Josh Huff, huffj@oregonstate.edu
Professor Bailey
CS 475 – Spring 2017
June 4, 2017

Project 6: Commentary

---

1. What machine you ran this on

My main workhorse does not have a GPU, and neither does flip. Therefore, my only option was rabbit.

The load averages were: 6.61, 6.85, 6.54.

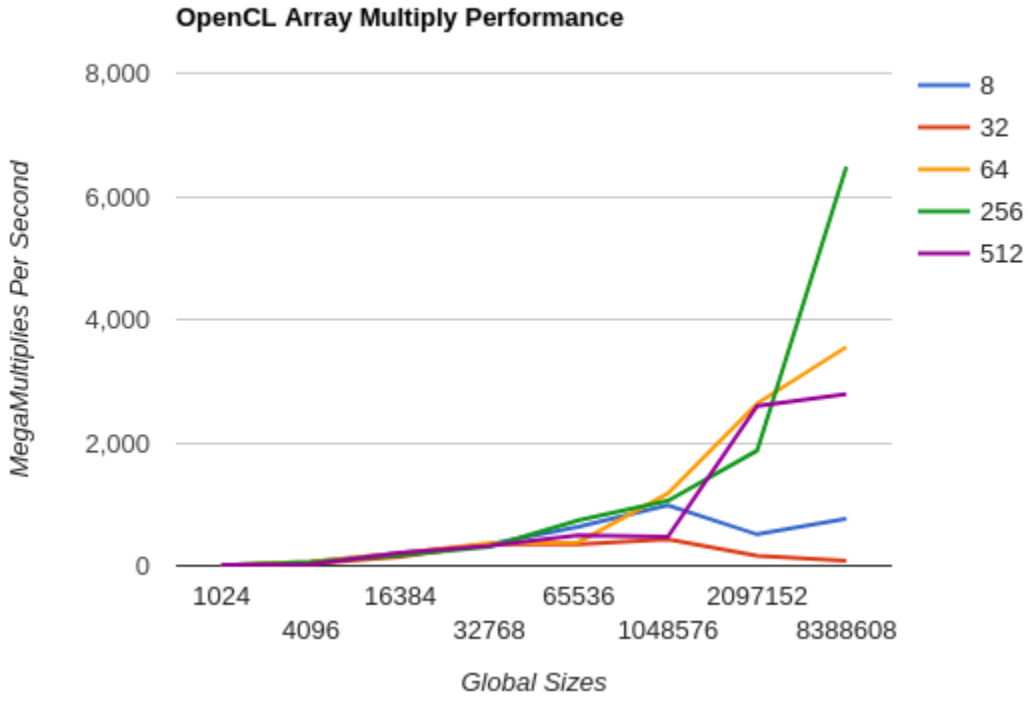
2. Show the tables and graphs.

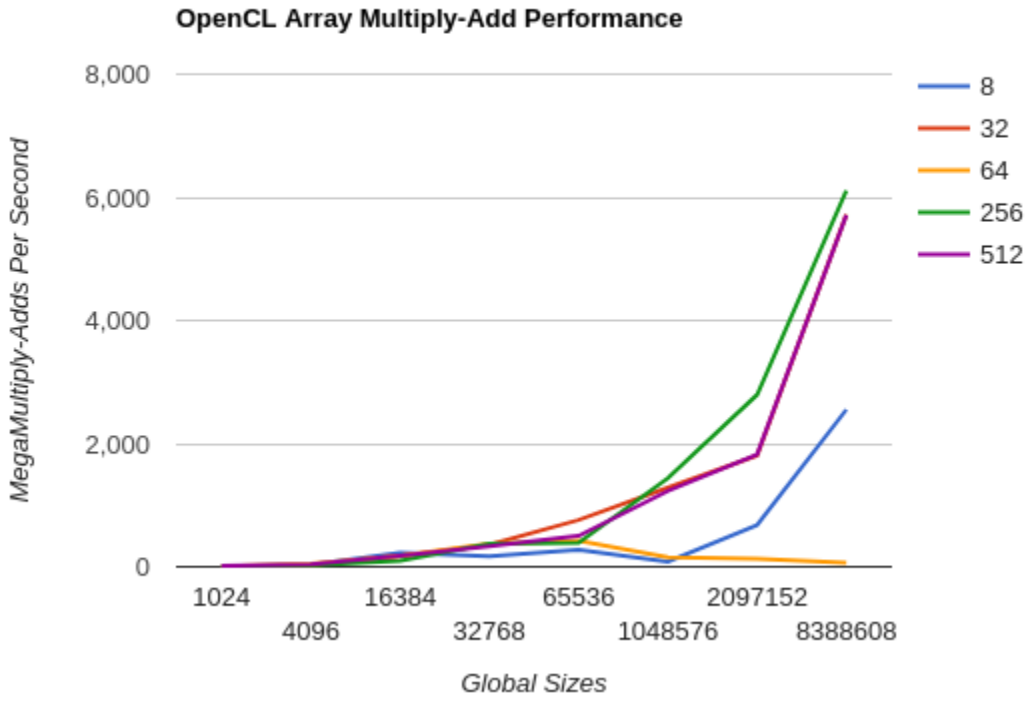
OpenCL Performance – Array Multiplication

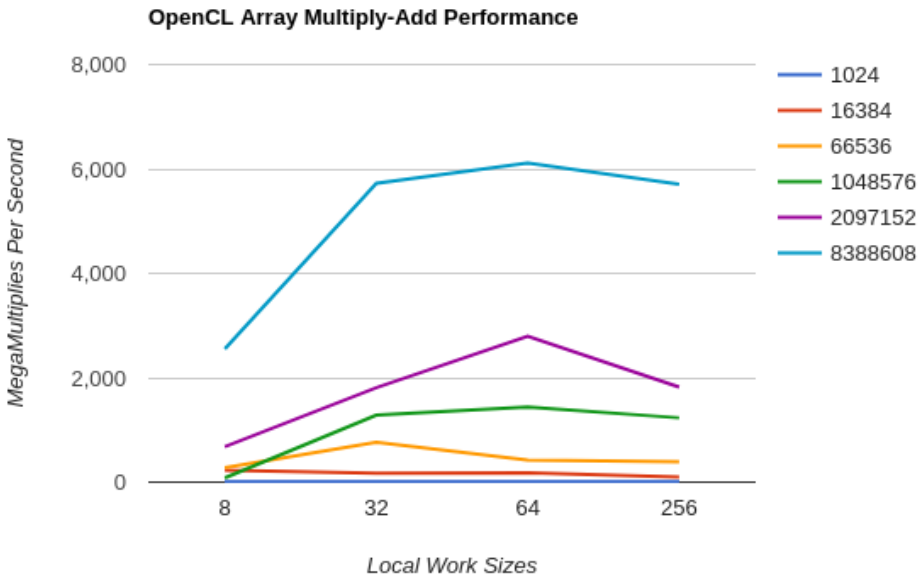
Global Work Size	Local Work Size				
	8	32	64	256	512
1024	8.89	7.26	6.34	4.29	17.55
4096	62.18	35.27	64.92	54.88	25.40
16384	204.97	143.67	202.54	169.65	213.99
32768	347.42	352.43	369.33	306.50	326.75
65536	636.76	347.43	385.07	739.59	497.91
262144	97.80	42.95	166.73	44.42	99.46
524288	49.64	414.93	758.97	639.85	551.44
1048576	982.87	430.11	1,172.87	1,054.37	468.97
2097152	509.42	159.65	2,634.94	1,871.46	2595.15
8388608	762.04	81.57	3,551.95	6,481.63	2785.78

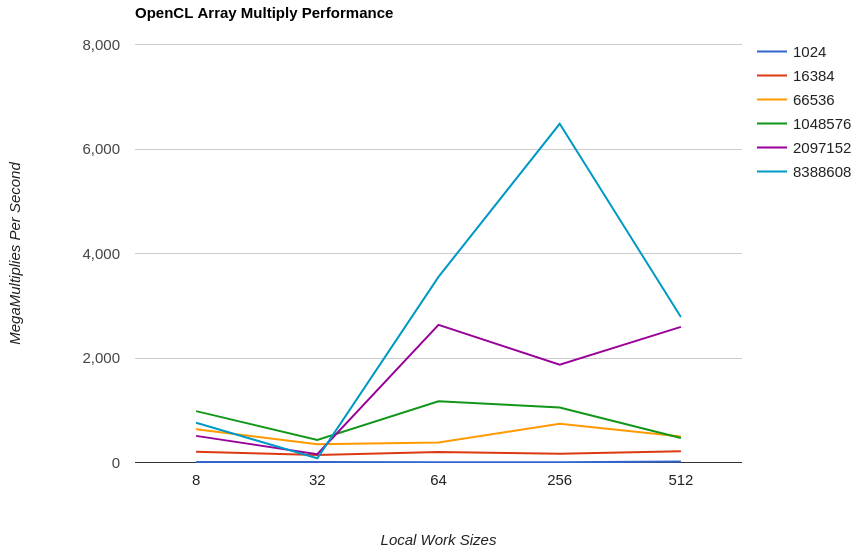
OpenCL Performance – Array Multiplication and Addition

Global Work Size	Local Work Size				
	8	32	64	256	512
1024	9.81	8.21	10.49	11.25	13.16
4096	36.39	25.78	48.42	26.61	37.22
16384	229.90	174.60	176.10	98.78	175.99
32768	173.29	355.54	370.81	376.99	336.45
65536	277.45	760.75	426.09	387.33	502.56
262144	355.76	24.98	23.83	227.96	374.53
524288	557.59	561.94	41.11	179.54	49.32
1048576	82.86	1,285.70	154.93	1,440.98	1231.63
2097152	677.45	1,807.48	133.57	2,794.99	1822.25
8388608	2,553.21	5,721.84	62.92	6,112.15	5709.32


OpenCL Array Multiply Performance
8,000
6,000
4,000
2,000
0
MegaMultiplies Per Second
1024
4096
16384
32768
65536
1048576
2097152
8388608
Global Sizes
8
32
64
256
512


OpenCL Array Multiply-Add Performance
8,000
6,000
4,000
2,000
0
MegaMultiply-Adds Per Second
1024
4096
16384
32768
65536
1048576
2097152
8388608
Global Sizes
8
32
64
256
512


OpenCL Array Multiply-Add Performance
8,000
6,000
4,000
2,000
0
MegaMultiplies Per Second
8
32
64
256
Local Work Sizes
1024
16384
66536
1048576
2097152
8388608


OpenCL Array Multiply Performance
8,000
6,000
4,000
2,000
0
MegaMultiplies Per Second
8
32
64
256
512
Local Work Sizes
1024
16384
66536
1048576
2097152
8388608

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

In viewing the graphs drawing the programs' performance against local work sizes, it appears that the substantially larger arrays (that is, those with 2 and 8 million elements) see a much greater performance jump than those that are 1 million elements and smaller.

This holds true for both Multiply and Multiply-Add.

Likewise, with some variance, the larger local workgroups seem to enjoy a performance increase over their smaller brethren.

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

Because GPUs are designed to handle serialized data, and they do so at the expense of some of the features CPUs enjoy (branching, the stack, pointers, etc.), it stands to reason that their potential will only be visually realized when they're given massive sets of data that is not burdened by conditionals and logic.

Provided the GPU hardware is powerful enough and the tasks its assigned to perform are "embarrassingly parallel," the GPU's performance will only improve as more and more data is handed to it.

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

They are surprisingly similar for the smaller arrays in particular. Depending on work group sizes, even the calculations on larger arrays can be strikingly similar.

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

Provided that the calculations are simple (i.e. not heavily reliant on branching or recursion) and that a programmer doesn't view the GPU as a "CPU, but faster" (which is only true under specific conditions), there are substantial gains in performance to be have – but only if the data being processed is extremely large.