Opencl 概述

Opencl(Open Computing Language) 首先由 Apple 提出，现由 Khronos Group 维护，是一个异构并行标准。因此 Opencl 只定义标准，不提供实现。Opencl 不仅可以用于 GPU，还可以用于 CPU，ARM，FPGA 等，这是 CUDA 不能比的。

Opencl 包含两部分，Opencl C/C++语言(内核编程语言)和主机端 API。

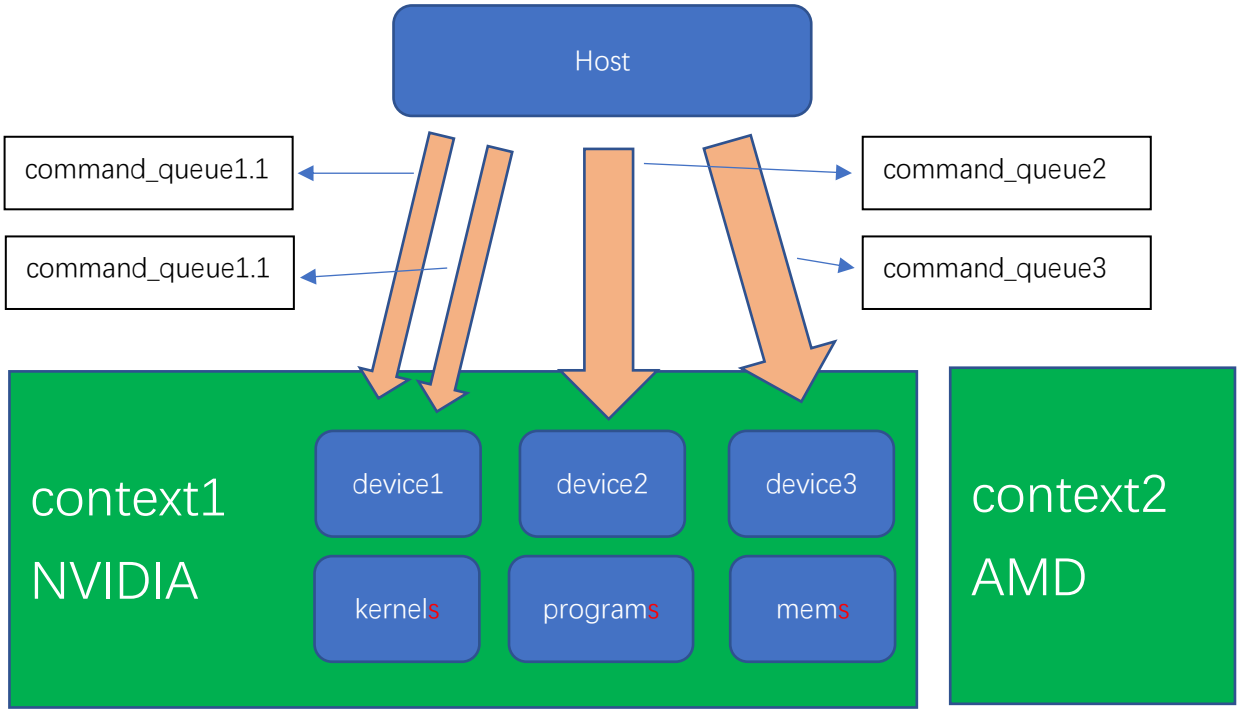
模型
- 平台模型
- 执行模型
- 存储器模型
- 编程模型

在此处键入公式。

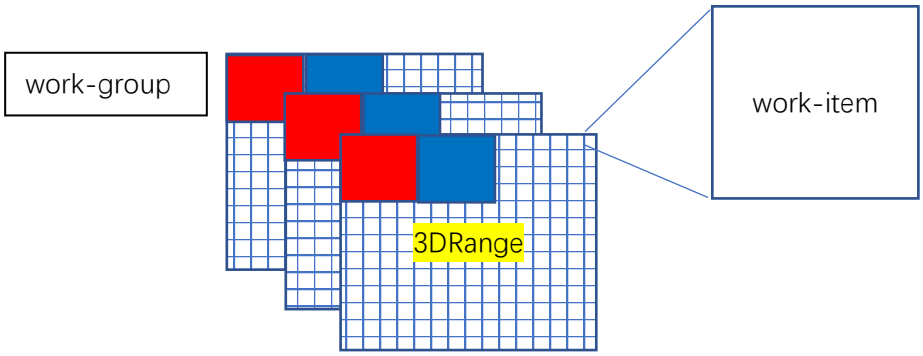
平台模型:

有且只有一个主机+至少一个设备

执行模型:



执行 kernel 时会创建一个 N 维网格(索引空间)，每一维是一个矩阵。N 的取值可以是 1，2 和 3，1 维网格刚好可以对应一幅灰度图，3 维网格刚好可以对应一幅彩色图。每一个格子执行内核的一个实例，称为(work-item)。



N 维索引空间又可以划分为若干个 work-group，所以 work-group 的大小和数量决定了索引空间的大小。work-item 在 work-group 中的坐标(local_id)，work-group 在索引空间中的坐标(group_id)以及 work-item 在索引空间中的坐标(global_id)都可以通过函数获得。值得注意的是，local_id 的起始坐标始终是 0，而 group_id 的起始坐标可能不为 0，因为每一维索引空间可能存在一个偏移。

存储器模型：

$$\begin{cases} \text{host mem} & \text{仅 host 读写} \\ \textit{global mem} & \text{device 全局读写} \\ \textit{constant mem} & \text{device 全局只读} \\ \textit{local mem} & \text{group 读写} \\ \textit{private mem} & \text{item 读写} \end{cases}$$

$$\begin{cases} \text{buffer} \\ \textit{image} \\ \textit{pipe} \end{cases}$$

编程模型：

小结

context 在 Opencl 中处于核心地位，Opencl 将编译运行用到的资源整合到 context 中。设备必须包含在 context 中，代码编译必须在 context 中。值得注意的是，一个 context 只能包含同一个平台(NVIDIA,AMD,intel,···)的设备。这是因为 Opencl 是运行时编译，实现并不知道会用哪个平台的实现。假如 context 中包含了不同平台，编译时并不知道将来在哪个平台的设备上执行，编译就无从谈起。

Workflow

1	clGetPlatformIDs()	两次，第一次获取平台数量，第二次获取平台信息
2	clGetDeviceIDs()	两次，第一次获取设备数量，第二次获取设备信息

3	clCreateContext()	将同一平台的若干设备组织为 context
4	clCreateCommandQueue()	为 context 中的某个设备创建 command queue
5	clCreateProgramWithSource()	在 context 中生成 program
6	clBuildProgram()	编译 program
7	clCreateKernel()	用 program 生成 kernel
8	clSetKernelArg()	为 kernel 设置参数
9	clEnqueueNDRangeKernel()	将 kernel push 到 command queue
10	clRelease…()	释放前面分配的资源，凡是有 Create 的都要 Release

上面表格中的第 4 步可以适当后移，因为它不依赖于 kernel，但要在第 9 步之前。