CS 475/575 -- Spring Quarter 2021

Project #6

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

100 Points

Due: May 30

This page was last updated: April 22, 2021

Note: The flip machines do not have GPU cards in them, so OpenCL will not run there. If your own system has a GPU, you can use that. You can also use the DGX machine, but please be good about sharing it.

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 three 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 = 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.

Some Help

Here is a Visual Studio solution for this,

I have updated the PrintInfo code into a function. You can get it here. It contains a function called PrintOpenclInfo( ). It also contains a function called SelectOpenclDevice( ) which selects what it thinks is the best system to run your OpenCL code on. Feel free to copy and paste these into your code.

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. **Helpful Hint:** The Array Multiply and the Array Multiply-Add can really be the same program. Write one single program that creates the 4 arrays. Pass A, B, and C into OpenCL, and return D. Then all you have to do between the Multiply and Multiply-Add tests is change one line in the .cl file.

3. 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.

4. Use performance units that make sense. Joe Parallel used "MegaMultiplies Per Second" and "MegaMultiply-Adds Per Second".

5. Make two graphs:
    1. Multiply and Multiply-Add performance versus Global Dataset 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-Dataset-Size curves

6. 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.

7. Note that this will ultimately compute just a single floating point scalar value.

8. Produce the product array on the GPU, and then do the reduction on it from the same kernel.

9. 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.

10. Try at last 3 different local work sizes, more if you want. Make it no smaller than 32. Make it no larger than 256.

11. Vary the size of the input array from 1K to 8M.

12. Plot another graph showing Multiply-reduction performance versus Input Array Size.

13. Use performance units that make sense. Jane Parallel used "MegaMultiply-Reductions Per Second".

14. 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?

Running OpenCL in Linux

First, you will need the following files:

1. cl.h
2. cl_platform.h

If you are on *rabbit* or the *DGX System*, compile, link, and run like this:

g++ -o printinfo printinfo.cpp /usr/local/apps/cuda/10.1/lib64/libOpenCL.so.1.1 -lm -fopenmp
./printinfo

If you are on your own system, change the library reference to whatever path your system has the library in.

Getting the right platform and device number:

Many of you now have multiple OpenCL-compatible platforms with multiple devices in your own systems.

I have updated the PrintInfo code into a function. You can get it here. It contains a function called PrintOpenclInfo( ). It also contains a function called SelectOpenclDevice( ) which selects what it thinks is the best system to run your OpenCL code on. Feel free to copy and paste these into your code.

Grading:

Feature	Points
Multiply table and graphs	20
Multiply-Add table and graphs	20
Multiply and Multiply-Add Commentary	20
Reduction tables and graphs	20
Reduction Commentary	20
Potential Total	**100**