DGIST
Daegu Gyeongbuk
Institute of Science & Technology

Introduction To GPU Programming With CUDA C++

IDS Lab Seminar – Matt Bodenham [March 2022]

IDSLab
Intelligent Digital Systems

Today's Seminar

- Introduction to GPUs ☺
  - Why GPUs?
  - CPU vs GPU
- CUDA & CUDA Architecture 😀
  - What is CUDA?
  - CUDA Architecture History
  - Streaming Multiprocessor
- CUDA Programming 😍
  - Heterogeneous Programming
  - Kernels, Threads, Blocks & Grids
  - Coding Examples

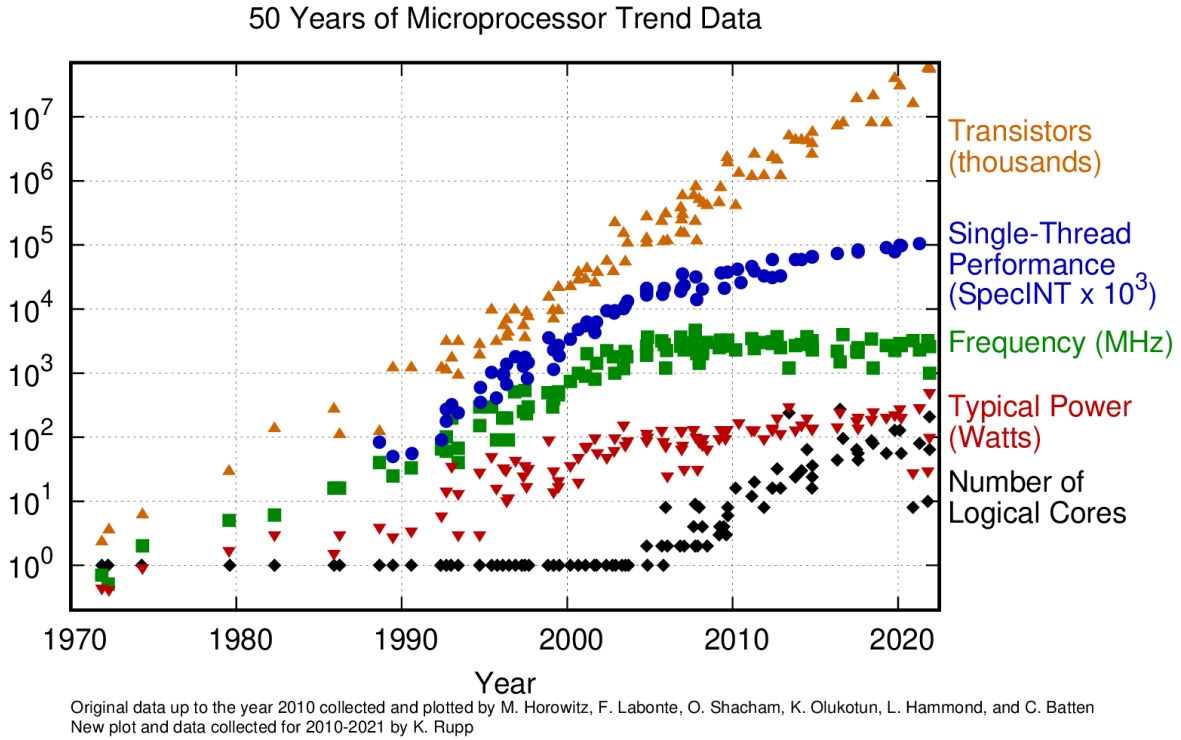
DGIST
Daegu Gyeongbuk
Institute of Science & Technology

Introduction to GPUs

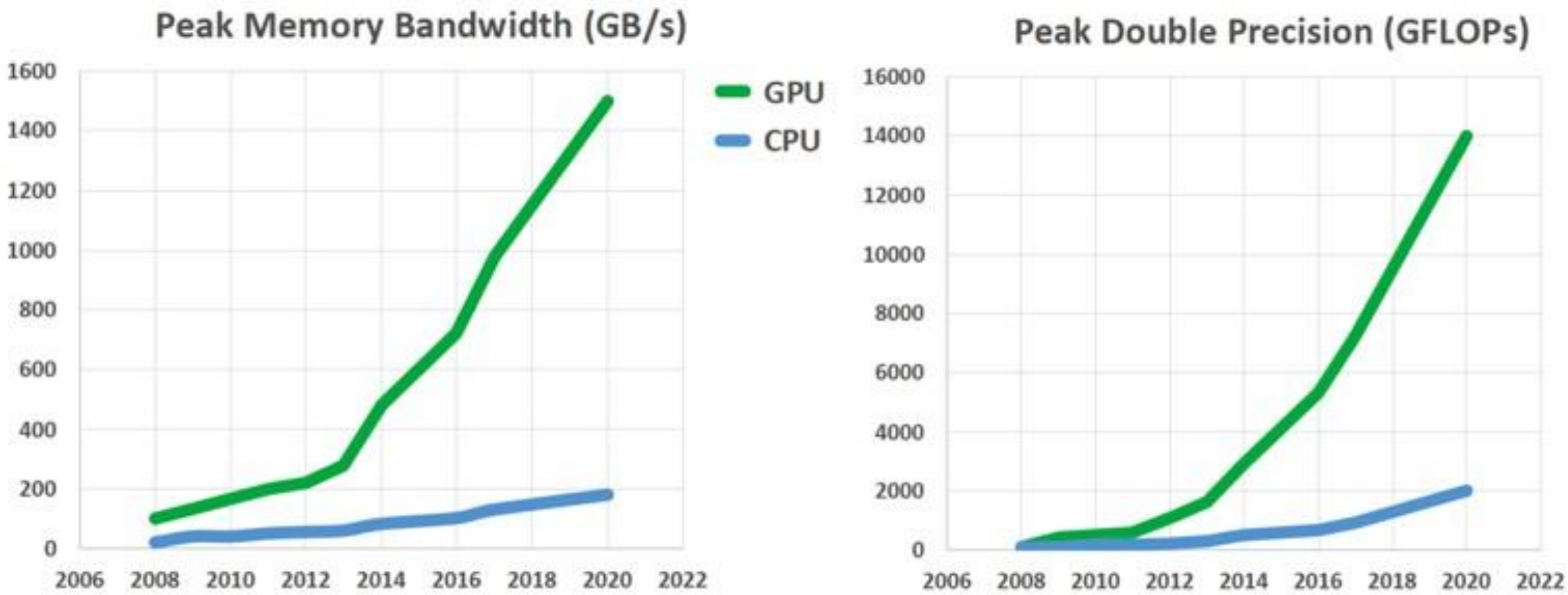
IDSLab
Intelligent Digital Systems

Why GPUs? Why now? 😢

- CPU single core performance is stagnating 😢
- Physical limit on clock speed per core
- 1970 – 2005 – **Increase clock speed** for more performance
- 2005 – Present – **Increase cores** for more performance
- Single Core -> Dual Core -> Multi Core
- More Cores = More Compute 😁

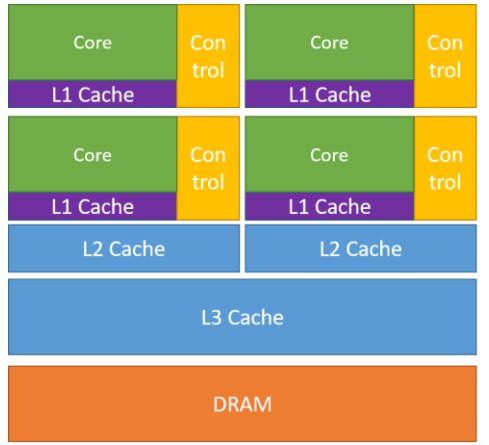


CPU vs GPU Compute

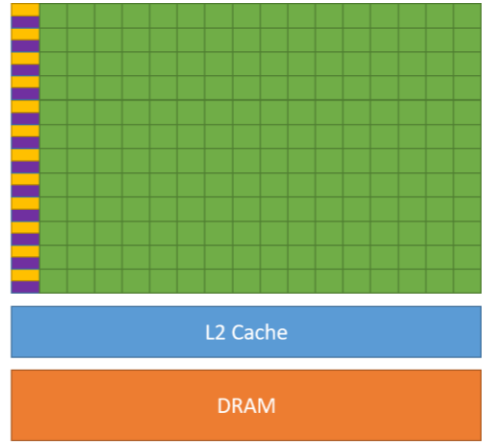


CPU vs GPU Architecture

CPU	GPU
Several cores	Many cores
Low latency	High throughput
Task parallelism	Data parallelism
Handful of operations at once	Thousands of operations at once
Low memory bandwidth	High memory bandwidth
Single control unit per core	Shared control logic between cores
Low core density	High core density



CPU



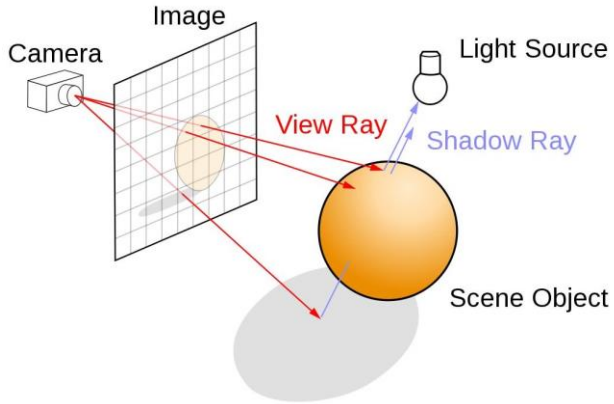
GPU

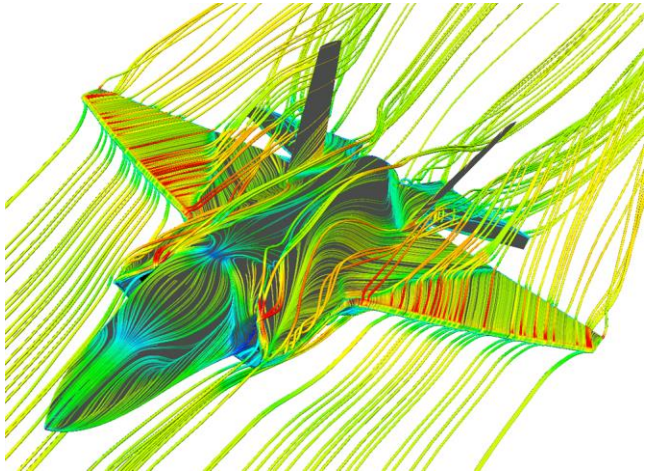
CPU vs GPU Architecture

Mythbusters Demo GPU versus CPU - YouTube

Applications for GPU Programming

- Artificial Intelligence (AI)
  - Computer Vison
  - Deep Learning
  - Machine Learning
- Computer Graphics
  - Ray Tracing
  - Real-time VFX
  - Augmented and Virtual Reality
- Simulation
  - Astrophysics
  - Computation Fluid Dynamics (CFD)
  - Medical





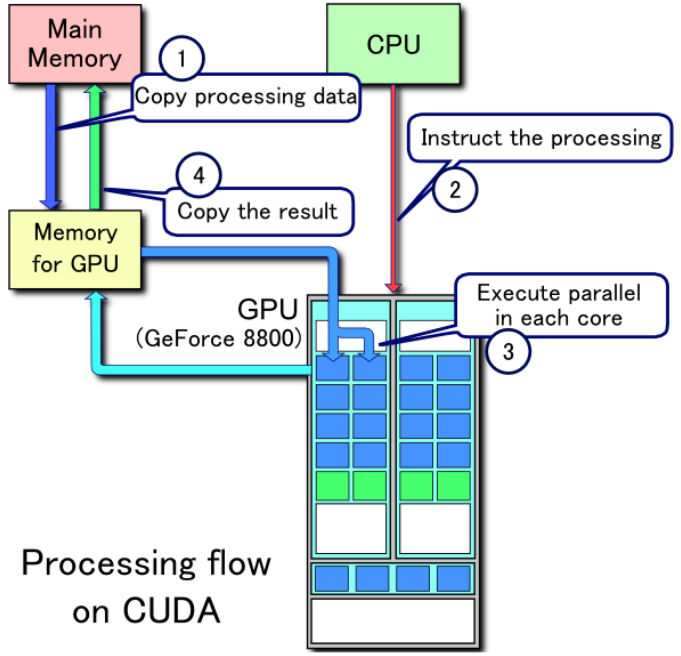
DGIST
Daegu Gyeongbuk
Institute of Science & Technology

CUDA & CUDA Architecture

IDSLab
Intelligent Digital Systems

What is CUDA?

- CUDA (Previously **C**ompute **U**nified **D**evice **A**rchitecture) is a parallel computing platform and API
- Developed by NVIDIA
- General-purpose computing on GPUs (GPGPU)
- Direct access to the GPU's virtual instruction set and parallel computational elements
- C, C++, and Fortran
- Provided libraries
  - cuBLAS – CUDA Basic Linear Algebra Subroutines library
  - cuFFT – CUDA Fast Fourier Transform library
  - cuRAND – CUDA Random Number Generation library
  - cuSPARSE – CUDA Sparse Matrix library



CUDA 1.0 😭

- Prior to the release of CUDA 😢
  - Vertex and pixel shaders only
  - No compute
- First CUDA GPU Released November 2006
  - GeForce 8800 GTX
  - Same year as The Elder Scrolls IV: Oblivion
- CUDA 1.0 Released June 2007 🎉
- CUDA 1.0 Innovations
  - IEEE compliant for single-precision floating-point
  - General computing instruction set
  - Software managed L1 cache – Shared memory

CUDA Architecture History

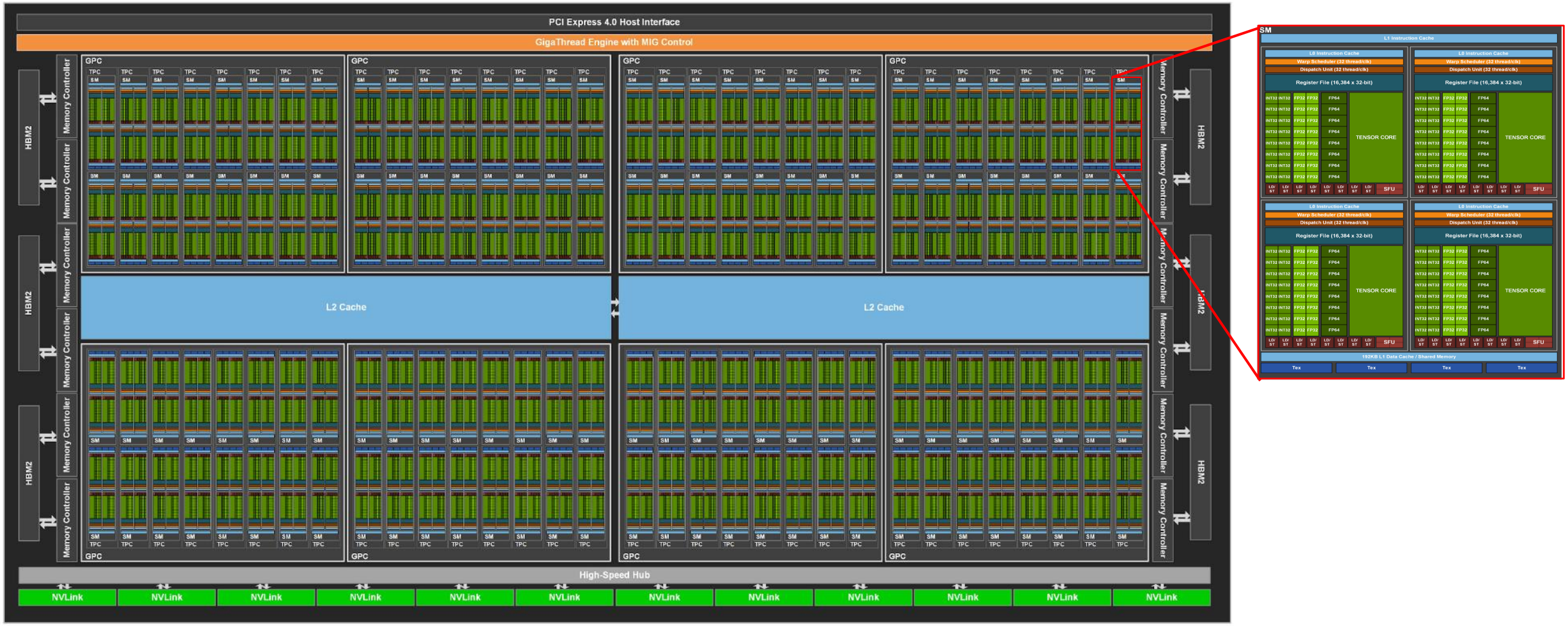
Year	Architecture	Series	Process	Notes
2006	Tesla	GeForce 8	65nm	First CUDA
2010	Fermi	GeForce 400	40nm	
2012	Kepler	GeForce 600	28nm	
2014	Maxwell	GeForce 900	28nm	
2016	Pascal	GeForce 10	16nm	Unified Memory
2017	Volta	Titan V	12nm	Tensor Cores
2019	Turing	GeForce 16	12nm	Ray Tracing (RTX)
2020	Ampere	GeForce 30	7nm	
2022	Lovelace	GeForce 40	TBC	TBC

CUDA Streaming Multiprocessor (SM) 😭

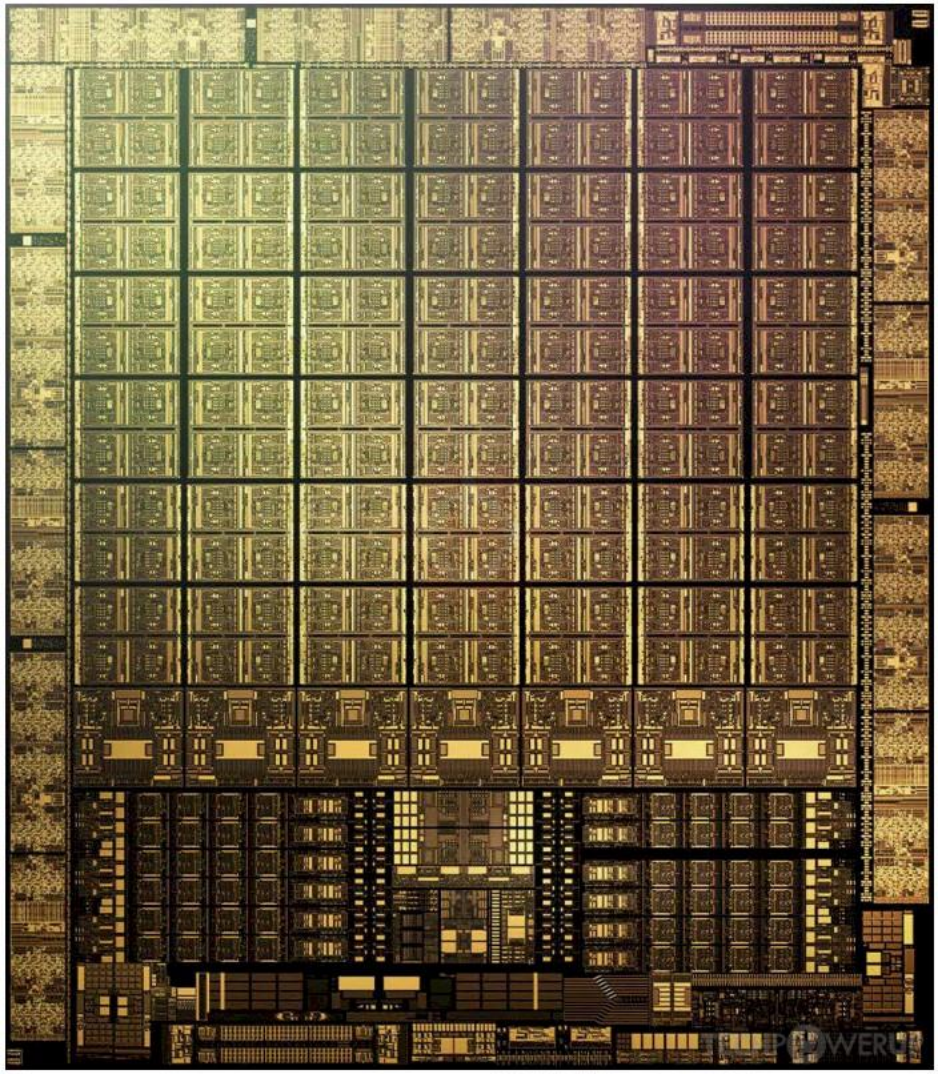
- Equivalent to a core
- A100 GPU SM
  - Purely compute
  - No ray tracing cores = no gaming 😭
- Data types
  - INT32
  - FLOAT32
  - FLOAT64
- Tensor cores 💪
  - Flexible data types
  - FP32, Tensor Float 32 (TF32), FP16, INT8, INT4 and bfloat16
  - Mixed precision computing
  - Advanced level programming
- L1 Data Cache – shared between blocks (low latency) 🏎

Lots of SMs - 108 SMs

A100 Architecture



Ampere Die Shot (Render)



DGIST
Daegu Gyeongbuk
Institute of Science & Technology

CUDA Programming

IDSLab
Intelligent Digital Systems

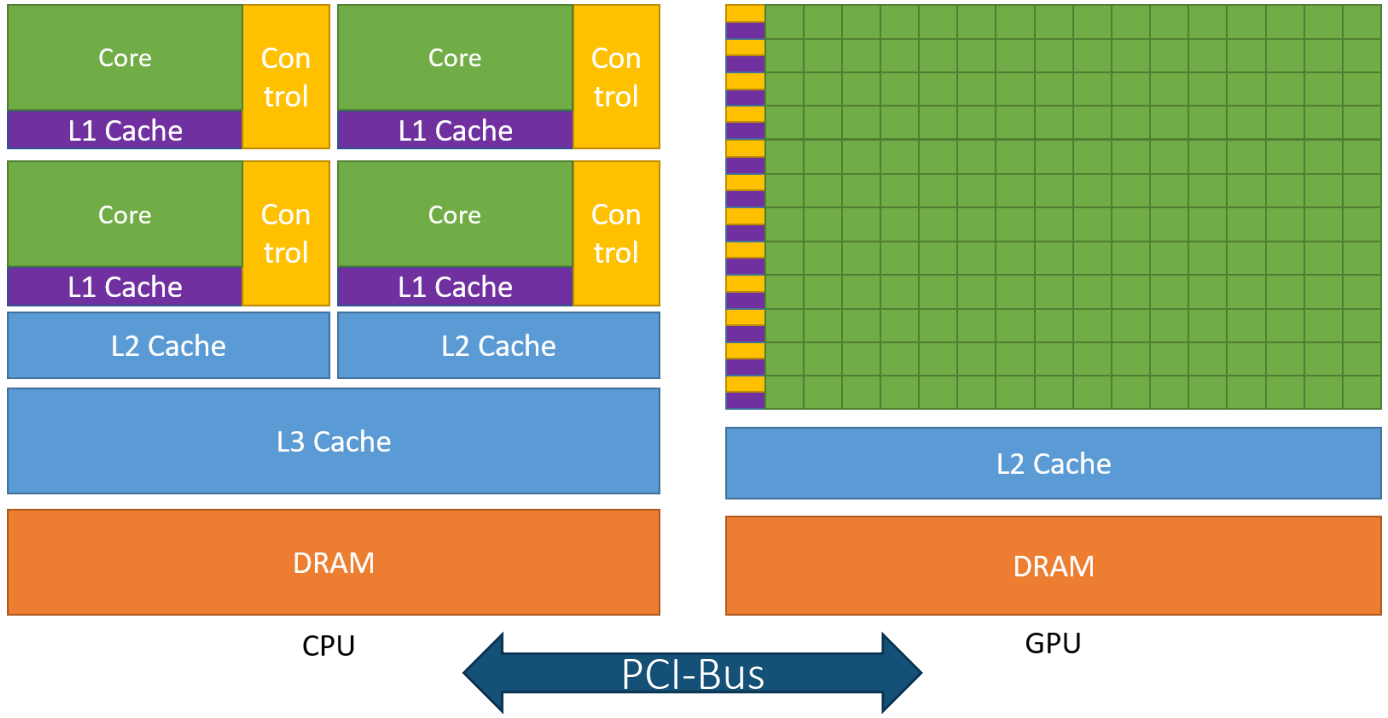
CUDA Programming

- Heterogeneous programming
  - Program CPU and GPU is same file
- C, C++, Fortran
- CUDA Wrappers for Python
  - CUDA Python – Cython/Python wrapper for CUDA runtime
  - CuPy – Numpy with CUDA compute
  - Numba – Python complier

Host & Device

Heterogeneous programming

- Heterogeneous programming
  - Both host and device
  - Make use of each's strengths
- CPU is host
  - Controls the program
- GPU is device
  - Typically performs inner loop operations
- Communication over PCI-Bus
  - PCI Bus is slow relative to host and device



Kernel & Threads

- Kernel
  - Usually inner-loop
  - Operation that is repeated many times
  - Must be data parallelizable
- Threads
  - Single Instruction, Multiple Threads (SIMT)
  - Repeating same kernel on different data points

```
for(int i = 0; i < N; i++){
    c[i] = a[i] + b[i]
}
```

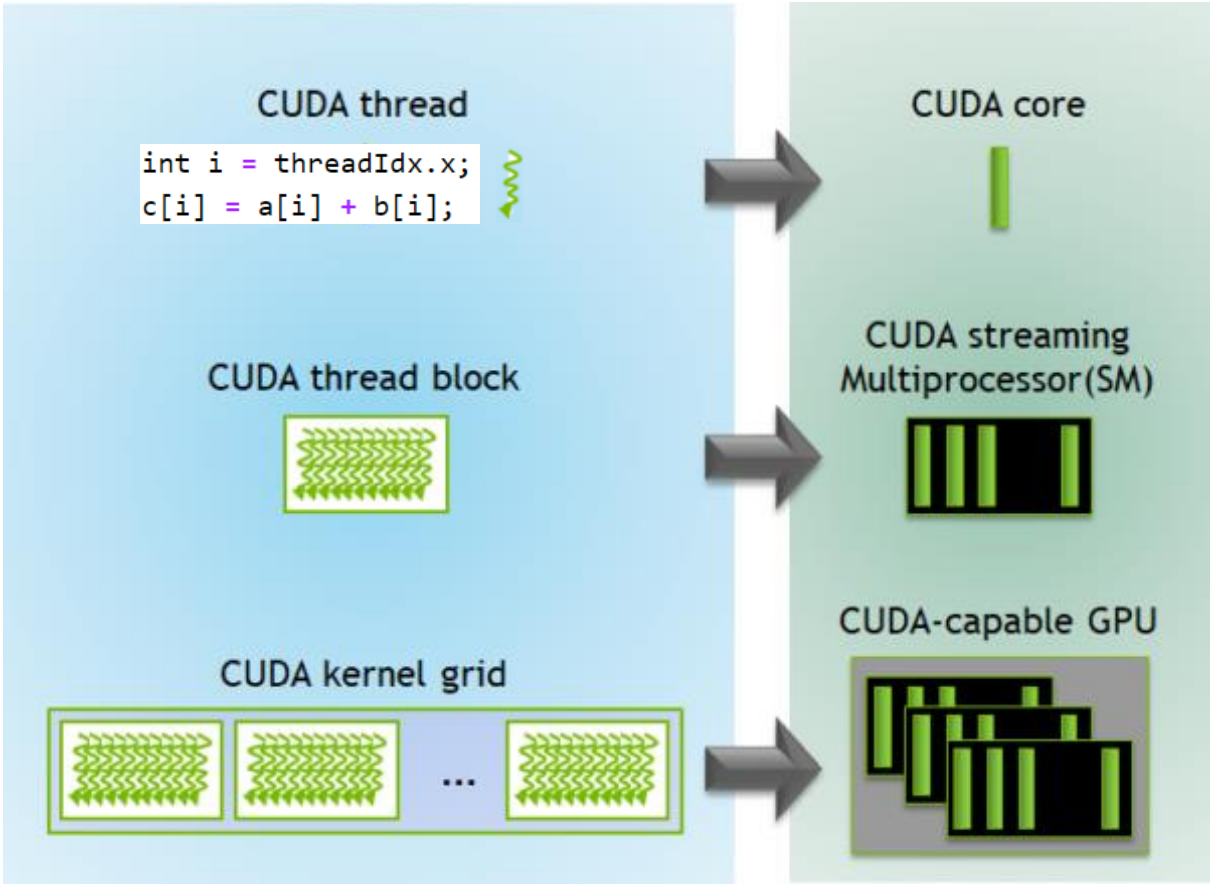
To Kernel

```
__global__ void addArrays(int *a, int *b, int *c){
  int i = threadIdx.x;
  c[i] = a[i] + b[i];
}
```

Thread Index	0	1	…	N
Code	`c[0] = a[0] + b[0]`	`c[1] = a[1] + b[1]`	…	`c[N] = a[N] + b[N]`

Threads, Blocks & Grids

- Thread – One thread runs a single kernel
- Block – Multiple treads working simultaneously
  - Single SM
  - Shared memory
- Grid – Queue of blocks
  - Multiple SM running simultaneously



Coding Examples 😍

Yay!

- Let’s go to Jupyter Notebook 🚀
- All files on our lab’s GitHub
  - https://github.com/IDS-Lab-DGIST/cuda-intro

Want to know more?

- YouTube - Tom Nurkkala
  - Intro to GPU Programming - https://youtu.be/G-EimI4q-TQ
  - CUDA Hardware - https://youtu.be/kUqkOAU84bA
  - CUDA Programming - https://youtu.be/xwbD6fL5qC8
- Textbook – CUDA By Example, Jason Sanders & Edward Kandrot
  - 2 copies in 623
  - 예제로 배우는 CUDA 프로그래밍 – 3 copies in 623?
  - https://www.notion.so/a6fe119cf6634cd095741858712 93efa
  - PDF - http://www.mat.unimi.it/users/sansotte/cuda/CUDA_by_Example.pdf
- NVIDIA Documentation
  - Programming Guide - https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
  - Best Practices - https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html