Технології графічного процесінгу

Лекція 2: Вступ до CUDA C

Кочура Юрій Петрович
iuriy.kochura@gmail.com
@y_kochura

Огляд

Сьогоднішні виклики обчислень

- Навички виконання обчислень є важливими для вивчення практично усіх дисциплін
- Наука про дані та машинне навчання стають основними навичками в більшості STEM
- Практично всі процесори багатоядерні, від мікроконтролерів до суперкомп'ютерів
- Бізнес та наукові відкриття потребують ШІ та прискорених обчислень

The future of computing: a conversation with John Hennessy (Google I/O '18)

Пристрої

Device type	Device name	Transistor count	Date of introduction	Designer(s)	MOS process	Area	Transistor density, tr./mm2
Deep learning engine / IPU[g]	Colossus GC2	23,600,000,000	2018	Graphcore	16 nm	~800 mm2	29,500,000
Deep learning engine / IPU	Wafer Scale Engine	1,200,000,000,000	2019	Cerebras	16 nm	46,225 mm2	25,960,000
Deep learning engine / IPU	Wafer Scale Engine 2	2,600,000,000,000	2020	Cerebras	7 nm	46,225 mm2	56,250,000
Network switch	NVLink4 NVSwitch	25,100,000,000	2022	Nvidia	N4 (4 nm)	294 mm2	85,370,000

—

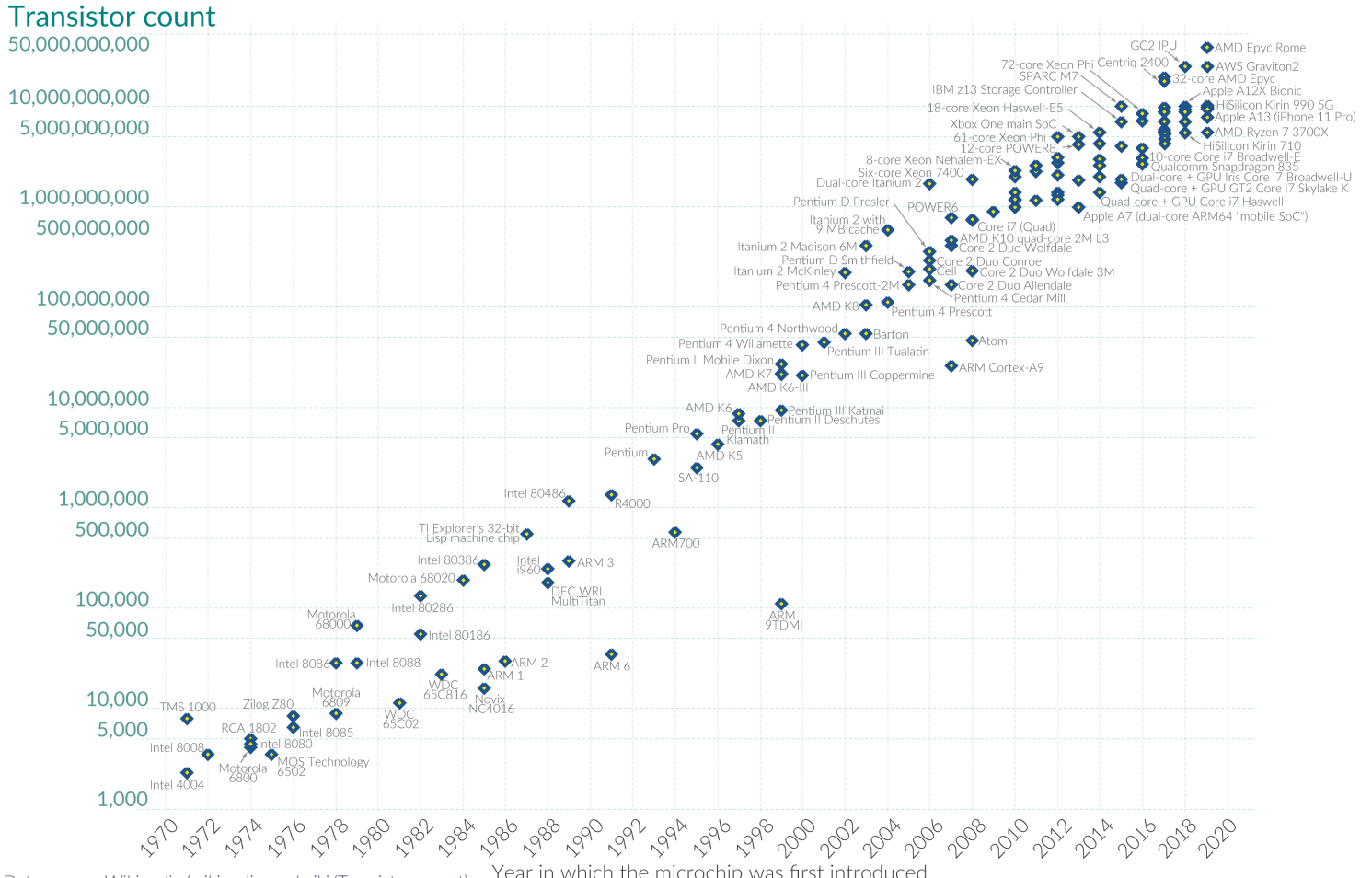
IPU: Intelligence Processing Unit

Джерело слайду: en.wikipedia.org

Moore's Law: The number of transistors on microchips doubles every two years

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years.
This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world's largest problems.

—

Джерело слайду: en.wikipedia.org

Сьогодні

- Масштабованість та портативність у гетерогенних паралельних обчисленнях
- GPU-прискорені vs лише CPU програми
- Способи прискорення виконання програм
- Приклади паралельних обчислень
- Процес компіляції CUDA C/C++
- Ядра: функції GPU
- Ієрархія потоків
- Демо

Масштабованість та портативність у гетерогенних паралельних обчисленнях

Масштабованість

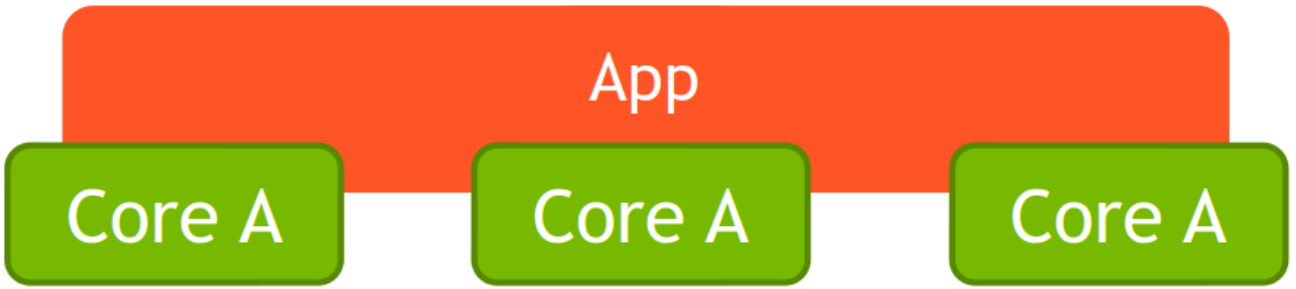
—
Джерело слайду: NVIDIA, DLI

Масштабованість

Той самий додаток ефективно працює на нових поколіннях ядер.

—
Джерело слайду: NVIDIA, DLI

Масштабованість



Той самий додаток ефективно працює на кількох однакових ядрах.

—
Джерело слайду: NVIDIA, DLI

Портативність

Той самий додаток ефективно працює на різних типах ядер.

—
Джерело слайду: NVIDIA, DLI

Портативність

Той самий додаток ефективно працює в системах з різною організацією та інтерфейсами.

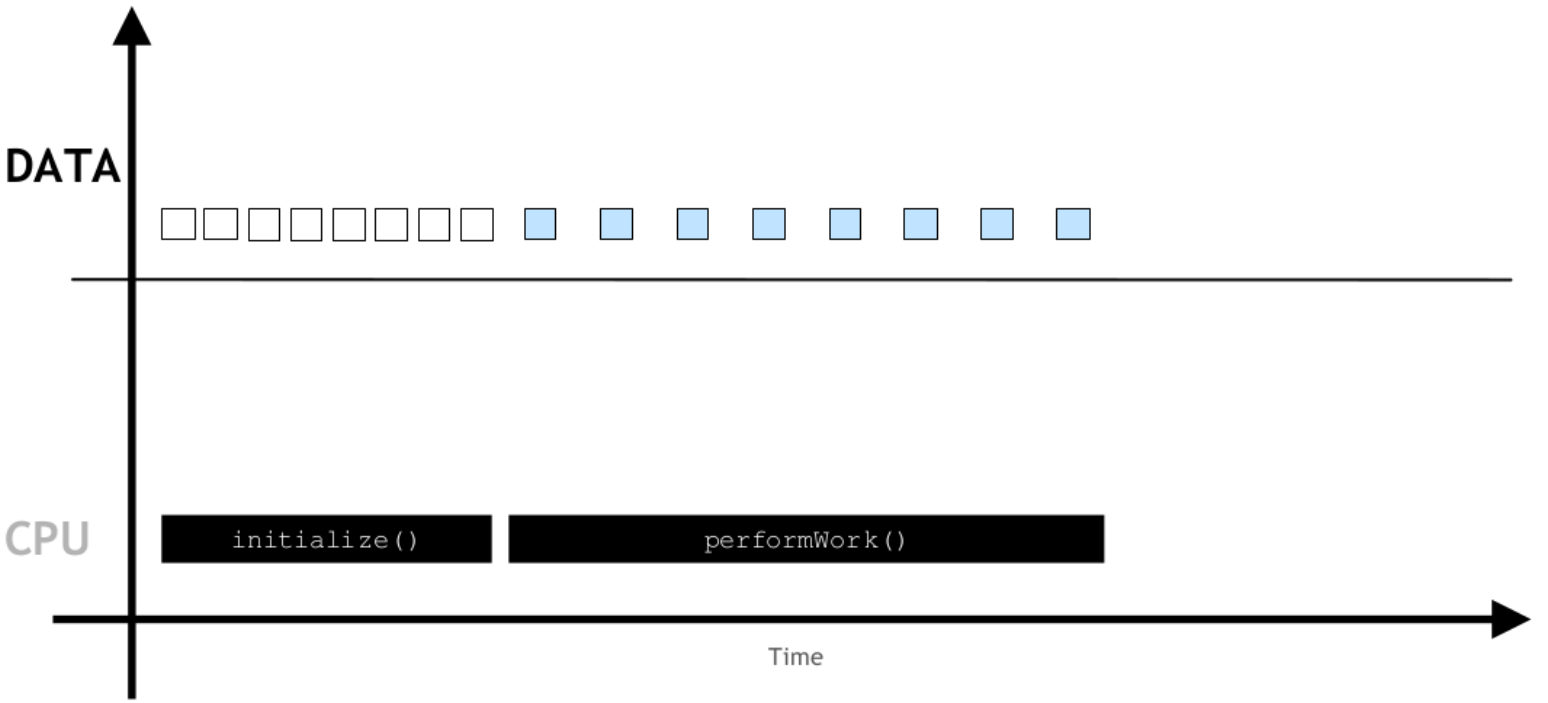
—
Джерело слайду: NVIDIA, DLI

GPU-прискорені
vs
лише CPU програми

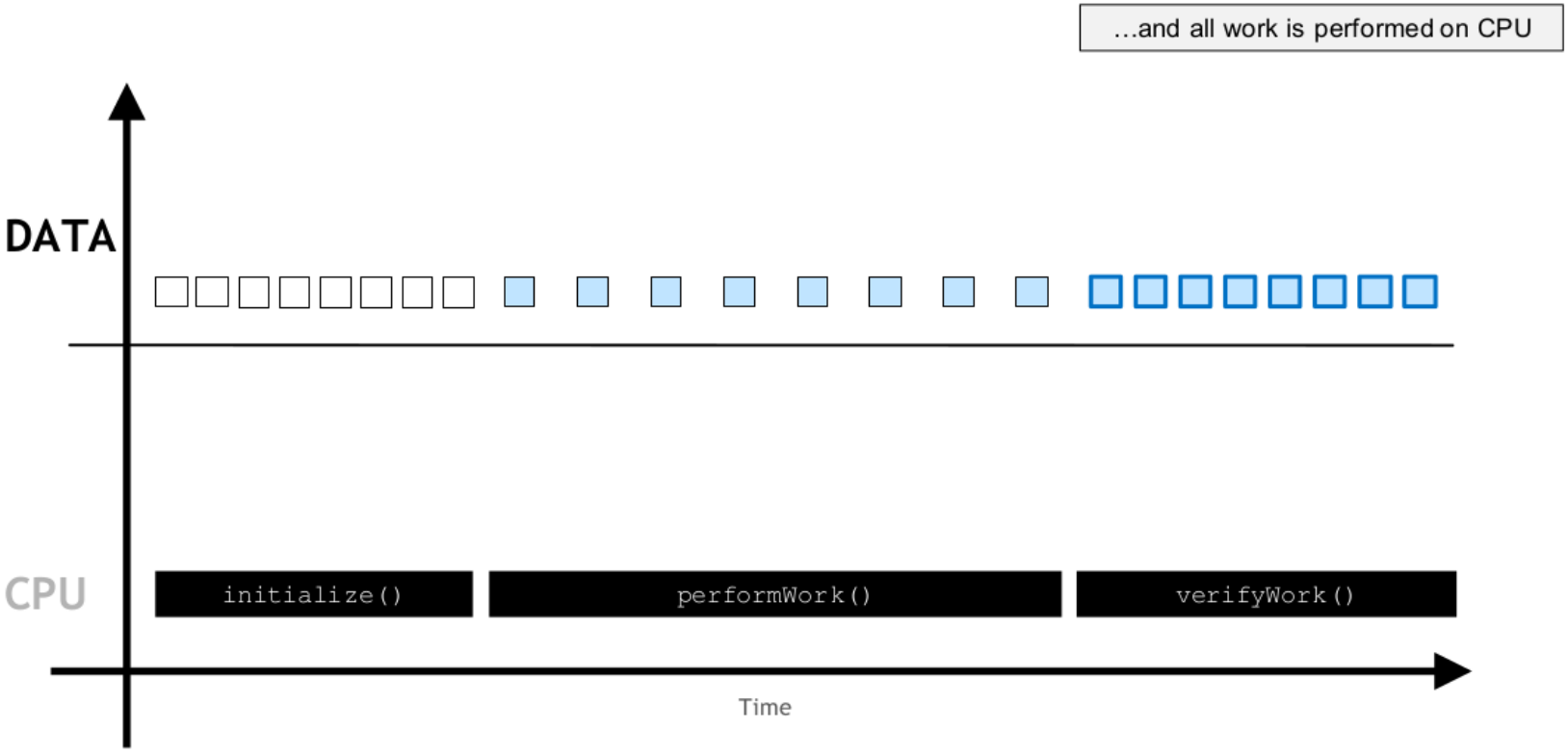
In **CPU-only applications** data is allocated on CPU

—
Джерело слайду: NVIDIA, DLI

…and all work is performed on CPU

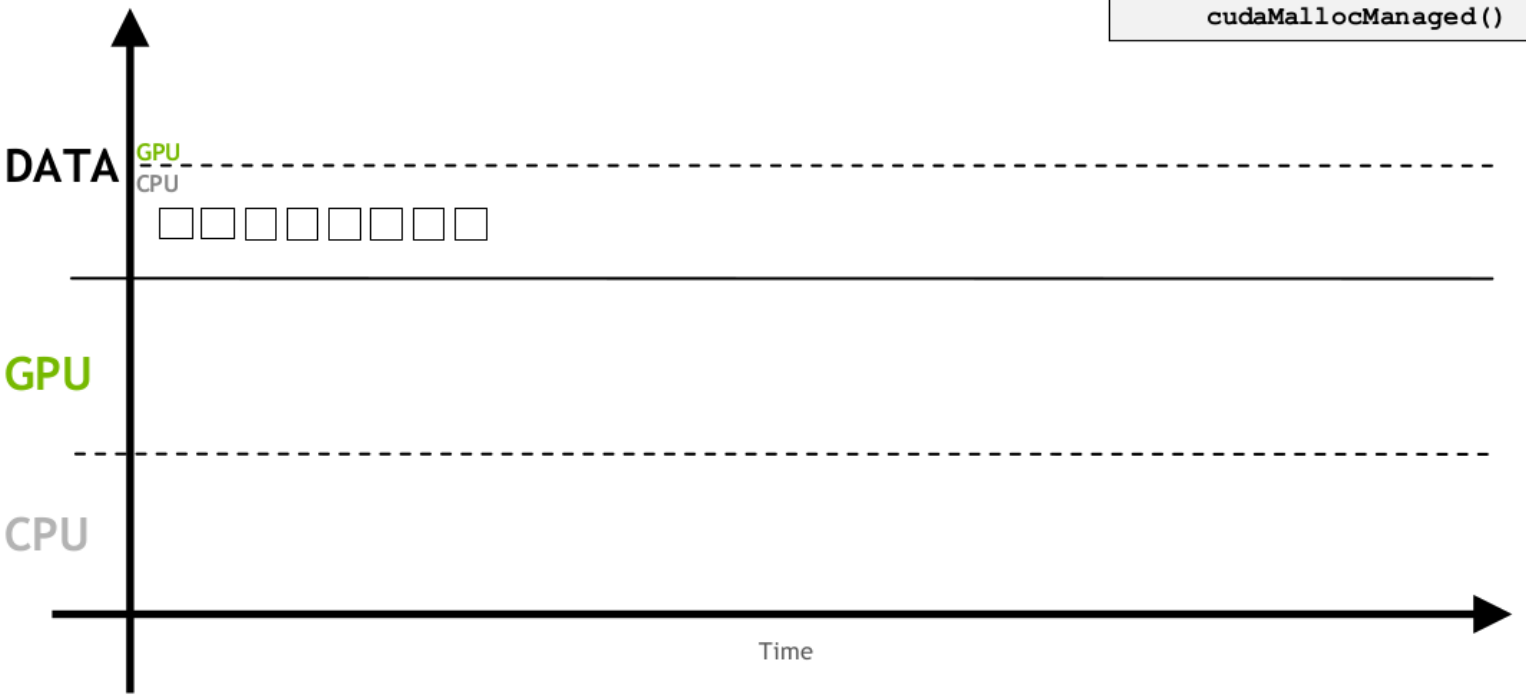


—
Джерело слайду: NVIDIA, DLI



—
Джерело слайду: NVIDIA, DLI

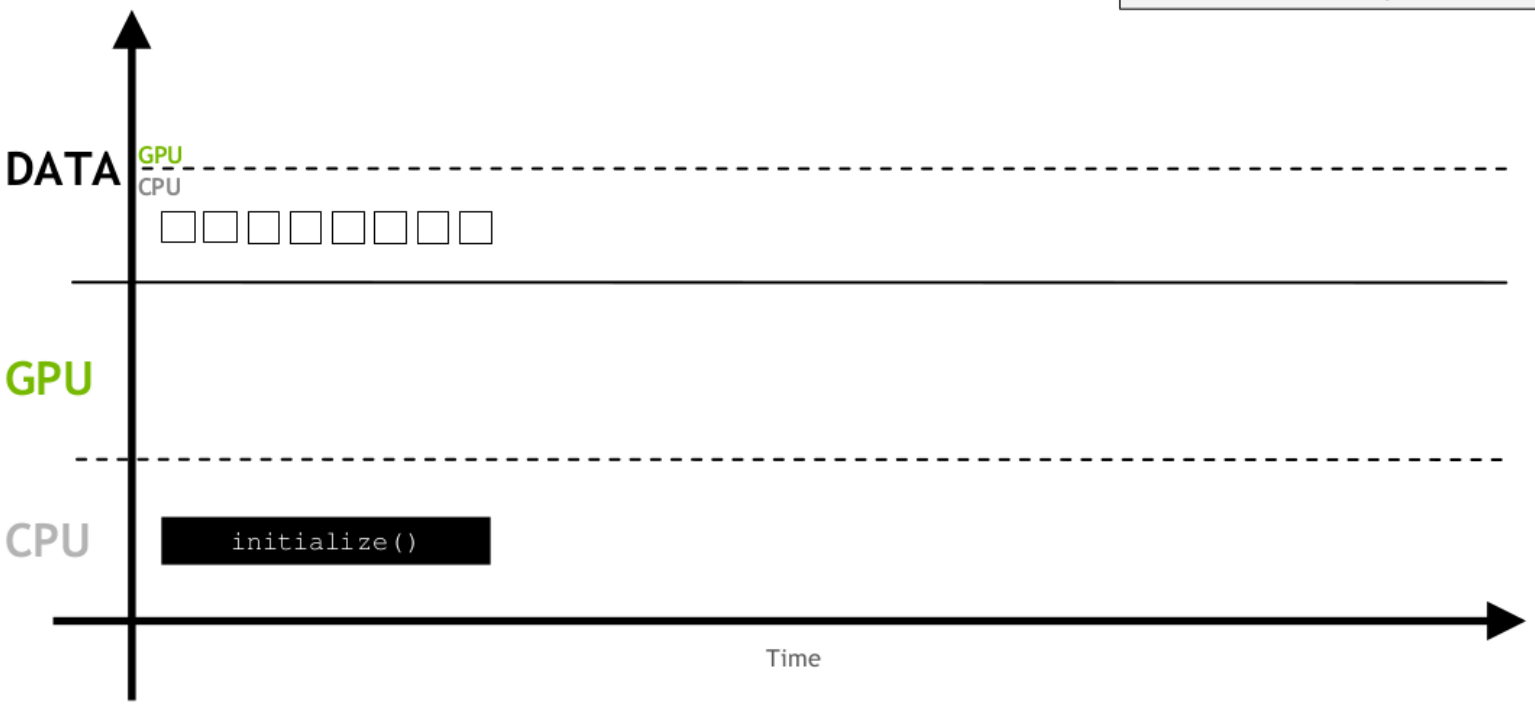
In **accelerated applications** data is allocated with **`cudaMallocManaged()`**



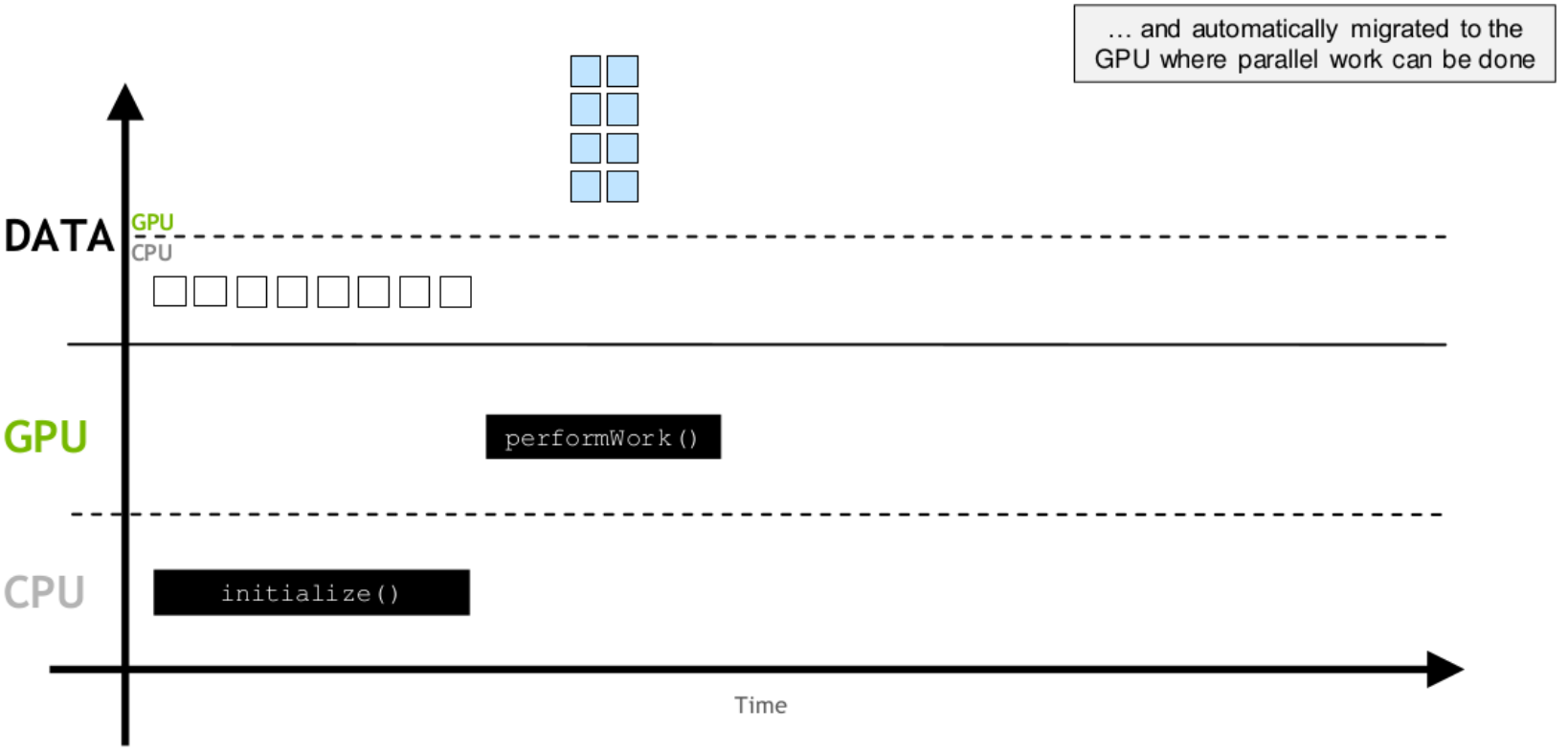
—

Джерело слайду: NVIDIA, DLI

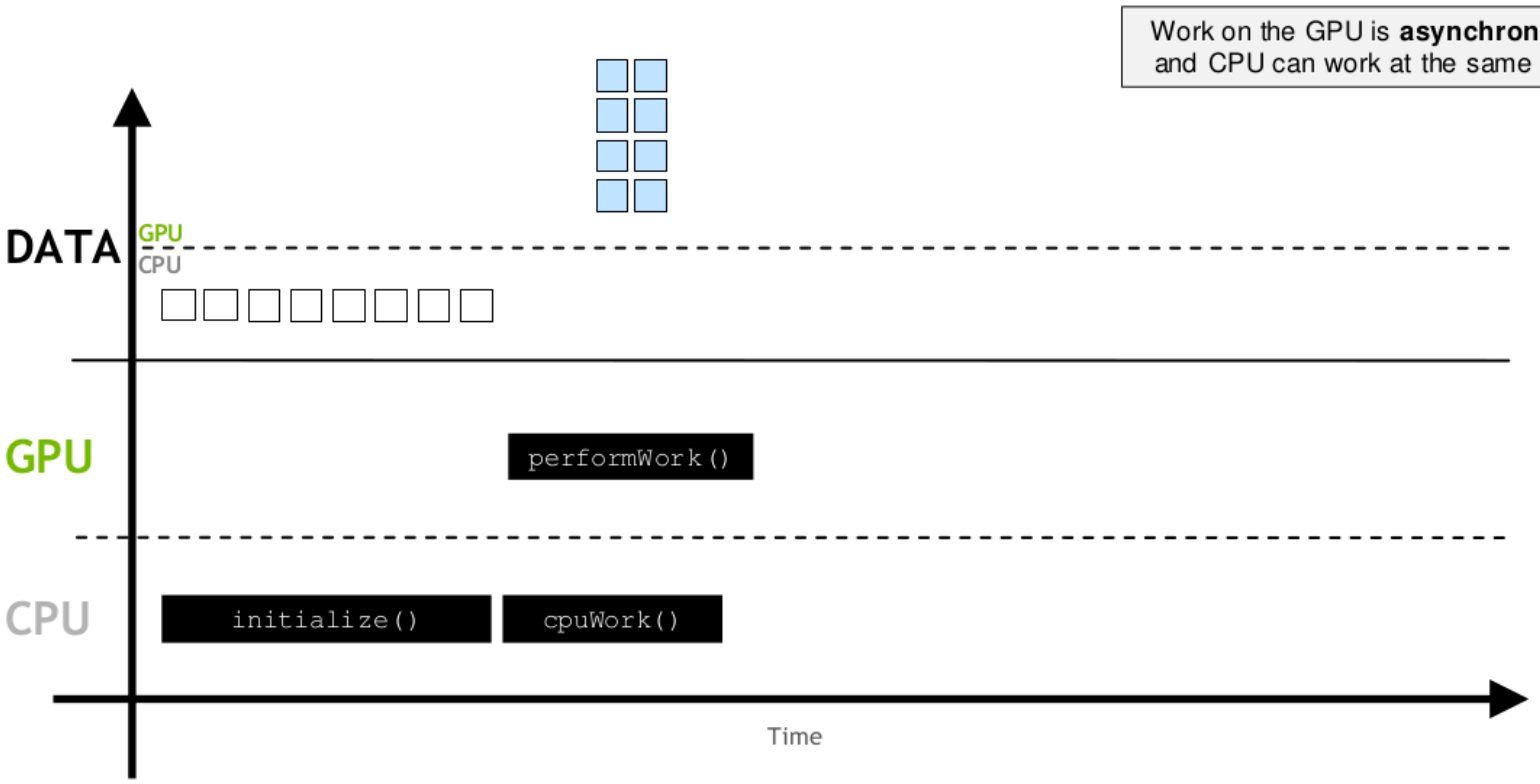
… where it can be accessed and worked on by the CPU



—
Джерело слайду: NVIDIA, DLI

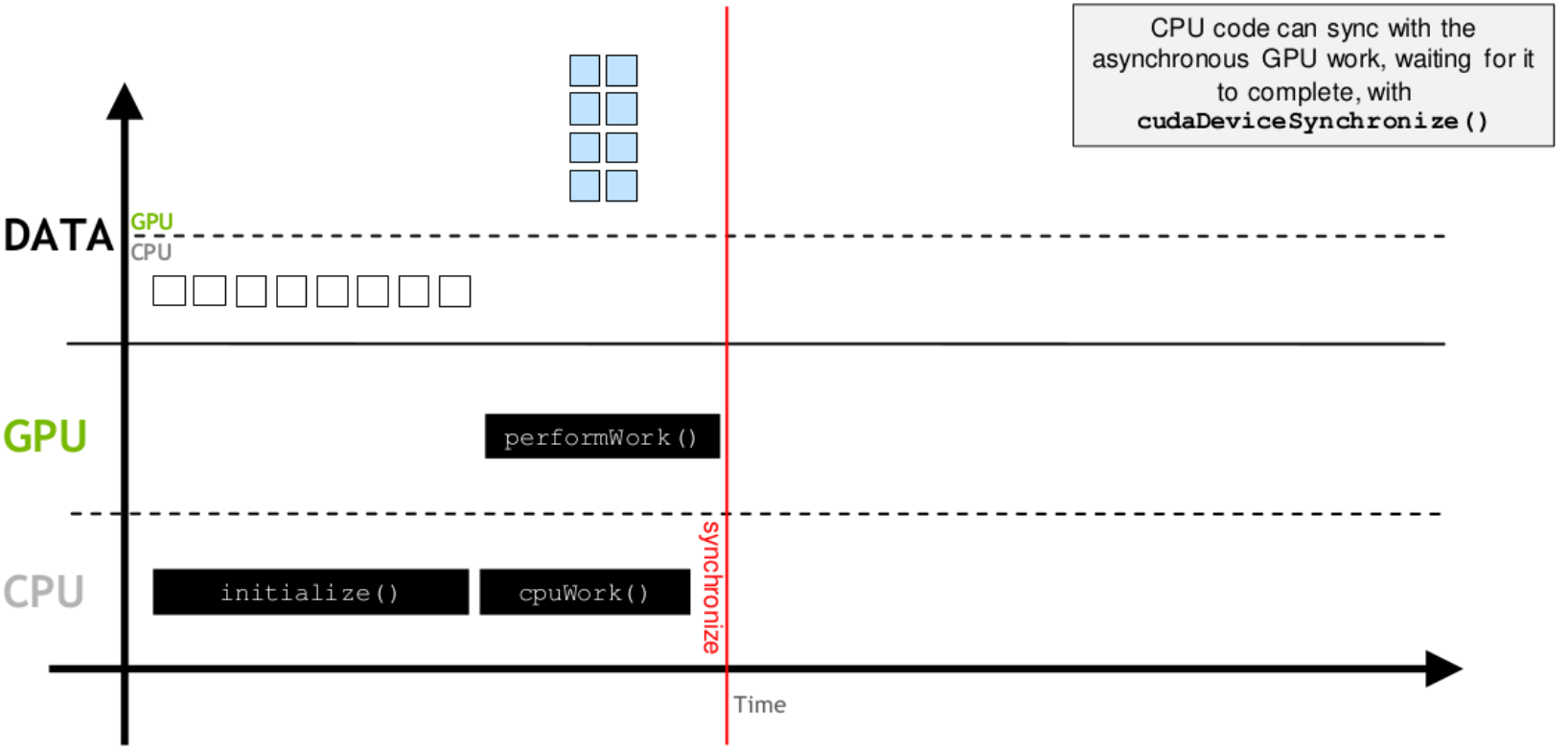


—
Джерело слайду: NVIDIA, DLI

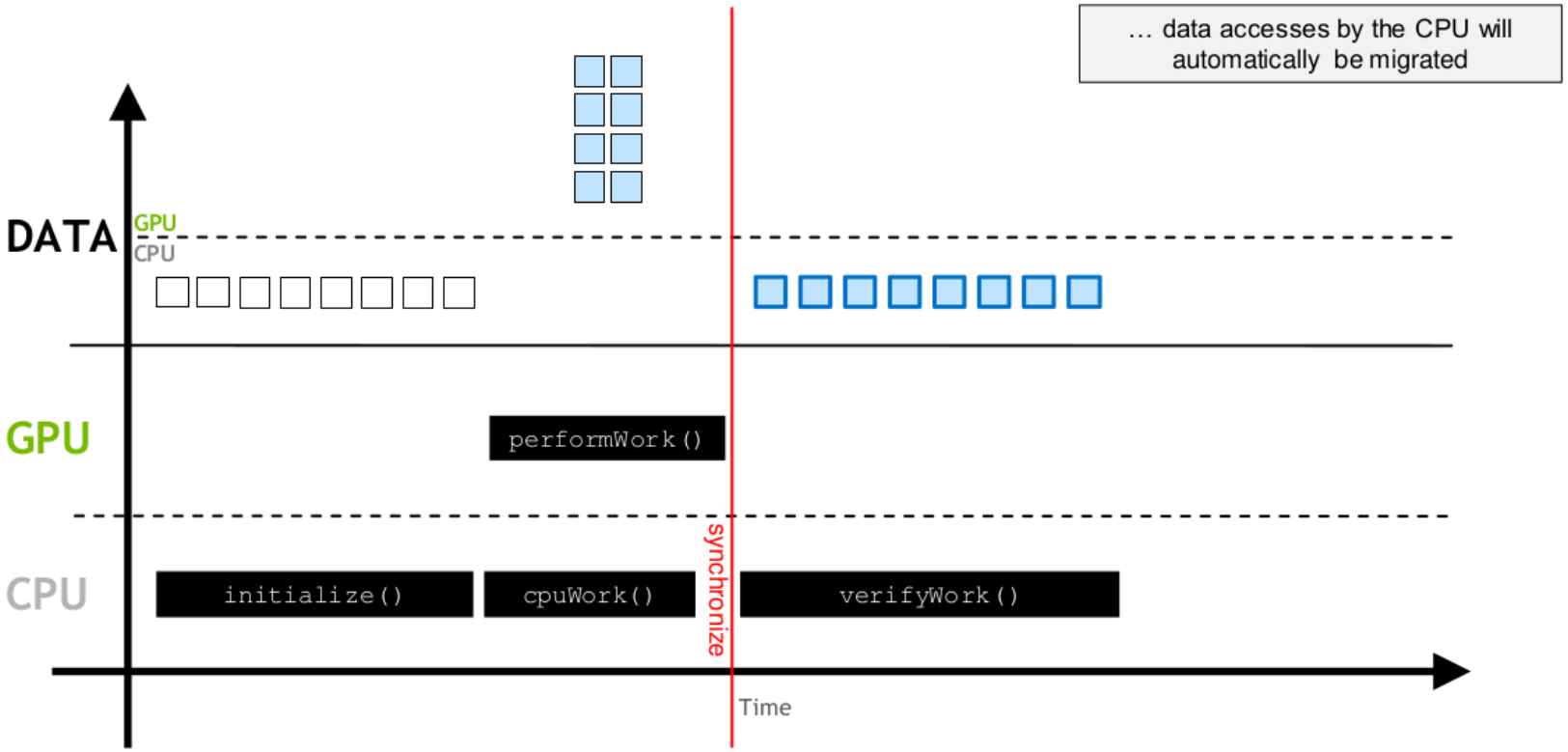


—

Джерело слайду: NVIDIA, DLI

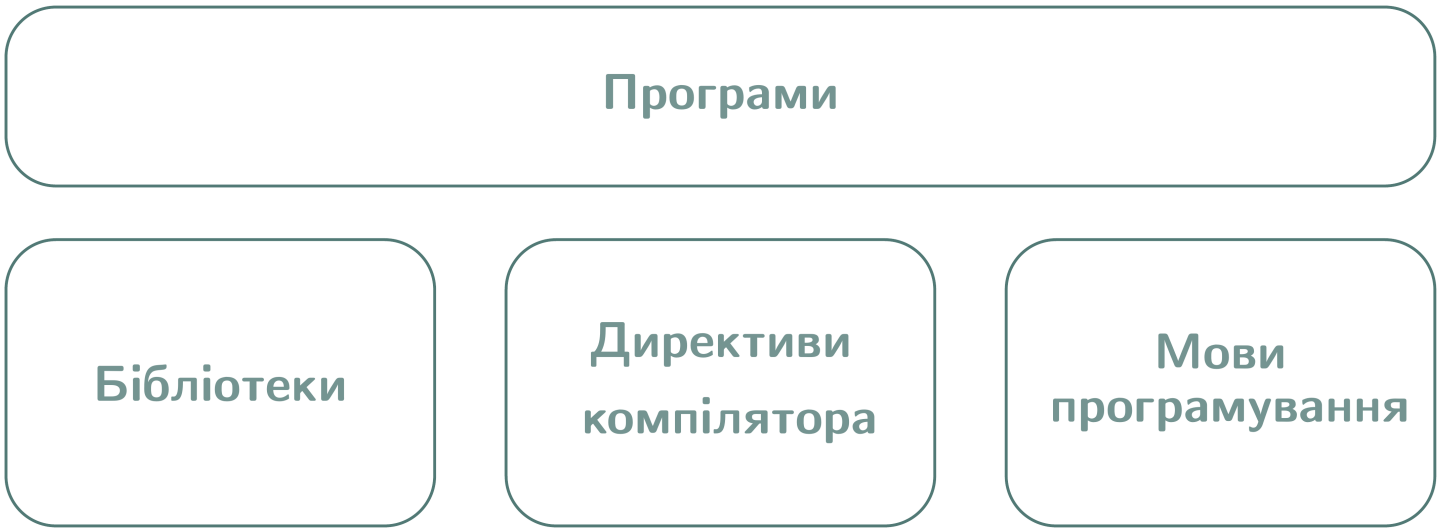


Джерело слайду: NVIDIA, DLI



Джерело слайду: NVIDIA, DLI

Способи прискорення виконання програм



Прості у використанні
Висока продуктивність

Прості у використанні
Портативний код

Висока продуктивність
Висока гнучкість

Бібліотеки

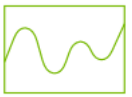
Бібліотеки: проста, високо-якісне прискорення

- **Простота використання:** використання бібліотек дозволяє прискорювати обчислень на GPU без поглиблених знань програмування GPU
- Багато бібліотек для прискорення обчислень на GPU дотримуються стандартних API, що дозволяє прискорювати роботу з мінімальними змінами коду
- **Якість:** бібліотеки пропонують високоякісні реалізації функцій, які зустрічаються у великій кількості додатків

Математичні бібліотеки

cuBLAS

GPU-accelerated basic linear algebra (BLAS) library



cuFFT

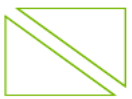
GPU-accelerated library for Fast Fourier Transforms

CUDA Math Library

GPU-accelerated standard mathematical function library

cuRAND

GPU-accelerated random number generation (RNG)



cuSOLVER

GPU-accelerated dense and sparse direct solvers

cuSPARSE

GPU-accelerated BLAS for sparse matrices

cuTENSOR

GPU-accelerated tensor linear algebra library

AmgX

GPU-accelerated linear solvers for simulations and implicit unstructured methods

—

Джерело слайду: GPU-Accelerated Libraries, NVIDIA

Бібліотеки паралельних алгоритмів

Thrust

Бібліотека паралельних алгоритмів і структур даних C++ з GPU

—

Джерело слайду: GPU-Accelerated Libraries, NVIDIA

Бібліотеки візуальної обробки

nvJPEG

High performance GPU-accelerated library for JPEG decoding

NVIDIA Performance Primitives

Provides GPU-accelerated image, video, and signal processing functions

NVIDIA Video Codec SDK

A complete set of APIs, samples, and documentation for hardware-accelerated video encode and decode on Windows and Linux

NVIDIA Optical Flow SDK

Exposes the latest hardware capability of NVIDIA Turing™ GPUs dedicated to computing the relative motion of pixels between images

—

Джерело слайду: GPU-Accelerated Libraries, NVIDIA

Бібліотеки комунікації

NVSHMEM

OpenSHMEM standard for GPU memory, with extensions for improved performance on GPUs.

NCCL

Open-source library for fast multi-GPU, multi-node communications that maximizes bandwidth while maintaining low latency.

—

Джерело слайду: GPU-Accelerated Libraries, NVIDIA

Бібліотеки глибинного навчання

NVIDIA cuDNN

GPU-accelerated library of primitives for deep neural networks

NVIDIA TensorRT™

High-performance deep learning inference optimizer and runtime for production deployment

NVIDIA Riva

Platform for developing engaging and contextual AI-powered conversation apps

NVIDIA DeepStream SDK

Real-time streaming analytics toolkit for AI-based video understanding and multi-sensor processing

NVIDIA DALI

Portable, open-source library for decoding and augmenting images and videos to accelerate deep learning applications

—

Джерело слайду: GPU-Accelerated Libraries, NVIDIA

Директиви компілятора

Директиви компілятора: проста, портативне прискорення

- **Простота використання:** Компілятор подбає про деталі керування паралелізмом і переміщенням даних
- **Портативність:** Код є загальним, не специфічним для будь-якого типу апаратного забезпечення та може бути розгорнутий на кількох мовах
- **Невизначеність:** Продуктивність коду може відрізнятися в різних версіях компілятора

OpenACC

- Директиви компілятора для C/C++ та Fortran

Приклад:

```
main()
{
    <serial code>
    #pragma acc kernels
    // automatically runs on GPU
    {
        <parallel code>
    }
}
```

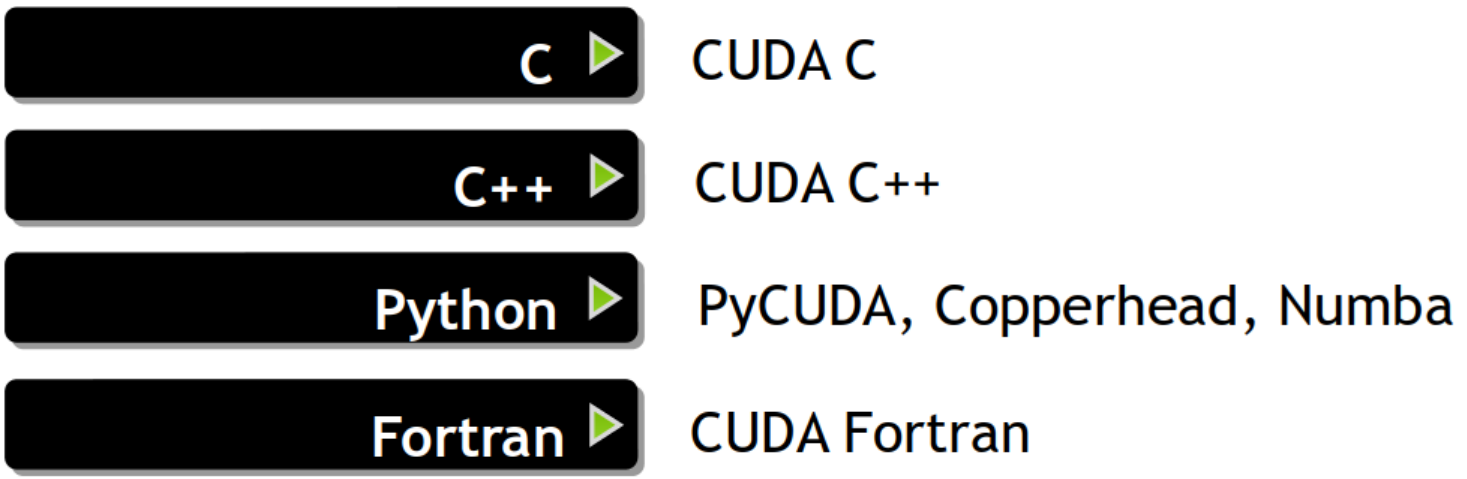
—
Джерело слайду: OpenACC Directives, NVIDIA

Мови програмування

Мови програмування: висока продуктивність та гнучке прискорення

- Продуктивність: Програміст може найкраще контролювати паралельність та переміщення даних
- Гнучкість: Обчислення не потрібно пристосовувати до обмеженого набору бібліотечних шаблонів або типів директив
- Багатослівність: Програмісту часто потрібно виразити більше деталей

Мови програмування **GPU**



GPU Computing Applications

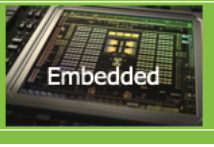
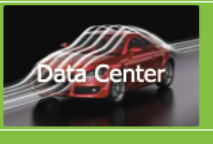
Libraries and Middleware

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
				

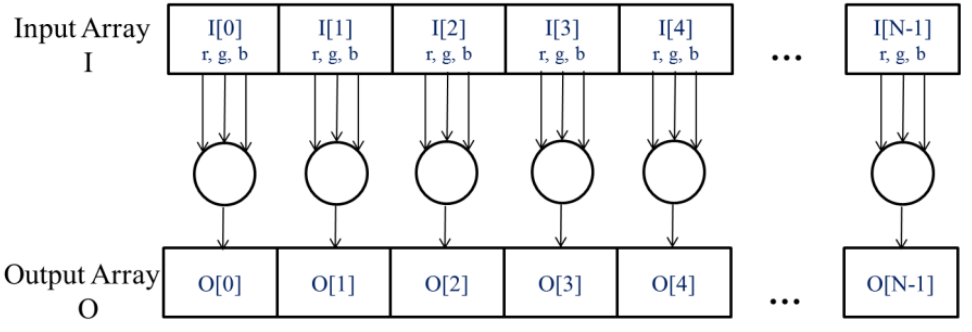
—

Джерело слайду: CUDA C++ Programming Guide, NVIDIA

Приклади паралельних обчислень

Перетворення **RGB** зображення у відтінки сірого

```
for each pixel {
     pixel = gsConvert(pixel)
}
// Every pixel is independent
// of every other pixel
```

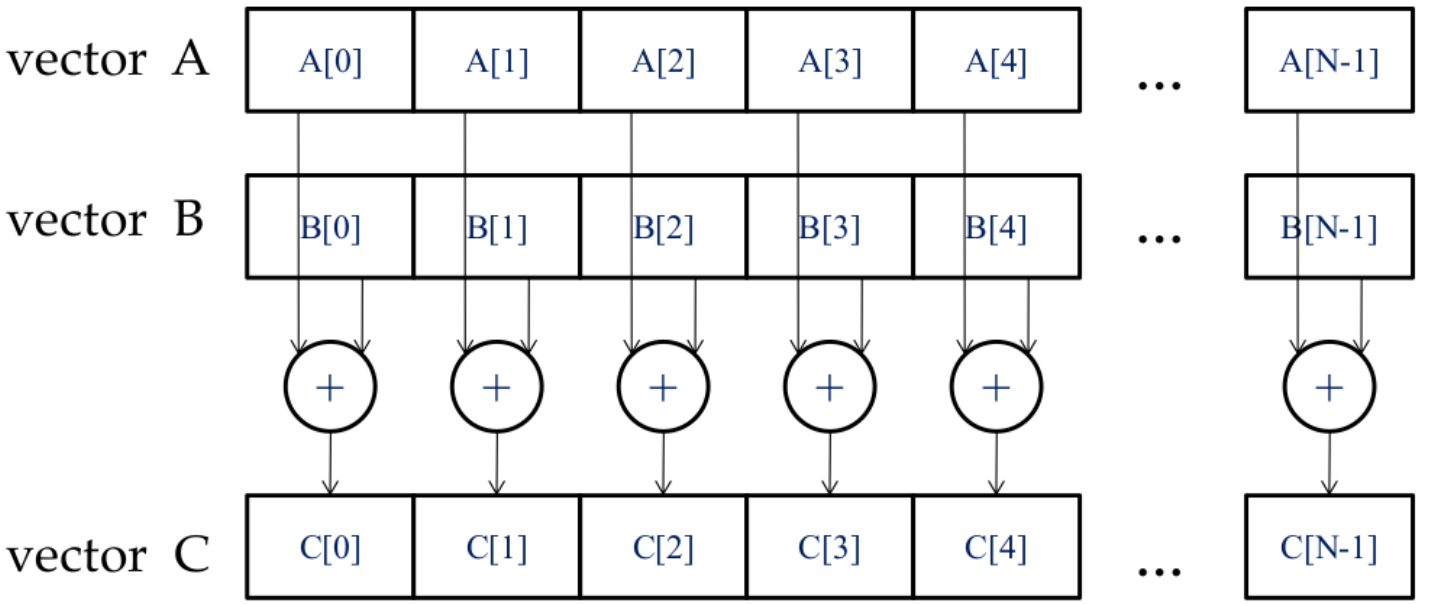


$$O = r \cdot 0.21 + g \cdot 0.72 + b \cdot 0.07$$

—

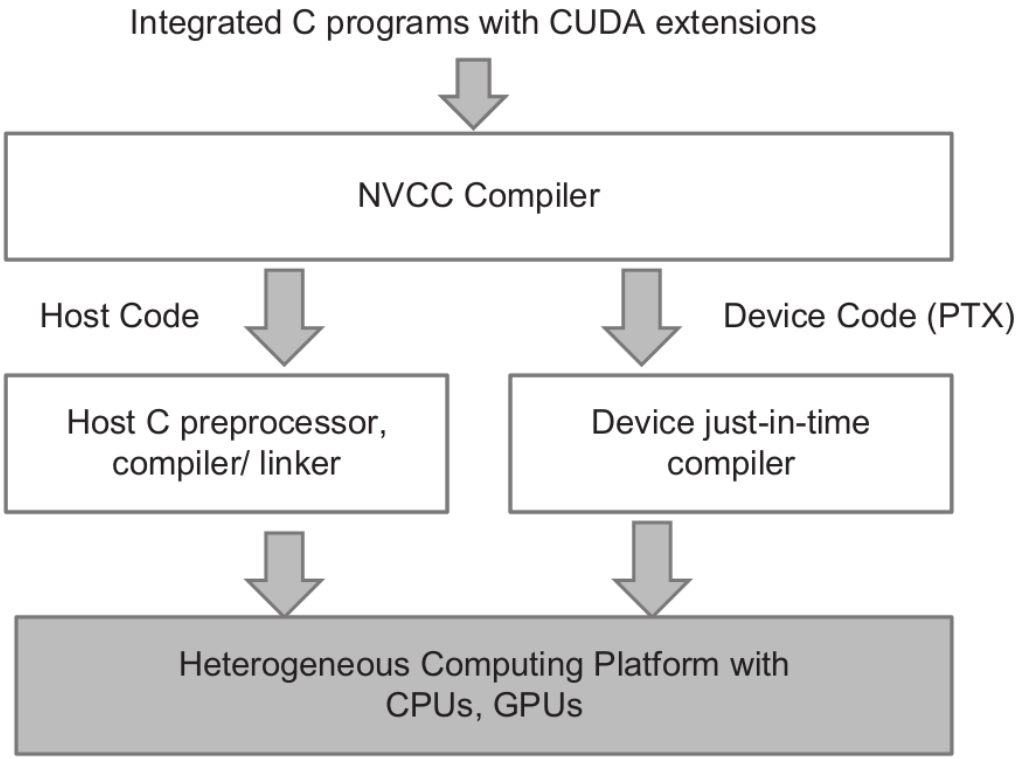
Джерело слайду: Applied Parallel Programming, David Kirk and Wen-mei W. Hwu

Додавання векторів



Процес компіляції **CUDA C/C++**

- Типовий код CUDA C/C++: хост (CPU) + пристрій (GPU)

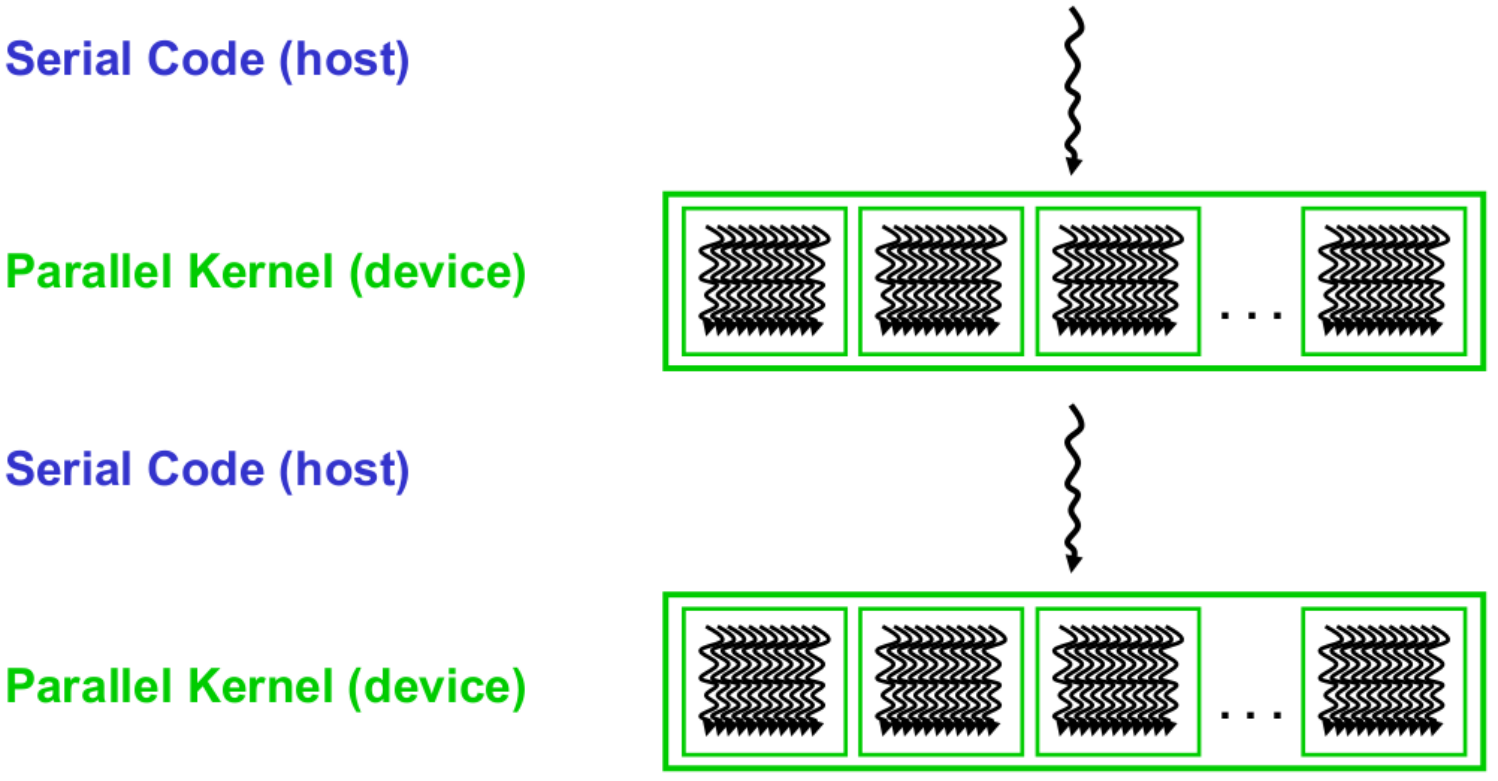


—

Джерело слайду: Programming Massively Parallel Processors, David Kirk and Wen-mei W. Hwu

CUDA/OpenCL — модель виконання

- Послідовні або помірно паралельні частини в коді, що виконуються хостом
- Високо паралельні частини в коді, що виконуються пристроєм: ядро



—

Джерело слайду: Applied Parallel Programming, David Kirk and Wen-mei W. Hwu

Ядра (Kernels)

- Ядро визначається за допомогою специфікатора `__global__`

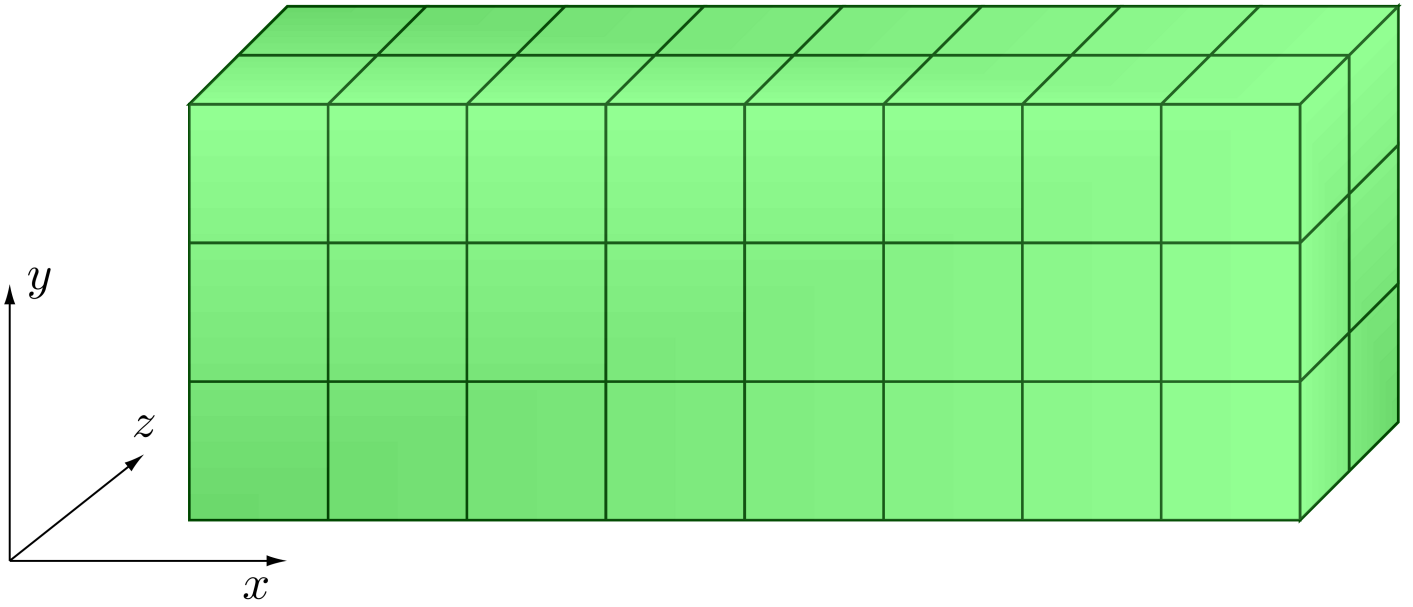
```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
int i = threadIdx.x;
C[i] = A[i] + B[i];
}
int main()
{
...
// Kernel invocation with N threads
VecAdd<<<1, N>>>(A, B, C);
...
}
```

—

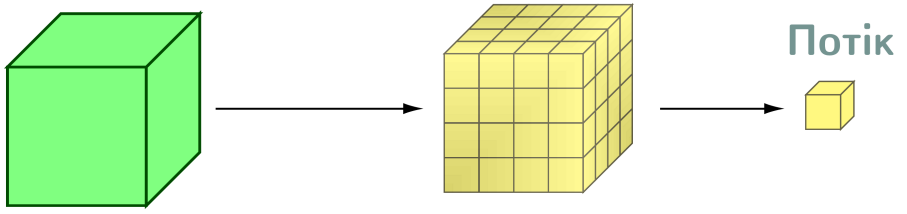
Джерело слайду: Programming Model, CUDA Toolkit v11.6.0

Ієрархія потоків

Сітка: масив блоків



Блок: масив потоків

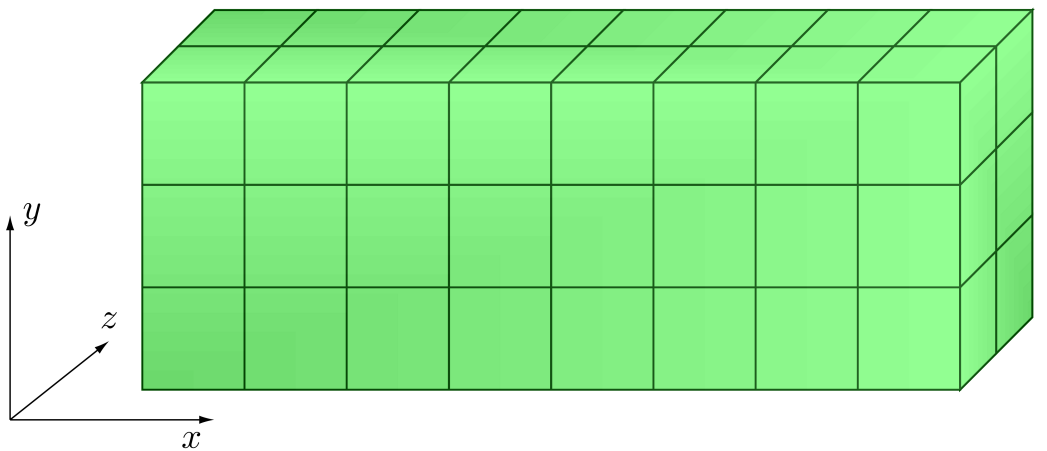


gridDim

Кількість блоків у кожному вимірі:

- `gridDim.x` $= 8$
- `gridDim.y` $= 3$
- `gridDim.z` $= 2$

Сітка: масив блоків

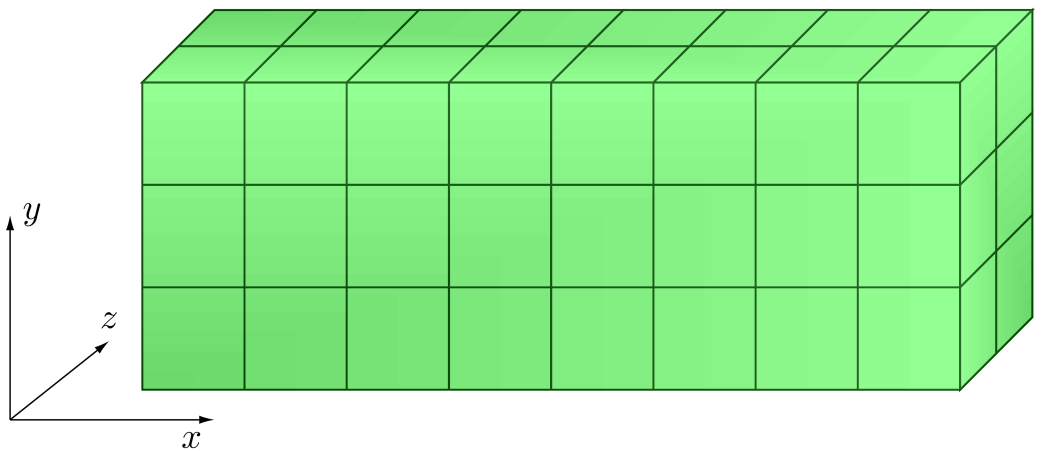


`blockIdx`

Кожен блок має унікальний індекс у сітці:

- `blockIdx.x` (від 0 до `gridDim.x` $- 1$)
- `blockIdx.y` (від 0 до `gridDim.y` $- 1$)
- `blockIdx.z` (від 0 до `gridDim.z` $- 1$)

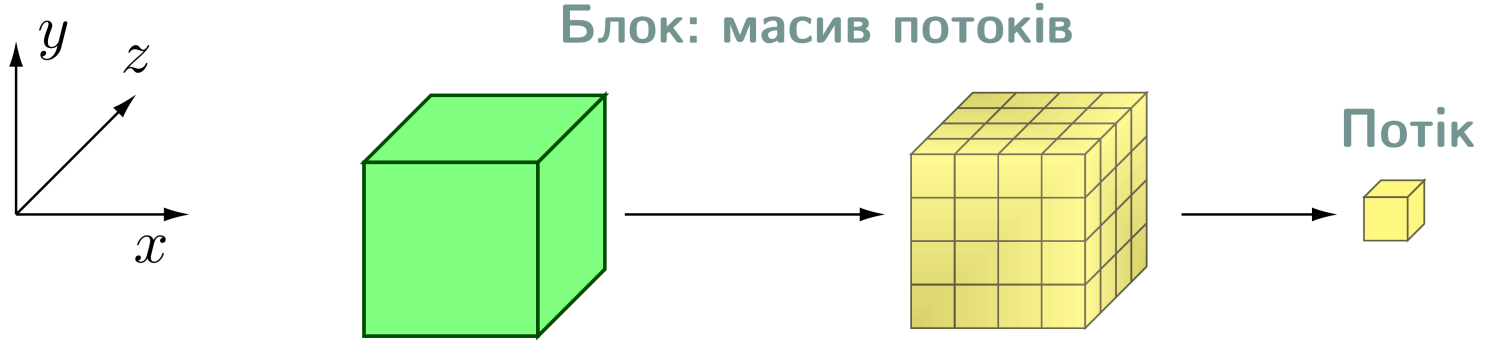
Сітка: масив блоків



`blockDim`

Кількість потоків у блоці:

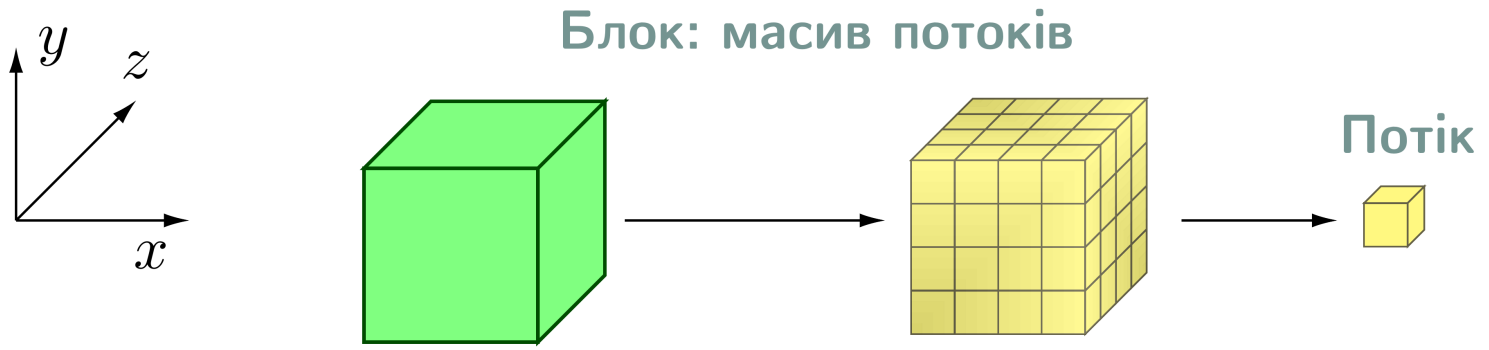
- `blockDim.x` $= 4$
- `blockDim.y` $= 4$
- `blockDim.z` $= 4$



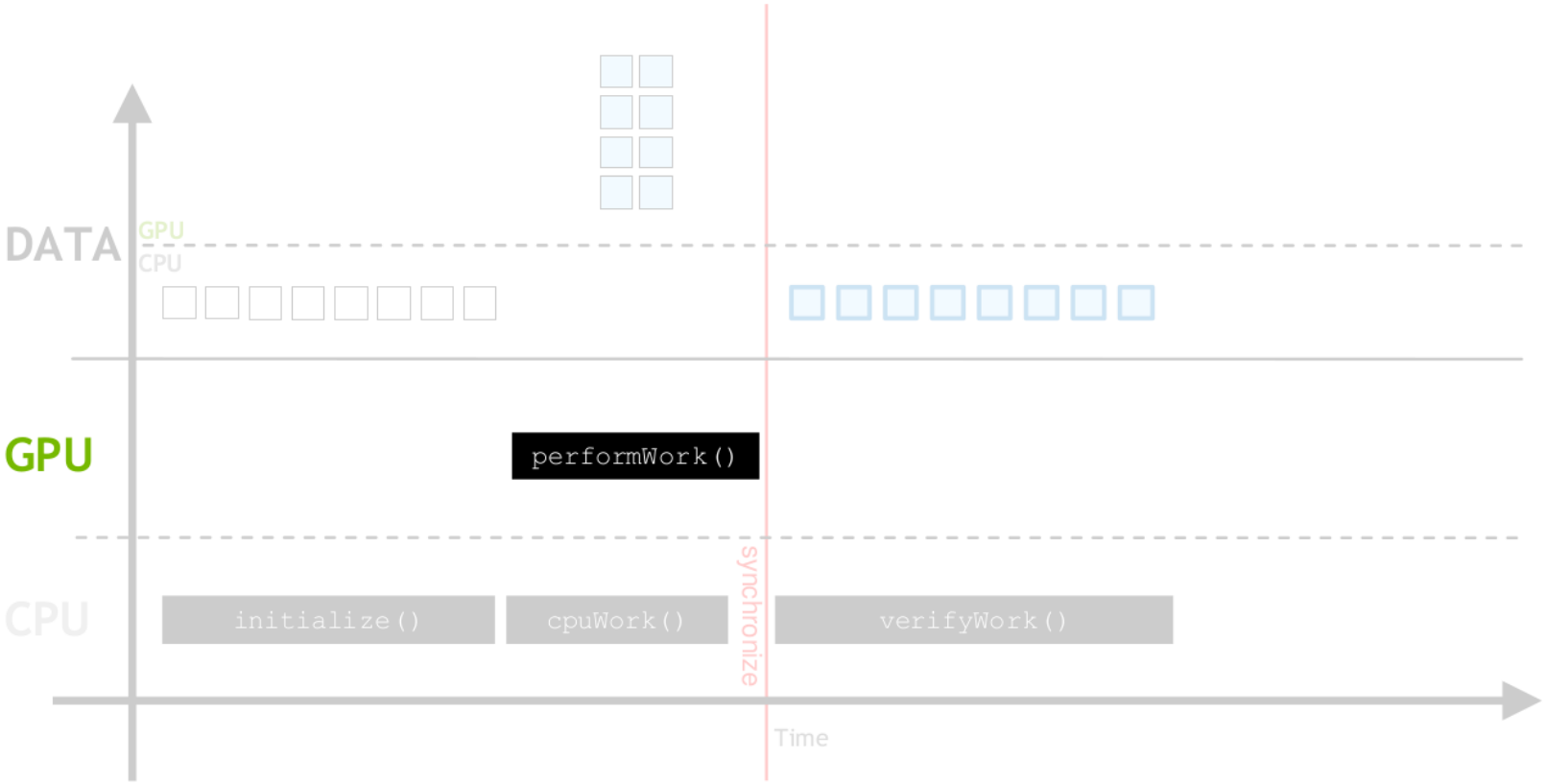
`threadIdx`

Кожен потік має унікальний індекс у блоці:

- `threadIdx.x` (від 0 до `blockDim.x` $- 1$)
- `threadIdx.y` (від 0 до `blockDim.x` $- 1$)
- `threadIdx.z` (від 0 до `blockDim.x` $- 1$)



Приклад: одновимірний випадок

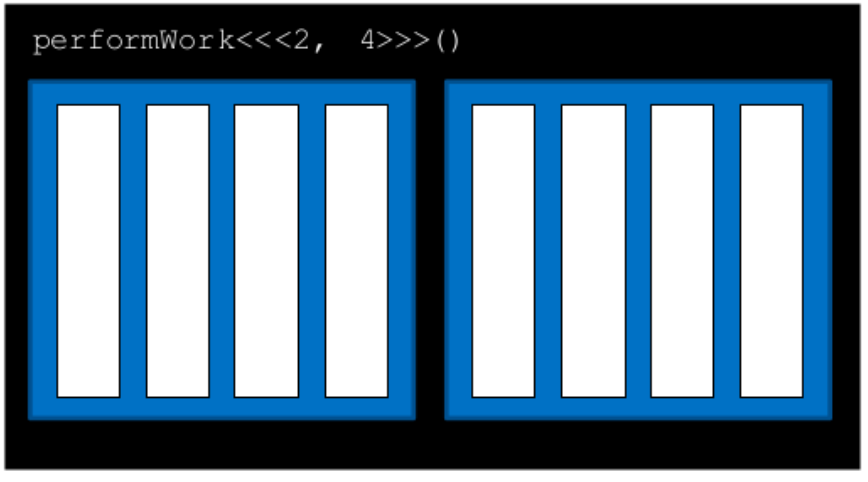


—

Джерело слайду: NVIDIA, DLI

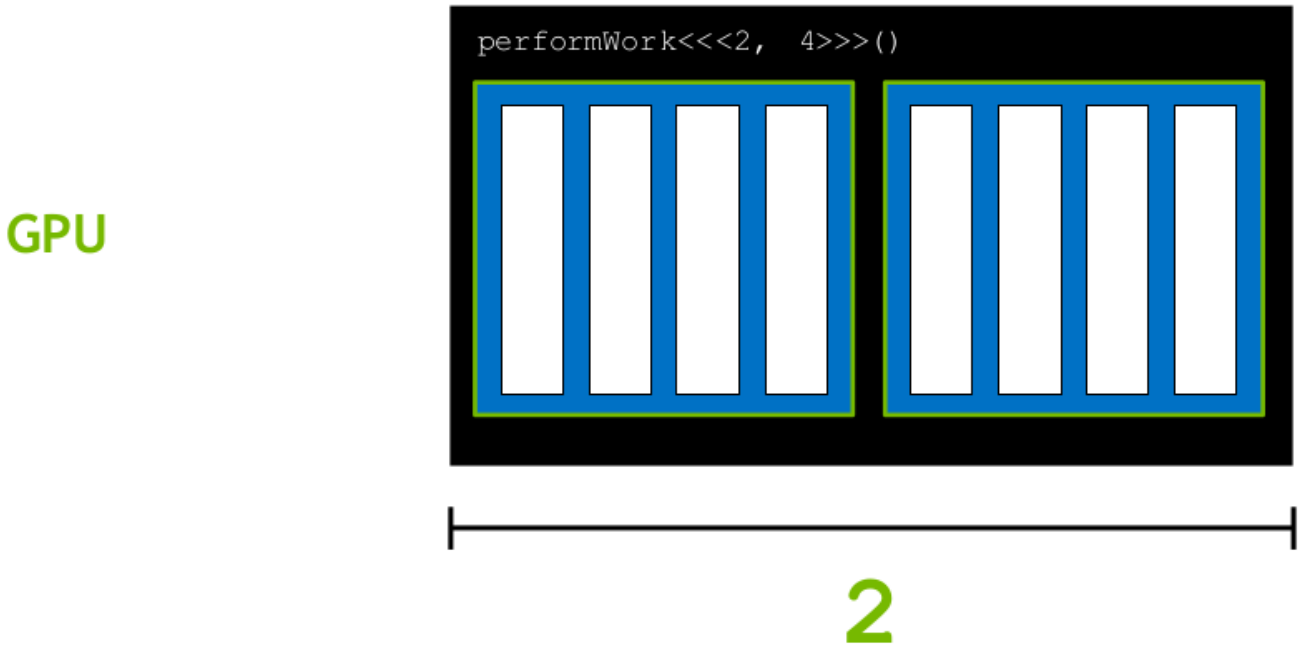
Для виконання ядра потрібно вказати кілікість блоків та кількість потоків в одному блоці

GPU



—
Джерело слайду: NVIDIA, DLI

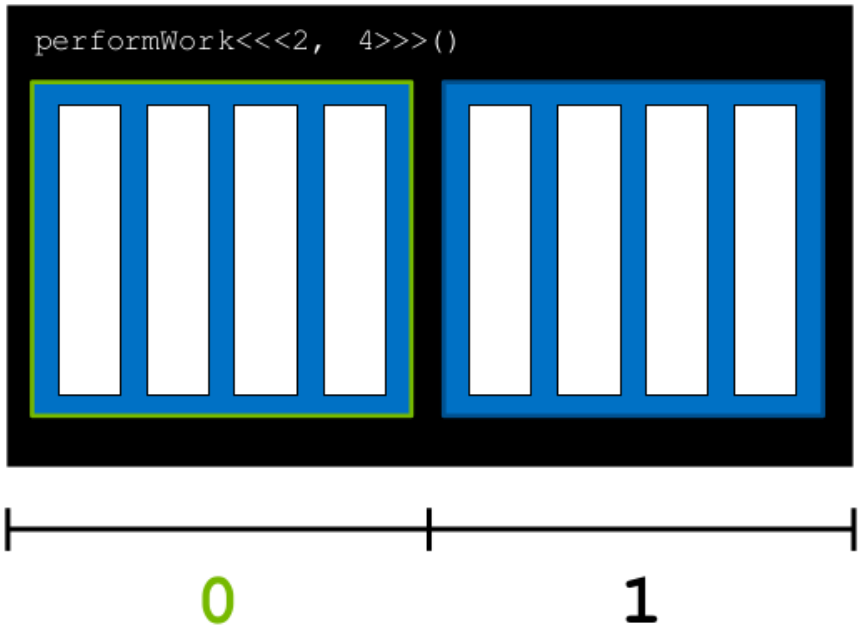
`gridDim.x` визначає кількість блоків у сітці, у цьому випадку $\texttt{gridDim.x} = 2$



—
Джерело слайду: NVIDIA, DLI

`blockIdx.x` визначає поточний індекс блоку у сітці, у цьому випадку $\texttt{blockIdx.x} = 0$

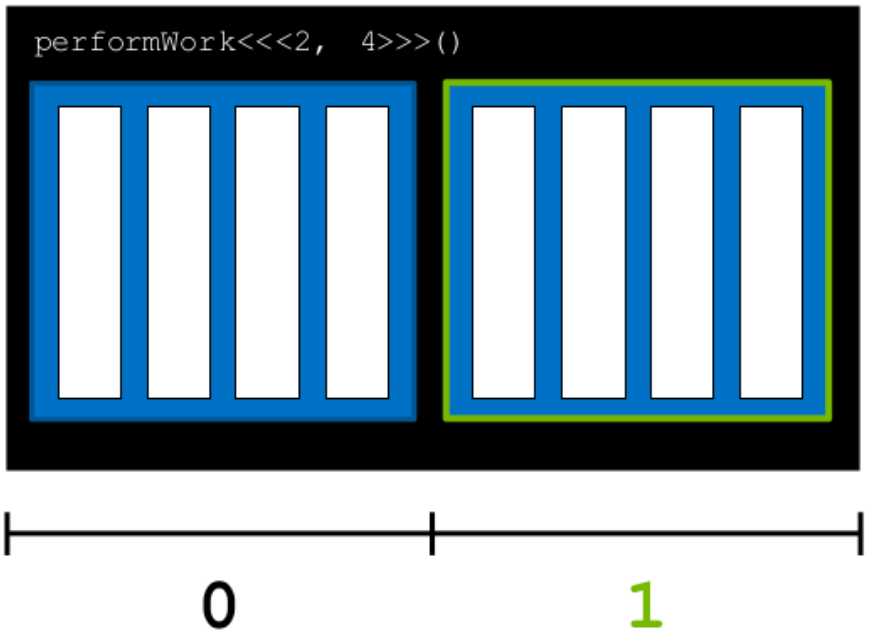
GPU



—
Джерело слайду: NVIDIA, DLI

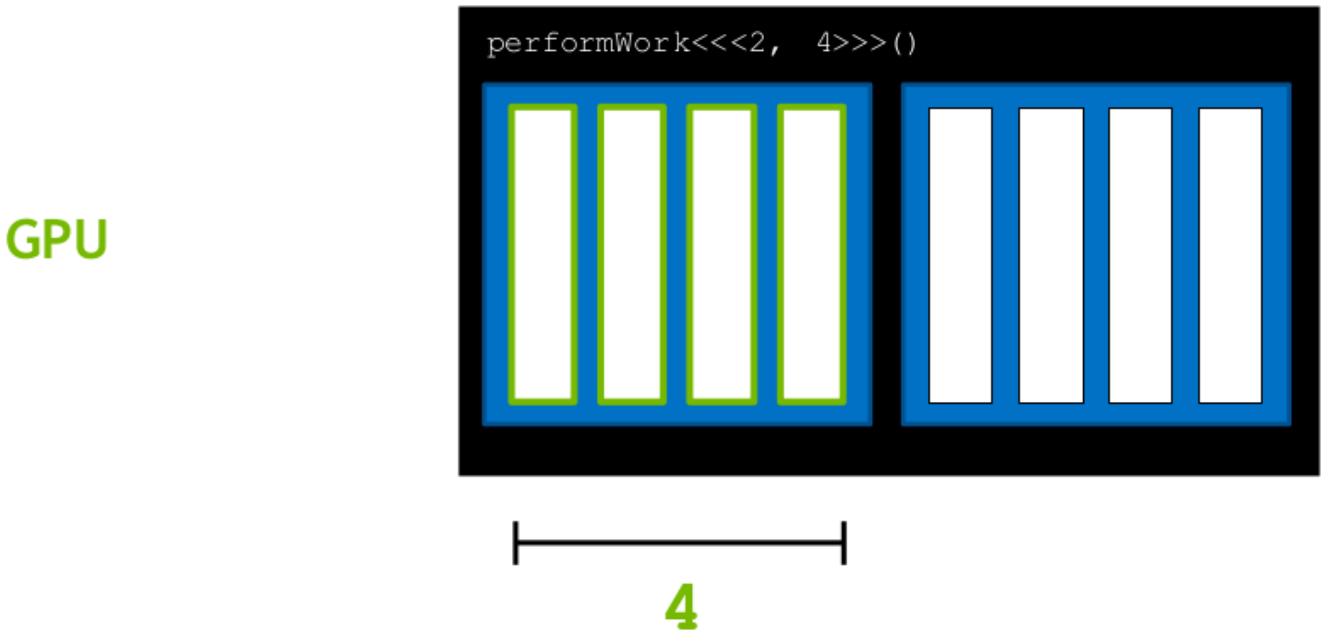
`blockIdx.x` визначає поточний індекс блоку у сітці, у цьому випадку `blockIdx.x` $= 1$

GPU



—
Джерело слайду: NVIDIA, DLI

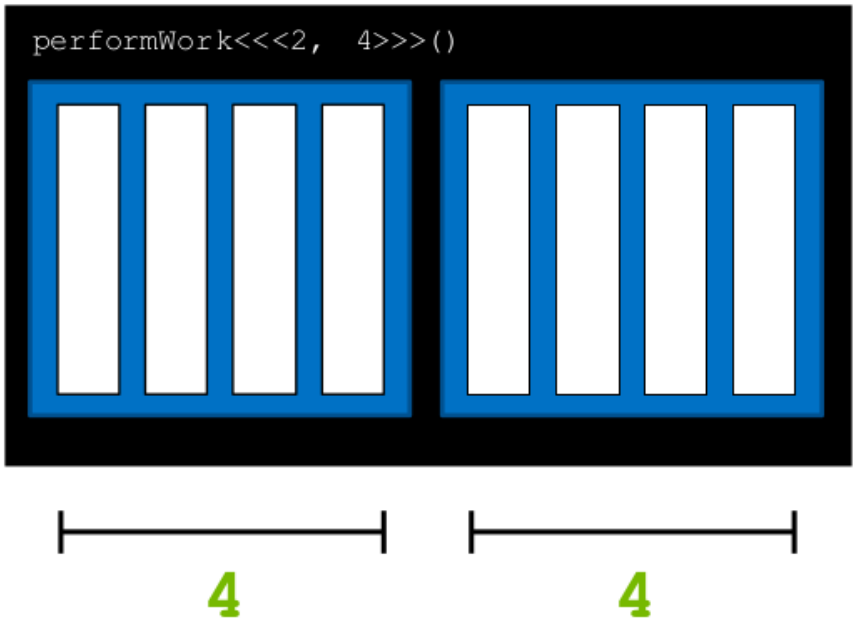
`blockDim.x` визначає кількість потоків у блоці, у цьому випадку $\texttt{blockDim.x} = 4$



—
Джерело слайду: NVIDIA, DLI

Усі блоки у сітці містять однакову кількість потоків

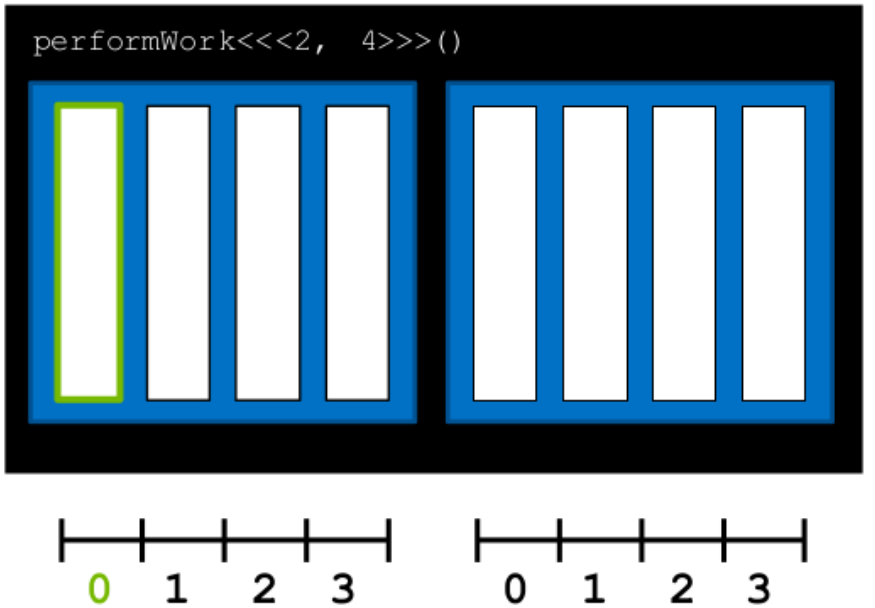
GPU



—
Джерело слайду: NVIDIA, DLI

`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку `threadIdx.x` $= 0$

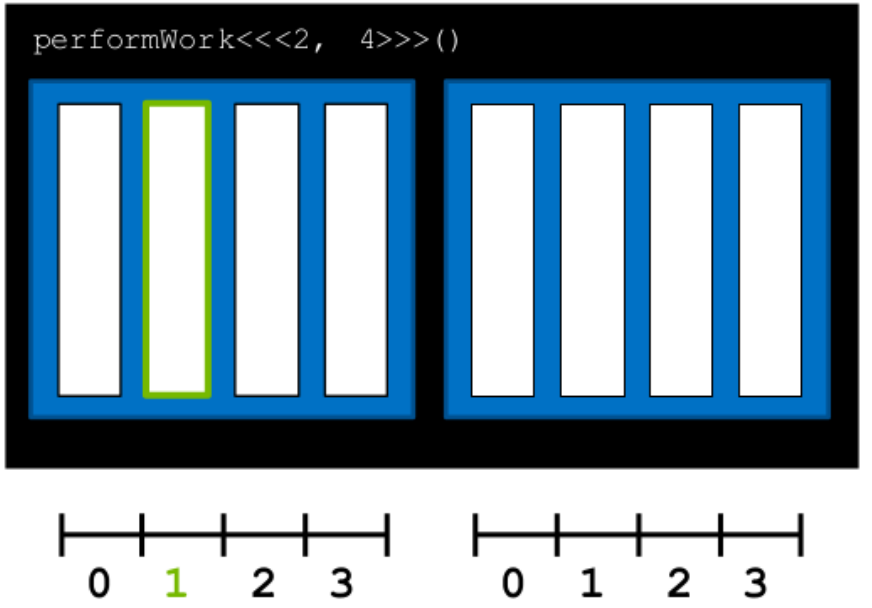
GPU



—
Джерело слайду: NVIDIA, DLI

`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку `threadIdx.x` $= 1$

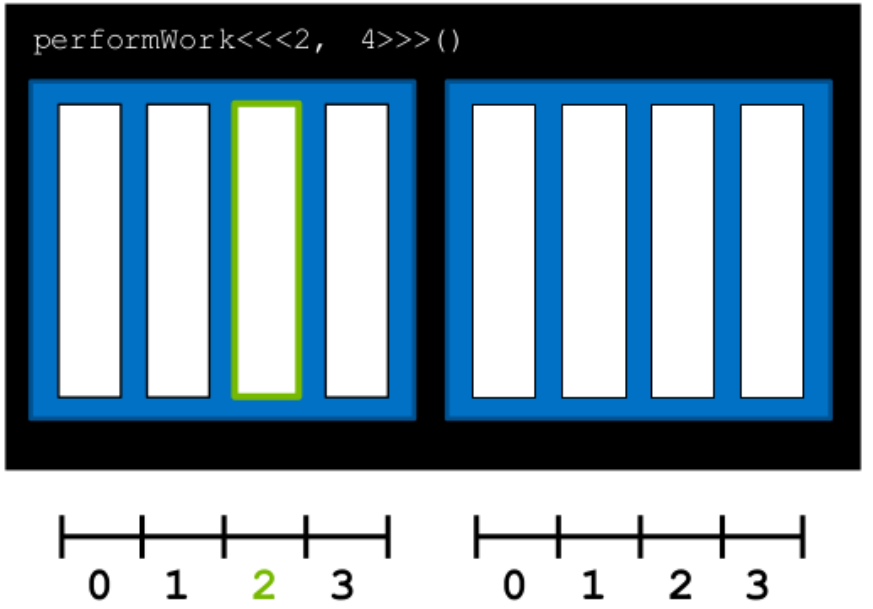
GPU



—
Джерело слайду: NVIDIA, DLI

`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку `threadIdx.x` $= 2$

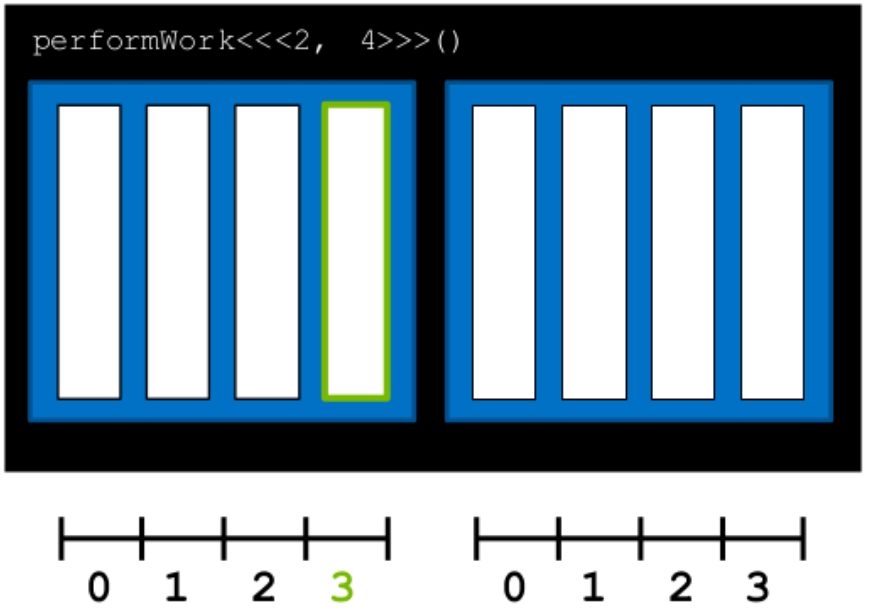
GPU



—
Джерело слайду: NVIDIA, DLI

`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку `threadIdx.x` $= 3$

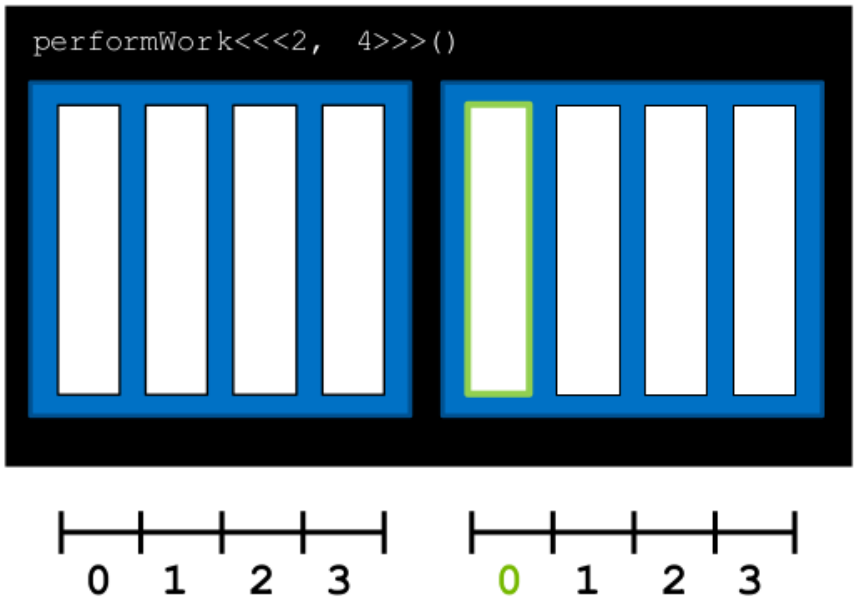
GPU



—
Джерело слайду: NVIDIA, DLI

`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку `threadIdx.x` $= 0$

GPU

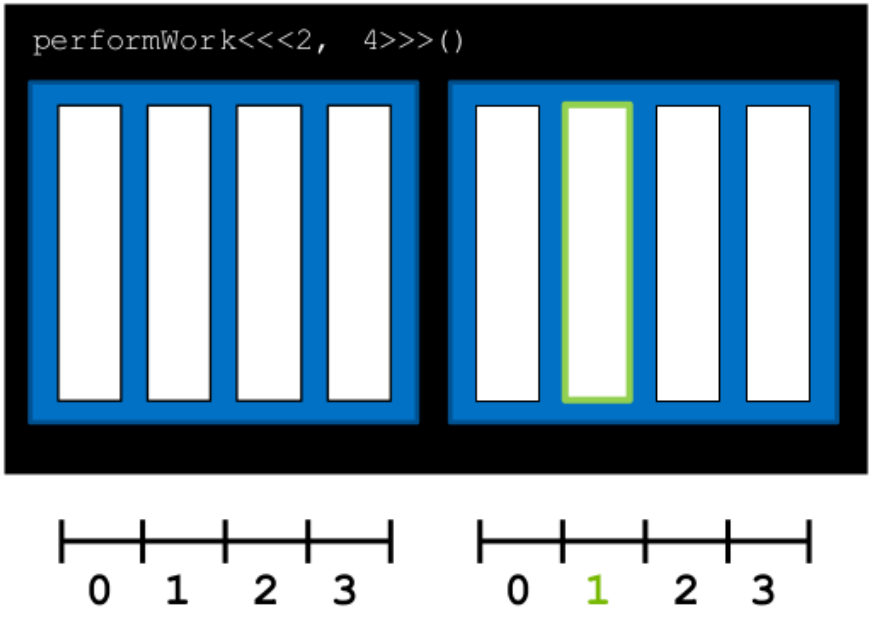


—
Джерело слайду: NVIDIA, DLI

`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку

`threadIdx.x` $= 1$

GPU

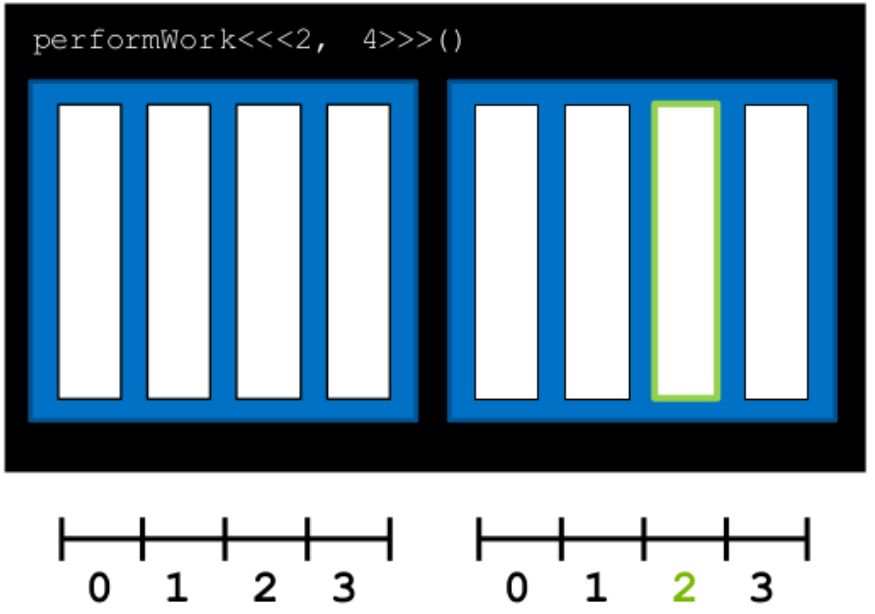


—

Джерело слайду: NVIDIA, DLI

`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку `threadIdx.x` $= 2$

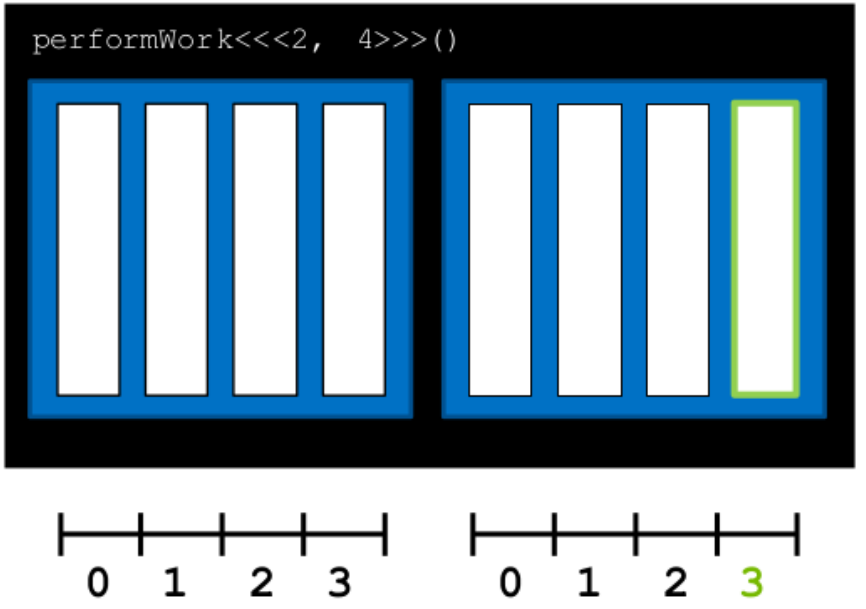
GPU



—
Джерело слайду: NVIDIA, DLI

`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку `threadIdx.x` $= 3$

GPU



—
Джерело слайду: NVIDIA, DLI

Демо: запит пристрою

Демо: додавання векторів

Додаток

- NVIDIA, CUDA Quick Start Guide

Кінець 🏁