RAISE Center of Excellence

Massively Parallel GPU Computing with CUDA: Introduction

Overview of CUDA Architecture and CUDA Programming Model

Arnis Lektauers, Riga Technical University
arnis.lektauers@rtu.lv

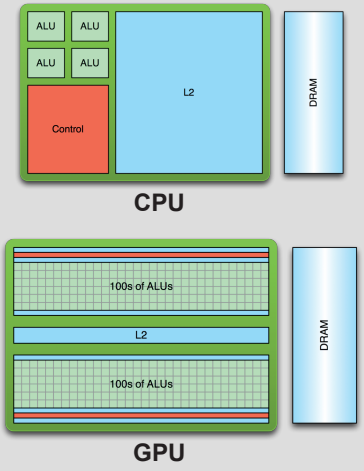
19. 01. 2022.

Day 1: Content

1. Overview of CUDA architecture and programming model:
   - GPU evolution
   - CUDA GPU architecture
2. Basic CUDA programming:
   - Brief revise of CUDA programming model
   - Key principles
   - Introduction to the concept of threads & blocks
   - Host-device data transfer
3. Hands-on exercises on writing simple CUDA programs:
   - Simple programs with C/C++
   - Using CUDA on HPC cluster
   - Simple programs with Python and CuPy

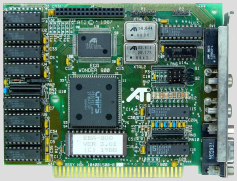
GPU Computing

- **GPU** - *Graphics Processing Unit*
  - Optimized for data-parallel, throughput computation
  - Traditionally used for real-time rendering
  - High computational density (100s of ALUs) and memory bandwidth (100+ GB/s)
  - Architecture tolerant of memory latency - 1000s of concurrent threads to hide latency (vs. large fast caches)
- **GPGPU** - *General Purpose Computing on GPU*

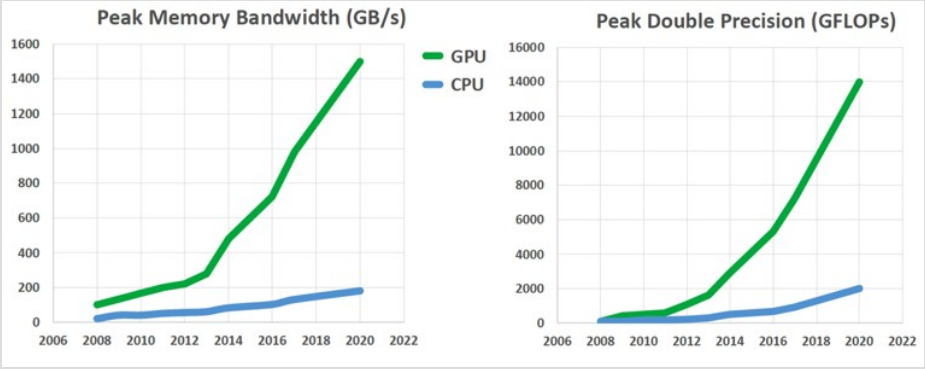


GPU History

1. 1951 - 1976: Formation of the base for today's GPUs
2. 1976 - 1995: The early days of 3D consumer graphics
3. 1995 - 1999: 3Dfx Voodoo: the game-changer
4. 2000 - 2006: The Nvidia vs. ATI Era Begins
5. 2006 - 2013: The modern GPU: stream processing units a.k.a. GPGPU
6. 2013 - 2020: Pushing GPU technology into new territory



CPU vs GPU Performance



Peak memory bandwidth in GB/s and peak double precision gigaflops for GPUs and CPUs since 2008

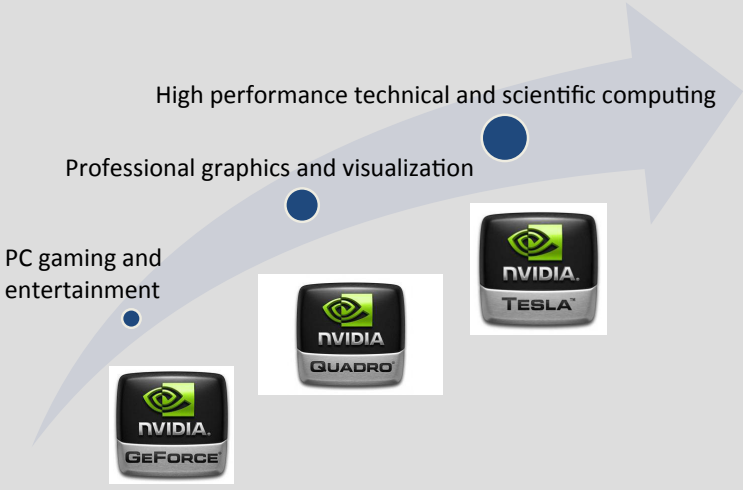
Source: The Next Platform: A Decade of Accelerated Computing Augurs Well for GPUs, 2019

What is CUDA?

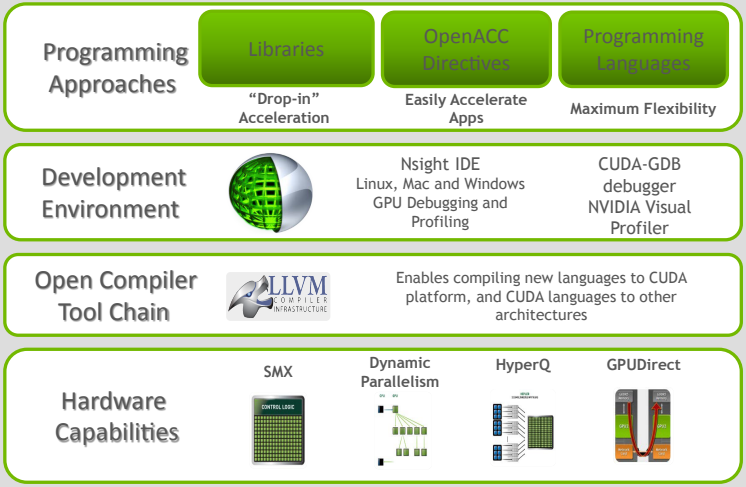
CUDA (*Compute Unified Device Architecture*) is a parallel computing platform and programming model created by NVIDIA and implemented by the graphics processing units (GPUs) that they produce

- Introduced in February 2007
- CUDA Architecture
  - Expose GPU computing for general purpose
  - Computing for general purpose
- CUDA C/C++
  - Based on industry-standard C/C++
  - Small set of extensions to enable heterogeneous programming
  - Straightforward APIs to manage devices, memory etc.

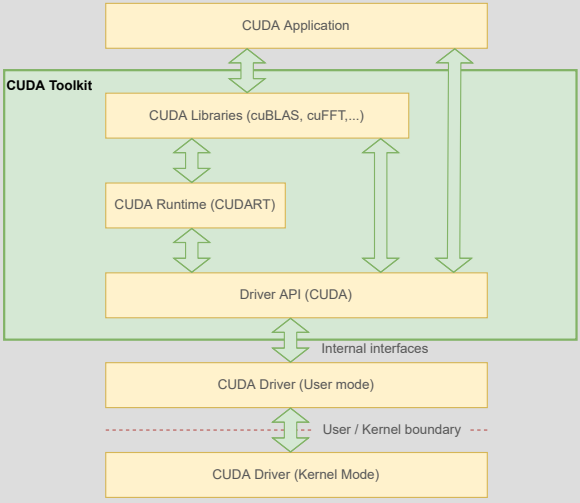
CUDA GPU Product Families



CUDA Parallel Computing Platform



CUDA Software Levels



GPU Hardware Generations

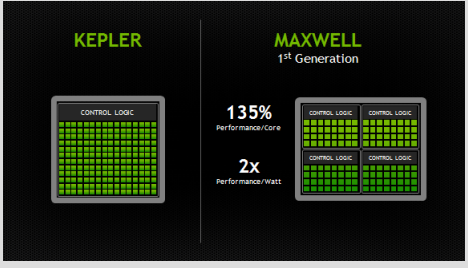
- *Tesla* hardware debuted in 2006, in the GeForce 8800 GTX (G80)
  - CUDA cores: 240
  - Compute capability: 1.0-1.3

- *Fermi* hardware debuted in 2010, in the GeForce GTX 480 (GF100)
  - CUDA cores: 512
  - 64-bit addressing
  - L1 and L2 cache
  - Compute capability: 2.0, 2.1

- *Kepler* hardware debuted in 2012, in the GeForce GTX 680 (GK104)
  - CUDA cores: 1536 (GK104), 2880 (GK110)
  - Compute capability: 3.0, 3.5

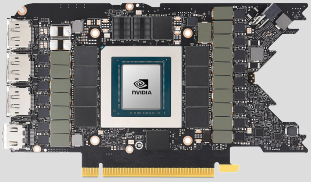
GPU Hardware Generations (1)

- *Maxwell* hardware debuted in 2014, in the GeForce GTX 750 Ti (GM107):
  - CUDA cores: 640 (GM107)
  - Compute capability: 5.0
- Two generations of Maxwell GPU:
  - 28nm: GM108, GM107
  - 20nm: GM206, GM204, GM200



GPU Hardware Generations (2)

- *Pascal* debuted in 2016, in the GeForce GTX 1080 (GP100):
  - CUDA cores: 3840
  - 16 nm technological manufacturing process
  - HBM2 3D memory
  - 4096-bit memory bus
  - *NVLink* - high speed GPU-CPU interconnection
  - Compute capability: 6.0
- *Volta* debuted in 2017, in the Tesla V100 (GV100)
- *Turing* debuted in 2018, in the GeForce RTX 2080 Ti (TU102)
- *Ampere* debuted in 2020, in the GeForce RTX 3080 (A100)



GPU Compute Capability

- The compute capability of a device describes its architecture, e.g.
  - Number of registers; sizes of memories; features & capabilities

Compute Capability	Main Features
1.0	CUDA basic support
1.3	Double precision, improved memory access, atomic functions
2.0	Cache improvements, 3D grid, surfaces, concurrent kernels, P2P
3.0	Improved performance, *Warp Shuffle* functions
3.5	Dynamic parallelism, *Funnel Shift* functions
5.0	Maxwell architecture support
6.0	Pascal architecture support
7.0	Volta architecture support
7.5	Turing architecture support
8.0; 8.6	Ampere architecture support

Fermi GPU Architecture

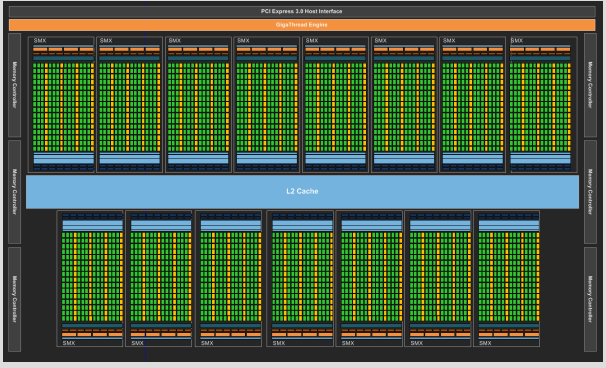
Fermi GF100 Processor

- 3.0B Transistors
- 16 SM units
- 870 GFLOP FP64
- 768 KB L2 Cache
- 384-bit GDDR5
- PCI Express Gen2

Kepler GPU Architecture

Kepler GK110 Processor

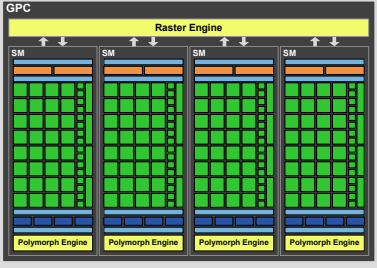
- 7.1B Transistors
- 15 SMX units
- > 1 TFLOP FP64
- 1.5 MB L2 Cache
- 384-bit GDDR5
- PCI Express Gen3



Fermi GPU Architecture
Graphics Processing Cluster (GPC)

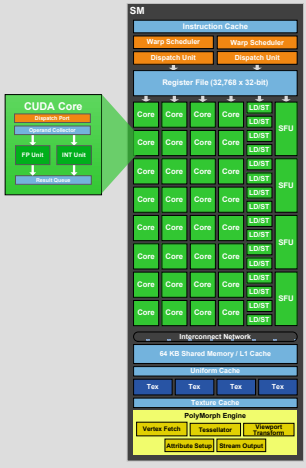
Graphics Processing Cluster (GPC) - dominant high-level GPU hardware block containing several *Streaming Multiprocessors* (SMs)

GPC encapsulates all key graphics processing units (e.g., *Scalable Raster Engine* for triangle setup, rasterization, and Z-cull)



Fermi GPU Architecture

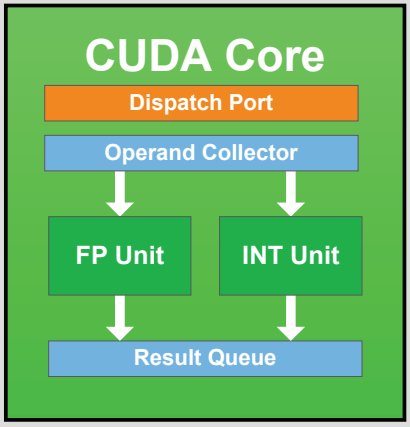
Streaming Multiprocessor (SM)



Streaming Multiprocessor (SM) - the part of the GPU that runs CUDA kernels

Each SM features:

- 32 CUDA processors (*Cores*)
- Special Function Units (SFUs) for single-precision mathematical approximations
- Dual warp scheduler
- Constant cache
- Shared memory
- Hardware for texture mapping
- *PolyMorph Engine* for vertex processing



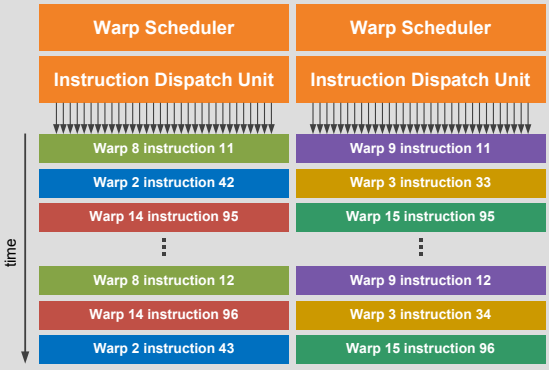
Each CUDA **Core** has a fully pipelined integer Arithmetic Logic Unit (ALU) and Floating Point Unit (FPU)

- IEEE 754-2008 floating-point standard
- Fused Multiply-Add (FMA) instruction for both single and double precision
- Logic unit
- Move, compare unit
- Branch unit

The SM schedules threads in groups of 32 parallel threads called **Warps**

Each SM features:

- Two warp schedulers
- Two instruction dispatch units, allowing two warps to be issued and executed concurrently

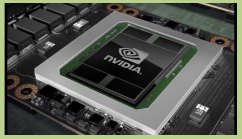


Dual warp scheduling

Pascal GPU Architecture

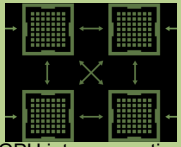
Tesla P100 GPU Accelerator for Servers

Pascal Architecture



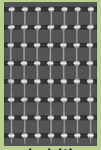
High computing performance

NVLink



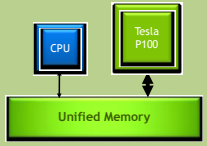
GPU interconnection for maximum scalability

CoWoS HBM2

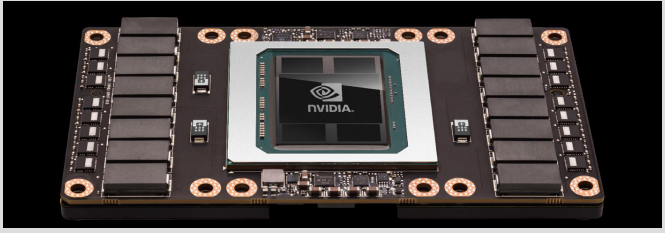


High bandwidth memory

Memory Driver



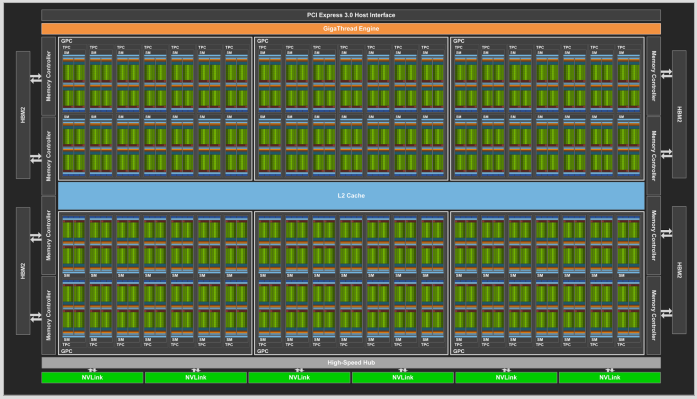
Easy programming with virtually unlimited memory



Pascal GPU Architecture

GP100 Processor

- 56 streaming multiprocessors
- 3584 CUDA cores
- 5.3 TF - double precision floating point format
- **10.6 TF** - standard precision floating point format
- 21.2 TF - half-precision floating point format
- 16 GB HBM2
- 720 GB/s memory bandwidth



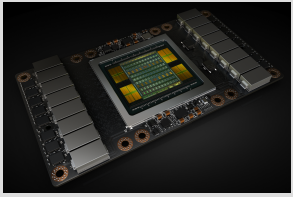
Pascal GPU Architecture

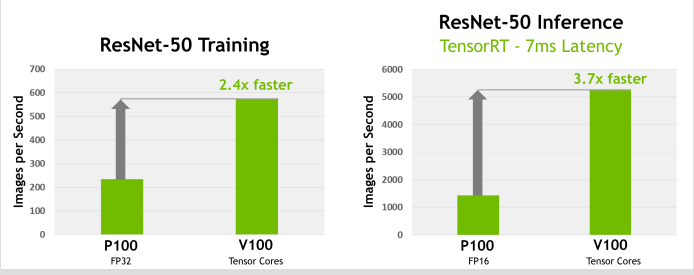
GP100 Streaming Multiprocessor

- 64 CUDA cores
- Register files: 256 KB
- Shared memory: 64 KB
- Active threads: 2048
- Active blocks: 32

Volta GPU Architecture

Tesla V100 GPU Accelerator for Servers

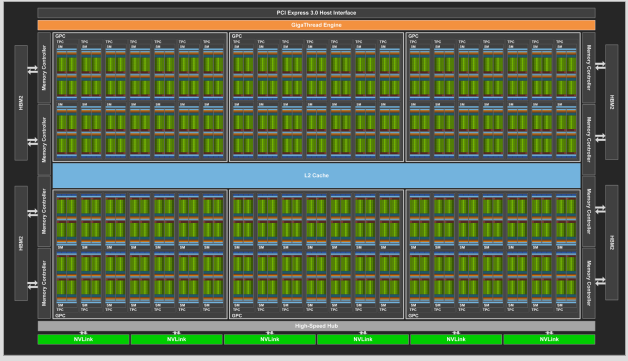




Volta GPU Architecture

V100 Processor

- 80 strreaming multitprocessors
- 5120 - standard precision floating point CUDA cores
- 7.5 TF - double precision floating point format
- **15 TF** - standard precision floating point format
- 120 TF - tensor core performance
- 16 GB HBM2
- 900 GB/s max. memory bandwidth



Volta GPU Architecture
V100 Streaming Multiprocessor

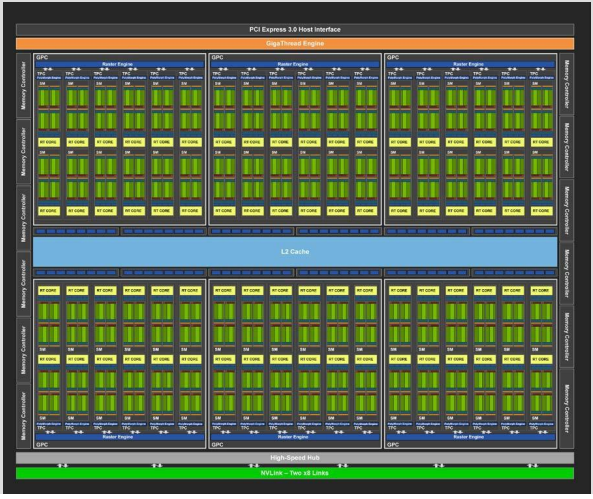
- 64 FP32 cores, 32 FP64 cores, 64 INT32 cores
- 8 new mixed precision FP16 / F32 tensor cores for deep learning matrix arithmetic
- The streaming multiprocessor is divided into 4 computing blocks, where each block contains:
  - 16 FP32, 16 INT32, 8 FP64 cores, 2 tensor cores
  - new L0 instruction cache
  - warp scheduler, dispatcher, 64 KB registry file
- Improved L1 data cache for higher performance
- Configurable max 96 KB of shared memory

Volta GPU Architecture

V100 Streaming Multiprocessor

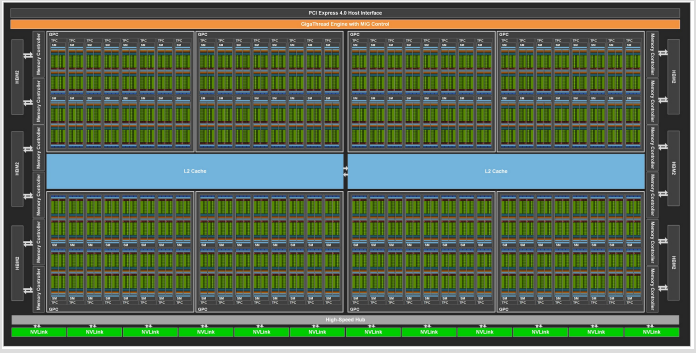
Turing GPU Architecture
TU102 Processor

- 72 streaming multiprocessors
- 4608 - CUDA cores
- **16.3 TF** - standard precision floating point format
- 576 - tensor cores
- 72 - real-time rendering kernels
- 24GB GDDR6
- 672 GB/s max memory bandwidth



Ampere GPU Architecture
GA102 Processor

- 72 streaming multiprocessors
- 10 752 - CUDA cores
- **30 TF** - standard precision floating point format
- 336 - tensor cores
- 84 - real-time rendering kernels
- 128 KB - L1 cache/shared memory
- Real-time high-quality ray tracing graphics



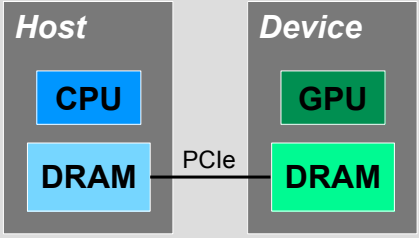
Basic CUDA Programming

Anatomy of a CUDA C/C++ Application

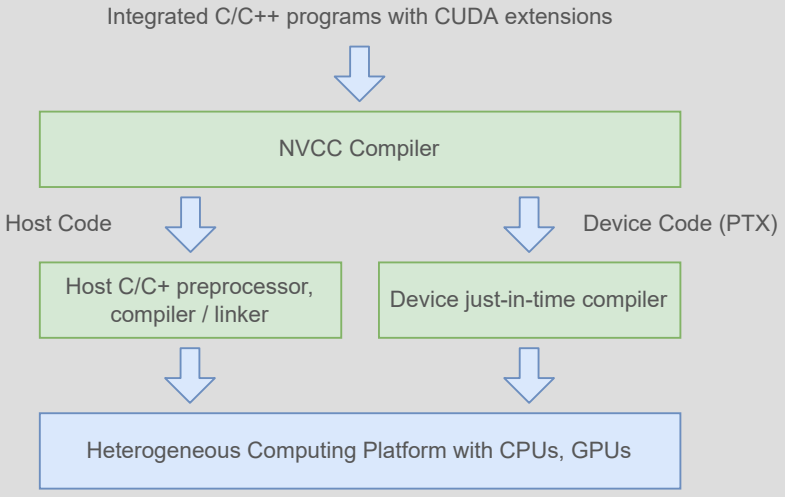
Host - the CPU and its memory (*Host Memory*)

Device - the GPU and its memory (*Device Memory*)

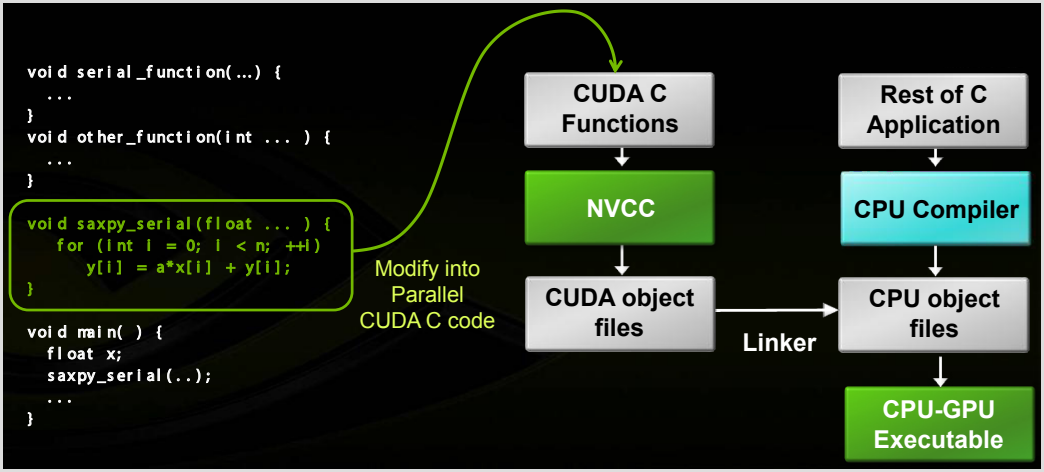
- Serial code executes in a *host* (CPU) thread
- Parallel code executes in many *device* (GPU) threads across multiple processing elements



Compiling CUDA C Application



Compiling CUDA C Applications (1)



CUDA C : C with a few keywords

Kernel: function called by the host that executes on the GPU as an array of threads in parallel

Kernel features:

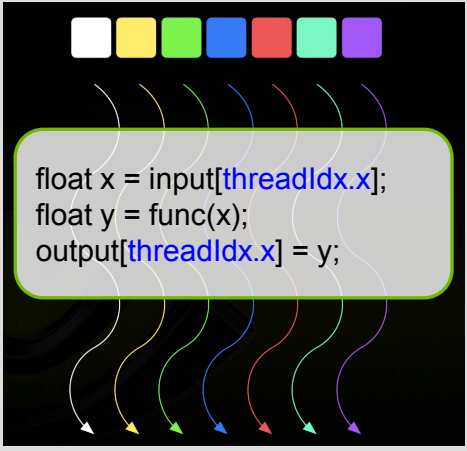
- Parallel portion of application
- Entire GPU executes kernel, many threads
- Can only access GPU memory
- No variable number of arguments
- No static variables

Functions must be declared with a qualifier:

- __global__ : GPU kernel function launched by CPU, must return void
- __device__: can be called from GPU functions
- __host__: can be called from CPU functions (default)
- __host__ and __device__ qualifiers can be combined

Threads

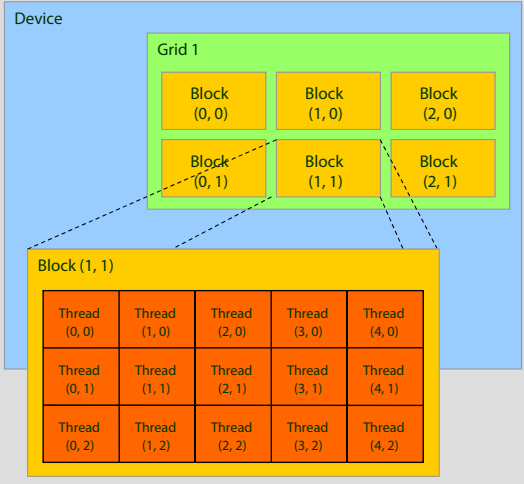
- CUDA threads:
  - Lightweight
  - Fast switching
  - 1000s execute simultaneously
- All threads execute the same code, can take different paths
- Each thread has an ID:
  - Select input/output data
  - Control decisions



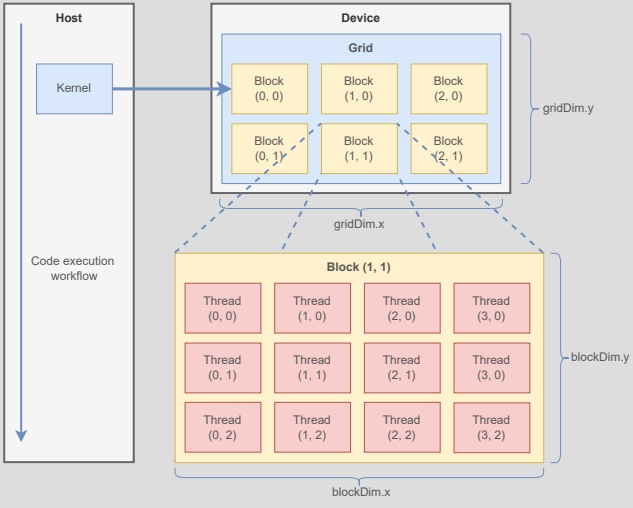
Thread Blocks and Grid

- Threads are grouped into **Blocks**:
  - Executes on a single Streaming Multiprocessor (SM)
  - Threads within a block can cooperate Light-weight synchronization
  - Data exchange
- Blocks are grouped into a **Grid:**
  - Thread blocks of a grid execute across multiple SMs
  - Thread blocks do not synchronize with each other
  - Communication between blocks is expensive
- A *kernel* is executed as a *grid* of *blocks* of *threads*

Thread Blocks and Grid (1)

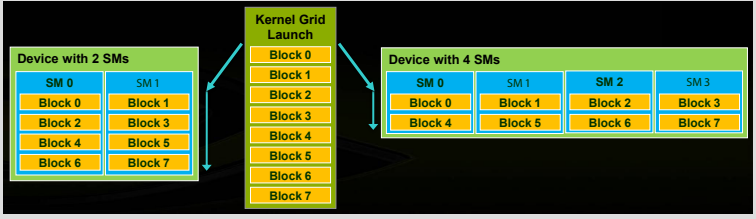


Thread Blocks and Grid (2)

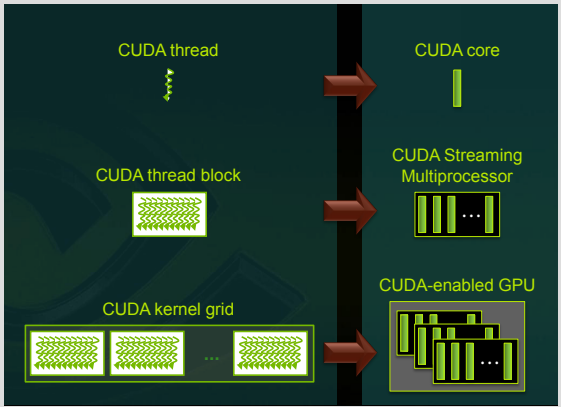


Thread Blocks

- Thread blocks allow cooperation
  - Cooperatively load/store blocks of memory all will use
  - Share results with each other or cooperate to produce a single result
  - Synchronize with each other
- Thread blocks allow scalability
  - Blocks can execute in any order, concurrently or sequentially
  - This independence between blocks gives scalability:
    - A kernel scales across any number of SMs



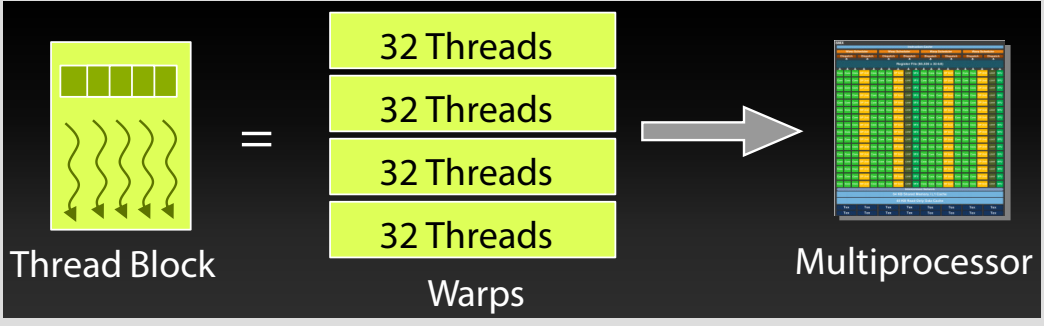
Kernel Execution



- Each thread is executed by a core
- Each block is executed by one SM and does not migrate
- Several concurrent blocks can reside on one SM depending on the blocks' memory requirements and the SM's memory resources
- Each kernel is executed on one device
- Multiple kernels can execute on a device at one time

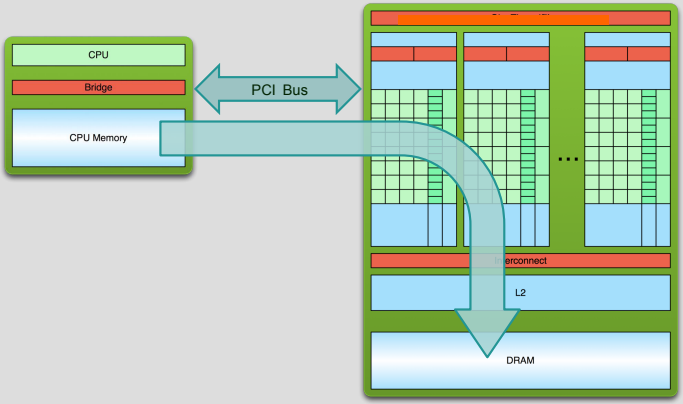
Warps

- A thread block consists of 32-thread warps
- A warp is executed physically in parallel (SIMD) on a multiprocessor



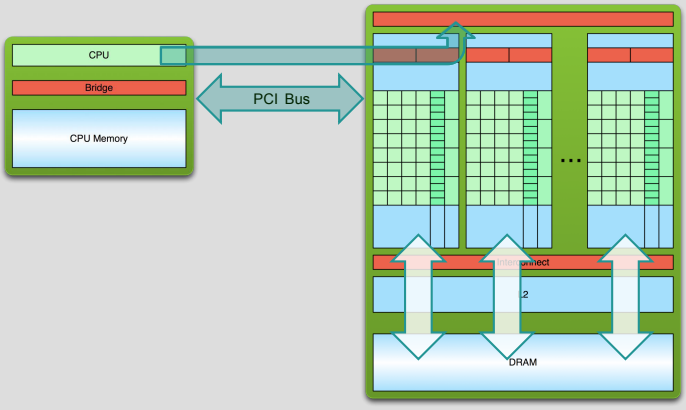
Simple Processing Flow

1. **Copy input data from CPU memory to GPU memory**



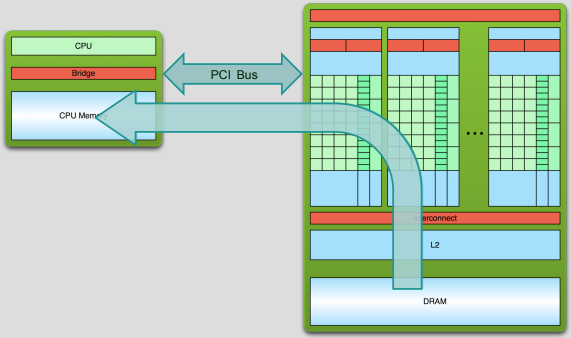
Simple Processing Flow (1)

1. Copy input data from CPU memory to GPU memory
2. **Load GPU program and execute, caching data on chip for performance**



Simple Processing Flow (2)

1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. **Copy results from GPU memory to CPU memory**



drive. enable. innovate.

RAISE Center of Excellence

The CoE RAISE project has received funding from the European Union's Horizon 2020 – Research and Innovation Framework Programme H2020-INFRAEDI-2019-1 under grant agreement no. 951733

Follow us: