Lyon Kee

[keel@oregonstate.edu](mailto:keel@oregonstate.edu)

Project #6: [OpenCL](http://cs.oregonstate.edu/~mjb/cs575/Projects/proj06.html)

**Key snippets of code**

A white background with black text

Description automatically generated

1. Program header

A computer screen shot of a program code

Description automatically generated

We observe that cl.h and cl\_platform.h is added which is needed for OpenCL, OpenMP for time measurement, and the rest are the standard libraries for operations.

1. Allocate the host memory buffers

A screen shot of a computer program

Description automatically generated

Before this, we check if kernel file is readable and also select the best OpenCL device. We read in data and store it into host device arrays of hX and hY.

1. Create an OpenCL context

A screen shot of a computer code

Description automatically generated

Created an OpenCL context and set it to ` cl\_context context` and check for CL\_SUCCESS. Check for CL\_SUCCESS before we proceed.

1. Create an OpenCL command queue

A screenshot of a computer program

Description automatically generated

Create a cl\_command\_queue to store commands for OpenCL to quickly grab commands to execute. Check for CL\_SUCCESS before we proceed.

1. Allocate the device memory buffers

A screen shot of a computer program

Description automatically generated

Used clCreateBuffer to allocate memory on the OpenCL device to allow the device to read and write to when it is doing computation. Check for CL\_SUCCESS before we proceed.

1. Write the data from the host buffers to the device buffers

A screen shot of a computer code

Description automatically generated

Writing data loaded from file -> host memory to device memory and waiting for it to finish writing.

1. Read the kernel code from a file

A black screen with colorful text

Description automatically generated

Reading our OpenCL code to create a kernel program

1. Compile and link the kernel code

A screen shot of a computer program code

Description automatically generated

Compile and link the kernel code that is loaded to a string in step 7

1. Create the kernel object

A screen shot of a computer program

Description automatically generated

Create a kernel from the compiled and linked program in step 8.

1. Setup the arguments to the kernel object

A screen shot of a computer program

Description automatically generated

Set kernel arguments which should align with the kernel program arguments below:  
A screenshot of a computer program

Description automatically generated

1. Enqueue the kernel object for execution

A screen shot of a computer code

Description automatically generated

Enqueue the Kernel and time the execution after waiting for it to finish running.

1. Read the results buffer back from the device to the host

A screenshot of a computer

Description automatically generated

Read the output of the Kernel back to host device to compute sum.

1. Clean everything up

**A screen shot of a computer program

Description automatically generated**

Clean up created objects relating to the OpenCL device.

**Tables of data**

Columns: Array Size

Rows: Work Elements

Values: megaPointsProcessedPerSecond

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sum of MegaPointsProcessedPerSecond** | **Array Size** |  |  |  |  |  |
| **Work Elements** | **4096** | **16384** | **65536** | **262144** | **1048576** | **4194304** |
| 8 | 108.68 | 402.71 | 1301.2 | 851.84 | 1553.27 | 1599.62 |
| 32 | 102.55 | 411.84 | 1567.16 | 997.16 | 2084.37 | 6297.17 |
| 64 | 108.28 | 443.89 | 1657.76 | 1011.41 | 2162 | 7150.43 |
| 96 | 241.49 | 979.85 | 3793.46 | 15259.4 | 60221.43 | 246152.14 |
| 128 | 108.87 | 426.04 | 1725.4 | 1016.4 | 2207.97 | 6726.27 |
| 256 | 108.27 | 433.16 | 1656.98 | 44.35 | 2220.25 | 7087.84 |
| 512 | 107.01 | 444.34 | 1669.25 | 1014.2 | 2042.92 | 7129.99 |

**Graphs of data**

Graph of performance versus DATASIZE with colored curves of constant LOCALSIZE In the graph above, we observe that as the Array Size increases, the performance increases as well, although we do observe that given specific “Work Elements” or constant local sizes, the performance peaks and stays constant even when we increase DATASIZE, this is likely the peak performance of the device with data being abundant so there is little inefficiency.

Graph of performance versus LOCALSIZE with colored curves of constant DATASIZE

The graph above shows that there is a sweet spot in the number of work elements, and a number above or below it will yield a similar performance. We observe that the best performance is at 96 work elements, from lecture it seems that each “gripper” has 32 work elements, and at 64, we have 2 sets which allows the scheduler to swap out memory when it is performing tasks that require waiting and it just seems like in this system, 96 was the perfect number which yielded the best performance.

**PDF Commentary**

1. What machine you ran this on

I ran this on the one Tesla V100 of the A100s that the class has access to, therefore I get dedicated CPU&GPU time and it is very reliable.

1. Show the table and the graphs

Shown in the above sections.

1. What patterns are you seeing in the performance curves? What difference does the size of data make? What difference does the size of each work-group make?

Graph analysis and evaluations are shown in the above sections.