Jeremy Prater

CS 475

Project 6 – OpenCL and Reduction

**Target Machine:**

I ran the test on my home office PC. It is a Core 2 Quad Q6600 with 8GB DDR2, a Geforce GTX 980 and a Geforce GTX 460. All tests were executed with marginal success. The results from multiplication and multiplication and addition were consistent with cache and prefetching models from previous projects. My multiplication and addition with reduction solution however had several issues with GPU crashes (a few which required an entire workstation reboot), and segmentation faults. I was diligent with allocation and releasing of memory buffers and shared objects between the CPU and GPU. After staying up very late and turning this in late, I believe the errors I experienced in my application were due to running Arch linux with open source NVidia drivers that may not necessarily have been hardware qualified on the video card in my desktop.

Example Error:

-> Elements: 131072 -> Local work size: 8 -> Num work groups: 16384 -> OPMode: 0  
-> Elements: 131072 -> Local work size: 8 -> Num work groups: 16384 -> OPMode: 1

clEnqueueNDRangeKernel failed: -5

-> Elements: 131072 -> Local work size: 8 -> Num work groups: 16384 -> OPMode: 2  
(ERROR)-(Exit Code 255)-(Unknown error code)  
  
(Fri Jun-2 5:08:10pm)-(CPU 1.42%:Free MB 3086)-(prater: ~/src/osu/CS\_475/project6)-(48K:8)

Error -5 indicate the follow issue:

|  |  |  |  |
| --- | --- | --- | --- |
| -5 | CL\_OUT\_OF\_RESOURCES |  | if there is a failure to allocate resources required by the OpenCL implementation on the device. |

**Table and graph:**

OP Mode is the current operation mode of the application. It is an enumeration of 0 being multiply, 1 being multiply and add, and 2 being multiply, add, and reduce. A local size of 32 was used because my video card could crash on using a larger size while using reduction. This was picked to normalize the results.

Table 1 : Average GPU Speeds. (Green is faster)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OP\_MODE | NUM\_ELEMENTS | LOCAL\_SIZE | NUM\_WORK\_GROUPS | megaMultsSecBest | megaMultsSecAvg |
| 0 | 1024 | 32 | 32 | 44.933957 | 30.829259 |
| 0 | 2048 | 32 | 64 | 94.356139 | 66.385952 |
| 0 | 4096 | 32 | 128 | 190.263845 | 158.315426 |
| 0 | 8192 | 32 | 256 | 358.716133 | 301.966918 |
| 0 | 16384 | 32 | 512 | 750.045779 | 613.961773 |
| 0 | 32768 | 32 | 1024 | 1493.936357 | 1185.768467 |
| 0 | 65536 | 32 | 2048 | 2657.152202 | 2125.239152 |
| 1 | 1024 | 32 | 32 | 45.209714 | 35.076782 |
| 1 | 2048 | 32 | 64 | 89.514397 | 72.06142 |
| 1 | 4096 | 32 | 128 | 181.648851 | 145.147346 |
| 1 | 8192 | 32 | 256 | 345.217011 | 281.938326 |
| 1 | 16384 | 32 | 512 | 694.502162 | 489.043974 |
| 1 | 32768 | 32 | 1024 | 1427.302027 | 1124.274775 |
| 1 | 65536 | 32 | 2048 | 2497.275433 | 2061.917957 |
| 2 | 1024 | 32 | 32 | 44.722016 | 14.130004 |
| 2 | 2048 | 32 | 64 | 93.627135 | 29.760996 |
| 2 | 4096 | 32 | 128 | 183.73481 | 59.083087 |
| 2 | 8192 | 32 | 256 | 264.821884 | 89.217063 |
| 2 | 16384 | 32 | 512 | 707.88509 | 235.407373 |
| 2 | 32768 | 32 | 1024 | 1325.566413 | 449.259506 |
| 2 | 65536 | 32 | 2048 | 1878.145276 | 782.106641 |

If reduction is not preformed, larger local gpu thread sizes can be used and different results occur. This could be caused by the size of the compiled code being loaded into the gpu kernel. Cross compute unit communication can also play a role in maximum computational throughput.

Table 2 : High data transfer rates

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OP\_MODE | NUM\_ELEMENTS | LOCAL\_SIZE | NUM\_WORK\_GROUPS | megaMultsSecBest | megaMultsSecAvg |
| 0 | 131072 | 8 | 16384 | 2294.677878 | 1250.707307 |
| 0 | 131072 | 8 | 16384 | 2282.847973 | 2010.362227 |
| 0 | 65536 | 64 | 1024 | 2867.719686 | 2098.595838 |
| 0 | 65536 | 512 | 128 | 2767.801369 | 2144.565775 |
| 0 | 65536 | 256 | 256 | 2726.009635 | 2124.316037 |
| 0 | 65536 | 128 | 512 | 2710.339038 | 2146.750484 |
| 0 | 65536 | 64 | 1024 | 2706.08624 | 2125.190927 |
| 0 | 65536 | 128 | 512 | 2676.686686 | 2165.777695 |

**Trend Analysis:**

As data buffer size increases, the CPU can feed the GPU data faster due to prefetching on the CPU side. The proper balance between CPU and GPU processing is an important thing to remember, adding too many logical branches inside of an OpenCL kernel can cause serious performance penalties. Simple bit masks and Boolean operators can be used with ease, but recursive or iterative functions could possibly be implemented on the CPU to better utilize the SIMD nature of GPU cores.

**Responsible use of GPU resources:**

Using GPU processing for parallel data analysis, 3d translation, rotation, or projection functions with large data sets is an ideal situation for this type of hardware. Ensuring that shared memory resources between the CPU and GPU which are either be reused multiple times, or freed correctly after being used is critical for program stability. Memory leaks can quickly consume limited GPU ram. It can be quite difficult to debug for remote environments where little information is accessible.