# D. CUDA动态并行（CUDA Dynamic Parallelism）

## D.1 简介

### D.1.1 概述

### D.1.2 术语

Grid：一个kernel函数以thread grids的形式发布；

Thread Block：Block是在同一个SM中执行的线程集合。

Kernel Fucntion：

## D.2 执行环境和内存模型

### D.2.1 执行环境

### D.2.1.1 父子线程格

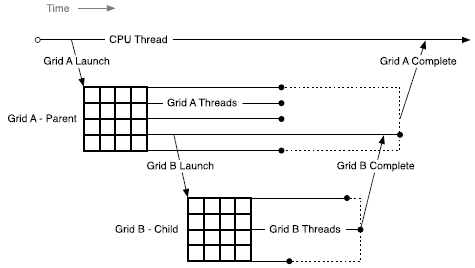


图14 父子线程格嵌套调用（Parent-Child Launch Nesting）

#### D.2.1.2 CUDA原语作用域

#### D.2.1.3 同步

#### D.2.1.4. 流与事件（Streams and Events）

**流：stream可用于grids之间的同步，同一stream中的grids同步执行；**

**事件：event可用于创建streams之间的依赖。**

**grid中创建的streams和events存在于block作用域，即实际上是由block创建streams和events的，在其他blocks中引用它是未定义的；**

**host创建的streams和events不可以用于kernel.**

#### **D.2.1.5 顺序和并行性（Ordering and Concurrency）**

**在一个block（指SM）中，发向同一个stream的kernels顺序执行；**

**相同block中的线程执行顺序取决于线程调度（thread schduling）——其实也是根据调度发向一个stream中——可以使用同步原语\_\_syncthreads()函数控制。**

**一个block中的streams由其内部的threads共享，所以隐式的NULL stream也共享。隐式的stream为“顺序执行”。如果需要并发，需要使用命名流（named streams）。**

**一个device上的不同thread blocks没有并发执行保证（no guarantee of concurrent execution）。**

**父子线程格（parent thread blocks and their child grids）之间没有并发保证。当父线程块创建线程格时，子线程格不保证会开始执行，直到父线程块到达显示同步点（例如，cudaDeviceSynchronize()）。**

**虽然并发常常很容易实现，但它可能会随着设备配置函数（a function of deviceconfiguration）、应用程序工作负载（application workload）和运行时调度（runtime scheduling）的不同而有所不同。因此，依赖于不同线程块（blocks）之间的任何并发是不安全的。**

#### **D.2.1.6 设备管理（device management）**

**设备运行时（runtime）不支持多gpu，设备运行时只能对当前正在其上执行的设备进行操作。但是，允许查询系统中任何支持CUDA的设备的属性。**

### **D.2.2 内存模型**

**父子线程格（Parent and child grids）拥有相同的global和constant内存储区，但是local和shared内存不同。**

#### **D.2.2.1 连贯性和一致性（****Coherence and Consistency）**

**Coherence：定义为向相同内存位置读写的行为；**

**Consistency：定义为向不同内存位置读写的行为。**

**D.2.2.1.1 全局内存（Global Memory）**

\_\_global\_\_ void child\_launch(int \*data) {

data[threadIdx.x] = data[threadIdx.x]+1;

}

\_\_global\_\_ void parent\_launch(int \*data) {

data[threadIdx.x] = threadIdx.x;

\_\_syncthreads();

if (threadIdx.x == 0) {

child\_launch<<< 1, 256 >>>(data);

cudaDeviceSynchronize();

}

\_\_syncthreads();

}

void host\_launch(int \*data) {

parent\_launch<<< 1, 256 >>>(data);

}

##### **D.2.2.1.2 零拷贝内存（Zero Copy Memory）**

零拷贝内存的连贯性和一致性（Coherence and Consistency）和全局内存一样。

kernel不能分配和释放零拷贝内存，但是可以使用主机程序（host program）传入的零拷贝内存指针。

##### **D.2.2.1.3 常量内存（Constant Memory）**

常量内存由主机提前设置，设备不能修改。

父子kernels之间自动继承常量内存，父子之间传递常量内存指针是允许的。

声明方式：\_\_constant\_\_

##### **D.2.2.1.4 共享和本地内存（Shared and Local Memory）**

共享内存：线程块（thread block）的私有内存；

本地内存：线程（thread）的私有内存。

共享和本地内存在父子之间不可见。

向发起的内核（kernel launch）传递本地内存和共享内存指针内存会引发NVIDIA编译器的警告。在运行时，编程人员可以通过调用内部（intrinsic）函数isGlobal()来确定指针是否指向全局内存，以向child launch安全地传递该（全局内存）指针。

调用cudaMemcpy\*Async()或cudaMemset\*Async()可能会在设备上调用新的kernel，以保证流语义（stream semantics），所以向这些函数传递共享内存和本地内存指针是不非法地。

##### **D.2.2.1.5 本地内存（Local Memory）**

本地内存是线程私有的，不允许传递其指针（即使是父子之间）。例如，下面的行为是非法的：

int x\_array[10]; // Creates x\_array in parent's local memory

child\_launch<<< 1, 1 >>>(x\_array); // Illegal!

所有向child kernel传递的内存需要被global-memory heap分配，即使是cudaMalloc()、new()或在全局作用域声明为\_\_device\_\_的内存，例如：

// Correct - "value" is global storage

\_\_device\_\_ int value;

\_\_device\_\_ void x() {

value = 5;

child<<< 1, 1 >>>(&value);

}

// Invalid - "value" is local storage

\_\_device\_\_ void y() {

int value = 5;

child<<< 1, 1 >>>(&value);

}

##### **D.2.2.1.6 纹理内存（Texture Memory）**

关于纹理访问（texture access），向映射到纹理的全局内存（the global memory region over which a texture is mapped）写入操作没有连贯性（incoherence）。

在child grid调用和child grid完成时，纹理内存（texture memory）的连贯性（coherence）被强制保证。这意味着在子内核启动之前对内存的写入（父级线程格的行为）会反映在子内核的纹理内存访问中。同样，子级对内存的写入将反映在父级对纹理内存的访问中，但这仅仅发生在父级对子级完成同步之后。父级和子级同时访问（可能会导致数据不一致。

## D.3 编程接口

### D.3.1 CUDA C++ 参考

#### D.3.1.2 流（Streams）

命名流（Named streams）和无名流（空流）(un-named streams, NULL streams）都可以在设备运行时获得。命名流被在thread-block中的线程使用，但是将不允许流的句柄（stream handles）想其他blocks或child/parent kernels传递，也就是说，流为thread-block私有。

相互独立的流是并发运行的。

所有的设备流（device streams）必须由cudaStreamCreateWithFlags()创建，并且使用cudaStreamNonBlocking参数，换句话说，在设备端的流总是并发执行的，不能相互同步；cudaStreamCreate只能被主机运行时调用，企图在设备端调用会引发编译期错误。

cudaStreamSynchronize()、cudaStreamQuery()在设备上不支持。所以，在设备上，如果想知道流发起（stream-launched）的kernels是否执行完毕，必须调用cudaDeviceSynchronize()。

##### D.3.1.2.1 隐式（NULL）流