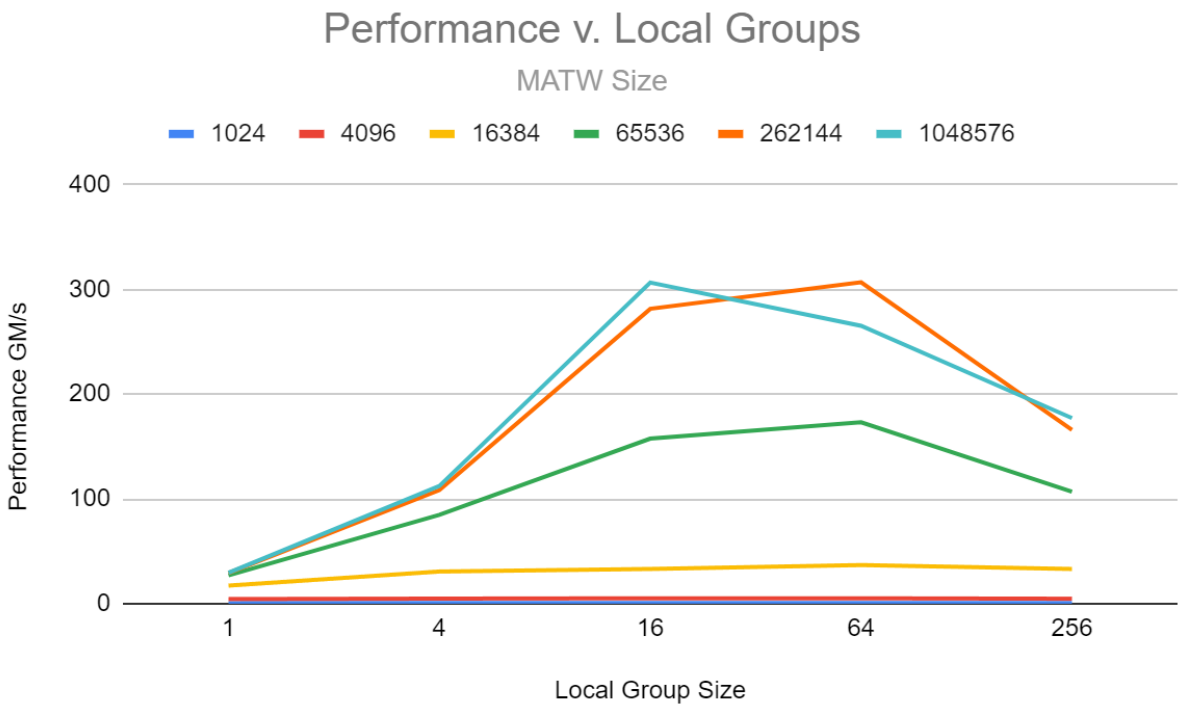
CS 475 Parallel Programming: Project 6 - OpenCL Matrix Mult

Conner Rhea
rheac@oregonstate.edu

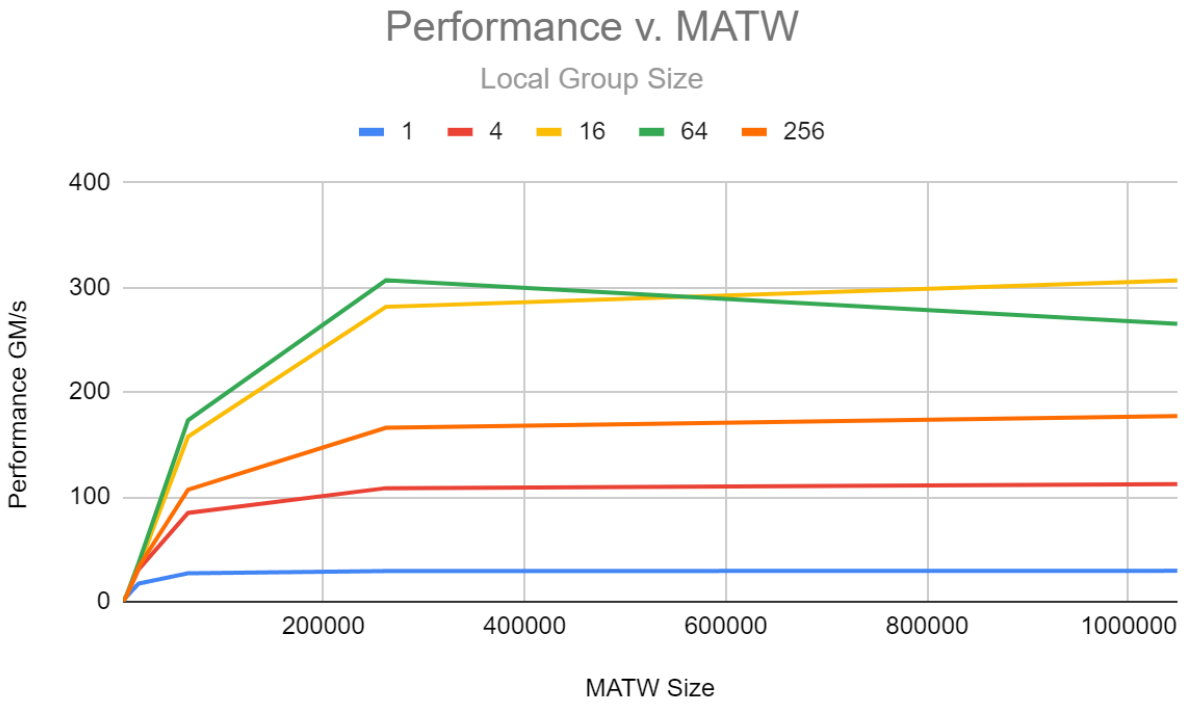
1. I ran the final trial on the DGX System at OSU, after testing on the Rabbit Server. I will include my bash file with my code.
2. Graphs and Tables Created:
   Performance v. Local Group Size

	MATW						
Localgroup	1024	4096	16384	65536	262144	1048576	Grand Total
1	0.58	4.43	17.27	27.12	29.25	29.47	108.12
4	0.65	4.86	30.71	84.75	108.36	112.31	341.64
16	0.66	5.16	33.15	157.54	281.44	306.54	784.49
64	0.66	5.14	36.88	173.11	306.79	265.21	787.79
256	0.64	4.75	33.13	106.8	165.95	177.21	488.48



Performance v. MATW

	Localgroup					
MATW	1	4	16	64	256	Grand Total
1024	0.58	0.65	0.66	0.66	0.64	3.19
4096	4.43	4.86	5.16	5.14	4.75	24.34
16384	17.27	30.71	33.15	36.88	33.13	151.14
65536	27.12	84.75	157.54	173.11	106.8	549.32
262144	29.25	108.36	281.44	306.79	165.95	891.79
1048576	29.47	112.31	306.54	265.21	177.21	890.74



3. When we look at the performance curves, performance goes up as the size gets larger similar to CUDA, however there is a noticeable difference. When group size starts to get large there are actually diminishing returns, which could indicate that there is a portion of the work that cannot be parallelized and may fall off even with large group sizes. Performance v. MATW however scales pretty cleanly as the size of the Matrix increases, which may suggest that similar to the Monte Carlo project GPU computing scales well as the problem size increases.
4. I think there must be an element in this problem that must be non-parallelizable, which explains the reason for Local Group Size to fall off as it gets larger, the system ends up having to wait for the groups to return. However since Matrix Size scaling upwards well makes sense due to the CUDA similarities.