Project #6
OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce

By Michael Rose

Due: May 27th

Written for: CS475 Spring 2016

Oregon State University

Table of Contents

Code

First.cl

```
kernel
void
ArrayMult( global const float *dA, global const float *dB, global float *dC )
{
        int gid = get_global_id( 0 );
        dC[gid] = dA[gid] * dB[gid];
}
kernel
void
ArrayMultAdd( global const float *dA, global const float *dB, global float
*dC ,global float *dD )
{
        int gid = get_global_id( 0 );

        dD[gid] = dA[gid] * dB[gid] +dC[gid];
}
kernel
void
ArrayMultRedux(global const float *dA, global const float *dB,local float
*prods, global float *dC)
{
        int gid = get_global_id( 0 );
        int numItems = get_local_size(0);
        int tnum = get_local_id(0);
        int wgNum = get_group_id(0);
        prods[tnum] = dA[gid] * dB[gid];
        // all threads execute this code simultaneously:
        for( int offset = 1; offset < numItems; offset *= 2 )
        {
                int mask = 2*offset - 1;
                barrier( CLK_LOCAL_MEM_FENCE );  // wait for completion
                if(  ( tnum & mask ) == 0 )
                {
                        prods[ tnum ] += prods[ tnum + offset ];
                }
        }
        barrier( CLK_LOCAL_MEM_FENCE );
        if( tnum == 0 )
                dC[ wgNum ] = prods[ 0 ];
}
```

What I did

I ran my code on my windows machine. It is a laptop with an integrated card. I modified ArrayMult for ArrayMultAdd by adding an extra global float array input and adding that to the function. For ArrayMultRedux, I took the code from the slides. For the c++ file, I copied the main() code into a function called mult(). For both ArrayMultAdd and ArrayMultRedux, I copied that mult() function and modified it for the proper inputs and return values.

The 3 functions take a local size and a global size input. This allows me to loop over the functions and create output for each value all in the same run.

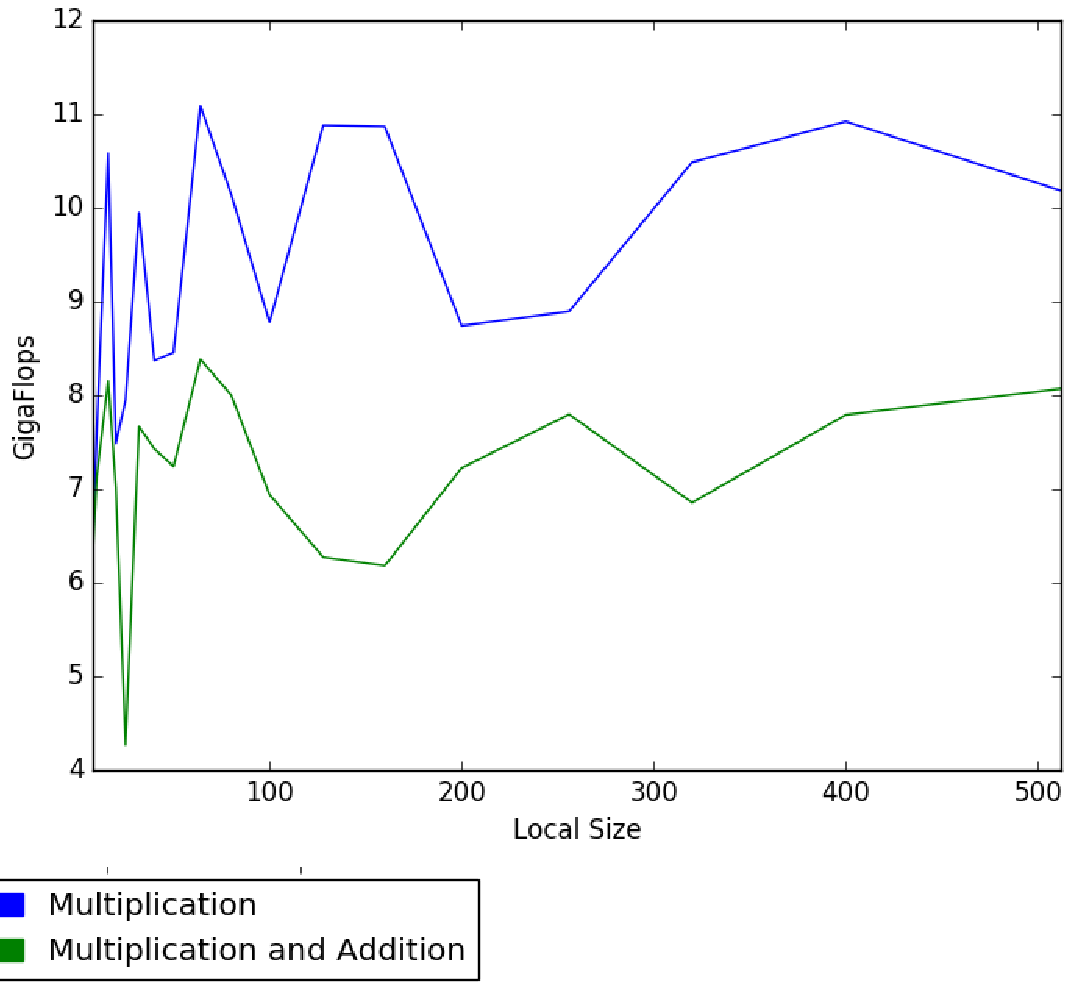
For local size, I commented out the actual function call of the cl function and just tested what values my computer could handle for local size. Once I got all the local sizes that my computer could handle, I uncommented the code and ran it on an array that only included the local sizes that worked.

I used 2 loops, one for varying global size and 1 for varying local size. Before I call the first loop, I create a file called global.csv and output the results to the file. For the 2nd loop, I created a file called local.csv and output the results to that file.

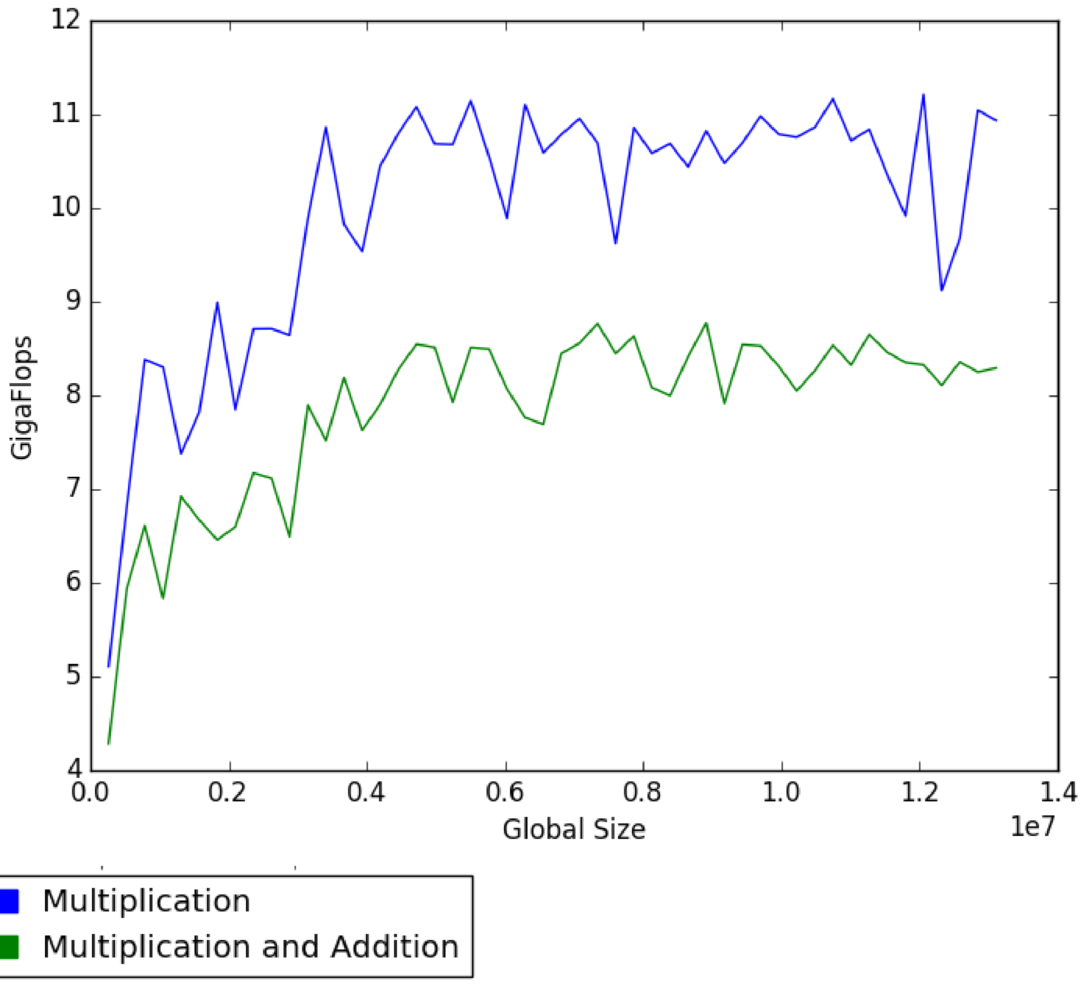
I modified the plot.py code that I wrote earlier this term to handle creating the plots. The reason why I output files is because sometimes the legend doesn't look good on the graph, so I output that as a 2nd file and just put both those files in writeup.

Results

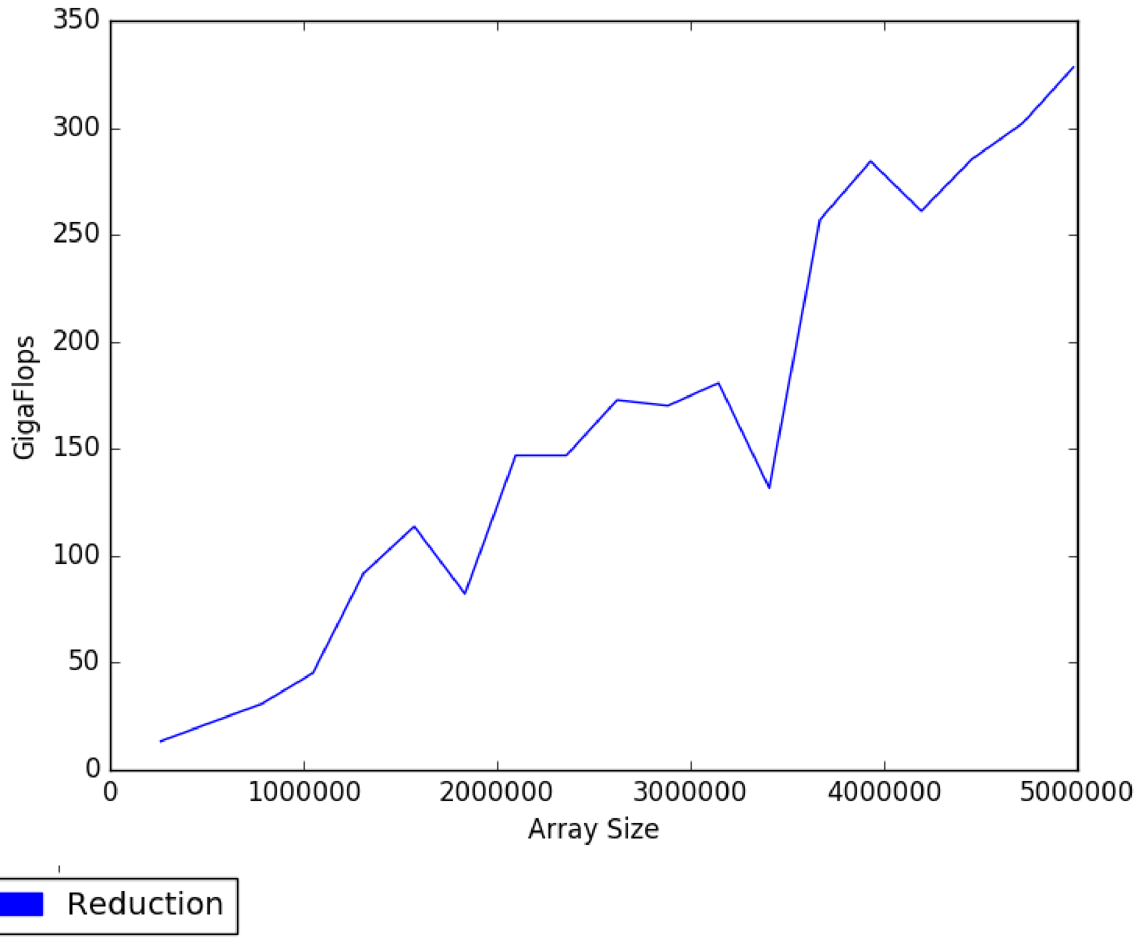
Graph: Local Size



Graph: Global Size



Graph: Reduction



Analysis

Notes Before I Break Down Each Graph

I was getting some really sporadic results. I could run the same code multiple times without anything else running. Sometimes I would get under 1 gigaflop of performance for every single function no matter the local and global sizes. Reduction was even more sporadic. Sometimes I would have huge drops in performance and sometimes I would have huge spikes even when the multiply and multiply-add functions were not doing that.

Local Size

I'm not sure exactly what happened with local size. I expected it to increase in gigaflops as the local size increased. I believe that part of the reason why I didn't see this increase is because Intel tries to optimize the code that is being run in visual studio to keep the chip cool. I'm guessing that my computer was running extra background processes because I ran all these after minimizing all the processes I could. Partway through the runs, the screen did go to sleep which could help explain the stability later on in the graph.

Global Size

The 2nd half of the graph is more what I expected. The global size should affect total runtime but not the number of calculations per second. I believe that the reason the first half is growing is because the GPU is not running at the maximum capacity and around 5 million global size, my computer hits that mark.

Reduction

I believe that my computer didn't hit its maximum performance for reduction on this graph. The reason why this keeps going up and is higher than multiply is because it isn't getting an extra array passed in that needs to be returned. Instead of returning an array of values, it is just returning a single value.

This reduction shows that the proper use of the GPU is to do any reduction on the on GPU rather than pass the data back to the CPU to do reduction. This is due to the time it takes the buffers to move data back and forth. The less use of buffers the faster the program will run. This is similar to the cache hits vs. cache misses.