ECE408 Fall 2023

Applied Parallel Programming

Lecture 21:
GPU as part of the PC Architecture

Objective

- To understand the major data transfer factors that dictate performance when using GPU as a compute co-processor for the CPU
  - The speeds and feeds of the traditional CPU world
  - The speeds and feeds when employing a GPU
  - To form a solid knowledge base for performance programming in modern GPU's

Common Structure of a CUDA Program

```
Global variables declaration

Function prototypes
    __global__ void kernelOne(…)

Main ()
    – allocate memory space on the device – cudaMalloc(&d_GlblVarPtr, bytes )
    – transfer data from host to device – cudaMemcpy(d_GlblVarPtr, h_Gl…)
    – execution configuration setup
    – kernel call – kernelOne<<<execution configuration>>>( args… )
    – transfer results from device to host – cudaMemcpy(h_GlblVarPtr,…)
    – optional: compare against golden (host computed) solution
•
```

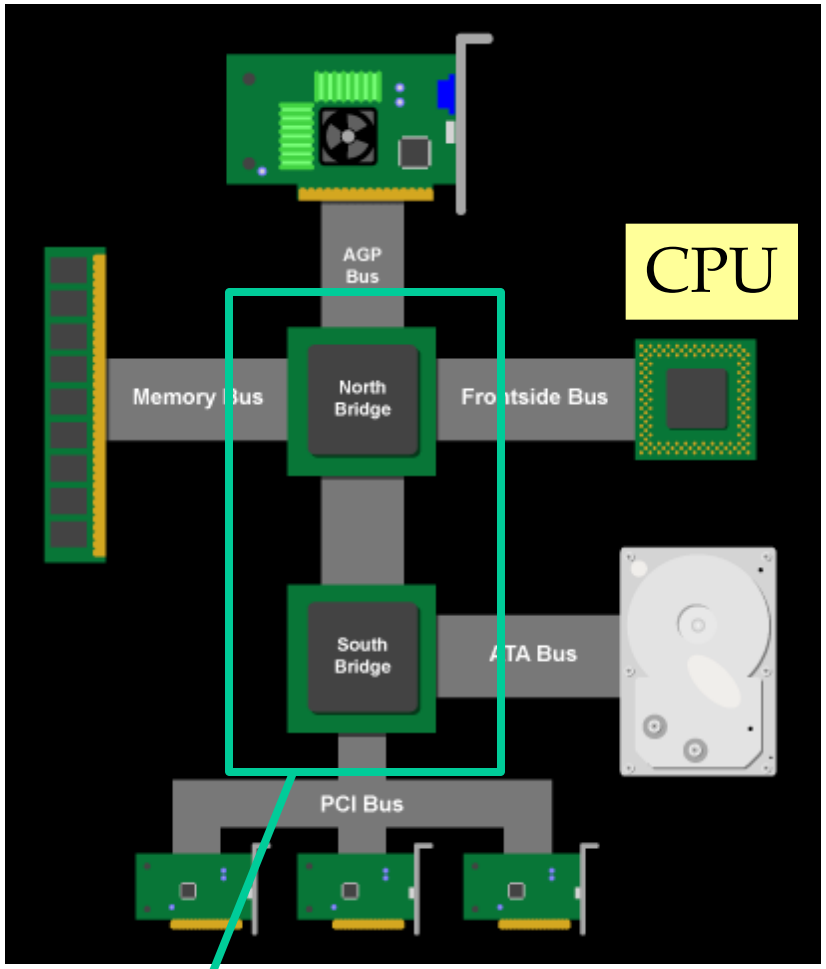
Repeat as needed

Data Transfer Bandwidth

- The Bandwidth between key components ultimately dictates system performance
  - Especially true for massively parallel systems processing massive amount of data
  - Tricks like buffering, reordering, caching can temporarily defy the rules in some cases
  - Ultimately, the performance falls back to what the bandwidths dictate

Classic PC Architecture

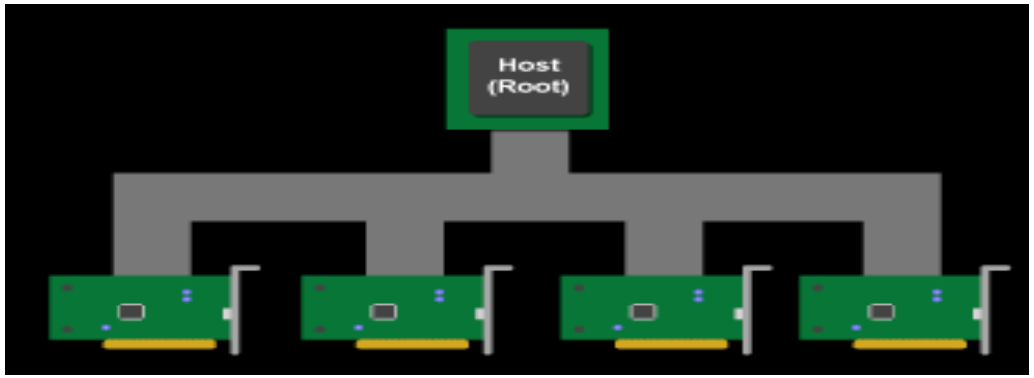
- Northbridge connects 3 components that must communicate at high speed
  - CPU, DRAM, video
  - Video also needs to have 1st-class access to DRAM
  - Previous NVIDIA cards are connected to AGP, up to 2 GB/s transfers
- Southbridge serves as a concentrator for slower I/O devices



Core Logic Chipset

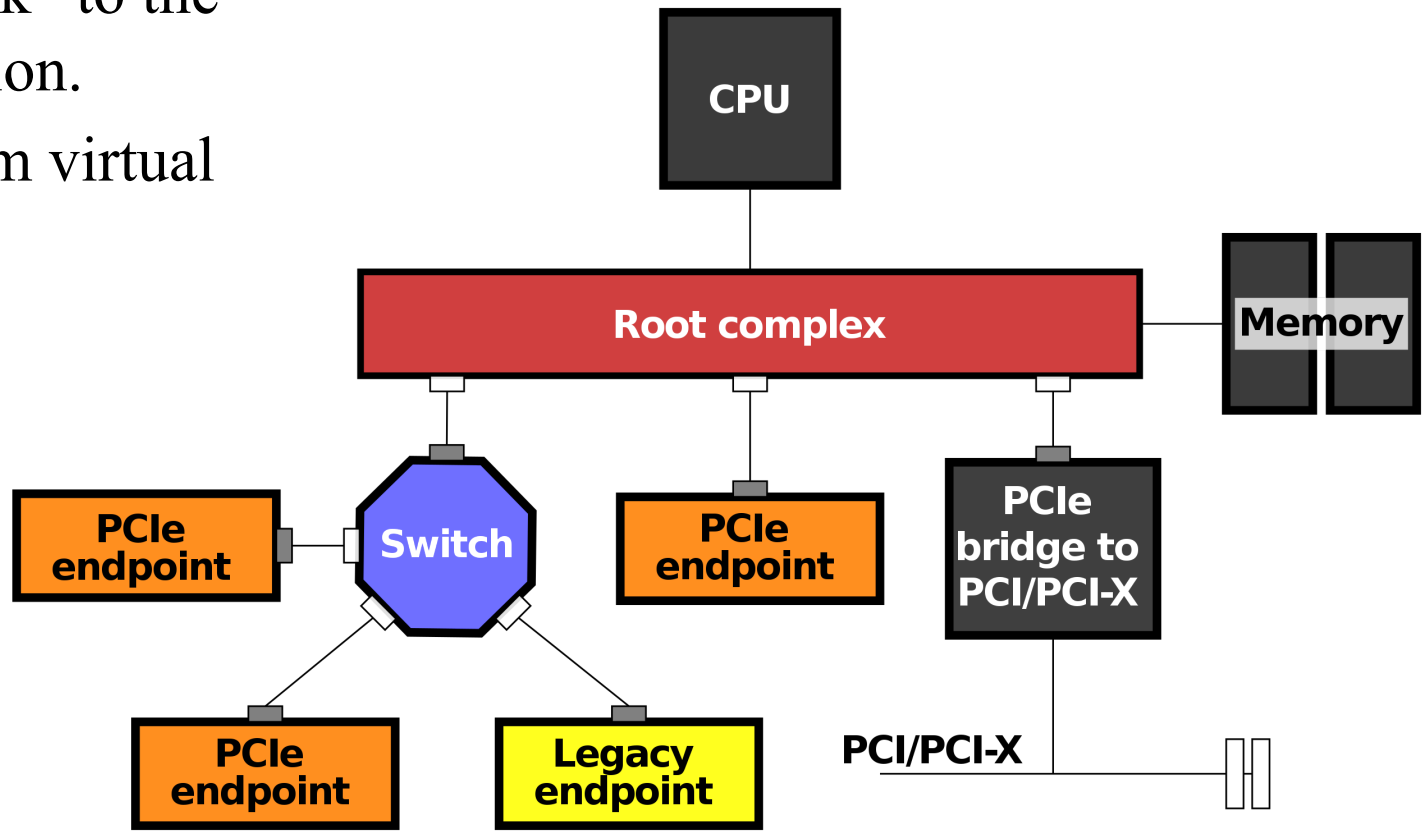
Original PCI Bus Specification:1992
A Humble Beginning

- Connected to the SouthBridge
  - Originally 33 MHz, 32-bit wide, 132 MB/second peak transfer rate
  - Later, 66 MHz, 64-bit, 528 MB/second peak
  - Upstream bandwidth remain slow for device (~256MB/s peak)
  - Shared bus with arbitration
    - Winner of arbitration becomes bus master and can connect to CPU or DRAM through the Southbridge and Northbridge



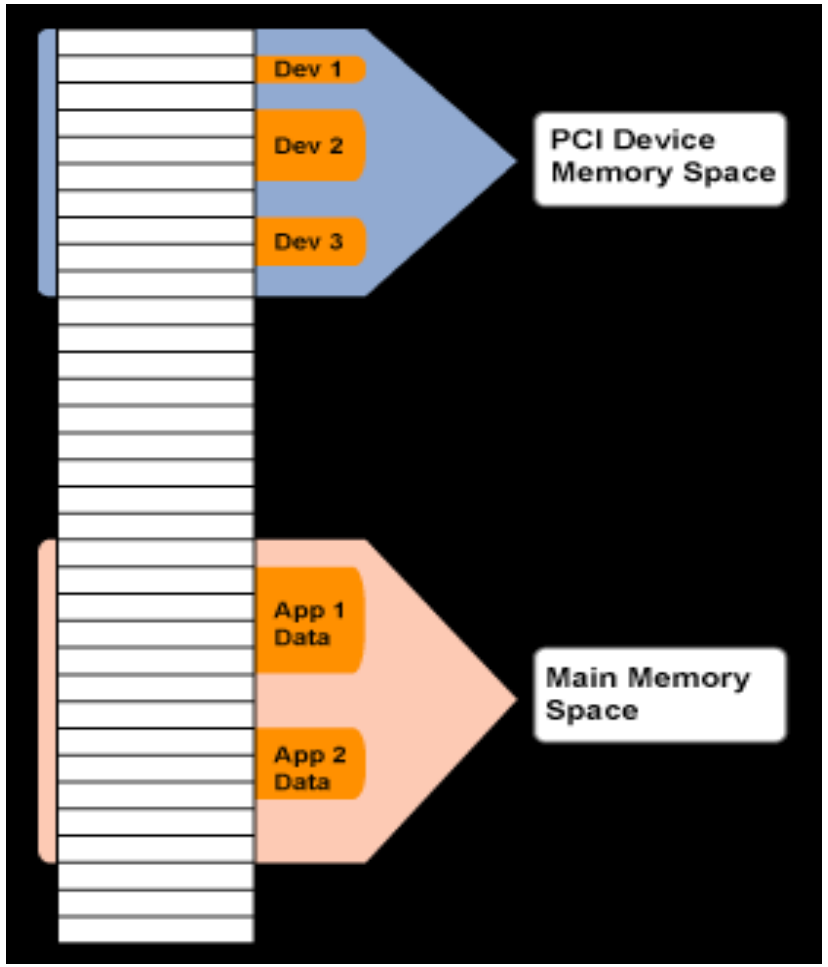
PCI Express or PCIe: 2003

- Switched, point-to-point connection
  - Each card has a dedicated “link” to the central switch, no bus arbitration.
  - Packet switches messages form virtual channel
  - Prioritized packets for QoS
    - E.g., real-time video streaming



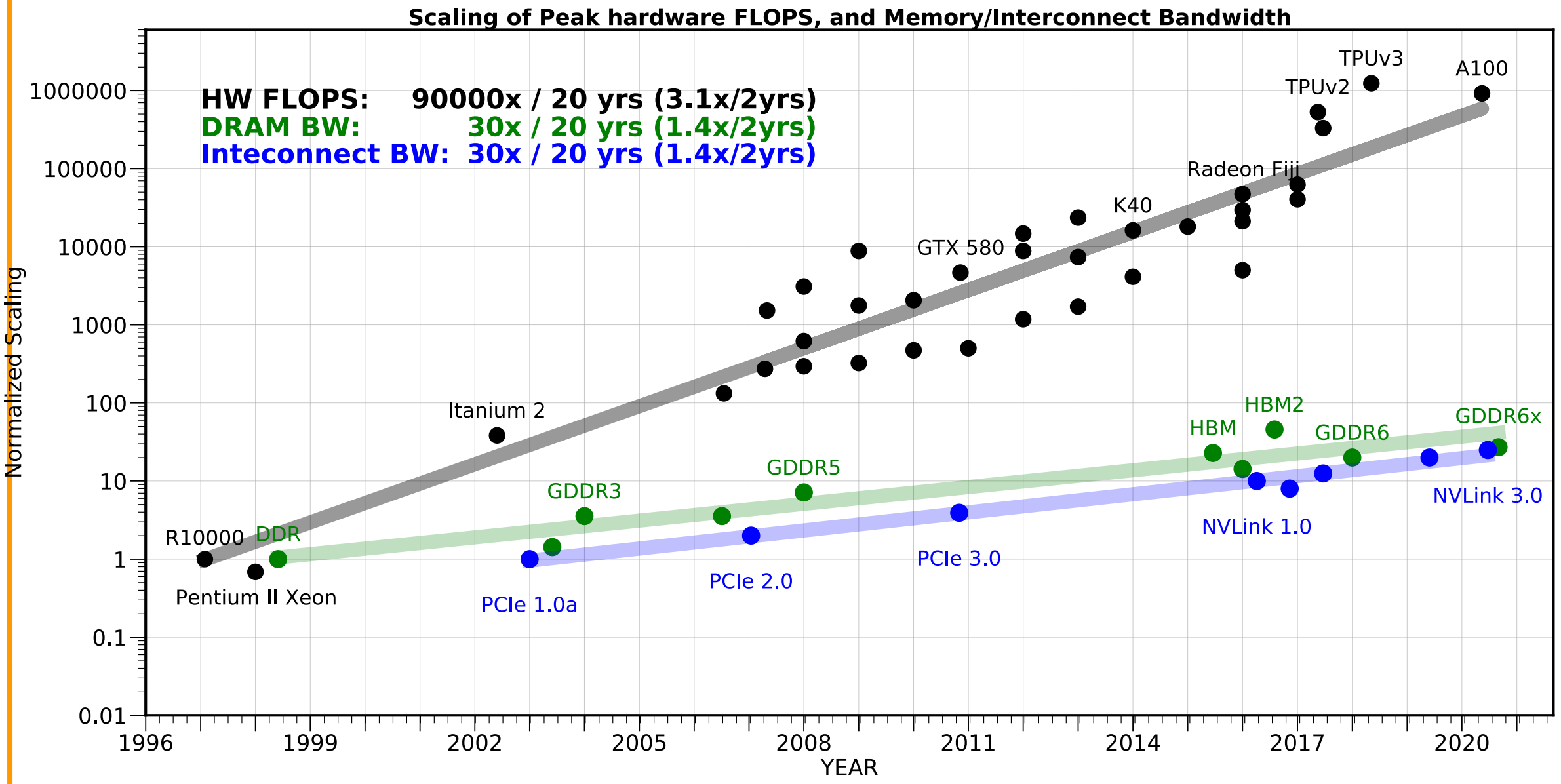
PCIe as Memory Mapped I/O

- PCIe device registers are mapped into the CPU's physical address space
  - Accessed through loads/ stores (kernel mode)
- Addresses are assigned to the PCIe devices at boot time
  - All devices listen for their addresses



PCIe Generations

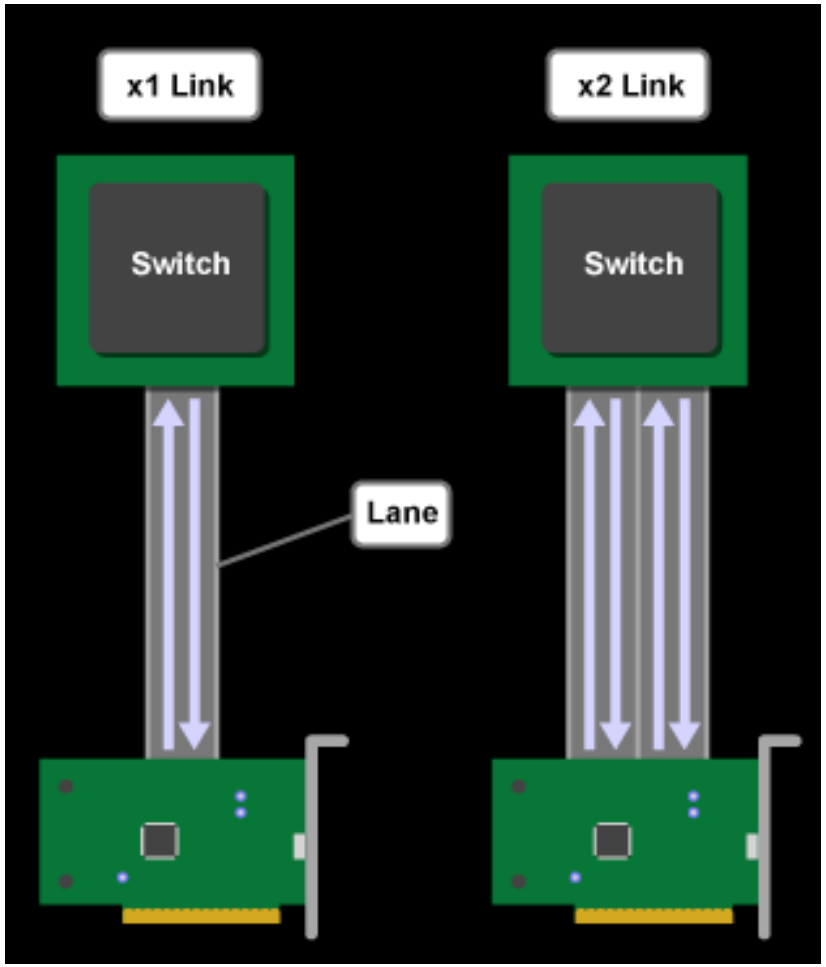
- Within each generation, the links can be scaled by making it wider
  - x1, x2, x4, x8, x16,…
- The goal is to double the speed with each new generation
- Current generation is PCIe 6.0 (2022). Transfer rate for x16 configuration is 256GB/s. For comparison, high-end memory bandwidth is 3TB/s.



Credit: Amir Gholami, et al, RiseLab Blog, March 2021

PCIe Gen 6 Lanes

- Each PCIe link consists of one or more lanes
  - Each lane is 1-bit wide (4 wires, each 2-wire pair can transmit 8GB/s in one direction)
    - Upstream and downstream simultaneous and symmetric
  - Each Link can combine 1, 2, 4, 8, 12, 16 lanes
  - Data is **128b/130b** encoded into 130 bits with equal number of 1's and 0's; net data rate 7.8768 GB/s per lane each way.
  - Peak data rates are 8GB/s (x1) 16 GB/s (x2), 32 GB/s (x4), 64 GB/s (x8), 128 GB/s (x16), each way
  - PCIe 6.0 doubles the data rate over PCIe 5.0



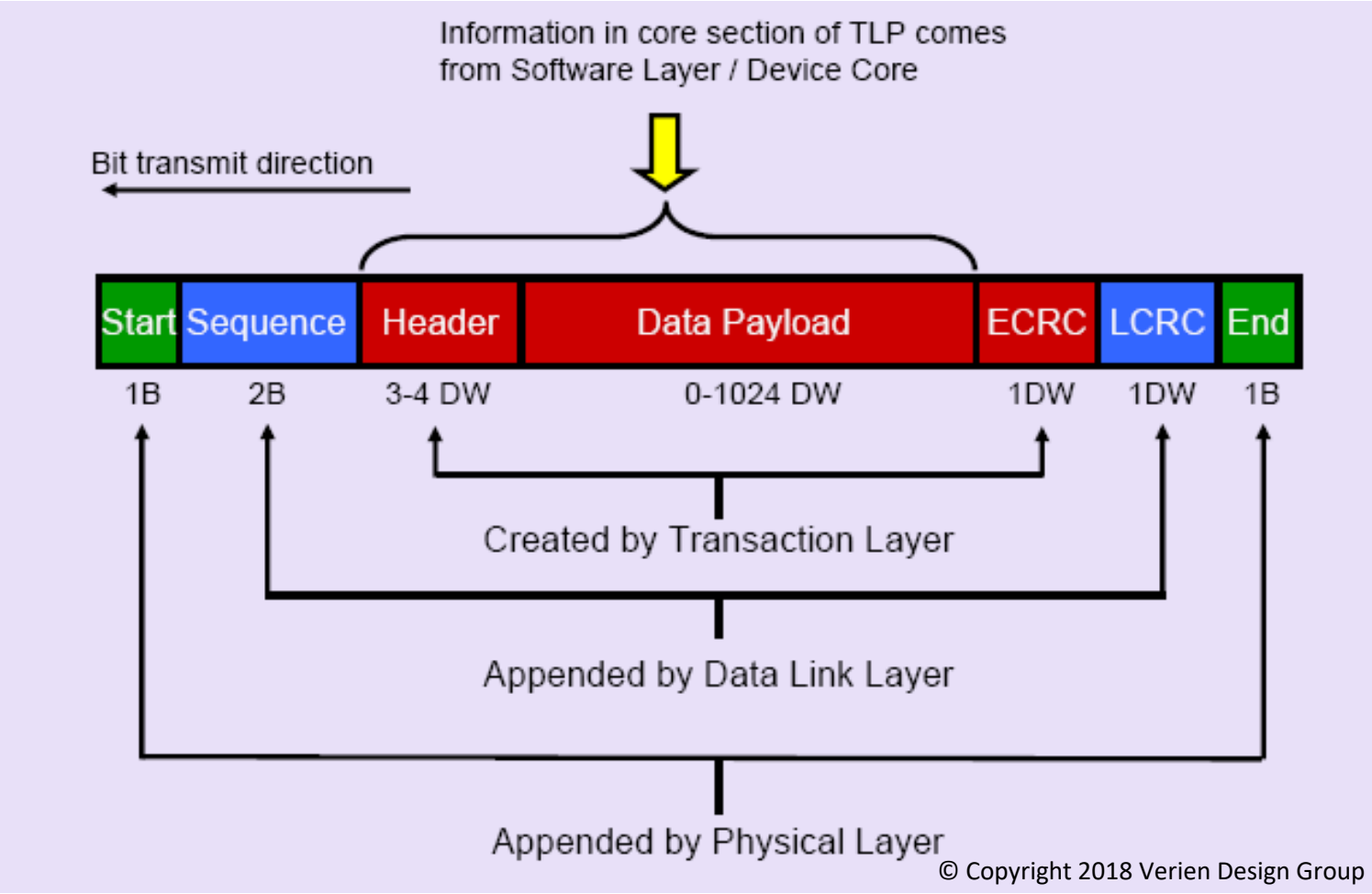
Foundation: 8/10 bit encoding

- Goal is to maintain DC balance while have sufficient state transition for clock recovery
- The difference of 1s and 0s in a 20-bit stream should be ≤ 2 AND there should be no more than 5 consecutive 1s or 0s in any stream
- 00000000, 00000111, 11000001 bad
- 01010101, 11001100 good
- Find 256 good patterns among 1024 total patterns of 10 bits to encode an 8-bit data
- An 20% overhead

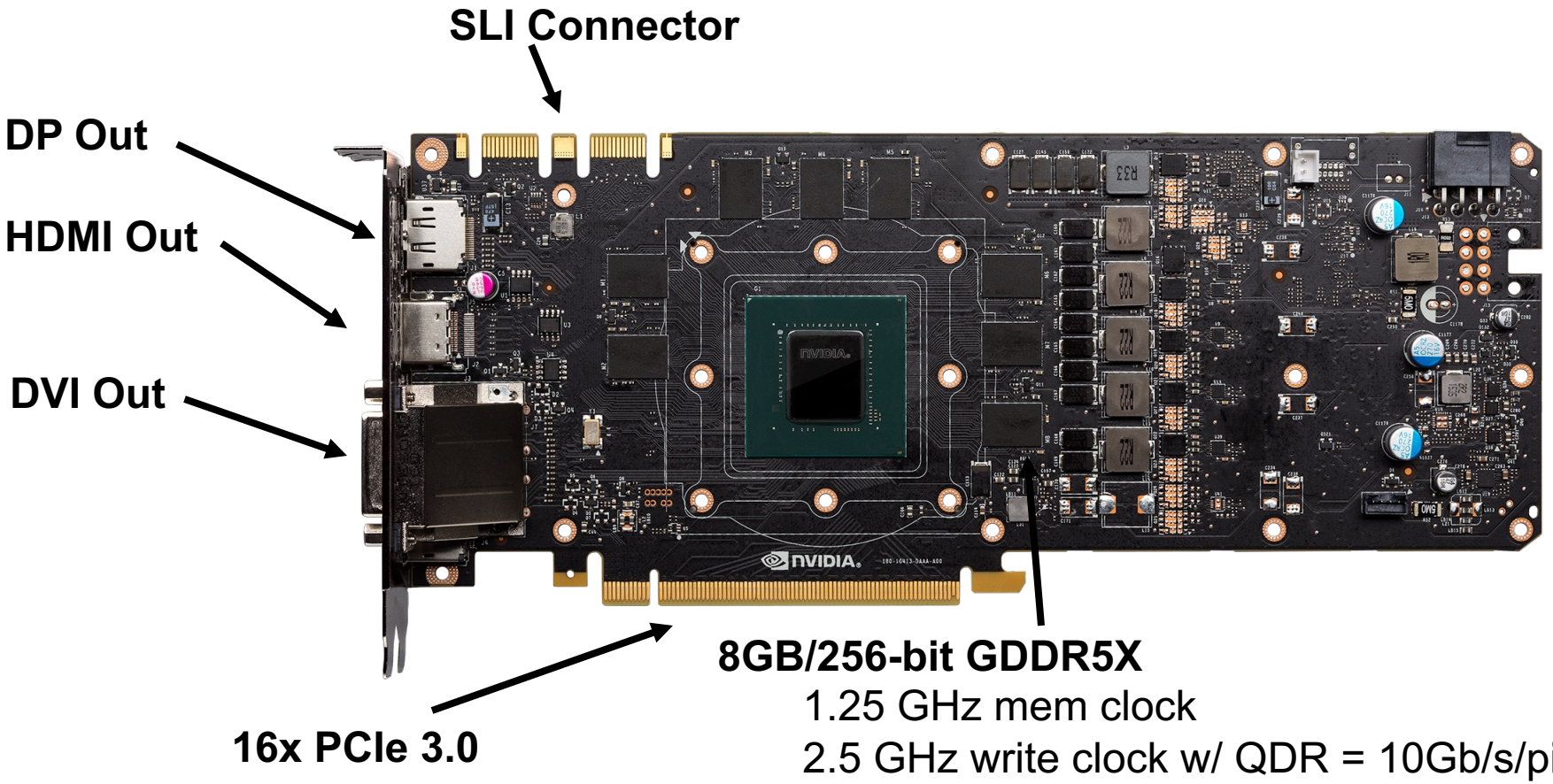
Current: 128/130 bit encoding

- Same goal: maintain DC balance while have sufficient state transition for clock recovery
- 1.5% overhead instead of 20%
- Scrambler function: long runs of 0s, 1s vanishingly small
- Different than guaranteed run length of 8/10b

PCIe Packet Format

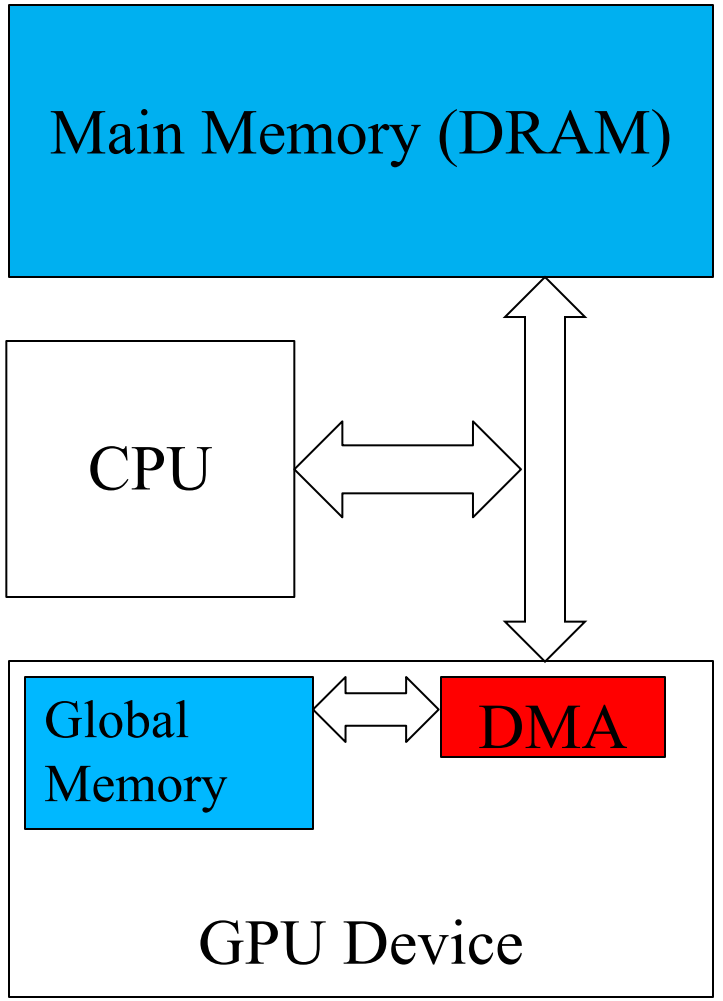


GeForce GTX 1080 (Pascal Architecture, 2016)



PCIe Data Transfer using DMA

- DMA (Direct Memory Access) is used to fully utilize the bandwidth of an I/O bus
  - DMA uses physical address for source and destination
  - Transfers a number of bytes requested by OS
  - Needs pinned memory



Pinned Memory

- DMA uses physical addresses
- The OS could accidentally page out the data that is being read or written by a DMA and page in another virtual page into the same location
- Pinned memory cannot not be paged out
- If a source or destination of a `cudaMemcpy()` in the host memory is not pinned, it needs to be first copied to a pinned memory – extra overhead
- `cudaMemcpy()` is much faster with pinned host memory source or destination

Allocate/Free Pinned Memory

- `cudaHostAlloc()`
  - Three parameters
  - Address of pointer to the allocated memory
  - Size of the allocated memory in bytes
  - Option – use cudaHostAllocDefault for now

- `cudaFreeHost()`
  - One parameter
  - Pointer to the memory to be freed

Using Pinned Memory

- Allocated memory and pointer used the same way those returned by `malloc()`
- Allocated memory cannot be paged by the OS
- `cudaMemcpy` should be about 2X faster with pinned memory
- Pinned memory is a limited resource whose over-subscription can have serious consequences

ANY MORE QUESTIONS?