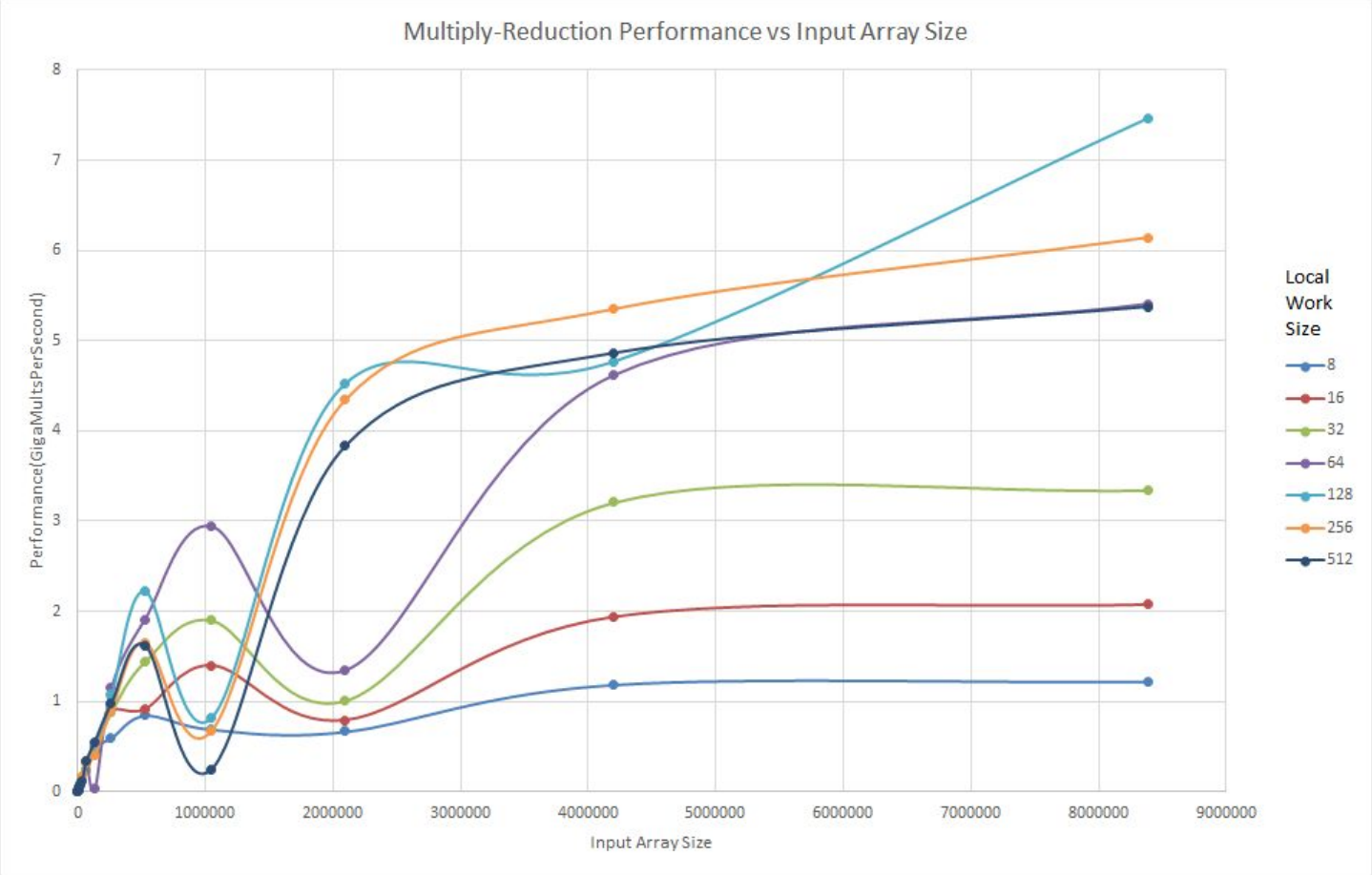
4194304	**32**	**131072**	**3.20771**
8388608	**32**	**262144**	**3.34469**
1024	**64**	**16**	**0.003963**
2048	**64**	**32**	**0.00900707**
4096	**64**	**64**	**0.0211469**
8192	**64**	**128**	**0.0441855**
16384	**64**	**256**	**0.0833255**
32768	**64**	**512**	**0.138783**
65536	**64**	**1024**	**0.319762**
131072	**64**	**2048**	**0.0314633**
262144	**64**	**4096**	**1.15912**
524288	**64**	**8192**	**1.91301**
1048576	**64**	**16384**	**2.94328**
2097152	**64**	**32768**	**1.35219**
4194304	**64**	**65536**	**4.61609**
8388608	**64**	**131072**	**5.4087**
1024	**128**	**8**	**0.00362072**
2048	**128**	**16**	**0.0103215**
4096	**128**	**32**	**0.0210937**
8192	**128**	**64**	**0.0417827**
16384	**128**	**128**	**0.0688453**
32768	**128**	**256**	**0.159729**
65536	**128**	**512**	**0.337532**
131072	**128**	**1024**	**0.466584**

262144	**128**	**2048**	**1.07399**
524288	**128**	**4096**	**2.22573**
1048576	**128**	**8192**	**0.827015**
2097152	**128**	**16384**	**4.52978**
4194304	**128**	**32768**	**4.76915**
8388608	**128**	**65536**	**7.46744**
1024	**256**	**4**	**0.0056917**
2048	**256**	**8**	**0.0107596**
4096	**256**	**16**	**0.020194**
8192	**256**	**32**	**0.0453402**
16384	**256**	**64**	**0.0828905**
32768	**256**	**128**	**0.163203**
65536	**256**	**256**	**0.326561**
131072	**256**	**512**	**0.399496**
262144	**256**	**1024**	**0.899293**
524288	**256**	**2048**	**1.64783**
1048576	**256**	**4096**	**0.673281**
2097152	**256**	**8192**	**4.34465**
4194304	**256**	**16384**	**5.34891**
8388608	**256**	**32768**	**6.14375**
1024	**512**	**2**	**0.00557488**
2048	**512**	**4**	**0.0111204**
4096	**512**	**8**	**0.01819**
8192	**512**	**16**	**0.0455445**

Kevin Harvell
harvellk@oregonstate.edu

16384	**512**	**32**	**0.0903117**
32768	**512**	**64**	**0.120872**
65536	**512**	**128**	**0.342667**
131072	**512**	**256**	**0.550138**
262144	**512**	**512**	**0.982056**
524288	**512**	**1024**	**1.62062**
1048576	**512**	**2048**	**0.249313**
2097152	**512**	**4096**	**3.83237**
4194304	**512**	**8192**	**4.85759**
8388608	**512**	**16384**	**5.37222**

Kevin Harvell
harvellk@oregonstate.edu
Project 5 - OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce



2. **What pattern are you seeing in this performance curve?**
   Overall, performance appears to trend up the larger the input array size gets. Interestingly, the local work size of 128 is more performant than any other for very large input array sizes. A local size of 256 comes mostly dominates in performance for all most input array sizes. Smaller local work sizes do not appear to perform as well as their counterparts. As usual, there is some squirrely behavior for smaller input array sizes. In general, performance is *much slower* than multiply and multiply-add.
3. **Why do you think the pattern looks this way?**
   Once again, because we are using the GPU which excels at performing the same calculation on tons of data, we see that performance gets better with larger input array sizes. I think there is a sweet spot when it comes to local work size that is reliant on the GPU's number of cores. If the local work size is too high or low, performance will suffer. The GPU's number of cores are finite, and thus has an optimum for local work size in order to process massive amounts of data.
4. **What does that mean for the proper use of GPU parallel computing?**
   For simpler operations, GPU parallel computing is unparalleled(see what I did there?) in terms of speeding up performance. It makes a tremendous impact on performance. Adding the aspect of a reduction in this example slowed performance significantly because GPU's are not intended to store

data as CPU's are. It seems that the simpler the operation being performed, and the more data, the better the GPU will process the data.

Summary:

I really enjoyed this project. I learned so much about GPU's. I honestly had no clue how they operate, but just knew that they made my video games look cool. It makes a lot of sense that they are best to perform quick calculations because anything visual relies on that speed to look better. I never would've thought that's what happens behind the scenes. It's interesting to think that the GPU can be used for things that are other than visual in nature, such as machine learning. It also got me thinking about bitcoin mining, which I've heard utilizes GPU's quite a bit.