# GPUDirect RDMA技术

参考GPUDirect RDMA.pdf和

https://docs.nvidia.com/cuda/gpudirect-rdma/index.html#abstract

## 1、概述

GPUDirect RDMA技术是Kepler级GPU和CUDA 5.0中引入的直连技术，使用PCIe的标准特性，可实现GPU与第三方peer设备之间的直接数据交换。第三方设备包括：NIC，视频查询设备或存储适配器(Nvme?)

需要Quadro或Tesla系列显卡。

两个设备必须共享相同的上游PCIe根，对平台还有一些限制。

需要修改设备驱动程序来启动该功能。

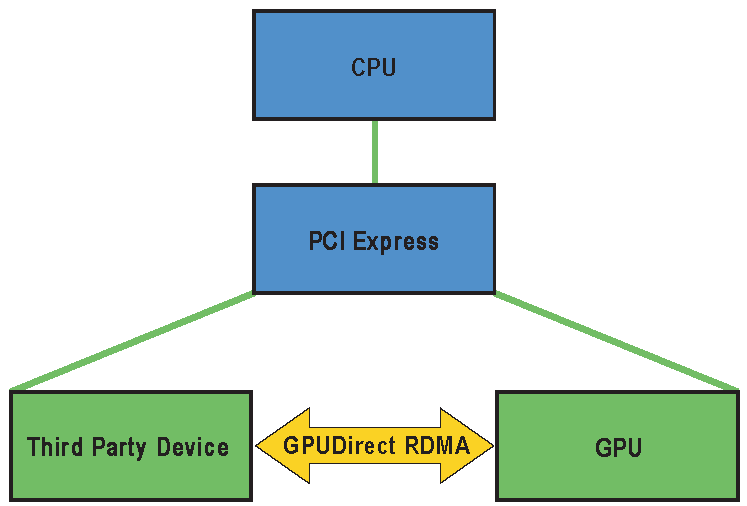


图1 Linux设备驱动模型下的GPUDirect RDMA

1.1 GPUDirect RDMA如何工作

当设置两个对等设备间GPUDirect RDMA通信时，从PCIe设备的角度来看，所有的物理地址是相同的。在该物理地址空间内，是线性窗口，称为PCI BARs。各设备至多有6个BAR寄存器，因此至多有6个活跃的32位BAR区域。64位BAR消耗2个BAR寄存器。PCIe设备读取和写出到对等设备的BAR地址，与读写系统内存的方式一样。

传统意义上，像BAR窗口的资源，使用CPU的MMU，映射到用户或内核地址空间，成为内存映射IO（MMIO）地址。但是，因为目前的操作系统没有足够的机制来交换驱动程序之间的MMIO区域，NVIDIA内核驱动输出函数，来实现必要的地址转换和映射。

为增加GPUDirect RDMA对设备驱动的支持，必须要修改内核驱动程序内的少量地址映射代码。该代码通常驻留在对get\_user\_pages()的当前调用。

GPUDirect RDMA的API和控制流与标准的DMA转移中使用的API非常相似。

1.2 标准的DMA转移

首先，大致介绍在用户层面初始化DMA转移。该过程中，存在如下部分：

* 用户程序；
* 用户通信库；
* 可实现DMA转移的设备的内核驱动。

一般的调用顺序如下：

1、用户程序通过用户通信库要求(request)转移。该操作取指向数据的指针（一个虚拟地址）和字节大小。

2、通信库必须确保对应虚拟地址的内存区域和大小已经为转移做好准备。如果没准备好，必须由内核驱动处理（下一步）。

3、内核驱动从用户通信库，接收虚拟地址和字节大小。然后，它要求内核将虚拟地址范围转换为物理页列表，确保他们已经准备好被转移(to or from)。我们称该操作为锁定（pinning）内存。

4、内核驱动使用页列表，启动物理设备的DMA引擎；

5、通信库初始化转移。

6、转移完成后，通信库最终清理所有用于锁定内存的资源。我们称为解锁（unpinning）内存。

1.3 GPUDirect RDMA转移

为支持GPUDirect RDMA转移的通信，需要对上述步骤做修改。首先，需要2个新的部分：

* 用户层的CUDA库；
* Nvidia内核驱动。

与UVA CUDA内存管理中的介绍，使用CUDA库的程序将地址空间分为GPU虚拟地址和CPU虚拟地址，通信库必须实施2个独立的路径。用户层CUDA库提供一个函数，使通信库可识别CPU或GPU地址。并且，对GPU地址，还能返回额外的用来单独识别由地址描述的GPU内存的元数据。

CPU地址和GPU地址的路径不同在于：内存是如何锁定和解锁的？对于CPU内存，这是内嵌的Linux内核处理的(get\_user\_pages()和put\_page())。但是，对于GPU内存，锁定和解锁是由NVIDIA内核驱动提供的函数完成的，细节参考Pinning GPU memory和Unpinning GPU memory。

## 2 设计的考虑

2.1 Lazy Unpinning优化

在BAR中锁定GPU设备内存是成本很高的操作，占用数ms。因此，应用程序应设计最小化overhead。

使用GPUDirect RDMA最直接的实施就是在各转移之前锁定内存，在转移完成之后立即解锁内存。不幸的是，这样做导致效率很低，因为锁定和解锁内存的成本也很高。但是，需要实施RDMA转移的剩下步骤可快速执行，无需进入内核（DMA列表可缓存，使用MMIO寄存器或指令列表代替）。

因此，lazily unpining内存是实施高效RDMA的关键。意味着在转移完成后，任然保持内存锁定状态。这利用了这样的事实：相同的内存区域将用于接下来的DMA转移，因此lazy unpinning接收了锁定/解锁操作的成本。

当BAR空间消耗完了，锁定新的区域将失败。

2.2注册缓冲

见7\_CUDALibraries/cuHook，展示如何调用CUDA API，用来诊断GPU内存的de/allocations。

2.3 解锁callback

2.4支持的系统

用lspci检查PCI拓扑：lspci -t

2.5 PCI BAR大小

## 3 如何实施具体任务

3.1 显示GPU BAR空间

查询被GPUDirect RDMA映射消耗的主要资源：

$ nvidia-smi -q

BAR1 Memory Usage

Total : 256 MiB

Used : 2 MiB

Free : 254 MiB

3.2锁定GPU内存

cuPointerSetAttribute()

3.3解锁GPU内存

nvidia\_p2p\_put\_pages()

## 4 参考

4.1 UVA CUDA内存管理的基础知识

Unified virtual addressing (UVA)是在CUDA 4.0中启用的内存地址管理系统，运行于64位进程的Fermi和Kepler GPU。UVA内存管理提供了GPUDirect RDMA操作的基础。在支持UVA的配置中，当CUDA运行时初始化，应用程序的虚拟地址（VA）范围分解为2个部分：CUDA管理的VA范围和OS还礼的VA范围。所有的CUDA管理的指针都位于该VA范围内，范围将一直位于进程的VA空间的头40个字节。

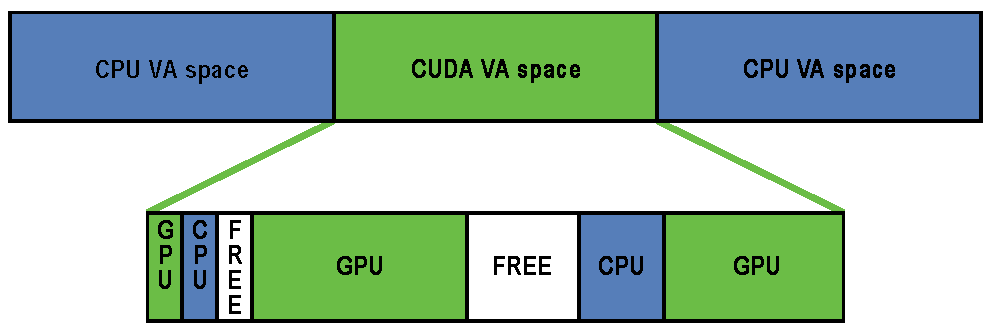


图2 CUDA VA空间地址

随后，在CUDA VA空间内，地址分为3类：

GPU—GPU内存支持的页。主机不能访问，访问中的VA不能在主机上有物理支持。

CPU

FREE