# CUDA内存模型

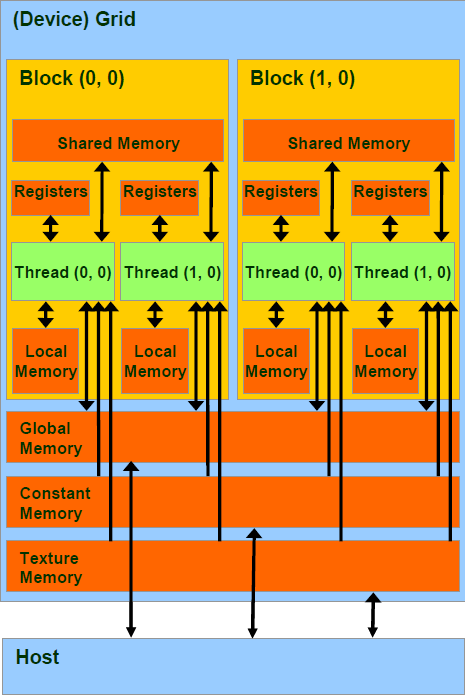


图1-1 CUDA存储器模型

对于任何一种编程语言，内存的访问和管理都是非常重要组成部分。在依靠硬件加速器的高性能计算领域，内存管理对性能会产生更大的影响。

许多应用程序包含大量数据读取和写入操作，这些操作往往会产生大量的延迟。如何快速的进行数据操作，提高数据读写效率，对降低访存延迟，提高内存带宽，获得更高的性能至关重要。

硬件方面，访存延迟低，容量大的内存从技术方面或经济方面往往是不可能的。因此，目前的计算机普遍采用内存层次结构模型来达到低延迟、高容量的目的。内存层次结构设计原理依赖于时间局部性和空间局部性。时间局部性假设如果某一个数据空间被访问，那么在一个短时间窗口内，这个数据空间会被再次访问。空间局部性假设如果某一个数据空间被访问，那么其相邻数据空间有可能同时被访问。内存层次结构由不同等级的存储器组成，图1-2展示了目前典型的内存层次结构，包含寄存器、缓存、内存、硬盘四个等级。存储器等级越高，访存延迟和消耗越低，带宽越高，但容量越小。内存层次结构能够使高时间局部性和空间局部性的数据存储到高等级的存储器中，从而实现大容量、低延迟的目的。

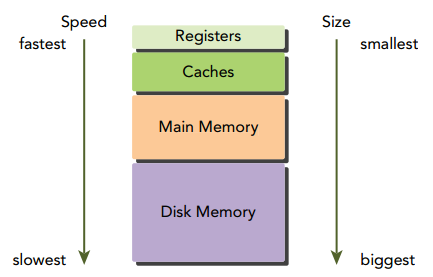


图1-2 典型的内存层次结构

内存层次结构能够从硬件方面提高访存性能，在软件方面，开发人员则可以优化应用程序，减少应用程序中不必要的数据读写操作，软硬结合，从而获得更大的性能提升。为了达到这一要求，开发人员必须充分了解编程语言的内存模型，利用局部性原理高效地对内存进行管理，以获得最低的延迟、最大的带宽和最高的性能。

CUDA作为目前流行高性能平台，有一套特定的内存模型，如图1-1所示。

在CUDA的内存模型中，共有六种类型的存储器，分别是寄存器、共享内存、本地内存、常量内存、纹理内存和全局内存。这六种内存可以分为两类，片上内存和片外内存。片上内存位于每一个流多处理器（SM, Stream-Multiprocessor）内，包括寄存器、共享内存和本地内存；片外内存可以被所有流多处理器访问，包括常量内存、纹理内存和全局内存。在近几代GPU架构（Fermi、Kepler）中，不同内存有不同的性质。以Kepler架构与Fermi架构对比为例，Kepler架构包含15个SM，Fermi架构为16个SM；寄存器的大小由32KB（Fermi）增长到64KB；L1 cache和共享内存的配置选项由Fermi架构的两个（共享内存-L1 cache：16KB-48KB， 48KB-16KB）增长到三个（16KB-48KB， 48KB-16KB，32KB-32KB）；L2 cache大小由768KB增长到1536KB，全局内存由6GB增长到12GB。

每一种内存都有不同的访存范围、生命周期和作用。核函数中的每一个线程拥有一块私有的本地内存，每一个线程块私有一块共享内存，该共享内存内的数据可以被同一线程块内的所有线程访问，共享内存的生命周期与线程块相同；常量内存和纹理内存都是可以被所有线程访问的只读内存，它们分别针对不同的应用场景进行了优化，纹理内存具有硬件插值、数据类型强制转化的功能，常量内存用于存储常量，具有低访存延迟、广播的功能。全局内存可供所有的线程进行读写操作。全局内存、常量内存和纹理内存内的数据在一个内核函数执行完成后将被继续保持，可以被同一程序中其也内核函数调用，生命周期与应用程序的生命周期相同。

在本章节中，将对常用的三种内存（全局内存、共享内存、常量内存）进行详细介绍。

## 1.1 全局内存

全局内存（Global Memory）是最大的，最常用的GPU内存，同时也是访存延迟最高的内存。Global代表该是指其作用域和使用寿命。在设备上的任何SM中的线程都对全局内存进行读写访问，其生命周期即整个应用程序的生命周期。

全局内存包括静态和动态两种申请方式。开发者可以使用\_\_device\_\_静态申请全局变量，也可以使用cud​​aMalloc()函数动态申请和使用cudaFree()函数动态释放的全局内存​​。全局内存指针可以作为核函数的参数传递到设备端。

线程执行时不能进行跨线程块同步。当多个线程同时写全局内存中相同位置的数据很有可能会造成访存冲突，这将导致不确定的程序行为。这种情况下，必须只用原子操作以保住数据写操作的正确性。

全局内存的访问带宽为32字节，64字节或128字节。当访问全局内存时，数据大小会根据访存带宽被自动对齐，即第一个数据地址必须是32，64或128的倍数。优化全局内存的访问行为将会大大优化应用的性能。当一个warp内的线程对全局内存中的数据进行读取时，所加载的数据大小由两个因素决定：①所涉及数据覆盖的内存大小；②内存地址的对齐。例如一个warp内的所有线程访问的数据大小为60字节，根据内存对齐原则，LDST（Load/Store）单元将读取64字节的数据到缓存中。

## 1.2 共享内存

在核函数中，用\_\_shared\_\_修饰的变量存储在共享内存中。共享内存也包括静态和动态两种申请方式。开发者这在核函数中使用\_\_shared\_\_ int sharedData[256]语句为每一个线程块静态地分配一块256\*sizeof(int)大小的共享内存；核函数在调用时，共有四个可配置参数，格式如下：

|  |
| --- |
| kernel\_name <<<grid\_size, block\_size, shared\_memory\_size, stream>>>(argument list); |

其中，grid\_size为核函数中block的数量，block\_size为一个block内的线程数量，shared\_memory\_size为动态申请的共享内存大小单位为byte，stream表示该核函数处在哪个流中执行，初始值为零，表示在默认流中执行。共享内存的动态申请方式为在核函数中不指定所使用共享内存大小，语句为：extern \_\_shared\_\_ int sharedData[]，由核函数的第三个参数动态设置线程块中可用共享内存大小。

共享内存为片上内存，相比于全局内存，共享内存具有更高的带宽和更低的访存延迟。每一个SM具有的共享内存大小有限，Kepler架构每个SM的默认共享内存大小为48K。SM中可同时运行的warp数量由两个个因素决定：共享内存资源和寄存器资源。开发者必须谨慎地使用共享内存，在线程块中过量的使用共享内存将会减少SM可承载的warp数量，降低核心占有率从而导致性能下降。共享内存在内核函数的内声明，其生命周期与线程块相同，当一个线程块执行完毕，它所占有的共享内存将被释放并分配给其他线程块。

共享内存是线程块内线程间通信的基本手段。线程块中的线程可以通过共享内存的数据操作来进行通信。使用共享内存必须使用\_\_syncthreads（）函数进行同步。该函数创建一个屏障，同一个线程块中的所有线程必须都到达屏障，线程才被允许继续执行。屏障的作用在于可以防止潜在的数据风险，比如脏数据，读写冲突等。

在每个SM中，共享内存和L1缓存共用同一块片上内存，它是静态地分配，但可在使用如下函数在运行时动态配置的：

|  |
| --- |
| CUDAError\_t CUDAFuncSetCacheConfig(const void\* func, enum CUDAFuncCache cacheConfig); |

该函数配置基于每个内核基础片上存储器的分区，设置用于通过func指定的内核函数的配置。以Kepler架构为例，支持的高速缓存配置如下：

|  |
| --- |
| CUDAFuncCachePreferNone：没有偏好（默认）  CUDAFuncCachePreferShared：48KB共享内存和16KB L1缓存  CUDAFuncCachePreferL1：48KB L1高速缓存和16KB共享内存  CUDAFuncCachePreferEqual：L1缓存和共享内存的大小相等分别为32KB |

## 1.3 常量内存

常量内存作为一块独特的内存区域，每个SM都有一个专用的常量缓存。用\_\_constant\_\_修饰的变量存储在常量内存中。常量内存必须在核函数之外的全局作用域进行声明。常量内存大小有限，在所有计算能力的显卡中，最多可使用 64 KB的常量内存。常量内存为只读内存，可以被同一个GPU的所有核函数中的线程进行读访问。

常量内存必须静态声明，在声明时必须指定数据类型和大小，如\_\_constant\_\_ int constant\_data[256]。常量内存赋值数据有两种方式，第一种是在声明时直接初始化赋值，第二种方式为在host端生成数据，调用cudaMemcpyToSymbol()函数将src中的数据复制到常量内存symbol中。

常量内存的使用场景较为特殊，当一个warp的所有线程从同一个内存地址读取数据情况时，常量内存会将该数据广播到该warp的其他的线程中，这种使用方式常量内存的性能最佳。例如，对于一个数学公式而言，将系数存储在常量内存中将获得最佳性能。

常量内存每一个时钟周期内的最大吞吐量为4B, 如果一个warp内的所有线程所需数据大小大于4B, 则需要多次读取。假如一个warp中的每个线程读取常量内存中不同数据，这种情况下，常量内存是不是最好的选择。

# 2 测试过程

用图表表示；不同的内存布局，不同的内存访问方式；

## 2.1 需求分析

## 2.2 实验设计

## 2.3 代码流程

第一段，

第二段，，不同的数据内部分布，不同的访问方式、不同的数据组织形式和不同的内存

第三段，简要描述测试代码流程，给出伪代码

# 3.实验结果

第一段，描述实验环境

第二段，

什么内存最快

1、不同数据组织形式（三幅图），不同数据内部分布（横坐标），不同内存（纵坐标），访存方式选顺序访问，选择合理的线程块大小和数据大小

什么样的访存方式最好

2、不同数据组织形式（三幅图），不同数据内部分布（横坐标），不同访问方式纵坐标），内存选全局内存，选择合理的线程块大小和数据大小

3、什么样的数据结构内部分布最好

不同数据组织形式（三幅图），不同访问方式（横坐标），不同数据内部分布纵坐标），内存选全局内存，选择合理的线程块大小和数据大小

4、什么样的数据组织形式最好

不同数据内部分布（三幅图），不同访问方式（横坐标），不同数据组织形式（纵坐标），内存选全局内存，选择合理的线程块大小和数据大小

# 4 总结

 得到简单的结论，什么内存最快，什么样的访问方式最好；