GPU虚拟化

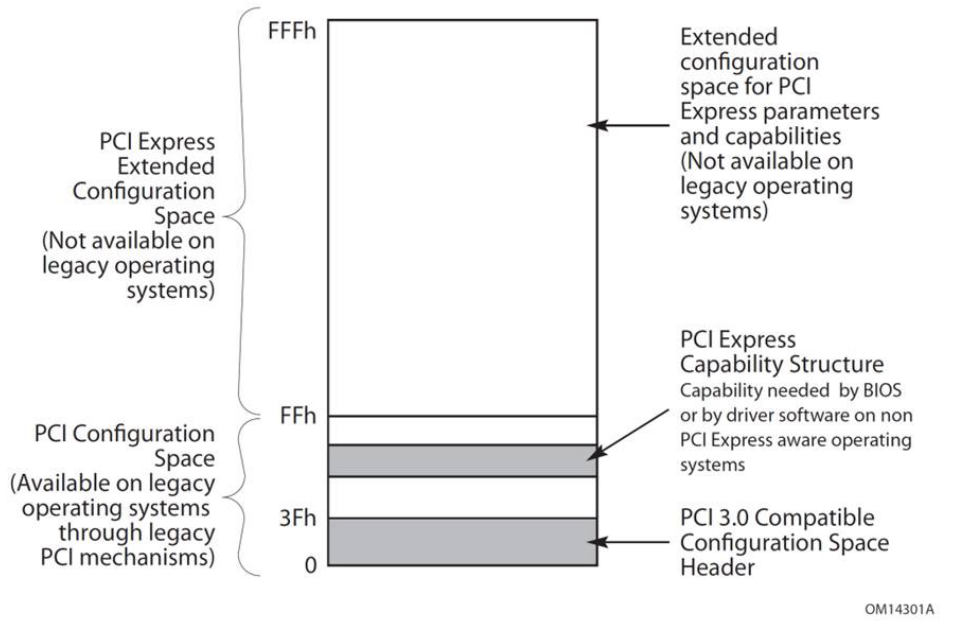
GPU作为PCIE设备的虚拟化

PCIE设备的两种资源：配置空间，MMIO

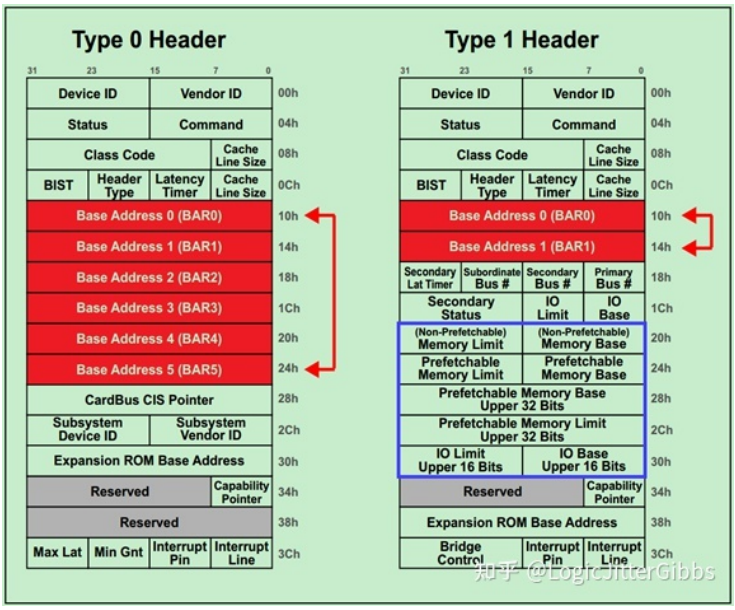
在网卡、显卡这样需要进行大量、快速数据传输的外设中，只有寄存器是不够的，因此这些设备还有一块自己的内存。PCIe架构定义了4种地址空间：配置空间、Memory空间、IO空间和Message空间。

1. 配置空间

   软件可以通过配置空间对设备的状态进行检查和控制。每个PCIe Function都有4KB的配置空间，地址范围为0x000-0xfff。前256B是和PCI兼容的配置空间，剩余的是PCIe扩展配置空间：



基地址寄存器（BAR）在配置空间中的位置如下图所示。其中Type 0 Header最多有六个BAR，Type 1 Header最多有2个BAR。这意味着对于endpoint来说，最多可以拥有6个不同的地址空间。但实际应用中基本用不到，通常1～3个BAR比较常见。



基地址寄存器（BAR）：外设内部的地址都是从0开始的。当PCI控制器接入多个PCI设备时，为了确保PCI上的内存地址不冲突，PCI总线配置BAR寄存器，从而使设备的内存空间和IO空间可用。

每个PCIe设备在BAR中描述自己需要占用多少地址空间，操作系统会据此将合理的基地址写入到相应的BAR寄存器中。对于被使用的BAR来说，其低比特位决定当前BAR支持的操作类型和可申请的地址空间的大小，是read-only的。软件只可以修改BAR的高比特位。

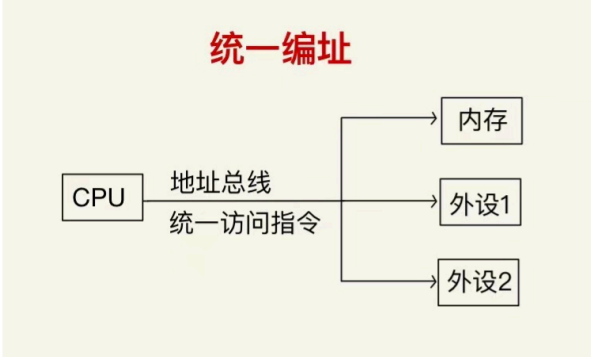
一旦BAR值确定了（have been programmed），其指定范围内的当前设备中的内部寄存器（或内部存储空间）就可以被访问了。当该设备确认某一个请求（request）中的地址在自己的BAR的范围内，便会接受请求并处理。

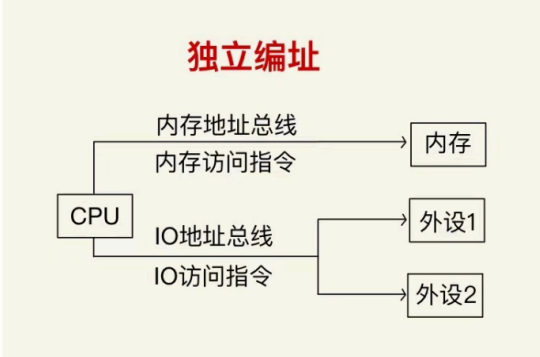
http://blog.chinaaet.com/justlxy/p/5100053328

2. Memory空间和IO空间

早期32位计算机的编址范围小，计算机对I/O空间和内存空间进行独立编址。因此，同一个地址，有可能表示I/O空间中的地址，也有可能表示内存空间中的地址。为了区分，内存操作与I/O操作使用不同的指令。

后来计算机普遍使用64位系统后，内存空间和I/O空间开始不再区分。（为了兼容一些较老的设备和软件，PCIe仍然支持I/O地址空间，只是建议在新开发的软件中采用MMIO。）





> MMIO：Memory Mapping I/O，即内存映射I/O，是PCI规范的一部分，I/O设备被放置在内存空间而不是I/O空间。从处理器的角度来看，内存映射I/O后系统设备访问起来和内存一样，可以使用读写内存的汇编指令完成，简化了程序设计的难度和接口复杂性。
>
> PCIe总线中有两种MMIO：P-MMIO（可预取）和NP-MMIO（不可预取）。P-MMIO有两个特点：读操作不存在副作用（不会改变值）；允许写合并。

3. Message空间

PCIE设备的两种能力：中断能力，DMA能力

一个典型的GPU设备的工作流程：

1. 应用层条用GPU支持的某个API，如OpenGL或CUDA
2. OpenGL或CUDA库，通过UMD（User Mode Driver），提交workload到KMD（Kernel Mode Driver）
3. KMD写CSR MMIO，把它提交给GPU硬件

> CSR：Control and Status Register，属于CPU自带的一类寄存器（与通用数据寄存器区分开）。在机器模式下，只要包括6类：处理器信息相关，中断配置相关，中断响应相关，存储器保护相关，性能统计相关，调试接口相关。
>
> CSR的访问与当前指令，或者说程序处于何种模式相关。不同模式下所能访问的CSR数量都不同。如果强行访问一个本不应该在该模式下访问的CSR，则会触发非法指令异常。

4. GPU硬件开始工作，完成后DMA到内存，发出中断给CPU
5. CPU找到中断处理程序（KMD此前向OS Kernel注册过的）并调用
6. 中断处理程序找到哪个workload被执行完毕，驱动唤醒相关应用

PCIe直通

VT-d：Intel Virualization Technology for Directed I/O。简单来说，就是将PCIe设备的资源直接分配给虚拟机，俗称PCIe直通（passthrough）。

虚拟机会独占这个直通的PCIe设备（1:1），不适合一台宿主机上有很多虚拟机的情况（无法1:N），因此不算真正的虚拟化，无法超卖。

SR-IOV

Single-root input/output virtualization，支持单个物理PCIe设备虚拟出多个虚拟PCIe设备，然后将虚拟PCIe设备直通到各虚拟机，以实现单个物理PCIe设备支撑多虚拟机的应用场景。

SR-IOV协议引入了两种类型功能的概念：物理功能（Physical Function, PF）和虚拟功能（Virtual Function, VF）。

简略版：

每个PF有标准的PCIe功能，能关联到多个VF。而每个VF都有与性能相关的资源，共享一个物理设备。所以就是PF具有完整的PCIe功能，VF能独立使用关键功能。

详细版：

SR-IOV标准允许在虚拟机之间高效共享PCIe（快速外设组件互连）设备，并且它是在硬件中实现的，可以获得能够与本机性能接近的I/O性能。

PF：包含SR-IOV功能的完整PCIe设备。PF作为普通的PCIe设备被发现、管理和配置。PF通过分配VF来配置和管理SR-IOV功能。禁用SR-IOV后，主机将在一个物理网卡上创建一个拥有完全配置或控制PCIe设备资源的能力。

VF：轻量级PCIe功能（I/O处理）的PCIe设备，每个VF都是由PF来生成管理的。VF具体数量限制受限于PCIe设备自身配置及驱动程序的支持。启用SR-IOV后，主机将在一个无力NIC上创建单个PF和多个VF。可以与物理功能以及同一物理功能关联的其他VF共享一个或多个物理资源。

每个SR-IOV设备都可有一个PF，并且每个PF最多有64000个与其关联的VF。PF可以通过寄存器创建VF，这些寄存器设计有专用于此目的的属性。一旦在PF中启用了SR-IOV，就可以通过PF的总线、设备和功能编号（路由ID）访问各个VF的PCI配置空间。

每个VF都有一个PCI内存空间，用于映射其寄存器集。VF设备驱动程序对寄存器集进行操作以启用其功能，并且显示为实际存在的PCI设备。创建VF后，可以直接将其指定给虚拟机或各个应用程序。此功能使得VF可以共享物理设备，并在没有CPU和虚拟机管理程序软件开销的情况下执行I/O。

IOMMU（I/O Memory Management Unit）负责I/O虚拟地址和物理内存地址转换。这样虚拟机就能够使用guest物理地址来对设备编程，通过IOMMU转换成物理主机内存地址。

虚拟机模拟软件VMM不再干预客户机的IO，IOMMU把客户机地址重映射为宿主机物理地址，这样能直接通过DMA在宿主机和VF设备之间进行高速数据搬移，并产生中断。当中断产生的时候，VMM根据中断向量识别出客户机，并将虚拟MSI中断通知给客户机。

PF和VF之间的通信：比如VF把客户机IO请求发给PF，PF也会把一些全局设备重置等事件发给VF。有的设备采用的是Doorbell机制，发送方把消息放入信箱，按一下门铃，产生中断通知接收方，接收方读到消息在共享寄存器做个标记，表示信息接收了。

zhuanlan.zhihu.com/p/630066202

深入理解：

VF有什么？

- 配置空间是虚拟的（特权资源）
- MMIO是物理的
- 中断和DMA，因为VF有自己的PCIe协议层的标识（Routing ID，就是BDF），从而拥有独立的地址空间

那么什么设备适合实现SR-IOV？需要满足：

- 硬件资源容易partition
- 无状态（或接近无状态）

常见PCIe设备中，最适合SR-IOV的就是网卡：一或多对TX/RX queue+一或多个终端，结合上一个Routing ID，就可以抽象为一个VF。而且它是近乎无状态的。

GPU存在的困难：虽然基本是无状态的，但硬件复杂度极高，partition很难实现。

API转发

在软件层面实现“GPU虚拟化”。以AWS Elastic GPU为例：

- VM中看不到真的或假的GPU，但可以调用OpenGL API进行渲染
- 在OpenGL API层，软件捕捉到该调用，转发给Host

- Host请求GPU进行渲染
- Host把渲染结果转发给VM

优点：灵活，不依赖于GPU厂商，不限于系统虚拟化环境

缺点：复杂度高，功能不完整

MPT (Mediated Pass-Through) 受控直通

是nVidia GRID vGPU、Intel GVT-g（KVMGT、XenGT）的实现思路

基本思路：

- 敏感资源，如配置空间，是虚拟的
- 关键资源，如MMIO中CSR部分，是虚拟的
- 性能关键资源，如MMIO中其他部分，硬件partition后直接分给VM
- Host上必须存在一个virtualization-aware的驱动程序，负责模拟和调度，实际上是vGPU的device-model

优点：1:N灵活性，高性能，渲染计算媒体的功能完整性

缺点：调试困难，必须有一个pGPU驱动，负责vGPU的模拟和调度工作

Compare

	功能完整性	性能	多租户	独立于厂商
PF直通	Y	Y	N	Y
API Forwarding	N	N~Y	Y	Y
MPT (GRID vGPU/AMD SRIOV)	Y	Y	Y	N