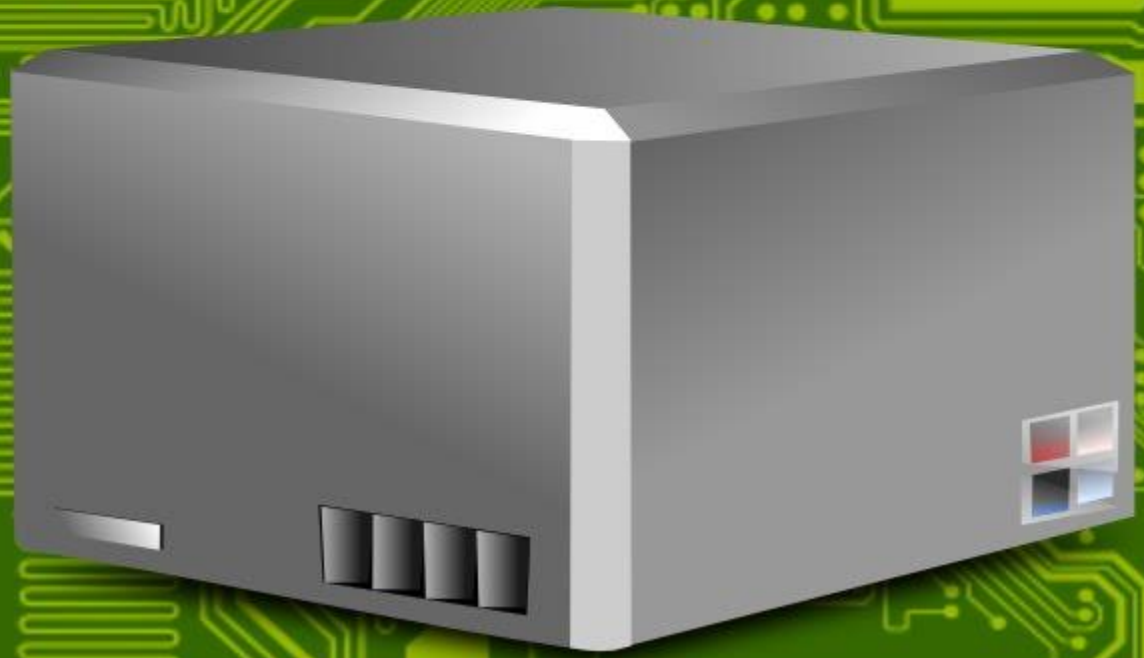
GPU Computing



CUDA编程模型

Hui Liu
Email: hui.sc.liu@gmail.com

GPU 计算基础知识

- CUDA编程模型是一个异构模型，需要CPU和GPU协同工作。
- 在CUDA中，host和device是两个重要的概念，我们用host指代CPU及其内存，而用device指代GPU及其内存。
- CUDA程序中既包含host程序，又包含device程序，它们分别在CPU和GPU上运行。
- host与device之间可以进行通信，这样它们之间可以进行数据拷贝。

CUDA 程序执行流程

1.分配host内存，并进行数据初始化;

2.分配device内存，并从host将数据拷贝到device上;

3.调用CUDA的核函数在device上完成指定的运算;

4.将device上的运算结果拷贝到host上 (性能)

5.释放device和host上分配的内存。

CUDA 程序

- 上面流程中最重要的一个过程是调用CUDA的核函数来执行并行计算.
- kernel 是 CUDA 中一个重要的概念，kernel 是 在device 上线程中并行执行的函数
- 核函数用 __global__ 符号声明，在调用时需要用<<<grid, block>>>来指定kernel要执行的线程数量
- 在CUDA中，每一个线程都要执行核函数，并且每个线程会分配一个唯一的线程号thread ID，这个ID值可以通过核函数的内置变量threadIdx来获得。

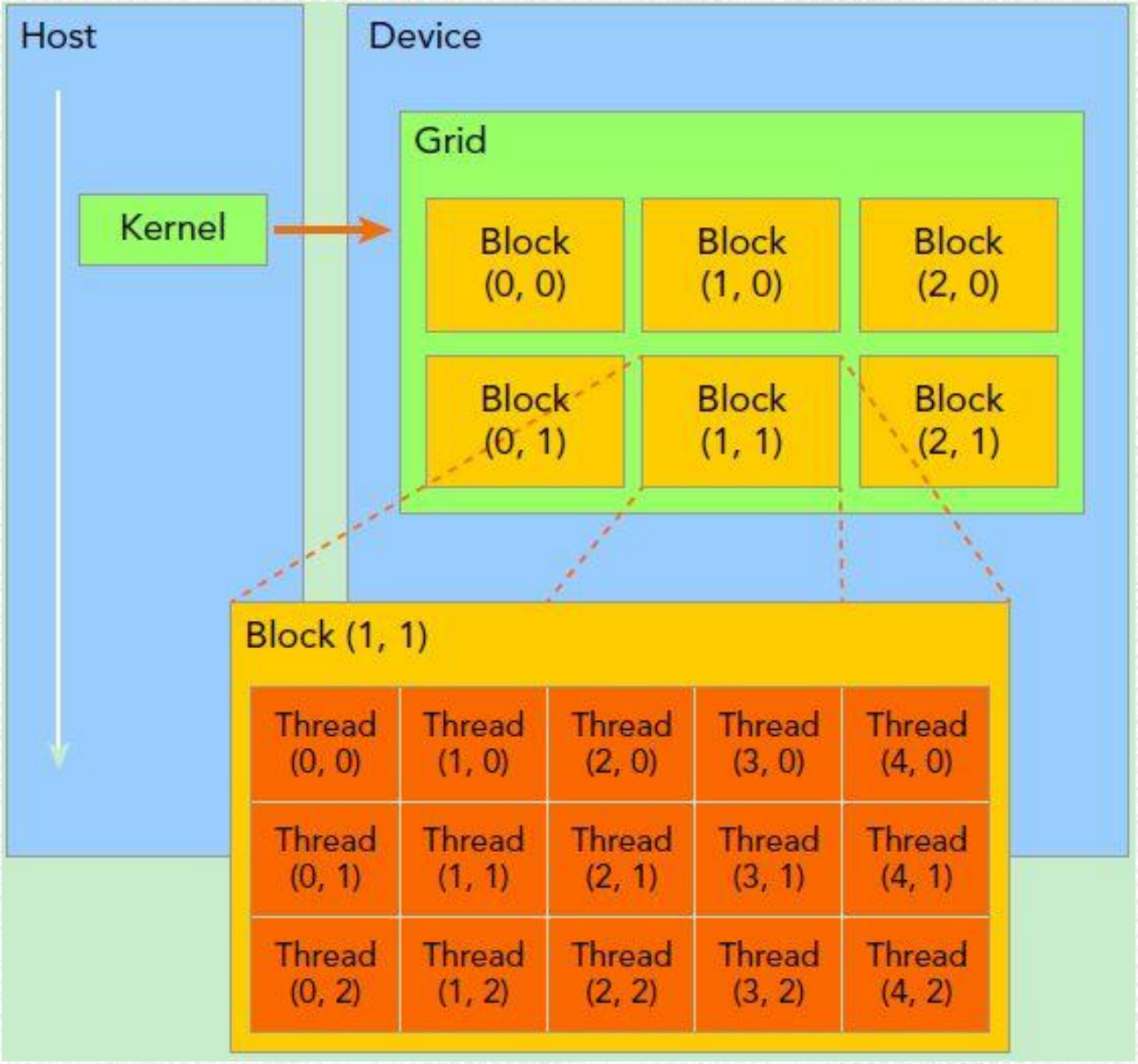
GPU 代码片段

```
// Kernel定义
__global__ void vec_add(double *x, double *y, double *z, int n)
{
    int i = get_tid();  // user-defined function

    if (i < n) z[i] = x[i] + y[id];
}

int main()
{
    int N = 1000000; // 1M
    int bs = 256;
    int gs = (N + bs - 1) / bs;

    // kernel, call GPU
    vec_add<<<gs, bs>>>(x, y, z, N);
}
```

CUDA程序层次结构

- GPU上很多并行化的轻量级线程。
- kernel在device上执行时实际上是启动很多线程，一个kernel所启动的所有线程称为一个网格 (grid)
- 同一个网格上的线程共享相同的全局内存空间，grid是线程结构的第一层次
- 网格又可以分为很多线程块 (block), 一个线程块里面包含很多线程，这是第二个层次。
- warp: 32个线程一组, 这是第三个层次

CUDA 程序架构


Host
Device
Grid
Kernel
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Block (1, 1)
Thread (0, 0)
Thread (1, 0)
Thread (2, 0)
Thread (3, 0)
Thread (4, 0)
Thread (0, 1)
Thread (1, 1)
Thread (2, 1)
Thread (3, 1)
Thread (4, 1)
Thread (0, 2)
Thread (1, 2)
Thread (2, 2)
Thread (3, 2)
Thread (4, 2)

CUDA程序层次结构

- grid 和 block 都是定义为dim3类型的变量
- dim3可以看成是包含三个无符号整数 (x, y, z) 成员的结构体变量，在定义时，缺省值初始化为1。
- grid和block可以灵活地定义为1-dim，2-dim以及3-dim结构
- 定义的grid和block如下所示，kernel在调用时也必须通过执行配置<<<grid, block>>>来指定kernel所使用的线程数及结构。
- 不同 GPU 架构, grid 和 block 的维度有限制

CUDA 程序调用

```
dim3 grid(3, 2);
dim3 block(5, 3);
kernel_fun<<< grid, block >>>(prams...);

dim3 grid(128,);
dim3 block(256);
kernel_fun<<< grid, block >>>(prams...);

dim3 grid(100, 120);
dim3 block(16,16,1);
kernel_fun<<< grid, block >>>(prams...);
```

CUDA程序层次结构

- GPU是异构模型，所以需要区分host和device上的代码，在CUDA中是通过函数类型限定词开区别host和device上的函数，主要的三个函数类型限定词如下：

1) __global__：在device上执行，从host中调用（一些特定的GPU也可以从device上调用），返回类型必须是void，不支持可变参数参数，不能成为类成员函数。

2) 注意用 __global__ 定义的kernel是异步的，这意味着host不会等待kernel执行完就执行下一步。

3) __device__：在device上执行，单仅可以从device中调用，不可以和__global__同时用。

4) __host__：在host上执行，仅可以从host上调用，一般省略不写，不可以和__global__同时用，但可和__device__，此时函数会在device和host都编译。

CUDA 内置变量

- 一个线程需要两个内置的坐标变量（blockIdx，threadIdx）来唯一标识，它们都是dim3类型变量，其中blockIdx指明线程所在grid中的位置，而threaIdx指明线程所在block中的位置:
- threadIdx包含三个值: threadIdx.x, threadIdx.y, threadIdx.z
- blockIdx同样包含三个值: blockIdx.x, blockIdx.y, blockIdx.z

- 一个线程块上的线程是放在同一个流式多处理器（SM)上的

- 单个SM的资源有限，这导致线程块中的线程数是有限制的，现代GPUs的线程块可支持的线程数可达1024个。

- 有时候，我们要知道一个线程在blcok中的全局ID，此时就必须还要知道block的组织结构，这是通过线程的内置变量blockDim来获得。它获取线程块各个维度的大小。

- 对于一个2-dim的block ，线程 的ID值为 ，如果是3-dim的block ，线程 的ID值为 。另外线程还有内置变量gridDim，用于获得网格块各个维度的大小。

GPU 代码片段

```
/* get thread id: 1D block and 2D grid */
#define get_tid() (blockDim.x * (blockIdx.x + blockIdx.y * gridDim.x) + threadIdx.x)

/* get block id: 2D grid */
#define get_bid() (blockIdx.x + blockIdx.y * gridDim.x)

__global__ void vec_add(double *x, double *y, double *z, int n)
{
   int i = get_tid();  // user-defined function

   if (i < n) z[i] = x[i] + y[id];
}
```

THANK YOU