Drake Seifert

CS475 Project 6

What machine you ran this on

All benchmark testing was ran on OSU's rabbit server.

Multiply and Multiply-Add

Show the tables

Multiply:
Left column ⇒ Local work size
Top row ⇒ Global work size
Elements ⇒ Performance in GigaMults/Sec

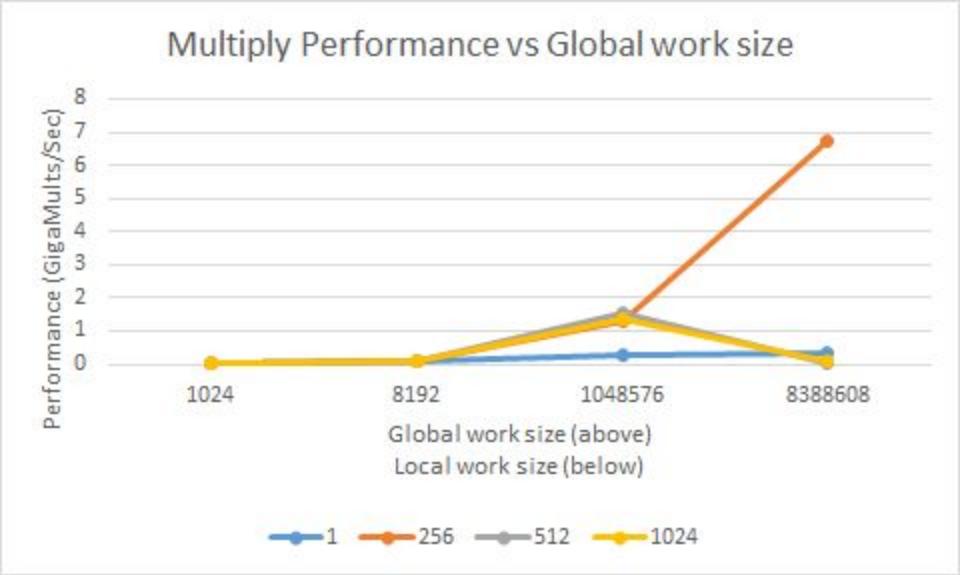
	1024	8192	1048576	8388608
1	0.01	0.103	0.295	0.327
256	0.014	0.106	1.28	6.705
512	0.011	0.083	1.55	0.059
1024	0.01	0.112	1.344	0.076

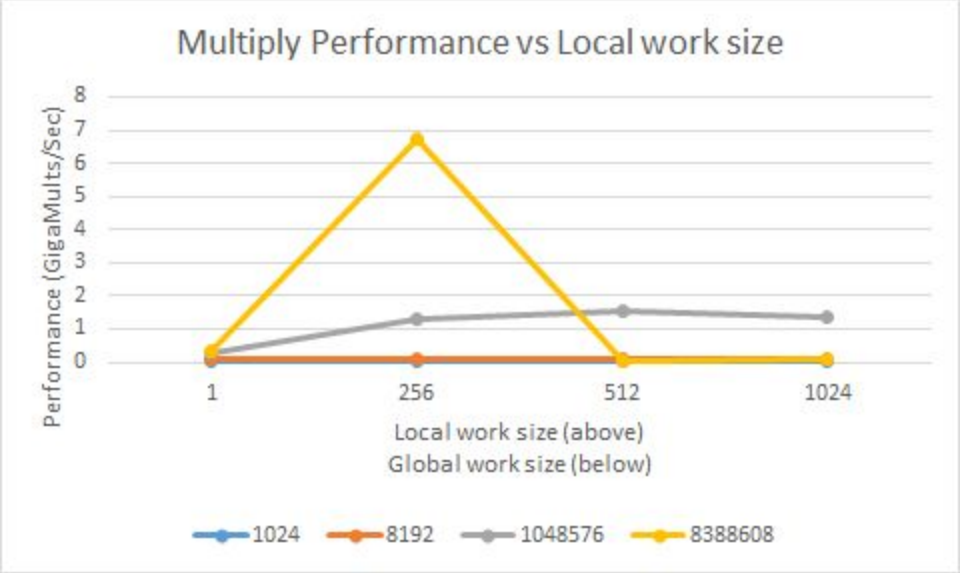
Multiply-Add:
Left column ⇒ Local work size
Top row ⇒ Global work size
Elements ⇒ Performance in GigaMults/Sec

	1024	8192	1048576	8388608
1	0.012	0.093	0.288	0.32
256	0.01	0.102	0.067	1.254
512	0.011	0.1	1.041	7.299
1024	0.01	0.089	0.744	1.903

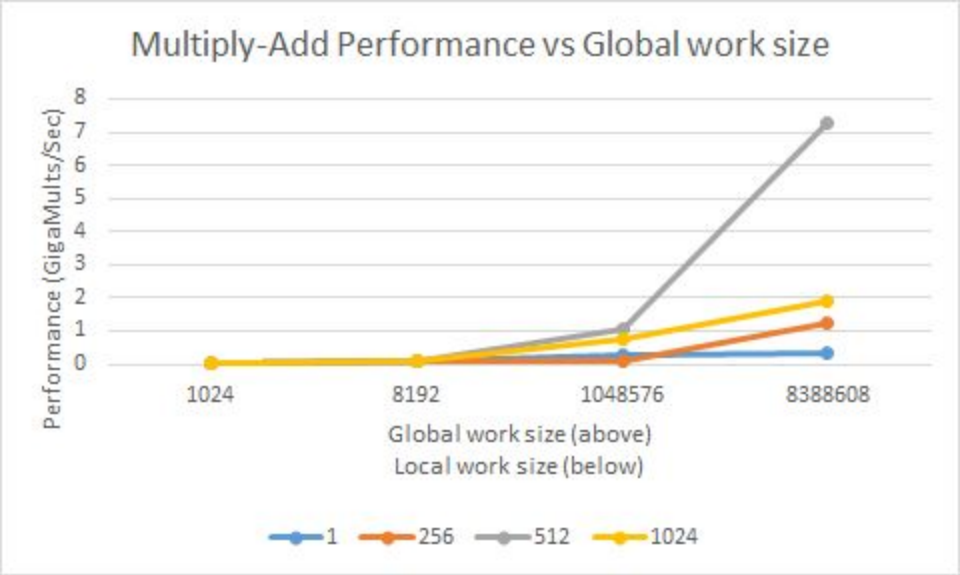
Show the graphs

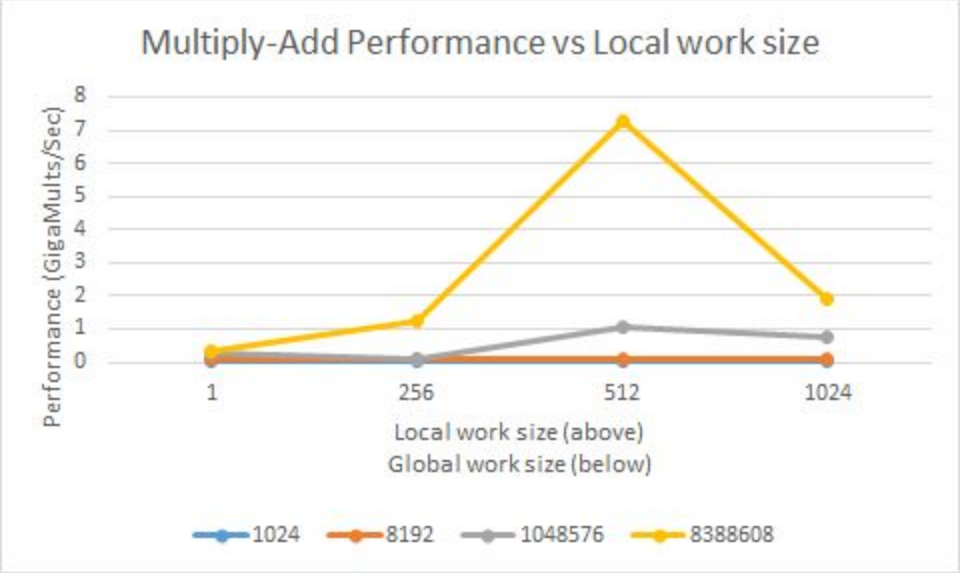
Multiply:





Multiply-Add:





What patterns are you seeing in the performance curves?

It appears that a larger global work size results in much better performance. In general a larger local work size also showed better performance, however both the multiply and the multiply-add cases had a “sweet spot” where there was an exception to this. It appears as though a local work size of 256 and a global work size of over 8 million gave by far superior performance over the opposing tests in the multiply case, and a local size of 512 with a global size of over 8 million in the multiply-add case.

Why do you think the patterns look this way?

In general, a larger global work size will allow the GPU cores to maximize their efficiency after a sufficient number of calculations. This is possibly because a larger global work size will decrease the overall overhead of the GPU and the GPU can do what it does best: computing simple functions. Different operations will have different “sweet spots” where the performance gains are optimal. It appears as though the local size sweet spot for the multiply case was around 256 and the local size sweet spot for the multiply-add case was around 512. This would mean a larger number of work groups is more optimal for the multiply case than the multiply-add case. This could possibly have to do with the complexity of the operation; a less complex operation can perform more optimally with a higher number of work groups than the corresponding more complex operation.

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

As mentioned above, the primary performance differences can be found based off the peak optimal values using a base of just over 8 million for the global work size. Looking at the graphs above shows an obvious deviation in performance in regards to the local work size; multiply’s optimal performance is around size 256 and multiply-add appears to be more optimal around 512.

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

When used properly, GPU parallel computing can be an absolute deadly tool. The important caveat to this is being able to perform tests that imply where the sweet spot of the performance gains lie. If used incorrectly, GPU computing can hardly be worth the effort. When used properly, no other method can compete.

<u>Multiply-Reduce</u>

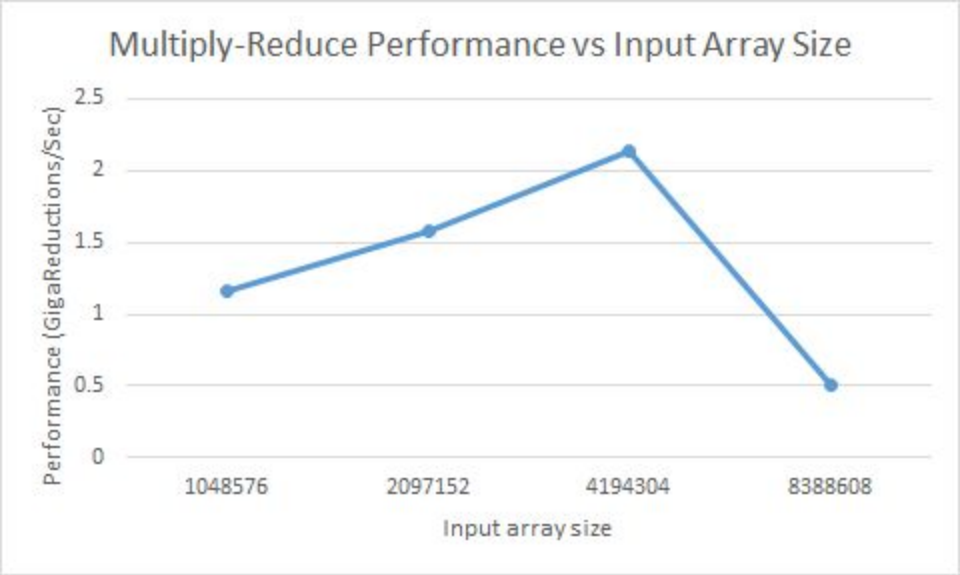
Show the table

Multiply-Reduce:
Left column ⇒ Local work size
Top row ⇒ Global work size
Elements ⇒ Performance in GigaMults/Sec

	1048576	2097152	4194304	8388608
32	1.167	1.579	2.142	0.51

Show the graph

Multiply-Reduce:



What pattern are you seeing in this performance curve?

A larger input array size tends to show greater performance gains for a local work size of 32 up until a global input size of about 4 million. The performance gains then begin to deplete as the array size gets larger.

Why do you think the pattern looks this way?

The GPU threads seem to max out in efficient when handling an array size of about 4 million. A smaller number may be less efficient because the full computing potential of the GPU cores are not fully taken advantage of due to the limited size. Too large of an array size may show performance loss due to the larger amount of overhead needed to track such a large reduction.

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

Similarly to the multiply and multiply-add cases, GPU parallel computing can be extremely powerful if used properly. It seems very necessary to run benchmark tests to find the sweet spot that a GPU will perform at for a given calculation. Otherwise, the GPU is hardly worth using.