Report 6

Introduction to CUDA and OpenCL

OpenCL Infrastructure.

Dariusz Biela

Karol Sawicki

1. Introduction

The seventh laboratory classes topic was OpenCL. We were trying to understand the C and C++ ways to create GPU kernels and comparing it with our first CUDA samples.

We had to try three exercises. Compare C and C++ DeviceInfo code with the CUDA deviceQuery code from the first lab classes. Investigate performance of pure C and C++ kernels and the way how the kernels are prepared. The last one was to prepare code for cascade vector addition.

2. Calculations and measurements

2.1. Device Info

Our First step was to investigate how we could know about GPUs in our system. The code was rather simple and the result of it was much simpler and smaller than the CUDA Device Query result.

Results from C and C++ codes were the same. Differences could have been seen only in the source code.

```
Number of OpenCL plaforms: 1

-------------------------
Platform: NVIDIA CUDA
        Vendor:  NVIDIA Corporation
        Version: OpenCL 1.2 CUDA 10.1.105

        Number of devices: 2
        ------------------------
                Name: GeForce RTX 2060
                Version: OpenCL C 1.2
                Max. Compute Units: 30
                Local Memory Size: 48 KB
                Global Memory Size: 5904 MB
                Max Alloc Size: 1476 MB
                Max Work-group Total Size: 1024
                Max Work-group Dims: (1024 1024 64)
```

Picture 1 Part of program result for RTX 2060.

2.2. Vector Addition Code

Our second task was to compare vector addition OpenCL code in C and C++. As an addition we were encouraged to compare efficiency in OpenCL and CUDA.

To do our comparisons we used already prepared code. We made slightly modifications, just to measure time of adding more vectors of variable length.

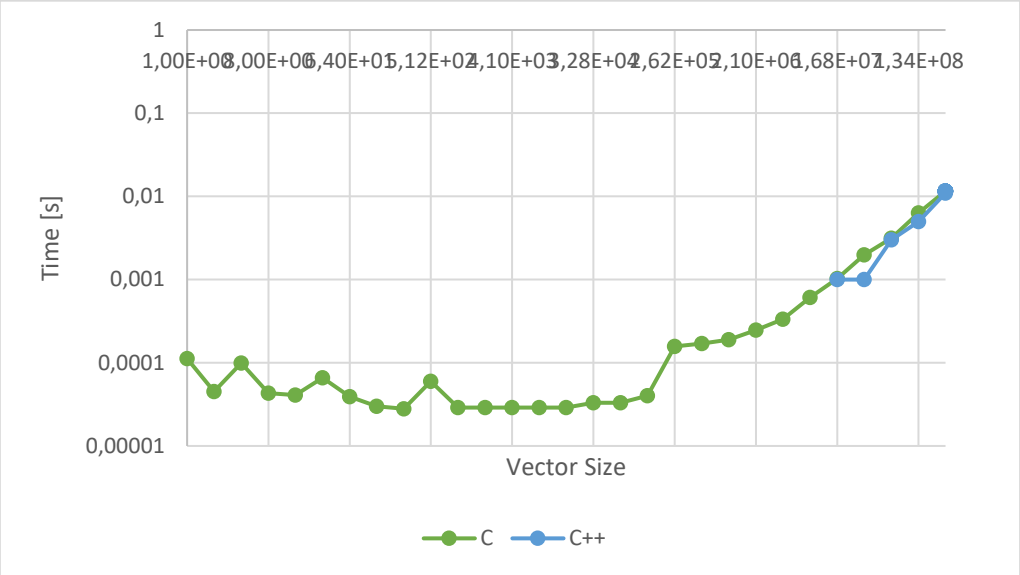

Figure 1 Comparison of adding vectors in C OpenCL and C++ OpenCL.

At the *Figure 1* we could see that our measurements for C++ were not good, in that case we decided to change the way of measuring time, we just changed the default time measure code to C++ chrono high_resolution_clock.

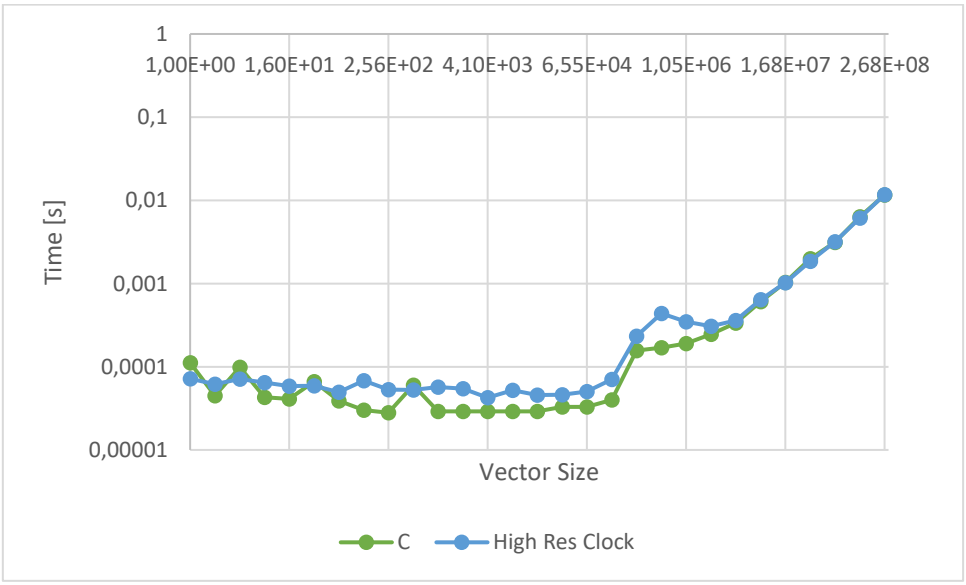

Figure 2 Time measurements with C++ chrono library.

The *Figure 2* shows that C++ code is comparable to the pure C code. For smaller vectors C code may be slightly faster, but the way of preparing kernel code in this language is really weird and we do not think that it would be worth it, because the result difference is in the really small margin of error. For the bigger vectors we could see that there is no difference at all.

After that we prepared simple CUDA kernel and without further optimization we invoked it with usage of managed memory.

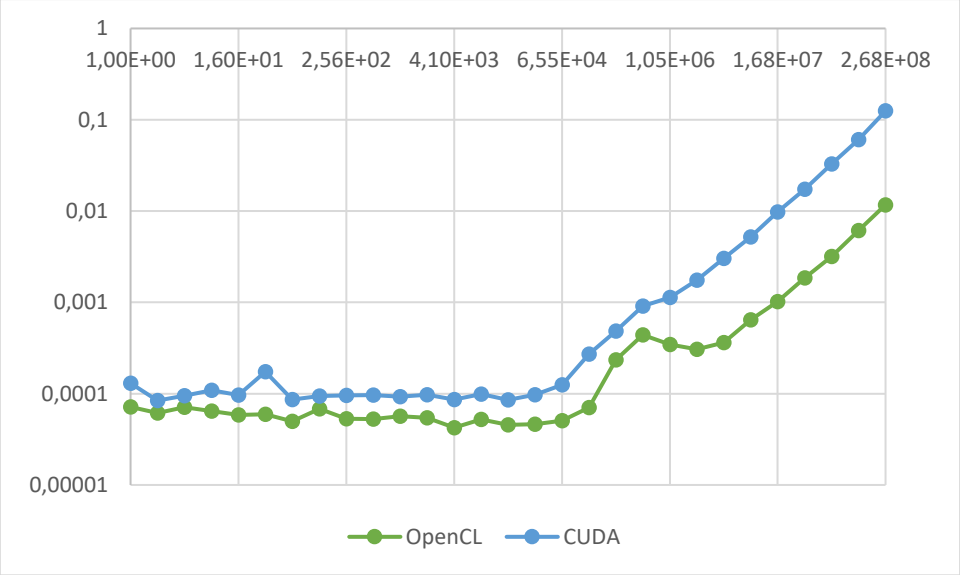

Figure 3 C++ OpenCL comparison with CUDA C++.

Figure 3 shows us that without any optimization CUDA kernel was slightly slower, the difference was not big, but further investigation probably can change results.

2.3. Cascade Addition.

The last task was to modify C++ OpenCL code. We had to write cascade addition of more vectors and then verify the results and compare code complexity to the C one.

Modifications were really simple, we just added more vectors to the host and device. By changing OpenCL *CL_MEM_WRITE_ONLY* memory flag to the *CL_MEM_READ_WRITE* we enabled possibility of not only writing, but reading memory by GPU too. At the end we just added more kernel executions and, in traditional CPU loop, checked results.

3. Conclusions

The way of handling OpenCL kernel in the C is really weird and inconvenient. Kernel should be prepared in the one string and preprocessed by specific functions to be executed at the end. C++ way of handling OpenCL is much more cleaner. It seems more simple and programmer friendly.

The Device Info code is simple an useful, the result provides us information about number of devices in the system, their names, available global memory, compute units and work group dimensions. The result is not as detailed as CUDA Device Query, but it should not be as surprise, because CUDA code is supplied by NVIDIA itself.

C and C++ codes in OpenCL are really similar in the case of performance, but we should take care about the way of measuring the execution time, because not every tool can be as accurate. When the C code is not faster than C++ we think that the C++ version of the OpenCL implementation is way better, because of its simplicity and convenience.

Unoptimized CUDA code is noticeably slower than the OpenCL code that we were investigating, especially for the bigger vectors, but we should remember that CUDA gave us many ways of optimizing code.