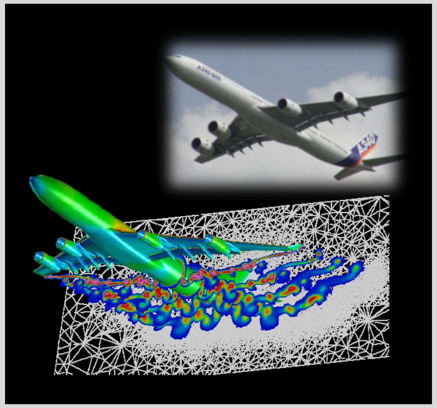


High Performance Computing

ADVANCED SCIENTIFIC COMPUTING

Prof. Dr. – Ing. Morris Riedel
Adjunct Associated Professor
School of Engineering and Natural Sciences, University of Iceland, Reykjavik, Iceland
Research Group Leader, Juelich Supercomputing Centre, Forschungszentrum Juelich, Germany

@Morris Riedel @MorrisRiedel @MorrisRiedel

LECTURE 7

Graphical Processing Units (GPUs)

October 28, 2019
Room V02-156

UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

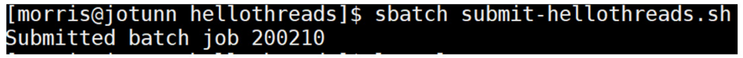
DEEP Projects

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

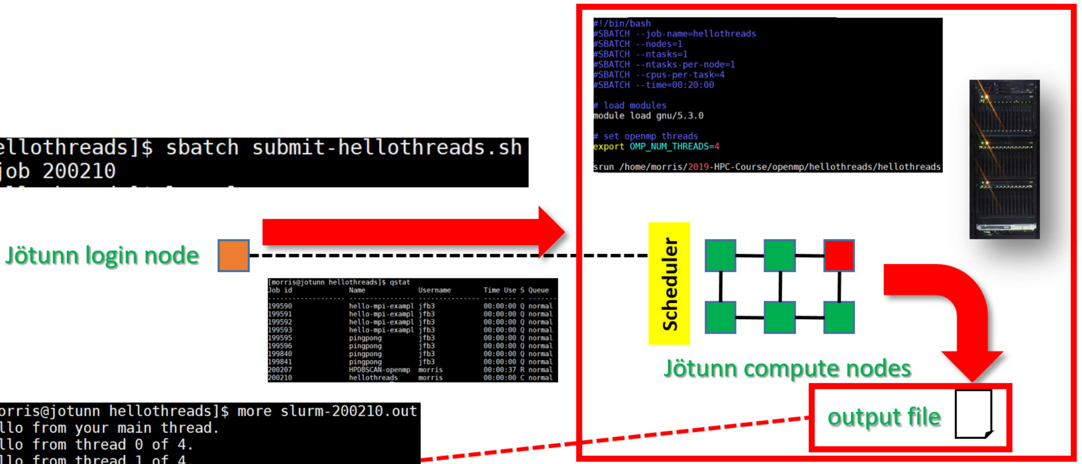
HAICU HELMHOLTZ ARTIFICIAL INTELLIGENCE COOPERATION UNIT

Review of Practical Lecture 6.1 – Understanding OpenMP Parallel Programming

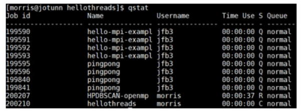
- Submission of Shared Memory Applications
  - Simple compiler directives enable parallelization in OpenMP
  - Remember application independent of number of threads
- Data Science & Machine Learning Applications
  - E.g. HPDBSCAN parallel code
  - Take advantage of simple OpenMP parallelization
  - Performance: OpenMP & MPI



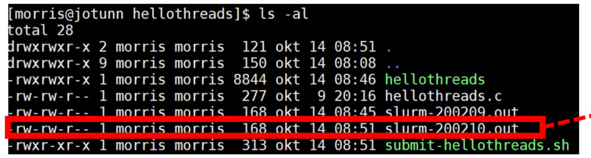
```
[morris@jotunn hellothreads]$ sbatch submit-hellothreads.sh
Submitted batch job 200210
```



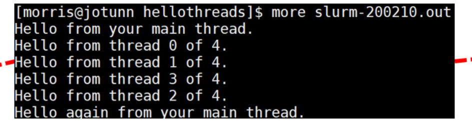
```
#!/bin/bash
#SBATCH --job-name=hellothreads
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4
#SBATCH --time=00:20:00

# load modules
module load gnu/5.3.0

# set openmp threads
export OMP_NUM_THREADS=4

srun /home/morris/2019-HPC-Course/openmp/hellothreads/hellothreads
```



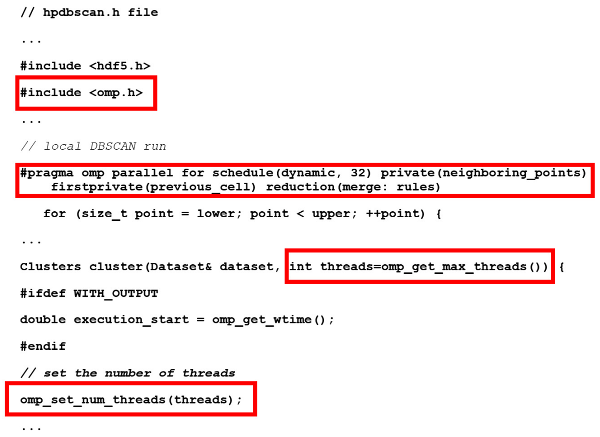
```
[morris@jotunn hellothreads]$ qstat
Job id               Name             Username        Time Use S Queue
-------------------- ---------------- --------------- -------- - -------
199590               hello-mpi-exampl jfb3            00:00:00 Q normal
199591               hello-mpi-exampl jfb3            00:00:00 Q normal
199592               hello-mpi-exampl jfb3            00:00:00 Q normal
199593               hello-mpi-exampl jfb3            00:00:00 Q normal
199595               pingpong         jfb3            00:00:00 Q normal
199596               pingpong         jfb3            00:00:00 Q normal
199840               pingpong         jfb3            00:00:00 Q normal
199841               pingpong         jfb3            00:00:00 Q normal
200207               HPDBSCAN-openmp  morris          00:00:37 R normal
200210               hellothreads     morris          00:00:00 C normal
```



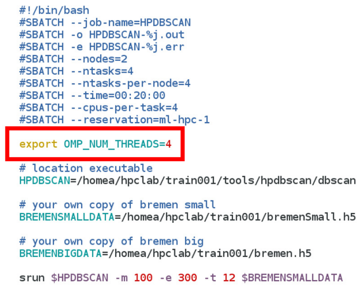
```
[morris@jotunn hellothreads]$ ls -al
total 28
drwxrwxr-x 2 morris morris  121 okt 14 08:51 .
drwxrwxr-x 9 morris morris  150 okt 14 08:08 ..
-rwxrwxr-x 1 morris morris 8844 okt 14 08:46 hellothreads
-rw-rw-r-- 1 morris morris  277 okt  9 20:16 hellothreads.c
-rw-rw-r-- 1 morris morris  168 okt 14 08:45 slurm-200209.out
-rw-rw-r-- 1 morris morris  168 okt 14 08:51 slurm-200210.out
-rwxr-xr-x 1 morris morris  313 okt 14 08:51 submit-hellothreads.sh
```



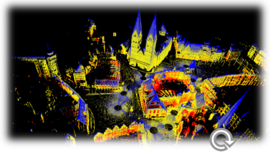
```
[morris@jotunn hellothreads]$ more slurm-200210.out
Hello from your main thread.
Hello from thread 0 of 4.
Hello from thread 1 of 4.
Hello from thread 3 of 4.
Hello from thread 2 of 4.
Hello again from your main thread.
```



```
// hpdbscan.h file
...
#include <hdf5.h>
#include <omp.h>
...
// local DBSCAN run
#pragma omp parallel for schedule(dynamic, 32) private(neighboring_points)
    firstprivate(previous_cell) reduction(merge: rules)
    for (size_t point = lower; point < upper; ++point) {
...
Clusters cluster(Dataset& dataset, int threads=omp_get_max_threads()) {
#ifdef WITH_OUTPUT
double execution_start = omp_get_wtime();
#endif
// set the number of threads
omp_set_num_threads(threads);
...
```



```
#!/bin/bash
#SBATCH --job-name=HPDBSCAN
#SBATCH -o HPDBSCAN-%j.out
#SBATCH -e HPDBSCAN-%j.err
#SBATCH --nodes=2
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=4
#SBATCH --time=00:20:00
#SBATCH --cpus-per-task=4
#SBATCH --reservation=ml-hpc-1

export OMP_NUM_THREADS=4

# location executable
HPDBSCAN=/homea/hpclab/train001/tools/hpdbscan/dbscan

# your own copy of bremen small
BREMENSMALLDATA=/homea/hpclab/train001/bremenSmall.h5

# your own copy of bremen big
BREMENBIGDATA=/homea/hpclab/train001/bremen.h5

srun $HPDBSCAN -m 100 -e 300 -t 12 $BREMENSMALLDATA
```



[1] M. Goetz & M. Riedel et al, Proceedings IEEE Supercomputing Conference, 2015

Outline of the Course

+ additional practical lectures & Webinars for our hands-on assignments in context

- 🟩 Practical Topics
- 🟦 Theoretical / Conceptual Topics

Outline

- General Purpose Graphical Processing Units (GPGPUs)
  - Multi-core CPUs vs. Many-core GPUs Revisited & TOP500 Impact
  - Terminology & NVIDIA Architecture Examples (Kepler, Pascal, Volta)
  - Understanding Node Architectures with GPUs & Summit HPC System
  - Interconnecting GPUs with NVLink/NVSwitch in 'islands' per Nodes
  - GPUDirect in Modular Supercomputing & Garpur Iceland HPC System

- **Promises from previous lecture(s):**
- ***Lecture 6 & Practical Lecture 6.1:*** **Lecture 7 will offer more details on OpenMP relationships of programming GPUs and similiarites to GPU programming using OpenACC**

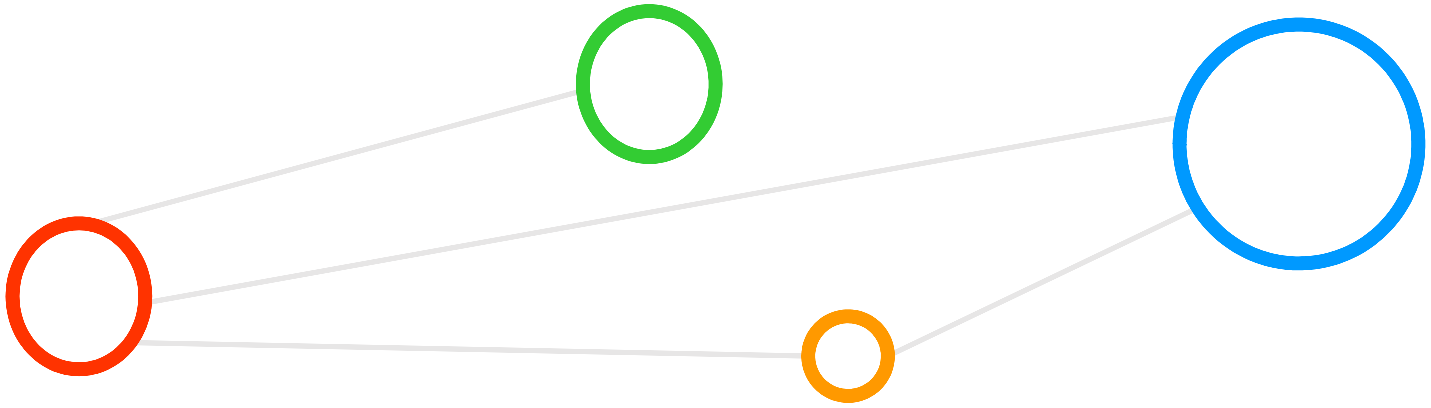
- **Note that this lecture is not a full programming course on GPUs with CUDA, OpenACC, and HIP that would require at least full 2-3 days**
- **The goal is to understand the enormous options to use GPUs today**

- GPU Libraries & Programming Models
  - Data Science Impacts with Deep Learning & Multi GPU Horovod Interconnect
  - Simulation Sciences Impacts with Libraries (e.g., cuBLAS for basic linear algebra)
  - NVIDIA GPUs & Compute Unified Device Architecture (CUDA) Programming Model
  - Emerging different Vendors & AMD Radeon Examples
  - OpenACC & HIP Standard Programming Models

Selected Learning Outcomes

- Students understand…
  - Latest developments in parallel processing & high performance computing (HPC)
  - How to create and use high-performance clusters
  - What are scalable networks & data-intensive workloads
  - The importance of domain decomposition
  - Complex aspects of parallel programming
  - HPC environment tools that support programming or analyze behaviour
  - Different abstractions of parallel computing on various levels
  - Foundations and approaches of scientific domain-specific applications
- Students are able to …
  - Programm and use HPC programming paradigms
  - Take advantage of innovative scientific computing simulations & technology
  - Work with technologies and tools to handle parallelism complexity

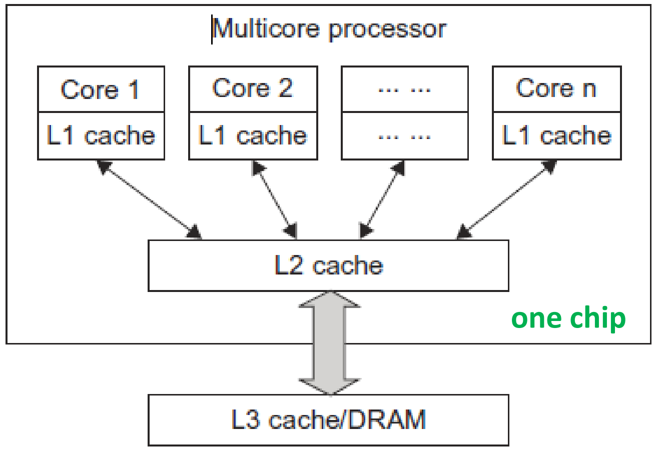
General Purpose Graphical Processing Units (GPGPUs)



Multi-core CPU Processors – Revisited (cf. Lecture 1)

- Significant advances in CPU (or microprocessor chips)
  - Multi-core architecture with dual, quad, six, or n processing cores
  - Processing cores are all on one chip

- Multi-core CPU chip architecture
  - Hierarchy of caches (on/off chip)
  - L1 cache is private to each core; on-chip
  - L2 cache is shared; on-chip
  - L3 cache or Dynamic random access memory (DRAM); off-chip

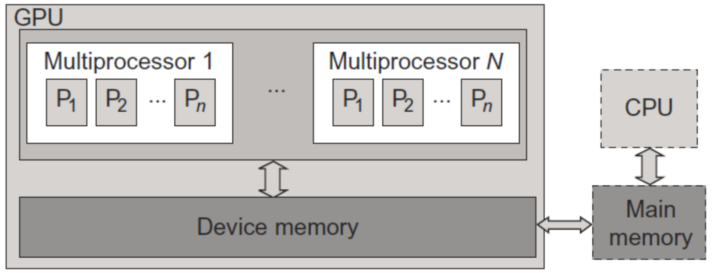


[3] Distributed & Cloud Computing Book

- **Clock-rate for single processors increased from 10 MHz (Intel 286) to 4 GHz (Pentium 4) in 30 years**
- **Clock rate increase with higher 5 GHz unfortunately reached a limit due to power limitations / heat**
- **Multi-core CPU chips have quad, six, or n processing cores on one chip and use cache hierarchies**

Many-core GPGPUs – Revisited (cf. Lecture 1)

- Use of very many simple cores
  - High throughput computing-oriented architecture
  - Use massive parallelism by executing a lot of concurrent threads slowly
  - Handle an ever increasing amount of multiple instruction threads
  - CPUs instead typically execute a single long thread as fast as possible
- Many-core GPUs are used in large clusters and within massively parallel supercomputers today
  - Named General-Purpose Computing on GPUs (GPGPU)
  - Different programming models emerge

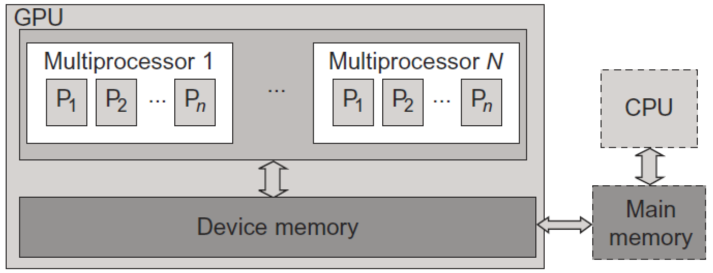


[3] Distributed & Cloud Computing Book

- **Graphics Processing Unit (GPU) is great for data parallelism and task parallelism**
- **Compared to multi-core CPUs, GPUs consist of a many-core architecture with hundreds to even thousands of very simple cores executing threads rather slowly**

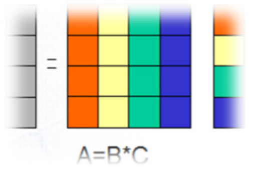
GPU Acceleration – Revisited (cf. Lecture 1)

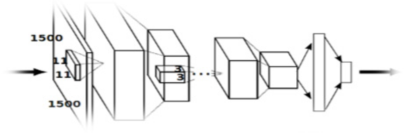
- GPU accelerator architecture example (e.g. NVIDIA card)
  - GPUs can have 128 cores on one single GPU chip
  - Each core can work with eight threads of instructions
  - GPU is able to concurrently execute 128 * 8 = 1024 threads
  - Interaction and thus major (bandwidth) bottleneck between CPU and GPU is via memory interactions
  - E.g. applications that use matrix – vector/matrix multiplication (e.g. deep learning algorithms)



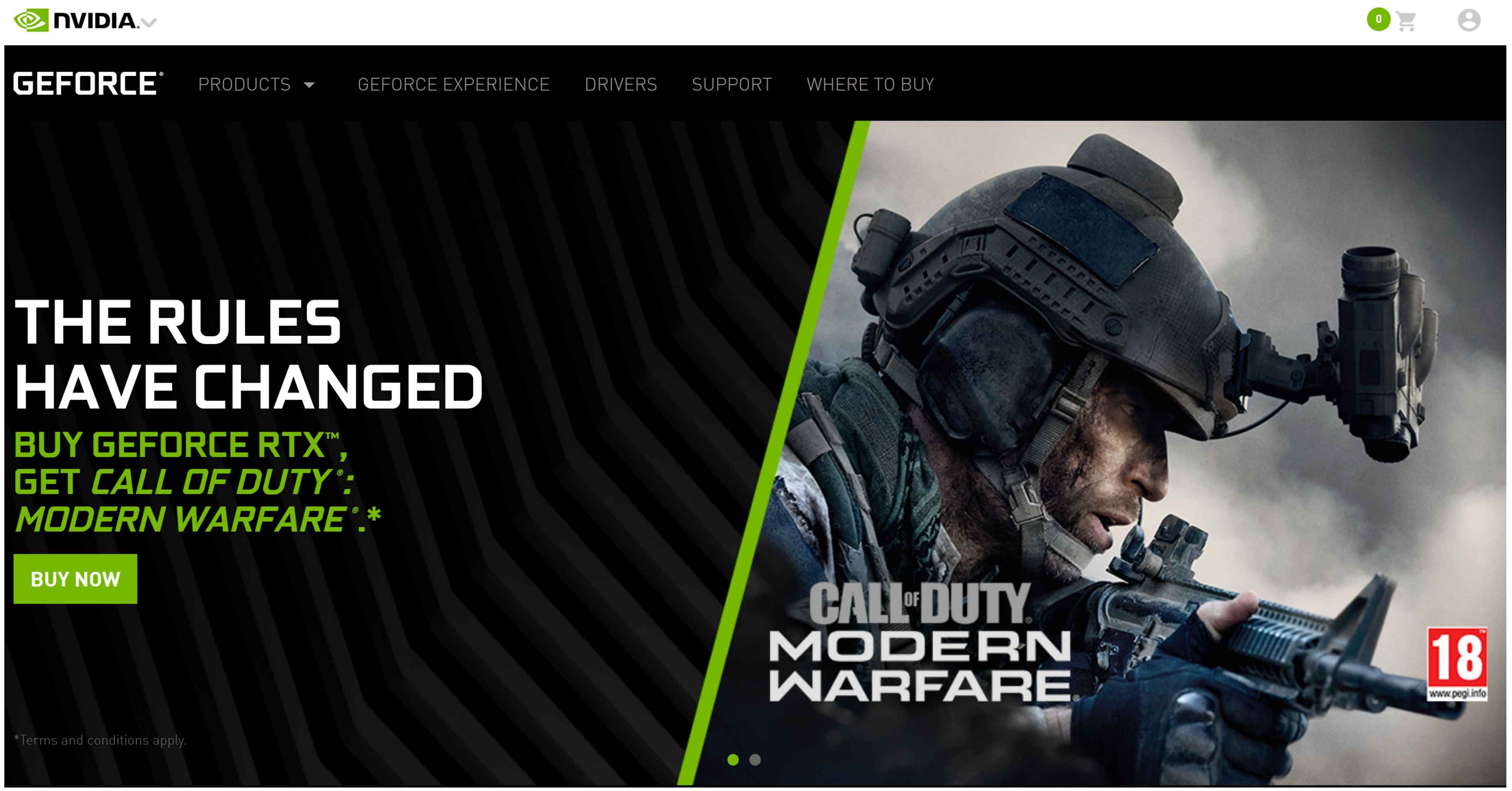
[3] Distributed & Cloud Computing Book

- CPU acceleration means that GPUs accelerate computing due to a massive parallelism with thousands of threads compared to only a few threads used by conventional CPUs
- GPUs are designed to compute large numbers of floating point operations in parallel





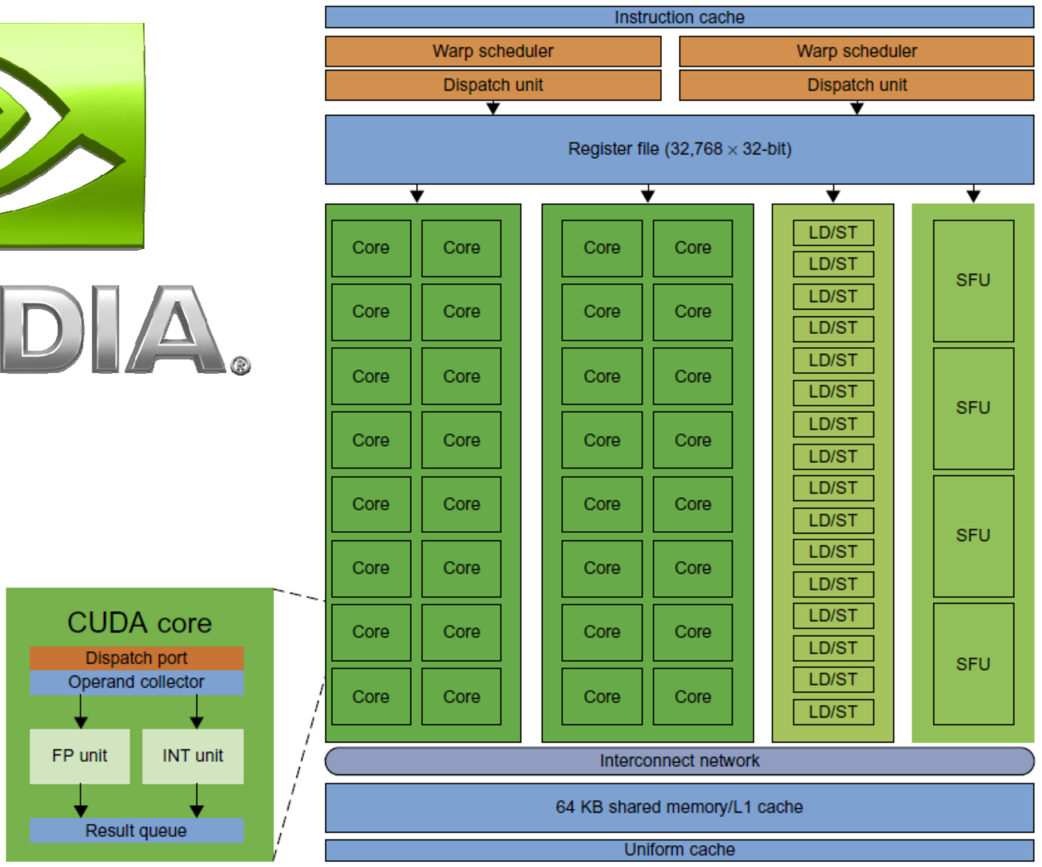
NVIDIA GEFORCE – Gaming Industry History Example powered by GPUs



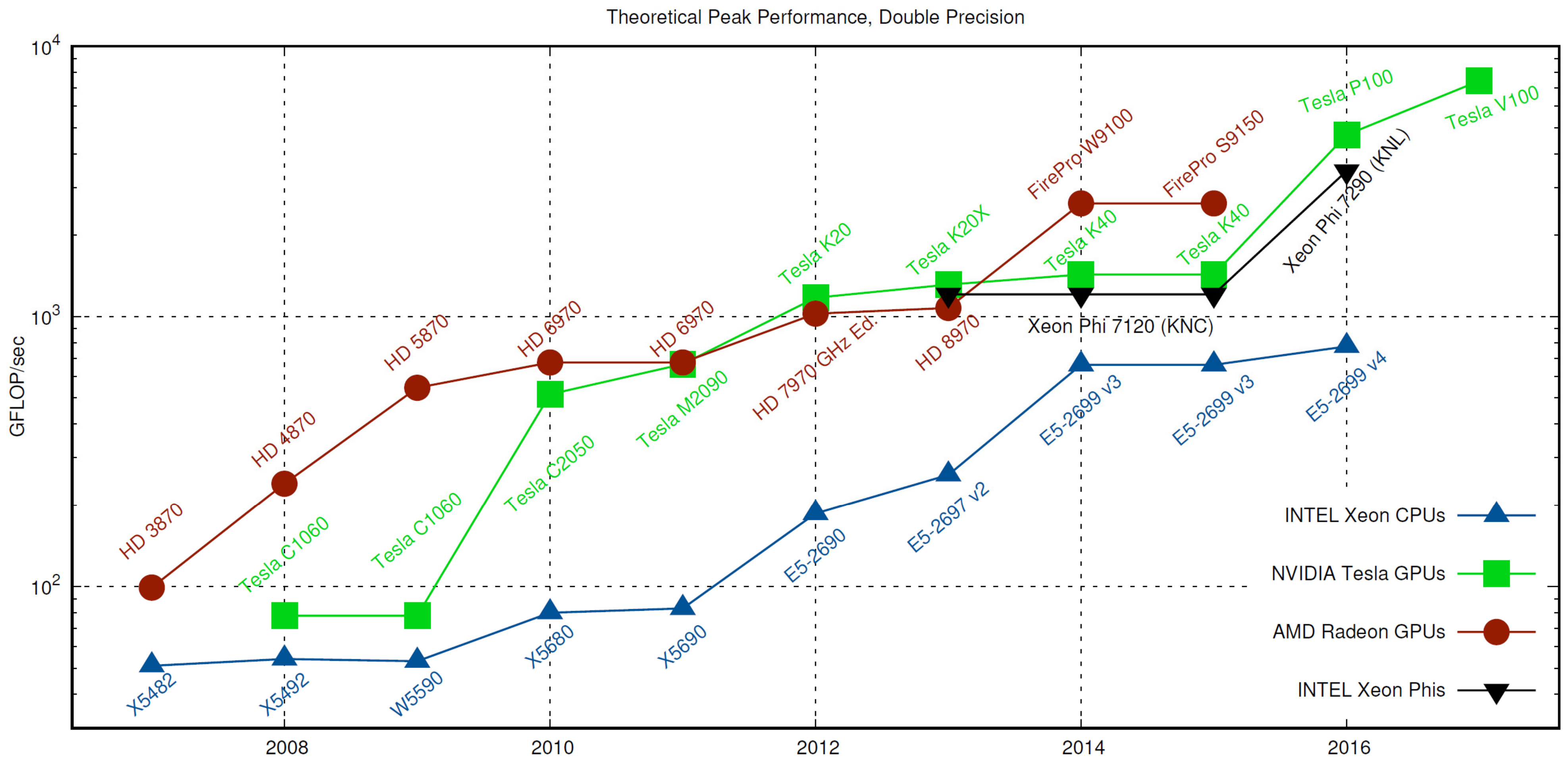
[11] NVIDIA GEFORCE

NVIDIA Fermi GPU Example – GPU Era before Kepler/Maxwell/Pascal/Volta

[3] Distributed & Cloud Computing Book



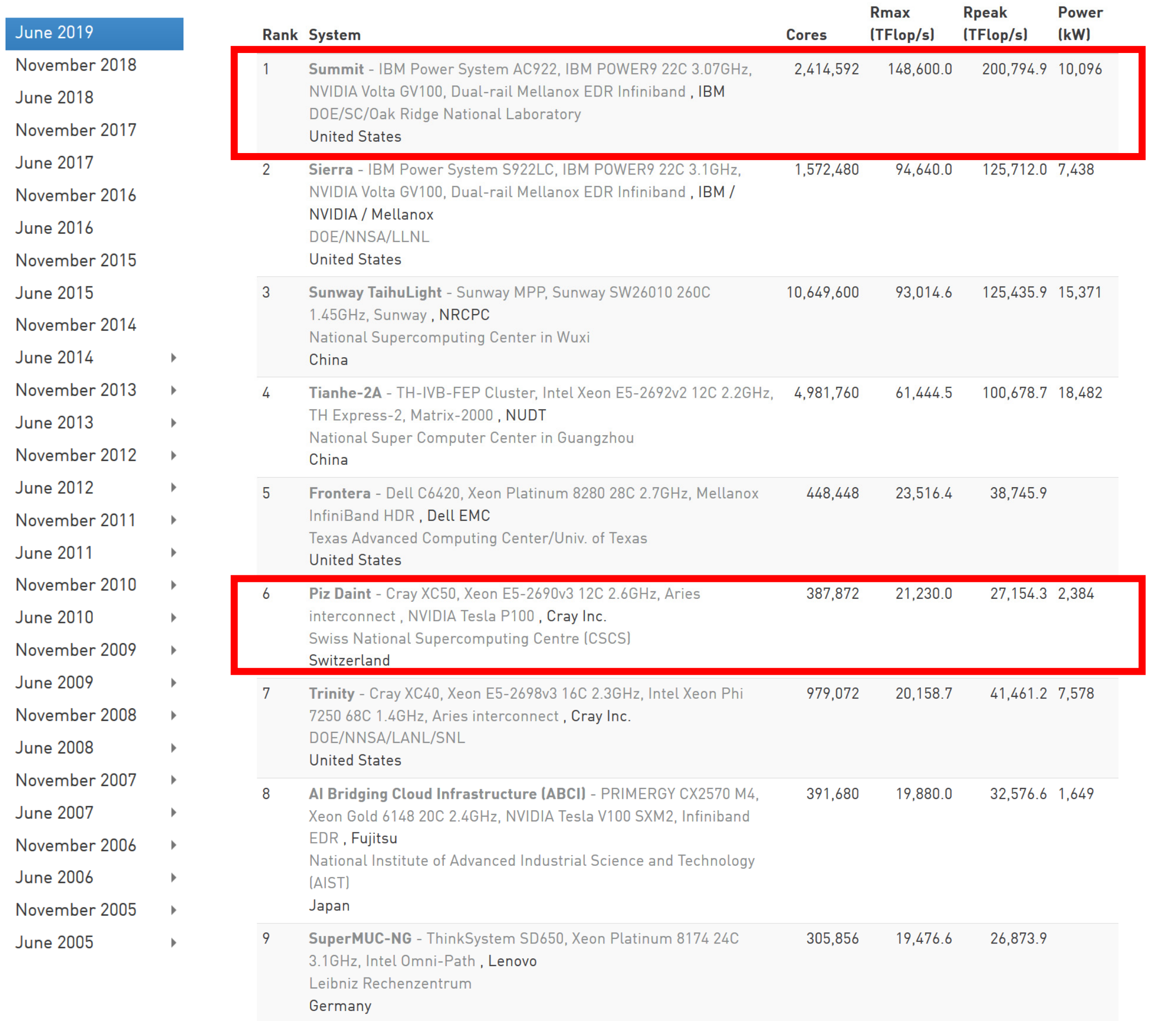
CPU/GPU Comparison & Evolution



[22] CPU/GPU Comparison

TOP 500 List (June 2019) – Revisited (cf. Lecture 1) – Impact on GPUs Today

[2] TOP500 Supercomputing Sites



Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
2	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
5	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR , Dell EMC Texas Advanced Computing Center/Univ. of Texas United States	448,448	23,516.4	38,745.9	
6	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	387,872	21,230.0	27,154.3	2,384
7	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/NNSA/LANL/SNL United States	979,072	20,158.7	41,461.2	7,578
8	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR , Fujitsu National Institute of Advanced Industrial Science and Technology (AIST) Japan	391,680	19,880.0	32,576.6	1,649
9	**SuperMUC-NG** - ThinkSystem SD650, Xeon Platinum 8174 24C 3.1GHz, Intel Omni-Path , Lenovo Leibniz Rechenzentrum Germany	305,856	19,476.6	26,873.9	

massive number of GPUs included

EU #1

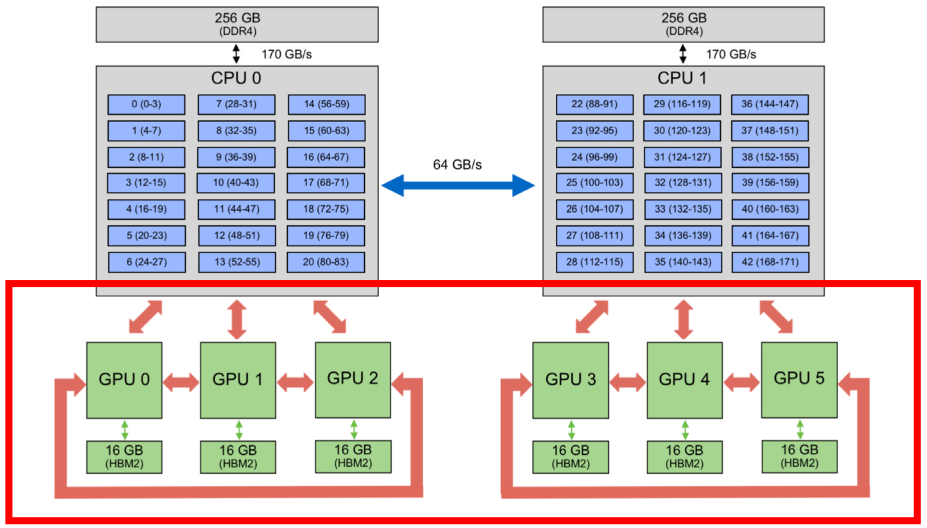
TOP 500 List (June 2019) – HPC #1 Machine Summit with Many-Core GPUs

- Selected Facts
  - Hosted at Department of Energy's (DOE) Oak Ridge National Laboratory (ORNL)
  - Space: As big as two tennis courts
  - Total number of nodes: 4608 Nodes
  - Each node: two 22-core Power9 IBM CPUs from Intel (Intel: 95.6% of all Top500 systems)
  - Each node: six(!) NVIDIA Tesla V100 GPUs (i.e., 4608 nodes x 6 GPUs = 27648 GPUs)
  - Nodes linked together with Mellanox dual-rail EDR Infiniband network
  - Power consumption: 13 Megawatts
  - Total system memory: >10 PetaByte (PB) 512 GigaBytes (GB) DDR4 + 96 GB HBM2 + ~ 1.6 TeraByte (TB) non-volatile memory (NVM)
  - File system: 250 PB IBM transferring data at 2.5 TB/s

- **Fastest HPC system in Top500 today is the Summit HPC machine hosted by US Oak Ridge National Laboratory (ORNL)**
- **The Summit HPC machine consists of 4608 nodes with each 2 22-core Power9 IBM CPUs from Intel per node & 6 GPUs NVIDIA Tesla V100 per node (27648 GPUs in total)**

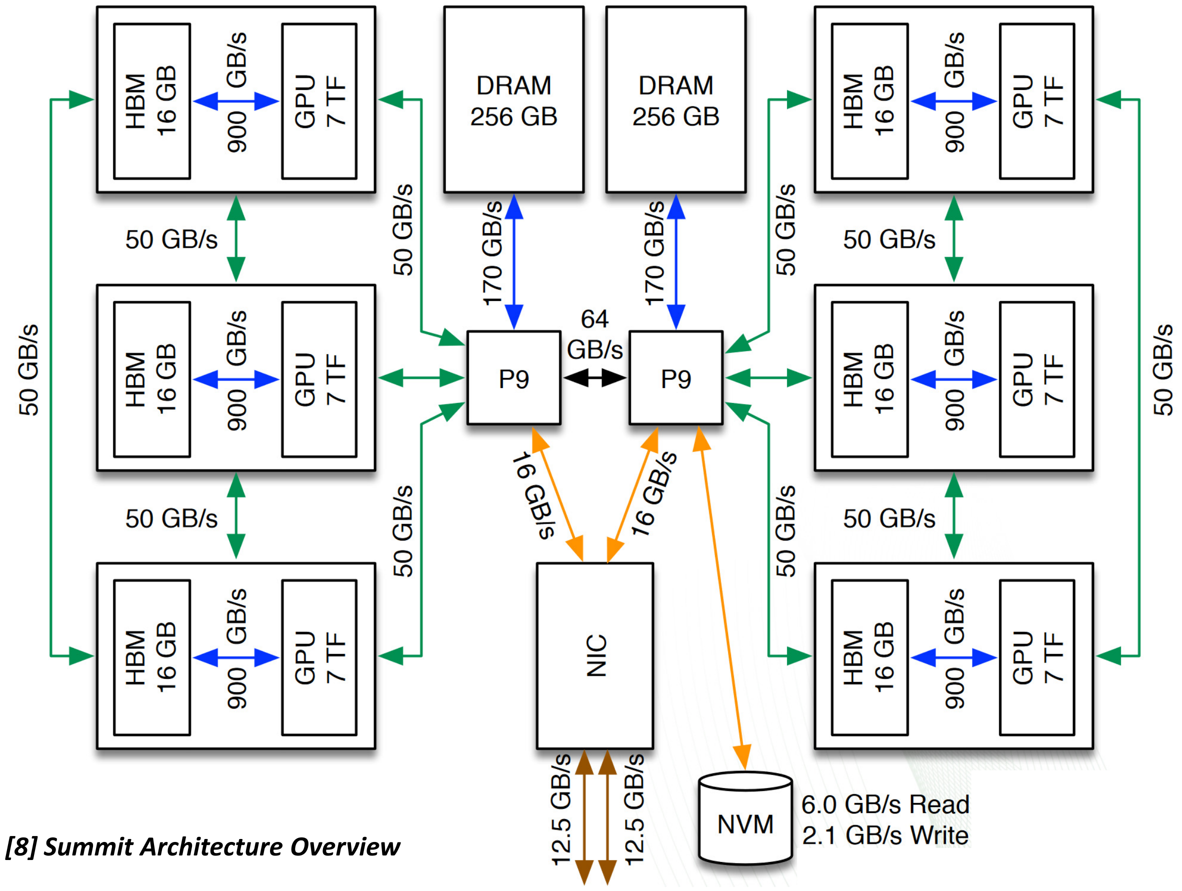
[7] OakRidge Supercomputer Summit

HPC #1 Machine Summit with Many-Core GPUs – Node Architecture with GPUs



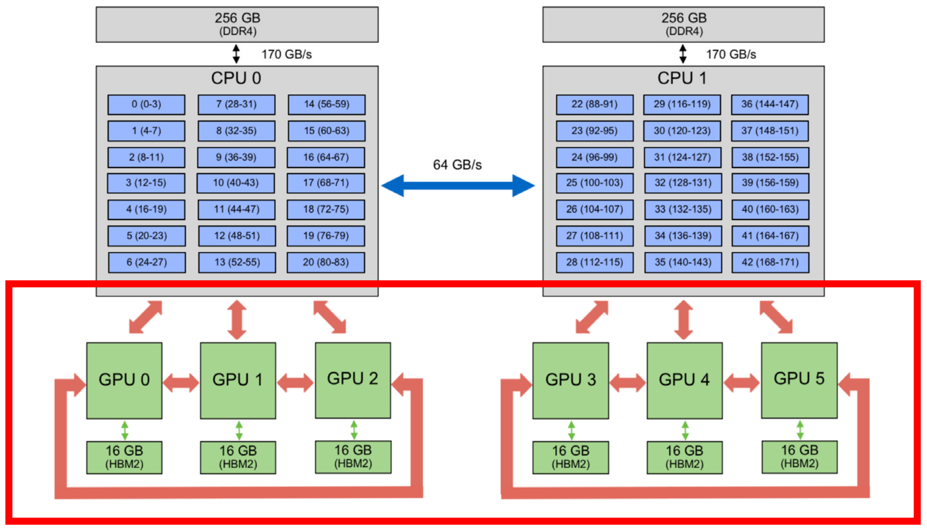
NVLink (v2) GPU Interconnect ~50 GB/s

- Facts
  - Coherent memory across entire node
  - NVLink (v2) fully interconnects 3 GPUs & 1 CPU with 50 GB/s on each side of node
  - PCIe Gen 4 connects NVM & Network Interconnect (NIC)



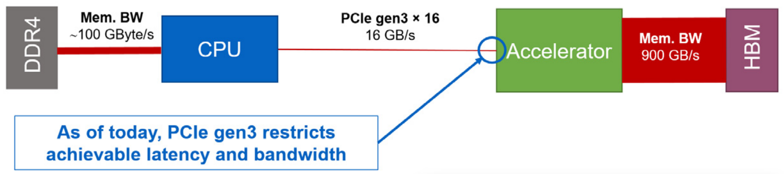
[8] Summit Architecture Overview

NVIDIA Tesla Volta (v100) GPU Example with High Bandwidth Memory 2 (HBM2)



HBM2 with 900 GB/s bandwidth

(note: also other GPUs use HBM2 like AMD Radeon Instinct)



[6] DEEP Projects Web Page *[12] JEDEC on HBM*

JEDEC® Global Standards for the Microelectronics Industry

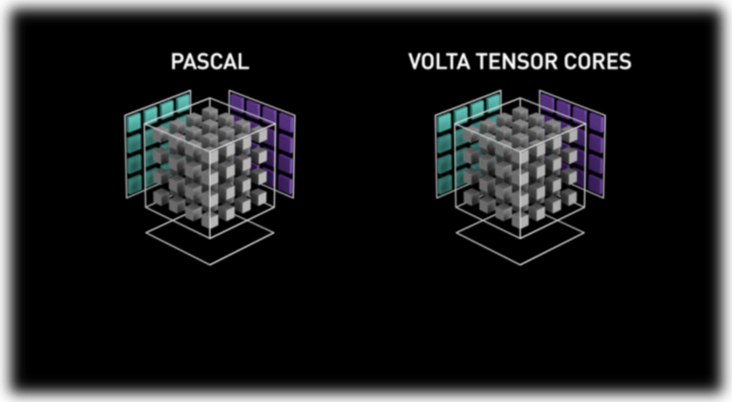
- **JEDEC is the global leader in the development of global standards for the microelectronics industry that approved HBM as an industry standard**
- **VIDIA Tesla Volta (v100) enables a high-performance random access memory (RAM) interface between the accelerator and so-called high bandwidth memory (version 2, HBM2) with up to 900 GB/s per GPU**
- **HBM achieves higher bandwidth while using less power in a substantially smaller form factor than DDR4 SDRAM (Double Data Rate 4 Synchronous Dynamic Random-Access Memory) by stacking up to 8 DRAMs (i.e., 3D circuit)**
- **Approach: requiring the memory and processor to be physically close, decreasing memory paths**

- Selected Facts
  - High-performance Random Access Memory (RAM) interface used in conjunction with HPC GPUs
  - HBM2 are typically on-package solutions which cannot be (re-)configured after manufacture
  - Manufactured by SK Hynix Inc (South Korea) & SAMSUNG

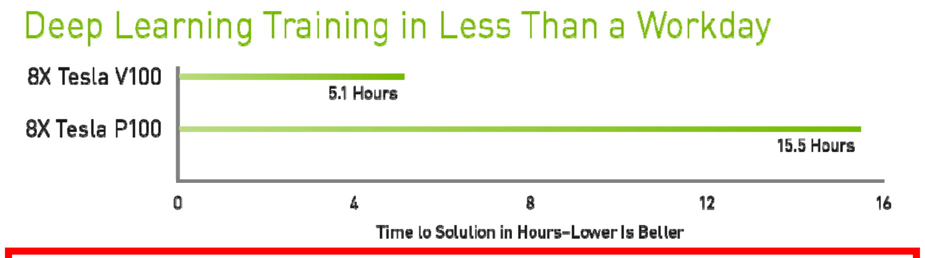
NVIDIA Tesla Volta (v100) GPU Example with Tensor Cores

- Selected Facts
  - 150 + 150 GB/s total bandwidth (NVLink v2)

- **NVIDIA Tesla Volta (v100) is equipped with over 21 billion transistors with 5120 CUDA cores & 640 tensor cores**
- **A tensor core is optimized for deep learning workloads by accelerating large matrix operations & perform mixed-precision matrix multiply & accumulate calculations in a single operation**
- **Predecessor of NVIDIA v100 was NVIDIA Tesla Pascal (p100) and widely used in HPC systems**
- **Predecessor of NVIDIA p100 was NVIDIA Tesla Kepler (e.g., K80 or K40) & used in HPC systems**

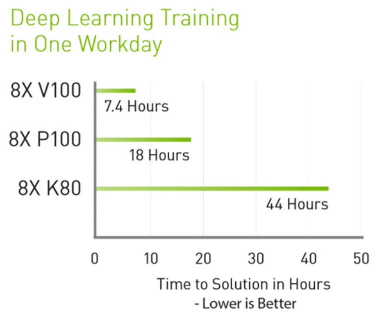


Mixed precision (e.g. FP16 vs. FP32 better for deep learning due to a regularization effect for example)

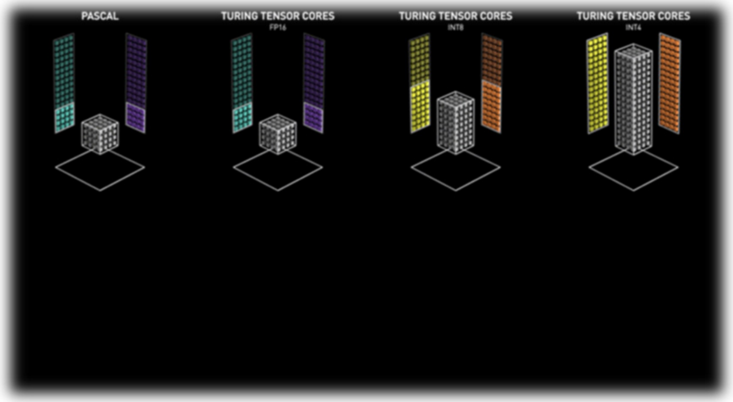


Server Config: Dual Xeon E5-2699 v4 2.6 GHz | 8X NVIDIA® Tesla® P100 or V100 | ResNet-50 Training on MXNet for 90 Epochs with 1.28M ImageNet Dataset.

(cf. Invited Lecture by Dr. G. Cavallaro on Deep Learning & Remote Sensing)



Server Config: Dual Xeon E5-2699 v4, 2.6GHz | 8x Tesla K80, Tesla P100 or Tesla V100 | V100 performance measured on pre-production hardware. | ResNet-50 Training on Microsoft Cognitive Toolkit for 90 Epochs with 1.28M ImageNet dataset



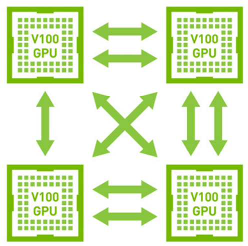
[8] Summit Architecture Overview *[9] NVIDIA Tesla Volta v100*

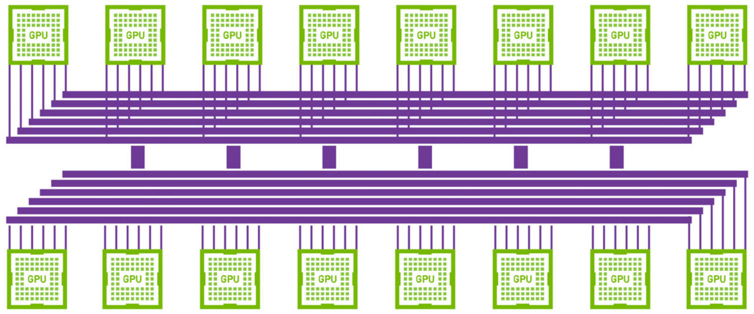
Understanding Interconnects of GPUs: NVLink/NVSwitch ‘Islands‘

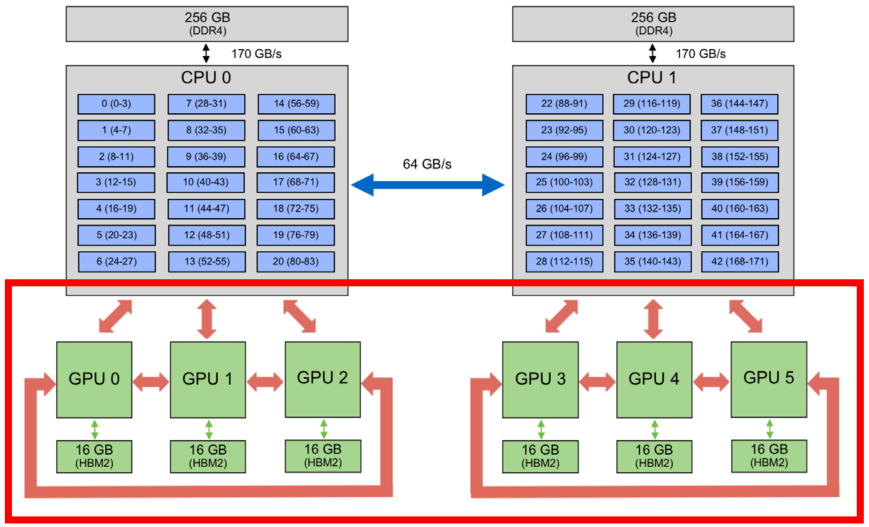
- Multi-GPU Communication (usually required for HPC)
  - E.g., NVLink enables GPU-to-GPU interconnects
  - E.g., NVSwitch enables all-to-all GPU communication / node (16 GPUs per server with 8 GPU pairs via 300 GB/s)
  - ‘Islands‘: Scaling up to the full HPC system with GPUs

[8] Summit Architecture Overview

[10] NVIDIA NVLink/NVSwitch



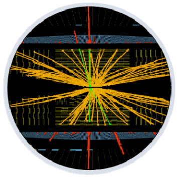




Summit Node Example : NVLink (v2) GPU Interconnect ~50 GB/s

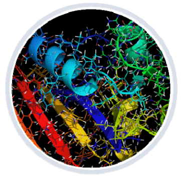
- **NVLink is a high-speed direct GPU-to-GPU interconnect that supports 6 NVLink connections per NVIDIA Tesla Volta v100 GPU (bandwidth 300 GB/s) and can interconnet up to 8 NVIDIA Tesla Volta v100 GPUs**
- **NVSwitch incorporates multiple NVLinks to provide all-to-all GPU communication within a single node**
- **NVLink/NVSwitch are considered as ‘Islands‘ since they do not scale with workloads to a full HPC machine like the GPUDirect interface is enabling**

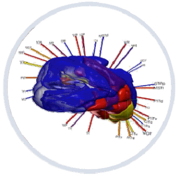
Application Co-Design of HPC Architectures – Modular Supercomputing Example


High Energy Physics

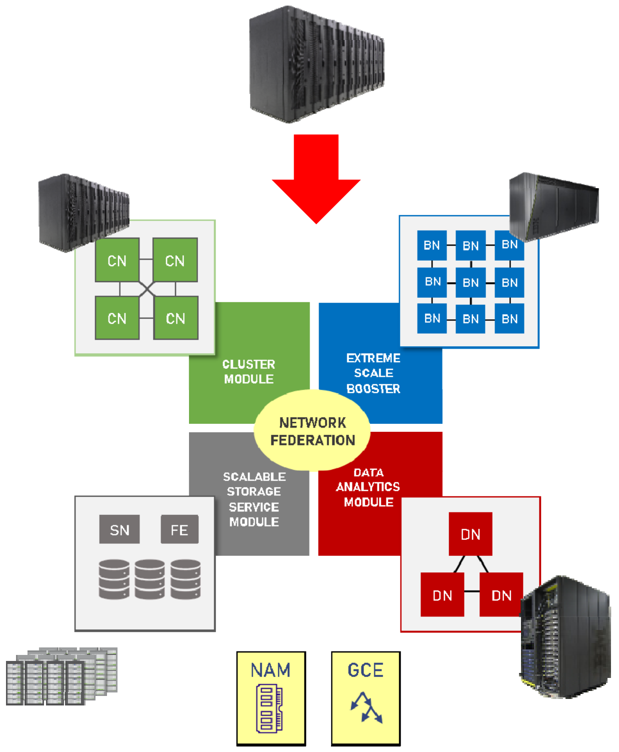
Earth Science

Space Weather

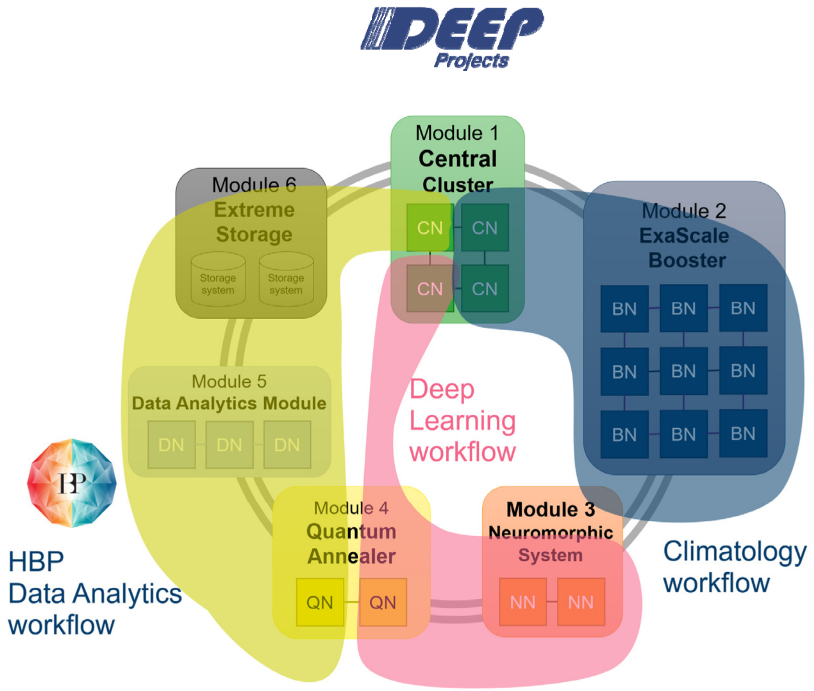

Molecular Dynamics


Neuroscience

Radio Astronomy

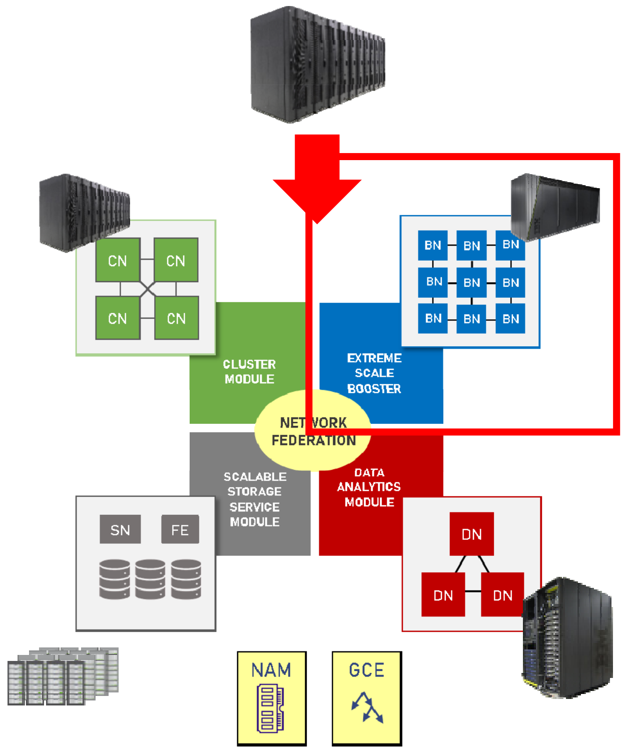


- The modular supercomputing architecture (MSA) enables a flexible HPC system design co-designed by the need of different application workloads



[6] DEEP Projects Web Page

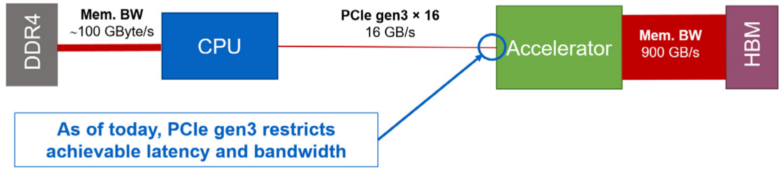
Modular Supercomputing Architecture – Scaling with GPUDirect Implementation

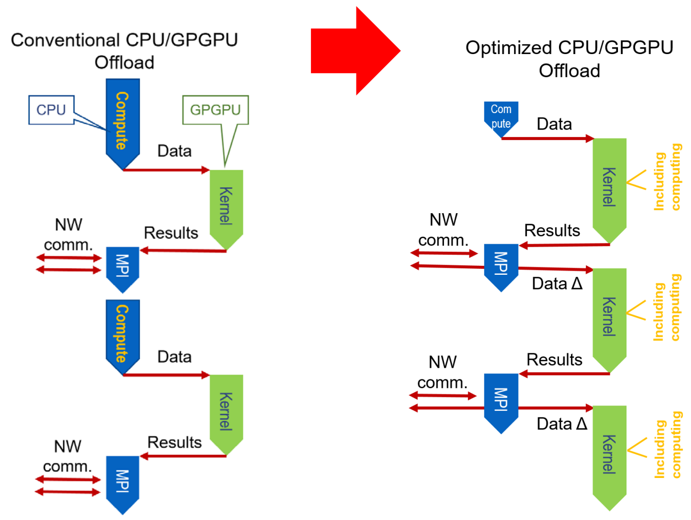


[6] DEEP Projects Web Page

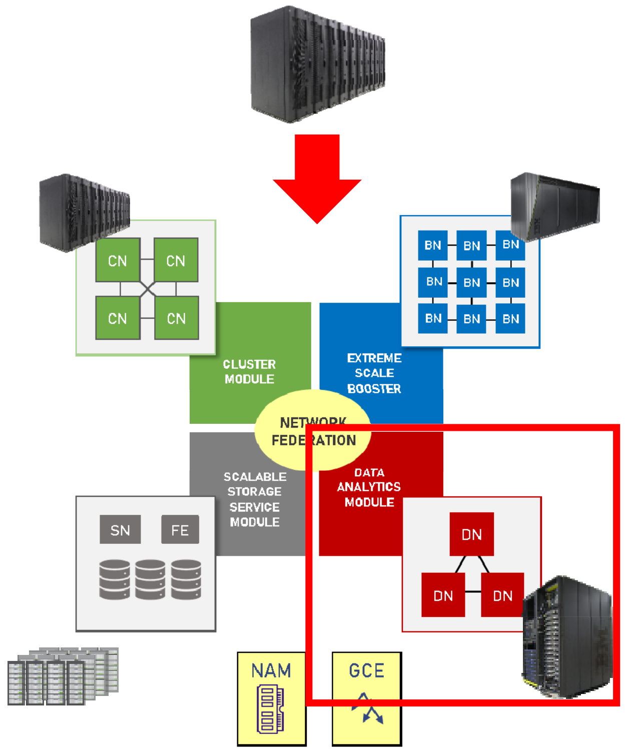
- Extreme Scale Booster (ESB)
  - Enables scalability based on many-core GPUs
  - Reduces load on host CPUs by enabling direct transfer of data through MPI to the GPUs (i.e., host CPU becomes more of a slim network driver in this context)
  - Ongoing research in DEEP-EST EU project

- **Innovative GPU interconnects are realized via GPUDirect implementations that go beyond the current limits of NVLink/NVSwitch 'islands'**





Modular Supercomputing Architecture – Data Analytics Module (DAM)



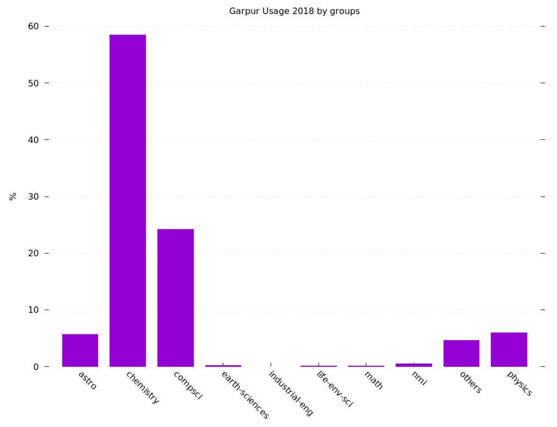
- Data Analytics Module (DAM)
  - Specific requirements for data science & analytics frameworks
  - 16 nodes with 2x Intel Xeon Cascade Lake; 24 cores
  - 1x NVIDIA V100 GPU / node
  - 1x Intel STRATIX10 FPGA PCIe3 / node
  - 384 GB DDR4 memory / node
  - 2 TB non-volatile memore / node
- DAM Prototype for teaching
  - 3 x 4 GPUs Tesla Volta V100
  - Slurm scheduling system

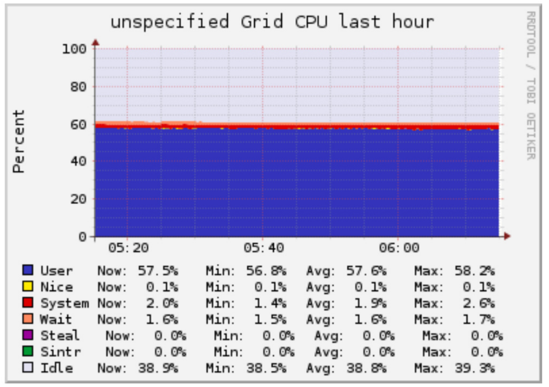
[6] DEEP Projects Web Page

➢ **The DAM prototype machine as part of the modular supercomputing architecture will be used in Assignment #2 for deep learning**

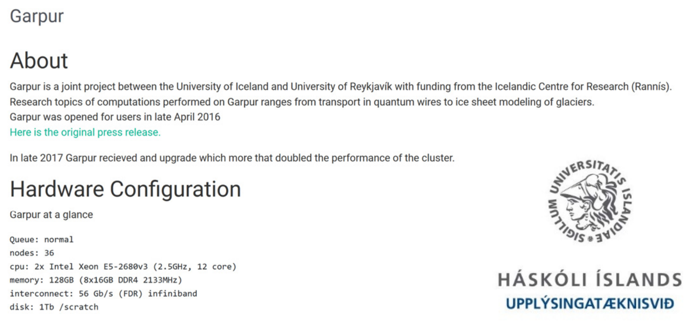
HPC System – Garpur Cluster – Offering GPUs for Research in Iceland

- Slurm Scheduling System
  - Usage via different queues
  - Several queues, here just selected examples
- Queue: Normal (36 nodes)
  - 2x Intel Xeon CPU (12 cores)
  - 128 GB Memory
- Queue: gpu (3 nodes)
  - 2x Tesla M2090 / node
- Queue: vgpu (2 nodes)
  - 1x Tesla Volta v100 / node
- Monitoring
  - Ganglia shows resource load





```
■ User   Now: 57.5%   Min: 56.8%   Avg: 57.6%   Max: 58.2%
■ Nice   Now:  0.1%   Min:  0.1%   Avg:  0.1%   Max:  0.1%
■ System Now:  2.0%   Min:  1.4%   Avg:  1.9%   Max:  2.6%
■ Wait   Now:  1.6%   Min:  1.5%   Avg:  1.6%   Max:  1.7%
■ Steal   Now:  0.0%   Min:  0.0%   Avg:  0.0%   Max:  0.0%
■ Sintr   Now:  0.0%   Min:  0.0%   Avg:  0.0%   Max:  0.0%
□ Idle   Now: 38.9%   Min: 38.5%   Avg: 38.8%   Max: 39.3%
```



Garpur

About

Garpur is a joint project between the University of Iceland and University of Reykjavík with funding from the Icelandic Centre for Research (Rannís).
Research topics of computations performed on Garpur ranges from transport in quantum wires to ice sheet modeling of glaciers.
Garpur was opened for users in late April 2016
Here is the original press release.

In late 2017 Garpur recieved and upgrade which more that doubled the performance of the cluster.

Hardware Configuration

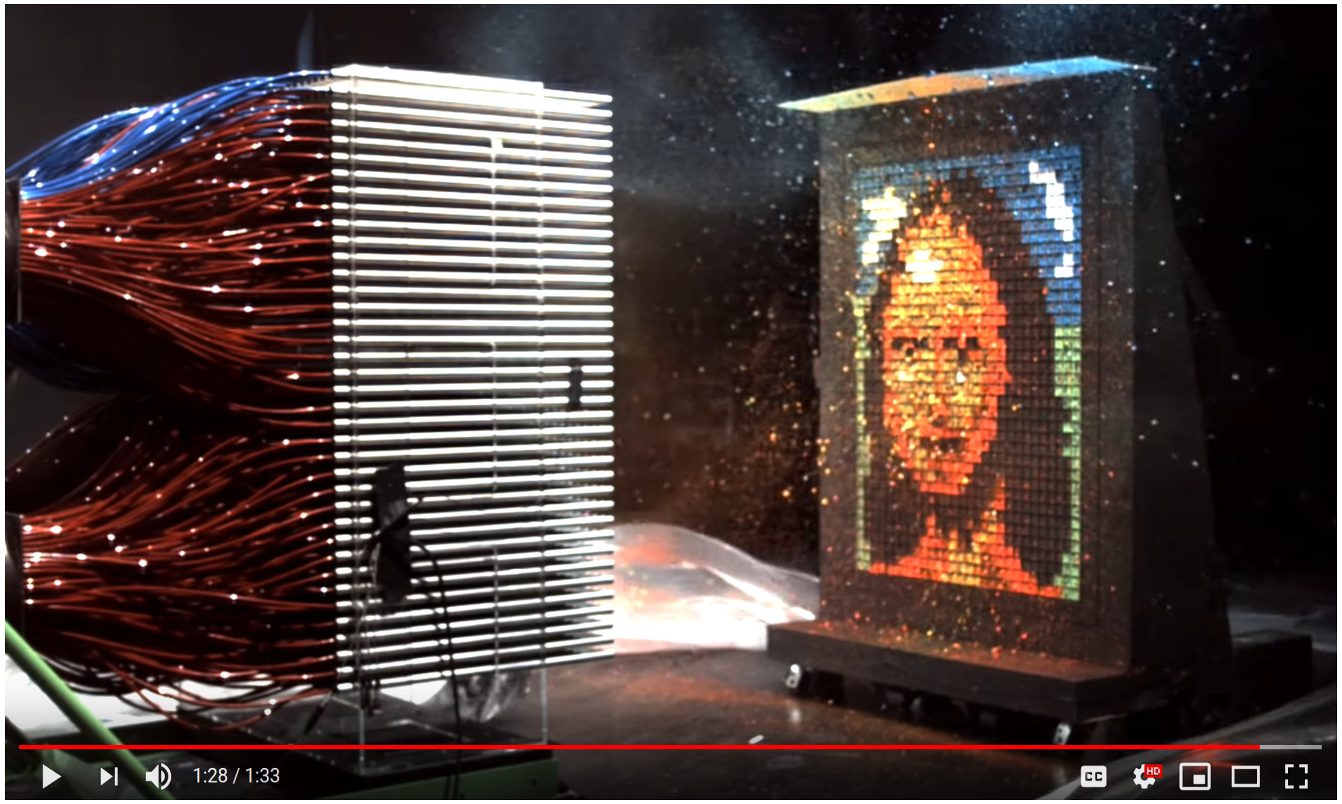
Garpur at a glance

```
Queue: normal
nodes: 36
cpu: 2x Intel Xeon E5-2680v3 (2.5GHz, 12 core)
memory: 128GB (8x16GB DDR4 2133MHz)
interconnect: 56 Gb/s (FDR) infiniband
disk: 1Tb /scratch
```

HÁSKÓLI ÍSLANDS
UPPLÝSINGATÆKNISVIÐ

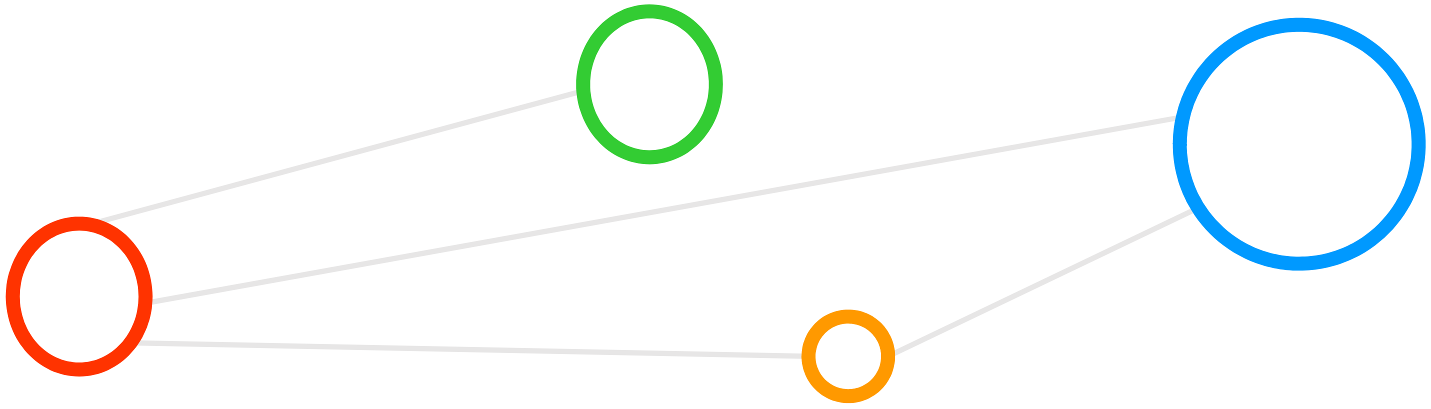
[13] Icelandic HPC Machines & Community

[Video] GPU vs. CPU Visual Demonstration



[14] Mythbuster GPU vs. CPU Video

GPU Libraries & Programming Models



Parallel Computing Example on Many-Core GPUs – A Case for using Libraries

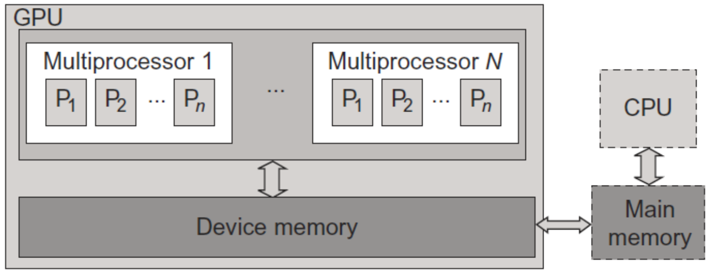
- General Purpose Graphical Processing Unit (GPGPU)
  - Designed to compute large numbers of floating point operations in parallel, but with moderate performance

- **Step 'zero': Data is loaded via the main memory of the CPU (i.e., host CPU memory) to the device memory of the GPU accessed by the many cores**

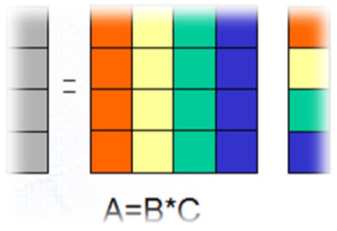
- **Step one: each GPU core has a column of matrix B (named as Bpart)**
- **Step one: each GPU core has an element of column vector C (named Cpart)**

- **Step two: Each GPU core performs an independent vector-scalar multiplication (i.e., independently based on their Bpart and Cpart contents)**

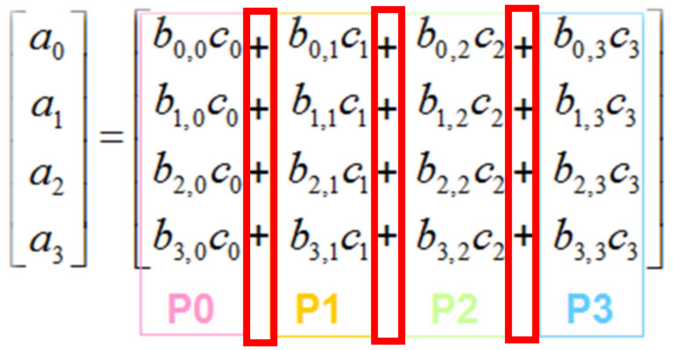
- **Step three: Each GPU core has a part of the result vector A (named Apart) and is written in device memory; results go to the main memory of CPU**



[3] Distributed & Cloud Computing Book



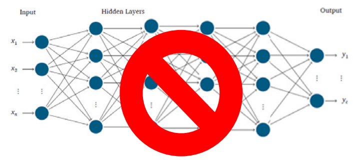
(nice parallelization possible via independent computing)

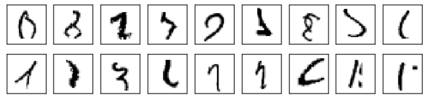


$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} b_{0,0}c_0 + b_{0,1}c_1 + b_{0,2}c_2 + b_{0,3}c_3 \\ b_{1,0}c_0 + b_{1,1}c_1 + b_{1,2}c_2 + b_{1,3}c_3 \\ b_{2,0}c_0 + b_{2,1}c_1 + b_{2,2}c_2 + b_{2,3}c_3 \\ b_{3,0}c_0 + b_{3,1}c_1 + b_{3,2}c_2 + b_{3,3}c_3 \end{bmatrix}$$

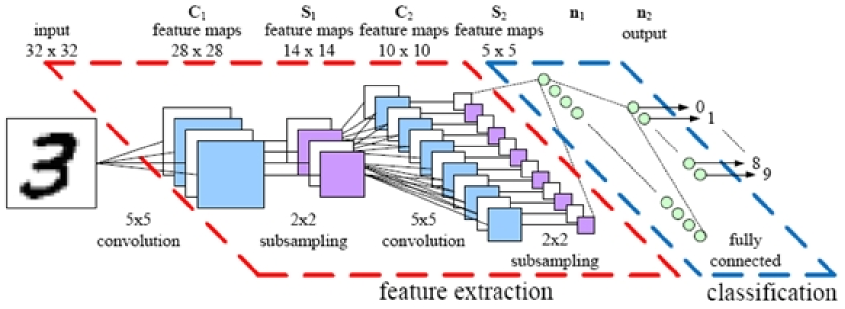
DEEP Learning takes advantage of Many-Core Technologies (cf. Lecture 1)

[4] *Neural Network 3D Simulation*





- **Innovation via specific layers and architecture types**

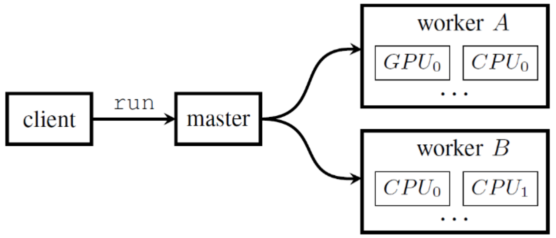


[5] A. Rosebrock

➢ **Lecture 8 will provide more details about parallel & scalable machine & deep learning algorithms and how many-core HPC is used**

Selected Deep Learning Frameworks & Tools used with GPUs

- TensorFlow
  - One of the most popular deep learning frameworks available today
  - Execution on multi-core CPUs or many-core GPUs



[17] Tensorflow Web page

- **Tensorflow is an open source library for deep learning models using a flow graph approach**
- **Tensorflow nodes model mathematical operations and graph edges between the nodes are so-called tensors (also known as multi-dimensional arrays)**
- **The Tensorflow tool supports the use of CPUs and GPUs (much more faster than CPUs)**
- **Tensorflow work with the high-level deep learning tool Keras in order to create models fast**
- **New versions of Tensorflow have Keras shipped with it as well & many further tools**

[18] Keras Web page

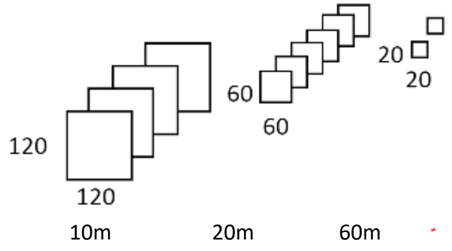
- Keras
  - Often used in combination with low-level frameworks like Tensorflow

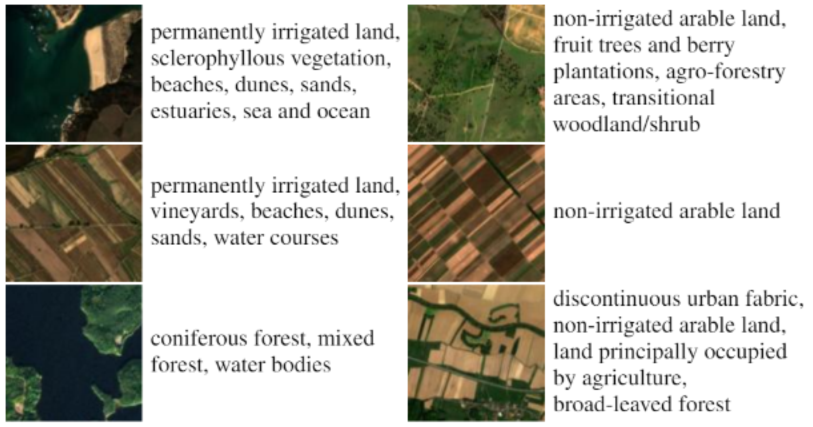
- **Keras is a high-level deep learning library implemented in Python that works on top of existing other rather low-level deep learning frameworks like Tensorflow, CNTK, or Theano**
- **Created deep learning models with Keras run seamlessly on CPU and GPU via low-level deep learning frameworks**
- **The key idea behind the Keras tool is to enable faster experimentation with deep networks**

➢ **Lecture 8 will provide more details about using Tensorflow & Keras in Deep Learning via Python for a wide variety of data science tasks**

Multispectral Remote Sensing Dataset Example (cf. Invited Lecture G. Cavallaro)

Datasets	Image type	Image per class	Scene classes	Annotation type	Total images	Spatial resolution (m)	Image sizes	Year	Ref.
BigEarthNet	Satellite MS	328 to 217119	43	Multi label	590,326	10 20 60	120x120 60x60 20x20	2018	*[9]* *G. Sumbul et al.*





permanently irrigated land, sclerophyllous vegetation, beaches, dunes, sands, estuaries, sea and ocean
non-irrigated arable land, fruit trees and berry plantations, agro-forestry areas, transitional woodland/shrub
permanently irrigated land, vineyards, beaches, dunes, sands, water courses
non-irrigated arable land
coniferous forest, mixed forest, water bodies
discontinuous urban fabric, non-irrigated arable land, land principally occupied by agriculture, broad-leaved forest

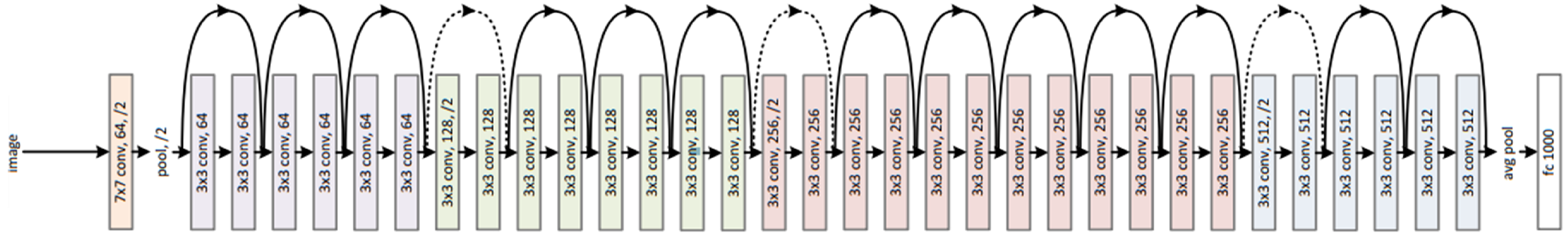
[15] G. Sumbul et al.

[16] Big Earth Net Dataset

➢ **Lecture 8 will provide more details about parallel & scalable machine & deep learning algorithms are used with remote sensing datasets**

Deep Learning via RESNET-50 Architecture

- Classification of land cover in scenes (cf. Invited Talk G. Cavallaro)
  - Very suitable for parallelization via distributed training on multi GPUs



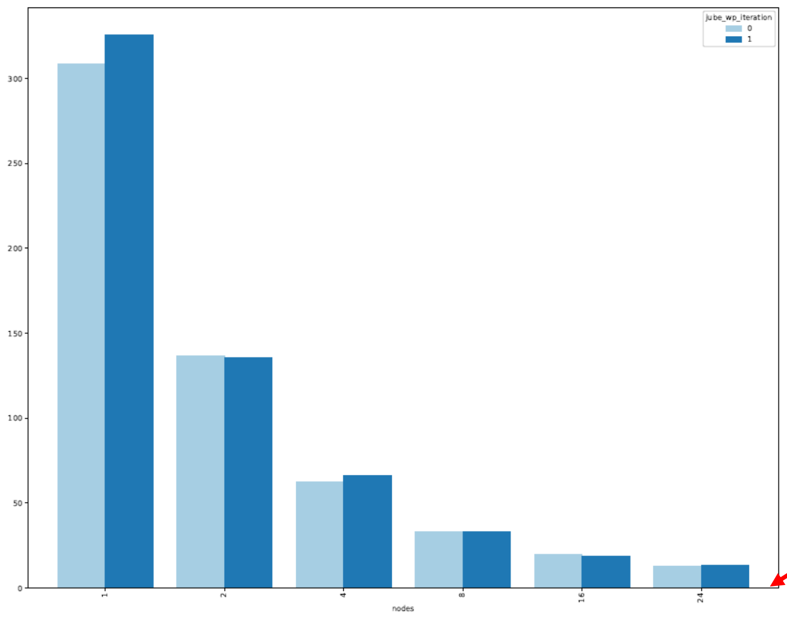
[20] RESNET

- **RESNET-50 is a known neural network architecture that has established a strong baseline in terms of accuracy**
- **The computational complexity of training the RESNET-50 architecture relies in the fact that is has ~ 25.6 millions of trainable parameters**
- **RESNET-50 still represents a good trade-off between accuracy, depth and number of parameters**
- **The setups of RESNET-50 makes it very suitable for parallelization via distributed training on multi GPUs**

➢ **Lecture 8 will provide more details about parallel & scalable machine & deep learning algorithms are used with remote sensing datasets**

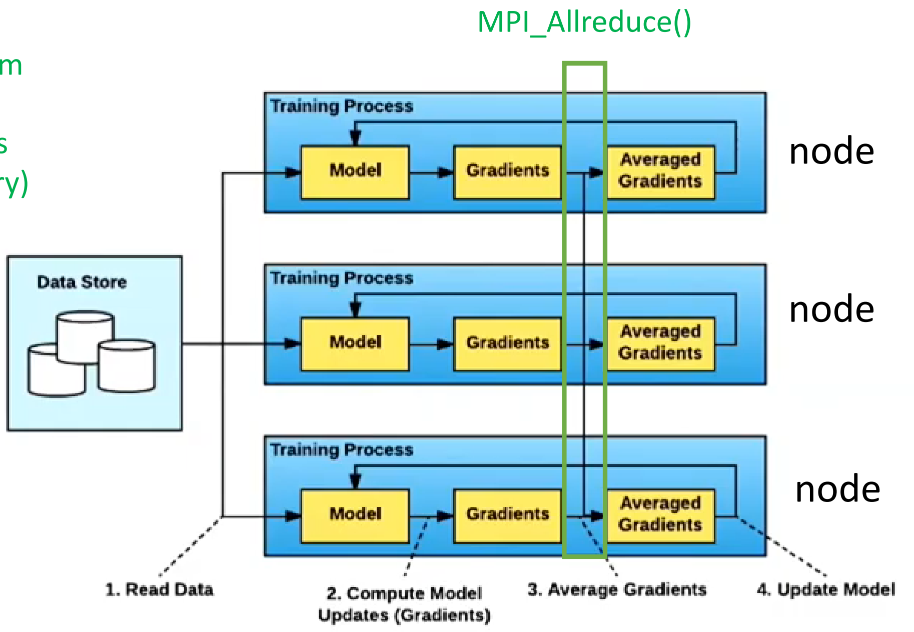
Distributed Training with Multi GPU Usage using Horovod

Time per epoch [sec]



A partition of the JUWELS system has 56 compute nodes, each with 4 NVIDIA V100 GPUs (equipped with 16 GB of memory)

24 nodes x 4 GPUs = 96 GPUs



[19] Horovod

- **Horovod is a distributed training framework used in combination with low-level deep learning frameworks like Tensorflow**
- **Horovod uses MPI for inter-process communication, e.g., MPI_Allreduce()**
- **Distributed training using data parallelism approach means: (1) Gradients for different batches of data are calculated separately on each node; (2) But averaged across nodes to apply consistent updated to the deep learning model in each node**

➢ **Lecture 8 will provide more details about using distributed training with Horovod & more examples of speed-ups with multi GPU usage**

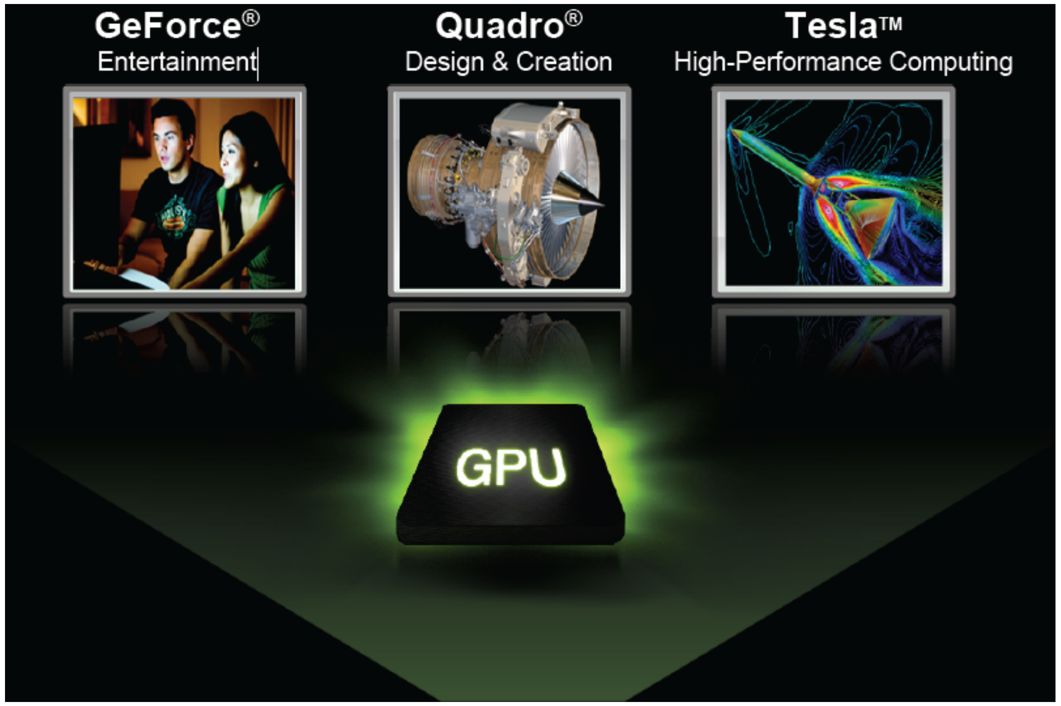
NVIDIA & Compute Unified Device Architecture (CUDA)

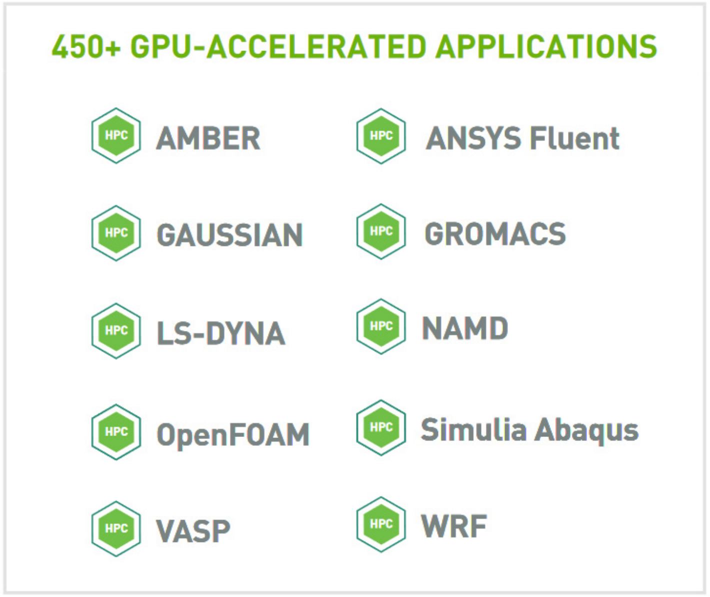
- Compute Unified Device Architecture (CUDA)
  - Industry standard programming model
  - Dominant since NVIDIA is major producer of GPGPUs in the market
  - Subset of programming language C
  - Defines a programming model and a memory model
- (Unlimited) Scalability
  - Parallel portions of application executed on the GPU device as kernels
  - Program for one thread can be instantiated on many parallel threads
  - Program runs on any number of processors without recompiling

- **NVIDIA Compute Unified Device Architecture (CUDA) is a vendor-specific programming model for NVIDIA GPUs**
- **CUDA Kernels run on NVIDIA GPUs and are written in CUDA to take advantage of many-core GPUs**

Different Types of NVIDIA GPUs & Simulation Sciences Impact with Libraries

- Example: Three 'different types of NVIDIA GPUs'
  - Designed for different levels of performance requirements





450+ GPU-ACCELERATED APPLICATIONS

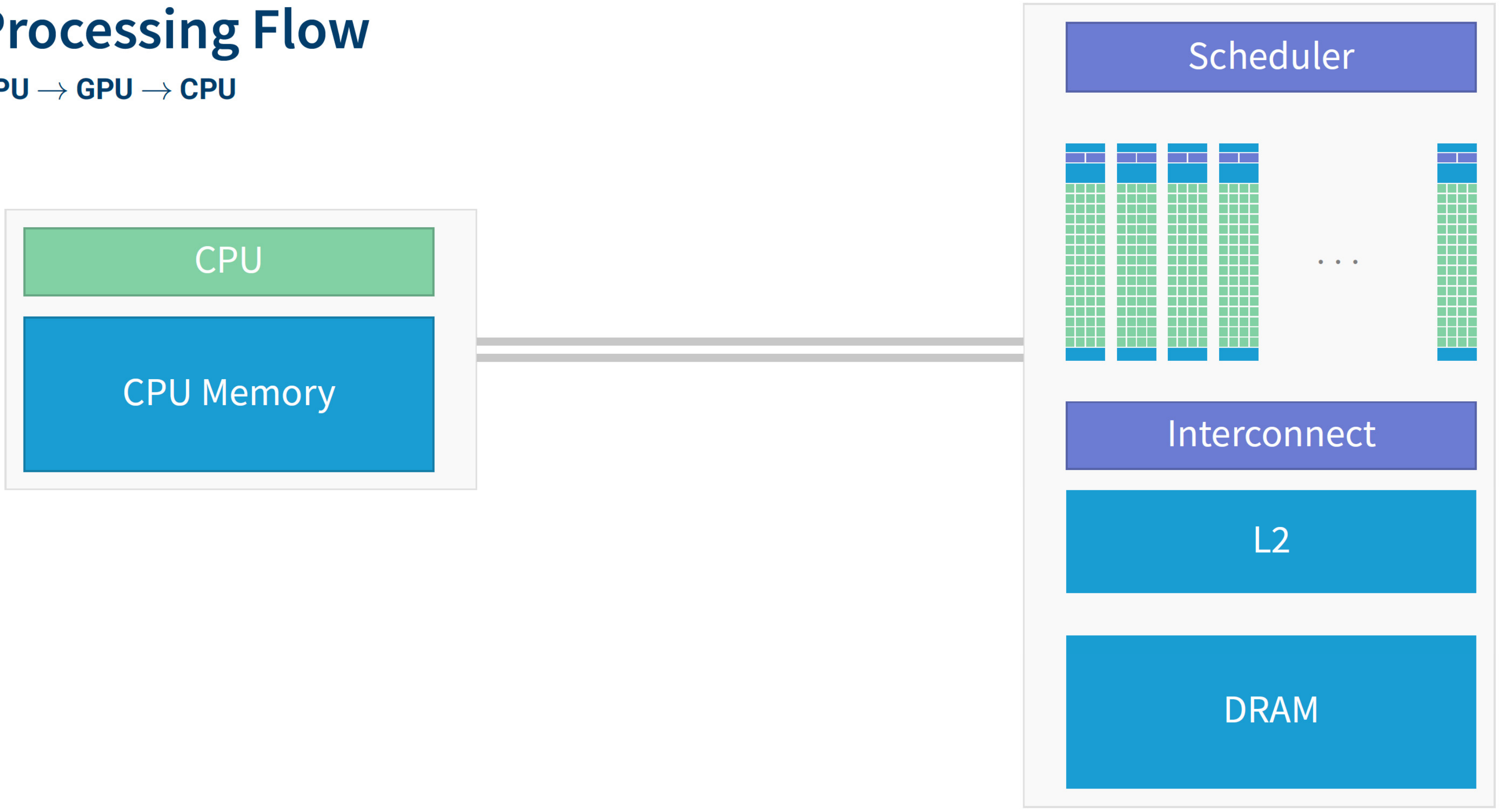
AMBER	ANSYS Fluent
GAUSSIAN	GROMACS
LS-DYNA	NAMD
OpenFOAM	Simulia Abaqus
VASP	WRF

[25] NVIDIA Training

Working with GPUs (1)

Processing Flow

CPU $\longrightarrow$ GPU $\longrightarrow$ CPU

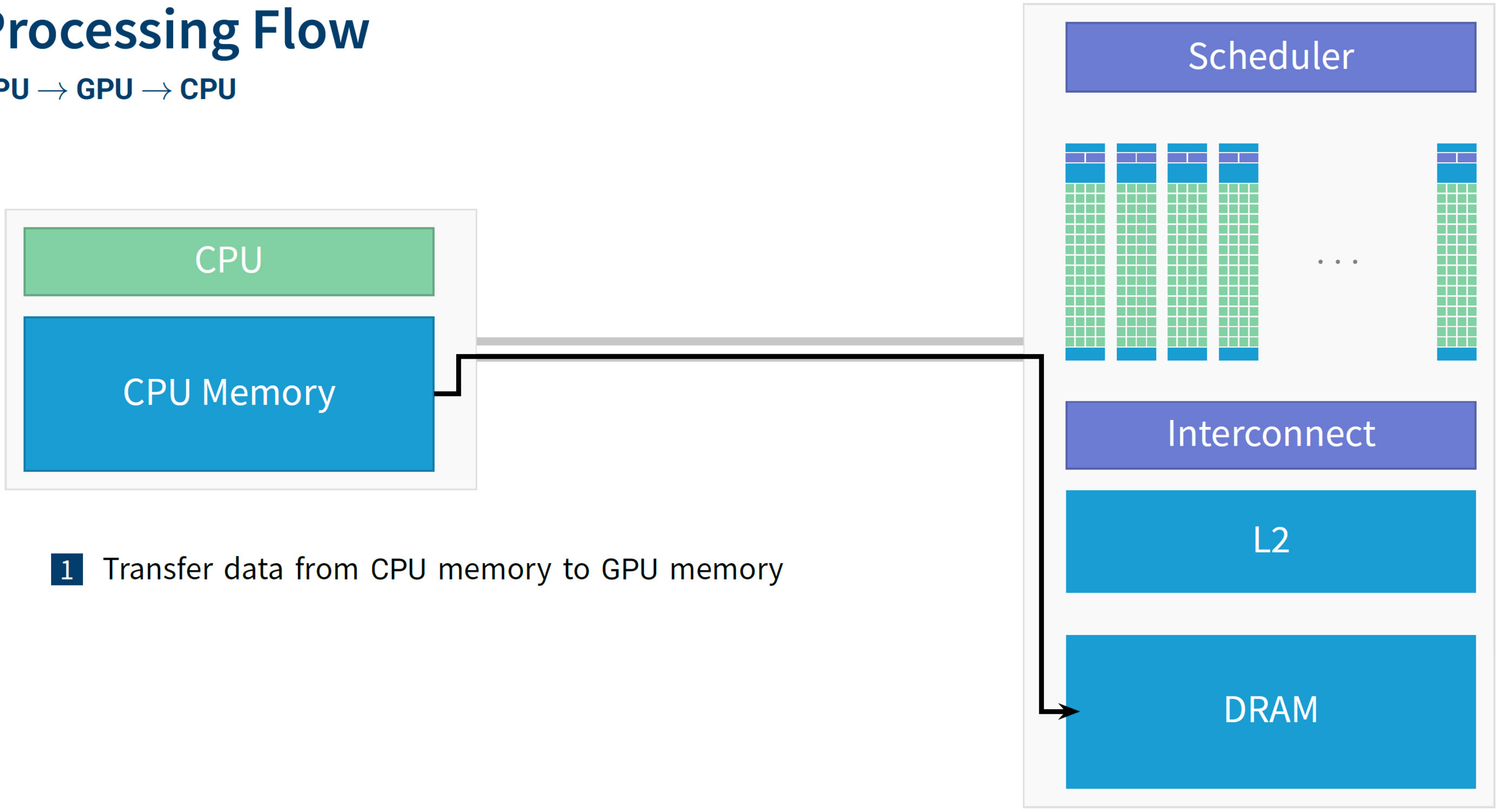


[21] JSC GPU Course

Working with GPUs (2)

Processing Flow

CPU $\longrightarrow$ GPU $\longrightarrow$ CPU



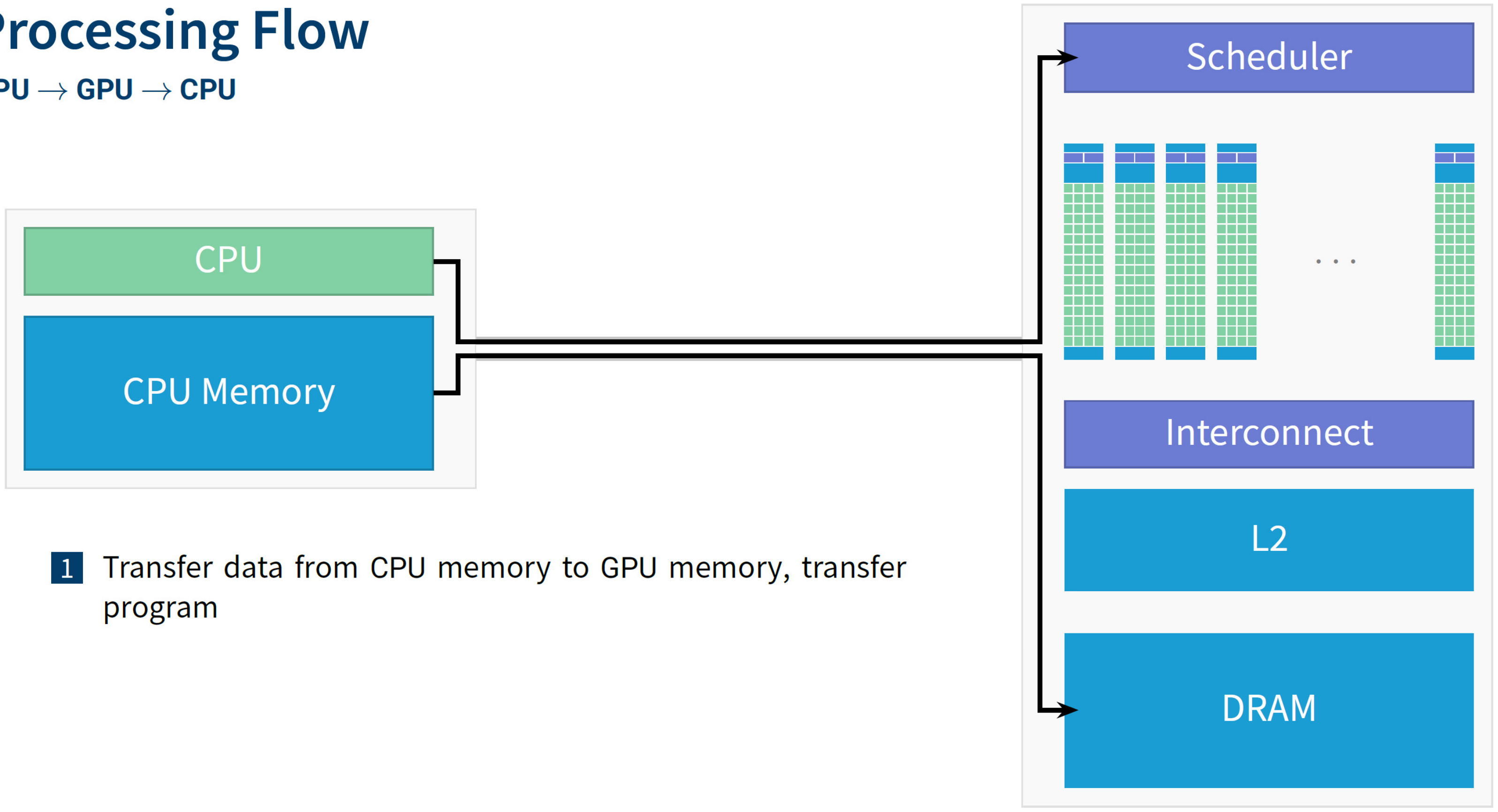
1 Transfer data from CPU memory to GPU memory

[21] JSC GPU Course

Working with GPUs (3)

Processing Flow

CPU → GPU → CPU



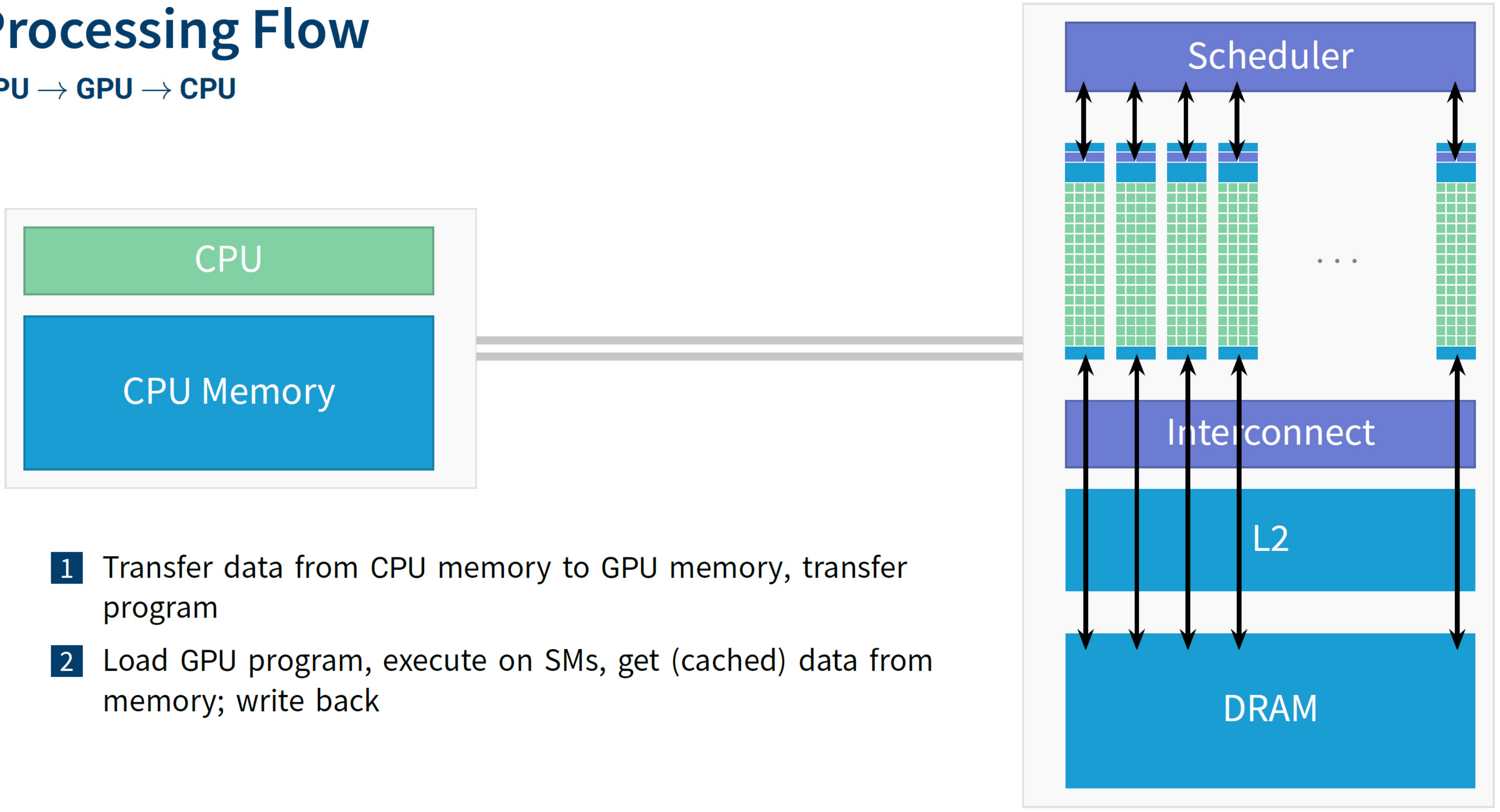
1. Transfer data from CPU memory to GPU memory, transfer program

[21] JSC GPU Course

Working with GPUs (4)

Processing Flow

CPU → GPU → CPU



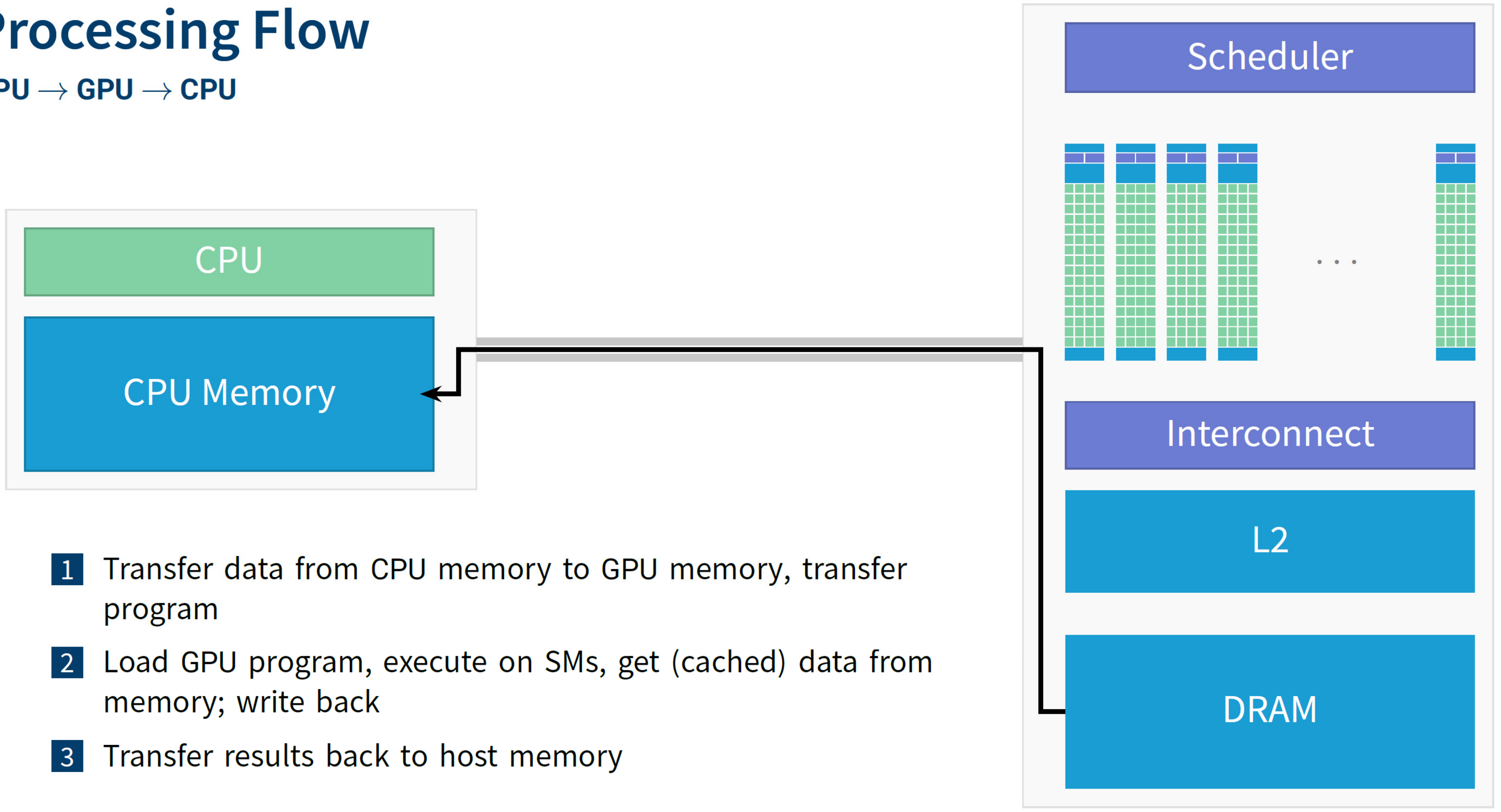
1. Transfer data from CPU memory to GPU memory, transfer program
2. Load GPU program, execute on SMs, get (cached) data from memory; write back

[21] JSC GPU Course

Working with GPUs (5)

Processing Flow

CPU → GPU → CPU



1. Transfer data from CPU memory to GPU memory, transfer program
2. Load GPU program, execute on SMs, get (cached) data from memory; write back
3. Transfer results back to host memory

[21] JSC GPU Course

Library cuBLAS for Parallel Algebra using GPUs (1)

- Standard basic linear algebra subroutines (BLAS)
  - Specification of 152 routines for linear algebra
  - Famous example: SAXPY

SAXPY: $\vec{y} = a\vec{x} + \vec{y}$, with single precision

- cuBLAS Library
  - Freely available as part of the CUDA Toolkit and OpenACC Toolkit

(simple CPU example version)

```c
void saxpy(int n, float a, float * x, float * y) {
  for (int i = 0; i < n; i++)
    y[i] = a * x[i] + y[i];
}

int a = 42;
int n = 10;
float x[n], y[n];
// fill x, y

saxpy(n, a, x, y);
```

- **Standard Basic Linear Algebra Subroutines (BLAS) represents a specification for low-level 152 routines for performing common linear algebra operations such as vector addition, scalar multiplication, dot prodcuts, linear combinations, and matrix multiplication**
- **cuBLAS is a library that enables parallel algebra using GPUs supporting all 152 BLAS routines**
- **SAXPY stands for single precision y = a * x + y**

[23] cuBLAS

Library cuBLAS for Parallel Algebra using GPUs (2)

```
int a = 42;  int n = 10;
float x[n], y[n];
// fill x, y

cublasHandle_t handle;
cublasCreate(&handle);

float * d_x, * d_y;
cudaMallocManaged(&d_x, n * sizeof(x[0]));
cudaMallocManaged(&d_y, n * sizeof(y[0]));
cublasSetVector(n, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector(n, sizeof(y[0]), y, 1, d_y, 1);

cublasSaxpy(n, a, d_x, 1, d_y, 1);

cublasGetVector(n, sizeof(y[0]), d_y, 1, y, 1);

cudaFree(d_x); cudaFree(d_y);
cublasDestroy(handle);
```

Initialize

Allocate GPU memory

Copy data to GPU

Call BLAS routine

Copy result to host

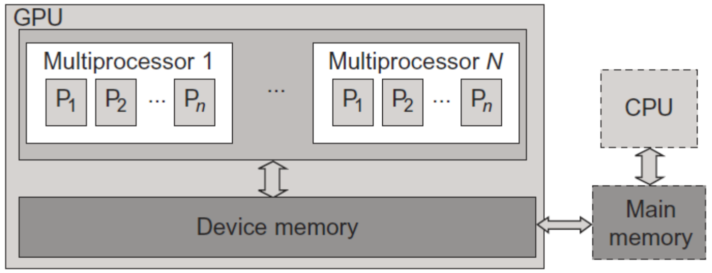
Finalize

[21] JSC GPU Course

Recent Support of OpenMP for Programming GPUs with Directives (cf. Lecture 6)

```
...
#pragma omp target map (tofrom:y), map(to:x)
#pragma omp teams num_teams(10) num_threads(10)
#pragma omp distribute
for (...) {
  ...
  #pragma omp parallel for
  for (...) {
  }
  ...
}
...
```

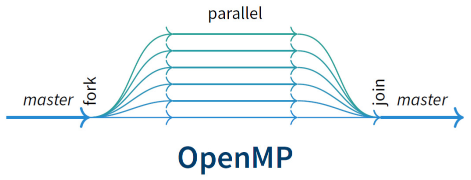
- OpenMP is the de-facto standard for multi-threaded programming on CPU
- OpenMP includes since version 4.0 (better since 4.5) also capabilities for programming GPUs

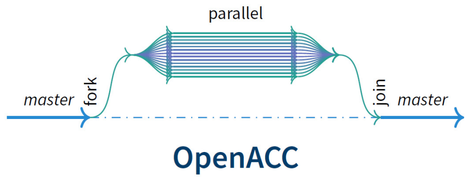


[9] Distributed & Cloud Computing Book

- OpenACC is similar to OpenMP, because it is modeled after OpenMP, but for accelerators

```
#pragma acc data copyout(y[0:N]) create(x[0:N])
{
double sum = 0.0;
#pragma acc parallel loop
for (int i=0; i<N; i++) {
    x[i] = 1.0;
    y[i] = 2.0;
}
#pragma acc parallel loop
for (int i=0; i<N; i++) {
    y[i] = i*x[i]+y[i];
}
}
```

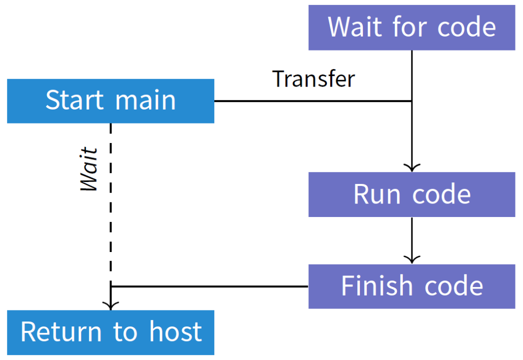




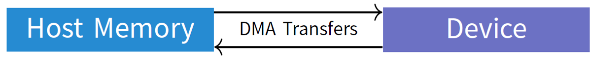
[21] JSC GPU Course

SAXPY with OpenACC Accelerator Model using GPUs

- OpenACC Accelerator Model
  - For computation & memory spaces
- Two separate memory spaces
  - Needs transfers back and forth
  - Transfers hidden from programmer
  - Memories are not coherent
  - Compiler & GPU runtime helps to make it easy for the developers



[21] JSC GPU Course



```
void saxpy_acc(int n, float a, float * x, float * y) {
  #pragma acc parallel loop copy(y) copyin(x)
  for (int i = 0; i < n; i++)
    y[i] = a * x[i] + y[i];
}

int a = 42;
int n = 10;
float x[n], y[n];
// fill x, y

saxpy_acc(n, a, x, y);
```

- **The OpenACC Accelerator model enables an easy acceleration of code elements with compiler directives similiar like OpenMP**
- **The OpenACC Accelerator model executes the main program on the host and the device code is transferred to accelerator**
- **In the openACC Accelerator model the execution on the accelerator is started and the host waits until return (exceptions with async)**

Impact & Changes of GPU Vendors – Example

- NVIDIA GPUs
  - Market dominance and most codes are written in NVIDIA CUDA C/C++
- AMD Radeon Instinct GPUs
  - Frontier Supercomputer expected to be fastest supercomputer with 1.5 exaflops of peak processing
  - Frontier will employ AMD Epyc CPUs and Radeon Instinct GPUs (CUDA is not supported by AMD hardware)

[21] JSC GPU Course

Heterogeneous Compute Interface for Portability (HIP)

- Selected Facts
  - Open source
  - Part of AMD ROCm software stack
  - Almost no performance penalty on NVIDIA GPUs + works on AMD GPUs
  - Includes tools to convert existing CUDA codes
  - Not a drop-in replacement for CUDA & developer knowledge is necessary for porting

CUDA	HIP
Thread	Thread/ Work item
Thread block	Workgroup
Warp	Wavefront
Global Memory	Global Memory
Shared memory	Local memory
Local memory	Private memory
Streaming Multiprocessor	Compute Unit

CUDA

```
__global__ void vector_add
(float *out, float *a, float *b, int n)
{
 int i= blockDim.x*blockIdx.x+threadIdx.x;
 if (i<n)
     {
      out[i] = a[i] + b[i];
     }
}
```

HIP

```
 __global__ void vector_add
(float *out, float *a, float *b, int n)
{
 int i= blockDim.x*blockIdx.x+threadIdx.x;
 if (i<n)
     {
      out[i] = a[i] + b[i];
     }
 }
```

[21] JSC GPU Course

- **Heterogeneous Compute Interface for Portability (HIP) is a C++ runtime API & Kernel language to write portable codes for AMD & NVIDIA GPUs**
- **The HIP kernel language is similar to CUDA in order to be easy for CUDA developers since most of the GPU codes are written in CUDA today**

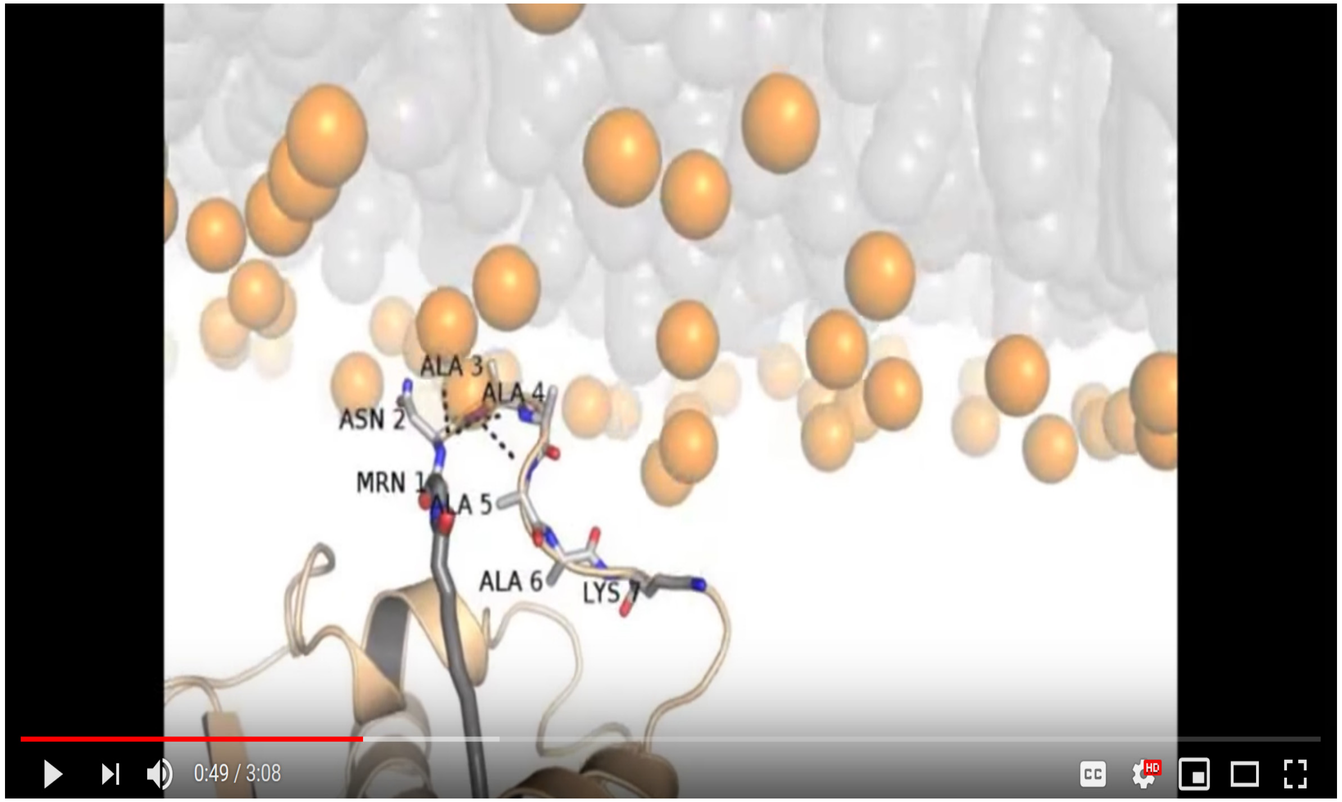
HIP vs CUDA Example

```
int main(){
float *a, *b, *out;
float *a_d, *b_d, *out_d;
a = (float*)malloc(sizeof(float) * N);
//same for b and out then fill a and b
cudaMalloc((void**)&a_d
        , sizeof(float) * N);
//same for b_d and out_d
cudaMemcpy(a_d, a, sizeof(float)
          * N, cudaMemcpyHostToDevice);
//same for b_d
vector_add
      <<<gridsize,blocksize,sh_memsize,stream>>>
      (out_d, a_d, b_d, N);
cudaMemcpy(out, out_d, sizeof(float)*N,
cudaMemcpyDeviceToHost);}
```

```
int main(){
float *a, *b, *out;
float *a_d, *b_d, *out_d;
a = (float*)malloc(sizeof(float) * N);
//same for b and out then fill a and b
hipMalloc((void**)&a_d
        , sizeof(float) * N);
//same for b_d and out_d
hipMemcpy(a_d, a, sizeof(float)
          * N, hipMemcpyHostToDevice);
//same for b_d
hipLaunchKernelGGL(vector_add,
      gridsize,blocksize,sh_memsize, stream,
      out_d, a_d, b_d, N);
hipMemcpy(out, out_d, sizeof(float)*N,
hipMemcpyDeviceToHost);}
```

[21] JSC GPU Course

[Video] Simulation Sciences Impact: AMBER Tool with GPUs



[24] Amber with GPUs Video

Lecture Bibliography

Lecture Bibliography (1)

- [1] M. Goetz, C. Bodenstein, M. Riedel, 'HPDBSCAN – Highly Parallel DBSCAN', in proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC2015), Machine Learning in HPC Environments (MLHPC) Workshop, 2015, Online: https://www.researchgate.net/publication/301463871_HPDBSCAN_highly_parallel_DBSCAN
- [2] TOP500 Supercomputing Sites, Online: http://www.top500.org/
- [3] K. Hwang, G. C. Fox, J. J. Dongarra, 'Distributed and Cloud Computing', Book, Online: http://store.elsevier.com/product.jsp?locale=en_EU&isbn=9780128002049
- [4] YouTube Video, 'Neural Network 3D Simulation', Online: https://www.youtube.com/watch?v=3JQ3hYko51Y
- [5] A. Rosebrock, 'Get off the deep learning bandwagon and get some perspective', Online: http://www.pyimagesearch.com/2014/06/09/get-deep-learning-bandwagon-get-perspective/
- [6] DEEP Projects Web page, Online: http://www.deep-projects.eu/
- [7] OakRidge Supercomputer Summit, Online: https://oakridgetoday.com/2019/06/17/ornls-summit-remains-worlds-most-powerful-supercomputer/
- [8] Summit Supercomputer Architecture Overview, Online: https://www.olcf.ornl.gov/wp-content/uploads/2019/05/Summit_System_Overview_20190520.pdf
- [9] NVIDIA Tesla Volta v100, Online: https://www.nvidia.com/en-us/data-center/tesla-v100/
- [10] NVIDIA NVLink/NVSwitch, Online: https://www.nvidia.com/en-us/data-center/nvlink/

Lecture Bibliography (2)

- [11] NVIDIA GEFORCE Gaming Industry, Online:
  https://www.nvidia.com/en-eu/geforce/
- [12] JEDEC on HBM, Online:
  https://www.jedec.org/news/pressreleases/jedec-updates-groundbreaking-high-bandwidth-memory-hbm-standard
- [13] Icelandic HPC Machines & Community, Online:
  http://ihpc.is
- [14] YouTube Video, 'Mythbusters Demo GPU versus CPU', Online:
  https://www.youtube.com/watch?v=-P28LKWTzrI
- [15] G. Sumbul, M. Charfuelan, B. Demir, V. Markl, BigEarthNet: A Large-Scale Benchmark Archive for Remote Sensing Image Understanding, IEEE International Conference on Geoscience and Remote Sensing Symposium, Yokohama, Japan, 2019.
- [16] Tensorflow Dataset 'Big Earth Net', Online:
  https://www.tensorflow.org/datasets/datasets#bigearthnet
- [17] Tensorflow, Online:
  https://www.tensorflow.org/
- [18] Keras Python Deep Learning Library, Online:
  https://keras.io/
- [19] Horovod: Uber's Open Source Distributed Deep Learning Framework for TensorFlow, Online:
  https://www.slideshare.net/databricks/horovod-ubers-open-source-distributed-deep-learning-framework-for-tensorflow
- [20] Kaiming He et al., 'Deep Residual Learning for Image Recognition', Online:
  https://arxiv.org/pdf/1512.03385.pdf

Lecture Bibliography (3)

- [21] A. Heerten et al., Introduction to GPU Programming JSC Course
- [22] CPU/GPU Performance Comparison, Online:
  https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardwarecharacteristics-over-time/
- [23] cuBLAS Parallel Algebra Library, Online:
  https://developer.nvidia.com/cublas
- [24] AMBER and GPUs, Online:
  https://www.youtube.com/watch?v=EcQCxU7wM38
- [25] SDSC, Nvidia Training – Introduction, Online:
  http://www.sdsc.edu/us/training/assets/docs/NVIDIA-01-Intro.pdf

Data
Science
Big
Research
science
big
analysis
methods
simulations
processing
technologies
TB
European
computing
Supercomputing
Scientific
computational
research
HPC
Juelich
Climate
modelling
structures
System
Computational
E.g
Centre
climate
Computing
nodes
centers
PB
community
Environment
scientific
storage
ENES
EUDAT
increasing
databases
brain
Project
datasets
performance
images
often
Service
data-intensive
distributed
large-scale
access
management
Hardware
DLCL
DLCLs
services
measurement
within
fusion
new
funding
concepts
Policy-based
forms
cross-disciplinary
resources
year
like
Cross-Disciplinary
EU
device
Analysis
Computer
expertise
disciplines
results
Approaches
construction
Fusion
Enable
Modelling
different
important
Key
Infrastructure
files
basis
DKRZ
Use
High
Test
etc
via
physical
IPCC
Common
Methods
Using
Work
GB
MapReduce
create
formats
model
international
SMAQ
engineering
References
codes
use
JSC
e.g
acquisition
Level
stored
Resources
analyze
Large
Summary
Matters
Concrete
external
levels
system
Landscape
compute
cores
hundreds
Nutshell
Services
network
AR5
Technologies
Earth
directory
Structure
General
current
project
Understanding
Germany
Provide
NASA
Energy
Simulation
systems
day
Processing
Health