Why is Peer to Peer DMA so Challenging on Linux?

Alex Deucher

What is Peer to Peer DMA and how does it work?

- Peer to Peer access via PCI BARs
  - Devices access remote PCI BARs directly just like system memory
  - Large or Resizeable BARs required for devices with large amounts of device memory
- Vendor specific interconnects
  - XGMI (AMD)
  - NVLink® (Nvidia)

Why would you want peer to peer DMA?

- Saves a trip through system memory
- Processing pipelines without CPU overhead or synchronization

In the beginning

- There was no IOMMU
  - Still largely the case on Windows
- PCIe chipsets maybe did not support it
  - No way to tell (still no official way today)

Out of tree solutions dominate

- Let’s pass around system physical addresses
- RDMA
  - Mellanox (GPUDirect™/PeerDirect™)
- GPU
  - AMD (DirectGMA, GPUDirect™/PeerDirect™)
  - Nvidia (CUDA®, GPUDirect™/PeerDirect™)
- HPC and workstation use cases dominate
- Large entrenched user bases
- Windows didn’t use IOMMU until recently
- Not upstreamable

In tree: P2P PCI

- Designed around NVMe fabrics offload
- Currently only supported in NVMe and RDMA drivers
- Programming model reflects use case
  - Providers – Expose P2P resources to other drivers
  - Clients – Make use of P2P resources via DMA
  - Orchestrators – Enables data flow between clients and providers
- Memory is allocated on the client side
- Resource is backed by struct pages (ZONE_DEVICE)
  - Client uses pci_p2pdma_map_sg()
- Assumes all memory is pinned
- https://www.kernel.org/doc/html/latest/driver-api/pci/p2pdma.html

In tree: P2P PCI

- Provider
  - Registers BAR (or portion of BAR) which provides struct pages for the memory
- Client
  - Allocates the memory from the Provider
  - Just calls pci_p2pdma_map_sg() instead of dma_map_sg() if the memory is device memory
- Orchestrator
  - Compiles a list of compatible clients involved in the transaction and allocates/frees memory from the provider(s)

In tree: dma-buf

- Framework for sharing buffers for hardware (DMA) access across drivers
- Provides synchronization mechanisms as well
  - Allows for unpinned memory to be shared using dma fences
- Originally system memory buffers only; expanded to cover device resources
- Easy for sharing buffers across applications using fds
- Used by GPU and V4L drivers
- Programming reflects use cases
  - Exporter – allocates the memory and shares
  - Importer – accesses the shared buffer
  - One exporter, multiple importers
- Memory allocated on the exporter side
- No struct page backing for resource
  - Importer uses dma_map_resource()
- https://www.kernel.org/doc/html/latest/driver-api/dma-buf.html

In tree: dma-buf

- Exporter registers callbacks to the dmabuf object
- Optional importer callbacks for peer to peer support
- Supports unpinned buffers via move_notify callback
  - Exporter will call importer’s move_notify callback when it moves a resource
  - Invalidates the attachment and recreates the mapping for the next use

In tree: HMM

- Heterogeneous Memory Management
- Not exactly a peer to peer sharing system
- Provides a mechanism for a shared virtual address space for multiple devices (CPU, GPU, FPGA, etc.)
- Device resources are backed by struct pages (ZONE_DEVICE), but only DEVICE_PRIVATE so no CPU access
- Page table mirroring between CPU and devices
- Requires devices capable of recoverable page faults or preemption in its current form
- https://www.kernel.org/doc/html/v4.18/vm/hmm.html

Major use cases: GPU + RDMA (+ NVMe)

- Each uses a different P2P framework
- GPUs can’t afford to reserve and pin device memory; it’s too precious
  - Need to be able to move and invalidate under memory pressure
  - On modern systems with large/resizable BAR, there may be more VRAM then system ram…
- RDMA drivers don’t support dma_fence synchronization objects
  - Assume memory will be pinned
- NVMe drivers want to allocate the memory on the peer
- Can we make it work?
  - Much easier for RDMA with ODP (On Demand Paging) support
  - Maybe we can pin VRAM on GPUs with large/resizeable BAR

Questions?

DISCLAIMER AND ATTRIBUTIONS

Disclaimer:

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD


AMD