Project 6: OpenCL

By Caleb Knight

knightca@oregonstate.edu

Introduction

This is a technical write-up for observing matrix multiplication using OpenCL. I ran these tests on OSU's Nvidia DGX-2 system. Each of these DGX servers has 16 NVidia Tesla v100 GPUs, 28TB of SSD disk, two 24-core Intel Xeon 8168 Platinum 2.7GHz CPUs, 1.5TB of DDR4-2666 system memory, all running on the CentOS 7 Linux OS.

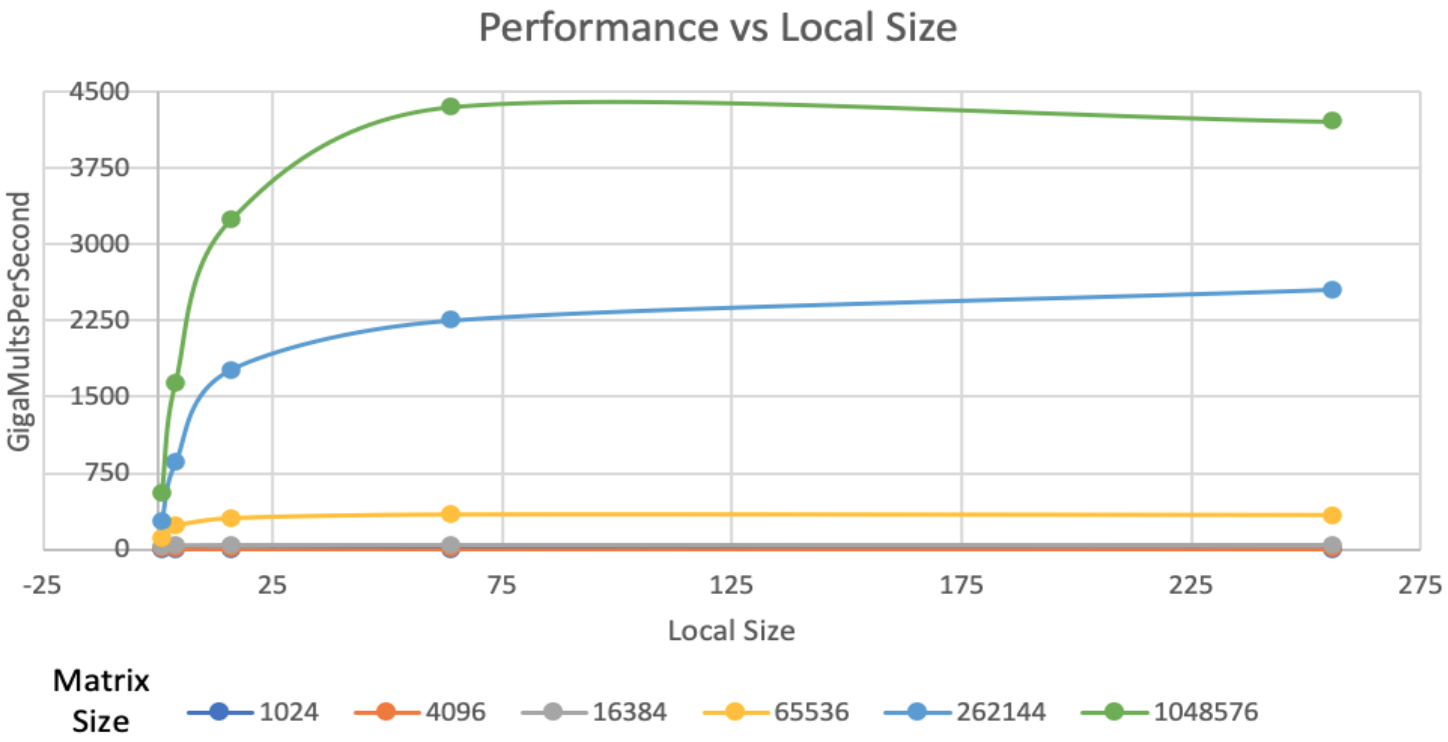
Table

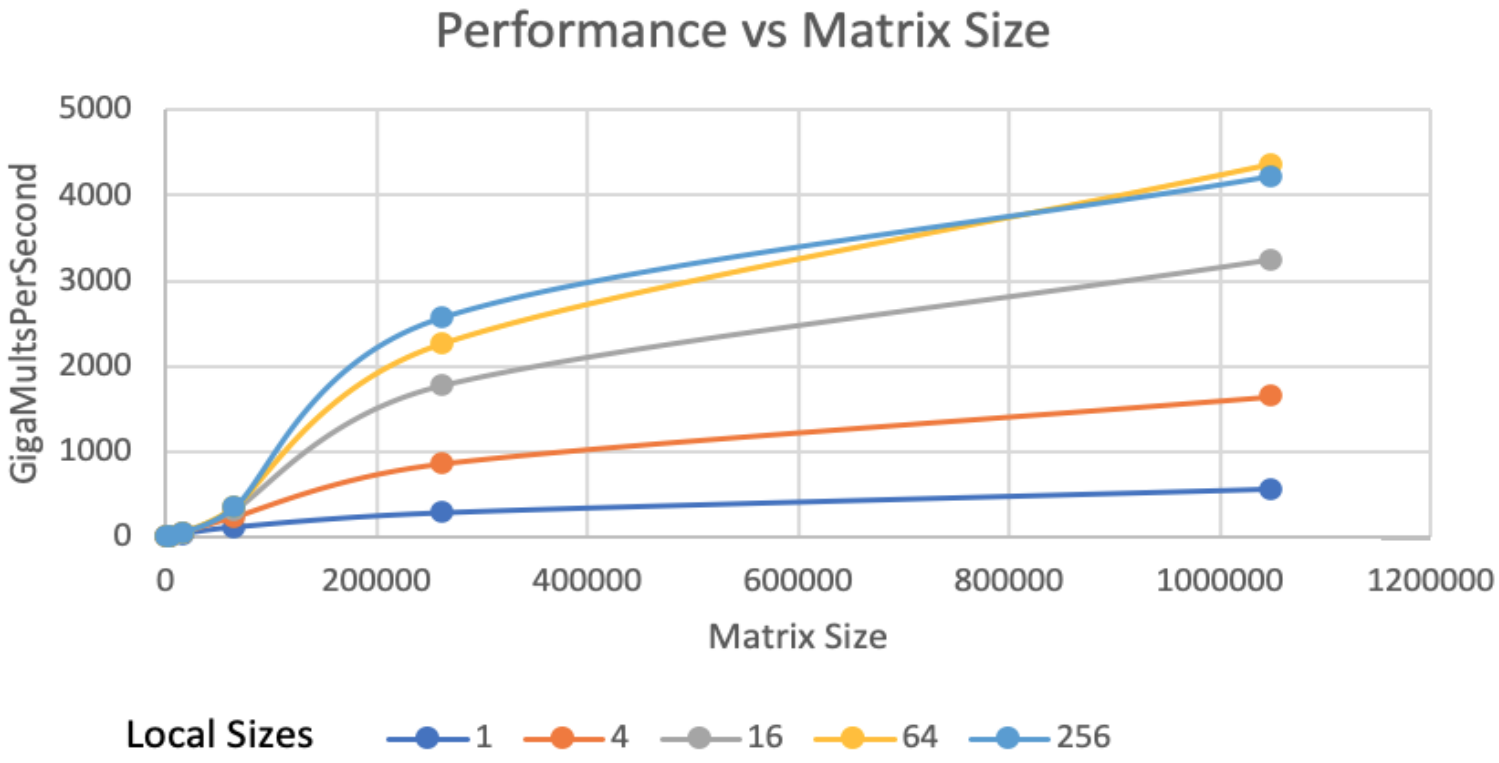
This table below shows the data from the test program. We have the size of the matrix on the left column, the work elements (which is the local size) in the middle, and their performance regarding matrix multiplication in GigMults/Second on the right.

Matrix Size	Work Elements	GigaMultsPerSecond
1024	1	0.59
1024	4	0.67
1024	16	0.65
1024	64	0.68
1024	256	0.65
4096	1	4.77
4096	4	5.97
4096	16	5.12
4096	64	5.19
4096	256	5.32

Matrix Size	Local Size	GigaMultsPerSecond
16384	1	27.75
16384	4	37.45
16384	16	42
16384	64	42.16
16384	256	42.64
65536	1	106.32
65536	4	223.47
65536	16	302.2
65536	64	338.57
65536	256	333.79
262144	1	273.72
262144	4	848.8
262144	16	1762.42
262144	64	2250.36
262144	256	2555.37
1048576	1	549.93
1048576	4	1637.26
1048576	16	3240.17
1048576	64	4356.1
1048576	256	4212.19

Graphs


Performance vs Local Size
GigaMultsPerSecond
4500
3750
3000
2250
1500
750
0
-25
25
75
125
175
225
275
Local Size
Matrix Size
1024
4096
16384
65536
262144
1048576


Performance vs Matrix Size
GigaMultsPerSecond
5000
4000
3000
2000
1000
0
0
200000
400000
600000
800000
1000000
1200000
Matrix Size
Local Sizes
1
4
16
64
256

Observations

For our graphs, we can see an increase in performance in both. This is expected but we will get into more detail.

For the first graph, we see the effect that the work group sizes have on our matrices. This is hard to depict accurately because, as we can see in our table, the initial data is extremely small. In this first graph, we see that all matrices had very low performance with our small work groups. We see the work groups take off after our data point 16, and increase very well up to work group size 64. This is true for all matrices (even if the small ones aren't visible). The reason for this revolves around the same idea as our block sizes or warps from previous projects. A full work group has a size of 32. So, anything under that (1, 4, or 16) isn't maximizing its' efficiency as there is left over space. But, this doesn't mean that 32 is the most efficient. As we can see here, 64 does much better. This is because if the first work group has to go fetch something from memory, we have another work group queued up and ready to go. If we didn't have this, we would have to wait until the first work group had fetched whatever it needed from memory, and started again. Now, after 64, we won't likely see much increase in performance, but rather a plateau. This is because the likelihood of us needing to go to the third, or fourth queued work group is very low. And by the time we get out to 256, this is 8 work groups, we have actually decreased in performance. It just doesn't make sense to have 7 work groups queued and waiting around.

Moving onto the second graph, we see each work group's performance being checked at different matrix sizes. With matrix multiplication, the larger the matrices, the better the performance (most of the time). The reason we know this is because as the size increases, we are given more indexes to process. Now if we did these each one at a time, this may not go well. But, because we use parallelization, we are now processing more matrix indexes over more processors. There may come a time when our matrix is so big though, that the efficiency of this multiplication basically stops compensating for the large matrix. We don't see that in our graph here and when testing, we got errors related to matrix size before we ever plateaued, but theoretically, it is possible. As stated previously, this is usually due to memory storage problems but can also be due to cache inefficiencies when the matrix size exceeds the cache size. Another possibility is that the complexity of the multiplication is more "expensive" than what its payoff is.