Adam Wright

wrighada@oregonstate.edu

CS-475

Project-6 written commentary

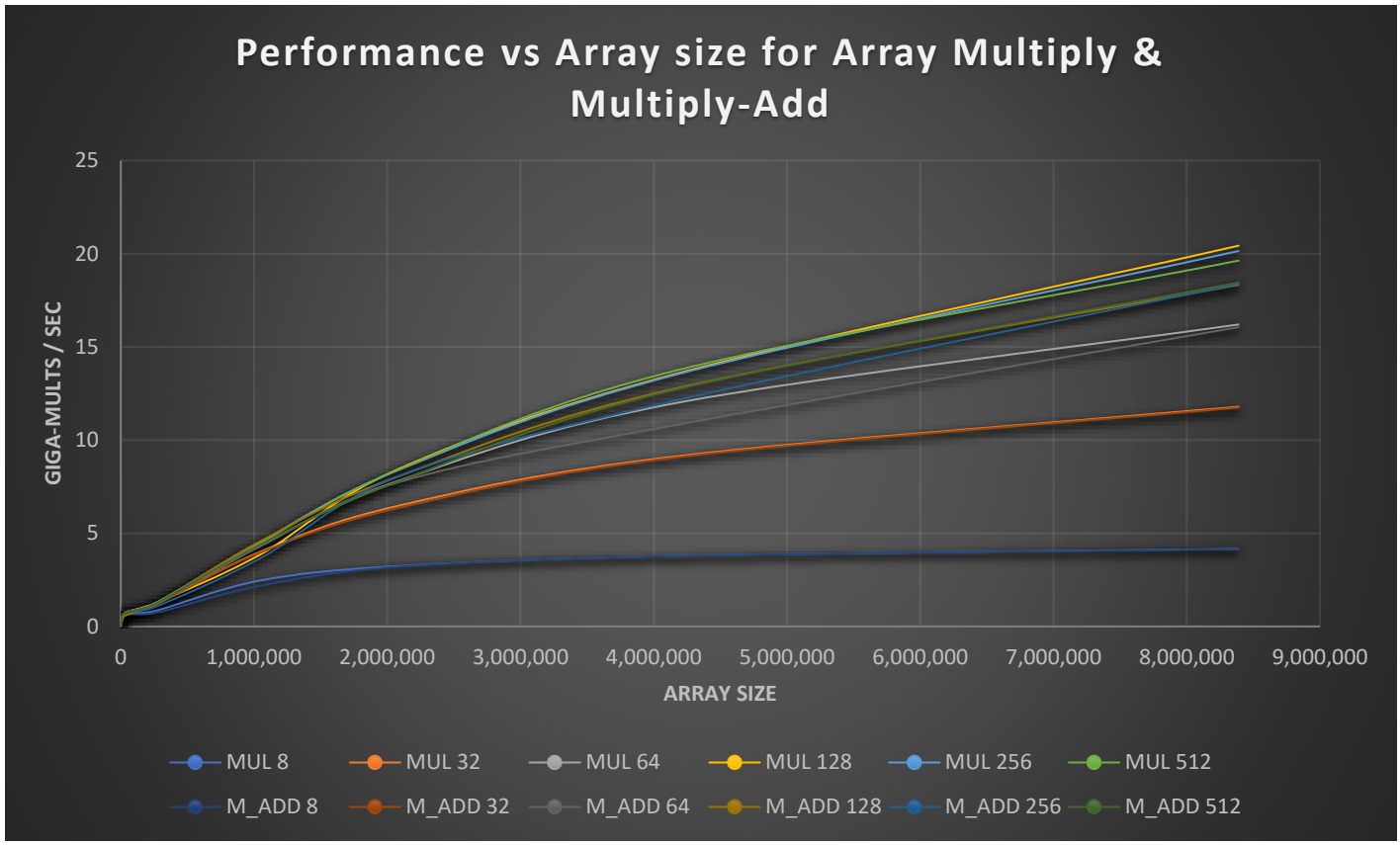
1. What machine did you run this on?

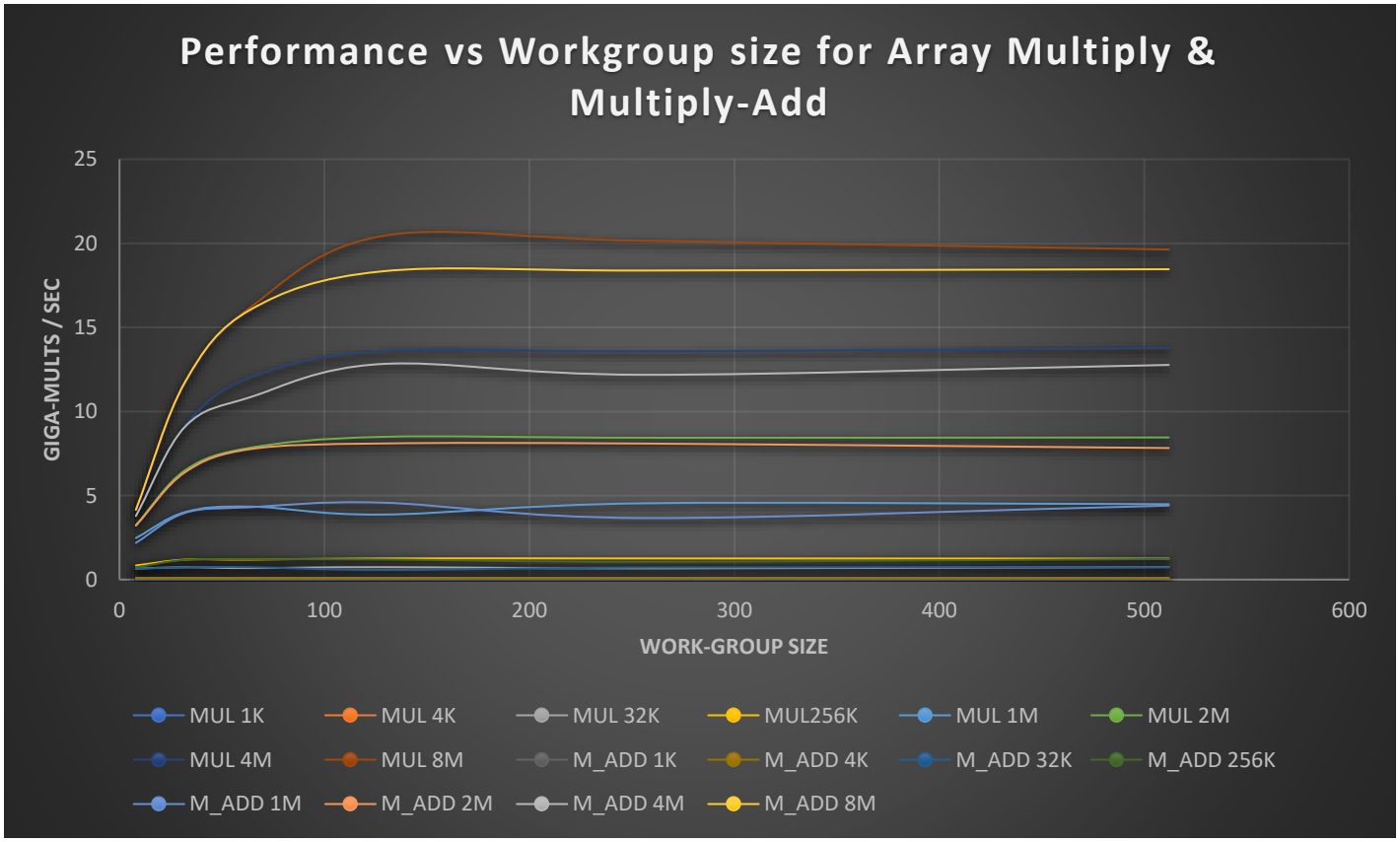
The tests were run on the OSU DGX GPU system.

2. Show the tables and graphs

```
        ********************************************
        **      CUDA Array Multiplication test    **
        ********************************************

           NMB(Array Size) vs. GigaMults / sec.

        1KB      4KB      32KB     256KB    1MB      2MB      4MB      8MB
8       0.022    0.089    0.653    0.843    2.471    3.245    3.788    4.165
32      0.024    0.093    0.736    1.191    4.015    6.502    9.152    11.790
64      0.024    0.091    0.686    1.191    4.346    7.847    12.026   16.202
128     0.023    0.093    0.741    1.259    3.865    8.491    13.609   20.432
256     0.020    0.092    0.668    1.261    4.540    8.436    13.569   20.147
512     0.023    0.091    0.747    1.249    4.485    8.455    13.784   19.631
```

```
        ********************************************
        **        CUDA Array Mult-Add test        **
        ********************************************

           NMB(Array Size) vs. GigaMults / sec.

        1KB      4KB      32KB     256KB    1MB      2MB      4MB      8MB
8       0.020    0.090    0.647    0.732    2.185    3.208    3.784    4.148
32      0.023    0.089    0.733    1.179    3.970    6.385    9.092    11.743
64      0.022    0.079    0.740    1.209    4.305    7.764    10.854   16.058
128     0.023    0.091    0.578    1.200    4.579    8.096    12.822   18.331
256     0.023    0.090    0.716    1.060    3.661    8.091    12.179   18.379
512     0.023    0.092    0.739    1.229    4.401    7.823    12.766   18.457
```


Performance vs Array size for Array Multiply & Multiply-Add
GIGA-MULTS / SEC
25
20
15
10
5
0
0
1,000,000
2,000,000
3,000,000
4,000,000
5,000,000
6,000,000
7,000,000
8,000,000
9,000,000
ARRAY SIZE
MUL 8
MUL 32
MUL 64
MUL 128
MUL 256
MUL 512
M_ADD 8
M_ADD 32
M_ADD 64
M_ADD 128
M_ADD 256
M_ADD 512


Performance vs Workgroup size for Array Multiply & Multiply-Add
GIGA-MULTS / SEC
25
20
15
10
5
0
0
100
200
300
400
500
600
WORK-GROUP SIZE
MUL 1K
MUL 4K
MUL 32K
MUL256K
MUL 1M
MUL 2M
MUL 4M
MUL 8M
M_ADD 1K
M_ADD 4K
M_ADD 32K
M_ADD 256K
M_ADD 1M
M_ADD 2M
M_ADD 4M
M_ADD 8M

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

The performance scales surprisingly reliably as the global array size grows, which shows that even the 8 MB arrays do not saturate the command queue sending or returning from the graphics cards. The growth is less than linear, but for work groups sizes of 128 or greater, the performance grows as the array size grows. It is also clear that the work group size of 128 is perfectly adequate, as the performance is fairly flat between the 128, 256, and 512 work group sizes.

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

It seems that on the DGX system, that a workgroup size of 128 is perfectly efficient and thus there is no gain when moving to larger work groups with more shared memory within the group. The groups smaller than 128 must not allow for enough warps to be swapped in when specific operations block. Also, it is clear that the 8 MB array is not the limit for the DGX system, though it might be on at or beyond the limit another lesser powered system. I believe that this is the case because DGX has powerful modern GPUs in it and there are likely more compute units which could be used.

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

The array multiply and the array multiply and add operations are very close to each other in performance. The Multiply-Add is slightly slower. The array multiply and add begins to clearly lag behind the multiply only operation once the array size reaches 256 KB. Clearly the fused multiply add is very close in performance to just the multiply operation and thus is a very efficient operation.

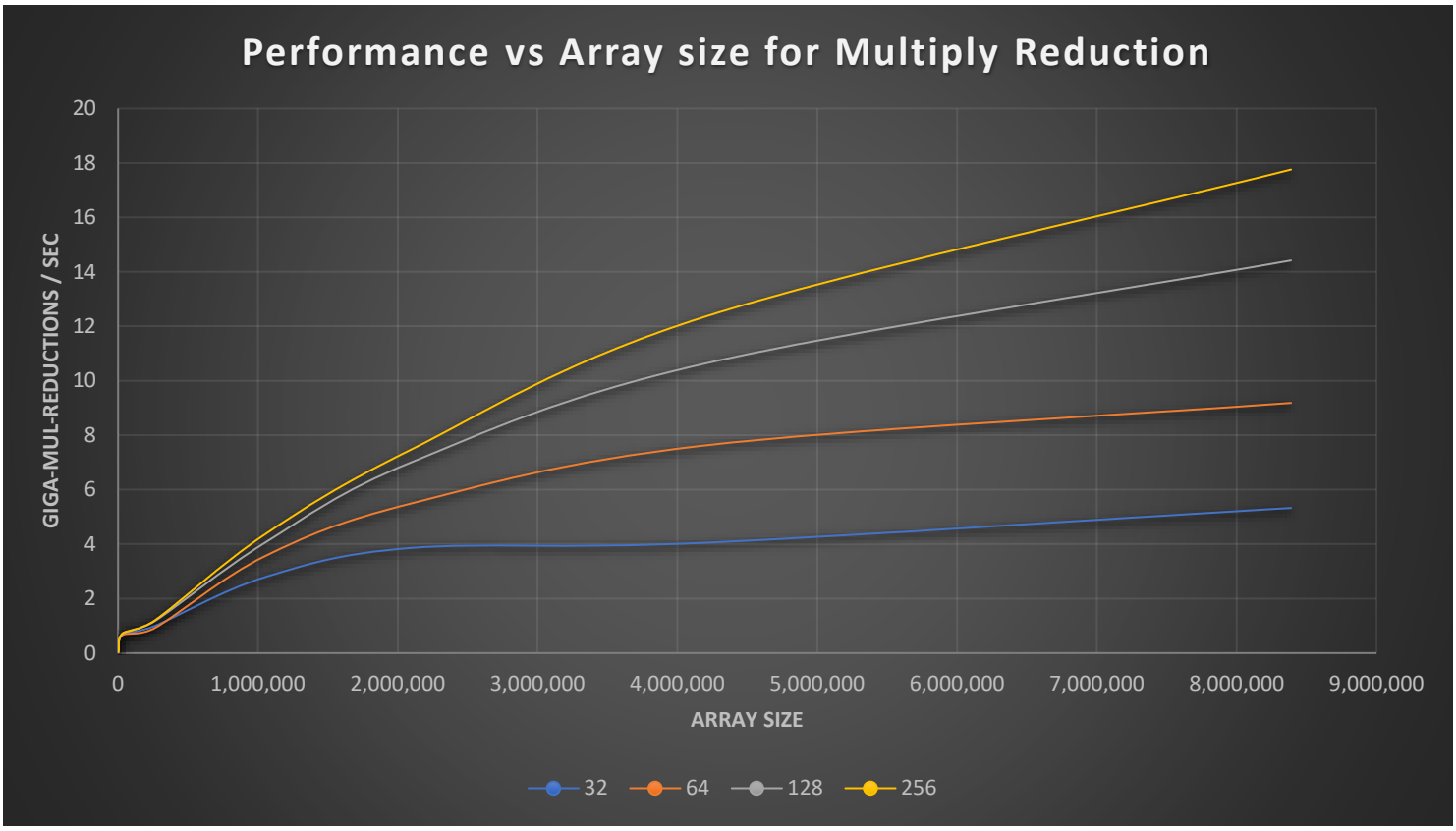
6. What does this mean for the proper use of GPU parallel computing?

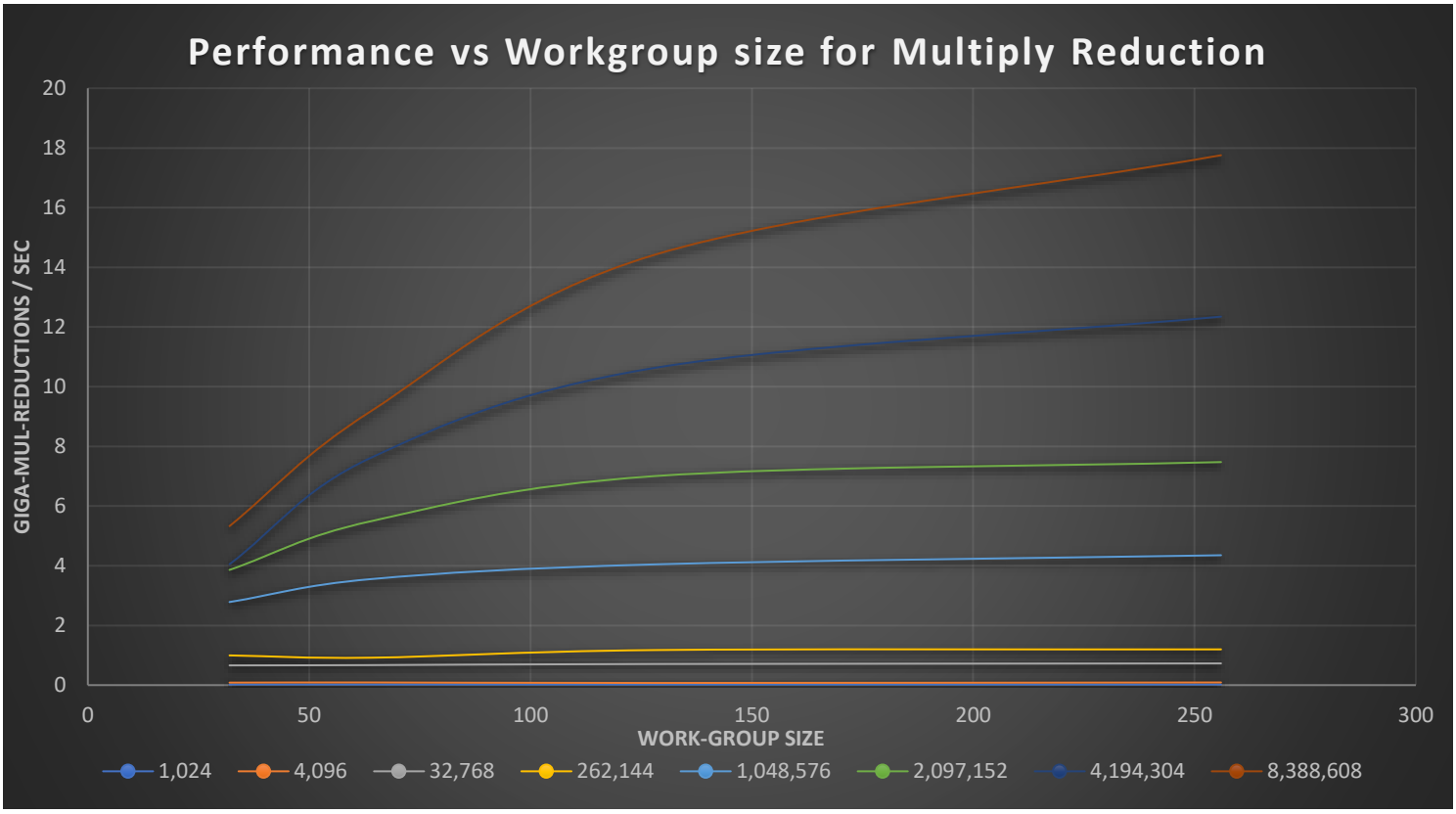
This means that the most performant method of multiplying two arrays or multiply two arrays and adding a value from a third array would be to do this by using GPU parallel computing. The cost of the equpment could be a concern, but this method is capable of doing tens of billions of operations per second, which is very impressive when compared to CPU parallel computing.

Multiply and reduce

1. Show the table and graphs

```
        ********************************************
        **        CUDA Array Reduction test       **
        ********************************************

           NMB(Array Size) vs. GigaMults / sec.
        1KB      4KB      32KB     256KB    1MB      2MB      4MB      8MB
32      0.023    0.085    0.661    0.996    2.785    3.861    4.051    5.329
64      0.023    0.090    0.671    0.919    3.554    5.495    7.623    9.186
128     0.022    0.074    0.708    1.175    4.046    7.005    10.629   14.419
256     0.022    0.090    0.728    1.199    4.353    7.475    12.348   17.757
```


Performance vs Array size for Multiply Reduction
GIGA-MUL-REDUCTIONS / SEC
0
2
4
6
8
10
12
14
16
18
20
0
1,000,000
2,000,000
3,000,000
4,000,000
5,000,000
6,000,000
7,000,000
8,000,000
9,000,000
ARRAY SIZE
32
64
128
256


Performance vs Workgroup size for Multiply Reduction
GIGA-MUL-REDUCTIONS / SEC
0
2
4
6
8
10
12
14
16
18
20
0
50
100
150
200
250
300
WORK-GROUP SIZE
1,024
4,096
32,768
262,144
1,048,576
2,097,152
4,194,304
8,388,608

2. What pattern are you seeing in this performance curve?

The performance scales reliably for Multiplication with reduction, but the work group size has a larger impact on the performance growth. Unlike the multiplication and multiplication with addition tests, the performance contines to grow beyond the 128 work group size. It continues to grow up to at least the 256 work group size, and may contine to grow beyond that. The performance also grows as the array size grows and like the two previous two tests, the growth is less than linear, but it is steady and likely continues beyond the 8 MB bound of our test.

3. Why do you think the pattern looks this way?

The largest portion of the work is being done on the GPU array and thus the only remaing portion to be done serially on the CPU is the final addition of the values in the returned work groups. This allows the GPU to handle the processing of the many values in parallel and then the CPU only needs to do the much smaller final summation with the following data values from the array being pipelined in cache to allow for an efficient final step.

4. What does this mean for the proper use of GPU parallel computing?

This shows that it is very efficient to to put the majority of an program's processing onto a GPU and then to conplete the remainder of the work on the host system. We were told to stop with a work group size of 256 and 8 MB array size, but it would be interesting to see if there was a continued increase in performance with a 512 work group size and a larger array size. This shows that an initial test run of an openCL program should be done to assess the bounds of a particular system before committing to final values for group size and the data set size. This proof of concept could be done along side a CPU only version to prove the value of either solution before moving forward with th final deployed solution.