



# Технології графічного процесінгу & розподілених обчислень

Лекція 2: Вступ до CUDA C

Кочура Юрій Петрович  
[iuriy.kochura@gmail.com](mailto:iuriy.kochura@gmail.com)  
[@y\\_kochura](#)

# Сьогодні

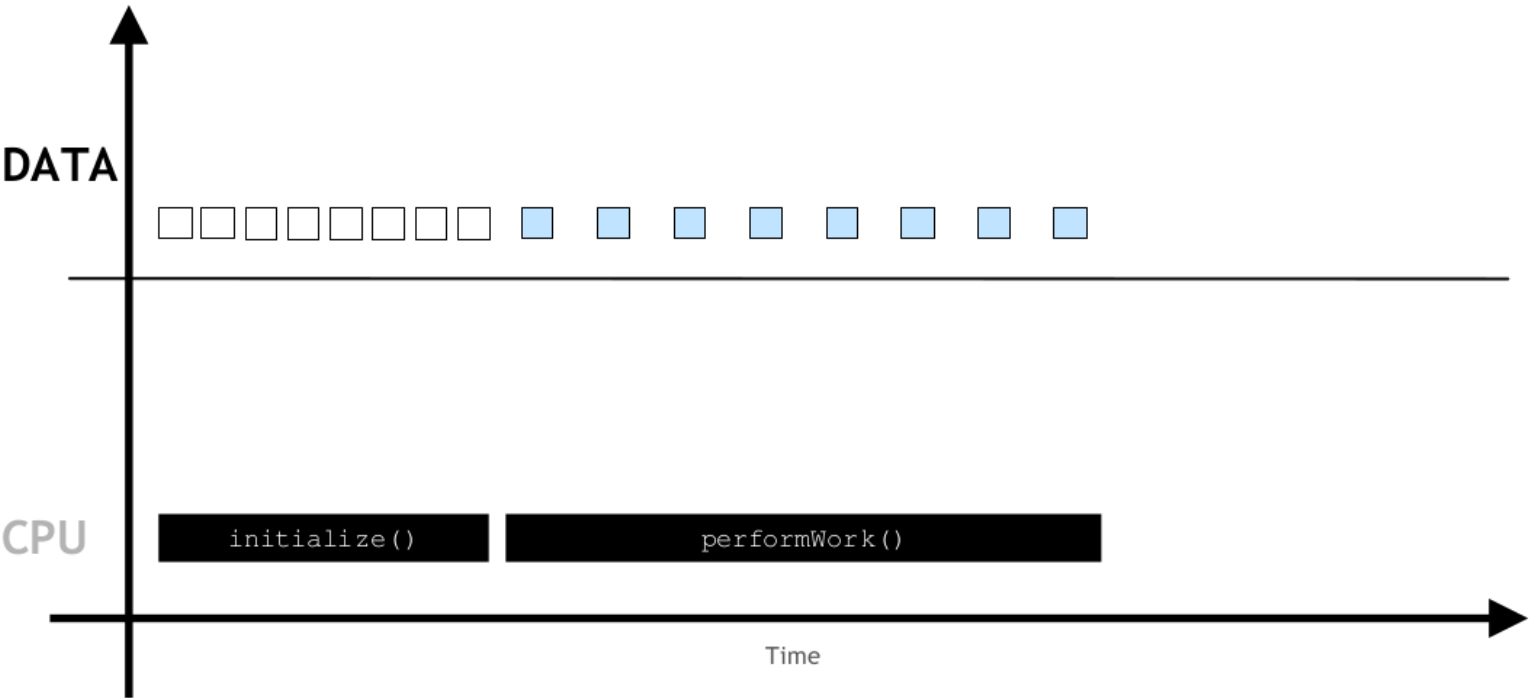
- GPU-прискорені vs лише CPU програми
- Способи прискорення виконання програм
- Приклади паралельних обчислень
- Процес компіляції CUDA C/C++
- Ядра: функції GPU
- Ієрархія потоків
- Демо

# GPU-прискорені vs лише CPU програми

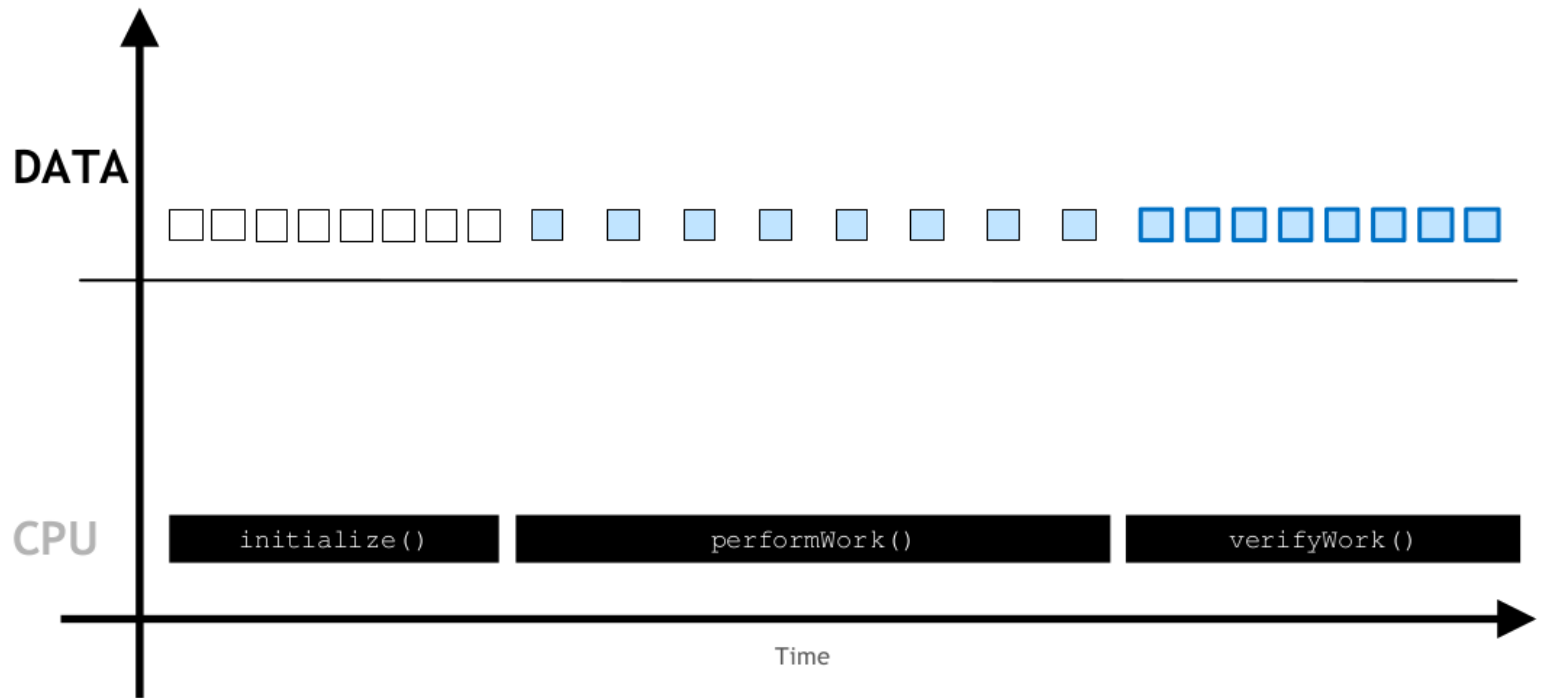


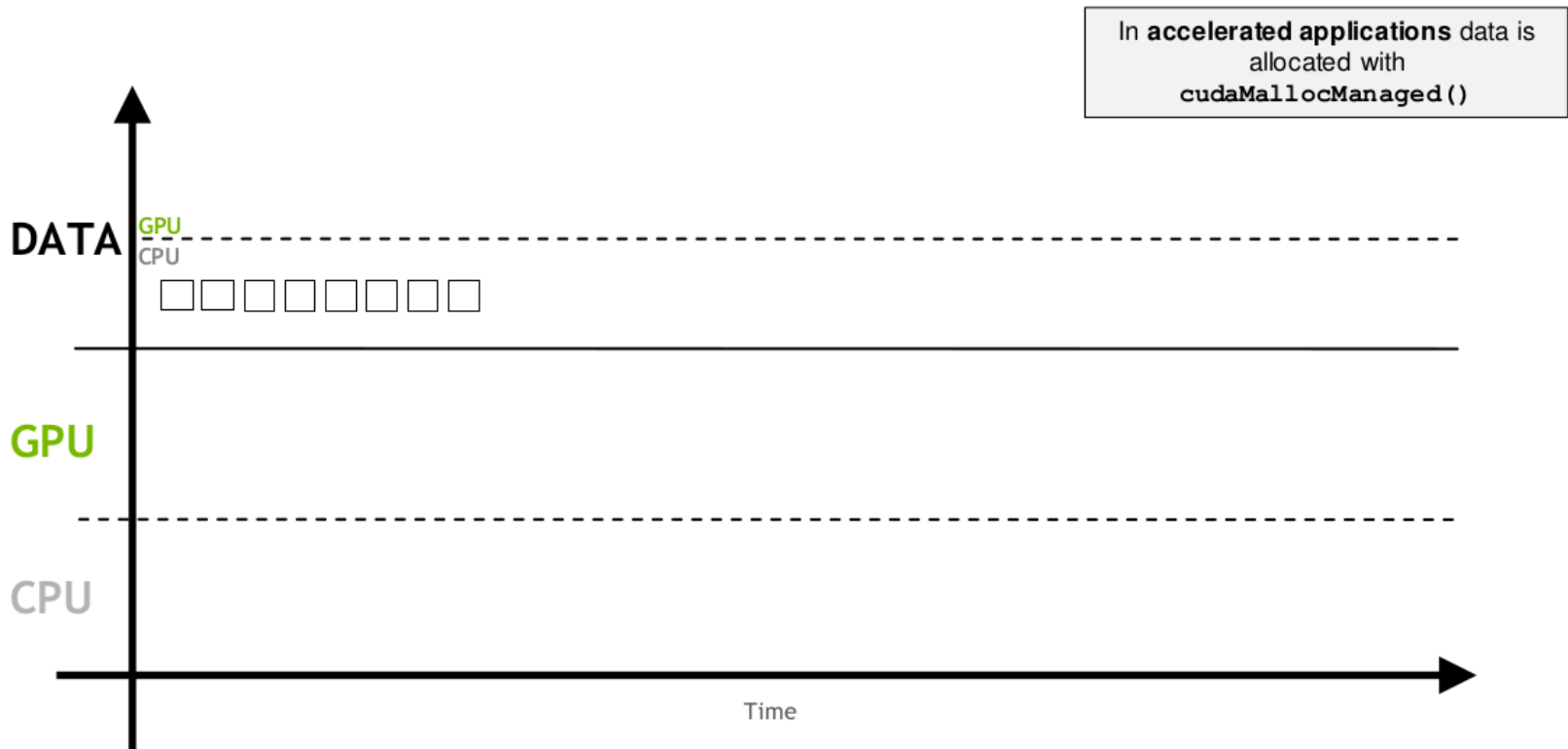
In **CPU-only applications** data is allocated on CPU

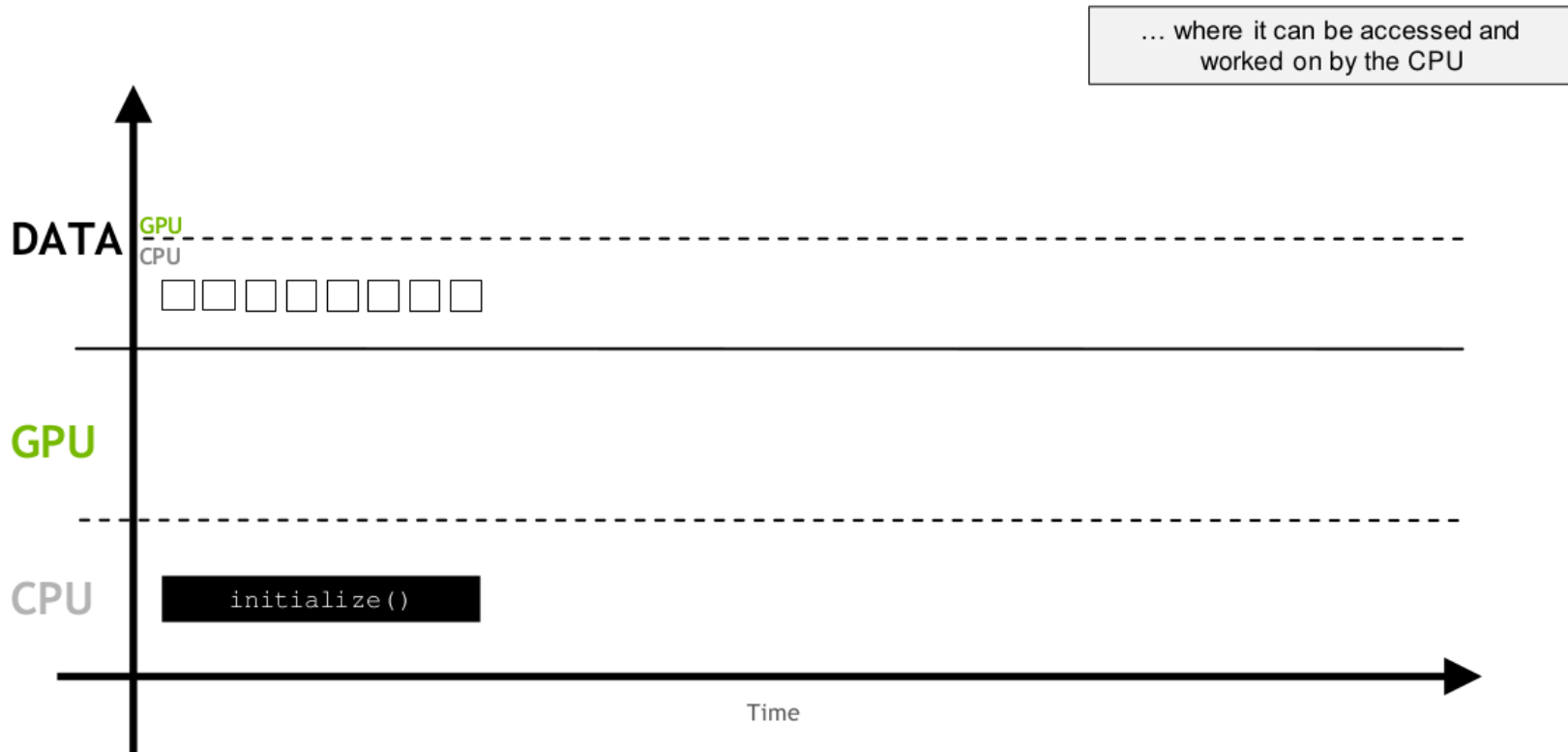
...and all work is performed on CPU



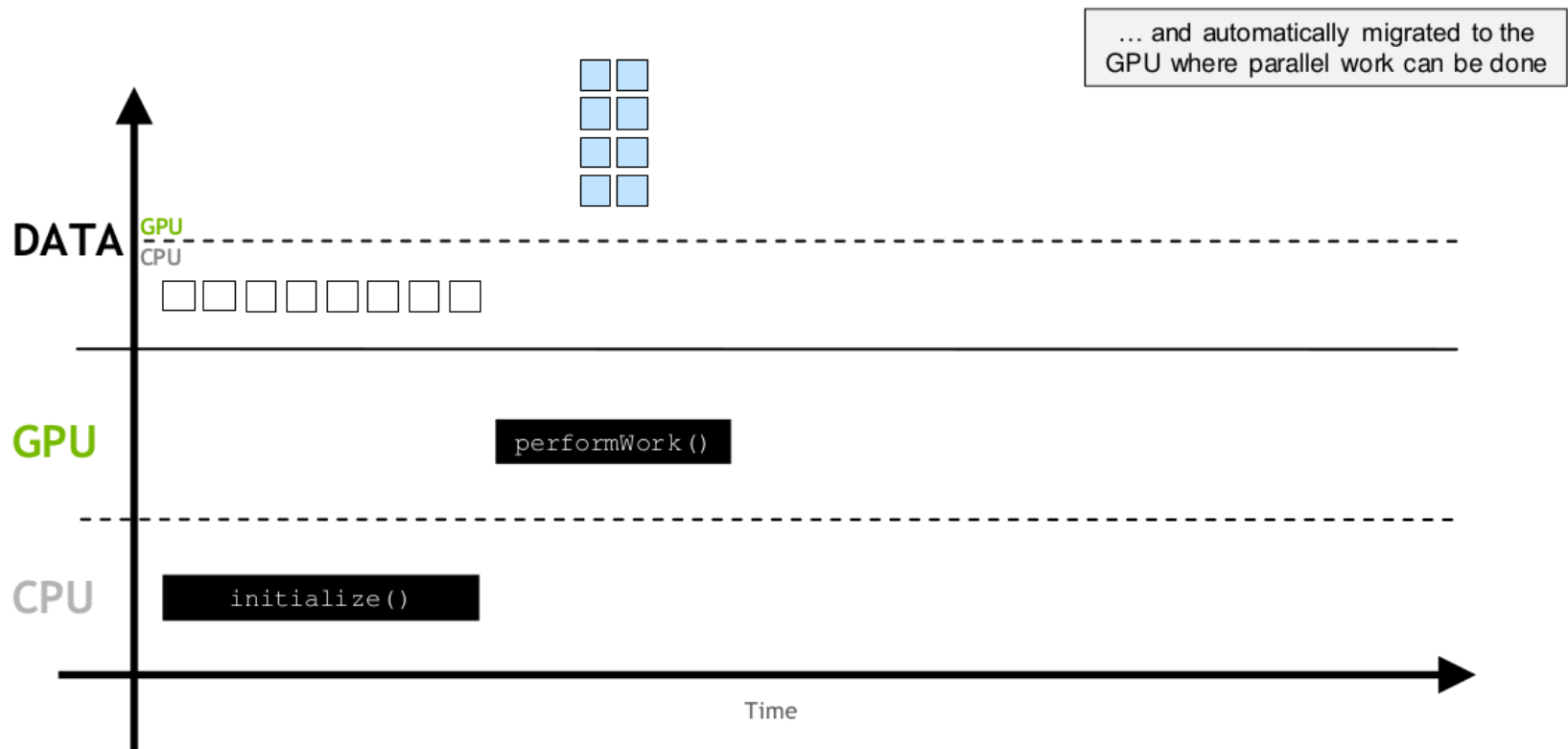
...and all work is performed on CPU

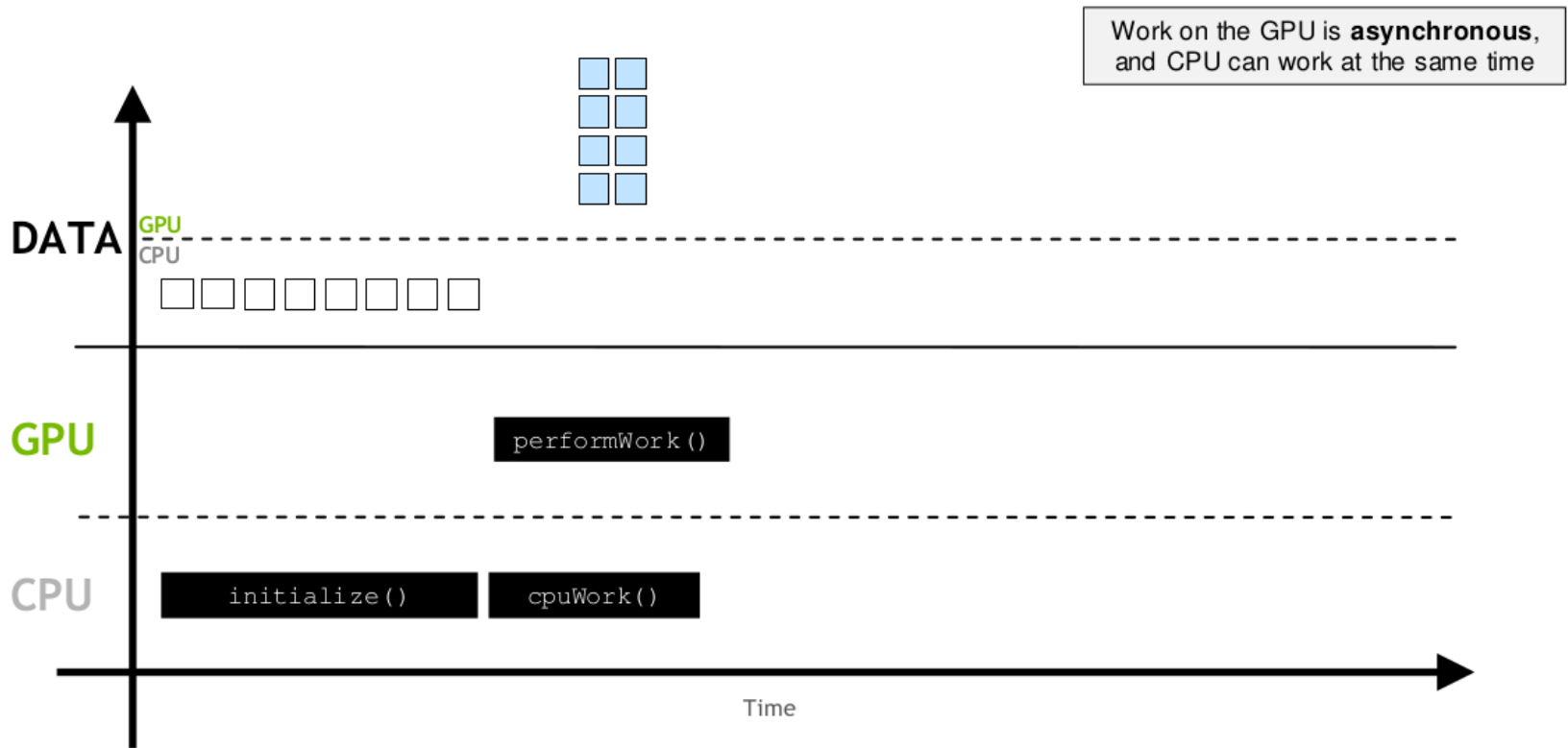


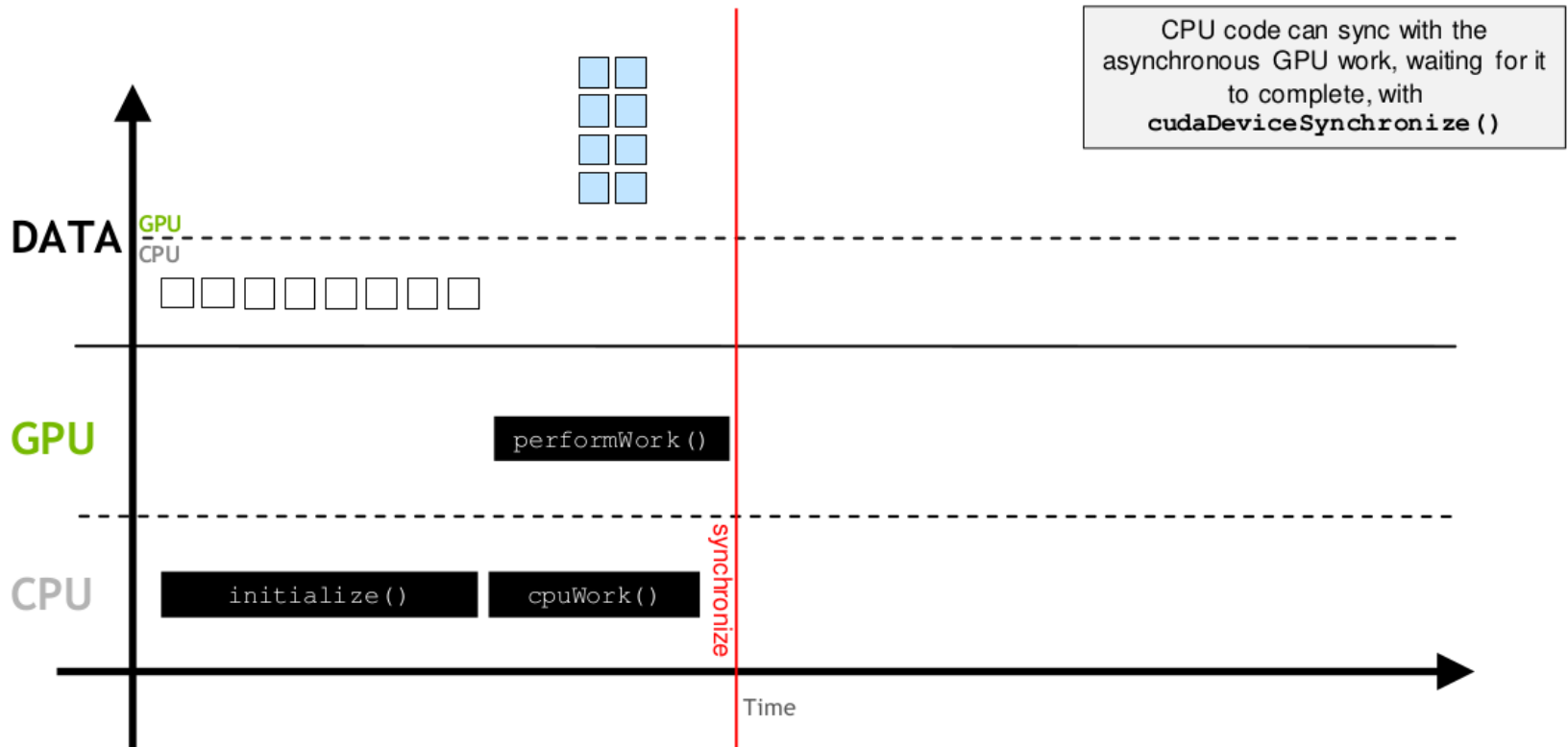


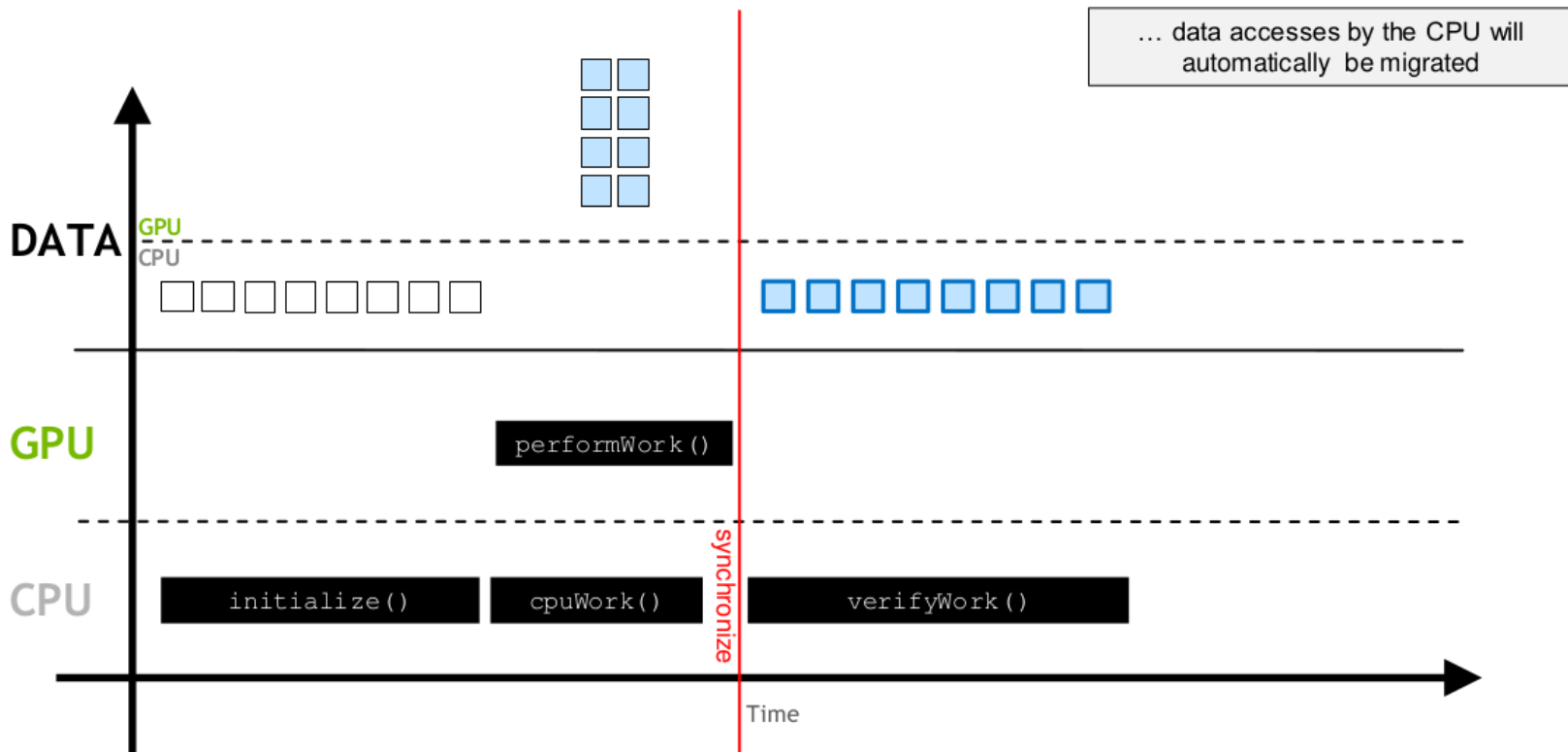












## Способи прискорення виконання програм



# Бібліотеки

## Бібліотеки: проста, високо-якісне прискорення

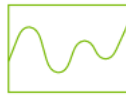
- **Простота використання:** використання бібліотек дозволяє прискорювати обчислень на GPU без поглиблених знань програмування GPU
- Багато бібліотек для прискорення обчислень на GPU дотримуються стандартних API, що дозволяє прискорювати роботу з мінімальними змінами коду
- **Якість:** бібліотеки пропонують 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



## Бібліотеки паралельних алгоритмів



Thrust

Бібліотека паралельних алгоритмів і структур даних C++ з GPU

## Бібліотеки візуальної обробки

### **nvJPEG**

High performance GPU-accelerated library for JPEG decoding

### **NVIDIA Performance Primitives**

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

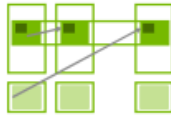
### **NVIDIA Video Codec SDK**

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

## 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.

## 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

# 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>  
    }  
}
```

# 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

C ▶

CUDA C

C++ ▶

CUDA C++

Python ▶

PyCUDA, Copperhead, Numba

Fortran ▶

CUDA Fortran

# GPU Computing Applications

## Libraries and Middleware

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

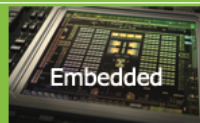
## Programming Languages

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

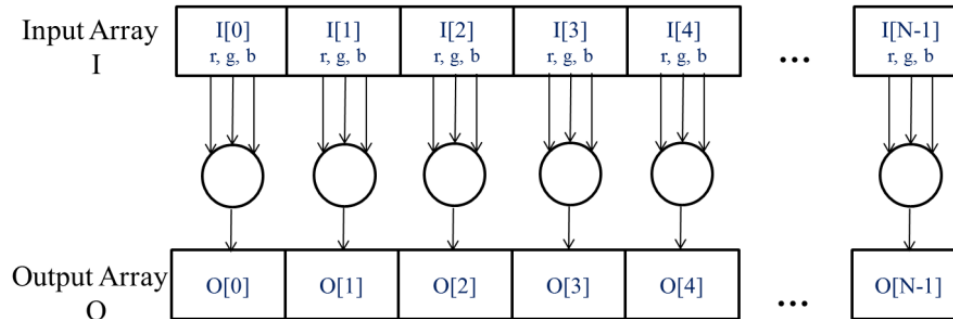


# Приклади паралельних обчислень

## Перетворення **RGB** зображення у відтінки сірого

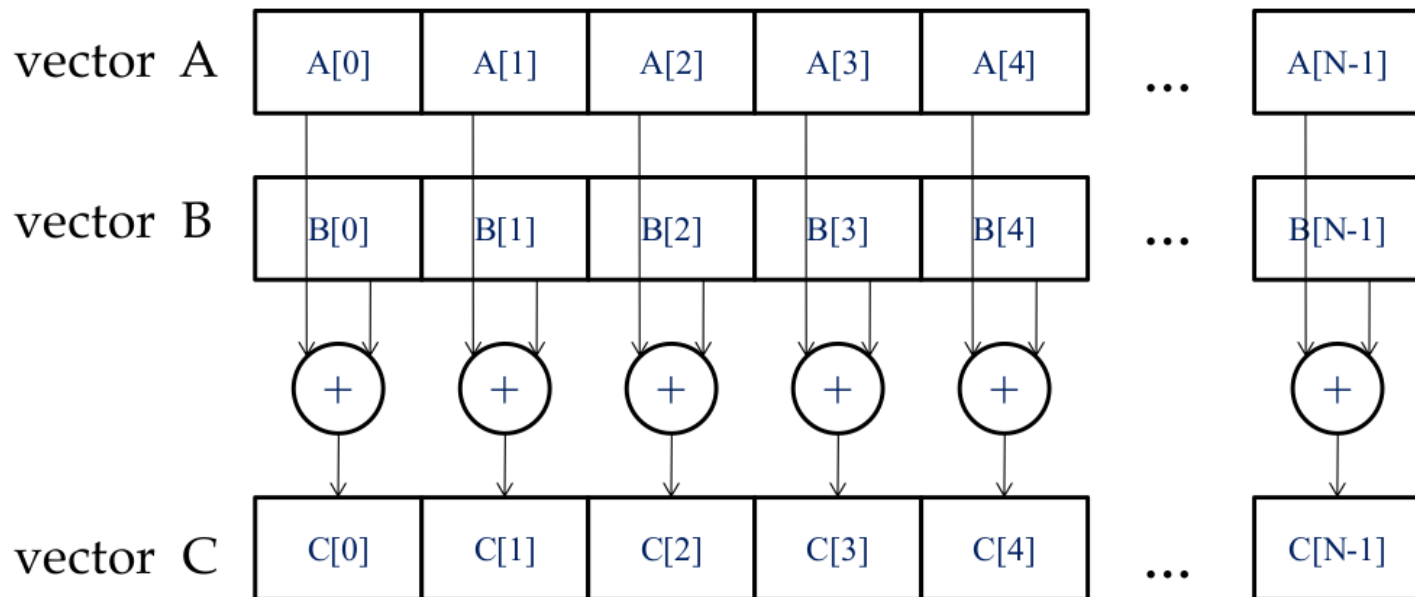


```
for each pixel {  
    pixel = gsConvert(pixel)  
}  
// Every pixel is independent  
// of every other pixel
```



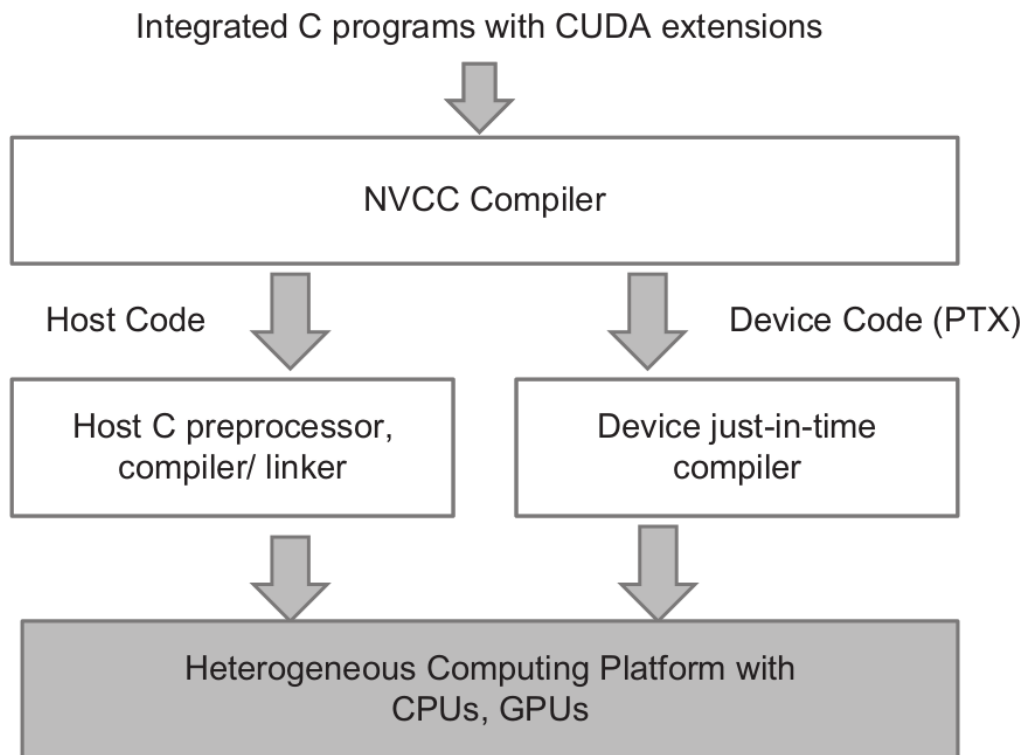
$$O = r \cdot 0.21 + g \cdot 0.72 + b \cdot 0.07$$

# Додавання векторів



## Процес компіляції **CUDA C/C++**

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



## CUDA/OpenCL — модель виконання

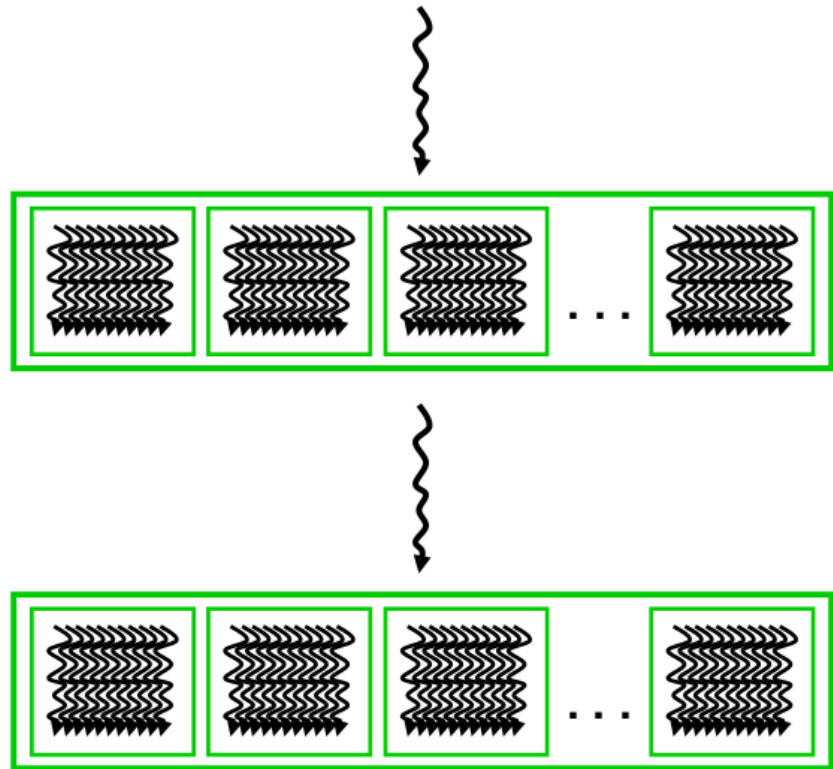
- Послідовні або помірно паралельні частини в коді, що виконуються **ХОСТОМ**
- Високо паралельні частини в коді, що виконуються **пристроєм**: ядро

Serial Code (host)

Parallel Kernel (device)

Serial Code (host)

Parallel Kernel (device)





# Ядра (Kernels)

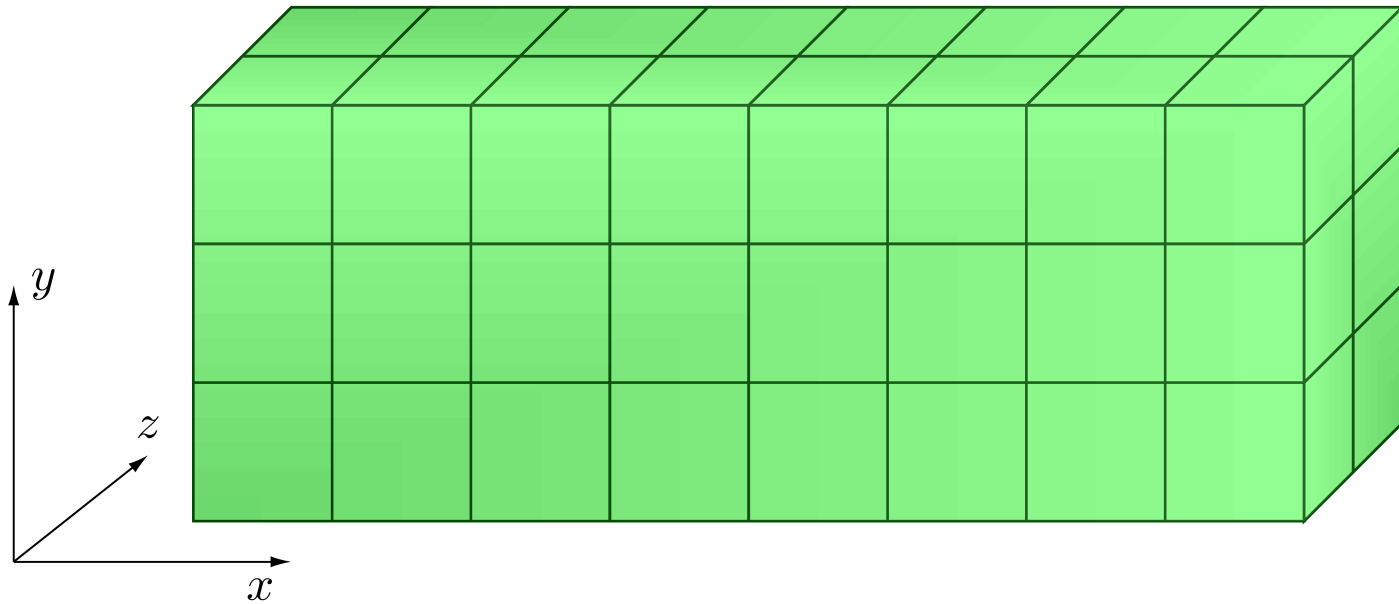
- Ядро визначається за допомогою специфікатора `__global__`

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

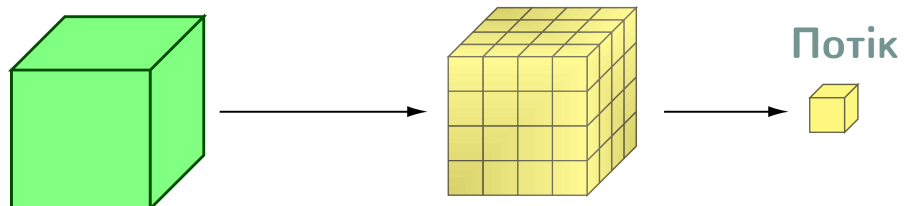
int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

# Ієрархія потоків

Сітка: масив блоків



Блок: масив потоків

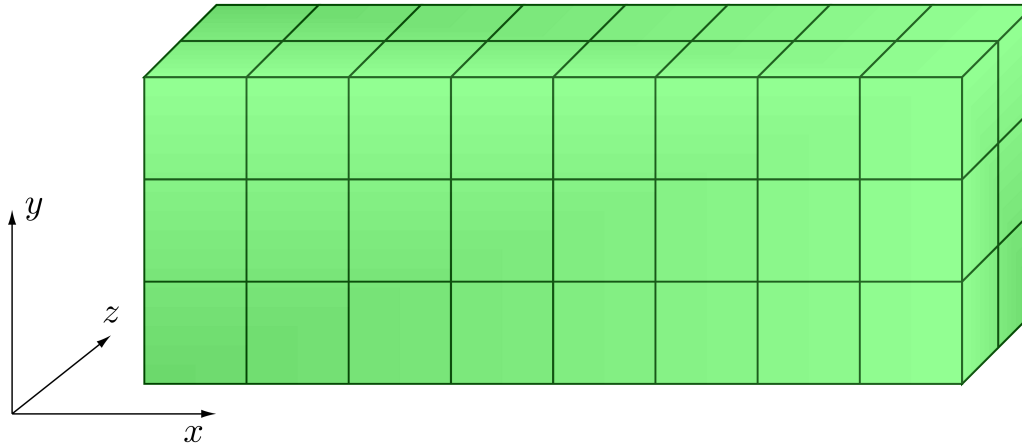


# gridDim

Кількість блоків у кожному вимірі:

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

Сітка: масив блоків

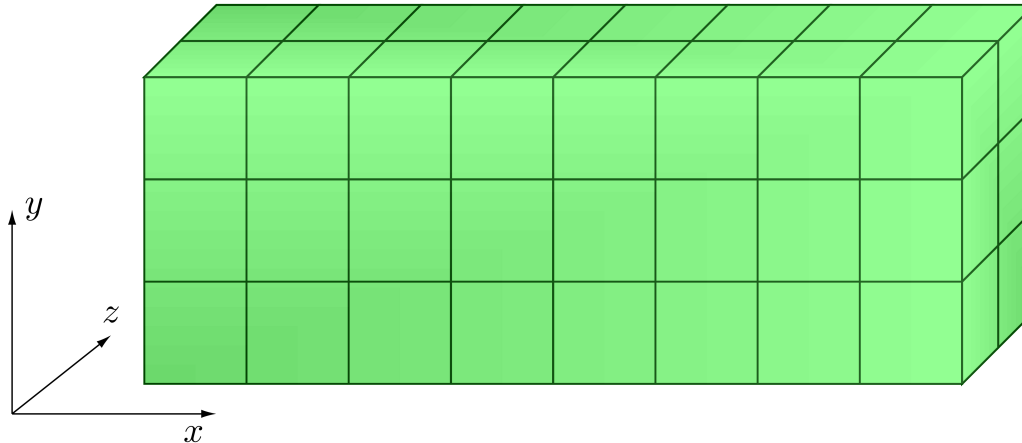


# blockIdx

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`)

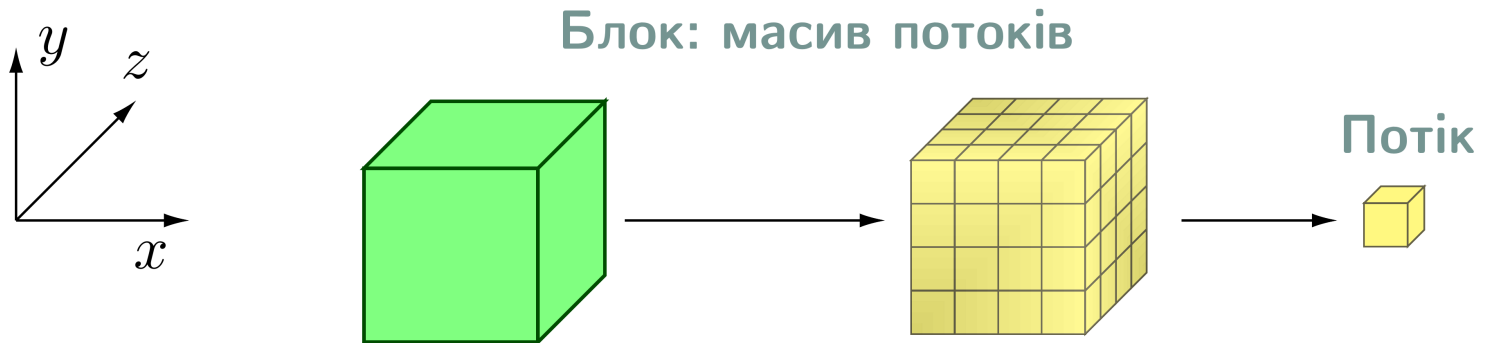
Сітка: масив блоків



# blockDim

Кількість потоків у блоці:

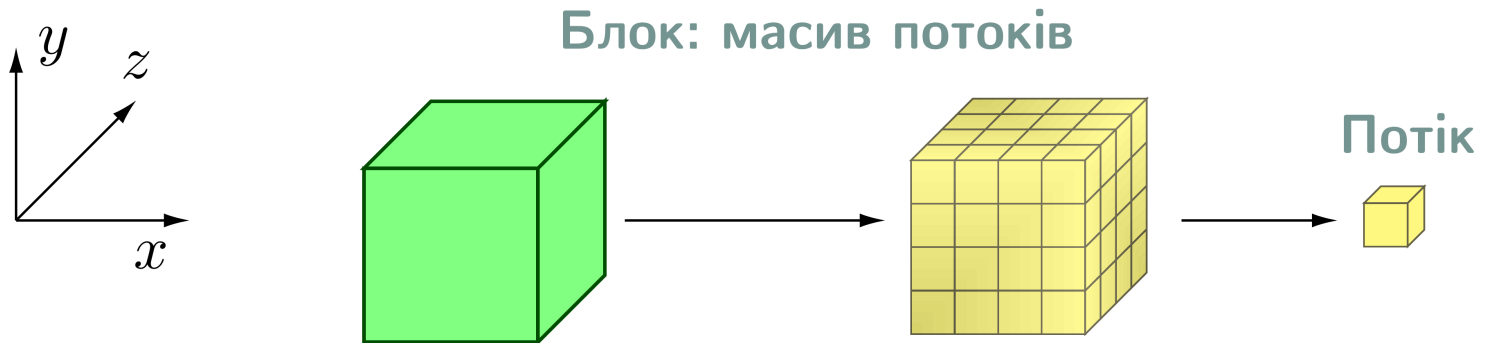
- `blockDim.x = 4`
- `blockDim.y = 4`
- `blockDim.z = 4`



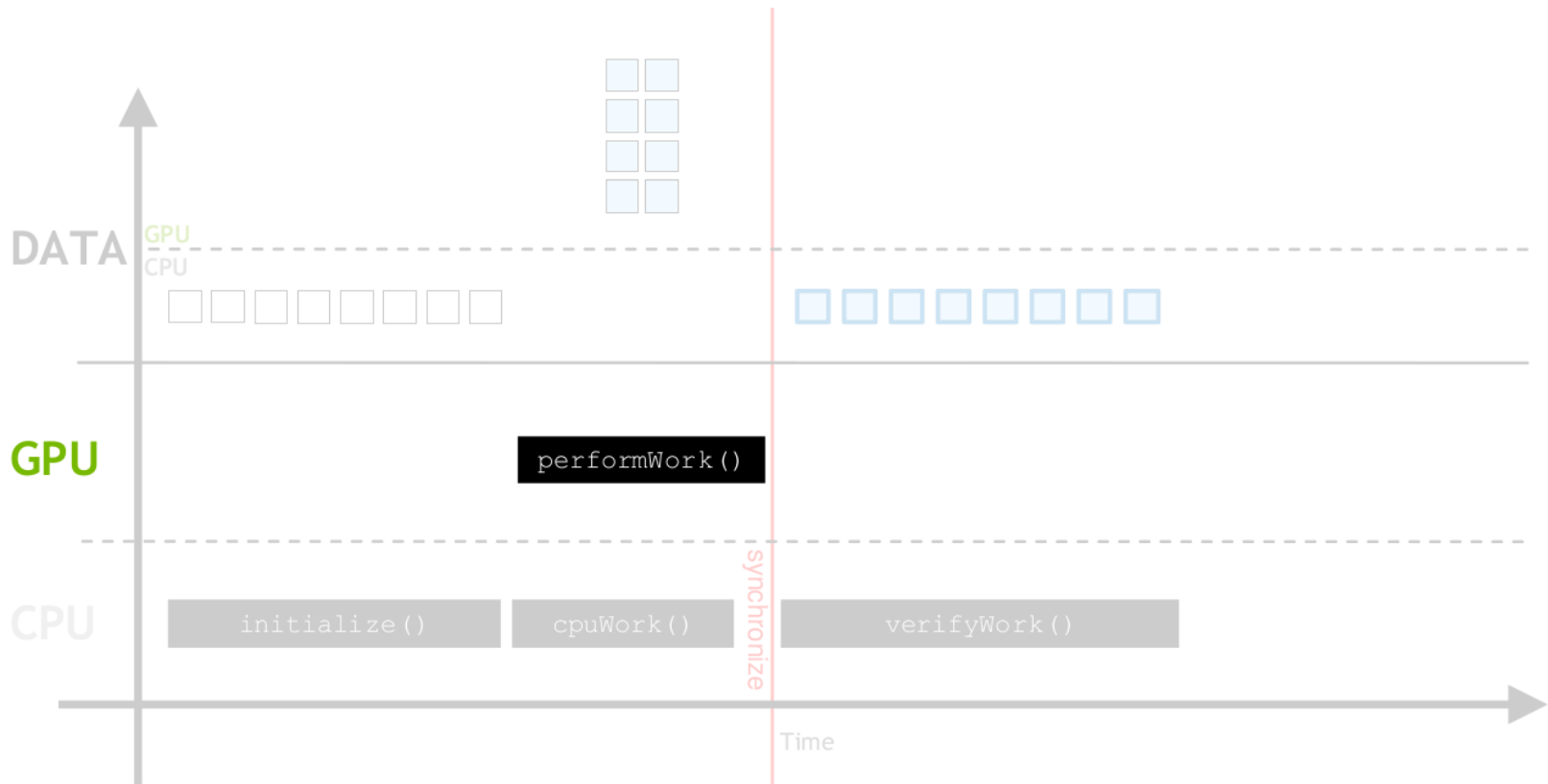
# threadIdx

Кожен потік має унікальний індекс у блоці:

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



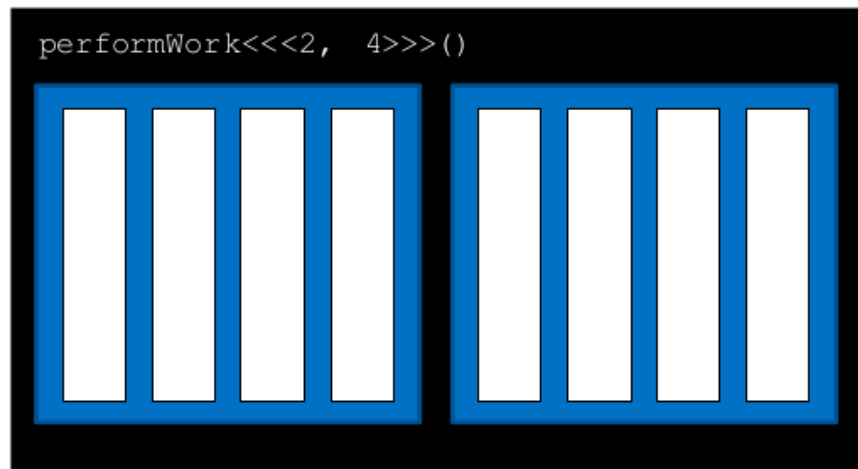
# Приклад: одновимірний випадок





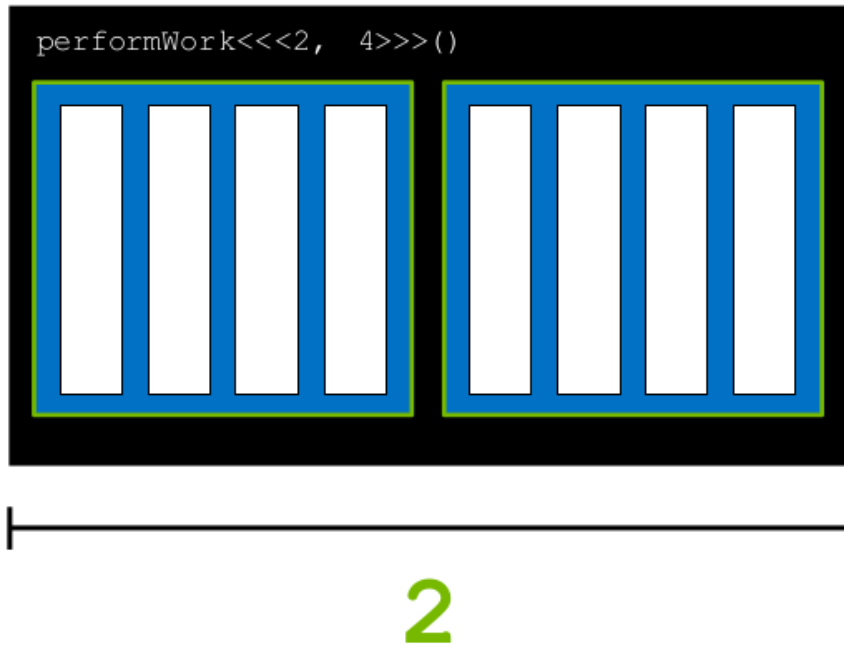
Для виконання ядра потрібно вказати **кількість блоків** та **кількість потоків** в одному блоці

GPU



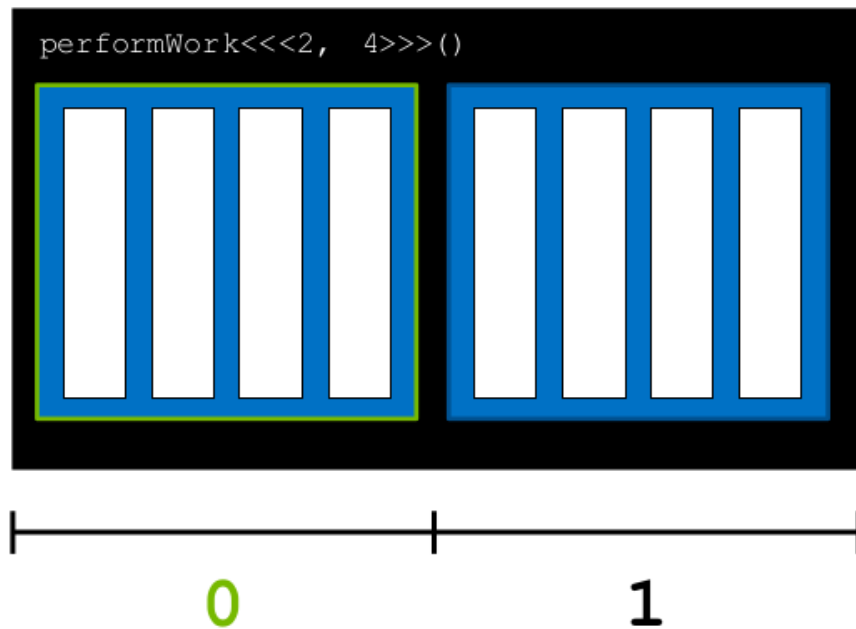
`gridDim.x` визначає кількість блоків у сітці, у цьому випадку `gridDim.x = 2`

GPU



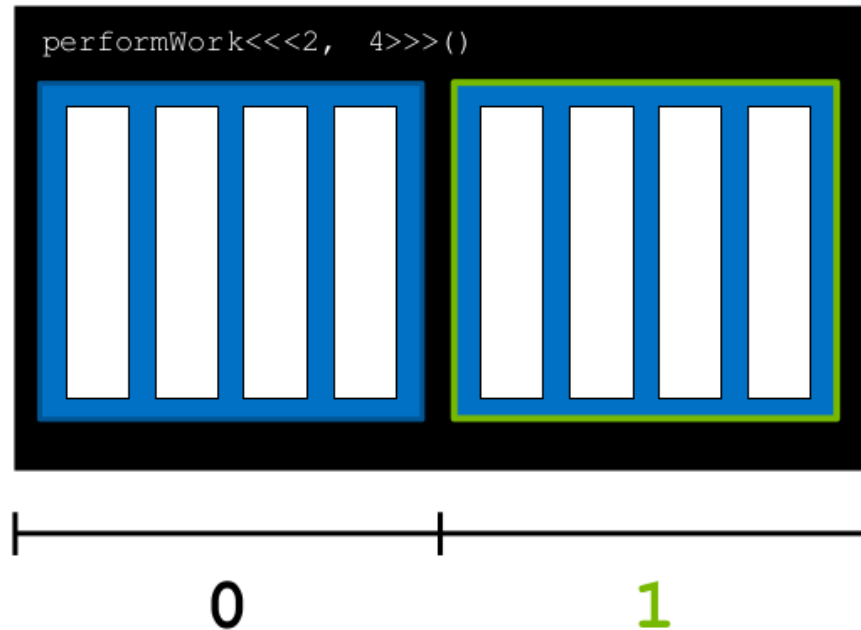
`blockIdx.x` визначає поточний індекс блоку у сітці, у цьому випадку `blockIdx.x = 0`

GPU



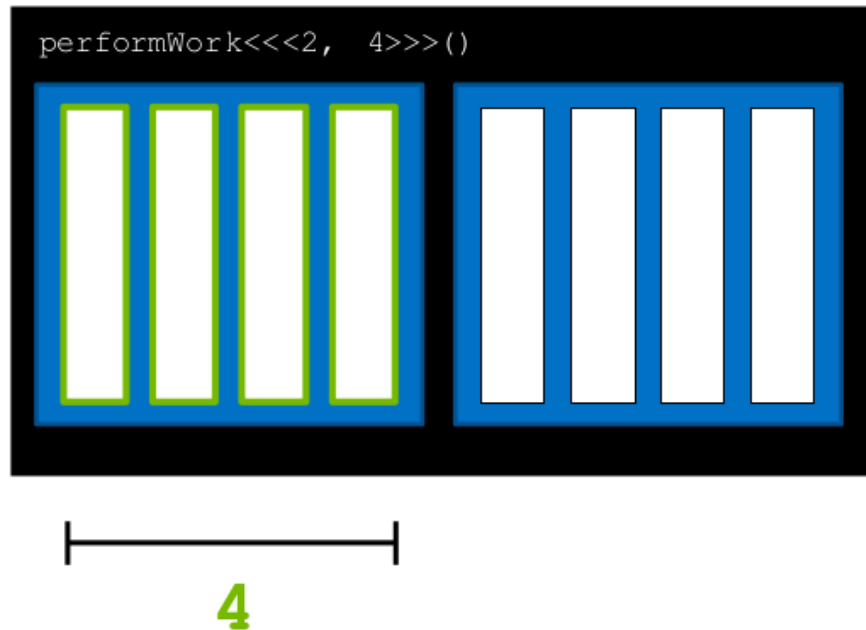
`blockIdx.x` визначає поточний індекс блоку у сітці, у цьому випадку `blockIdx.x = 1`

GPU



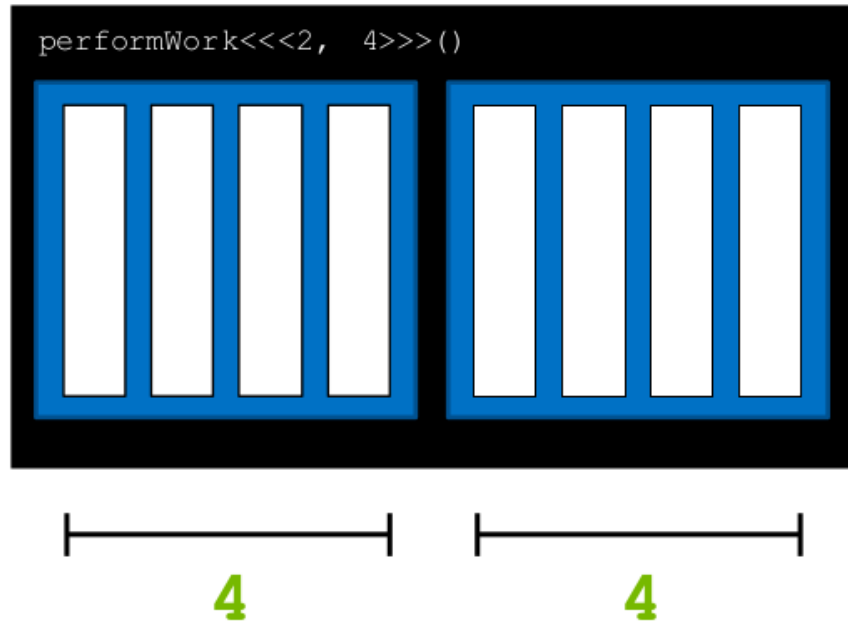
`blockDim.x` визначає кількість потоків у блоці, у цьому випадку `blockDim.x = 4`

GPU



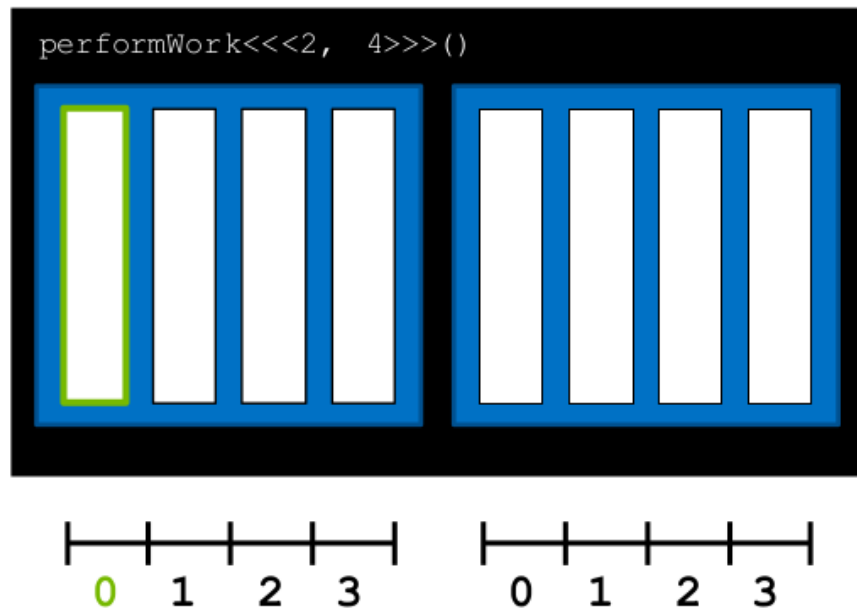
Усі блоки у сітці містять **однакову** кількість потоків

GPU



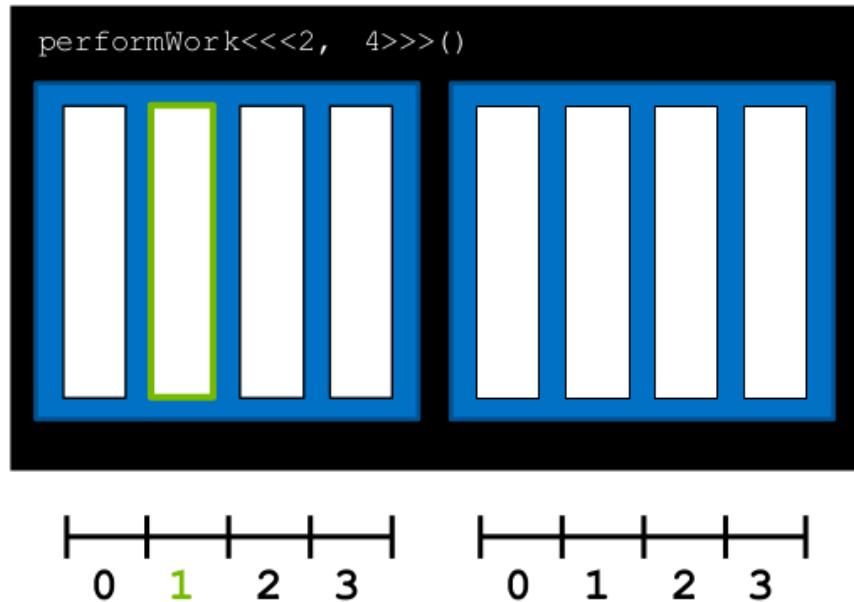
`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку  
`threadIdx.x = 0`

GPU



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

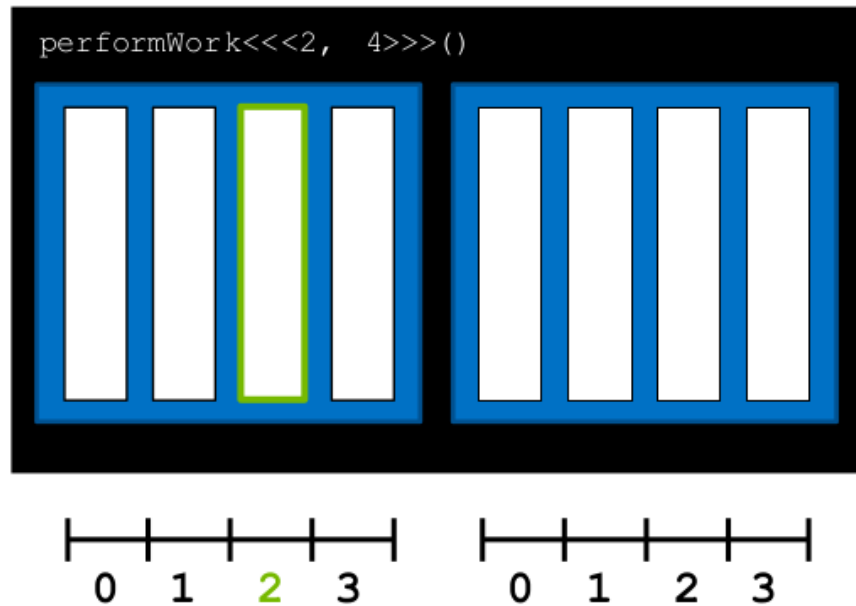
GPU





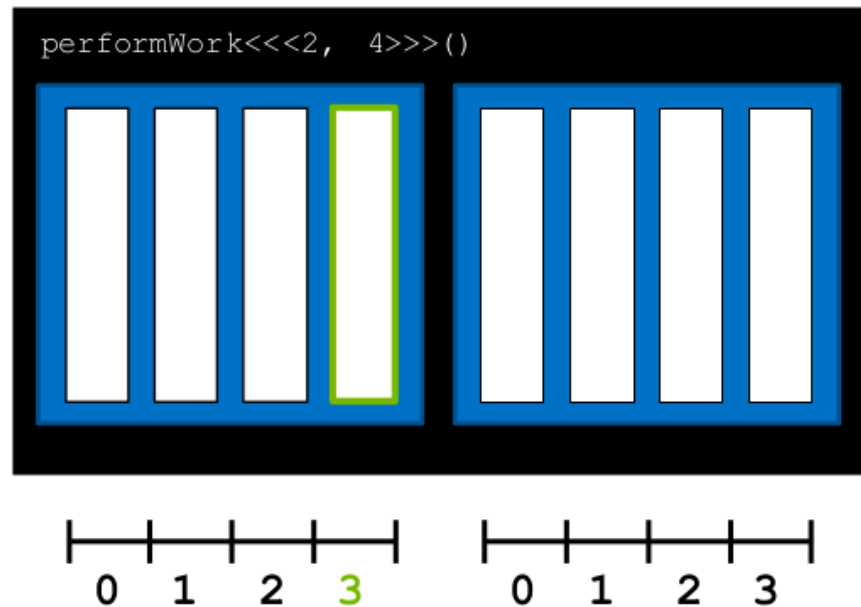
`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку  
`threadIdx.x = 2`

GPU



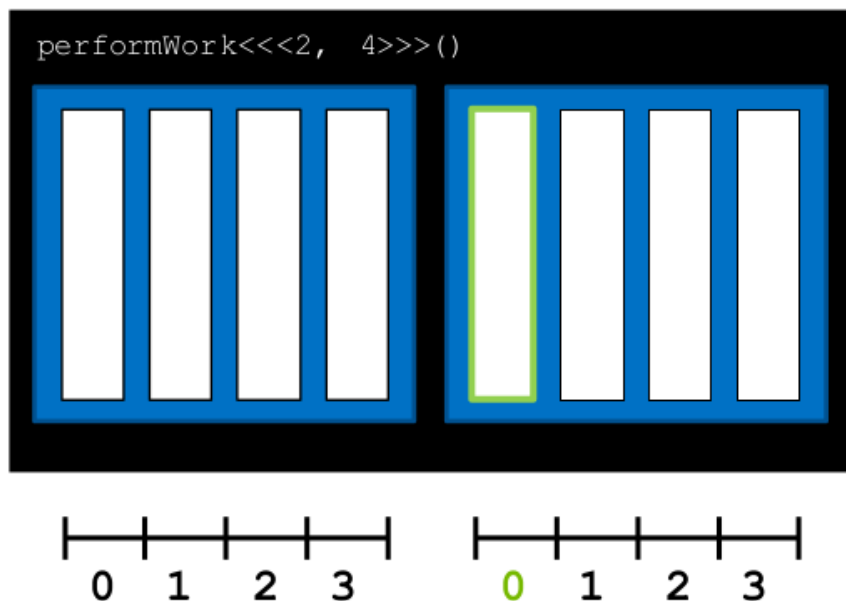
`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку  
`threadIdx.x = 3`

GPU



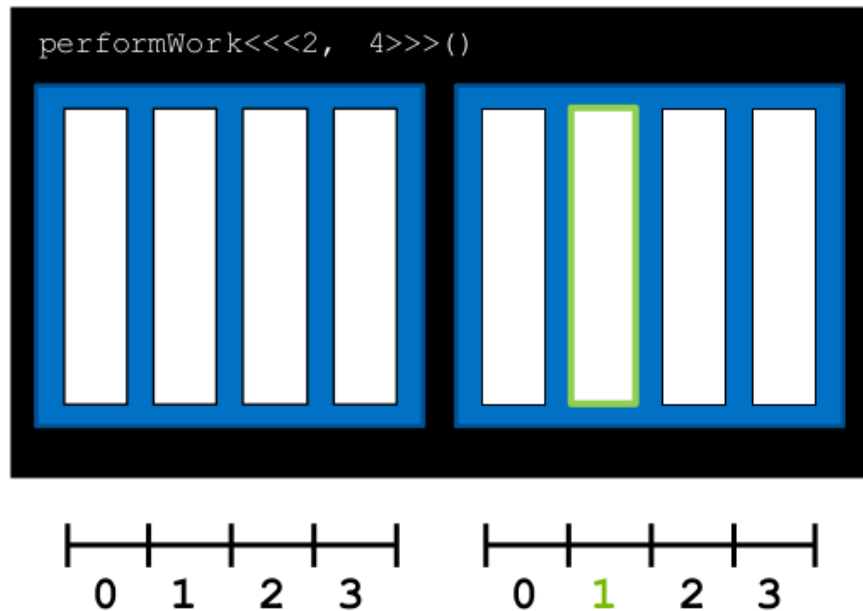
`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку  
`threadIdx.x = 0`

GPU



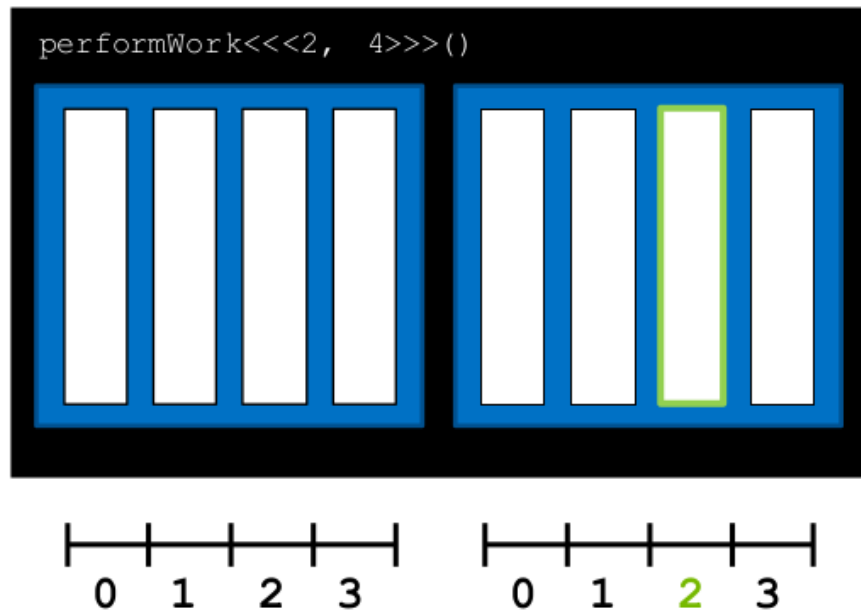
`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку  
`threadIdx.x = 1`

GPU



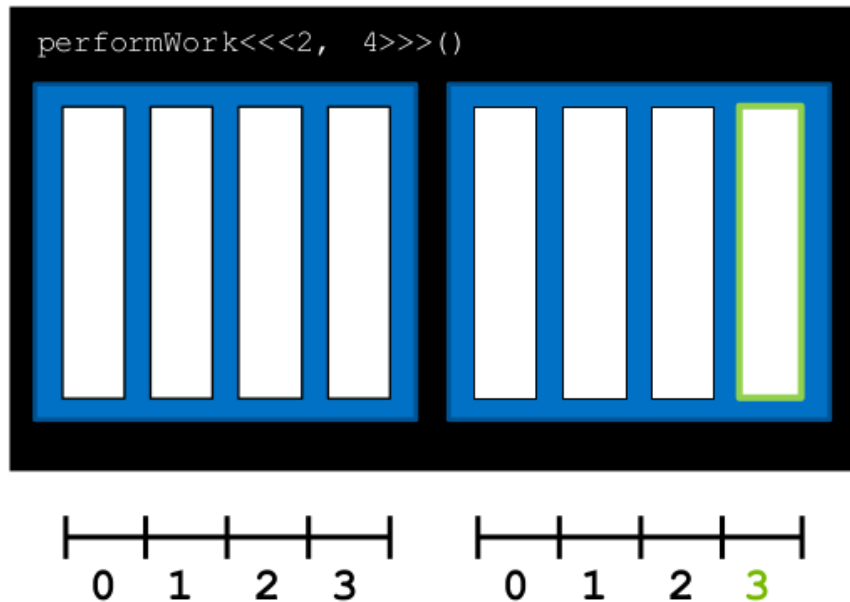
`threadIdx.x` визначає індекс потоку у межах одного блоку, у цьому випадку  
`threadIdx.x = 2`

GPU



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

GPU



# Демо: запрос устройю



# Демо: додавання векторів





# Додаток

- NVIDIA, [CUDA Quick Start Guide](#)

Кінець 