CS/ECE 217

GPU Architecture and Parallel Programming

Lecture 16: GPU within a computing system

Objective

- To understand the major factors that dictate performance when using GPU as an 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

Review- Typical 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

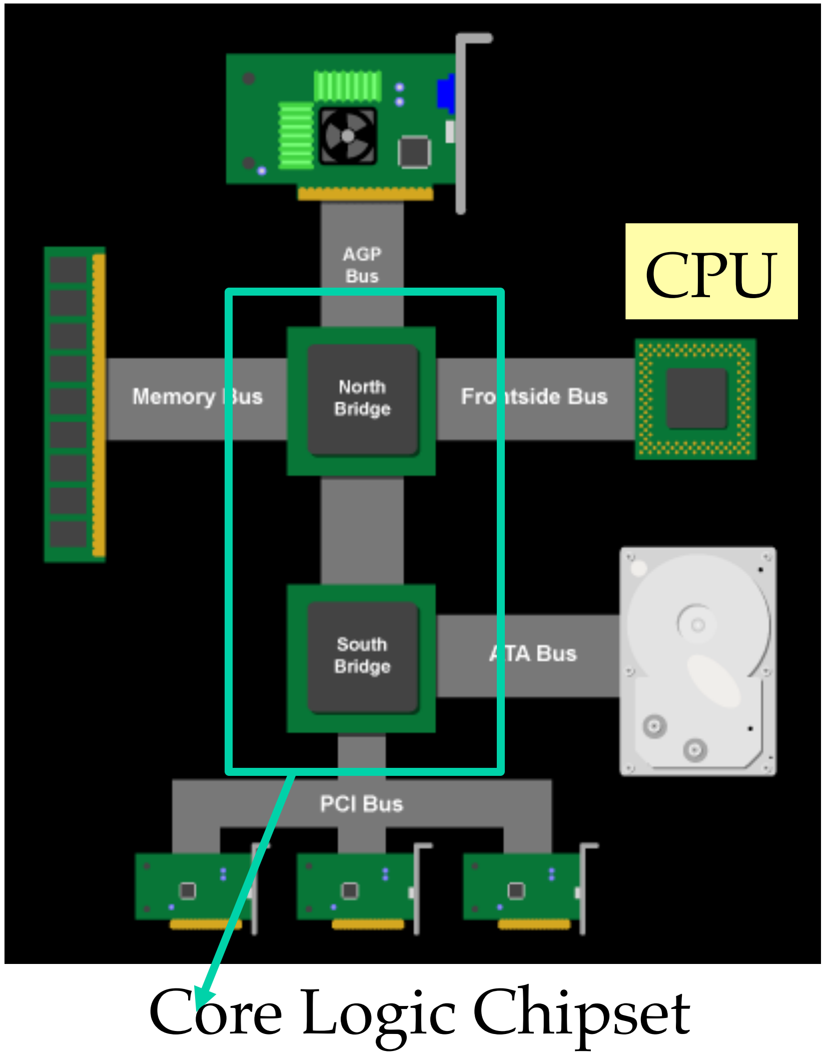
- Kernel – void kernelOne(type args,…)
  - variables declaration - __local__, __shared__
    - automatic variables transparently assigned to registers or local memory
  - syncthreads()…

Bandwidth – Gravity of Modern Computer Systems

- 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 "speeds and feeds" dictate

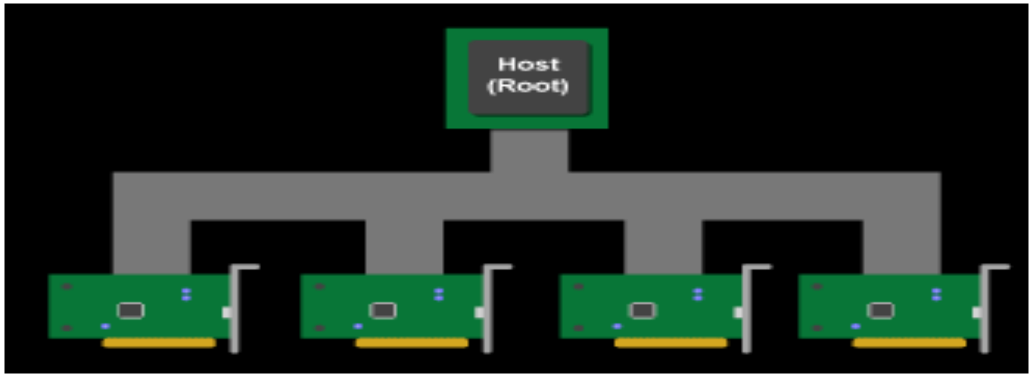
Classic PC architecture

- Northbridge connects 3 components that must be 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



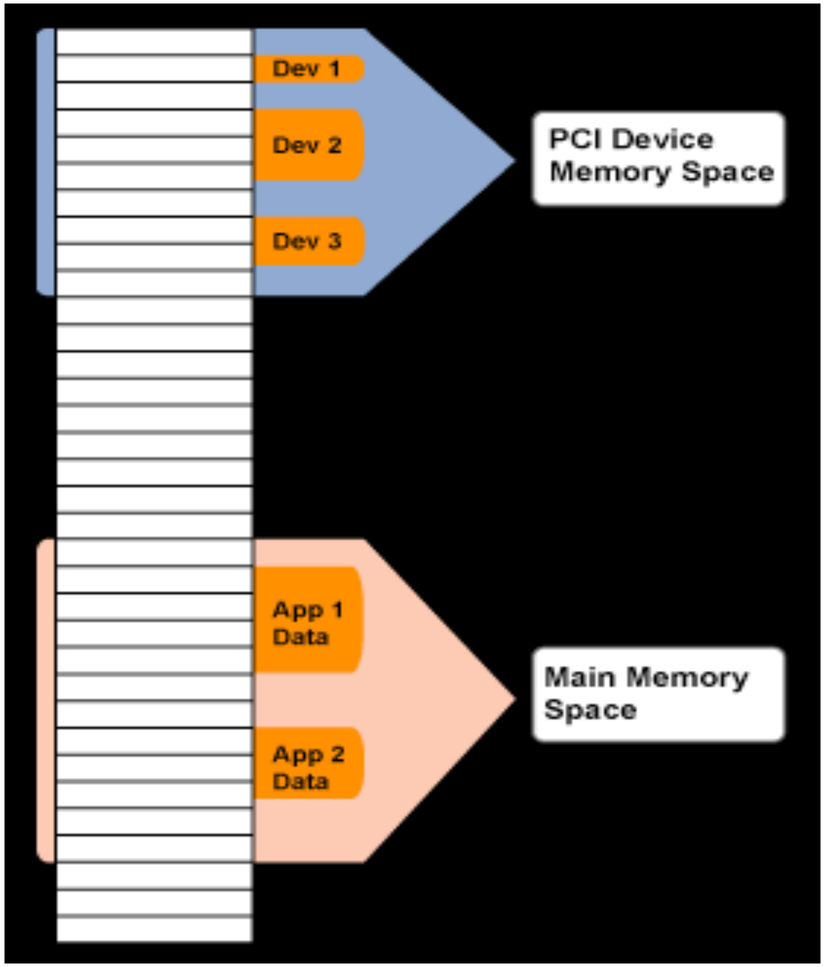
(Original) PCI Bus Specification

- Connected to the southBridge
  - Originally 33 MHz, 32-bit wide, 132 MB/second peak transfer rate
  - More recently 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



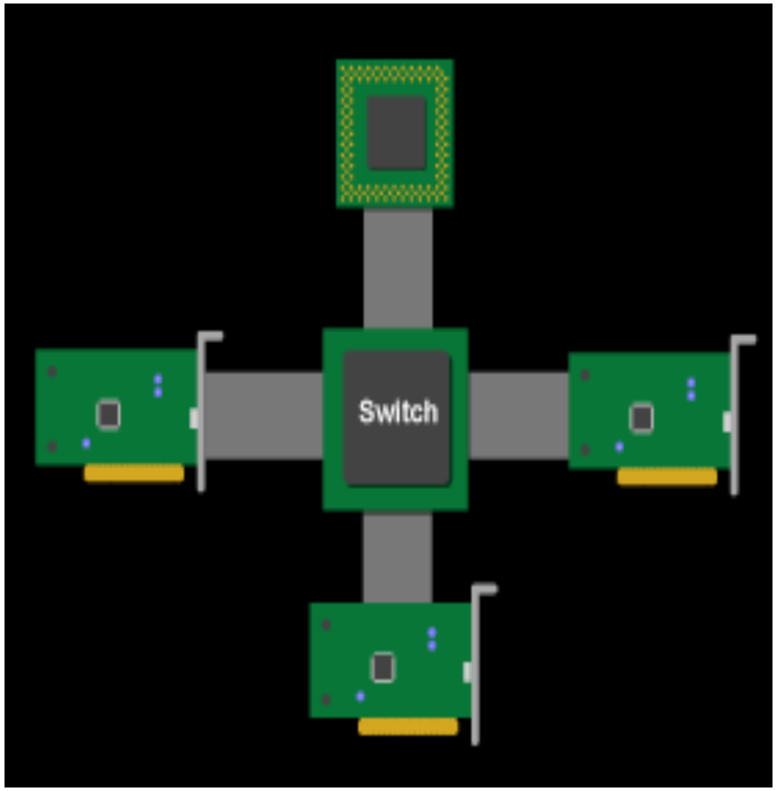
PCI as Memory Mapped I/O

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



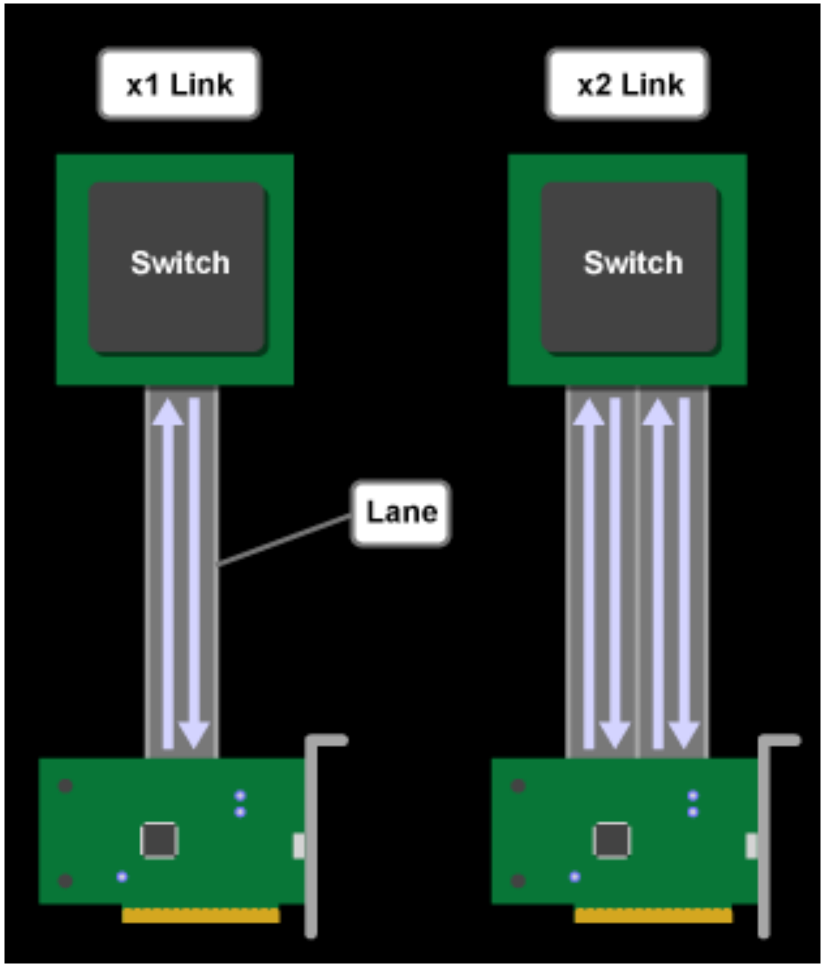
PCI Express (PCIe)

- 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 2 Links and Lanes

- Each link consists of one or more lanes
  - Each lane is 1-bit wide (4 wires, each 2-wire pair can transmit 2.5Gb/s in one direction)
    - Upstream and downstream now simultaneous and symmetric
  - Each Link can combine 1, 2, 4, 8, 12, 16 lanes- x1, x2, etc.
  - Each byte data is **8b/10b** encoded into 10 bits with equal number of 1's and 0's; net data rate 2 Gb/s per lane each way.
  - Thus, the net data rates are 250 MB/s (x1) 500 MB/s (x2), 1GB/s (x4), 2 GB/s (x8), 4 GB/s (x16), each way

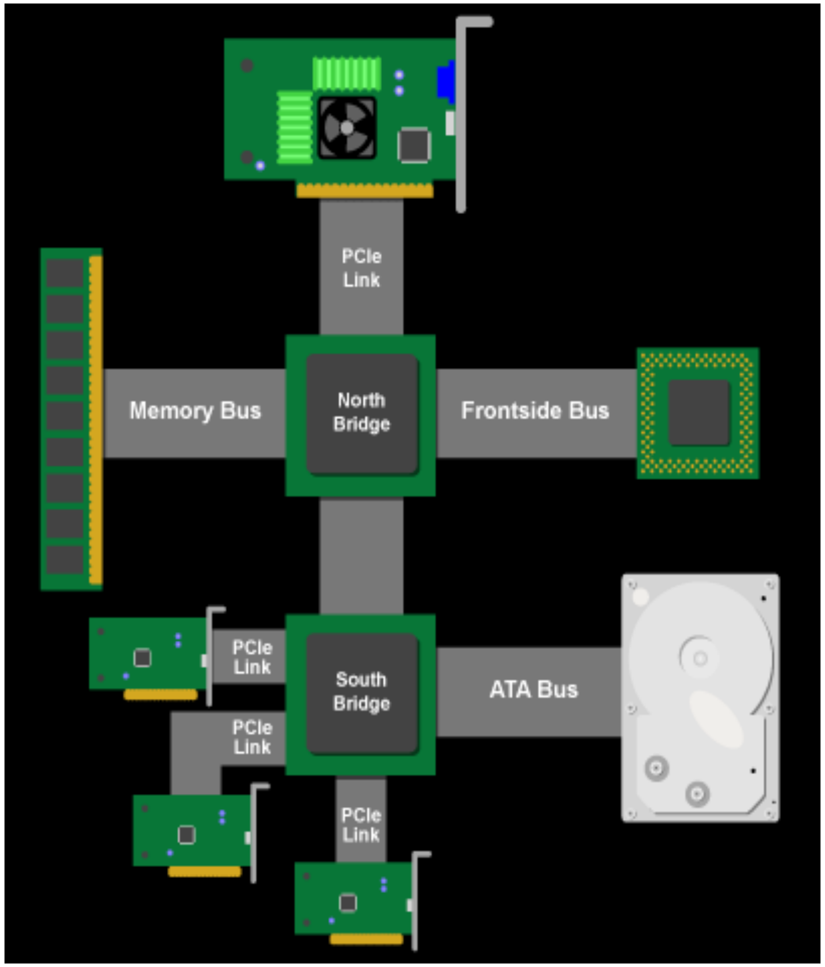


8/10 bit encoding

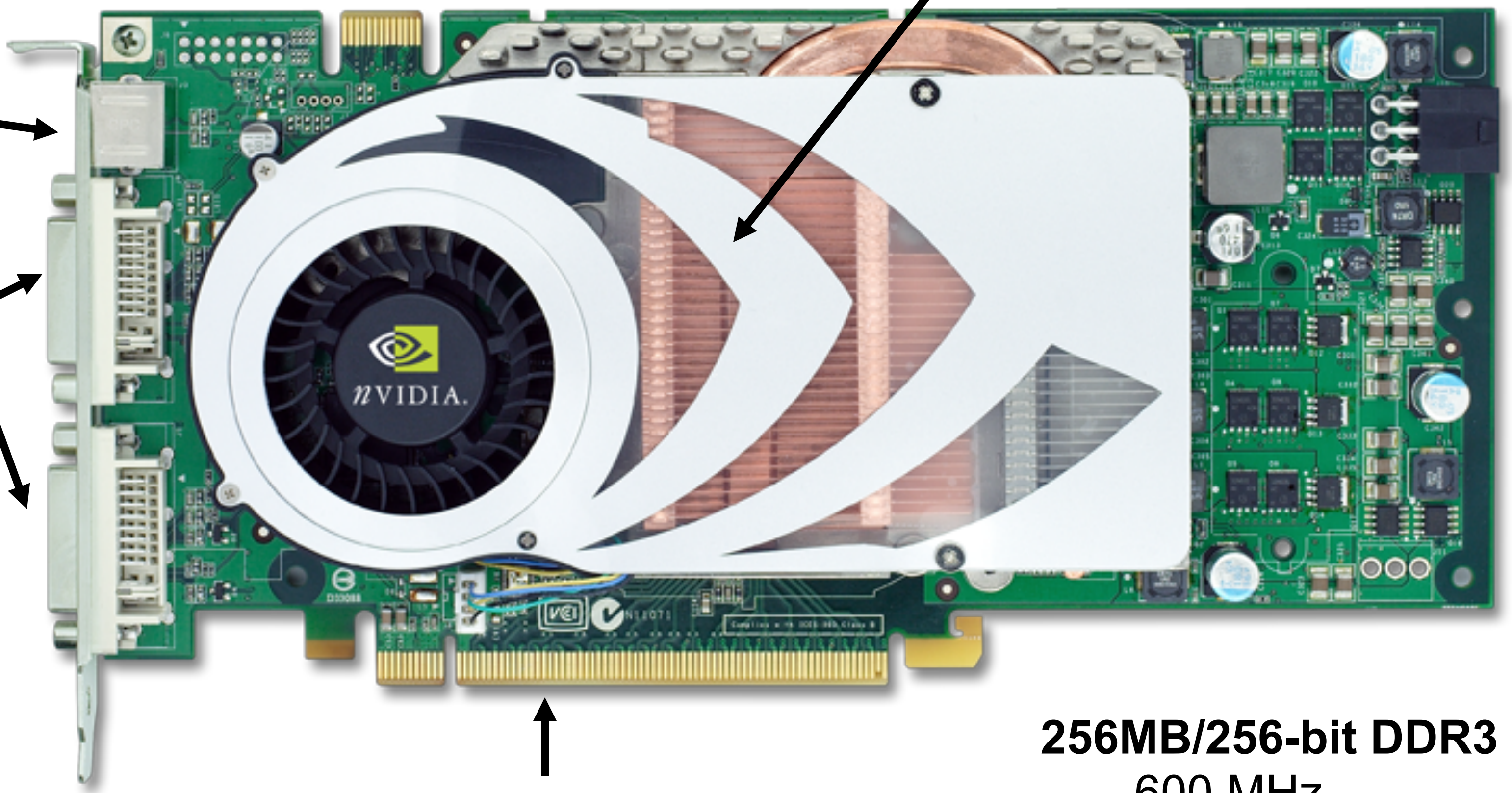
- Goal is to maintain DC balance while have sufficient state transition for clock recovery
- The difference of 1s and 0s in a 20-bit stream should be ≤ 2
- There should be no more than 5 consecutive 1s or 0s in any stream
- 00000000, 00000111, 11000001 bad
- 01010101, 11001100 good
- Find 256 good patterns among 1024 total patterns of 10 bits to encode an 8-bit data
- A 25% overhead

PCIe PC Architecture

- PCIe forms the interconnect backbone
  - Northbridge/Southbridge are both PCIe switches
  - Some Southbridge designs have built-in PCI-PCIe bridge to allow old PCI cards
  - Some PCIe I/O cards are PCI cards with a PCI-PCIe bridge
- Source: Jon Stokes, PCI Express: An Overview
  - http://arstechnica.com/articles/paedia/hardware/pcie.ars



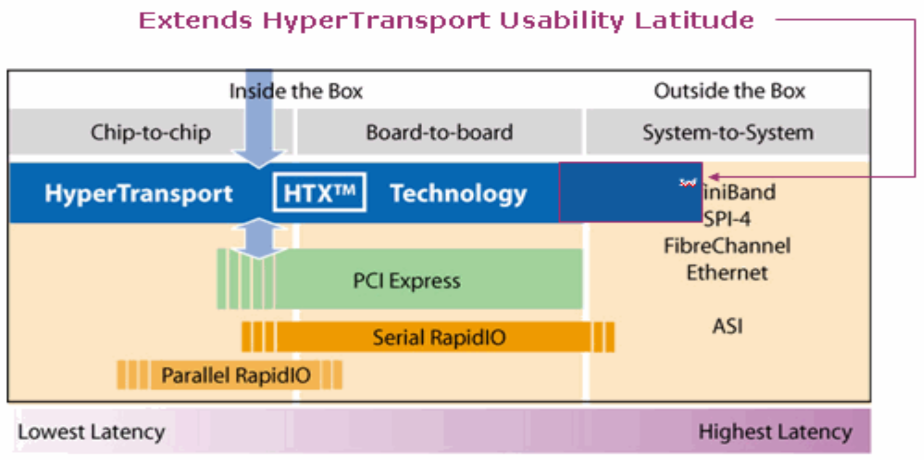
GeForce 7800 GTX Board Details



256MB/256-bit DDR3
600 MHz
8 pieces of 8Mx32

HyperTransport™ Feeds and Speeds

- Primarily a low latency direct chip-to-chip interconnect, supports mapping to board-to-board interconnect such as PCIe
- HyperTransport ™ 1.0 Specification
  - 800 MHz max, 12.8 GB/s aggregate bandwidth (6.4 GB/s each way)
- HyperTransport ™ 2.0 Specification
  - Added PCIe mapping
  - 1.0 - 1.4 GHz Clock, 22.4 GB/s aggregate bandwidth (11.2 GB/s each way)
- HyperTransport ™ 3.0 Specification
  - 1.8 - 2.6 GHz Clock, 41.6 GB/s aggregate bandwidth (20.8 GB/s each way)
  - Added AC coupling to extend HyperTransport ™ to long distance to system-to-system interconnect



Courtesy HyperTransport ™ Consortium

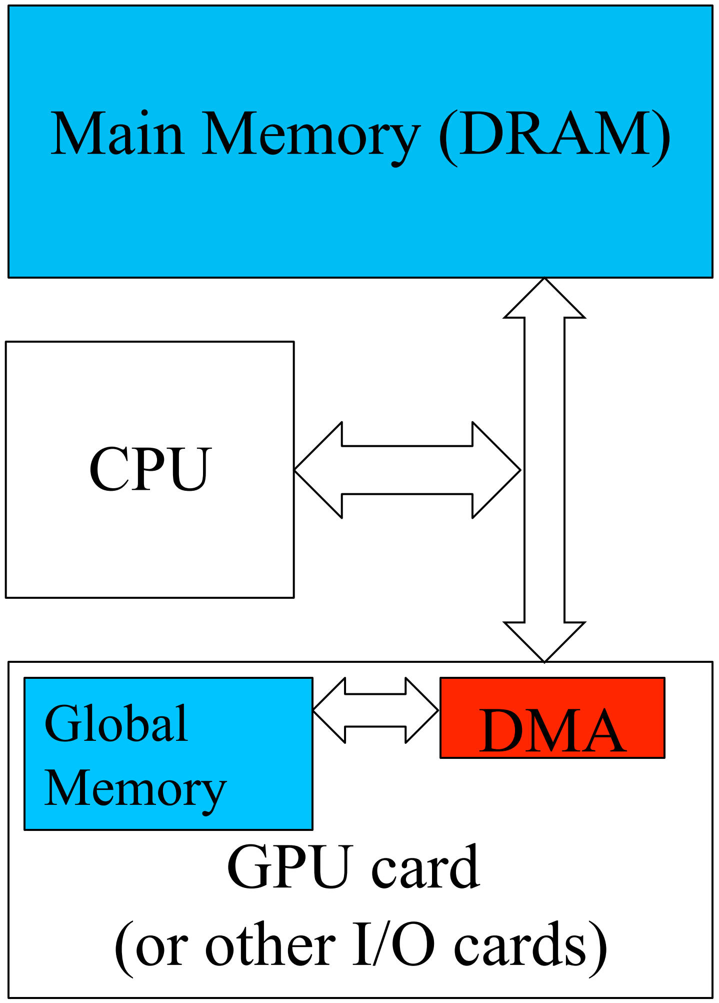
Source: "White Paper: AMD HyperTransport Technology-Based System Architecture

PCIe 3

- A total of 8 Giga Transfers per second in each direction
- No more 8/10 encoding but uses a polynomial transformation at the transmitter and its inverse at the receiver to achieve the same effect
- So the effective bandwidth is double of PCIe 2

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 (a.k.a. Page Locked 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

- Use the allocated memory and its pointer the same way those returned by malloc();
- The only difference is that the allocated memory cannot be paged by the OS
- The cudaMemcpy function should be about 2X faster with pinned memory
- Pinned memory is a limited resource whose over-subscription can have serious consequences

Important Trends

- Knowing yesterday, today, and tomorrow
  – The PC world is becoming flatter
  – CPU and GPU are being fused together
  – Outsourcing of computation is becoming easier…

ANY MORE QUESTIONS?