PCI Peer-to-Peer DMA Support

The PCI bus has pretty decent support for performing DMA transfers between two devices on the bus. This type of transaction is henceforth called Peer-to-Peer (or P2P). However, there are a number of issues that make P2P transactions tricky to do in a perfectly safe way.

One of the biggest issues is that PCI doesn't require forwarding transactions between hierarchy domains, and in PCIe, each Root Port defines a separate hierarchy domain. To make things worse, there is no simple way to determine if a given Root Complex supports this or not. (See PCIe r4.0, sec 1.3.1). Therefore, as of this writing, the kernel only supports doing P2P when the endpoints involved are all behind the same PCI bridge, as such devices are all in the same PCI hierarchy domain, and the spec guarantees that all transactions within the hierarchy will be routable, but it does not require routing between hierarchies.

The second issue is that to make use of existing interfaces in Linux, memory that is used for P2P transactions needs to be backed by struct pages. However, PCI BARs are not typically cache coherent so there are a few corner case gotchas with these pages so developers need to be careful about what they do with them.

Driver Writer's Guide

In a given P2P implementation there may be three or more different types of kernel drivers in play:

- Provider - A driver which provides or publishes P2P resources like memory or doorbell registers to other drivers.
- Client - A driver which makes use of a resource by setting up a DMA transaction to or from it.
- Orchestrator - A driver which orchestrates the flow of data between clients and providers.

In many cases there could be overlap between these three types (i.e., it may be typical for a driver to be both a provider and a client).

For example, in the NVMe Target Copy Offload implementation:

- The NVMe PCI driver is both a client, provider and orchestrator in that it exposes any CMB (Controller Memory Buffer) as a P2P memory resource (provider), it accepts P2P memory pages as buffers in requests to be used directly (client) and it can also make use of the CMB as submission queue entries (orchestrator).
- The RDMA driver is a client in this arrangement so that an RNIC can DMA directly to the memory exposed by the NVMe device.
- The NVMe Target driver (nvmet) can orchestrate the data from the RNIC to the P2P memory (CMB) and then to the NVMe device (and vice versa).

This is currently the only arrangement supported by the kernel but one could imagine slight tweaks to this that would allow for the same functionality. For example, if a specific RNIC added a BAR with some memory behind it, its driver could add support as a P2P provider and then the NVMe Target could use the RNIC's memory instead of the CMB in cases where the NVMe cards in use do not have CMB support.

Provider Drivers

A provider simply needs to register a BAR (or a portion of a BAR) as a P2P DMA resource using :c:func:`pci_p2pdma_add_resource()`. This will register struct pages for all the specified memory.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\pci\[linux-master][Documentation][driver-api][pci]p2pdma.rst`, line 69); *backlink*

Unknown interpreted text role "c:func".

After that it may optionally publish all of its resources as P2P memory using :c:func:`pci_p2pmem_publish()`. This will allow any orchestrator drivers to find and use the memory. When marked in this way, the resource must be regular memory with no side effects.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\pci\[linux-master][Documentation][driver-api][pci]p2pdma.rst`, line 73); *backlink*

Unknown interpreted text role "c:func".

For the time being this is fairly rudimentary in that all resources are typically going to be P2P memory. Future work will likely expand this to include other types of resources like doorbells.

Client Drivers

A client driver typically only has to conditionally change its DMA map routine to use the mapping function

:c:func:`pci_p2pdma_map_sg()` instead of the usual :c:func:`dma_map_sg()` function. Memory mapped in this way does not need to be unmapped.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\pci\[linux-master][Documentation][driver-api][pci]p2pdma.rst`, line 86); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\pci\[linux-master][Documentation][driver-api][pci]p2pdma.rst`, line 86); *backlink*

Unknown interpreted text role "c:func".

The client may also, optionally, make use of :c:func:`is_pci_p2pdma_page()` to determine when to use the P2P mapping functions and when to use the regular mapping functions. In some situations, it may be more appropriate to use a flag to indicate a given request is P2P memory and map appropriately. It is important to ensure that struct pages that back P2P memory stay out of code that does not have support for them as other code may treat the pages as regular memory which may not be appropriate.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\pci\[linux-master][Documentation][driver-api][pci]p2pdma.rst`, line 91); *backlink*

Unknown interpreted text role "c:func".

Orchestrator Drivers

The first task an orchestrator driver must do is compile a list of all client devices that will be involved in a given transaction. For example, the NVMe Target driver creates a list including the namespace block device and the RNIC in use. If the orchestrator has access to a specific P2P provider to use it may check compatibility using :c:func:`pci_p2pdma_distance()` otherwise it may find a memory provider that's compatible with all clients using :c:func:`pci_p2pmem_find()`. If more than one provider is supported, the one nearest to all the clients will be chosen first. If more than one provider is an equal distance away, the one returned will be chosen at random (it is not an arbitrary but truly random). This function returns the PCI device to use for the provider with a reference taken and therefore when it's no longer needed it should be returned with pci_dev_put().

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\pci\[linux-master][Documentation][driver-api][pci]p2pdma.rst`, line 104); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\pci\[linux-master][Documentation][driver-api][pci]p2pdma.rst`, line 104); *backlink*

Unknown interpreted text role "c:func".

Once a provider is selected, the orchestrator can then use :c:func:`pci_alloc_p2pmem()` and :c:func:`pci_free_p2pmem()` to allocate P2P memory from the provider. :c:func:`pci_p2pmem_alloc_sgl()` and :c:func:`pci_p2pmem_free_sgl()` are convenience functions for allocating scatter-gather lists with P2P memory.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\pci\[linux-master][Documentation][driver-api][pci]p2pdma.rst`, line 118); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\pci\[linux-master][Documentation][driver-api][pci]p2pdma.rst`, line 118); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\pci\[linux-master][Documentation][driver-api][pci]p2pdma.rst, line 118); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\pci\[linux-master][Documentation][driver-api][pci]p2pdma.rst, line 118); *backlink*

Unknown interpreted text role "c:func".

Struct Page Caveats

Driver writers should be very careful about not passing these special struct pages to code that isn't prepared for it. At this time, the kernel interfaces do not have any checks for ensuring this. This obviously precludes passing these pages to userspace.

P2P memory is also technically IO memory but should never have any side effects behind it. Thus, the order of loads and stores should not be important and ioreadX(), iowriteX() and friends should not be necessary.

P2P DMA Support Library

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\pci\[linux-master][Documentation][driver-api][pci]p2pdma.rst, line 140)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/pci/p2pdma.c
   :export:
```