Discuss, Relfect and Contrast

In this exercise, we're going to briefly discuss the usability of OpenCL with our own experiences, and its performance versus CUDA introduced by Nvidia on some simple test cases in past assignments.

Usability of OpenCL

In assignment II and III, we were required to utilize CUDA toolkit provided by Nvidia, and our experience was that CUDA is super easy to grasp and understand. The hard part in writing CUDA code is not brand new functions or grammars but the throughput-oriented structure that we need to understand. The correspondence between functions and their usage is pretty clear and easy to understand once we have known the organization and layout of GPU.

In this assignment, we're required to switch to OpenCL, which makes us remember the time when we can use CUDA. The first impression of this library is verbosity, where we have to write tones of code just to implement a simple HelloWorld program. Another point we can not stand is that the arguments of functions are too hard to remember, which makes it difficult to write code without documentation. Overall, we think OpenCL is much less elegant and hard to use.

Performance

In the SAXPY application, we did simple element multiplication and addition. We used Google Colab as our platform, where Telsa K80 was equipped. Our expectation was CUDA and OpenCL should have similar performance, and actually this was the case. However, we found it strange when we varied the size of the array when testing. The performance of OpenCL remained roughly the same when we varied the array size from 5000 to 200000, while the performance of CUDA fluctuated a little bit, which varied from 0.024 ms to 0.036 ms when the size was changed.

This is quite strange because we did the test on the same platform and both of them used global memory and the same block size. Our assumption is that the difference lies in subtle differences in memory access patterns and some optimization techniques used by CUDA and OpenCL. In CUDA we specifically specified to use global memory, while in OpenCL the program might implicitly transfer data to shared memory, which leads to performance difference. There also might be some measurement errors which are hard to avoid.

In the particle simulation, the CUDA version is better with a large number of particles. We expected that the CUDA one should perform a little better for the overhead the OpenCL drivers may bring. However, we still are not quite sure why the gap became huge with more particles. Our guess is that OpenCL drivers did some adaption with data transfer for universal use of other devices. But even though compiled by nvvc, the program can not be profiled by nvvc so we failed to figure it out.