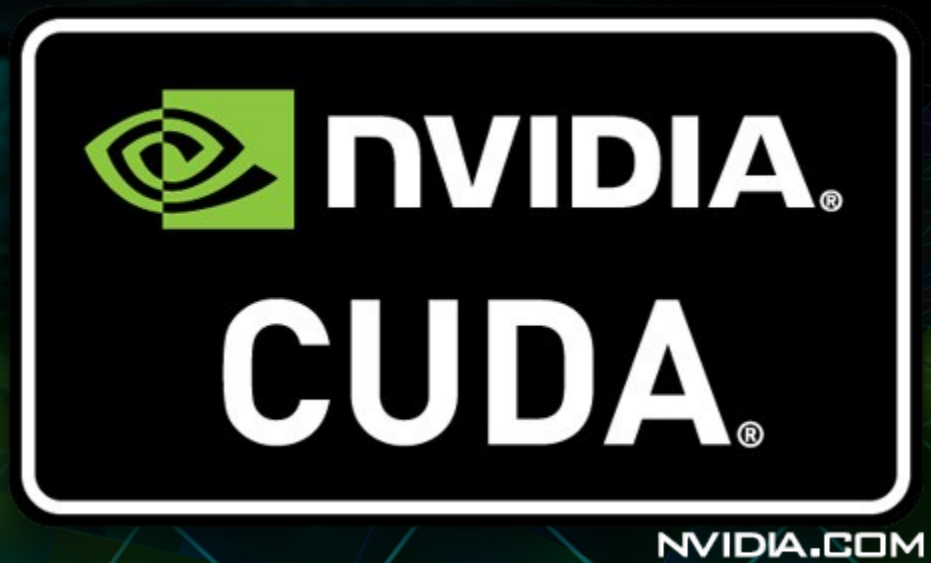


Introduction to utilizing the power of Nvidia GPUs

Agenda

1. Introduction
    - 1.1 The Benefits of using GPUs
    - 1.2 Performance Demo
    - 1.3 Cuda, short overview
2. Programming Model
3. Implementation
    - 3.1 Kernels
    - 3.2 Thread hierarchy
    - 3.3 Heterogeneous programming
4. Usage
    - 4.1 Installation
    - 4.2 IDEs & Tools
5. Demo

Introduction

The Benefits of using GPUs

Performance Demo

Results:

- Matrix size: 3440 x 3440
- C++: 22.7181 seconds
- Cuda: 0.563215 seconds

=> 3934% Performance increase!

Cuda – Short Overview

- Introduced in 2006
- General purpose parallel computing plattform
- Only available for Nvidia GPUs
- Use Cases:
  - Machine Learning
  - Computational Chemistry
  - Bioinformatics
  - Computational Fluid Dynamics
  - Data Science
  - Weather and Climate

GPU Computing Applications

Libraries and Middleware

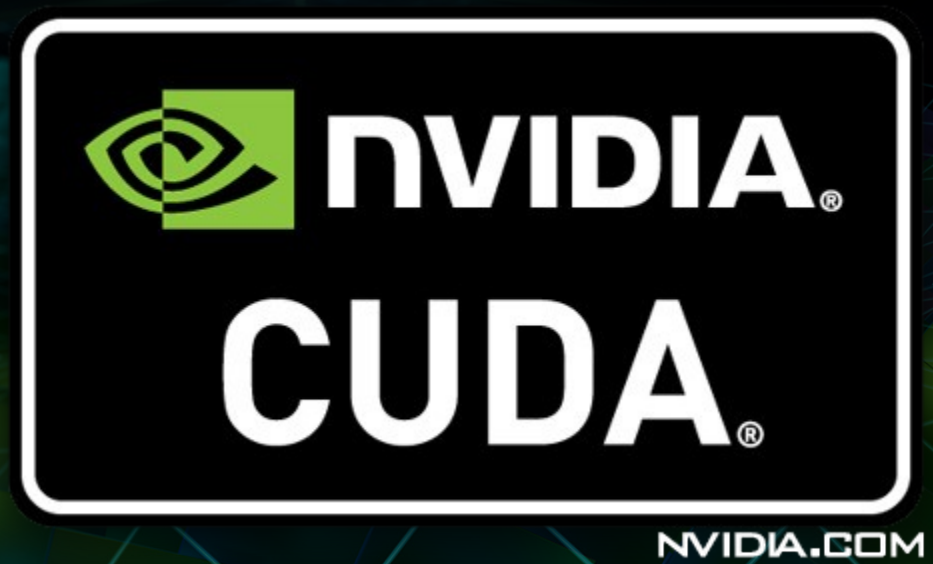
cuDNN TensorRT	cuFFT cuBLAS cuRAND cuSPARSE	CULA MAGMA	Thrust NPP	VSIPL SVM OpenCurrent	PhysX OptiX iRay	MATLAB Mathematica

Programming Languages

C	C++	Fortran	Java Python Wrappers	DirectCompute	Directives (e.g. OpenACC)

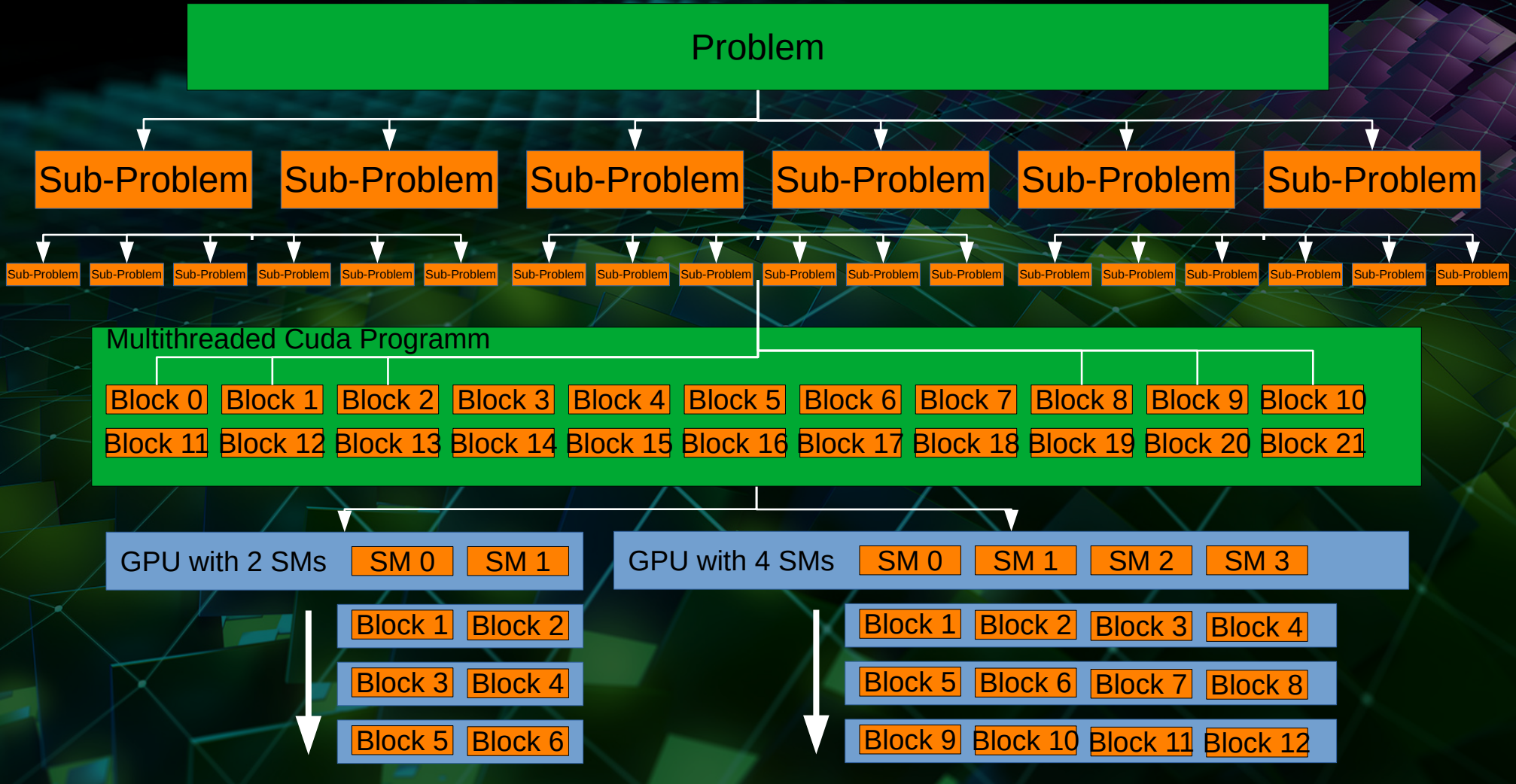
CUDA-Enabled NVIDIA GPUs

NVIDIA Ampere Architecture (compute capabilities 8.x)				Tesla A Series
NVIDIA Turing Architecture (compute capabilities 7.x)		GeForce 2000 Series	Quadro RTX Series	Tesla T Series
NVIDIA Volta Architecture (compute capabilities 7.x)	DRIVE/JETSON AGX Xavier		Quadro GV Series	Tesla V Series
NVIDIA Pascal Architecture (compute capabilities 6.x)	Tegra X2	GeForce 1000 Series	Quadro P Series	Tesla P Series
				



Programming model

Programming model



Implementation

Kernels

- Kernel represents smallest possible sub-Problem
- C++ function
- Gets executed N times by N ***different*** Cuda threads
- Defined by the __global__ declaration specifier:

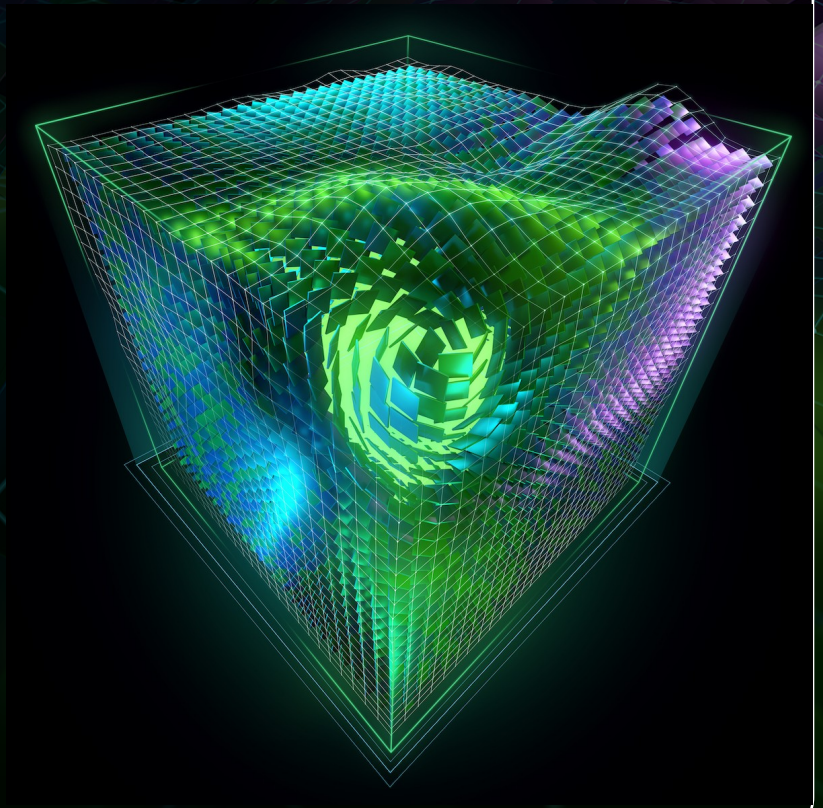
```
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```

- Execution configuration syntax:

```
// Kernel invocation with N threads
VecAdd<<<1, N>>>(A, B, C);
```

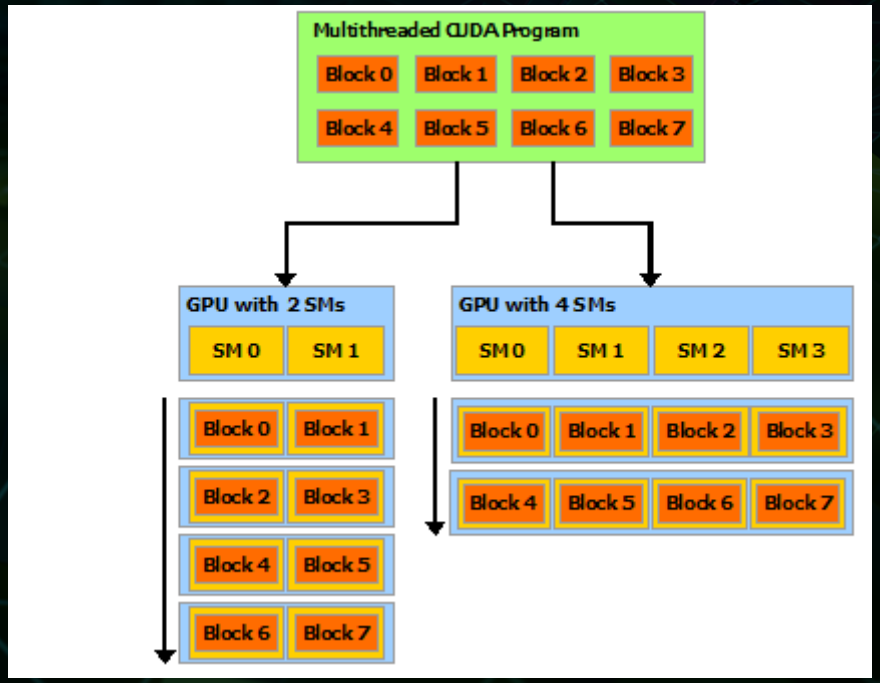
Thread hierarchy

- Each thread gets a ***unique*** threadID
  - accessed by build-in variables:
    int I = threadIdx.x
- ThreadID is a 3 component Vector
  - → Thread identification using up to 3-dimensional indices
  - → Threads are arranged in blocks of threads
    - → Thread Blocks



Thread hierarchy

- Number of Threads per Block is set by user



➔ Calculate blockSize and number of Blocks!

Thread hierarchy

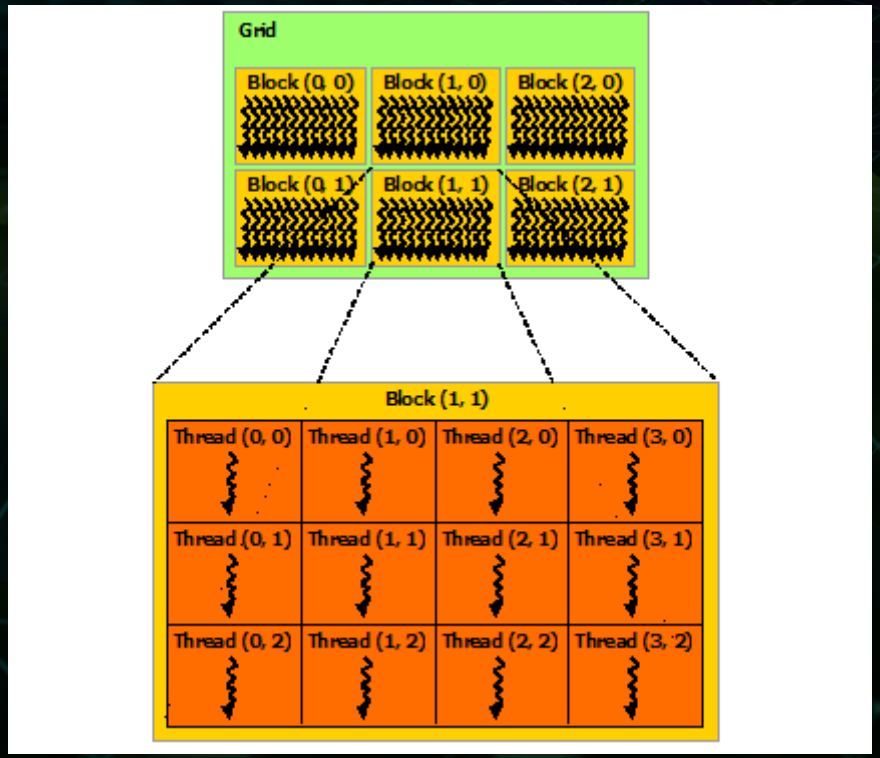
RTX 2070super:

- 40 Streaming Multiprocessors (Sms)
- 2560 Cuda Cores / Shading Units

- BlockSize = CudaCores / Sms
  blockSize = 2560 / 40 = 64
- NumberOfBlocks = (N + blockSize – 1) / blockSize
  N = total number of Threads needed to solve the Problem
  numBlocks = (10000 + 64 – 1) / 64 ≈ 158

Thread hierarchy

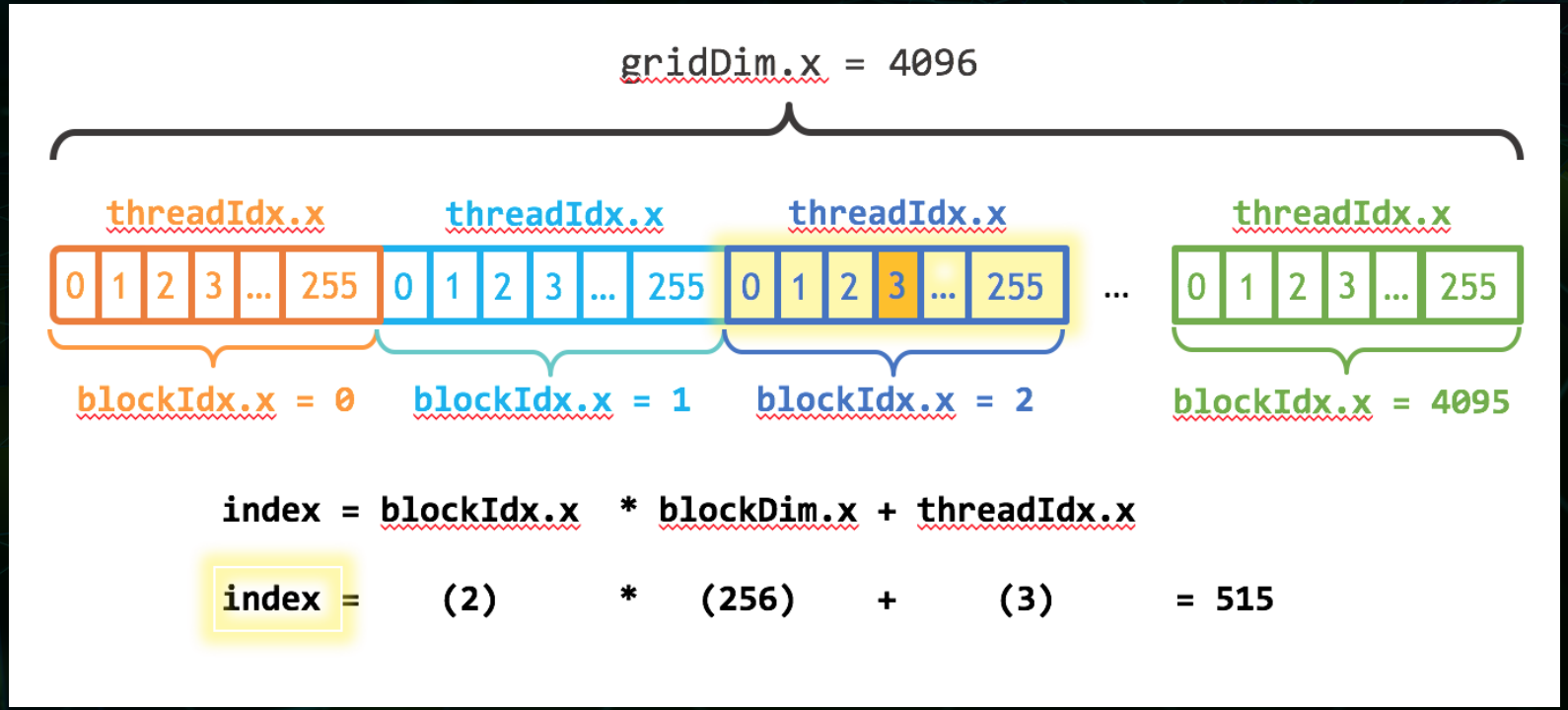
- Blocks get organized in a Grid



- Configuration specified by the Execution Configuration Syntax

Thread hierarchy

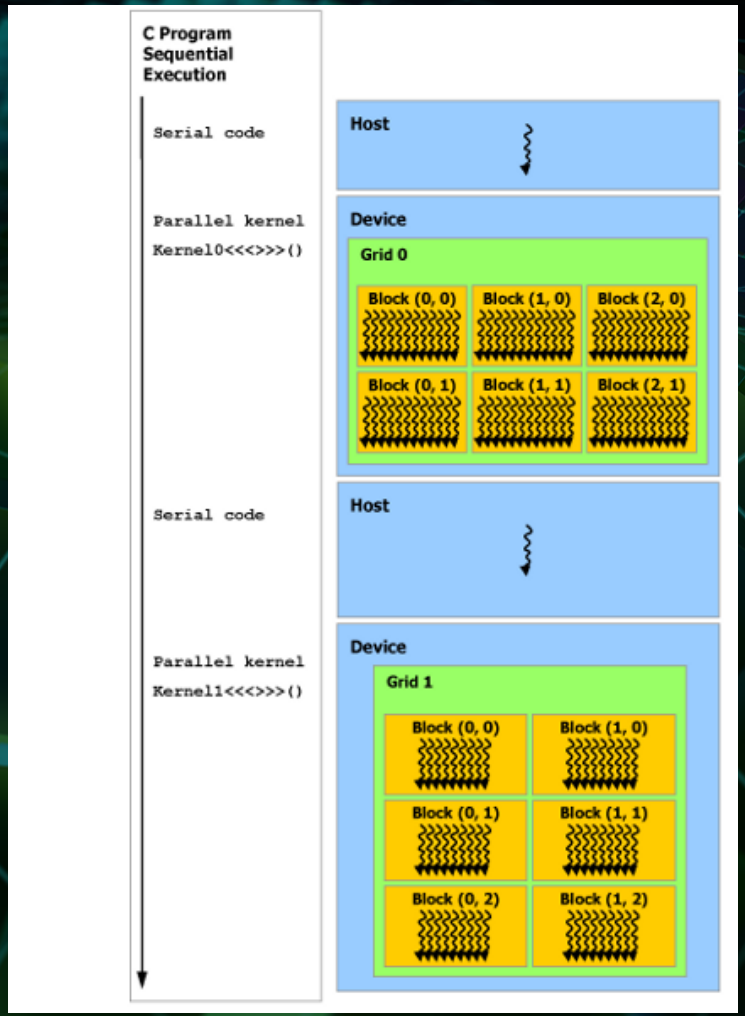
- BlockDim = number of threads in current block
- BlockIdx = ID of current block
- GridDim = number of threads in currend Grid

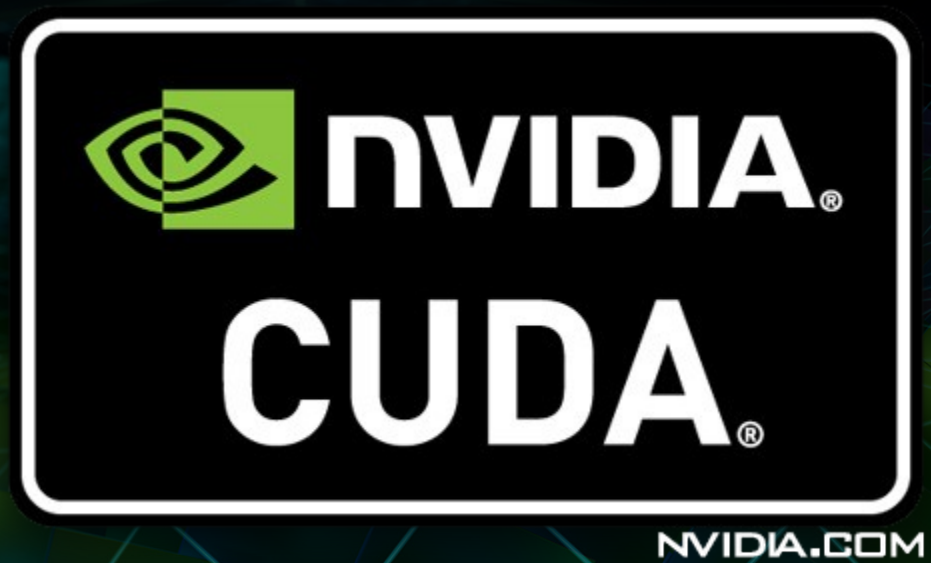


Heterogeneous Programming

- Cuda Programming Model assumes:
  - Cuda Threads get executed on a physically seperate device
  - Both devices maintain their own seperate memory Spaces
    - → Host memory & device memory
- → Memory needs to be allocated on both Devices
- → Data needs to be transferred between both devices

Heterogeneous Programming





Usage

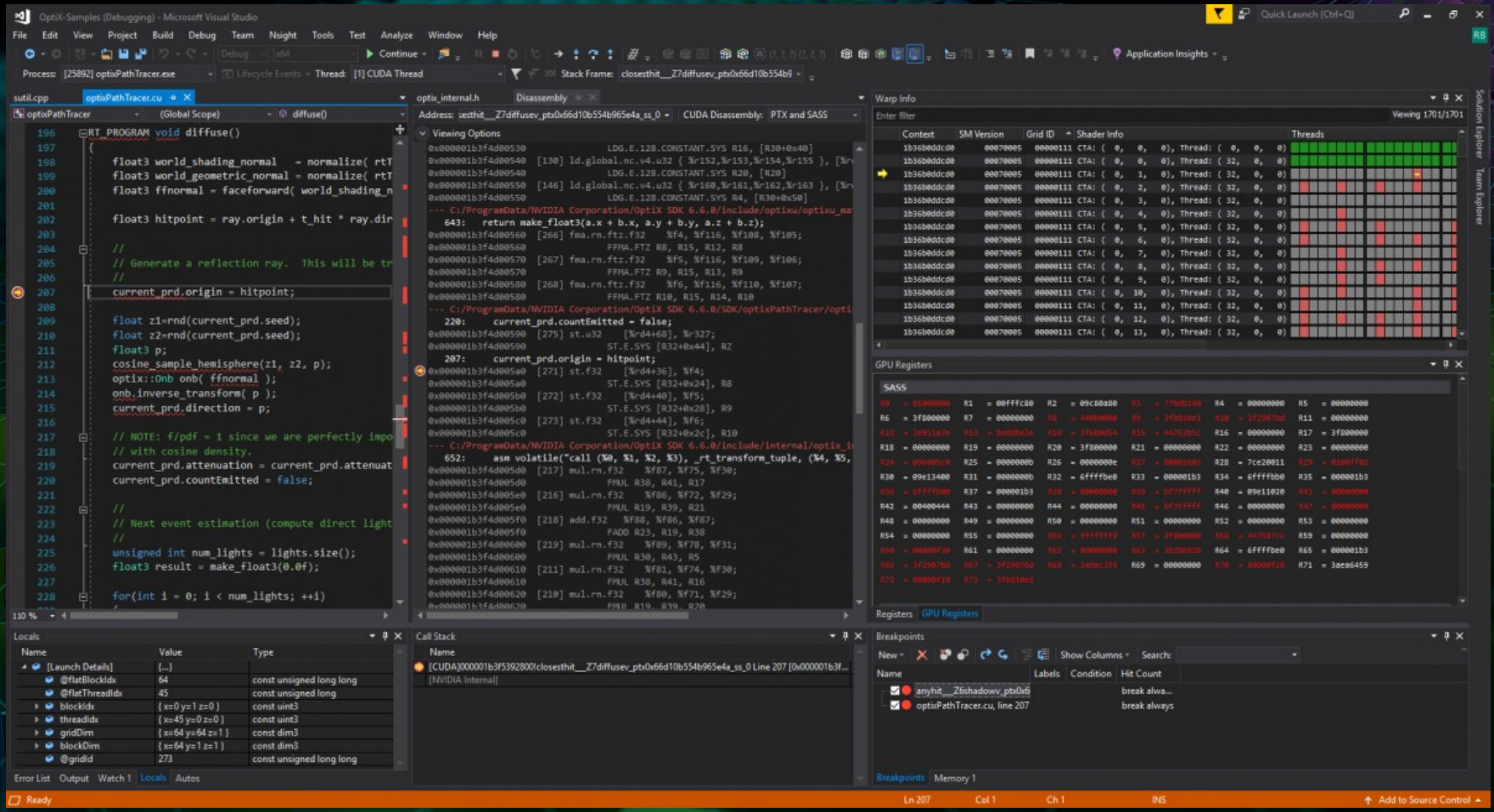
Installation

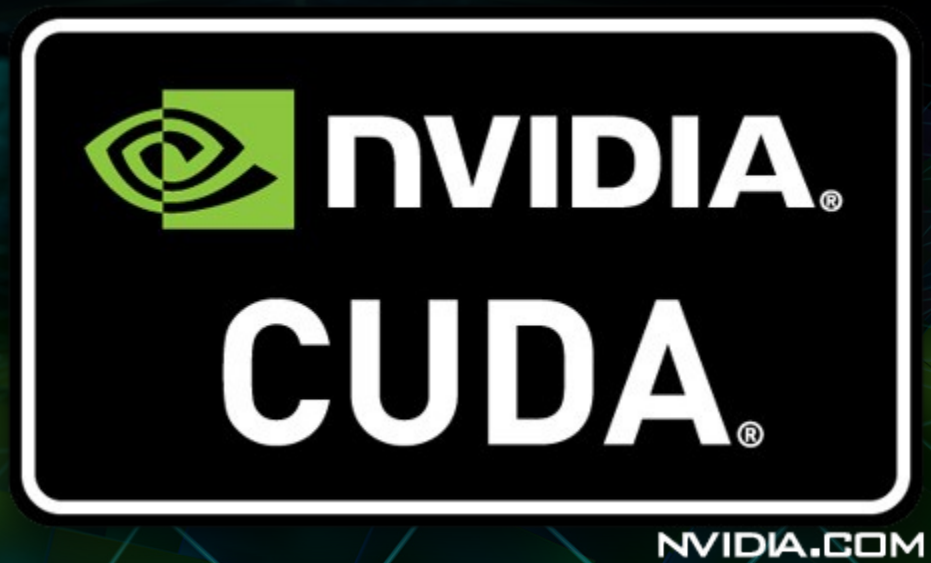
- Download & Install the CUDA TOOLKIT
  https://developer.nvidia.com/cuda-downloads
- Prerequirements:
  - Cuda Capable GPU:
    https://developer.nvidia.com/cuda-gpus
  - You can code and compile Cuda Kernels without a Nvidia GPU
  - Cuda is also available in cloud services:
    - Amazon AWS
    - Microsoft Azure
    - ...

IDEs & Tools

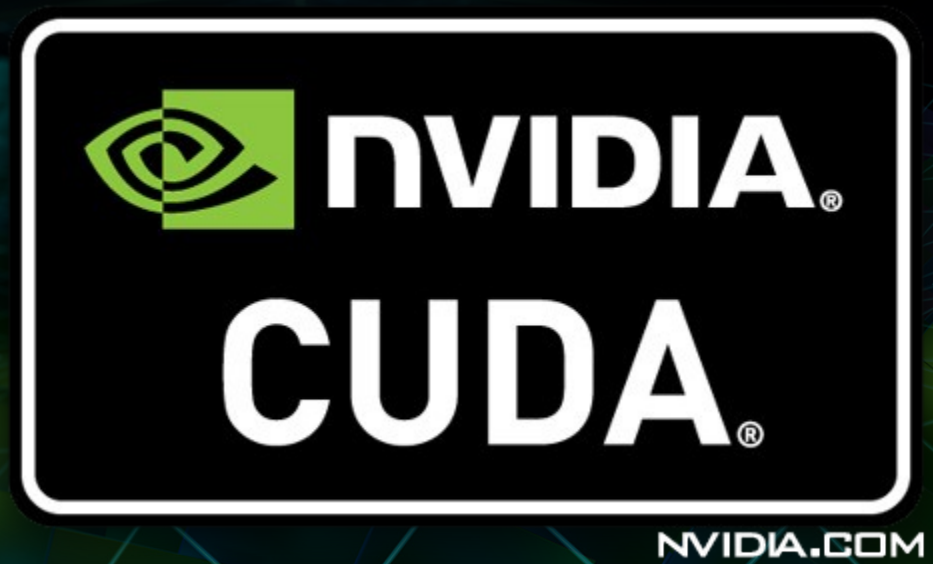
- IDEs:
  - Nsight Studio (eclipse based)
  - Clion (doesn’t include debugging functionality)
  - Visual Studio
- Tools:
  - Nvidia Visual Profiler: Performance analysis for Cuda Kernels

IDEs & Tools


OptiX-Samples (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Team Nsight Tools Test Analyze Window Help
sutil.cpp
optixPathTracer.cu
RT_PROGRAM void diffuse()
float3 world_shading_normal = normalize(rtT
float3 world_geometric_normal = normalize(rtT
float3 ffnormal = faceforward(world_shading_n
float3 hitpoint = ray.origin + t_hit * ray.dir
// Generate a reflection ray. This will be tr
current_prd.origin = hitpoint;
float z1=rnd(current_prd.seed);
float z2=rnd(current_prd.seed);
float3 p;
cosine_sample_hemisphere(z1, z2, p);
optix::Onb onb(ffnormal);
onb.inverse_transform(p);
current_prd.direction = p;
// NOTE: f/pdf = 1 since we are perfectly impo
// with cosine density.
current_prd.attenuation = current_prd.attenuat
current_prd.countEmitted = false;
// Next event estimation (compute direct light
unsigned int num_lights = lights.size();
float3 result = make_float3(0.0f);
for(int i = 0; i < num_lights; ++i)
optix_internal.h
Disassembly
Warp Info
GPU Registers
SASS
Locals
Call Stack
Breakpoints
Ready



Demo



More Info

More Info

- Nvidia Cuda Zone:
  https://developer.nvidia.com/cuda-zone
- Nvidia developer Programm
- Programming Guide:
  https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html