3.2.5 异步并发执行

CUDA将下列操作作为独立的任务，可以彼此并发操作:

* 主机上的计算
* 设备上的计算
* 主机向设备内存传递
* 设备向主机内存传递
* 设备上的内存传递
* 设备间的内存传递

3.2.5.1 主机和设备间的并发执行

如果硬件计数器（hardware counters）被分析器（profiler, 例如Nsight、Visual Profiler）收集，kernels启动将是同步的；

如果Async内存复制函数相关的内存不是页锁定内存（Page-locked），那么也是同步的。

3.2.5.2 并发内核执行（Concurrent Kernel Exection）

计算能力2.x及以上的设备支持多个kernels并发执行。可以通过设备属性concurrentKernels查询，如果为1表示支持。

启动kernels的最大数量（即设备上允许驻留的最大grids数目）取决于计算能力，如[表15](https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#features-and-technical-specifications__technical-specifications-per-compute-capability)所示。

来自一个CUDA上下文的kernel不能与来自另外一个CUDA上下文的kernel并发执行。

使用大量纹理或本地内存的kernel不大可能与其他kernels并发执行。

3.2.5.3 数据传输与内核执行重叠（Overlap of Data Transfer and Kernel Execution）

应用程序可以通过asyncEngineCount设备属性查询是否支持数据传输与内核执行重叠（大于0表示支持）。如果主机内存想要参与这种重叠，那么必须是页锁定内存。

当然也支持在设备内的内存复制和内核执行重叠。设备内的内存复制由标准内存复制函数（源地址、目的地址）触发。

3.2.5.4 并发数据传输（Concurrent Data Transfers）

计算能力2.x及以上支持与设备的并发数据传输。可以通过设备属性asyncEngineCount查询，如果为2表示支持。主机内存必须使用页锁定内存以支持重叠。

3.2.5.5 流（Streams）

流是同步操作的集合。不同流之间是并发执行，且并发行为不能保证（例如，kernels之间的通信是未定义的）。

3.2.5.4.1 创建与销毁（Creation and Destruction）

// 创建两个流

cudaStream\_t stream[2];

for (int i = 0; i < 2; ++i)

cudaStreamCreate(&stream[i]); // 创建流

float\* hostPtr; // 主机指针

cudaMallocHost(&hostPtr, 2 \* size); // 分配页锁定内存（page-locked）

// 异步执行

for (int i = 0; i < 2; ++i) {

// 异步内存拷贝：从主机到设备

cudaMemcpyAsync(inputDevPtr + i \* size, hostPtr + i \* size,

size, cudaMemcpyHostToDevice, stream[i]);

// 启动kernel（总是异步的，但是命名流可以控制同步行为）

MyKernel <<<100, 512, 0, stream[i]>>>

(outputDevPtr + i \* size, inputDevPtr + i \* size, size);

// 异步内存拷贝：从设备到主机）

cudaMemcpyAsync(hostPtr + i \* size, outputDevPtr + i \* size,

size, cudaMemcpyDeviceToHost, stream[i]);

}

// 销毁流

for (int i = 0; i < 2; ++i)

cudaStreamDestroy(stream[i]);

3.2.5.4.2 默认流（Default Stream）

3.2.5.4.3 显示同步（Explicit Synchronize）

cudaDeviceSynchronize() 等待，直到所有主机线程的所有流中的所有前面的命令都已完成。

cudaStreamSynchronize() 将一个流作为参数并等待，直到给定流中的所有前面的命令都已完成。它可用于将主机与特定的流同步（会阻塞主机线程），允许其他流继续在设备上执行。

cudaStreamWaitEvent()