lrz

PRACE

NVIDIA | DEEP LEARNING INSTITUTE

Fundamentals of Accelerated Computing with CUDA C/C++

Dr. Momme Allalen | CSC | 11.05.2022

CSC

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

Overview

- The workshop is co-organized by LRZ, CSC, IT4Innovations and NVIDIA Deep Learning Institute (DLI) for the Partnership for Advanced Computing in Europe (PRACE).
- NVIDIA Deep Learning Institute (DLI) offers hands-on training for developers, data scientists, and researchers looking to solve challenging problems with deep learning.
- This 4-days workshop offered for the first time online combines lectures about fundamentals of Deep Learning for Multiple Data Types and Multi-GPUs with lectures about Accelerated Computing with OpenACC and CUDA C/C++
- Learn how to train and deploy a neural network to solve real-world problems, how to generate effective descriptions of content within images and video clips, how to effectively parallelize training of deep neural networks on Multi-GPUs and how to accelerate your applications with OpenACC and CUDA C/C++.
- 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

10:00-10:15	Intro - **CUDA C/C++**
10:15-12:00	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**
14:30-15:45	Asynchronous Streaming and Visual Profiling for Accelerated Applications with **CUDA C/C++**
15:45-16:00	Q&A, Final Remarks

Workshop Webpage

- **Lecture material will be made available under:**
  - **https://tinyurl.com/dli-workshop-csc**

- **Access CUDA C/C++ Code:**
- See the: Chat Window

Training Setup

- To get started, follow these steps:
- Create an NVIDIA Developer account at http://courses.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 http://courses.nvidia.com/dli-event and enter the event code provided by the instructor.
- You're ready to get started.

And now …

Enjoy the course!

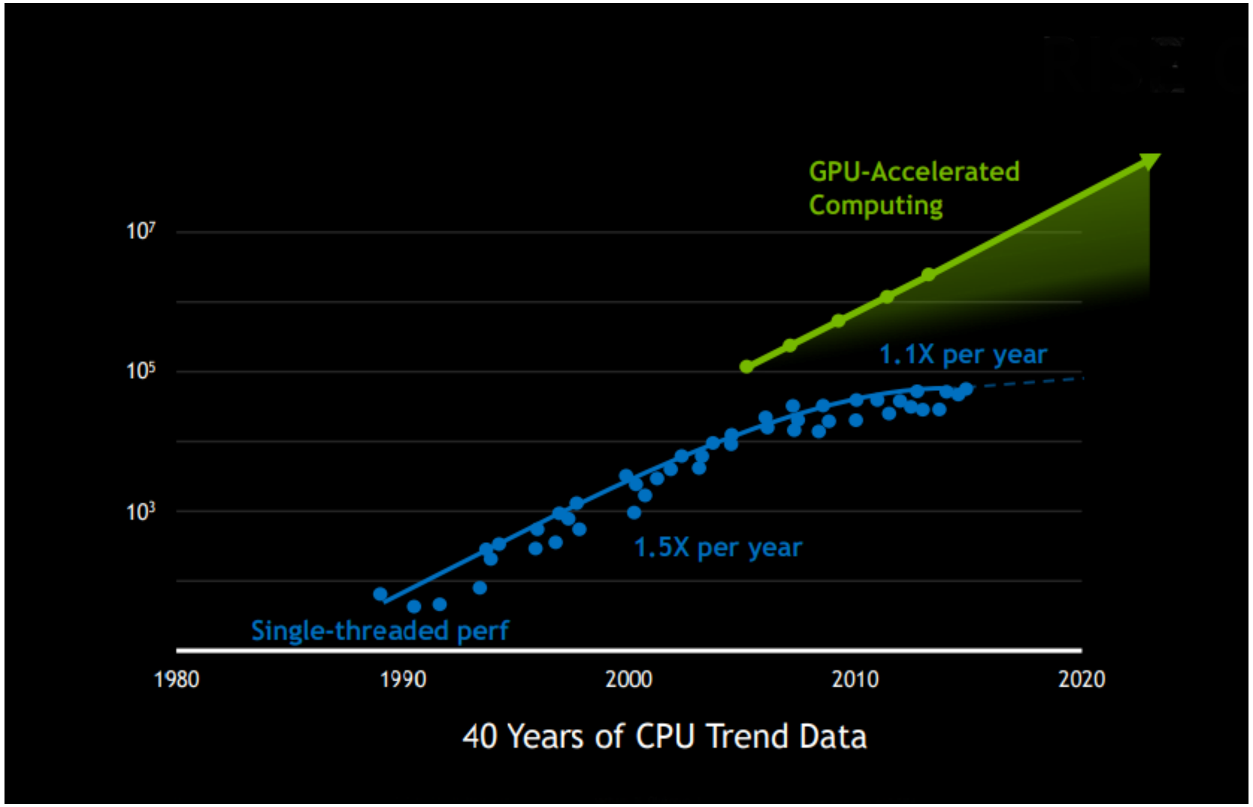
Why do we need to program for GPU?

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 !

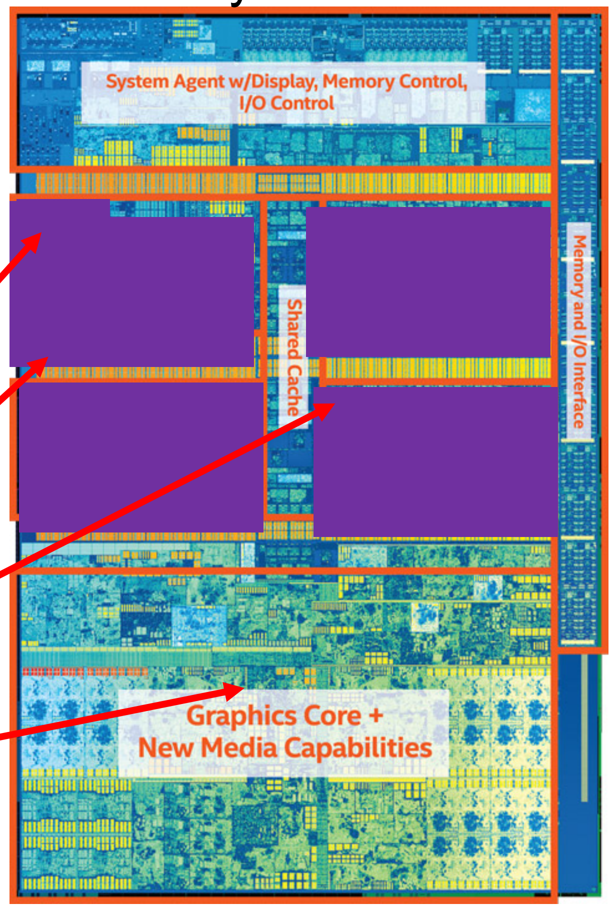


Why do we need to program for GPU?

Typical example Intel chip: **Core i7 7th Gen**

- 4*CPU cores
- with hyperthreading
- Each with 8-wide AVX instructions
- GPU with 1280 processing elements

Intel Kaby Lake-S

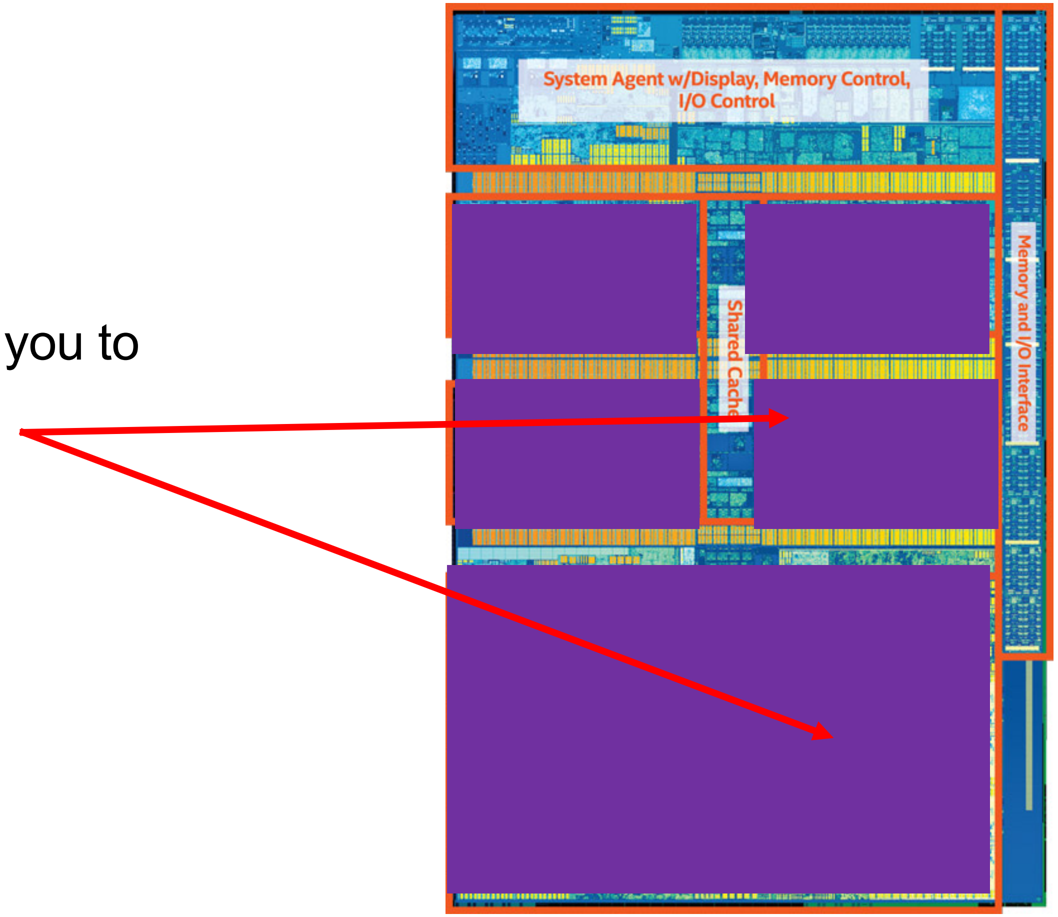


Programming on chip:
- Serial C/C++ .. Code alone only takes advantage of a very small amount of the available resources of the chip
- Using vectorisation allows you to fully utilise the resources of a single hyper-thread
- Using multi-threading allows you to fully utilise all CPU cores

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.

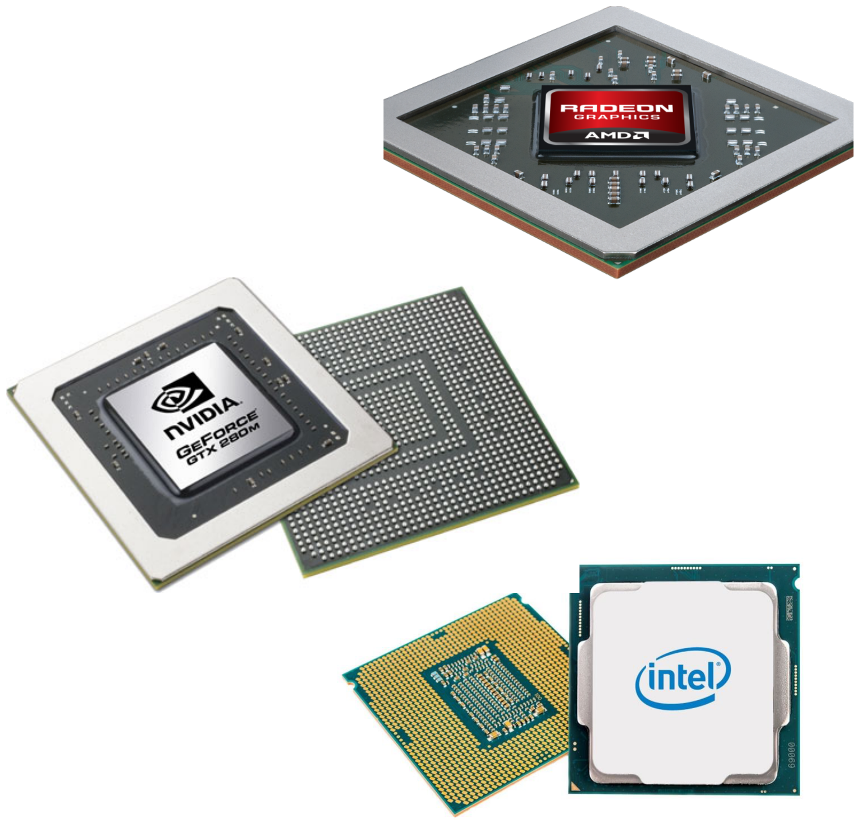


Why do we need to program for GPU?

PRACE NVIDIA DEEP LEARNING INSTITUTE lrz

GPU programming:

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



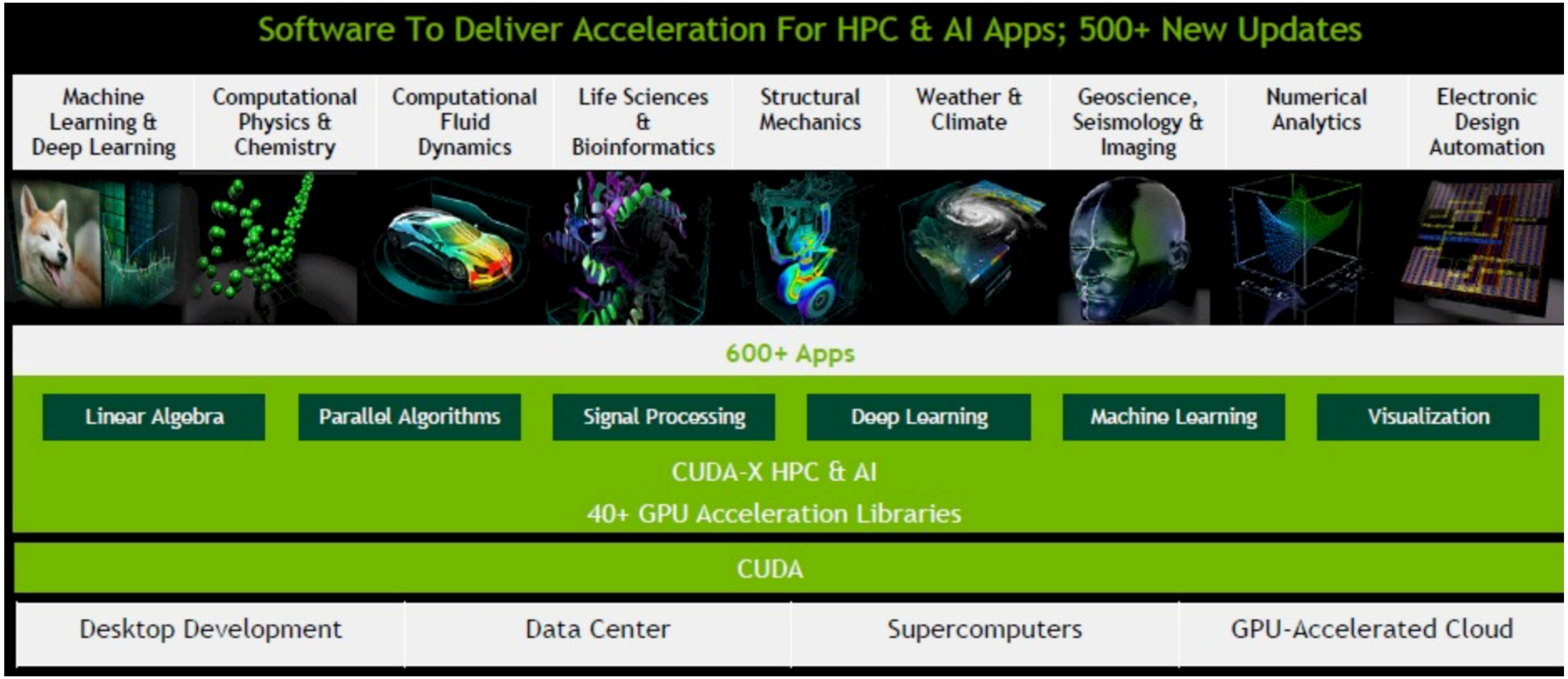
- oneAPI & DPC++
- HIP
- **CUDA C/C++**
- Kokkos
- HPX
- OpenCL
- SYCL …

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 “accelerators” on HPC?
Top500.org

Rank	System	Cores	(TFlop/s)	(TFlop/s)	(kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**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
3	**Sierra** - IBM Power System AC922, 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
4	**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
5	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70,870.0	93,750.0	2,589
6	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,460.0	79,215.0	2,646
7	**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
8	**JUWELS Booster Module** - Bull Sequana XH2000 , AMD EPYC 7402 24C 2.8GHz, NVIDIA A100, Mellanox HDR InfiniBand/ParTec ParaStation ClusterSuite, Atos Forschungszentrum Juelich (FZJ) Germany	449,280	44,120.0	70,980.0	1,764
9	**HPC5** - PowerEdge C4140, Xeon Gold 6252 24C 2.1GHz, NVIDIA Tesla V100, Mellanox HDR Infiniband, DELL EMC Eni S.p.A. Italy	669,760	35,450.0	51,720.8	2,252
10	**Voyager-EUS2** - ND96amsr_A100_v4, AMD EPYC 7V12 48C 2.45GHz, NVIDIA A100 80GB, Mellanox HDR Infiniband, Microsoft Azure	253,440	30,050.0	39,531.2	

NVIDIA GPUs

www.top500.org

Why do we need “accelerators” on HPC? Green top500

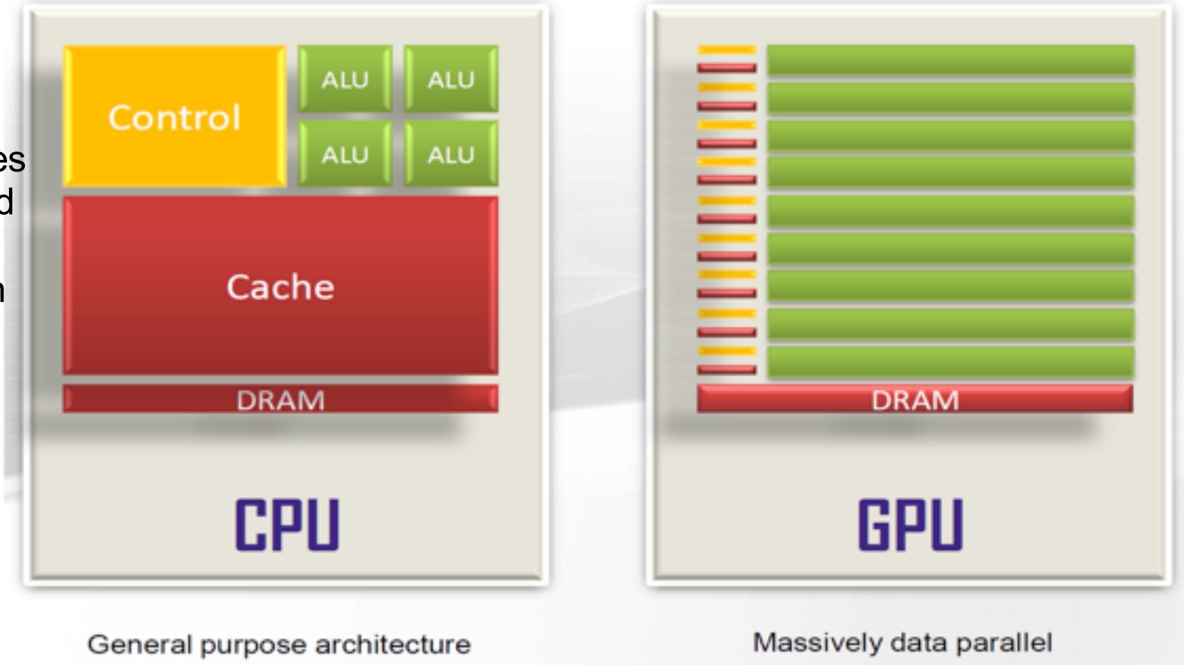
NVIDIA GPUs

THE GREEN 500™

Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	Power (kW)	Efficiency (GFlops/watts)
1	301	**MN-3** - MN-Core Server, Xeon Platinum 8260M 24C 2.4GHz, Preferred Networks MN-Core, MN-Core DirectConnect, Preferred Networks Preferred Networks Japan	1,664	2,181.2	55	39.379
2	291	**SSC-21 Scalable Module** - Apollo 6500 Gen10 plus, AMD EPYC 7543 32C 2.8GHz, NVIDIA A100 80GB, Infiniband HDR200, HPE Samsung Electronics South Korea	16,704	2,274.1	103	33.983
3	295	**Tethys** - NVIDIA DGX A100 Liquid Cooled Prototype, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100 80GB, Infiniband HDR, Nvidia NVIDIA Corporation United States	19,840	2,255.0	72	31.538
4	280	**Wilkes-3** - PowerEdge XE8545, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 80GB, Infiniband HDR200 dual rail, DELL EMC University of Cambridge United Kingdom	26,880	2,287.0	74	30.797
5	30	**HiPerGator AI** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Infiniband HDR, Nvidia University of Florida United States	138,880	17,200.0	583	29.521
[illegible]	[illegible]	[illegible] SD650-N V2, Xeon Platinum 8360Y 36C 2.4GHz, NVIDIA A100 SXM4 40 GB, Infiniband HDR, Lenovo SURF Netherlands	[illegible]	[illegible]	[illegible]	[illegible]
7	5	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70,870.0	2,589	27.374
8	71	**Karolina, GPU partition** - Apollo 6500, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Infiniband HDR200, HPE IT4Innovations National Supercomputing Center, VSB-Technical University of Ostrava Czechia	71,424	6,752.0	311	27.213
9	45	**MeluXina - Accelerator Module** - BullSequana XH2000, AMD EPYC 7452 32C 2.35GHz, NVIDIA A100 40GB, Mellanox HDR InfiniBand/ParTec ParaStation ClusterSuite, Atos LuxProvide Luxembourg	99,200	10,520.0	390	26.957
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

GPU vs CPU Architecture

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

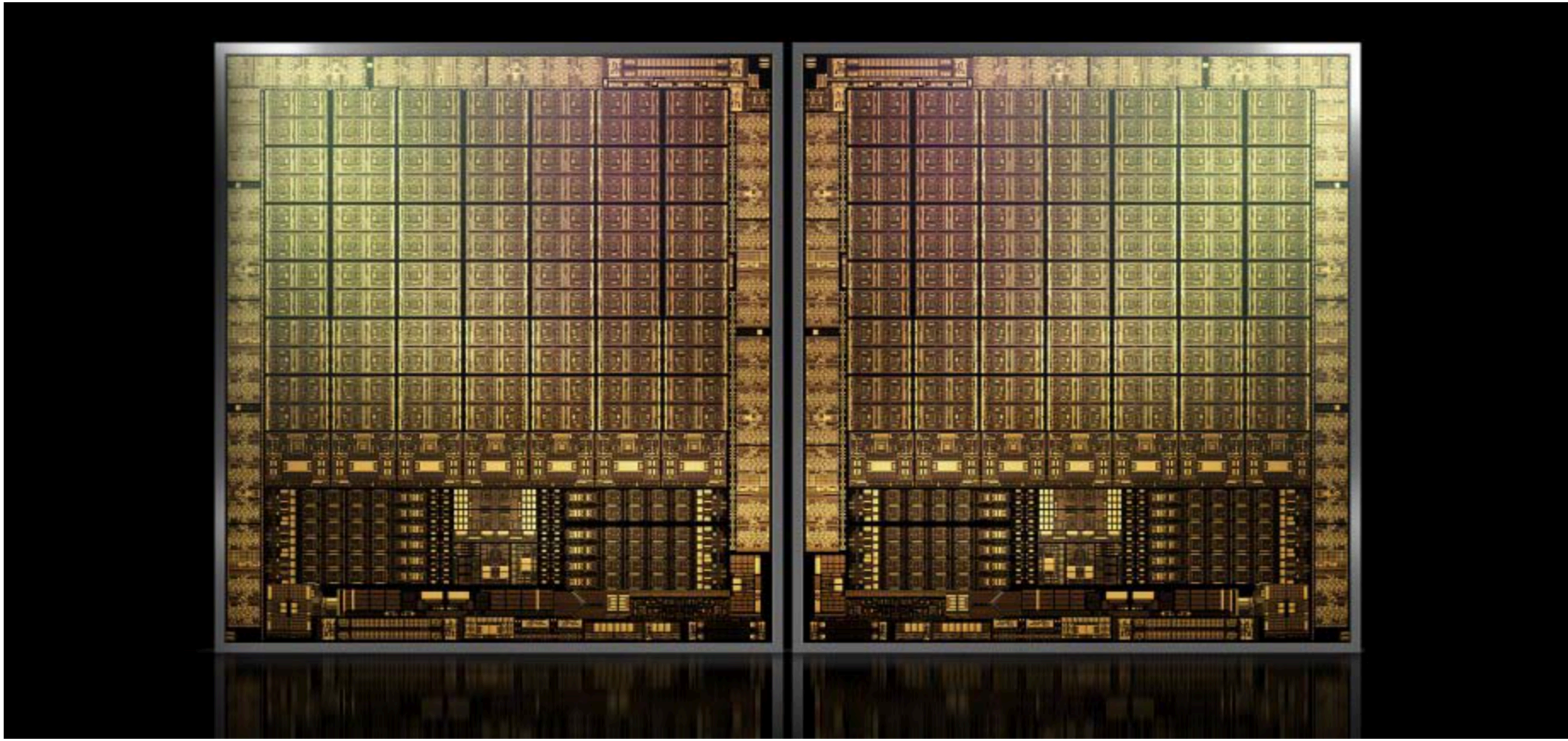


* Large number of small cores
* Less control structured and more processing units
*Less flexible program model
*There're more restrictions but Requires a lot less power

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


PCI Express 3.0 Host Interface
GigaThread Engine
GPC
TPC
SM
Memory Controller
HBM2
L2 Cache
High-Speed Hub
NVLink


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



•Hopper GPU (H100) with over 80 Billion Transistors on an 814 mm²
•80 GB memory
•First support PCIe Gen5 and utilize the HBM3 enabling 3TB/s.
•30Tflops of peak FP64, 60Tflops with FP64 tensor-core or 32FP performance.

What and Why CUDA C/C++ ?

- CUDA = ”Compute Unified Device Architecture”

  * Introduced and released in 2006 for the GeForce 8800*

- GPU = dedicated super-threaded, massively data parallel - co-processor

C/C++ plus a few simple extensions

- Compute oriented drivers, language, and tools

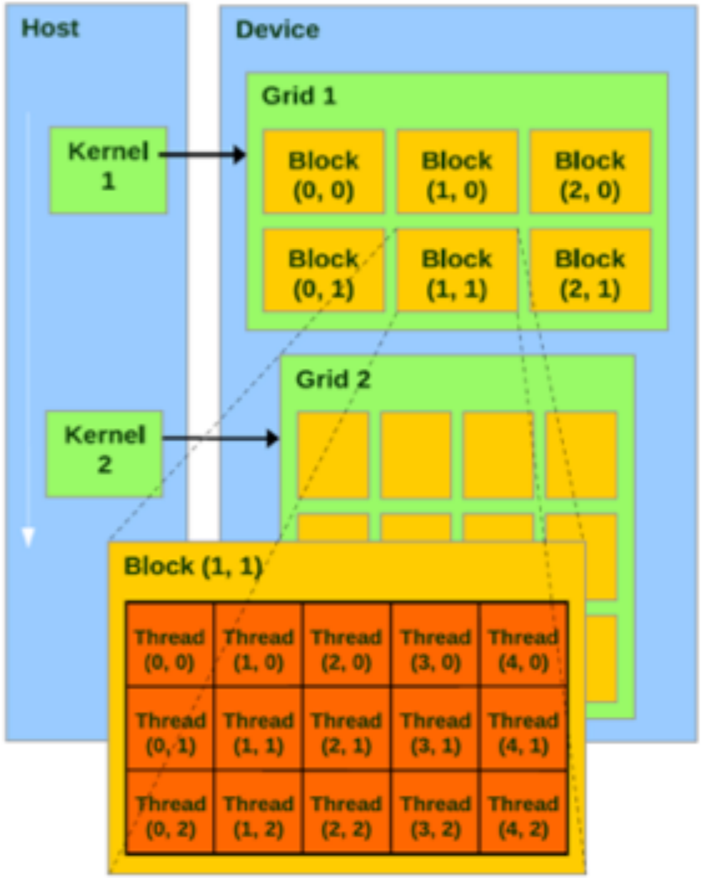
Allows HPC programmers to model problems and achieve up to 100x performance.

Documentations:

CUDA_C_Programming_Guide.pdf
CUDA_C_Getting_Started.pdf
CUDA_C_Toolkit_Release.pdf

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



CUDA C/C++

Terminology:

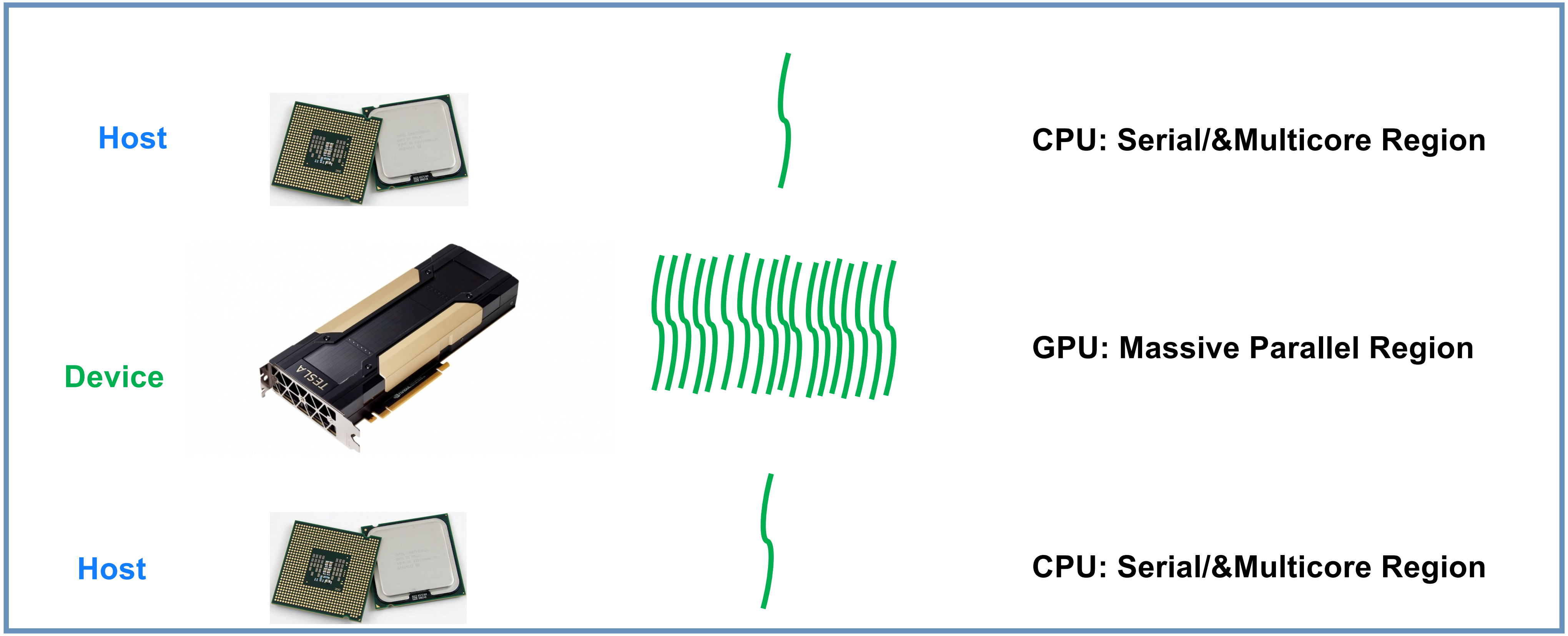
Host: The CPU and ist memory
(host memory)

Host

Device: The GPU and ist memory (device memory)

Device

CUDA Devices and Threads Execution Model



CUDA C/C++

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++

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

PRACE NVIDIA DEEP LEARNING INSTITUTE lrz

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 **NVIDIA Command Line Profiler (nprof)** to profile accelerated application performance.
- 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 increased performance.
- Employ an iterative development cycle to rapidly accelerate and deploy applications.

CSC VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

CUDA® PROFILING TOOLS

nvvp: 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.

Nsight Systems

Nsight Compute

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

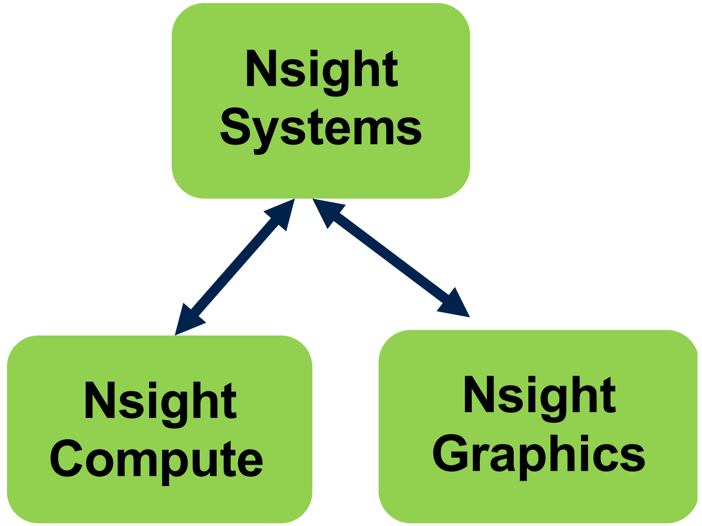
NSIGHT PRODUCT FAMILY

Standalone Performance Tools:

Ns- Systems – System-wide application algorithm tuning

Ns- Compute – Debug/&Profile specific CUDA kernels

Ns- Graphics – Analyze/&Optimize specific graphics workloads



IDE Plugins

Nsight Eclipse Edition/Visual Studio – editor, debugger, some perf analysis

Nvprof will be replaced with **nsys –profile=true**

Docs/product: **https://developer.nvidia.com/nsight-systems**

NSIGHT SYSTEMS

System-wide application algorithm tuning
- Multi-process tree support

Locate optimization opportunities
- Visualize millions of events on a very fast GUI timeline
- Or gaps of unused CPU and GPU time

Balance your workload across multiple CPUs and GPUs
- CPU algorithms, utilization, and thread state
- GPU streams, kernels, memory transfers, etc

Multi-platform: Linux & Windows, x86-64, Tegra, Power, MacOSX (host only)

GPUs: Volta, Turing

Docs/product: **https://developer.nvidia.com/nsight-systems**

NSIGHT COMPUTE

CUDA Kernel profiler

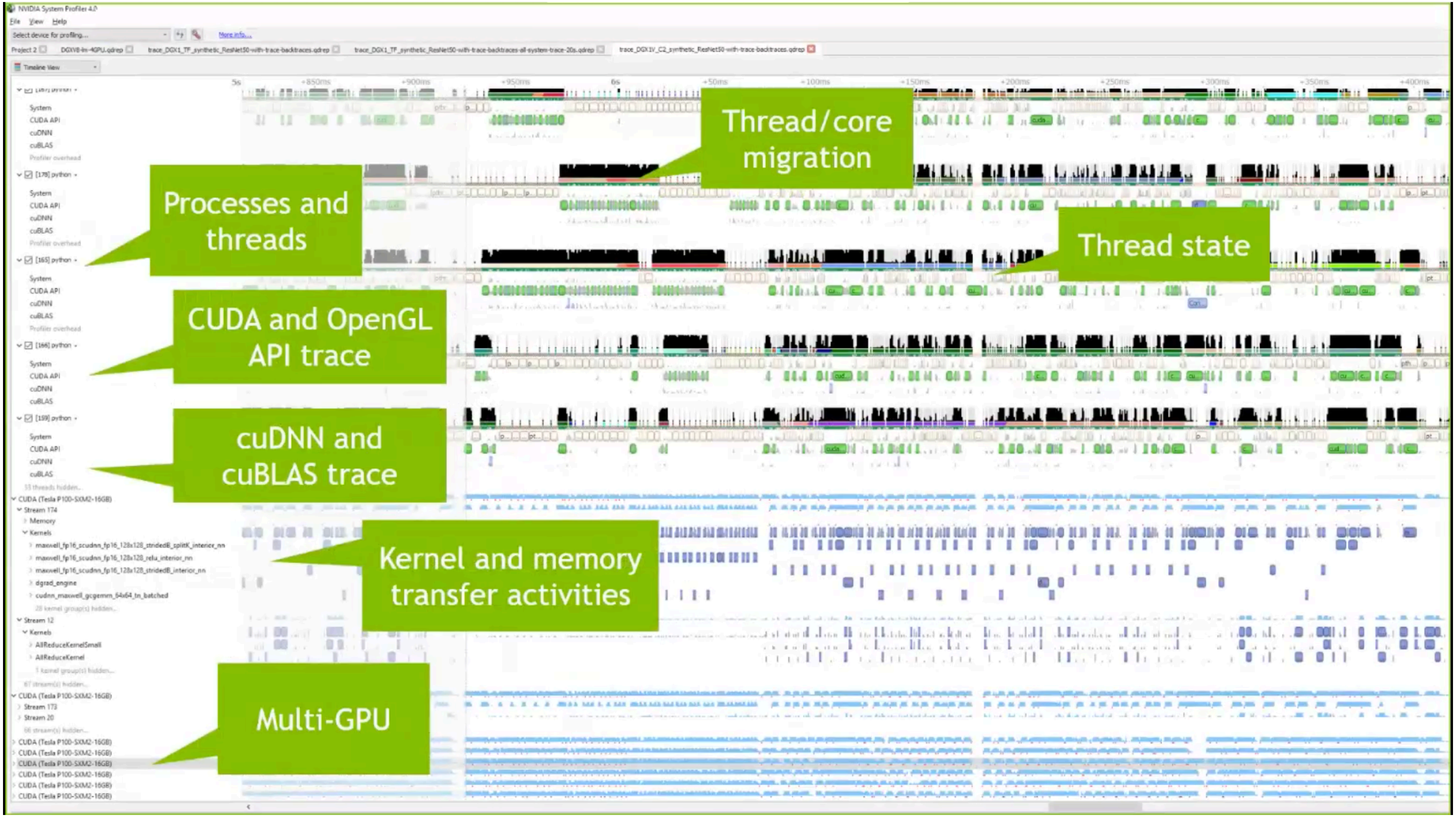
Targeted metric sections for various performance aspects (Debug/&Profile)

Very high freq GPU perf counter, customizable data collection and presentation (tables, charts ..,)

Python-based rules for guided analysis (or postprocessing)

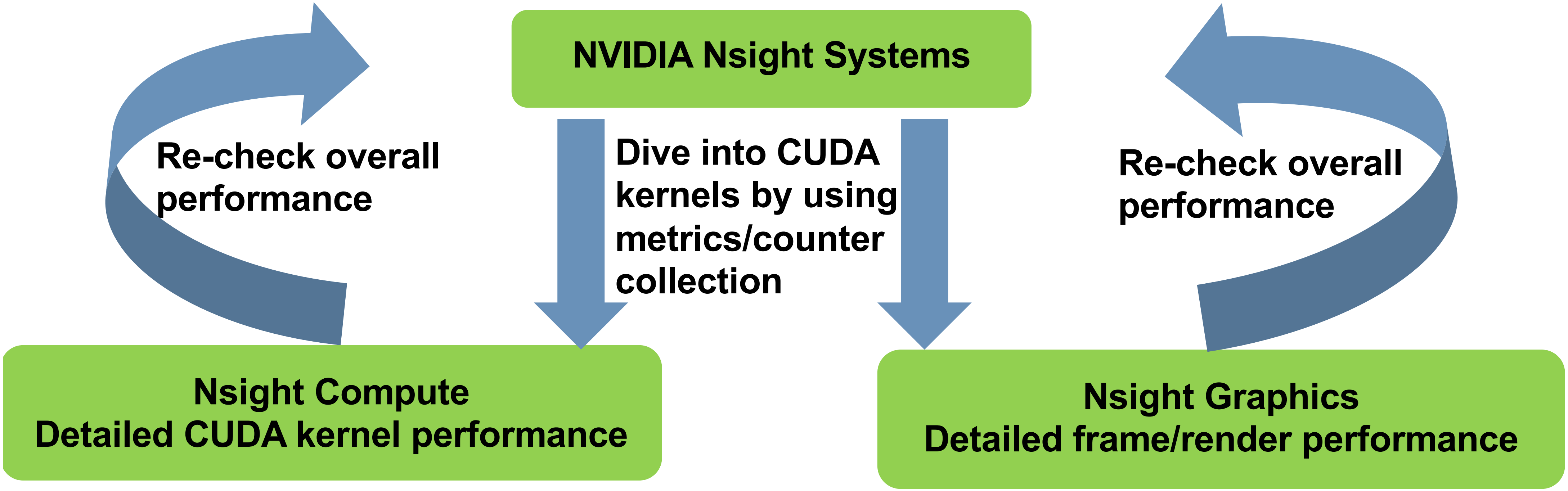
GPUs: Volta, Turing, Amper

Docs/product: **https://developer.nvidia.com/nsight-systems**



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

NSIGHT PRODUCT FAMILY



Nsight Systems - Analyze application algorithm system-wide
Nsight Compute - Debug/optimize CUDA kernel
Nsight Graphics - Debug/optimize graphics workloads

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 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++

DEEP LEARNING INSTITUTE

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 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 faulting 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:

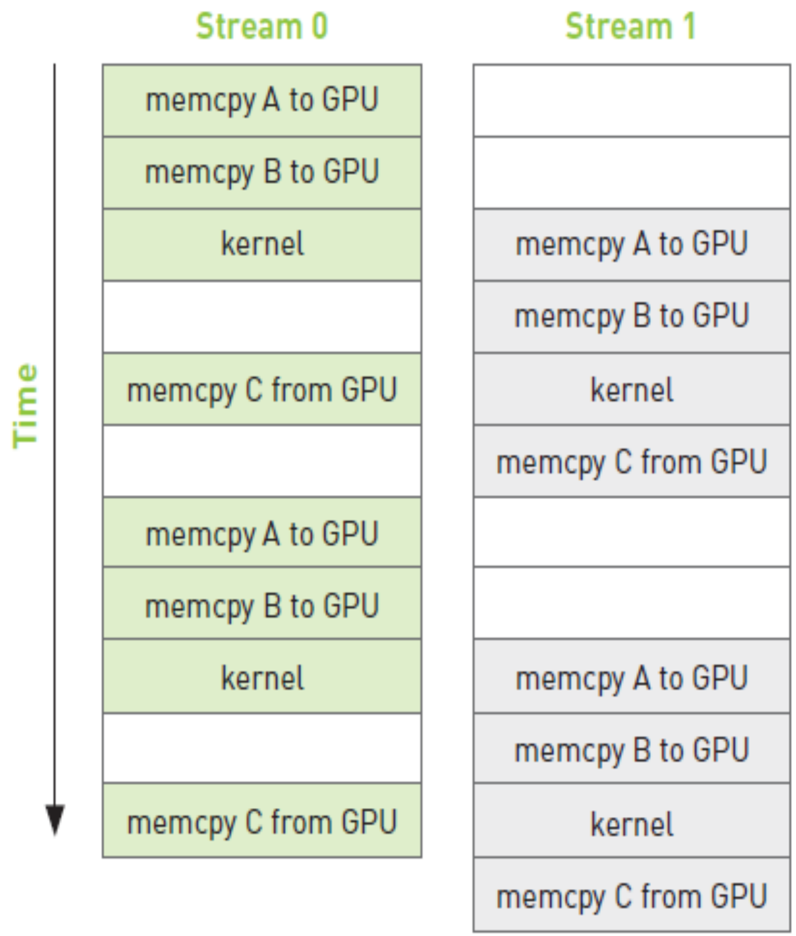
- Use the **Nsight Systems** to visually profile the timeline of GPU-accelerated CUDA applications.
- Use Nsight Systems to identify, and exploit, optimization opportunities in GPU-accelerated CUDA applications.
- Utilize CUDA streams for concurrent kernel execution in accelerated applications.
- **(Optional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

CSC

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

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);
}
```

THANK YOU

Instructor: Dr. Momme Allalen

www.nvidia.com/dli