**深度学习（三十六）异构计算CUDA学习笔记（1）**

2016年05月26日 12:33:10

阅读数：5770

**异构计算CUDA学习笔记（1）**

**原文地址**：<http://blog.csdn.net/hjimce/article/details/51506207>

**作者**：hjimce

近日因为感觉自己在深度学习工程化之路比较薄弱，故此开始学习CUDA编程，弥补自己在这方面的缺陷。写笔记以记录自己对cuda编程的一些简单理解。

个人感觉学习CUDA，最重要两点是：

1、理解Grid、Block、线程之间的层次关系；

2、理解存储器的层次关系（共享存储器、显卡等）。

**一、硬件知识：**

GPU硬件知识：Grid网格(每个显卡的个数)-》block-》线程

**1、Grid**：一个Grid代表一块GPU芯片，所有的线程共享显存数据；每个grid就相当于一块显卡。

**2、Block**：在每一个GPU芯片里面包含着多个block,每个block包含了512或者1024个线程。

每个线程的ID号可以通过一维0~1024索引，也可以通过二维dx\*dy=1024索引，或者通过三维dx\*dy\*dz=1024。这个就像图像opencv访问某个像素点一样，可以通过一维访问、或者二维访问：i+width\*j。

每个块里各自有一个共享数据存储的区域，只有块内的线程可以访问；在一个块内，共享变量的修改，可能需要用到等待所有的线程处理完毕，然后再修改共享变量，可以采用syncthreads（）函数用于等待。

3、**Thread**：每个block包含多个thread。

GPU存储空间：

(1)block中的每个线程都有自己的寄存器和local memory；

(2)block中的所有线程共享一个shared memory；

(3)一个grid共享一个global memory（或者称之为显存）、常量存储器、纹理存储器。

根据这些存储器的不同，我们后面定义的变量的时候，也要使用限定符，告诉程序，我们所要定义的变量是位于那个存储器，具体后面再解释。

**二、CUDA编程步骤：**

1、设置显卡编号：cudaSetDevice；

2、为显卡开辟变量内存:cudaMalloc；

3、把cup上的数据拷贝到GPU上：cudaMemcpy；

4、调用内核函数\_\_global\_\_类型函数；

5、把计算结果拷贝到CPU上：cudaMemcpy；

6、释放显存空间cudaFree；

示例代码：

**[cpp]** [view plain](https://blog.csdn.net/hjimce/article/details/51506207) [copy](https://blog.csdn.net/hjimce/article/details/51506207)

1. //计算a、b相加，得到c，size输入向量的维度
2. cudaError\_t addWithCuda(**int** \*c, **const** **int** \*a, **const** **int** \*b, unsigned **int** size)
3. {
4. **int** \*dev\_a = 0;
5. **int** \*dev\_b = 0;
6. **int** \*dev\_c = 0;
7. cudaError\_t cudaStatus;
9. //选择显卡
10. cudaStatus = cudaSetDevice(0);
11. **if** (cudaStatus != cudaSuccess) {
12. fprintf(stderr, "cudaSetDevice failed!  Do you have a CUDA-capable GPU installed?");
13. **goto** Error;
14. }
16. // 在显存上，开辟空间，存储变量c
17. cudaStatus = cudaMalloc((**void**\*\*)&dev\_c, size \* **sizeof**(**int**));
18. **if** (cudaStatus != cudaSuccess) {
19. fprintf(stderr, "cudaMalloc failed!");
20. **goto** Error;
21. }
22. // 在显存上，开辟空间，存储变量a
23. cudaStatus = cudaMalloc((**void**\*\*)&dev\_a, size \* **sizeof**(**int**));
24. **if** (cudaStatus != cudaSuccess) {
25. fprintf(stderr, "cudaMalloc failed!");
26. **goto** Error;
27. }
28. // 在显存上，开辟空间，存储变量b
29. cudaStatus = cudaMalloc((**void**\*\*)&dev\_b, size \* **sizeof**(**int**));
30. **if** (cudaStatus != cudaSuccess) {
31. fprintf(stderr, "cudaMalloc failed!");
32. **goto** Error;
33. }
35. //把数据a、b拷贝到显存上
36. cudaStatus = cudaMemcpy(dev\_a, a, size \* **sizeof**(**int**), cudaMemcpyHostToDevice);
37. **if** (cudaStatus != cudaSuccess) {
38. fprintf(stderr, "cudaMemcpy failed!");
39. **goto** Error;
40. }
42. cudaStatus = cudaMemcpy(dev\_b, b, size \* **sizeof**(**int**), cudaMemcpyHostToDevice);
43. **if** (cudaStatus != cudaSuccess) {
44. fprintf(stderr, "cudaMemcpy failed!");
45. **goto** Error;
46. }
48. // 设置核函数
49. addKernel<<<1, size>>>(dev\_c, dev\_a, dev\_b);
51. // Check for any errors launching the kernel
52. cudaStatus = cudaGetLastError();
53. **if** (cudaStatus != cudaSuccess) {
54. fprintf(stderr, "addKernel launch failed: %s\n", cudaGetErrorString(cudaStatus));
55. **goto** Error;
56. }
58. // cudaDeviceSynchronize waits for the kernel to finish, and returns
59. // any errors encountered during the launch.
60. cudaStatus = cudaDeviceSynchronize();
61. **if** (cudaStatus != cudaSuccess) {
62. fprintf(stderr, "cudaDeviceSynchronize returned error code %d after launching addKernel!\n", cudaStatus);
63. **goto** Error;
64. }
66. //把计算结果拷贝到cpu
67. cudaStatus = cudaMemcpy(c, dev\_c, size \* **sizeof**(**int**), cudaMemcpyDeviceToHost);
68. **if** (cudaStatus != cudaSuccess) {
69. fprintf(stderr, "cudaMemcpy failed!");
70. **goto** Error;
71. }
73. Error:
74. cudaFree(dev\_c);//释放空间
75. cudaFree(dev\_a);
76. cudaFree(dev\_b);
78. **return** cudaStatus;
79. }

**三、CUDA内置函数库**

1、Cuda的函数命名规则：以cuda开头+该函数功能名，函数功能名每个单词的第一个字母都是大写，比如：cudaSetDevice，也就是cuda+Set+Device。一些常用的函数功能名基本上都是和c语言一样，比如：cudaMalloc、cudaMemcpy。

2、GPU称之为设备device；device(0)表示设置显卡号码，多显卡，在CUDA程序中，我们可以采用:cudaSetDevice(0)函数，表示选用第一块显卡进行计算。

3、CPU称之为主机host；

所以cuda在定义函数、变量的时候，前面会有个限定词：host、device、global，三者分别表示定义的函数：在cpu调用执行、在gpu调用执行、cpu调用gpu执行。

**四、CUDA自定义函数**

因为程序是在gpu、cpu不同的设备上混合使用的，所以在自定义函数的时候，需要加入函数限定词\_\_device\_\_ , \_\_global\_\_, \_\_host\_\_；这些是用来告诉程序，你定义的这个函数是要在GPU上调用执行，还是要在CPU上调用执行。

(1)\_\_device\_\_：表示从GPU上调用，在GPU上执行；

(2)\_\_global\_\_：表示在CPU上调用，在GPU上执行，也就是所谓的内核(kernel)函数；内核主要用来执行多线程调用。

(3)\_\_host\_\_：表明在CPU上调用，在CPU上执行，这是默认时的情况，也就是传统的C函数。

示例：

**[cpp]** [view plain](https://blog.csdn.net/hjimce/article/details/51506207) [copy](https://blog.csdn.net/hjimce/article/details/51506207)

1. \_\_global\_\_ **void** addKernel(**int** \*c, **const** **int** \*a, **const** **int** \*b)
2. {
3. }

定义了一个函数名为addkernel的函数，该函数限定词为global，表示该函数由cpu调用，由GPU执行。

**四、重要类型变量**

**1、dim3**

这个类型是线程索引必备数据结构，是一个向量类，类似于opencv的vec3；

在定义dim3变量的时候，为指定的分量都自动被初始化为1，dim3类型的变量定义完毕后，我们可以采用.x,.y.z访问每个维度的数值。

例子：

**[cpp]** [view plain](https://blog.csdn.net/hjimce/article/details/51506207) [copy](https://blog.csdn.net/hjimce/article/details/51506207)

1. dim3 bb(10,20);
2. std::cout <<bb.x<<","<<bb.y<<","<<bb.z<<std::endl;

这个时候可以看到输出bb.z=1；

切记上面会自动为未被分配的分量初始化为1。这个对于后面<<< >>>的配置，理解非常重要，因为<<< ……  >>>的前两个输入参数是dim3类型，如果你输入参数是一个数值a，那么会被自动转换成dim(a,1,1)向量。

**2、cudaError、cudaDeviceProp等程序信息**

除了int、float等还有，cuda内部定义的一些结构体：

cudaError#让我们获得相关信息；

cudaDeviceProp#获得设备的相关参数，比如GPU线程个数、显存大小；

**3、变量类型**

int3、int2……等可以用于定义一个变量是3维、2维整型向量，当然还有其他float向量等

**五、配置内核函数**

内核函数的调用格式：函数名<<<Dg,Db,Ns,s>>>(函数参数);

内核函数的输入参数就是我们开辟显存，然后从CPU上把数据拷贝到GPU的变量

调用内核函数都需要配置参数，<<<dim3 Dg,dim3 Db,size\_t Ns,cudaStream\_t s>>>内核函数配置参数：

Dg、Db都是dim3类型，可能我们在使用的时候直接输入数值<<<2，50>>>，这样其实系统会自动进行类型转换，把2转换成dim3 Dg(2,1,1)。

示例：

<<<2，50>>>，表示采用2个block，每个block启用50个线程进行计算，这样算下来一共有100个线程；

<<<1, size>>>表示运行时配置符号，里面1表示只分配一个线程组（又称线程块、Block）、size表示每个线程组有size个线程。size个线程都会调用这个核函数，我们可以根据threadid获取当前调用该函数的线程。这样设置参数表示我们只用了一组线程块，同时调用了该线程块的size个线程。

<<<size，1>>>，那么就会有size个线程块，每个线程块只启用了一个线程计算。

六、**CUDA内置变量**

Cuda内置为我们定义了几个经常用到的变量，这些变量基本都是dim3类型。

1、gridDim：利用这个变量，我们可以通过gridDim.x、gridDim.y、gridDim.z，知道网格三个维度的尺寸，这个变量除非是GPU集群，单显卡这种肯定用不到，忽略。

blockDim：表示块在三个方向的长宽高

上面gridDim、blockDim其实等于在我们调用内核函数的时候，配置<<< >>>所需要的参数

2、blockldx、threadldx:分别用于索引当前线程所在的块，块中的线程编号

 七、**CUDA变量自定义**

我们在定义变量的时候，在程序中，如果不声明限定符的话，那么默认都是定义在cpu内存上。现在gpu还有显存、共享存储器、缓存等，所有在定义变量的时候，要声明限定符，告诉程序我们所定义的变量要存在哪里。

1、\_\_device\_\_：表明声明的数据存放在显存中，所有的线程都可以访问

2、\_\_shared\_\_：表示数据存放在共享存储器在，只有在所在的块内的线程可以访问，其它块内的线程不能访问

3、\_\_constant\_\_：表明数据存放在常量存储器中，可以被所有的线程访问，也可以被主机通过运行时库访问

参考文献：

1、《NVIDIA CUDA计算统一设备架构》

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*转载请保留原文地址、作者信息\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*