

# Project #6

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

120 Points

Due: June 2

### Introduction

There are many problems in scientific computing where you want to do arithmetic on multiple arrays of numbers (matrix manipulation, Fourier transformation, convolution, etc.). This project is in two parts:

1. Multiply two arrays together using OpenCL:  
 $D[gid] = A[gid] * B[gid];$   
Benchmark it against both input array size (i.e., the global work size) and the local work size (i.e., number of work-items per work-group).
2. Multiply two arrays together and add a third using OpenCL:  
 $D[gid] = A[gid] * B[gid] + C[gid];$   
Benchmark it against both input array size (i.e., the global work size) and the local work size (i.e., number of work-items per work-group).
3. Perform the same array multiply as in #1, but this time with a reduction:  
 $Sum = \text{summation}\{ A[:] * B[:] \};$   
Benchmark that against input array size (i.e., the global work size). You can pick a local work size and hold that constant.

### Requirements:

**First, work on the Array Multiply and the Array Multiply-Add portions:**

1. Start with the [first.cpp](#) and [first.cl](#) files. That code already does array multiplication for one particular combination of global work size and local work size.
2. Make this all work for global work sizes in (at least) the range 1K to 8M, and local work sizes in (at least) the range 8 to 512, or up to the maximum work-group size allowed by your system. How you do this is up to you. Use enough values in those ranges to make good graphs.

3. Use performance units that make sense. Jane Parallel used "MegaMultiplies Per Second" and "MegaMultiply-Adds Per Second".
4. Make two graphs:
  1. Multiply and Multiply-Add performance versus Global Work Size, with a series of colored Constant-Local-Work-Size curves
  2. Multiply and Multiply-Add performance versus Local Work Size, with a series of colored Constant-Global-Work-Size curves
5. Your commentary PDF should tell:
  1. What machine you ran this on
  2. Show the tables and graphs
  3. What patterns are you seeing in the performance curves?
  4. Why do you think the patterns look this way?
  5. What is the performance difference between doing a Multiply and doing a Multiply-Add?
  6. What does that mean for the proper use of GPU parallel computing?

**Then, write another version of the code that turns it into a Multiply+Reduce application.**

6. Note that this will ultimately compute just a single floating point scalar value.
7. Produce the product array on the GPU, and then do the reduction on it from the same kernel.
8. Return an array, the same size as the number of work-groups. Each element of the array will have the sum from all the items in one work-group. Add up the elements of the array yourself.
9. Try at last 3 different local work sizes, more if you want. Make it no smaller than 32. Make it no larger than 256.
10. Vary the size of the input array from 1K to 8M.
11. Plot another graph showing Multiply-reduction performance versus Input Array Size.
12. Use performance units that make sense. Jane Parallel used "MegaMultiply-Reductions Per Second".
13. To your PDF write-up add:
  1. Show this table and graph
  2. What pattern are you seeing in this performance curve?
  3. Why do you think the pattern looks this way?
  4. What does that mean for the proper use of GPU parallel computing?