



Uni.lu HPC School 2021

PS10b: Introduction to OpenCL Programming

Uni.lu High Performance Computing (HPC) Team
T. Carneiro L. Koutsantonis

University of Luxembourg (UL), Luxembourg

High Performance
Computing &
Big Data Services

 hpc.uni.lu

 hpc@uni.lu

 @ULHPC

LU  **EMBOURG**
LET'S MAKE IT HAPPEN

<https://hpc.uni.lu/>

Introduction: what's OpenCL?

- In **OpenACC** the compiler is responsible for the parallelization...
 - which might not work in all scenarios
 - may not result in the best performance
 - It is not portable

In turn...

- **OpenCL** is a standard for heterogeneous programming
 - multicore CPUs, GPUs (AMD, Intel, ARM),
 - FPGAs, Apple M1, tensor cores, and ARM
- **with minor or no modifications.**

Introduction: portability

- OpenCL vs. CUDA - Portability (**)



VS.

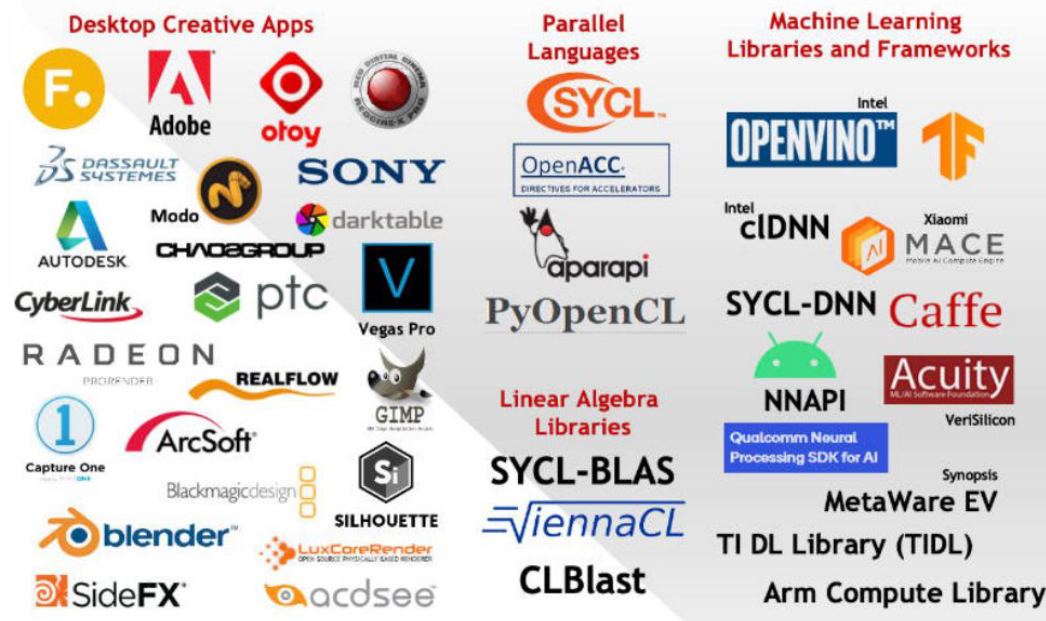


Images from:

<https://www.khronos.org/opencl/>

Introduction: target audience

- **Target audience of OpenCL:** programmers that aim at programming portable heterogeneous code and that want full control of the parallelization process.
 - **Portability has a price:**
OpenCL is much lower level than OpenACC and even then CUDA.



Images from:

<https://www.khronos.org/opencl/>

Have components/ are programmed in OpenCL



Introduction: objectives

- In this tutorial, we teach how to perform the sum of two vectors $C=A+B$ in OpenCL
- **Objectives of the tutorial:**
 - The **main objective** of this tutorial is to introduce for students of the HPC school the heterogeneous programming standard - OpenCL
- **This tutorial covers:**
 - Check for OpenCL-capable device(s);
 - Memory allocation on the device;
 - Data transfer to the device;
 - Retrieve data from the device;
 - Compile C/C++ programs that launch OpenCL kernels.

The OpenCL Platform Model

- **Similar to the CUDA Programming model:**
 - **The code has two parts:** the host and the device parts.
 - **According to the OpenCL specification:** "The model consists of a host (usually the CPU) connected to one or more OpenCL devices (e.g., GPUs, FPGAs)."

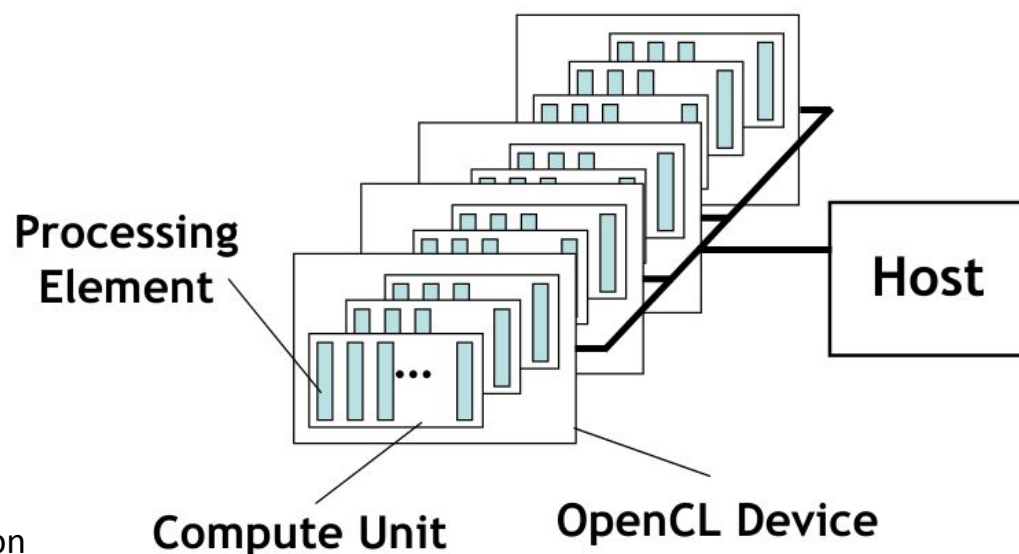
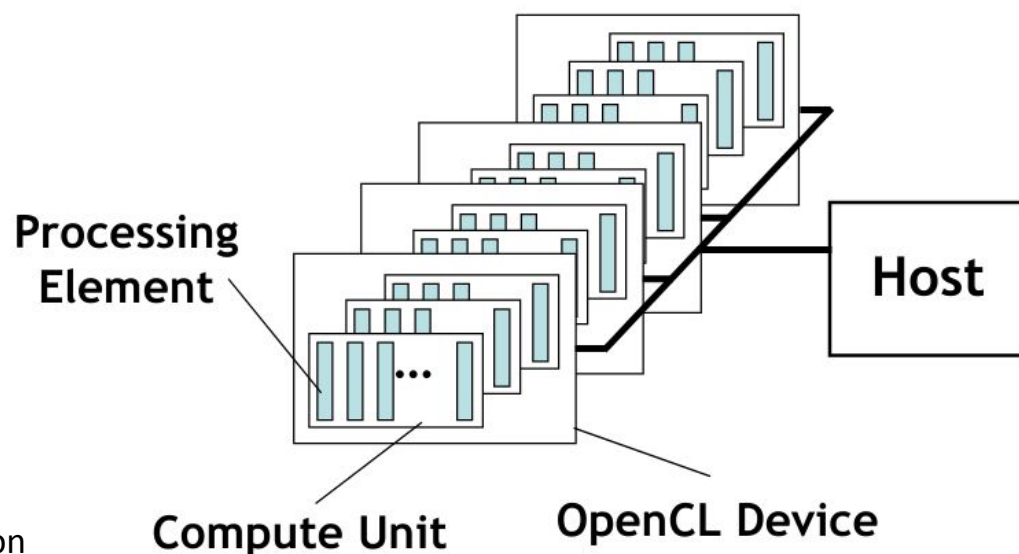


Image from the OpenCL specification
3.0.9

The OpenCL Platform Model

- **Similar to the CUDA Programming model:**
 - **The code has two parts:** the host and the device parts.
 - **According to the OpenCL specification:** "The model consists of a host (usually the CPU) connected to one or more OpenCL devices (e.g., GPUs, FPGAs)."



Host: launches the code that is executed on the device(s).

Image from the OpenCL specification
3.0.9

The OpenCL Platform Memory Model

- **Similar to the CUDA programming model:**
 - the host communicates with the *device(s)* through the global memory of the device(s).
 - As in the CUDA programming model, there is a **memory hierarchy** on the device.

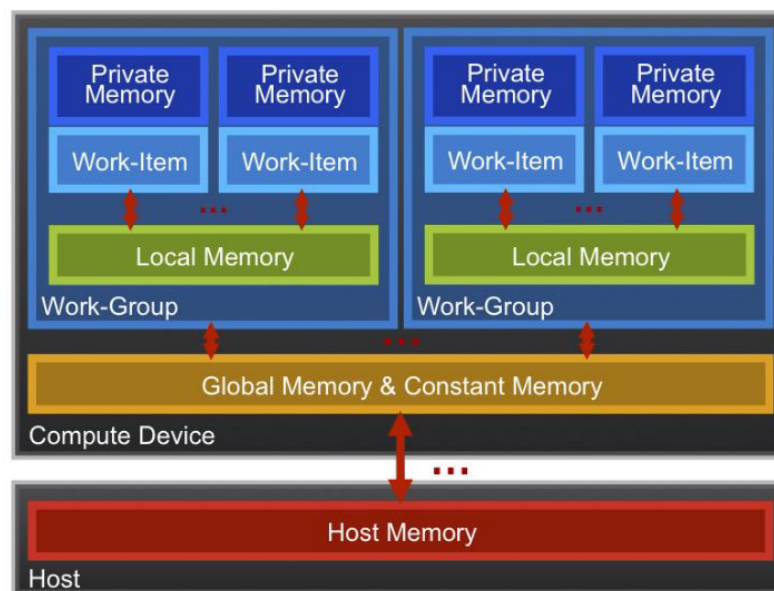


Image from Mattson, T.,
McIntosh-Smith, S., Koniges, A.
OpenCL: a Hands-on Introduction

Starting an OpenCL Context

- Initially, OpenCL needs to initialize a context
 - Contexts are used by the OpenCL runtime for *managing* command queues, memory, and for executing kernels on one or more devices connected to the context (*The OpenCL Cookbook*).

```
cl::Context context(CL_DEVICE_TYPE_DEFAULT);
```

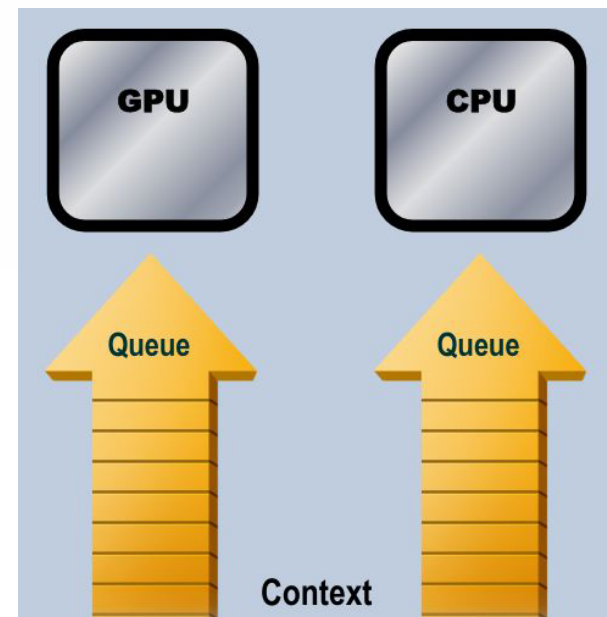


Image from Mattson, T., McIntosh-Smith, S., Koniges, A. *OpenCL: a Hands-on Introduction*

Starting an OpenCL Context

- Initially, OpenCL needs to initialize a context
 - Contexts are used by the OpenCL runtime for *managing* command queues, memory, and for executing kernels on one or more devices connected to the context (*The OpenCL Cookbook*).

```
cl::Context context(CL_DEVICE_TYPE_DEFAULT);
```

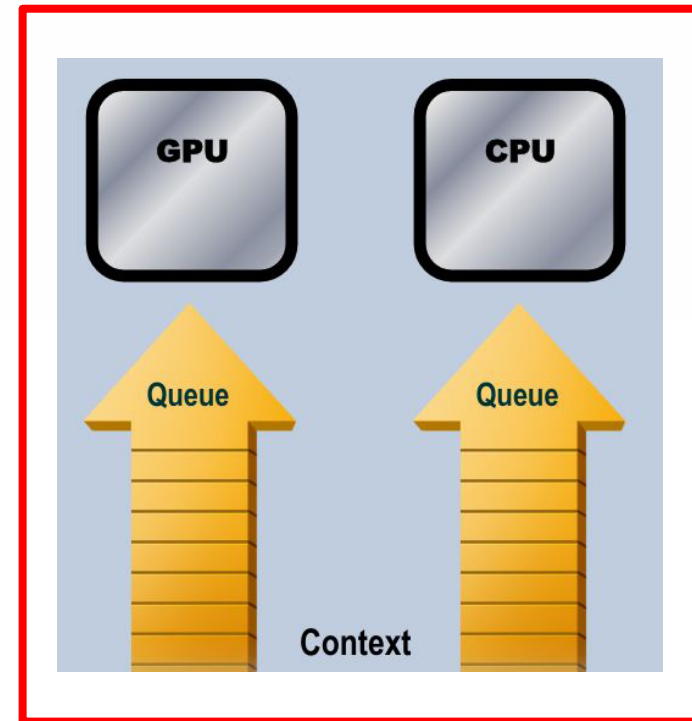


Image from Mattson, T., McIntosh-Smith, S., Koniges, A. *OpenCL: a Hands-on Introduction*

Starting an OpenCL Context

- **Initially, OpenCL needs to initialize a context**
 - Contexts are used by the OpenCL runtime for *managing* command queues, memory, and for executing kernels on one or more devices connected to the context [0].

```
cl::Context context(CL_DEVICE_TYPE_DEFAULT);
```

- **Then, it is required to initialize a queue:**
 - Push commands to the device.
 - For those who program in CUDA, OpenCL queues similar to CUDA streams.

```
cl::CommandQueue queue(context);
```

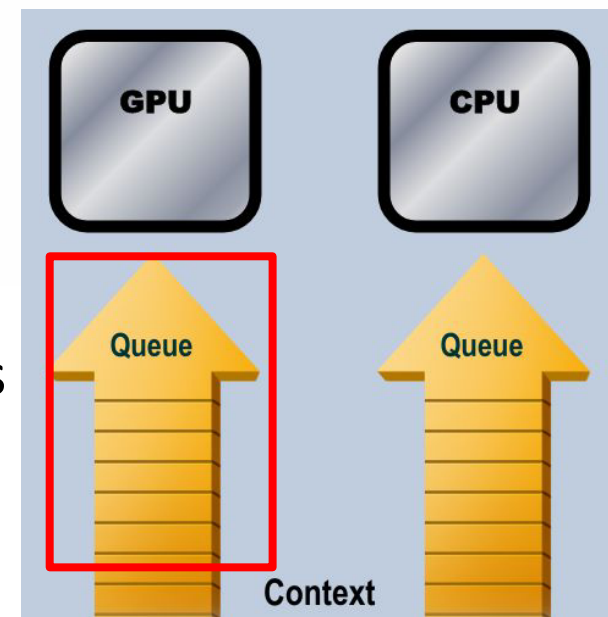


Image from Mattson, T., McIntosh-Smith, S., Koniges, A. *OpenCL: a Hands-on Introduction*

Allocating Memory on the Device: cl::Buffer

- As the device works with vectors A,B,C, we create three buffers:

```
cl::Buffer A_d(context, CL_MEM_READ_ONLY, sizeof(int) * SIZE);  
cl::Buffer B_d(context, CL_MEM_READ_ONLY, sizeof(int) * SIZE);  
cl::Buffer C_d(context, CL_MEM_WRITE_ONLY, sizeof(int) * SIZE);
```

- Using the queue, we write to the buffers of A and B to send their values of A and B from the host to the device:

```
queue.enqueueWriteBuffer(A_d, CL_TRUE, 0, sizeof(int) * SIZE, A_h);  
queue.enqueueWriteBuffer(B_d, CL_TRUE, 0, sizeof(int) * SIZE, B_h);
```

Online Compilation of the Kernel

- For the sake of greater portability, the kernel is provided as a string:

```
std::string kernel_code =  
    "    void kernel_simple_add(global const int* A, global const int* B, global int* C){ "  
    "        C[get_global_id(0)]=A[get_global_id(0)]+B[get_global_id(0)];        "  
    "    }"
```

- Kernels need to return the *void* type.
- The *global* keyword means that the buffer lies on the *global memory*

Building the Kernel

- To build the kernel, we append the kernel string to the program source:

```
cl::Program::Sources sources;  
sources.push_back({ kernel_code.c_str(), kernel_code.length() });
```

- Then, we create a program object using the program source as argument:

```
cl::Program program(context, sources);
```

- **Important:** in this tutorial we use the online compilation of the kernel. Thus, compilation errors of the *device* code are known in **execution time**.

```
if (program.build({ default_device }) != CL_SUCCESS)
```

Building the Kernel

- From the *program*, which contains the *simple_add* kernel, we instantiate a kernel object for execution with three *cl:buffers* as arguments:

```
cl::make_kernel<cl::Buffer, cl::Buffer, cl::Buffer> simple_add(cl::Kernel(program, "simple_add"));
```

- Now, we are able to invoke the kernel *simple_add*:

```
cl::NDRange global(SIZE);  
simple_add(cl::EnqueueArgs(queue, global), A_d, B_d, C_d).wait();
```

- And retrieve the result by using the *queue*:

```
queue.enqueueReadBuffer(buffer_C, CL_TRUE, 0, sizeof(int) * SIZE, C);
```

Compiling the Code

- It is simple to compile an OpenCL code with *gcc* or *g++*. In short, it is only required to add `-lOpenCL` flag to the compilation command.

```
$ g++ -lOpenCL exercise1.cpp
```

- **Exercise 1:** what's the output of the first exercise? Is the default OpenCL platform of Iris the NVIDIA platform?
- **Exercise 2:** following the online tutorial, compile *exercise1.cpp* using the NVIDIA implementation of OpenCL (`-l` and `-L`).
- What's the OpenCL platform after compiling *exercise1.cpp* using the NVIDIA implementation of OpenCL?
- **Obs:** *solutions are in the online tutorial.*

Conclusions

- OpenCL is an **open standard** for heterogeneous programming.
 - However, it exposes the programmer to **much lower-level** concepts than OpenACC and CUDA.
- Depending on the application, one might get a similar performance compared to OpenCL or CUDA using OpenACC.
- **Thus, the programmer needs to decide whether to follow a lower level approach:**
 - Need for portability - OpenCL;
 - The OpenACC compiler can't accelerate the code properly;
 - The OpenACC code became too complex and it is worth to move to OpenCL or CUDA.



Thank you for your attention!



?

University of Luxembourg, Belval Campus
Maison du Nombre, 4th floor
2, avenue de l'Université
L-4365 Esch-sur-Alzette
mail: hpc@uni.lu

<https://hpc.uni.lu/>



References

- [The OpenCL Cookbook](#)
- [Tutorial: Simple start with OpenCL and C++](#)
- [Khronos OpenCL Working Group. The OpenCL Specification \(Oct. 2021\)](#)
- [Smistad, E. Getting started with OpenCL and GPU Computing, Feb. 22, 2018 \(Access on Oct. 28, 2021\).](#)
- [Mattson, T., McIntosh-Smith, S., Koniges, A. OpenCL: a Hands-on Introduction](#)