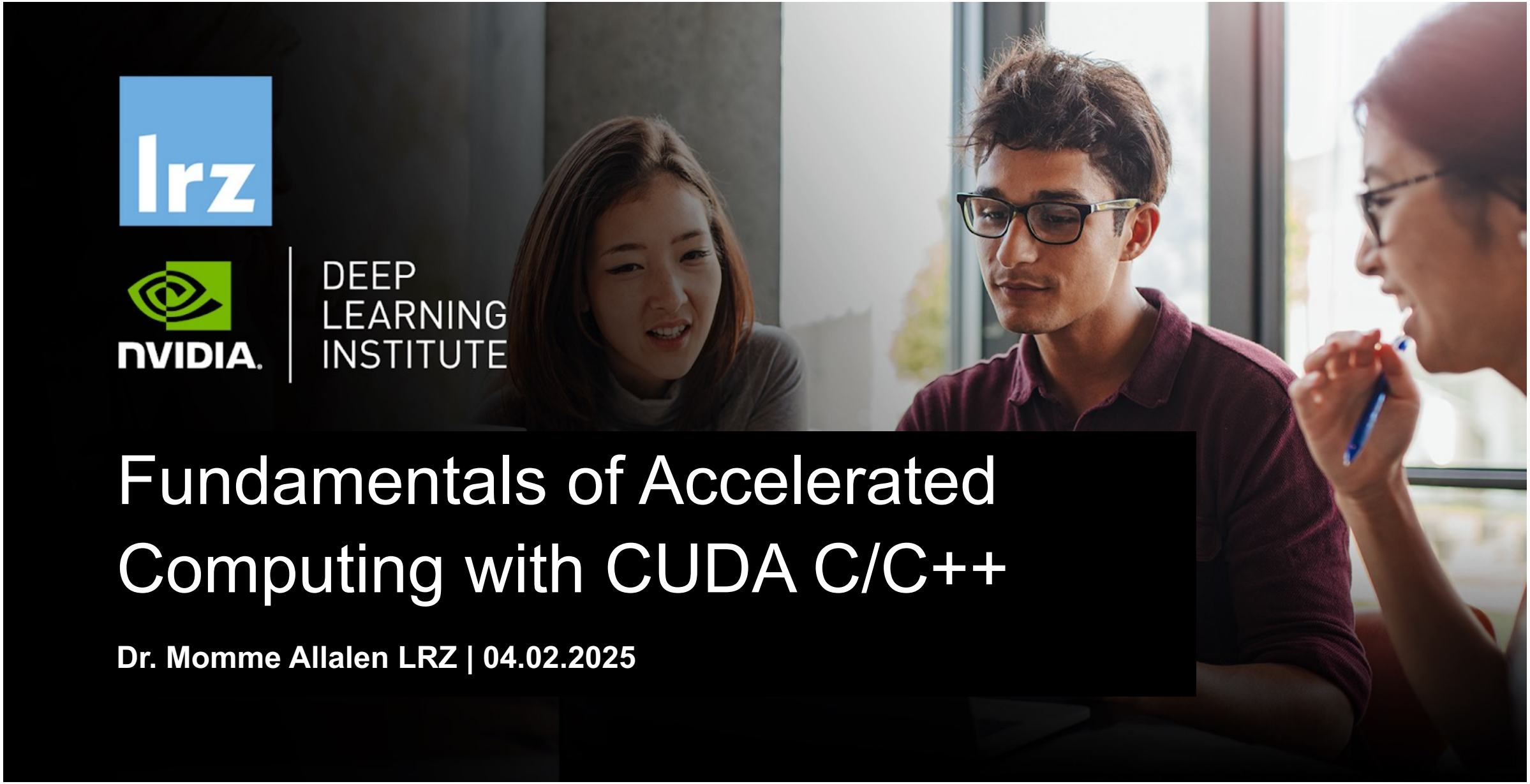
Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften

DEEP
LEARNING
INSTITUTE



Fundamentals of Accelerated Computing with CUDA C/C++

Dr. Momme Allalen LRZ | 04.02.2025

Overview

- The workshop is co-organized by LRZ, Erlangen National High Performance Computing Center (NHR@FAU) and NVIDIA Deep Learning Institute (DLI).

- NVIDIA Deep Learning Institute (DLI) offers hands-on training for developers, data scientists, and researchers looking to solve challenging problems with deep learning.

- The lectures are interleaved with many hands-on sessions using Jupyter Notebooks. The exercises will be done on a fully configured GPU-accelerated workstation in the cloud.

DEEP LEARNING INSTITUTE

DLI Mission: Help the world to solve the most challenging problems using AI and deep learning

We help developers, data scientists and engineers to get started in architecting, optimizing, and deploying neural networks to solve real-world problems in diverse industries such as autonomous vehicles, healthcare, robotics, media & entertainment and game development.

Fundamentals of Accelerated Computing with CUDA C/C++

- You learn the basics of **CUDA C/C++** by:

  - Accelerating CPU-only applications to run their latent parallelism on GPUs.
  - Utilizing essential **CUDA memory** management techniques to optimize accelerated applications
  - Exposing accelerated application potential for concurrency and exploiting it with **CUDA streams**
  - Leveraging command line and visual profiling to guide and check your work.

  - Upon completion, you'll be able to accelerate and optimize existing C/C++ CPU-only applications using the most essential **CUDA tools** and techniques. You'll understand an iterative style of CUDA development that will allow you to ship accelerated applications fast.

Tentative Agenda

10:00-10:20 Introduction to GPU Programming

10:20-12:00 Accelerating Applications with **CUDA C/C++**

12:00-13:00 Lunch break

13:00-14:20 Managing Accelerated Application Memory with **CUDA** Unified Memory and **nsys**

14:20-14:30 Coffee break

14:30-15:50 Asynchronous Streaming and Visual Profiling for Accelerated Applications with **CUDA C/C++**

15:50-16:00 Q&A and Final Remarks

Workshop Webpage

- **Lecture material will be made available under**:
  - https://tinyurl.com/hdli1w24
- **Access CUDA C/C++ Code** :
  - See the **Chat Window**

Training Setup

- To get started, follow these steps:
- Create an NVIDIA Developer account at https://learn.nvidia.com/join Select "Log in with my NVIDIA Account" and then '"Create Account".

- If you use your own laptop, make sure that WebSockets works for you:
  Test your Laptop at http://websocketstest.com
  - Under ENVIRONMENT, confirm that '"WebSockets" is checked yes.
  - Under WEBSOCKETS (PORT 80]. confirm that "Data Receive", "Send", and "Echo Test" are checked yes.
  - If there are issues with WebSockets, try updating your browser.
    We recommend Chrome, Firefox, or Safari for an optimal performance.

- Visit https://learn.nvidia.com/dli-event and enter the event code provided by the instructor.
- You're ready to get started.

And now ….

DEEP
LEARNING
INSTITUTE

Enjoy the course !

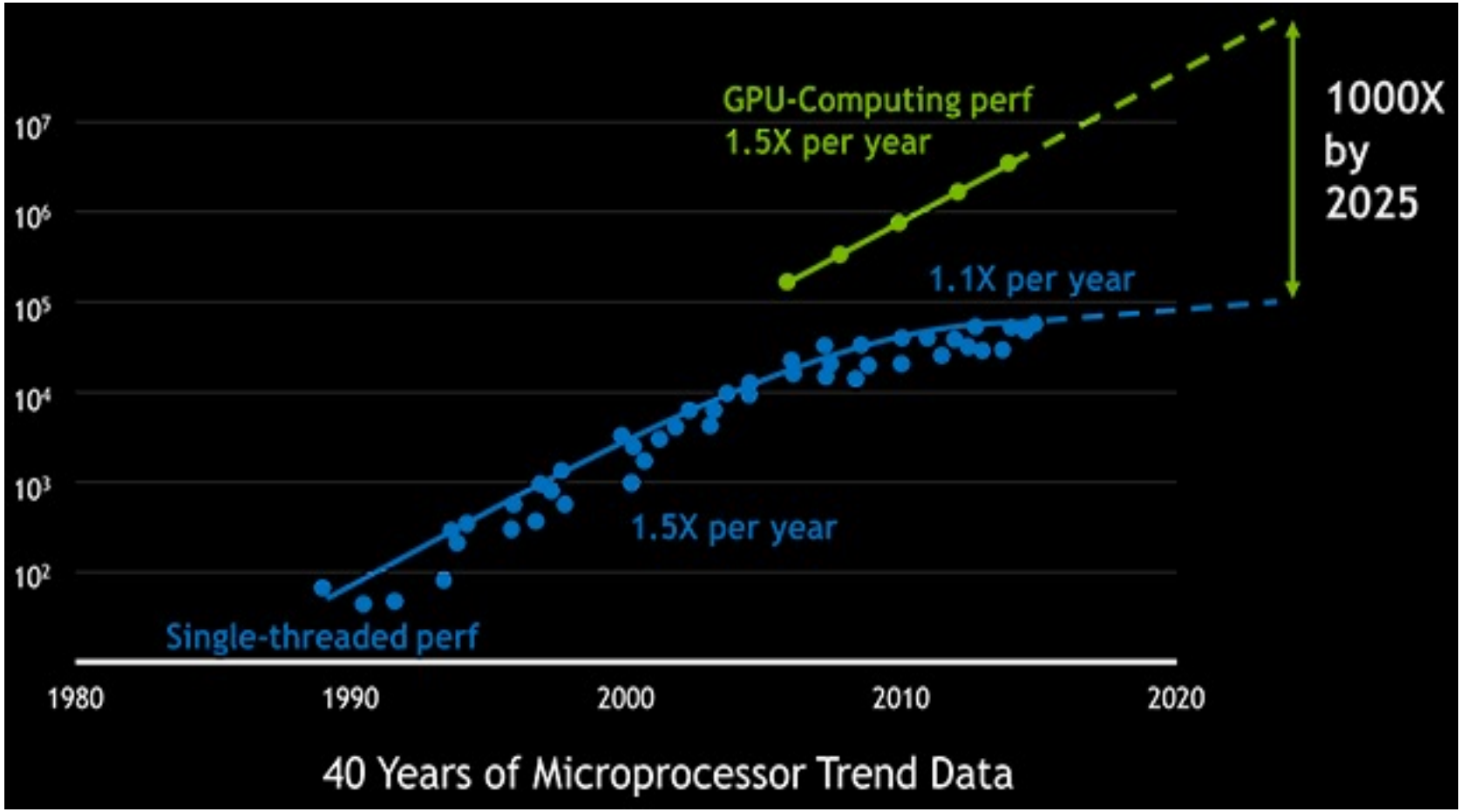
Why do we need to program for GPU?

Moore's law is dead !

The long-held notion that the processing power of computers increases exponentially every couple of years has hit its limit….

The free lunch is over..

Future is parallel !



Accelerated Systems

- Systems that use specialized hardware like GPUs to boost performance of applications.
- Originally designed for graphics, now used for parallel processing tasks.
- Crucial for applications requiring high computational power, such as AI, HPC simulations, and big data problems.

Why do we need to program for GPU?

- Typical example Intel chip: Core i7 Gen Kaby Lake processors
  - 4*CPU cores
  - With hyperthreading
  - Each with 8-wide AVX instructions
  - GPU with 1280 processing elements

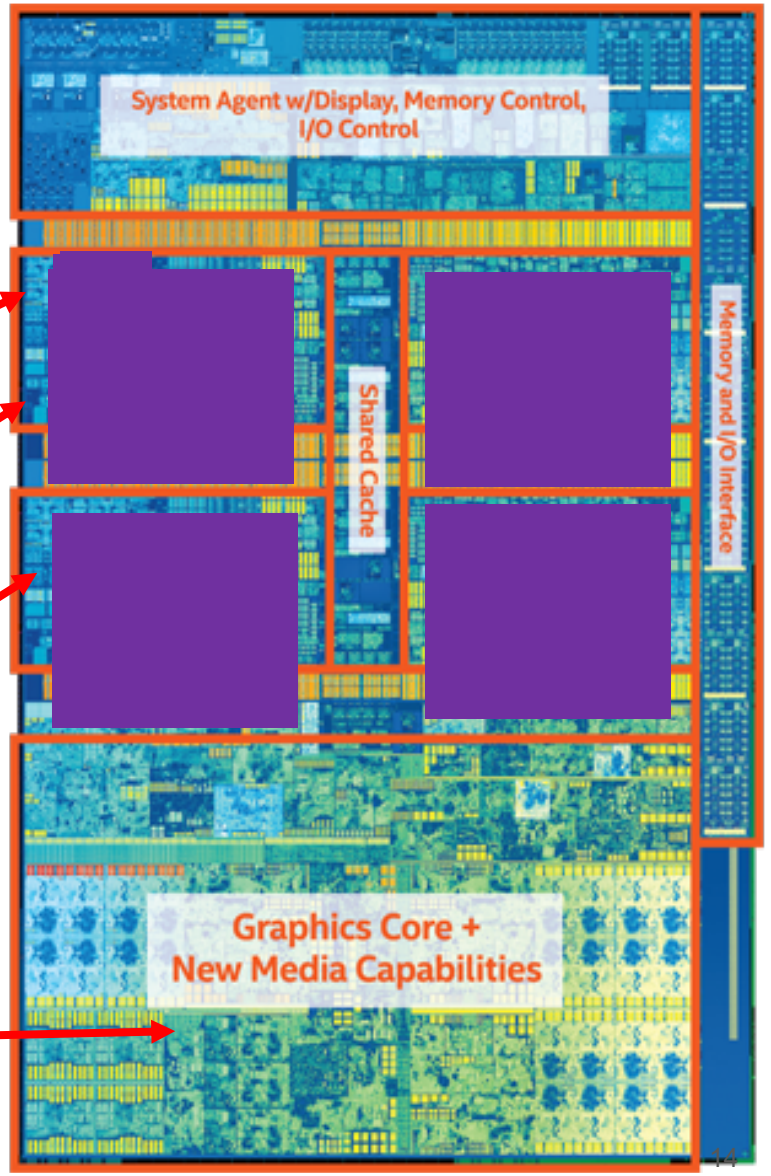
- Programming on chip:

- Serial C/C++ .. Code only takes advantage of a very small amount of the available resources of the chip

- Using vectorisation allows you to fully utilise the resources of a single hyper-thread

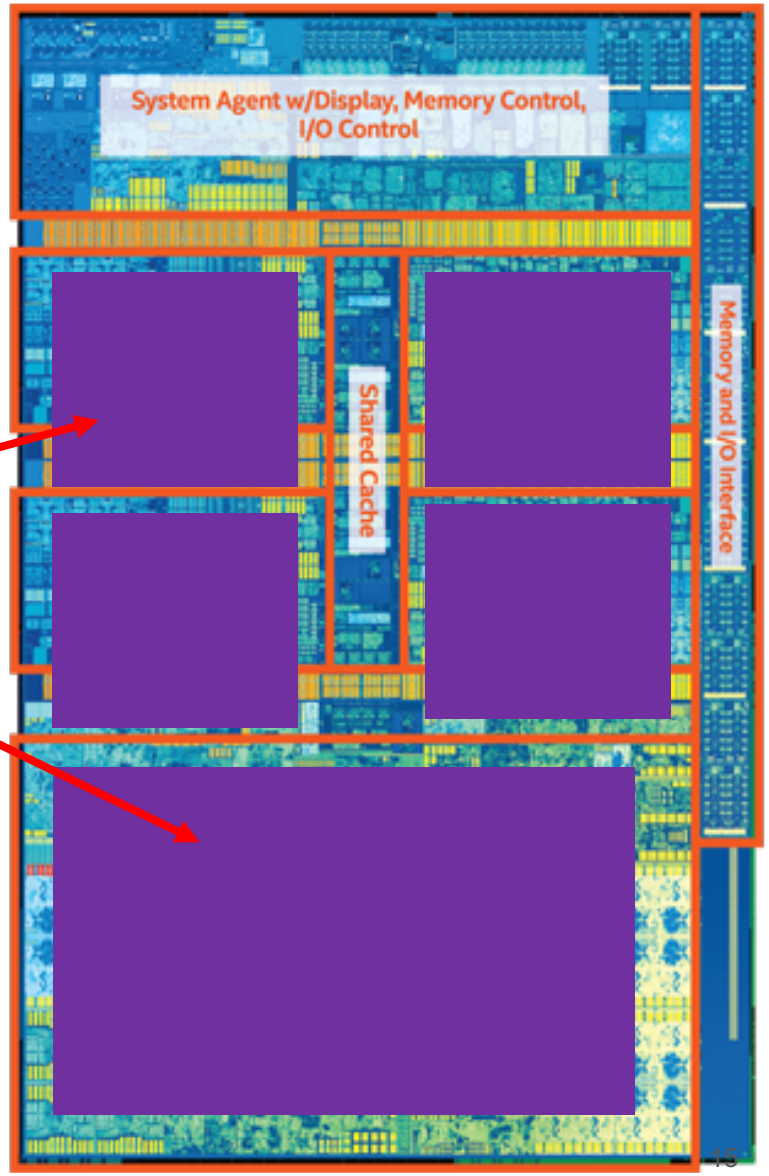
- Using multi-threading allows you to fully utilise all CPU cores

GPU need to be used ?



Why do we need to program for GPU?

- **Using heterogeneous programming allows you to dispatch and fully utilise the entire chip**

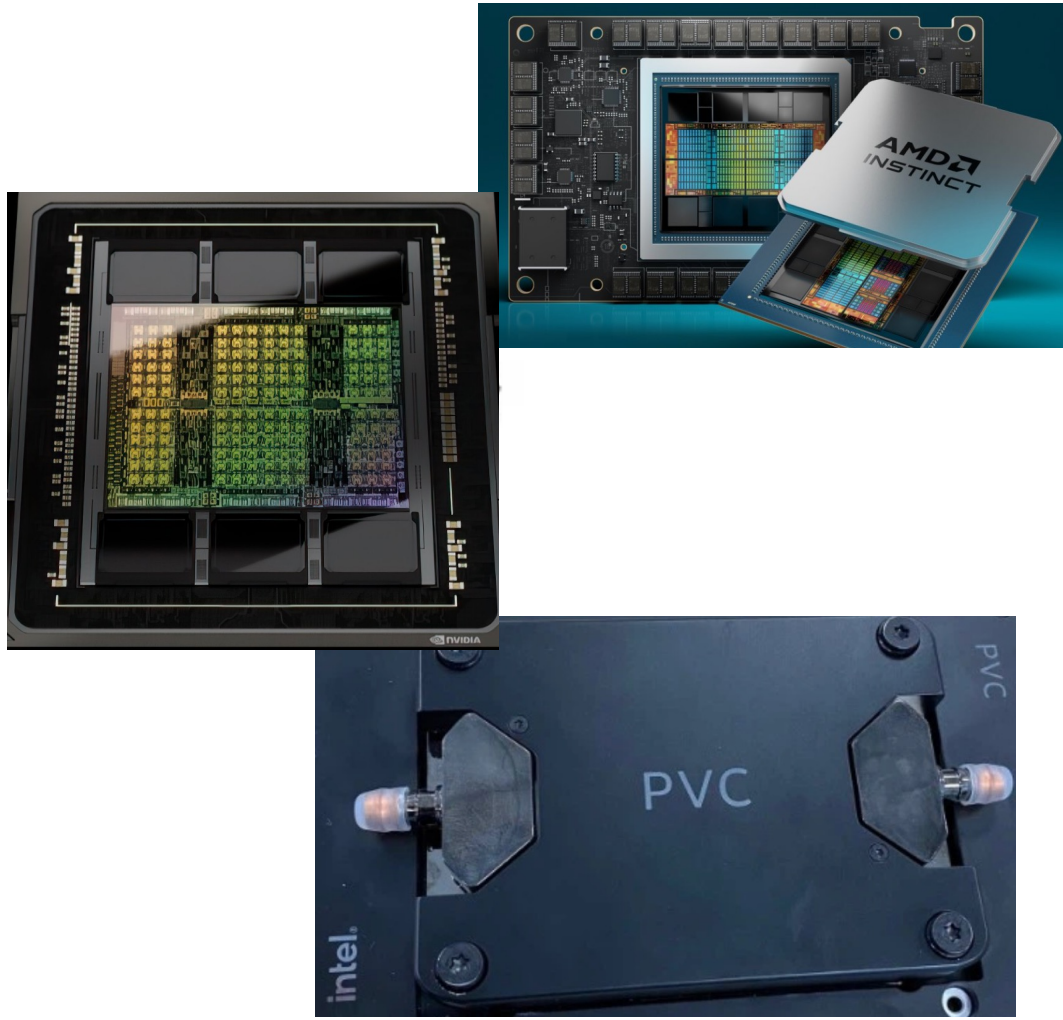


Why do we need to program for GPU?

GPU programming:

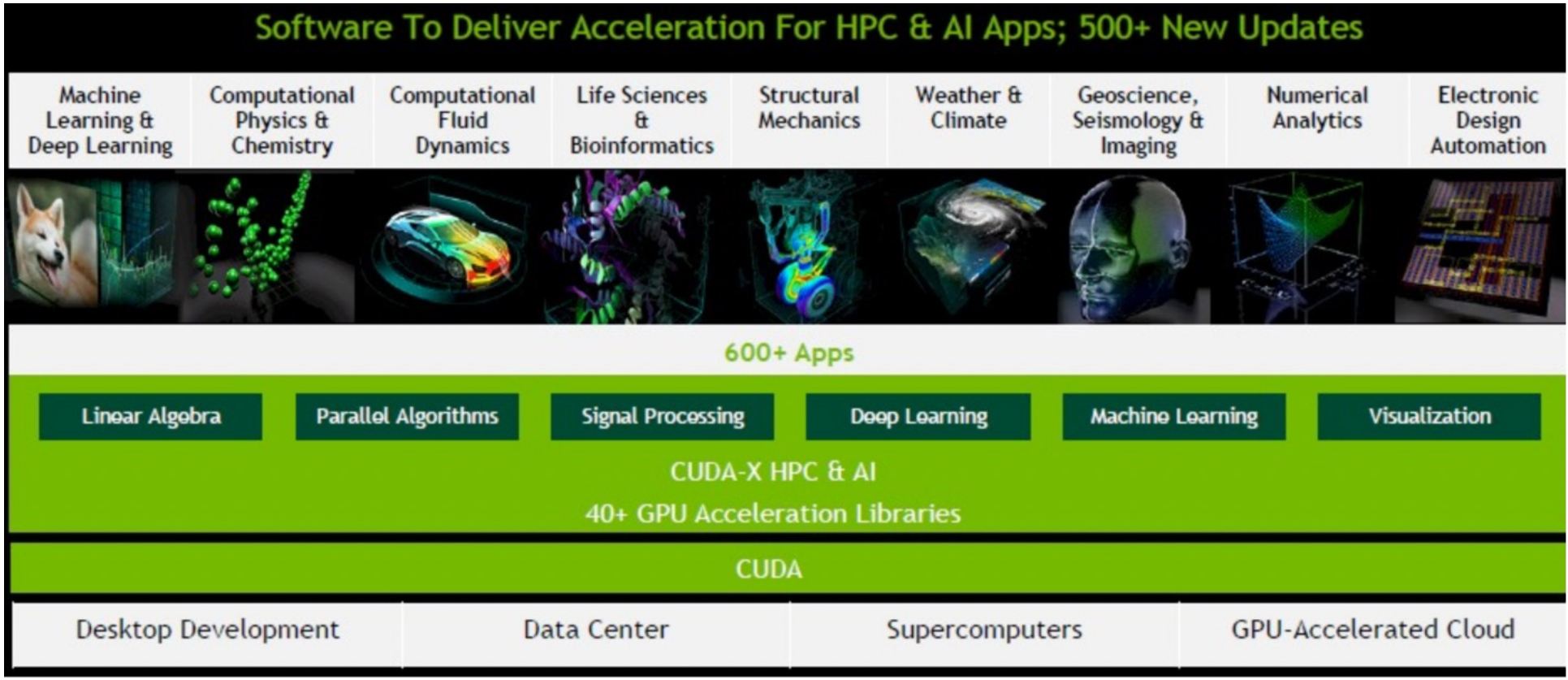
-Limited only to a specific domain
-Separate source solutions
-Verbose low Levels APIs

- **oneAPI & DPC++**
- **CUDA C/C++**
- **HIP**
- **SYCL**
- **OpenCL**
- **Kokkos**
- **HPX**
- **DirectCompute, Vulkan, Metal …**



Why do we need GPUs on HPC?

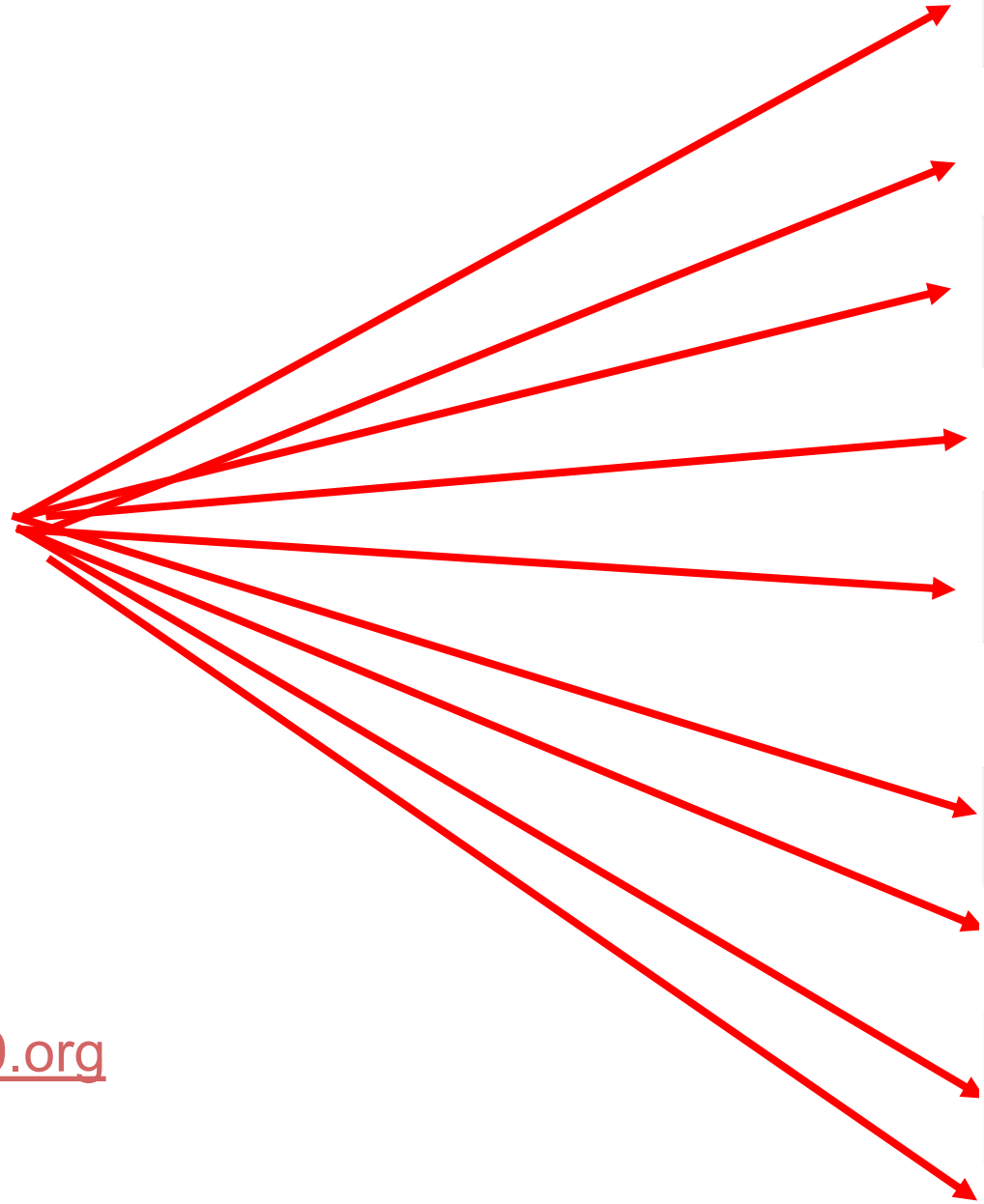
- Increase in parallelism
- Today almost a similar amount of efforts on using CPUs *vs* GPUs by real applications
- GPUs well-suited to deep learning.



NVIDIA Software uses CUDA

Why do we need “GPU accelerators” on HPC?

GPU-accelerated systems



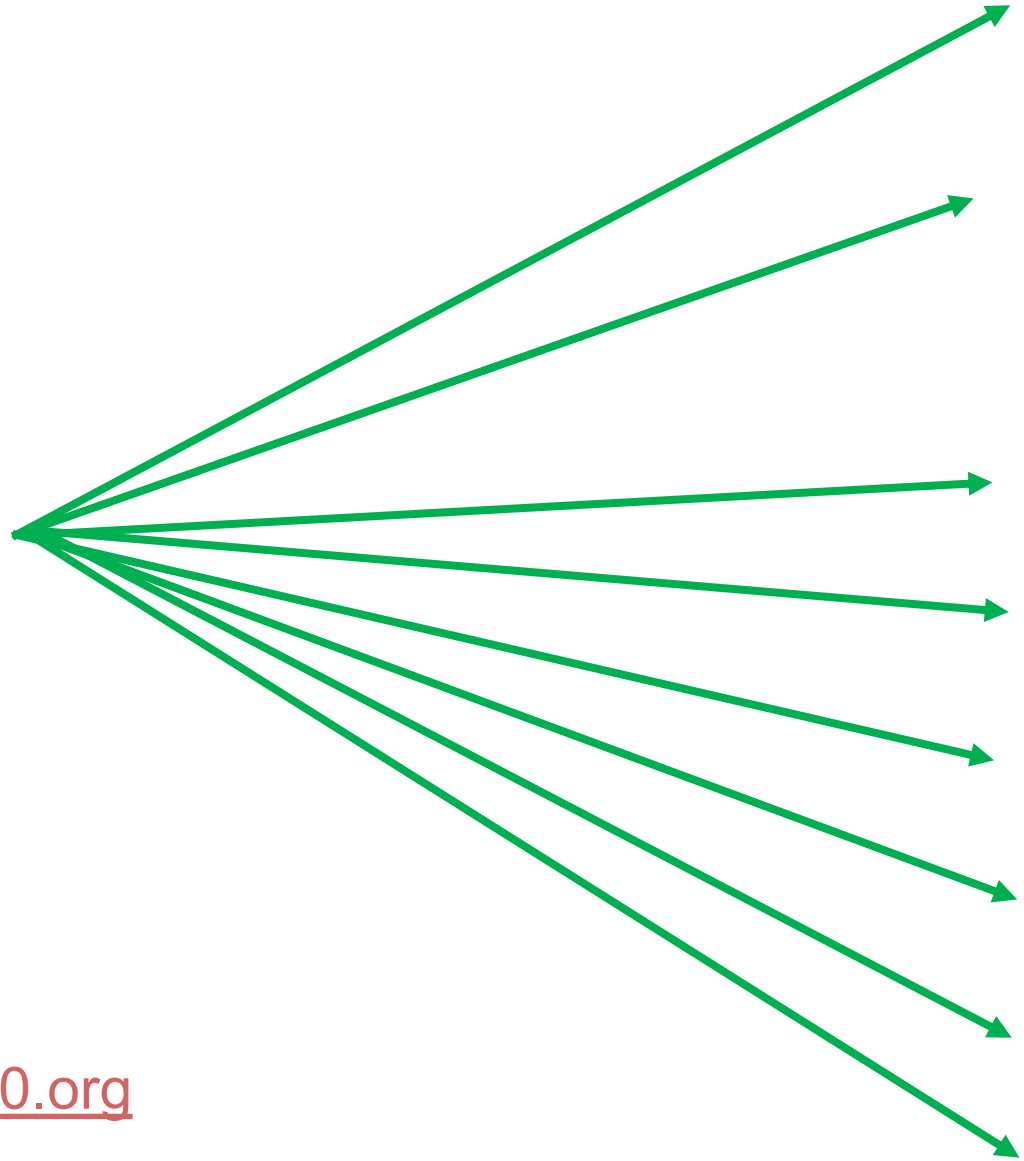
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,039,616	1,742.00	2,746.38	29,581
2	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
4	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
5	**HPC6** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, RHEL 8.9, HPE Eni S.p.A. Italy	3,143,520	477.90	606.97	8,461
6	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
7	**Alps** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Cray OS, HPE Swiss National Supercomputing Centre (CSCS) Switzerland	2,121,600	434.90	574.84	7,124
8	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107
9	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, EVIDEN EuroHPC/CINECA Italy	1,824,768	241.20	306.31	7,494
10	**Tuolumne** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	1,161,216	208.10	288.88	3,387

TOP500 The List.

www.top500.org

Why do we need “GPU accelerators” on HPC?

NVIDIA GPU accelerated Systems

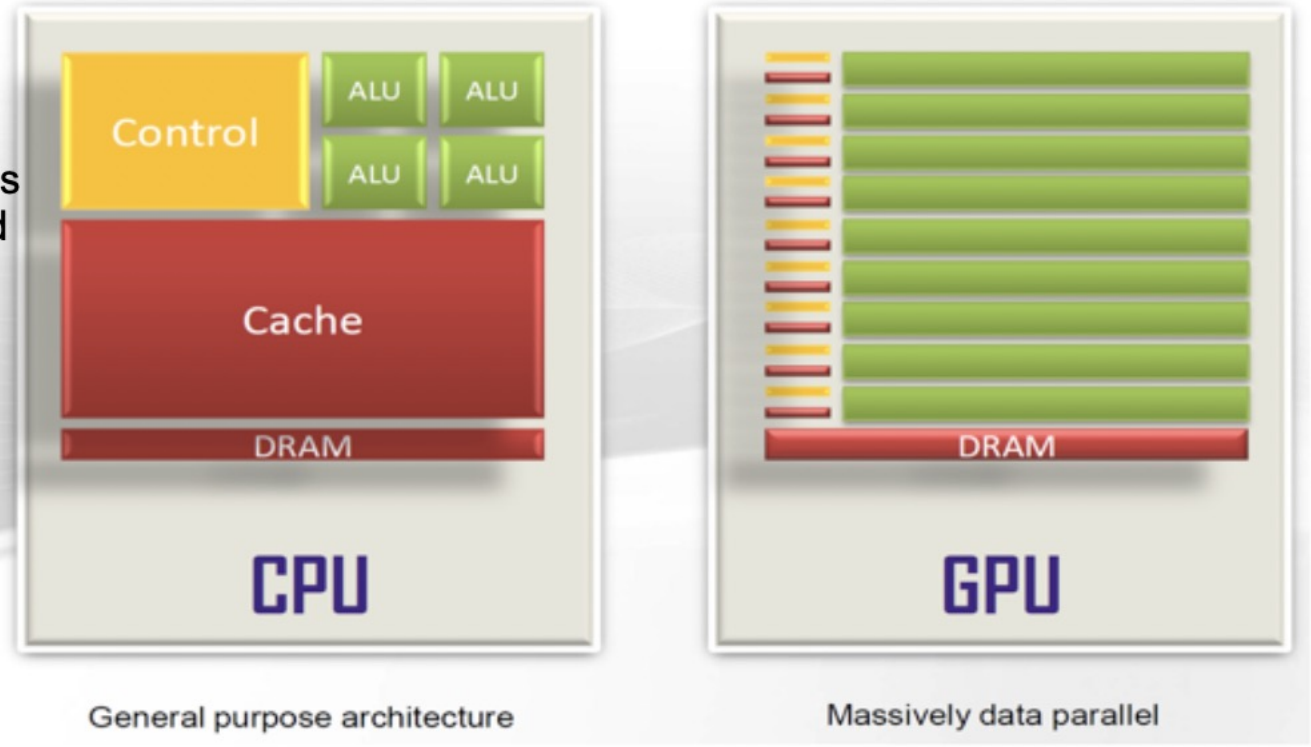


Rank	TOP500 Rank	System	Cores	Rmax (PFlop/s)	Power (kW)	Efficiency (GFlops/watts
1	222	**JEDI** - BullSequana XH3000, Grace Hopper Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, ParTec/EVIDEN EuroHPC/FZJ Germany	19,584	4.50	67	72.733
2	122	**ROMEO-2025** - BullSequana XH3000, Grace Hopper Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, Red Hat Enterprise Linux, EVIDEN ROMEO HPC Center - Champagne-Ardenne France	47,328	9.86	160	70.912
3	440	**Adastra 2** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, RHEL, HPE Grand Equipement National de Calcul Intensif - Centre Informatique National de l'Enseignement Suprieur (GENCI-CINES) France	16,128	2.53	37	69.098
4	155	**Isambard-AI phase 1** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE University of Bristol United Kingdom	34,272	7.42	117	68.835
5	51	**Capella** - Lenovo ThinkSystem SD665-N V3, AMD EPYC 9334 32C 2.7GHz, Nvidia H100 SXM5 94Gb, Infiniband NDR200, AlmaLinux 9.4, MEGWARE TU Dresden, ZIH Germany	85,248	24.06	445	68.053
6	18	**JETI - JUPITER Exascale Transition Instrument** - BullSequana XH3000, Grace Hopper Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Linux and Modular Operating System, ParTec/EVIDEN EuroHPC/FZJ Germany	391,680	83.14	1,311	67.963
7	69	**Helios GPU** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Cyfronet Poland	89,760	19.14	317	66.948
8	369	**Henri** - ThinkSystem SR670 V2, Intel Xeon Platinum 8362 32C 2.8GHz, NVIDIA H100 80GB PCIe, Infiniband HDR, Lenovo Flatiron Institute United States	8,288	2.88	44	65.396
9	338	**HoreKa-Teal** - ThinkSystem SD665-N V3, AMD EPYC 9354 32C 3.25GHz, Nvidia H100 94Gb SXM5, Infiniband NDR200, Lenovo Karlsruher Institut für Technologie (KIT) Germany	13,616	3.12	50	62.964
10	49	**rzAdams** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	129,024	24.38	388	62.803

www.top500.org

GPU vs CPU Architectural Differences

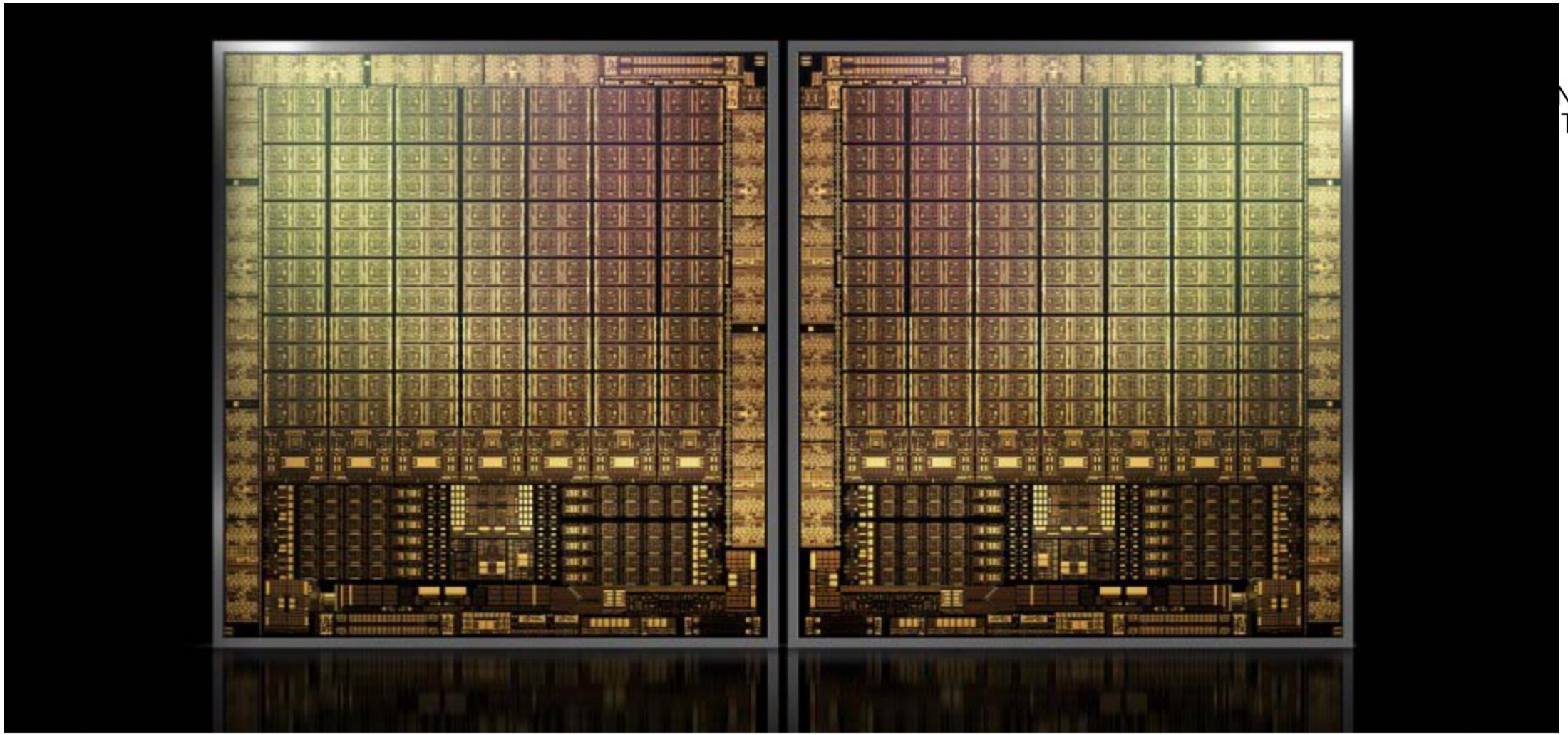
* Small number of large cores
* More control structures and less processing units
*Optimised for latency which requires quite a lot of power



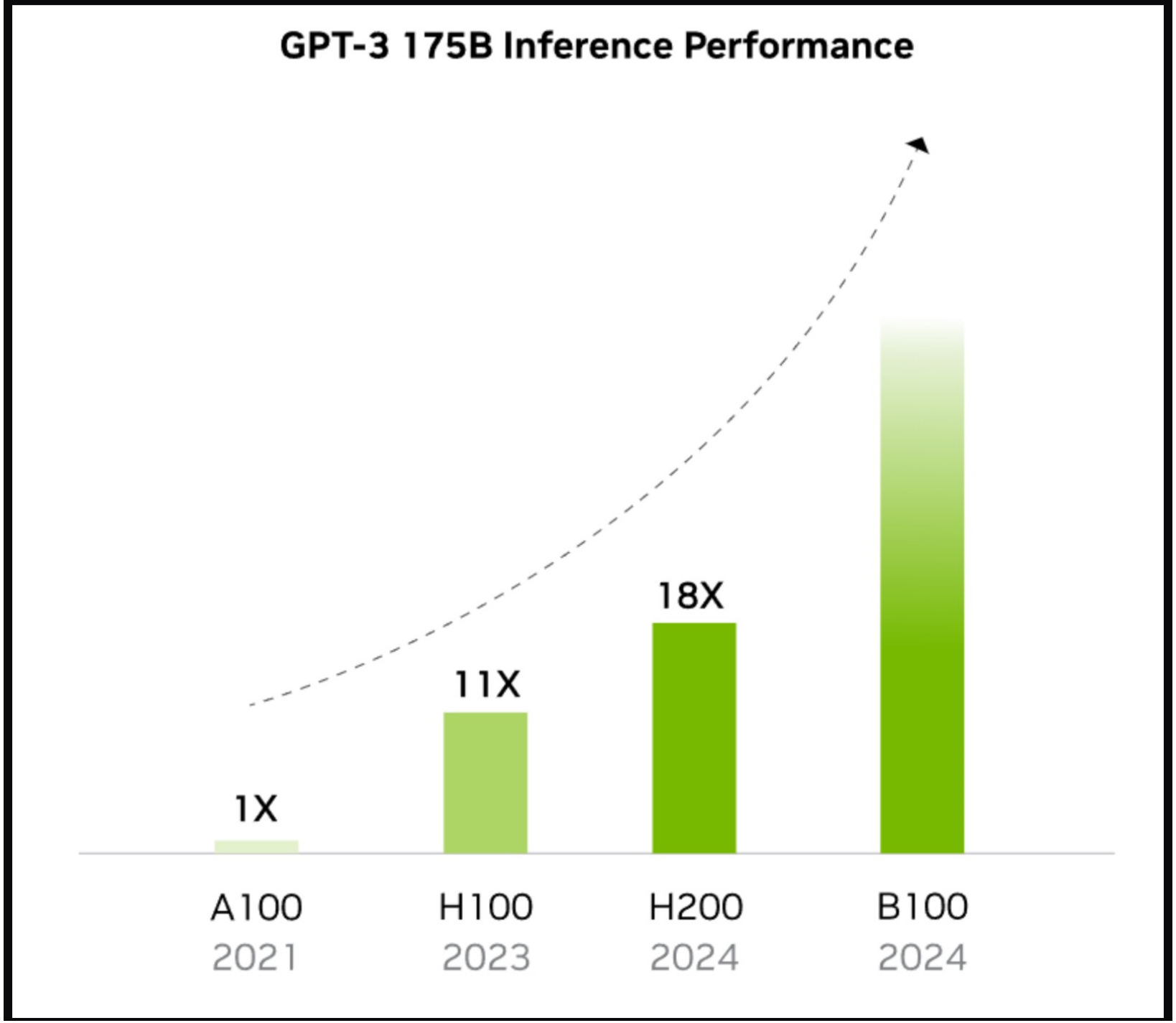
* Large number of small cores
* Less control structured and more processing units
*Less flexible program model
*There're more restrictions but Requires a lot less power

•GPU devotes more transistors data processing rather than data caching and flow control. Same problem executed on many data elements in parallel.


PCI Express 4.0 Host Interface
GigaThread Engine with MIG Control
GPC
TPC
SM
L2 Cache
L2 Cache
Memory Controller
HBM2
High-Speed Hub
NVLink

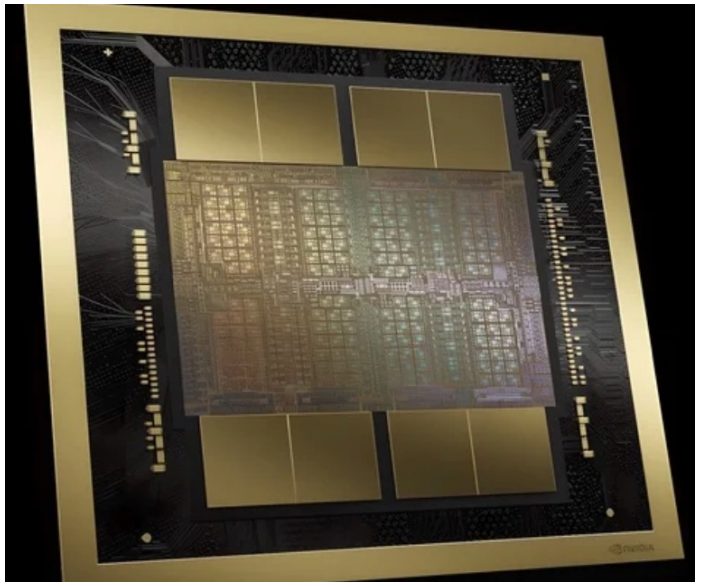


- Hopper GPU (H100) with over 80 Billion Transistors on an 814 mm^2
- 89 GB memory
- First support PCIe gen5 and utilize the HBM3 enabling 3TB/s
- 30 Tflops of peak FP64, 60Tflops with FP64 tensor-core or 32 FP performance



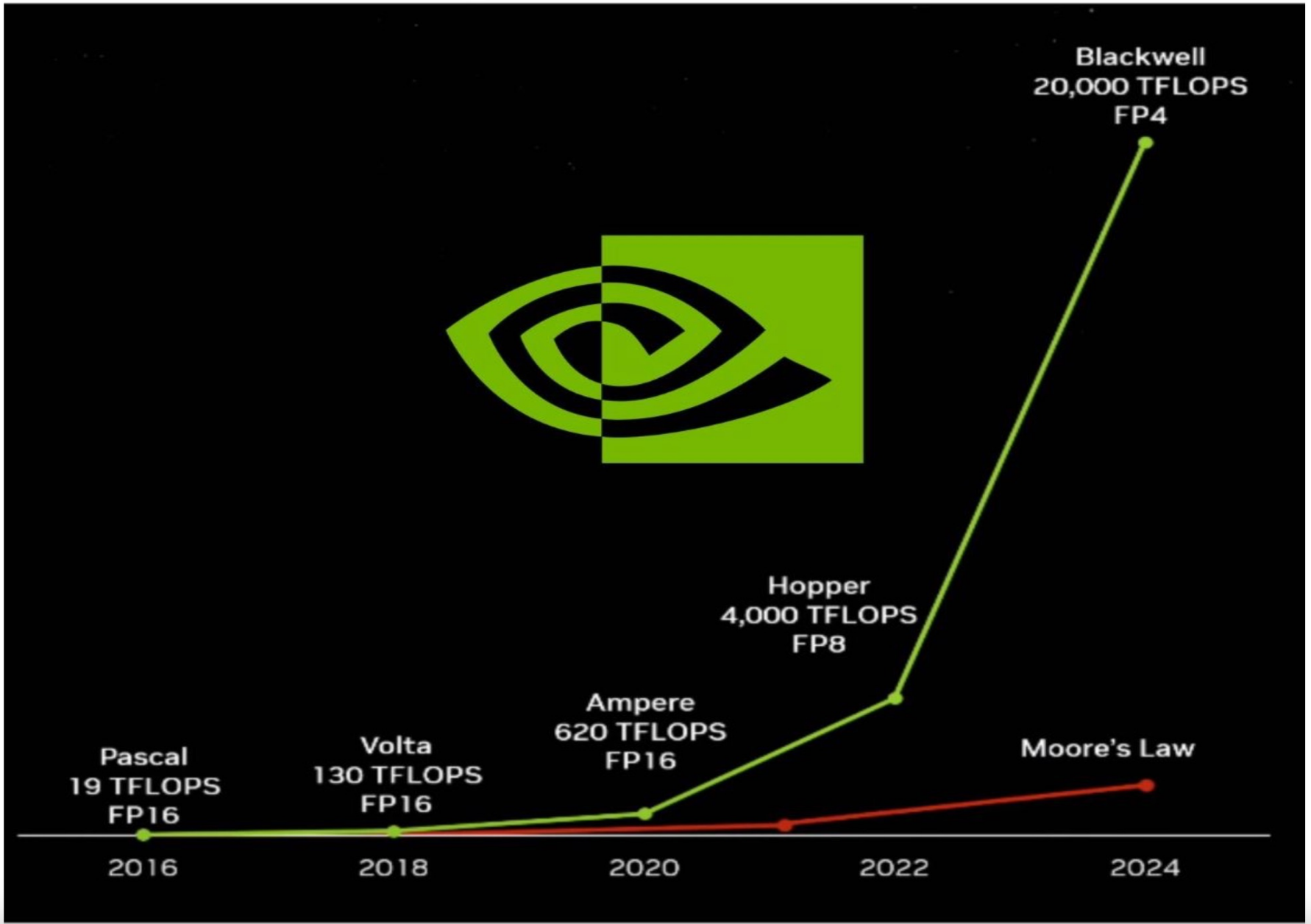
NVIDIA DEEP LEARNING INSTITUTE

lrz



Source NVIDIA

NVIDIA, with no solid competition, is out here competing against **Moore's Law** instead.



NVIDIA DEEP LEARNING INSTITUTE

lrz

Source NVIDIA

What and Why CUDA C/C++ ?

CUDA = ”Compute Unified Device Architecture”

* Introduced and released in 2006 for the GeForce 8800 *

- GPU = massively data parallel - co-processor

C/C++ plus a few simple extensions

- Compute oriented drivers, language, and tools

Documentations:

CUDA_C_Programming_Guide.pdf
CUDA_C_Getting_Started.pdf
CUDA_C_Toolkit_Release.pdf

CUDA Programming model

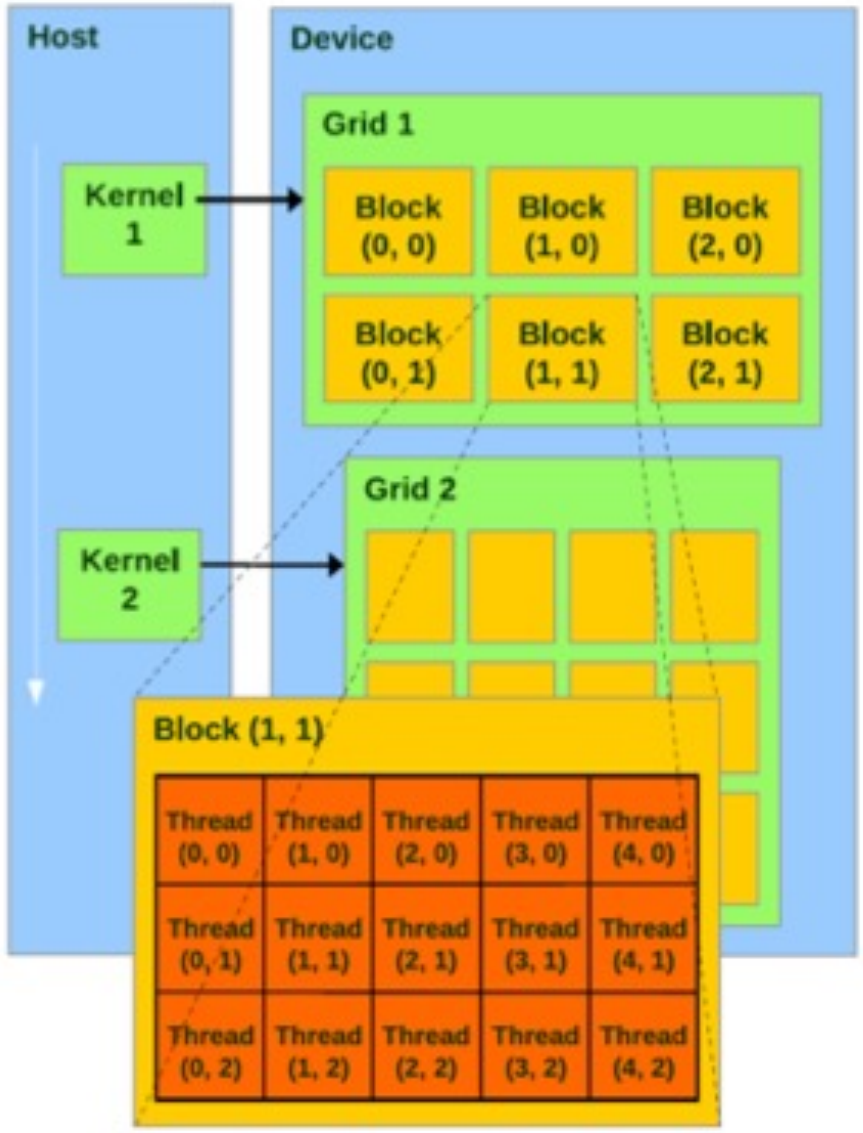
Hierarchical Threading: Grid → Blocks → Threads.

Memory Hierarchy: Global, Shared, and Local memory.

Synchronization: Managing data dependencies and race conditions.

CUDA Programming Model

- A kernel is executed as a grid of thread blocks
- All threads share data memory space
- A thread block is a batch of threads that can cooperate with each other by:
  - Synchronizing their execution
  - Efficiently sharing data through a low latency shared memory
- Tow threads from two different blocks cannot cooperate
- Sequential code launches asynchronously GPU kernels



Why use CUDA C/C++ ?

- **Massive Parallelism**: CUDA allows exploitation of thousands of GPU cores for parallel processing.
- **Performance**: GPUs can significantly speed up computations, especially for data-heavy applications.
- **Industry Standard**: Widely used in high performance scientific computing, AI, big data and graphics.

Terminology:

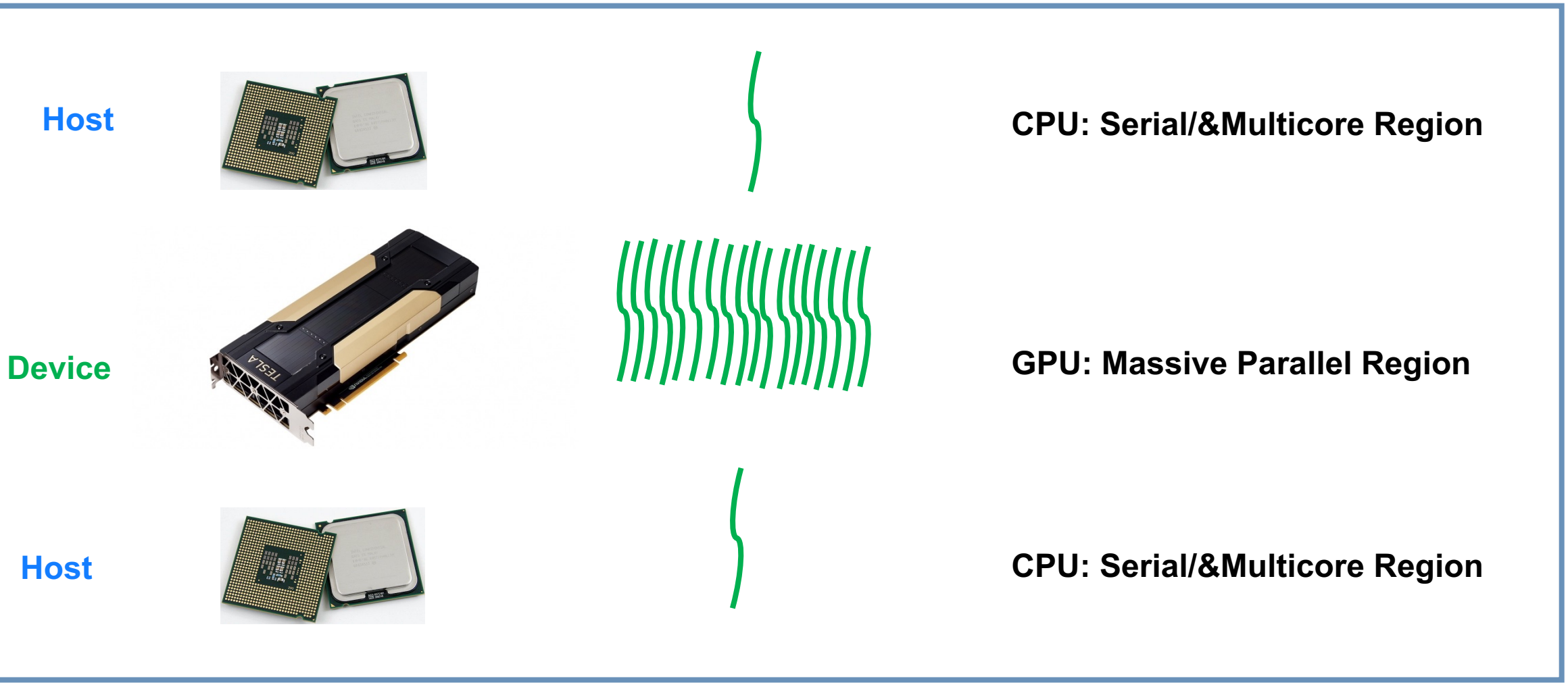
Host: The CPU and ist memory (host memory)

Host

Device: The GPU and ist memory (device memory)

Device

CUDA Devices and Threads Execution Model



How CUDA C/C++ works

The CPU allocates memory on the GPU
The CPU copies data from CPU to GPU
The CPU launches kernels on the GPU
The CPU copies data to CPU from GPU

NVCC Compiler

- NVIDIA provides a CUDA-C compiler

→ nvcc

- NVCC splits your code in 2: **Host** code and **Device** code.

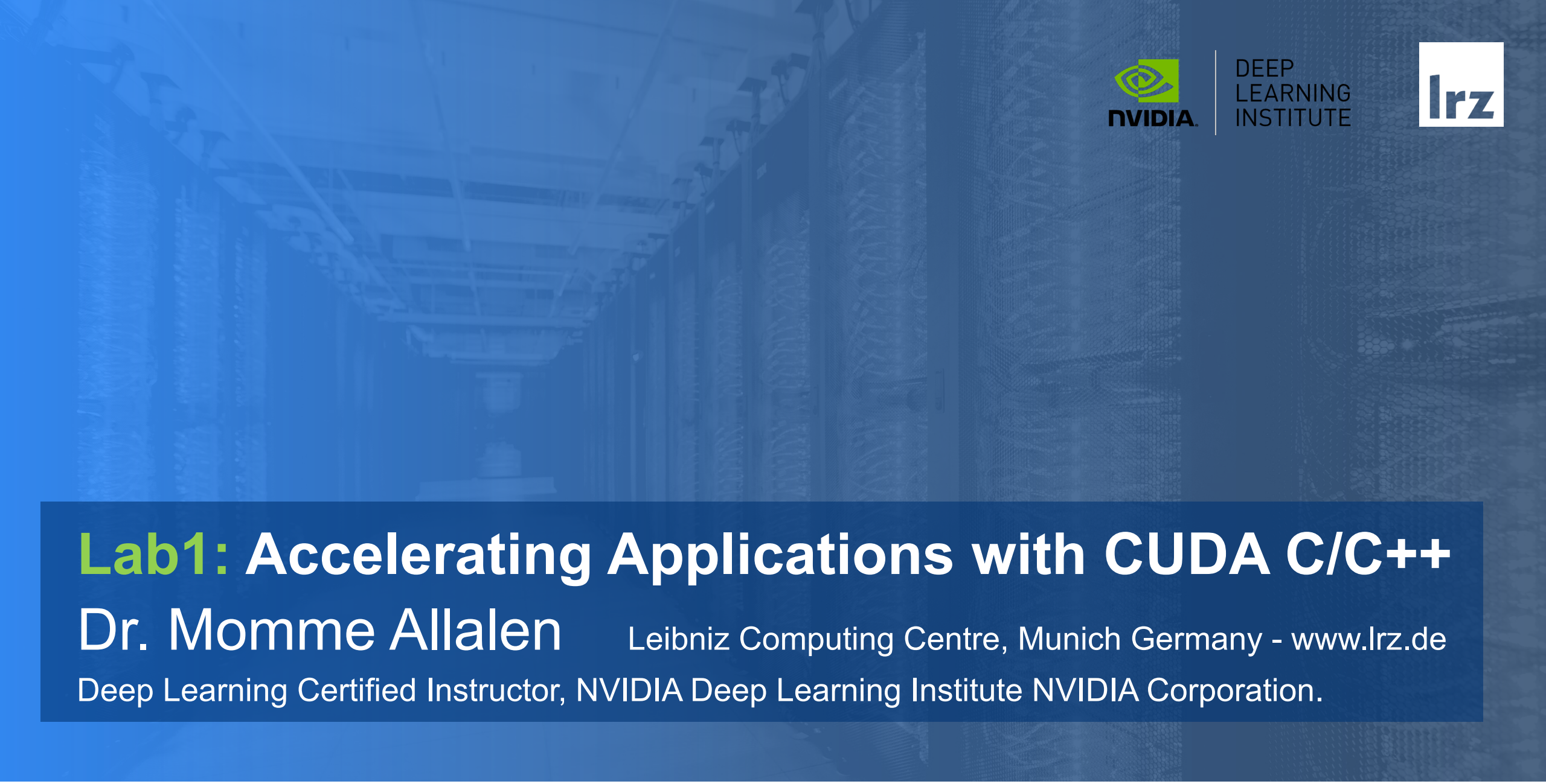
- **Device** code sent to NVIDIA device compiler.

- nvcc is capable of linking together both host and device code into a single executable.

- Convention: C++ source files containing CUDA syntax are typically given the extension **.cu**.

- For „.**cpp**“ extension use:

nvcc –x cu –arch=sm_70 –o exe code.**cpp**



Lab1: Accelerating Applications with CUDA C/C++

Dr. Momme Allalen Leibniz Computing Centre, Munich Germany - www.lrz.de

Deep Learning Certified Instructor, NVIDIA Deep Learning Institute NVIDIA Corporation.

Lab1: Accelerating Applications with CUDA C/C++

NVIDIA DEEP LEARNING INSTITUTE lrz

Prerequisites

You should already be able to:

- Declare variables, write loops, and use if / else statements in C.
- Define and invoke functions in C.
- Allocate arrays in C.
- No previous CUDA knowledge is required.

Objectives

By the time you complete this lab, you will be able to:

- Write, compile, and run C/C++ programs that both call **CPU functions** and **launch GPU kernels**.
- Control parallel **threadhierarchy** using **execution configuration**.
- Refactor serial loops to execute their iterations in parallel on a **GPU**.
- Allocate and free memory available to both **CPUs** and **GPUs**.
- Handle errors generated by CUDA code.
- Accelerate **CPU-only applications**.

nvc; nvc++ Compiler

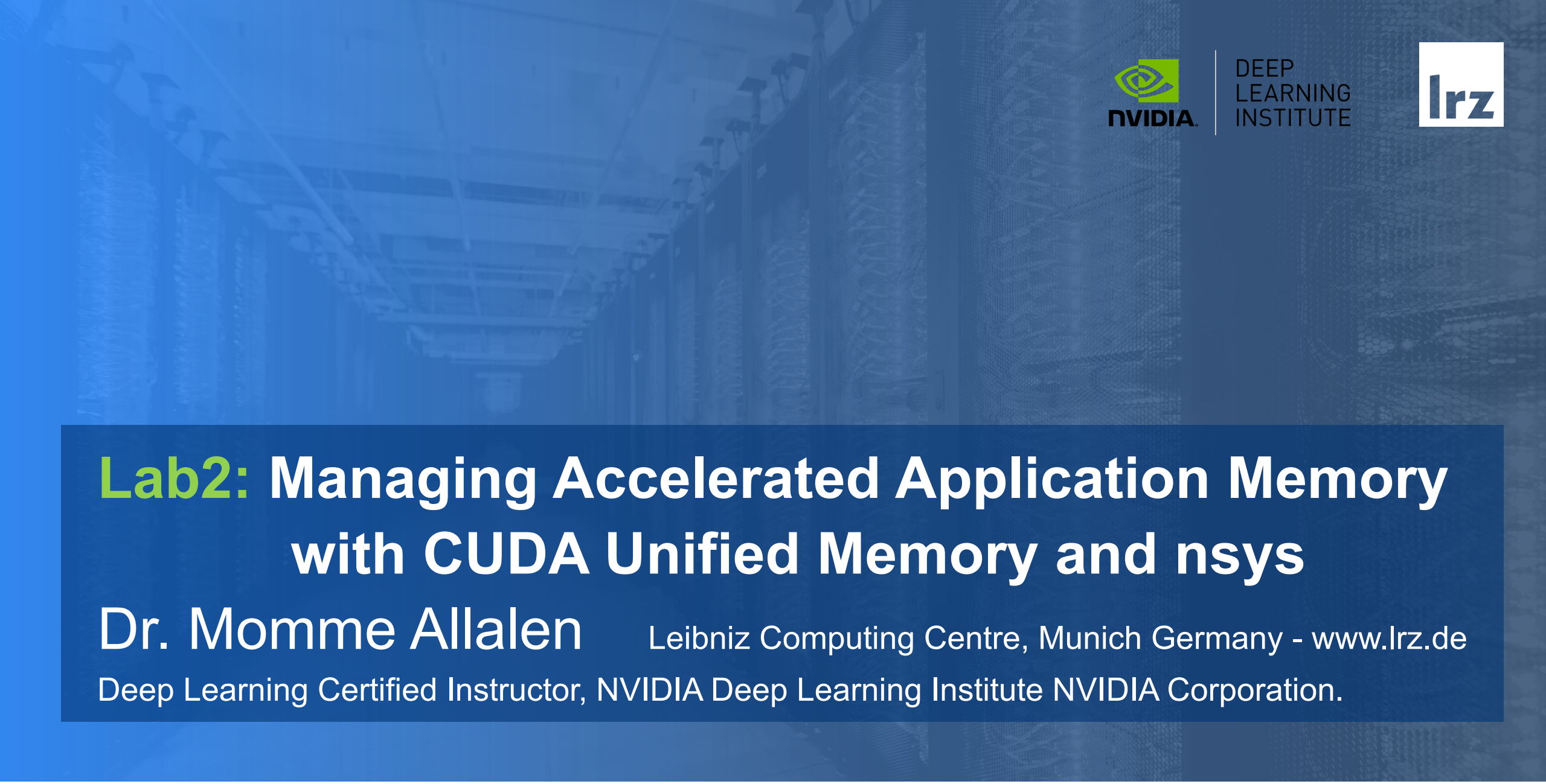
nvc :is a C11 compiler for NVIDIA GPUs and AMD, Intel, OpenPOWER, and Arm CPUs. It invokes the C compiler, assembler, and linker for the target processors with options derived from its command line arguments. nvc supports ISO C11, supports GPU programming with OpenACC, and supports multicore CPU programming with OpenACC and OpenMP.

nvc++ : is a C++17 compiler for NVIDIA GPUs and AMD, Intel, OpenPOWER, and Arm CPUs. It invokes the C++ compiler, assembler, and linker for the target processors with options derived from its command line arguments. nvc++ supports ISO C++17, supports GPU and multicore CPU programming with C++17 parallel algorithms, OpenACC, and OpenMP.

nvfortran, nvcc Compiler

nvfortran : is a Fortran compiler for NVIDIA GPUs and AMD, Intel, OpenPOWER, and Arm CPUs. It invokes the Fortran compiler, assembler, and linker for the target processors with options derived from its command line arguments. nvfortran supports ISO Fortran 2003 and many features of ISO Fortran 2008, supports GPU programming with CUDA Fortran, and GPU and multicore CPU programming with ISO Fortran parallel language features, OpenACC, and OpenMP.

nvcc : is the CUDA C and CUDA C++ compiler driver for NVIDIA GPUs. nvcc accepts a range of conventional compiler options, such as for defining macros and include/library paths, and for steering the compilation process. nvcc produces optimized code for NVIDIA GPUs and drives a supported host compiler for AMD, Intel, OpenPOWER, and Arm CPUs.



Lab2: Managing Accelerated Application Memory with CUDA Unified Memory and nsys

Dr. Momme Allalen Leibniz Computing Centre, Munich Germany - www.lrz.de

Deep Learning Certified Instructor, NVIDIA Deep Learning Institute NVIDIA Corporation.

Lab2: Managing Accelerated Application Memory with CUDA Unified Memory and nsys

Prerequisites

You should already be able to:

- Write, compile, and run C/C++ programs that both call CPU functions and **launch** GPU **kernels**.
- Control parallel **thread hierarchy** using **execution configuration**.
- Refactor serial loops to execute their iterations in parallel on a GPU.
- Allocate and free Unified Memory.

Objectives

By the time you complete this lab, you will be able to:

- Use the Nsight Systems command-line tool (**nsys**) to profile accelerated application performance.
- Laverage and understanding of **Streaming Multiprocessors** to optimize execution configurations.
- Understand the behavior of **Unified Memory** with regard to page faulting and data migrations.
- Use **asynchronous memory prefetching** to reduce page faults and data migrations for improving performance.
- Employ an iterative development cycle to rapidly accelerate and deploy applications.

CUDA® PROFILING TOOLS

nvvc: NVIDIA visual profiler
nvprof: tool to understand and optimize the performance of your CUDA, OpenACC or OpenMP applications,
Application level opportunities
- Overall application performance
  - Overlap CPU and GPU work, identify the bottlenecks (CPU or GPU)
- Overall GPU utilization and efficiency
  - Overlap compute and memory copies
  - Utilize compute and copy engines effectively.

Kernel level opportunities
- Use memory bandwidth efficiently
- Use compute resources efficiently
- Hide instruction and memory latency

There are more features, example for Dependency Analysis
Command: **nvprof** --dependency-analysis --cpu-thread-tracing on ./ executable_cuda

Nsight Systems
Nsight Compute

THE NSIGHT SUITE COMPONENTS

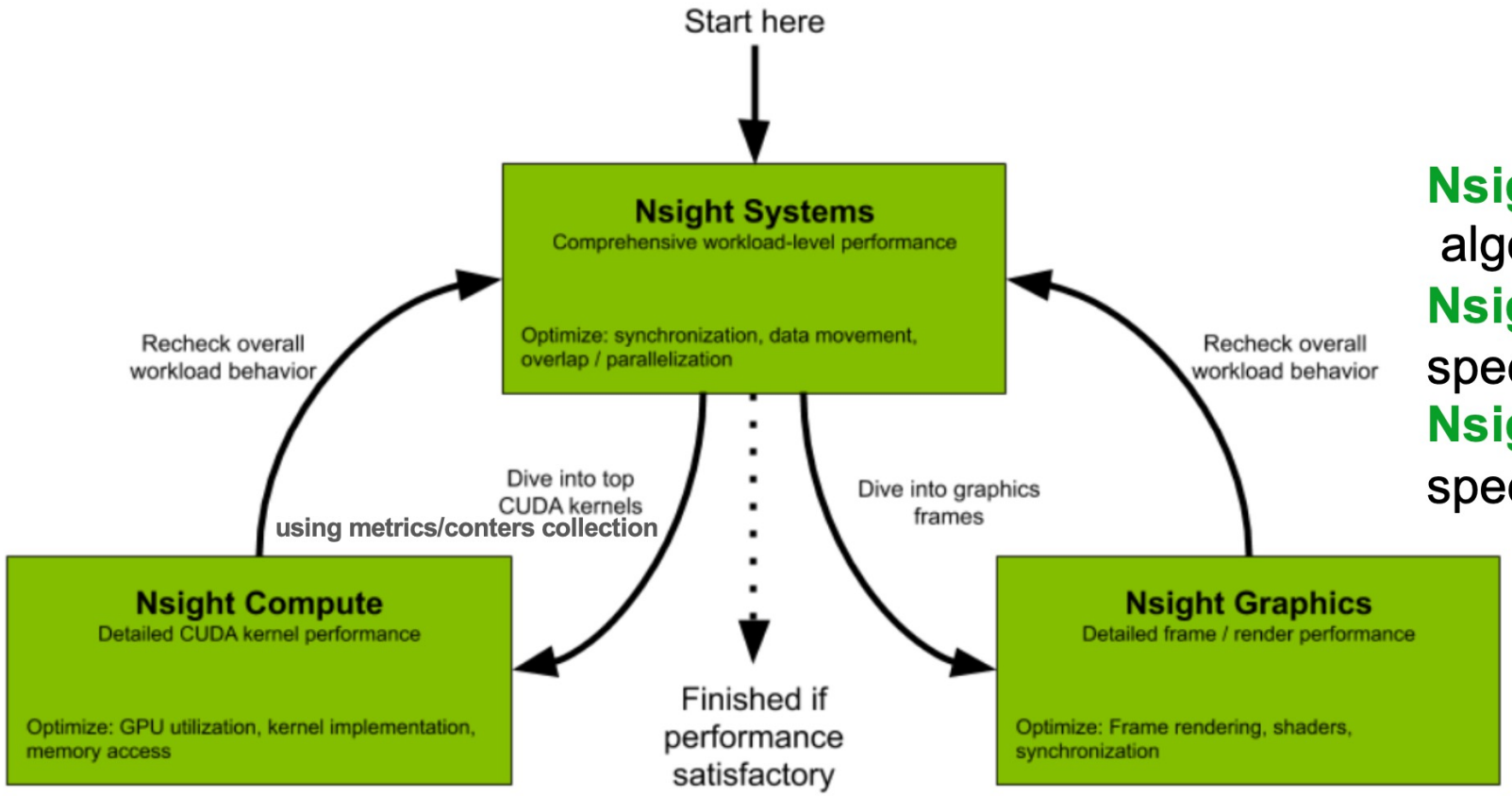


Figure 1. Flowchart describing working with new NVIDIA Nsight tools for performance optimization

Nsight Systems – Analyze application algorithm system wide

Nsight Compute – Debug/&optomize specific CUDA kernels

Nsight Graphics – Debug/&optimize specific graphics workloads

nvprof replaced with **nsys –profile....**

https://developer.nvidia.com/nsight-systems

NVIDIA NSIGHT SYSTEMS

- Support: **MPI**, **OpenACC, OpenMP**

- Complex data mining capabilities enable going beyond basic statistics.
- Support multiple simultaneous sessions.
- **MPI trace** feature enables analysing when the threads are busy or blocked in long-running **MPI** standard functions. This is available on **OpenMPI**, **MPICH** and **NVShmem**.
- **OpenACC** trace shows where code has been offload and parallelized onto the GPU, helping to analyse activities executed on both CPUs and GPUs in parallel.

- **OpenMP tracing** is available for compilers supporting **OpenMP5** and the **OMPT interface**. This feature traces parallel regions distributed across multiple threads or offloaded to the GPU.
- Supports **CUDA** graphs. It provides insights into the execution of **CUDA** kernels and **CUDA** graphs. Kernels can be traced back through graph launch, instantiation, and code creation to identify the origin of the GPU kernel execution.

https://developer.nvidia.com/nsight-systems

Command Line Options nsys

Command	Description
profile	A fully formed profiling description requiring and accepting no further input. The command switch options used (see below table) determine when the collection starts, stops, what collectors are used (e.g. API trace, IP sampling, etc.), what processes are monitored, etc.
start	Start a collection in interactive mode. The start command can be executed before or after a launch command.
stop	Stop a collection that was started in interactive mode. When executed, all active collections stop, the CLI process terminates but the application continues running.
cancel	Cancels an existing collection started in interactive mode. All data already collected in the current collection is discarded.
launch	In interactive mode, launches an application in an environment that supports the requested options. The launch command can be executed before or after a start command.
shutdown	Disconnects the CLI process from the launched application and forces the CLI process to exit. If a collection is pending or active, it is cancelled
export	Generates an export file from an existing .nsys-rep file. For more information about the exported formats see the /documentation/nsys-exporter directory in your Nsight Systems installation directory.
stats	Post process existing Nsight Systems result, either in .nsys-rep or SQLite format, to generate statistical information.
analyze	Post process existing Nsight Systems result, either in .nsys-rep or SQLite format, to generate expert systems report.
status	Reports on the status of a CLI-based collection or the suitability of the profiling environment.
sessions	Gives information about all sessions running on the system.

https://docs.nvidia.com/nsight-systems/UserGuide/index.html

NVIDIA® Tools Extension SDK (NVTX)

- C-based Application Programming Interface (API) for annotating events, code ranges, and resources in your applications
- Codes which integrate NVTX can use NVIDIA Nsight, Tegra System Profiler, and Visual Profiler to capture and visualize these events and ranges.

```
[allalen1@jwlogin22 v2]$ ncu -h | grep nvtx
  --nvtx                                Enable NVTX support.
  --nvtx-include arg                    Adds include statement to the NVTX filter, which allows selecting kernels to
  --nvtx-exclude arg                    Adds exclude statement to the NVTX filter, which allows selecting kernels to
  --print-nvtx-rename arg (=none)        Select how NVTX should be used for renaming:
                                           per-nvtx
Usage of --nvtx-include and --nvtx-exclude:
  ncu --nvtx --nvtx-include "Domain A@Range A"
  ncu --nvtx --nvtx-exclude "Range A]"
  ncu --nvtx --nvtx-include "Range A" --nvtx-exclude "Range B"
```

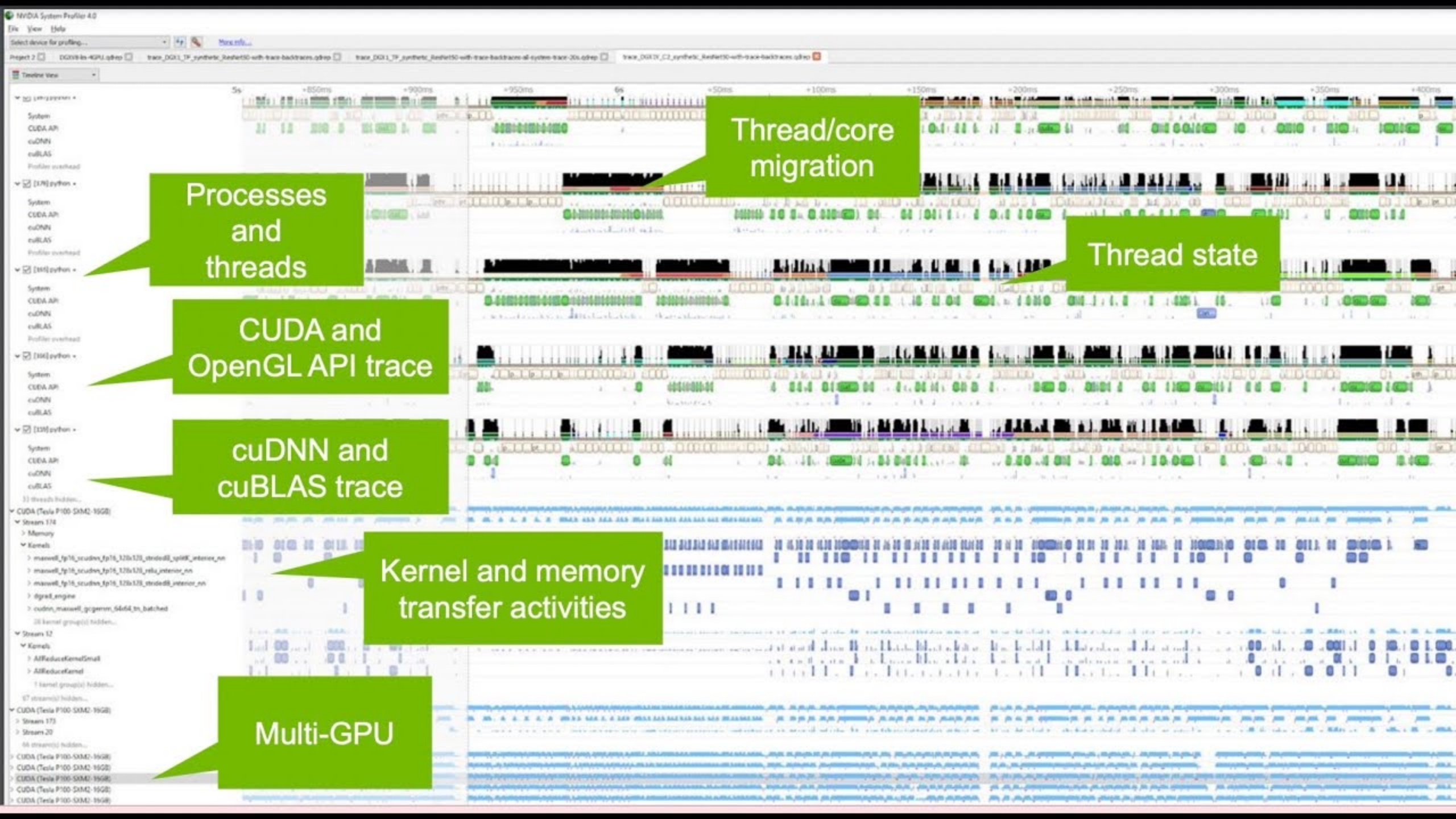
https://docs.nvidia.com/nsight-visual-studio-edition/nvtx/index.html

NVIDIA® Tools Extension SDK (NVTX)

```c
#include <nvToolsExt.h>
#include <sys/syscall.h>
#include <unistd.h>

static void wait(int seconds) {
    nvtxRangePush(__FUNCTION__);
    nvtxMark("Waiting...");
    sleep(seconds);
    nvtxRangePop();
}

int main(void) {
    nvtxNameOsThread(syscall(SYS_gettid), "Main Thread");
    nvtxRangePush(__FUNCTION__);
    wait(1);
    nvtxRangePop();
}
```

nsys profile –t nvtx --stats=true …
Or for Julia code:
nsys profile -t nvtx,cuda -o output_file.qdrep
julia --project=../../ script.jl

https://docs.nvidia.com/nsight-visual-studio-edition/2020.1/nvtx/index.html


Thread/core migration
Processes and threads
Thread state
CUDA and OpenGL API trace
cuDNN and cuBLAS trace
Kernel and memory transfer activities
Multi-GPU

NVIDIA Tools Extension API Library (NVTX)

The NVIDIA Tools Extension SDK (NVTX) is a C-based Application Programming Interface (API) for annotating events, code ranges, and resources in your applications.
Applications which integrate NVTX can use NVIDIA Nsight VSE to capture and visualize these events and ranges.

```
void Wait(int waitMilliseconds)
{
        nvtxNameOsThread("MAIN");
        nvtxRangePush(__FUNCTION__);
        nvtxMark(>"Waiting...");
        Sleep(waitMilliseconds);
        nvtxRangePop();
}
int main(void)
{
        nvtxNameOsThread("MAIN");
        nvtxRangePush(__FUNCTION__);
        Wait();
        nvtxRangePop();
}
```

nsys profile –t nvtx --stats=true …

https://docs.nvidia.com/nsight-visual-studio-edition/2020.1/nvtx/index.html

Lab3: Asynchronous Streaming, and Visual Profiling for Accelerated Applications with CUDA C/C++

Dr. Momme Allalen Leibniz Computing Centre, Munich Germany - www.lrz.de

Deep Learning Certified Instructor, NVIDIA Deep Learning Institute NVIDIA Corporation.

Lab3: Asynchronous Streaming, and Visual Profiling with CUDA C/C++

Prerequisites

To get the most out of this lab, you should already be able to:

- Write, compile, and run C/C++ programs that both call CPU functions and launch GPU kernels.
- Control the parallel thread hierarchy using execution configuration.
- Refactor serial loops to execute their iterations in parallel on a GPU.
- Allocate and free CUDA Unified Memory.
- Understand the behaviour of Unified Memory with regard to page faults and data migrations.
- Use asynchronous memory prefetching to reduce page faults and data migrations.

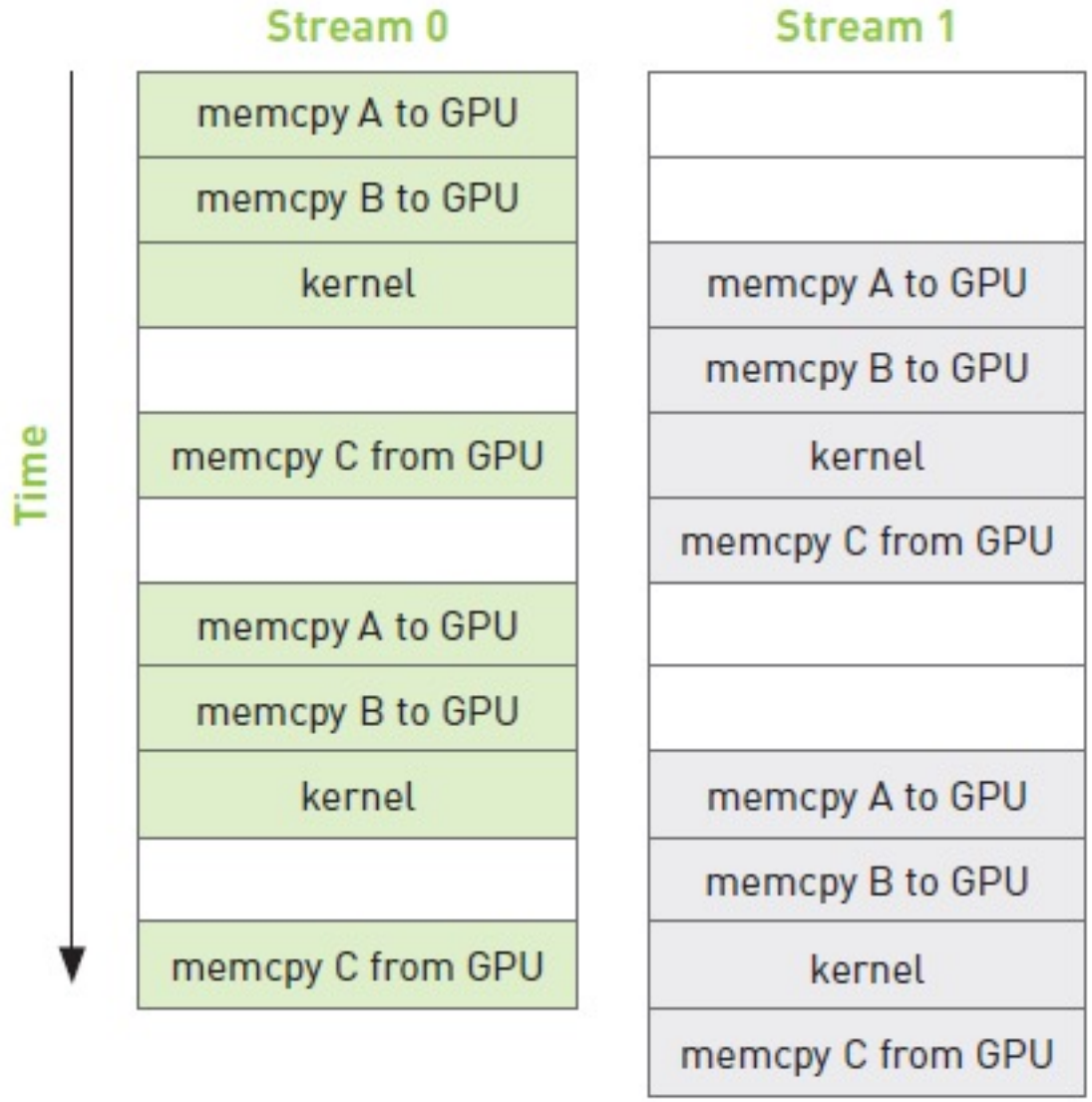
Objectives

By the time you complete this lab, you will be able to:

- Use **Nsight Systems** to visually profile the timeline of GPU-accelerated CUDA applications.
- Use **Nsight Systems** to identify, and exploit, optimization opportunities in GPU-accelerated CUDA applications.
- Utilize **CUDA streams** for concurrent kernel execution in accelerated applications.
- **(Optional Advanced Content)** Use manual memory allocation, including allocating pinned memory, in order to asynchronously transfer data in concurrent CUDA streams.

Multiple Streams

Overlap copy with kernel



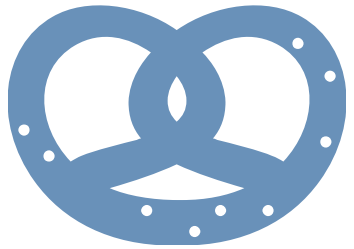
Multiple Streams

```
for (int i=0; i<FULL_SIZE; i+= N*2) {
// copy the locked memory to the device, async
cudaMemcpyAsync(dev_a0, host_a+i, N * sizeof(int),cudaMemcpyHostToDevice, stream0);
cudaMemcpyAsync(dev_b0, host_b+i, N * sizeof(int),cudaMemcpyHostToDevice, stream0);

kernel<<<N/256,256,0,stream0>>>( dev_a0, dev_b0, dev_c0 );

// copy the data from device to locked memory
cudaMemcpyAsync(host_c+i, dev_c0, N * sizeof(int),cudaMemcpyDeviceToHost, stream0);
// copy the locked memory to the device, async
cudaMemcpyAsync(dev_a1,host_a+i+N, N * sizeof(int),cudaMemcpyHostToDevice, stream1);
cudaMemcpyAsync(dev_b1,host_b+i+N, N * sizeof(int),cudaMemcpyHostToDevice, stream1);

kernel<<<N/256,256,0,stream1>>>( dev_a1, dev_b1, dev_c1 );

// copy the data from device to locked memory
cudaMemcpyAsync(host_c+i+N,dev_c1, N * sizeof(int),cudaMemcpyDeviceToHost, stream1);
}
```

Deep Learning and GPU Programming Workshop

We continue at 13:00 CEST

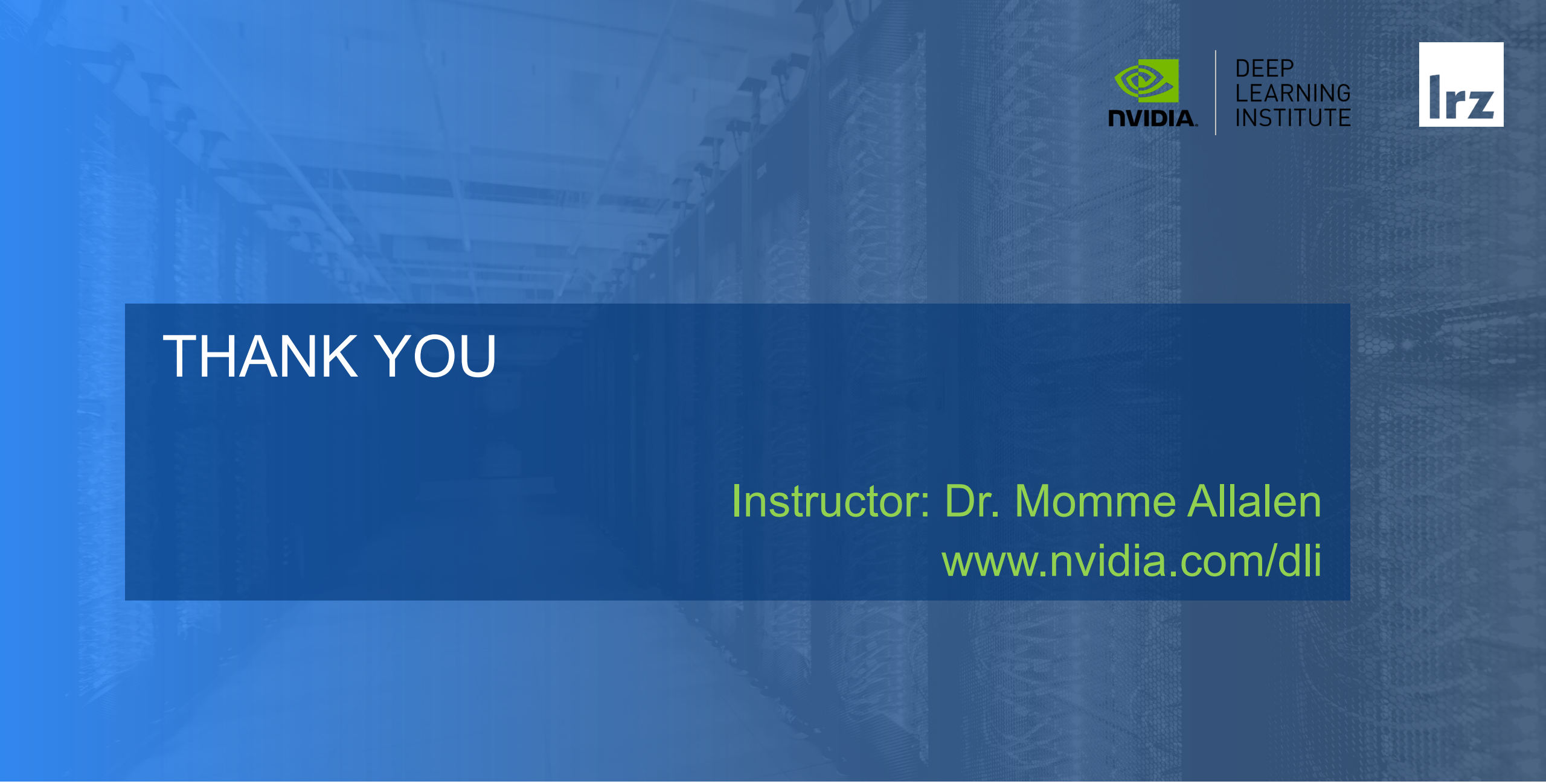
Enjoy the lunch break!



Deep Learning and GPU Programming Workshop

We continue with Asynchronous Streaming at 14:30 CEST

Enjoy your coffee break!


DEEP
LEARNING
INSTITUTE
lrz
THANK YOU
Instructor: Dr. Momme Allalen
www.nvidia.com/dli