Parallel Computing with GPUs

GPU Architectures
Part 1 – Introduction to GPUs

The University Of Sheffield.

Dr Paul Richmond
http://paulrichmond.shef.ac.uk/teaching/COM4521/

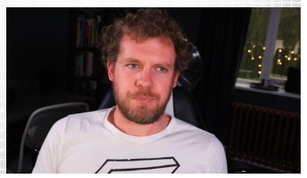
This Lecture (learning objectives)

- Introduction to GPUs
  - Compare latency with throughput and identify how this relates to CPU and GPU architectures
  - Identify examples from Flynn's taxonomy
  - Classify the taxonomy of a GPU

Latency vs. Throughput

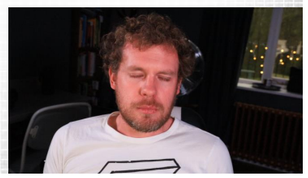
- Latency: The time required to perform some action
  - Measure in units of time
- Throughput: The number of actions executed per unit of time
  - Measured in units of what is produced

- E.g. *An assembly line manufactures GPUs. It takes **6 hours** to manufacture a GPU but the assembly line can manufacture **100 GPUs per day**.*

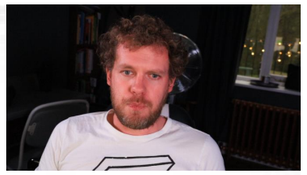


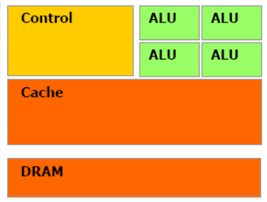
CPU vs GPU

- CPU
  - Latency oriented
  - Optimised for serial code performance
  - Good for single complex tasks
- GPU
  - Throughput oriented
  - Massively parallel architecture
  - Optimised for performing many similar tasks simultaneously (data parallel)

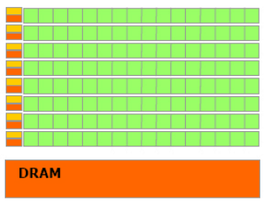


CPU vs GPU





CPU



GPU

- Large Cache
  - Hide long latency memory access
- Powerful Arithmetic Logical Unit (ALU)
  - Low Operation Latency
- Complex Control mechanisms
  - Branch prediction etc.

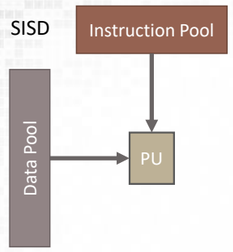
- Small cache
  - But faster memory throughput
- Energy efficient ALUs
  - Long latency but high throughput
- Simple control
  - No branch prediction

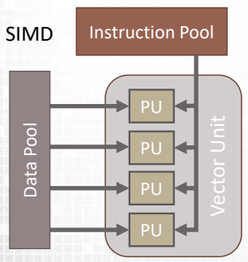
Data Parallelism

- Program has many similar threads of execution
  - Each thread performs the same behaviour on different data
  - Good for high throughput
- We can classify an architecture based on instructions and data (Flynn's Taxonomy)
  - Instructions:
    - Single instruction (SI)
    - Multiple Instruction (MI)
    - *Single Program (SP)*
    - *Multiple Program (MP)*

    } *Not part of the original taxonomy*
  - Data:
    - Single Data (SD) – w.r.t. *work item not necessarily single word*
    - Multiple Data (MD)
  - e.g. SIMD = Single Instruction and Multiple Data

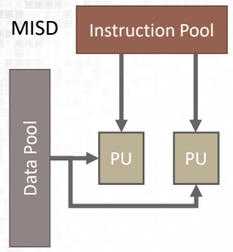
SISD and SIMD

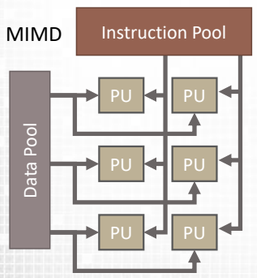




- SISD
  - Classic von Neumann architecture
  - PU = Processing Unit
- SIMD
  - Multiple processing elements performing the same operation simultaneously
  - E.g. Early vector super computers
  - Modern CPUs have SIMD instructions
    - But are not SIMD in general

MISD and MIMD





- MISD
  - E.g. Pipelined architectures
- MIMD
  - Processors as functionally asynchronous and independent
  - Different processors may execute different instructions on different data
  - E.g. Most parallel computers
  - E.g. OpenMP programming model

SPMD and MPMD

- SPMD
  - Multiple autonomous processors simultaneously executing a program on different data
  - Program execution can have an independent path for each data point
  - E.g. Message passing on distributed memory machines.
- MPMD
  - Multiple autonomous processors simultaneously executing at least two independent programs.
  - Typically client & host programming models fit this description.
  - E.g. Sony PlayStation 3 SPU/PPU combination, Some system on chip configurations with CPU and GPUs

Taxonomy of a GPU

- What taxonomy best describes data parallelism with a GPU?

- **Obvious Answer**: SIMD

- **Less Obvious answer**: SPMD

- **Slightly confusing answer**: SIMT (Single Instruction Multiple Thread)
  - This is a combination of both it differs from SIMD in that;
    1) Each thread has its own registers
    2) Each thread has multiple addresses
    3) Each thread has multiple flow paths
  - We will explore this in more detail when we look at the hardware!
  - http://yosefk.com/blog/simd-simt-smt-parallelism-in-nvidia-gpus.html

The University Of Sheffield.

Taxonomy of a GPU

- What taxonomy best describes data parallelism with a GPU?
  - SISD?
  - SIMD?
  - MISD?
  - MIMD?
  - SPMD?
  - MPMD?

Summary

- Introduction to GPUs
  - Compare latency with throughput and identify how this relates to CPU and GPU architectures
  - Identify examples from Flynn's taxonomy
  - Classify the taxonomy of a GPU

- Next Lecture: Programming GPUs

Parallel Computing with GPUs

GPU Architectures
Part 2 – Programming GPUs

The University Of Sheffield.

Dr Paul Richmond
http://paulrichmond.shef.ac.uk/teaching/COM4521/

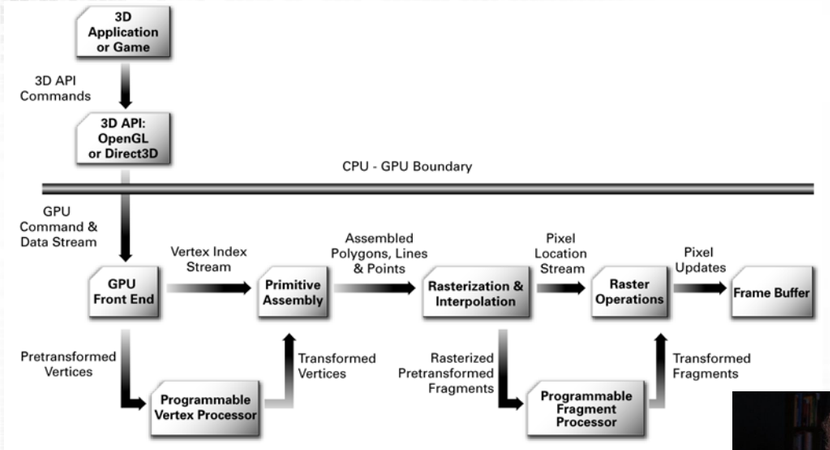
This Lecture (learning objectives)

- Programming GPUs
  - Summarise history around the development of GPU programming techniques
  - Compare a range of approaches for GPU programming

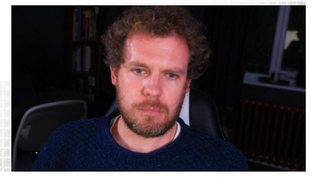
GPU Early History

- Hardware has evolved from the demand for increased quality of 3D computer graphics
- Initially specialised processors for each part of the graphics pipeline
  - Vertices (points of triangles) and Fragments (potential pixels) can be manipulated in parallel
- The stages of the graphics pipeline became programmable in early 2000's
  - NVIDIA GeForce 3 and ATI Radeon 9700
  - DirectX 9.0 required programmable pixel and vertex shaders

The Graphics Pipeline



Source: *NVidia Cg Users Manual*



GPGPU

- General Purpose computation on Graphics Hardware
  - First termed by Mark Harris (NVIDIA) in 2002
  - Recognised the use of GPUs for non graphics applications
- Requires mapping a problem into graphics concepts
  - Data into textures (images)
  - Computation into shaders
- Later unified processors were used rather than fixed stages
  - 2006: GeForce 8 series

Unified Processors and CUDA

- Compute Unified Device Architecture (CUDA)
  - First released in 2006/7
- Targeted new bread of unified “streaming multiprocessors”
- C like programming for GPUs
  - No computer graphics: General purpose programming model
  - Revolutionised GPU programming for general purpose use

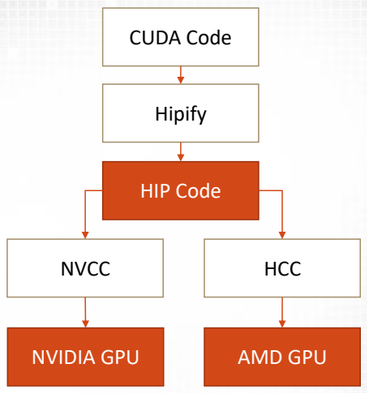
Directive based GPU programming

- GPU Accelerated Directives (OpenACC)
  - Helps compiler auto generate code for the GPU
  - Very similar to OpenMP
  - *Pros: Performance portability, limited understanding of hardware required*
  - *Cons: Limited fine grained control of optimisation*
- OpenMP 4.0
  - GPU offload for parallelism
  - *Pros: Platform and hardware independent, write once*
  - *Cons: Difficult to obtain high performance or use cutting edge features*

```
#pragma omp target data map (to: c[0:N], b[0:N]) map(tofrom: a[0:N])
#pragma omp target teams distribute parallel for
for (j=0; j<N; j++){
    a[j] = b[j]+scalar*c[j];
}
```

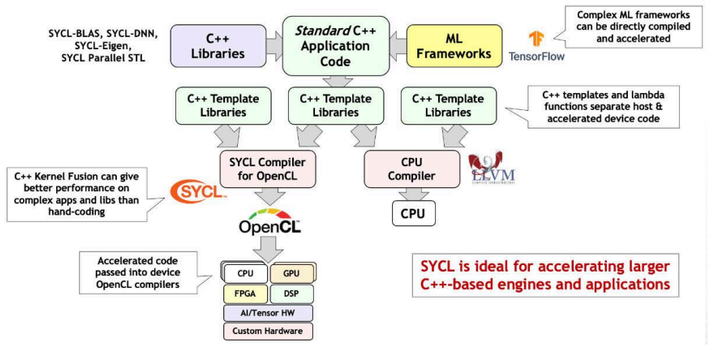
ROCm and HIP

- Radeon Open Compute (ROCm)
  - Platform and runtime for Gpu compute
  - AMD open equivalent of CUDA
- Heterogeneous-Compute Interface for Portability (HIP)
  - C++ interface
  - One to one replacement for CUDA
  - HIP source to source conversion tools
- *Pros: Can run on AMD and NVIDIA GPU hardware*
- *Cons: Subset of the CUDA language, not truly performance portable*



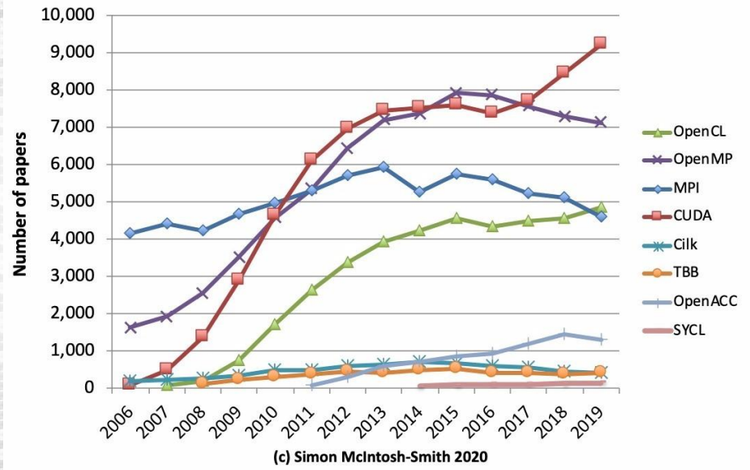
OpenCL and SYCL

- OpenCL: Multiple architecture support (CPUs/GPUs/FPGA)
  - Lower level than CUDA
  - Portability diminished if code is “targeted”
- SYCL: Based on modern C++ (C++17)
  - Performance portable
  - Implementations supported by different vendors (e.g. hipSYCL)



https://www.khronos.org/sycl/

HPC language use in research



https://www.nextplatform.com/2020/04/28/programming-in-the-parallel-universe/

Summary

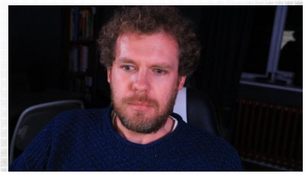
- Programming GPUs
  - Summarise history around the development of GPU programming techniques
  - Compare a range of approaches for GPU programming
- Next Lecture: NVIDIA Hardware Model

Parallel Computing with GPUs

GPU Architectures
Part 3 – GPU Hardware

The University Of Sheffield.

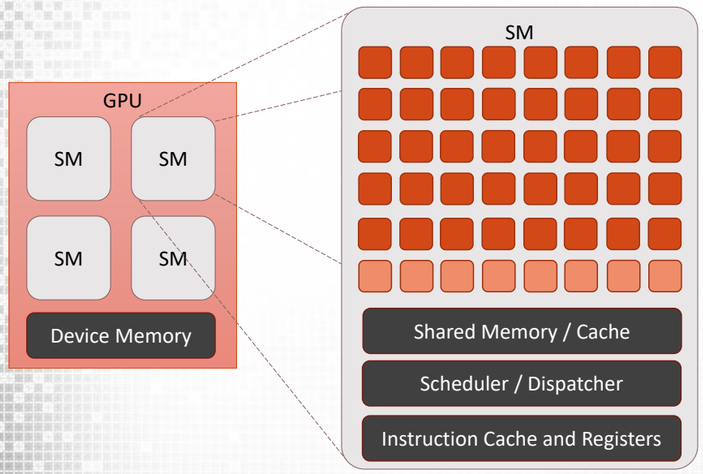
Dr Paul Richmond
http://paulrichmond.shef.ac.uk/teaching/COM4521/



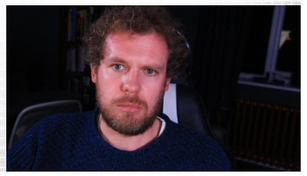
This Lecture (learning objectives)

- NVIDIA GPU Hardware
  - Explain the NVIDIA hardware model and key terminology
  - Compare the hardware variants and identify changing architectural characteristics
  - Give examples of GPU usage at different system scales

Hardware Model



- NVIDIA GPUs have a 2-level hierarchy
  - Each Streaming Multiprocessor (SMP) has multiple vector "CUDA" cores
  - The number of SMs varies across different hardware implementations
  - The design of SMPs varies between GPU families
  - The number of cores per SMP varies between GPU families

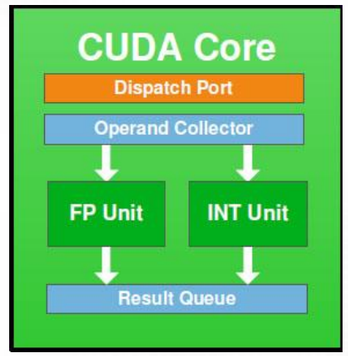


NVIDIA GPU Range

- GeForce
  - Consumer range
  - Gaming oriented for mass market
- Quadro Range
  - Workstation and professional graphics
- Tesla
  - Number crunching boxes
  - Much better support for double precision
  - Faster memory bandwidth
  - Better Interconects
- Mobile …

NVIDIA CUDA Core

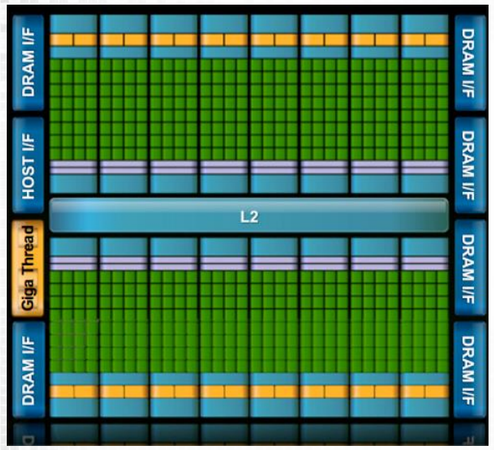
- CUDA Core
  - Vector processing unit
  - Either FP32, FP64
    - Volta onwards: INT32 and Tensor Cores
  - Works on a single operation by initiating instructions on clock cycles



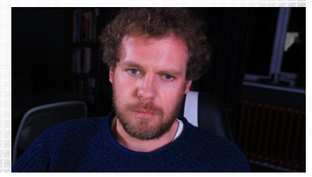
Tesla Range Specifications

	"Kepler" K20	"Kepler" K40	"Maxwell" M40	Pascal P100	Volta V100	Ampere A100
32bit CUDA cores	2496	2880	3072	3584	5120	6912
Chip Variant	GK110	GK110B	GM200	GP100	GV100	GA100
Cores per SM	192	192	128	64	64	64
Single Precision Performance	3.52 Tflops	4.29 Tflops	7.0 Tflops	9.5 Tflops	15.4 Tflops	**19.5 Tflops**
Double Precision Performance	1.17 TFlops	1.43 Tflops	*0.21 Tflops*	4.7 Tflops	7.8Tflops	**9.7 Tflops**
Memory Bandwidth	208 GB/s	288 GB/s	288GB/s	720GB/s	900GB/s	**1555 GB/s**
Memory	5 GB	12 GB	12GB	12/16GB	16/32GB	40 GB

Fermi Family of Tesla GPUs

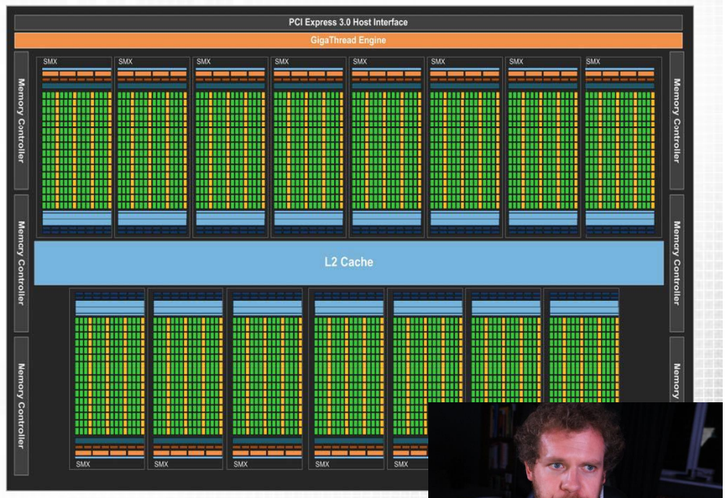


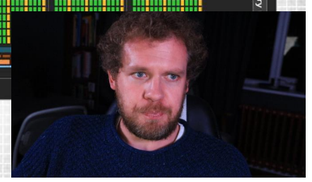
- Chip partitioned into Streaming Multiprocessors (SMPs)
- 32 vector cores per SMP
- Not cache coherent. No communication possible across SMPs.



Kepler Family of Tesla GPUs

- Streaming Multiprocessor Extreme (SMX)
- Huge increase in the number of cores per SMX
  - Smaller 28nm processes
- Increased L2 Cache
- Cache coherency at L2 not at L1





Maxwell Family Tesla GPUs

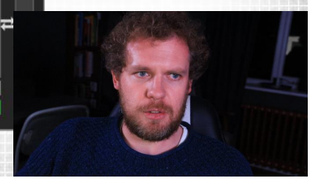
- Streaming Multiprocessor Module (SMM)
- SMM Divided into 4 quadrants (GPC)
  - Each has own instruction buffer, registers and scheduler for each of the 32 vector cores
- SMM has 90% performance of SMX at 2x energy efficiency
  - 128 cores vs. 192 in Kepler
  - BUT small die space = more SMMs
- 8x the L2 cache of Kepler (2MB)
- 20nm transistor size

Pascal P100 GPU

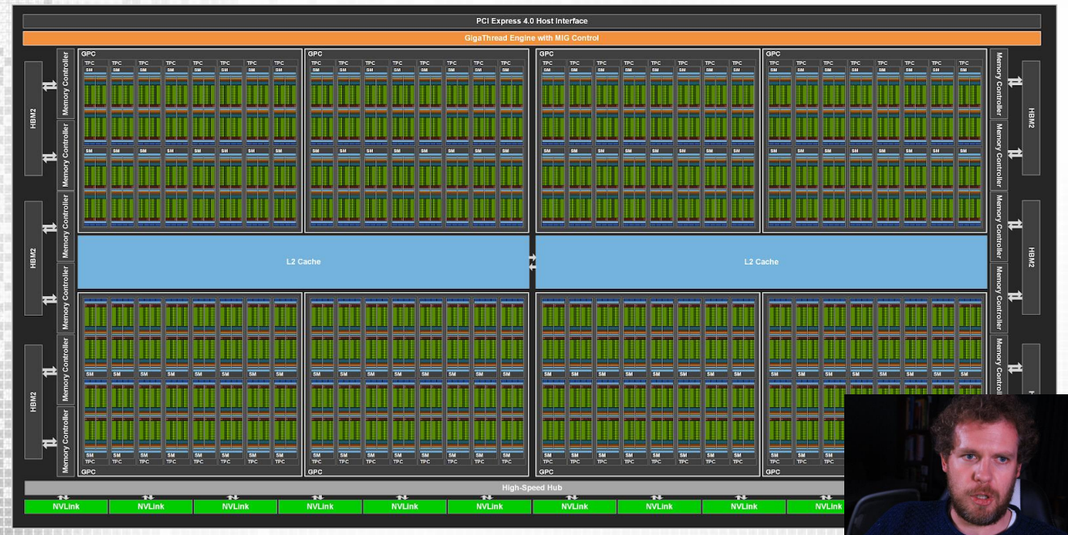
- Many more SMPs
- More GPCs
- Each CUDA core is more efficient
  - More registers available
- Same die size as Maxwell
  - Transistor shrink to 16nm
- Memory bandwidth improved drastically
  - NVLink

Volta V100 GPU

- Massive silicone size
  - Die shrink (12nm)
- CUDA cores and tensor cores

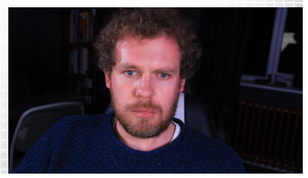


Ampere A100



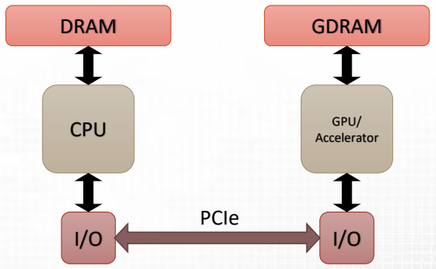
Warp Scheduling

- GPU Threads are always executed in groups called warps (32 threads)
  - **Warps are transparent to users**
- SMPs have zero overhead warp scheduling
  - Warps with instructions ready to execute are eligible for scheduling
  - Eligible warps are selected for execution on priority (context switching)
  - All threads (in a warp) execute the same instruction (SIMD) when executed on the vector processors (CUDA cores)
- The specific way in which warps are scheduled varies across families
  - Fermi, Kepler and Maxwell have different numbers of warp schedulers and dispatchers



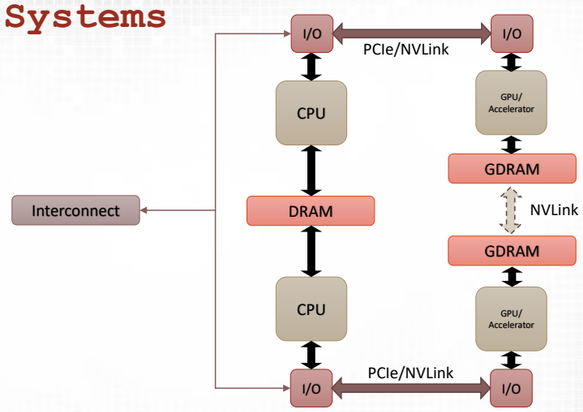
Accelerated Systems

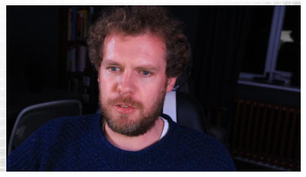
- CPUs and Accelerators are used together
  - GPUs cannot be used instead of CPUs
  - GPUs perform compute heavy parts
- Communication is via PCIe bus
  - PCIe 3.0: up to 8 GB per second throughput
  - NVLINK: 5-12x faster than PCIe 3.0



Larger Accelerated Systems

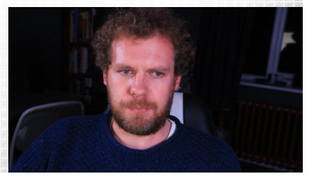
- Can have multiple CPUs and Accelerators within each “Shared Memory Node”
  - CPUs share memory but accelerators do not!





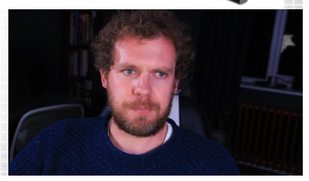
Simple Accelerated Workstation

- Insert your accelerator into PCI-e
- Make sure that
  - There is enough space
  - Your power supply unit (PSU)is up to the job
  - You install the latest GPU drivers

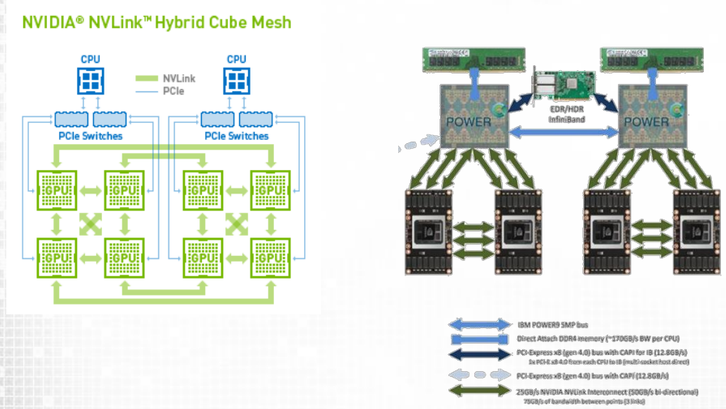


GPU Workstation Server

- Multiple Servers can be connected via interconnect
- Several vendors offer GPU servers
- For example 2 multi core CPUs + 4 GPUS
- Make sure your case and power supply are upto the job!



DGX, Power Systems and Supercomputers



Summary

- NVIDIA GPU Hardware
  - Explain the NVIDIA hardware model and key terminology
  - Compare the hardware variants and identify changing architectural characteristics
  - Give examples of GPU usage at different system scales