Технології графічного процесінгу & розподілених обчислень

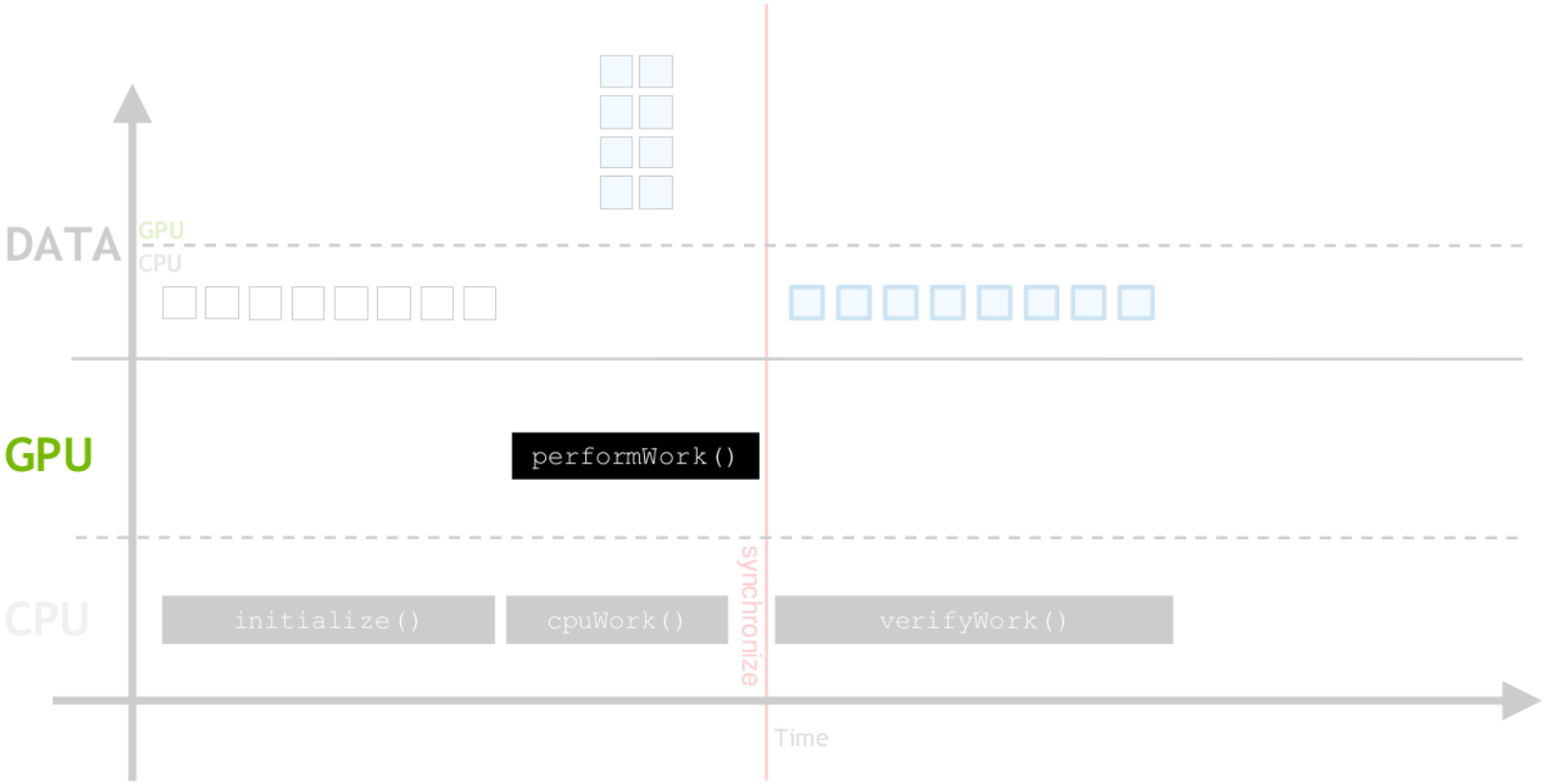
Лекція 3: Вступ до CUDA C II

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

Сьогодні

- Координація паралельних потоків
- Невідповідність розміру сітки
- Цикли: крок за сіткою

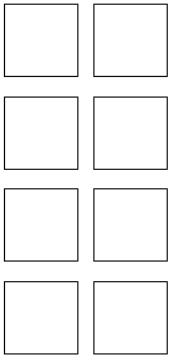
Координація паралельних потоків



—

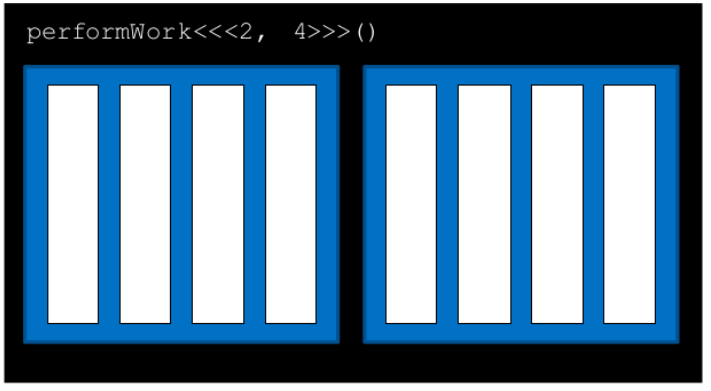
Джерело слайду: NVIDIA, DLI

GPU
DATA



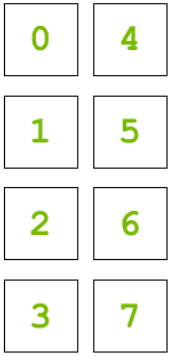
Assume data is in a 0 indexed vector

GPU



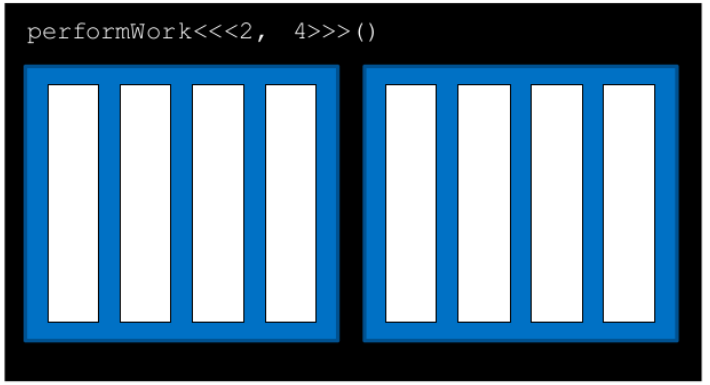
—
Джерело слайду: NVIDIA, DLI

GPU
DATA

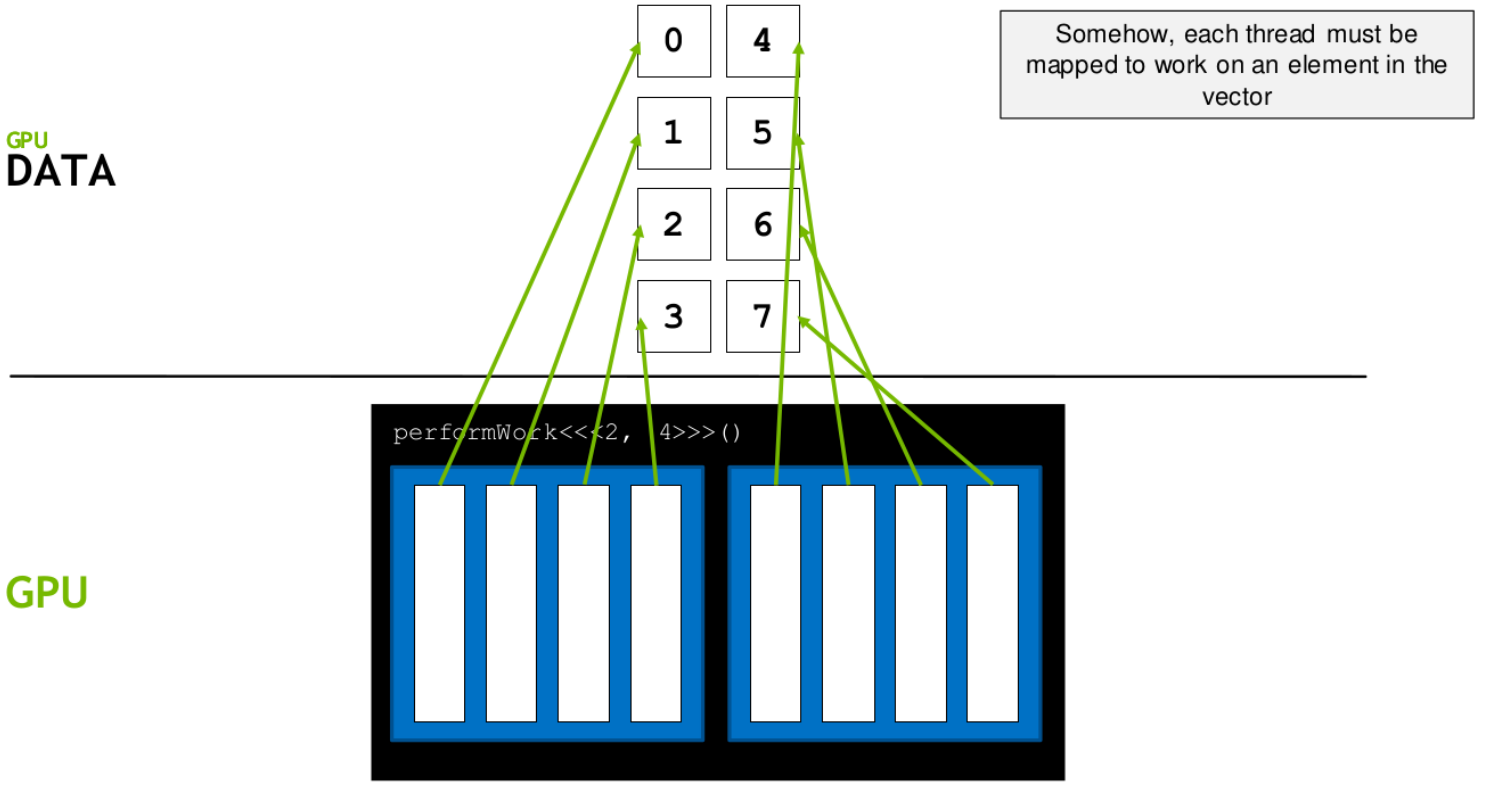


Assume data is in a 0 indexed vector

GPU



—
Джерело слайду: NVIDIA, DLI



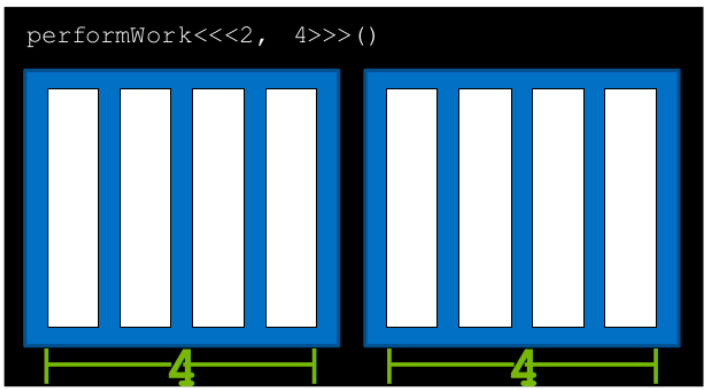
—
Джерело слайду: NVIDIA, DLI

GPU
DATA

0	4
1	5
2	6
3	7

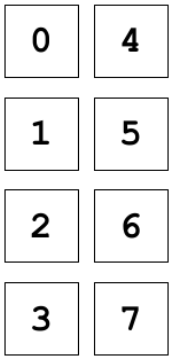
Recall that each thread has access to the size of its block via **`blockDim.x`**

GPU



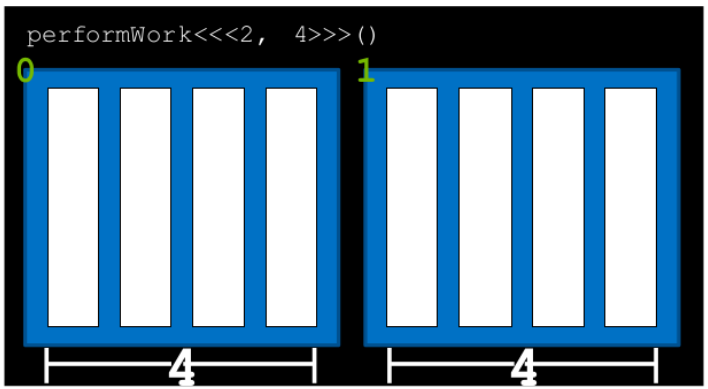
—
Джерело слайду: NVIDIA, DLI

GPU
DATA



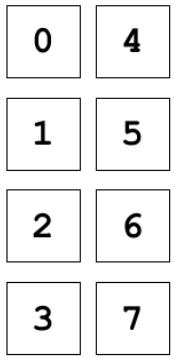
…and the index of its block within the grid via **`blockIdx.x`**

GPU



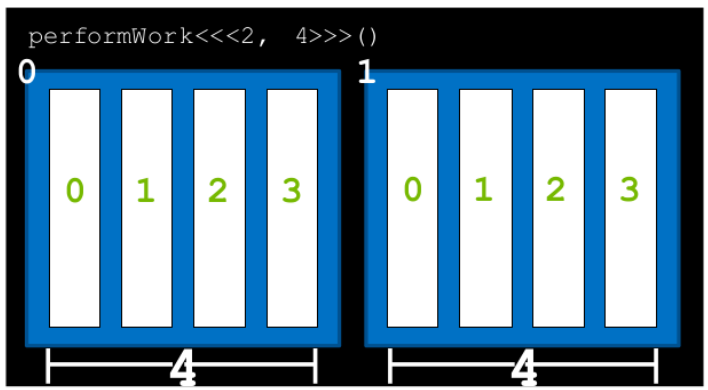
—
Джерело слайду: NVIDIA, DLI

GPU
DATA



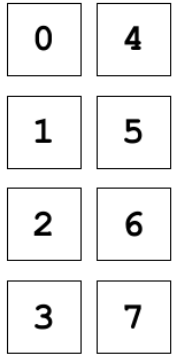
…and its own index within its block via **`threadIdx.x`**

GPU



—
Джерело слайду: NVIDIA, DLI

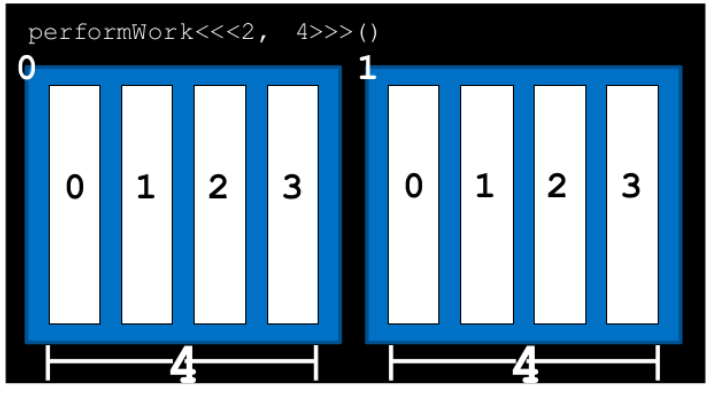
GPU
DATA



Using these variables, the formula **`threadIdx.x + blockIdx.x * blockDim.x`** will map each thread to one element in the vector

---

GPU



—
Джерело слайду: NVIDIA, DLI

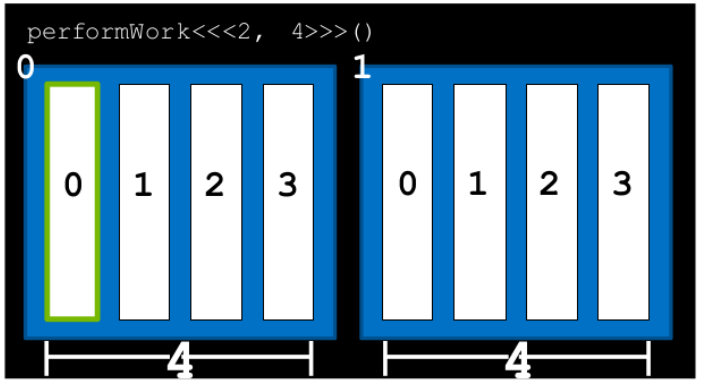
GPU
DATA

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
0		0		4

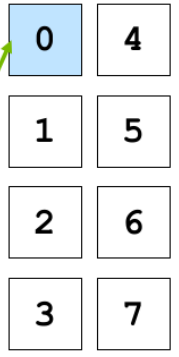
dataIndex
?

GPU



—
Джерело слайду: NVIDIA, DLI

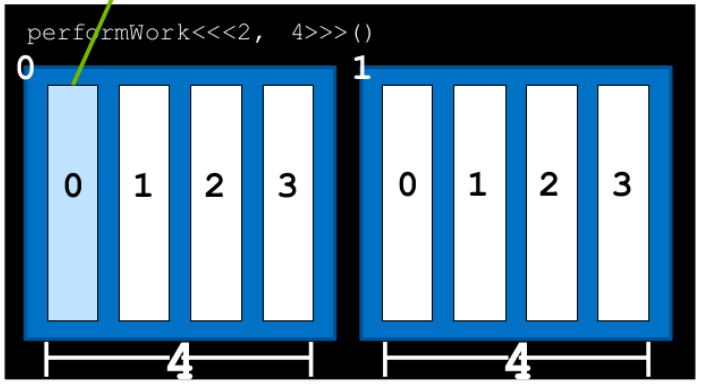
GPU
DATA



threadIdx.x	+	blockIdx.x	*	blockDim.x
0		0		4

dataIndex
0

GPU



—
Джерело слайду: NVIDIA, DLI

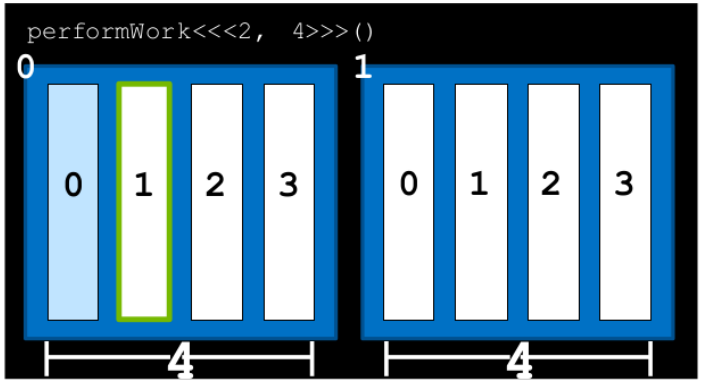
GPU
DATA

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
1		0		4

dataIndex
?

GPU



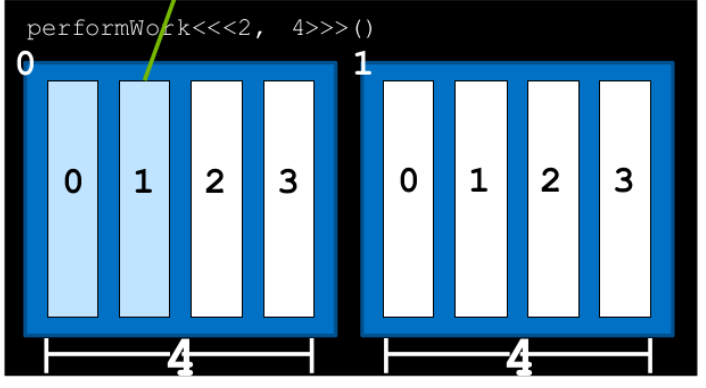
GPU
DATA

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
1		0		4

dataIndex
1

GPU



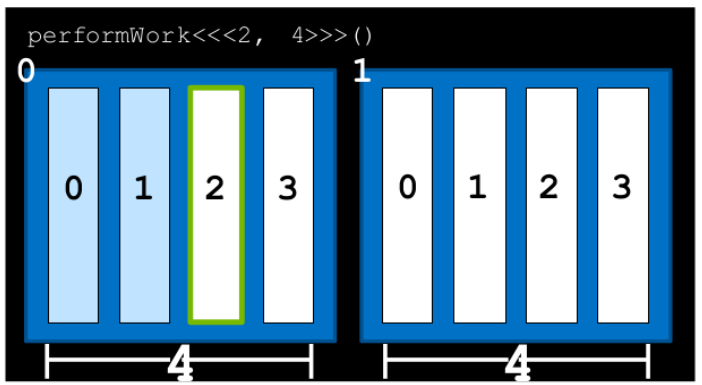
GPU
DATA

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
2		0		4

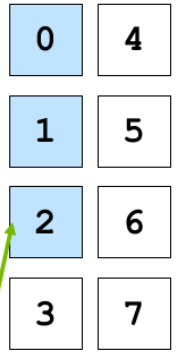
dataIndex
?

GPU



Джерело слайду: NVIDIA, DLI

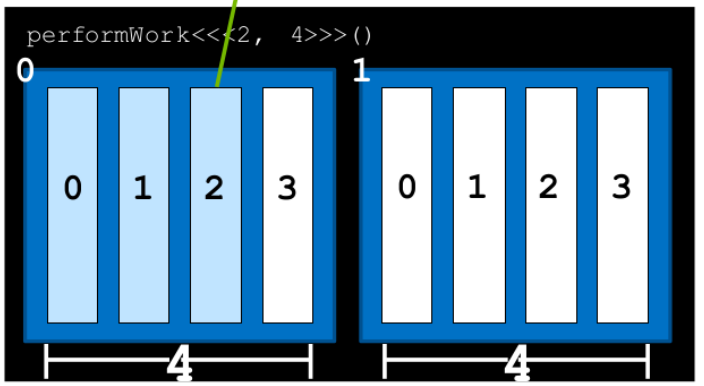
GPU
DATA



threadIdx.x	+	blockIdx.x	*	blockDim.x
2		0		4

dataIndex
2

GPU



—
Джерело слайду: NVIDIA, DLI

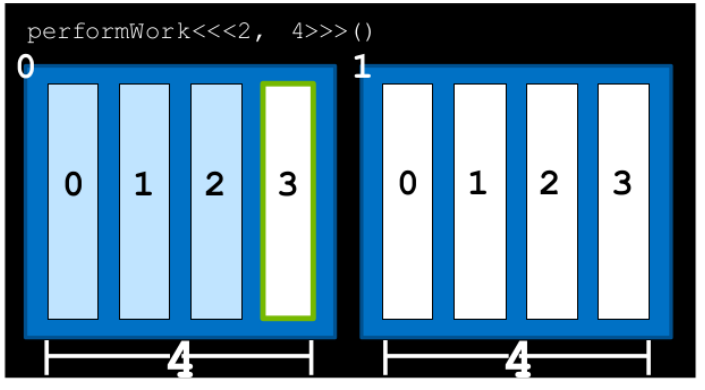
GPU
DATA

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
3		0		4

dataIndex
?

GPU



Джерело слайду: NVIDIA, DLI

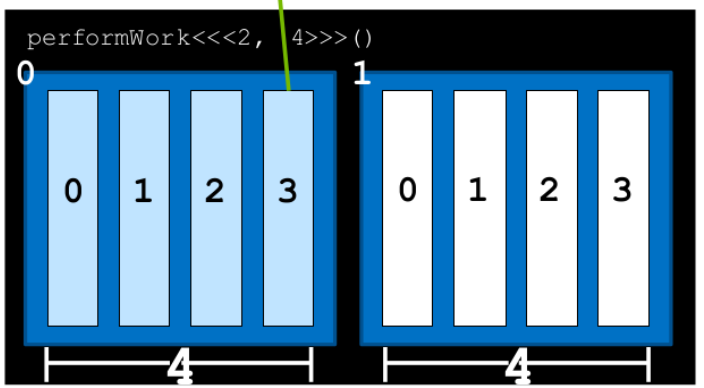
GPU
DATA

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
3		0		4

dataIndex
3

GPU



—
Джерело слайду: NVIDIA, DLI

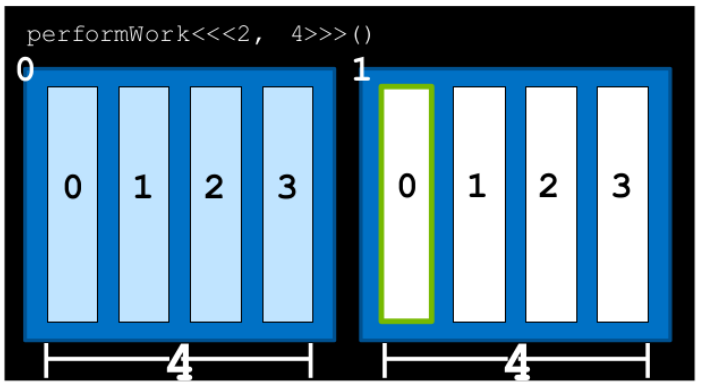
GPU
DATA

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
0		1		4

dataIndex
?

GPU



—
Джерело слайду: NVIDIA, DLI

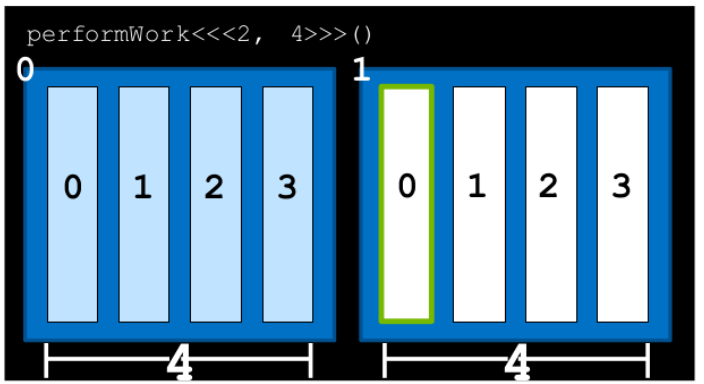
GPU
DATA

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
0		1		4

dataIndex
?

GPU



—
Джерело слайду: NVIDIA, DLI

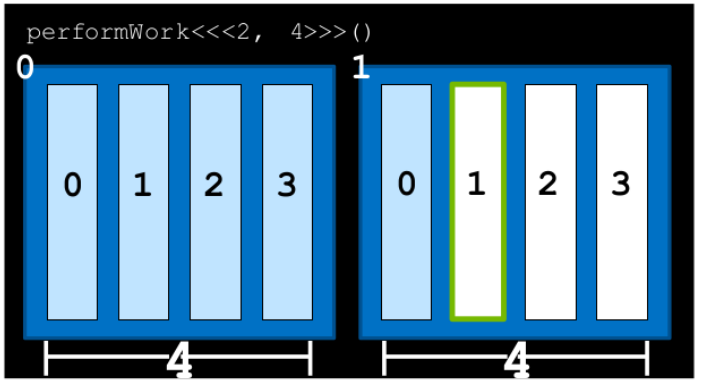
GPU
DATA

0	4
1	5
2	6
3	7

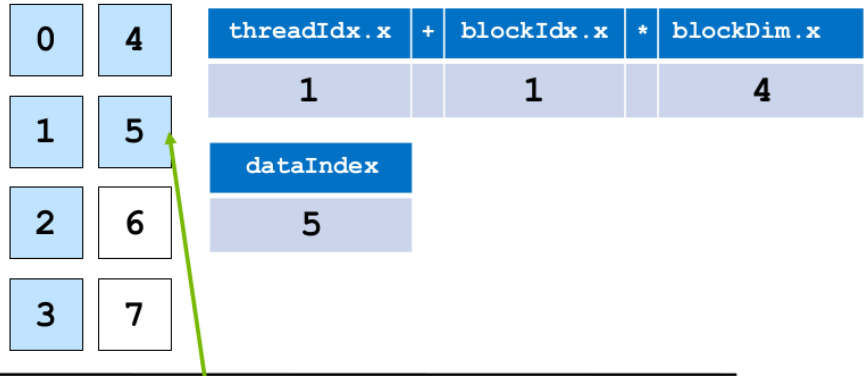
threadIdx.x	+	blockIdx.x	*	blockDim.x
1		1		4

dataIndex
?

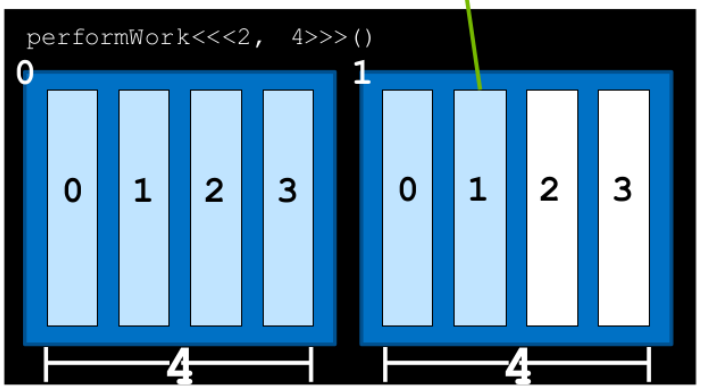
GPU



GPU
DATA



GPU



—
Джерело слайду: NVIDIA, DLI

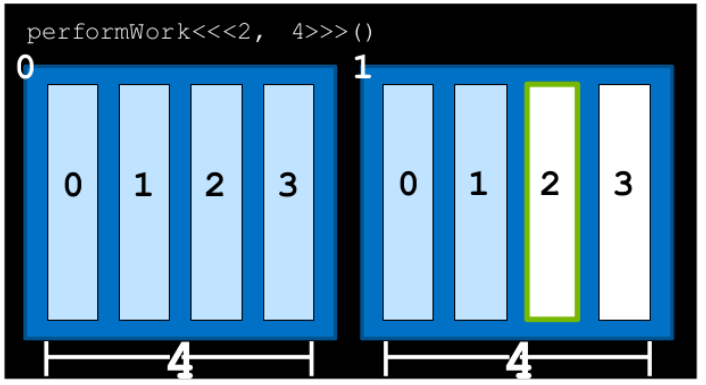
GPU
DATA

0	4
1	5
2	6
3	7

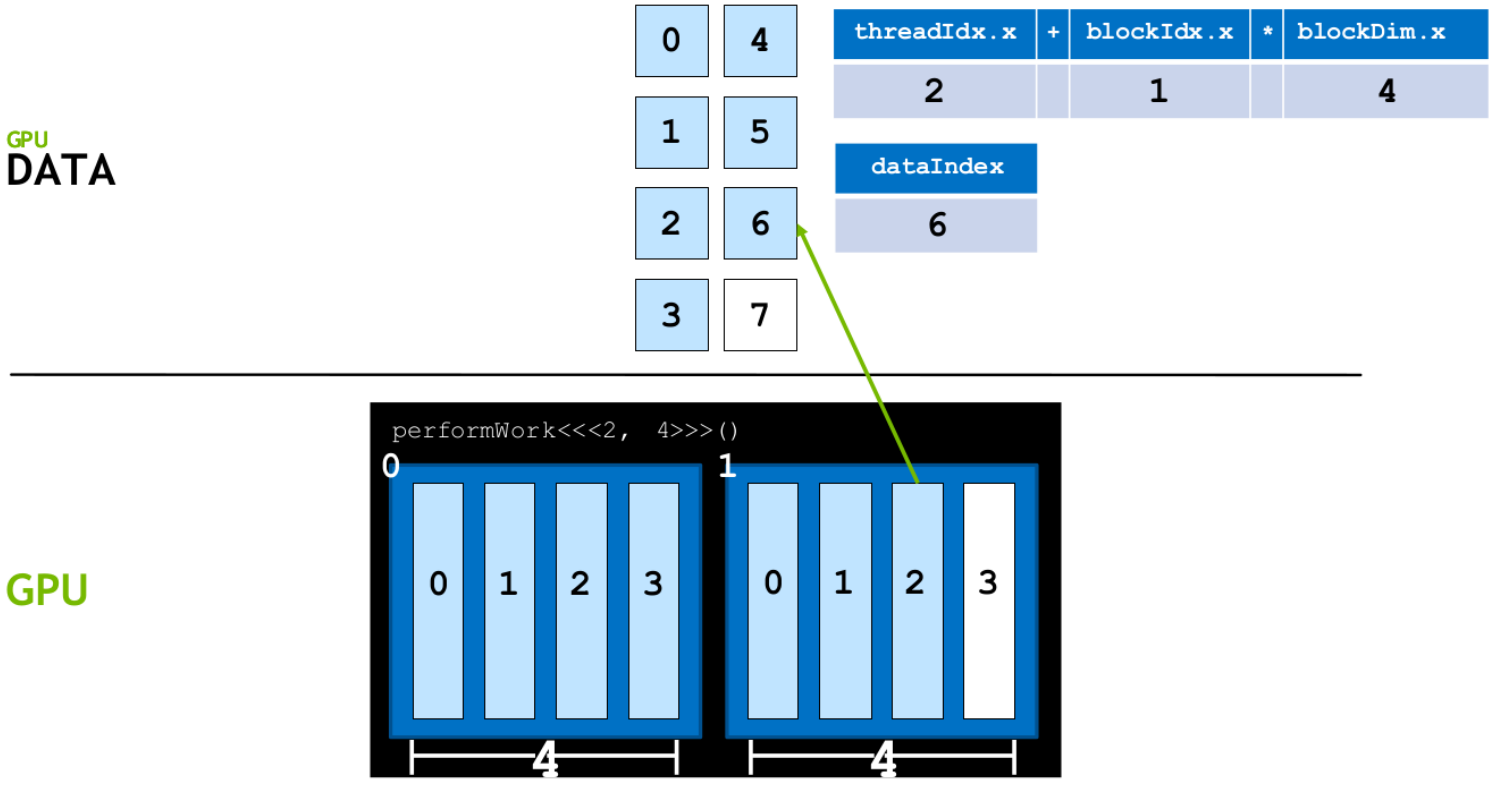
threadIdx.x	+	blockIdx.x	*	blockDim.x
2		1		4

dataIndex
?

GPU



—
Джерело слайду: NVIDIA, DLI



—

Джерело слайду: NVIDIA, DLI

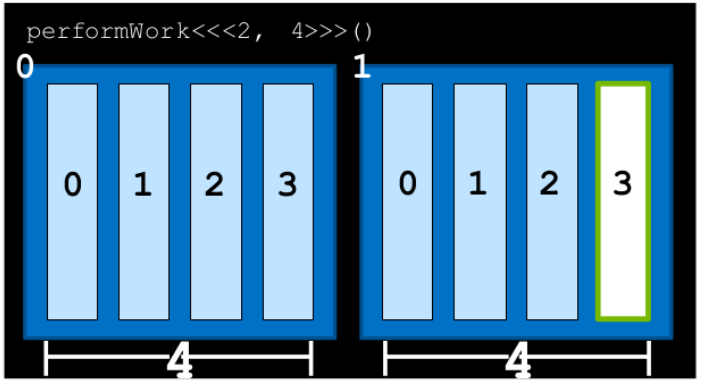
GPU
DATA

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
3		1		4

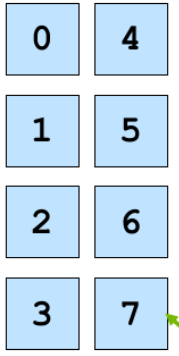
dataIndex
?

GPU



Джерело слайду: NVIDIA, DLI

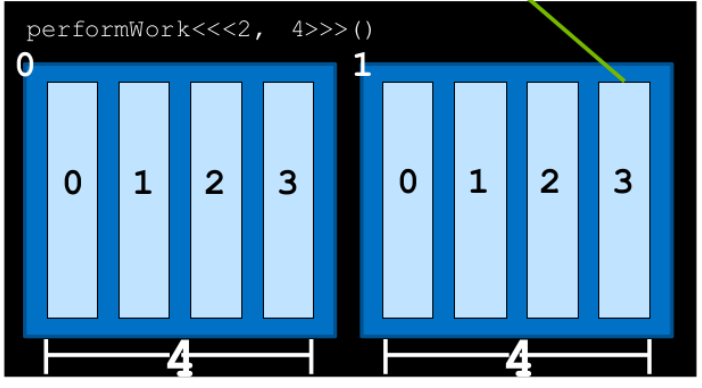
GPU
DATA



threadIdx.x	+	blockIdx.x	*	blockDim.x
3		1		4

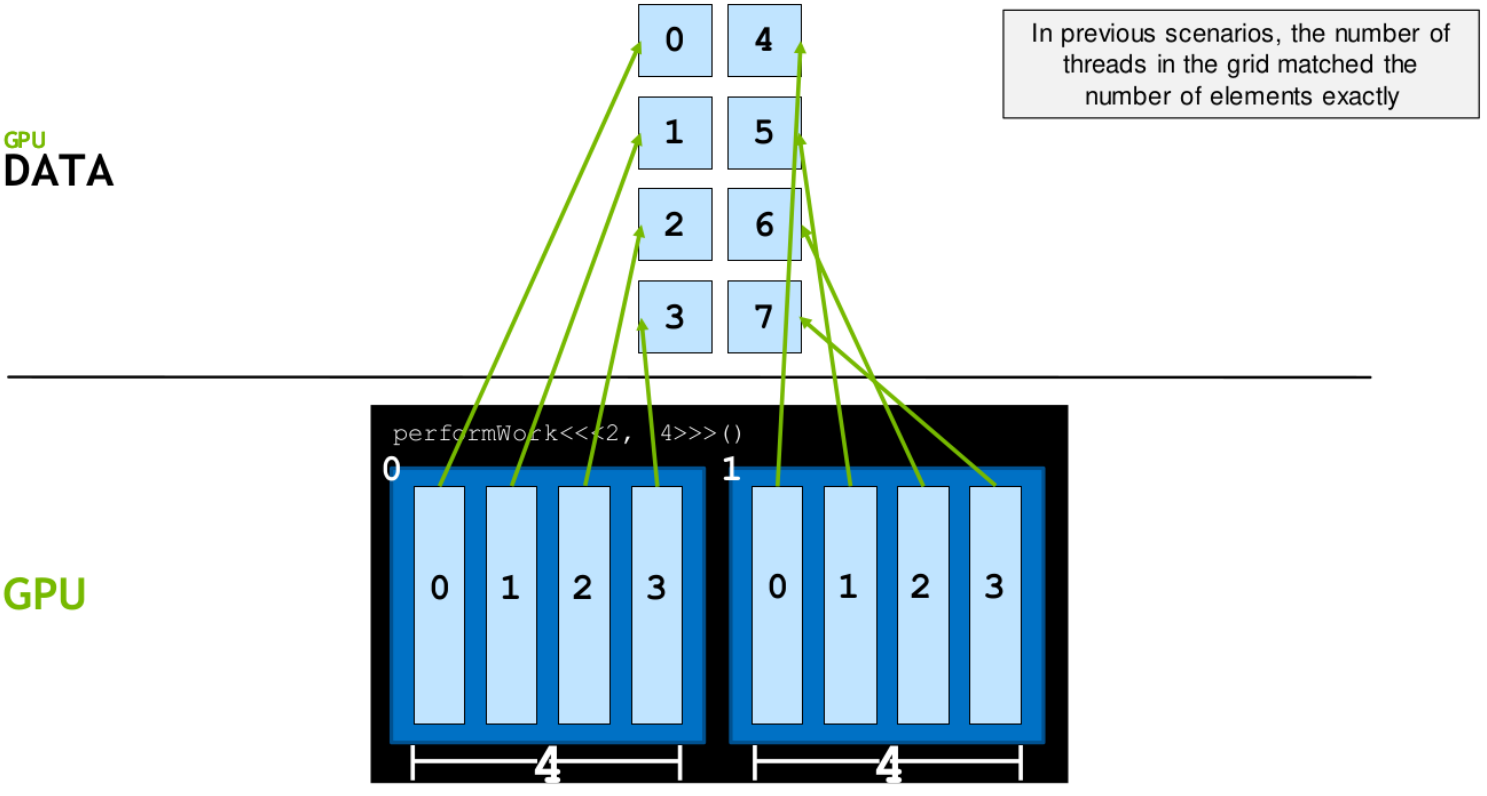
dataIndex
7

GPU



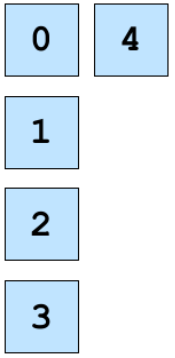
—
Джерело слайду: NVIDIA, DLI

Невідповідність розміру сітки



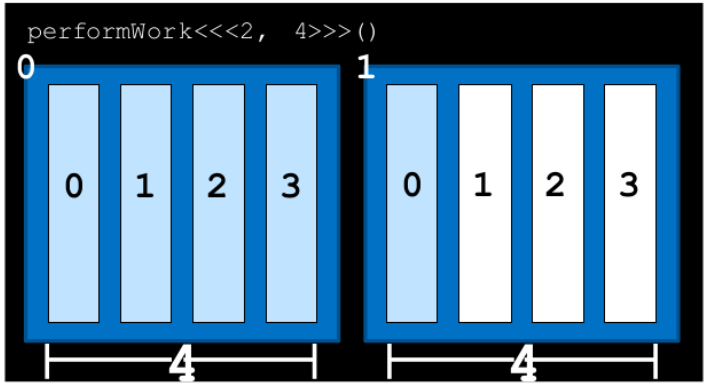
Джерело слайду: NVIDIA, DLI

GPU
DATA

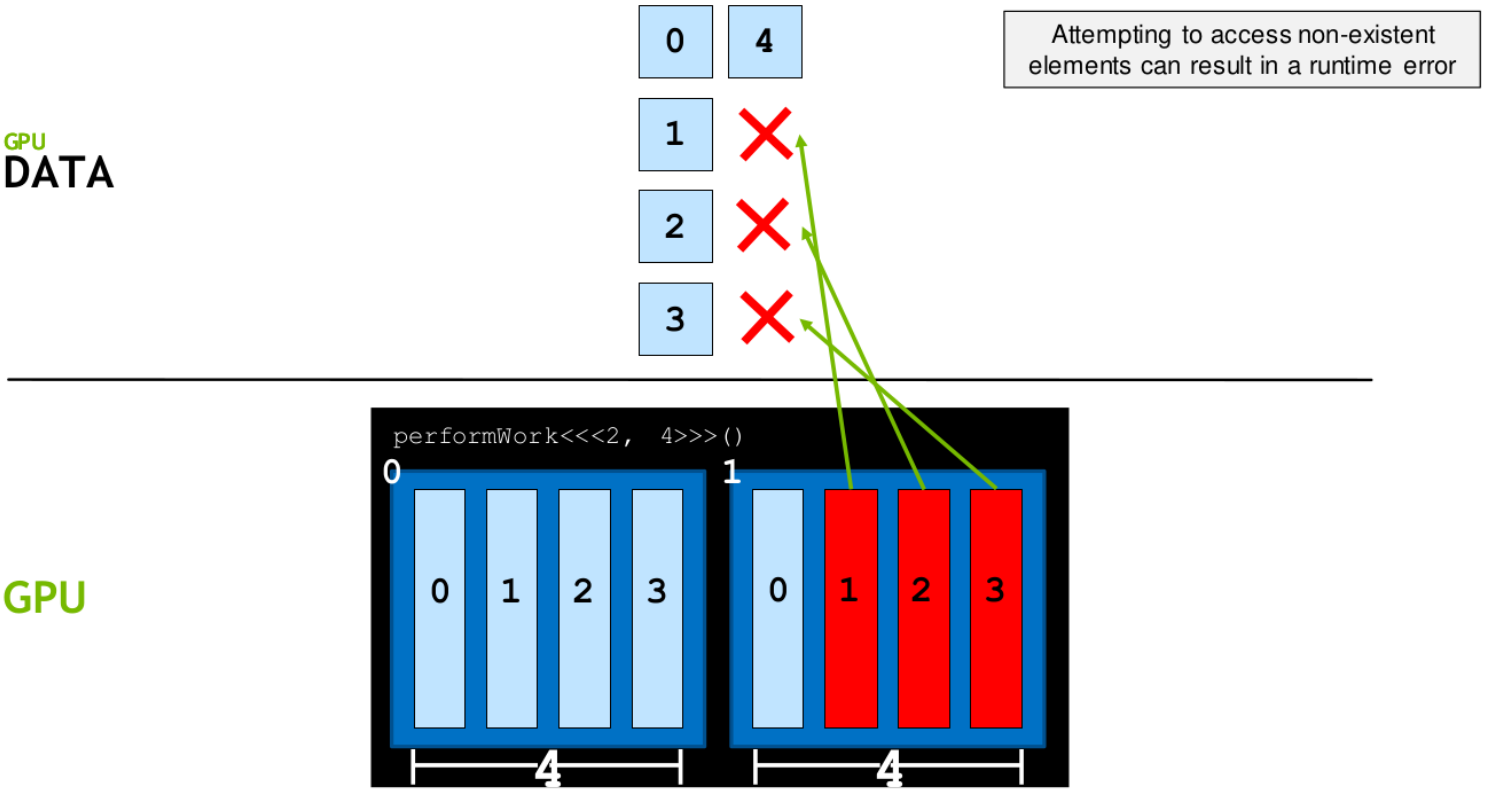


What if there are more threads than work to be done?

GPU

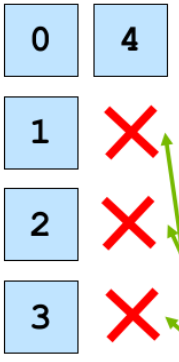


—
Джерело слайду: NVIDIA, DLI



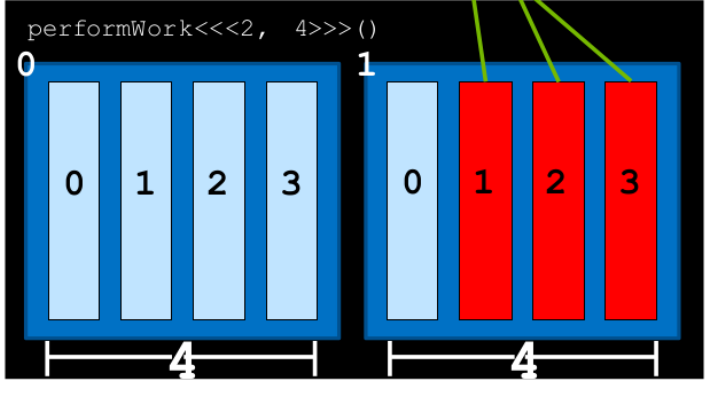
Джерело слайду: NVIDIA, DLI

GPU
DATA

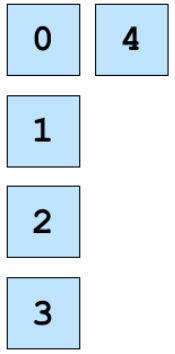


Code must check that the **dataIndex** calculated by **threadIdx.x + blockIdx.x * blockDim.x** is less than **N**, the number of data elements.

GPU



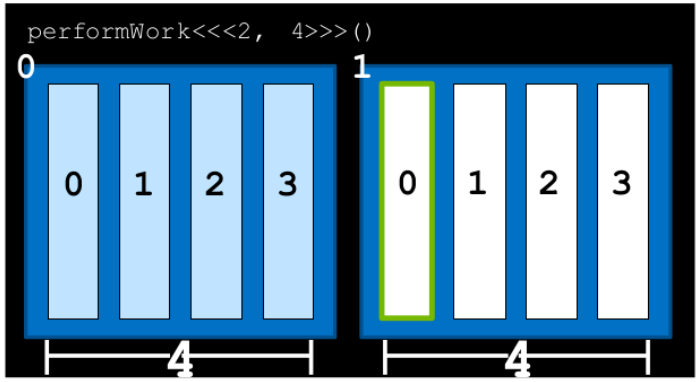
GPU
DATA



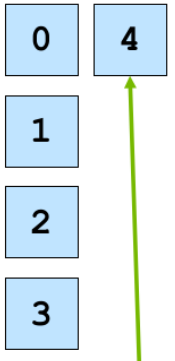
threadIdx.x	+	blockIdx.x	*	blockDim.x
0		1		4

dataIndex	<	N	=	Can work
4		5		?

GPU



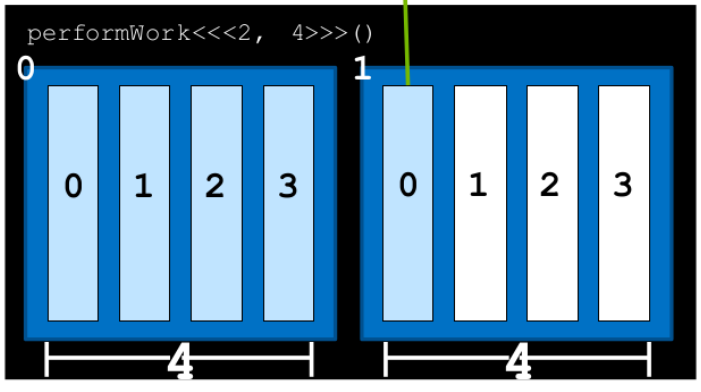
GPU
DATA



threadIdx.x	+	blockIdx.x	*	blockDim.x
0		1		4

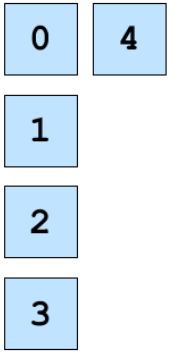
dataIndex	<	N	=	Can work
4		5		true

GPU



—
Джерело слайду: NVIDIA, DLI

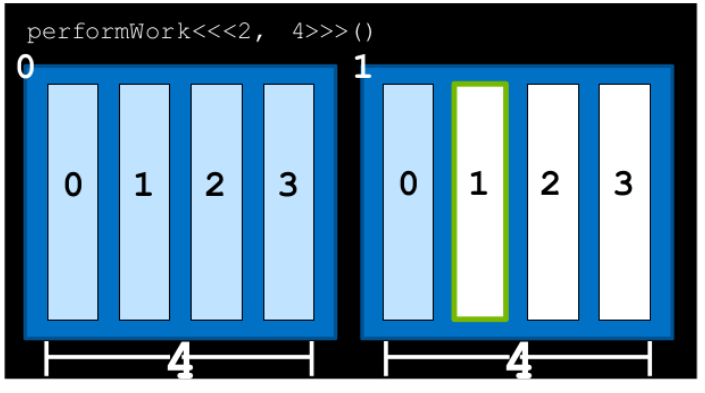
GPU
DATA



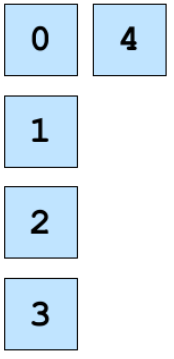
threadIdx.x	+	blockIdx.x	*	blockDim.x
1		1		4

dataIndex	<	N	=	Can work
5		5		?

GPU



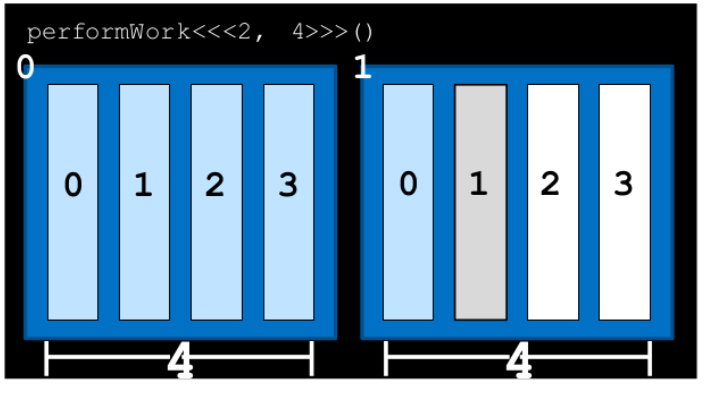
GPU
DATA



threadIdx.x	+	blockIdx.x	*	blockDim.x
1		1		4

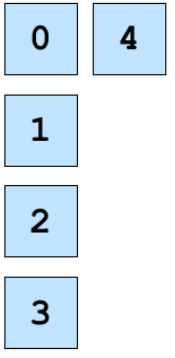
dataIndex	<	N	=	Can work
5		5		false

GPU



—
Джерело слайду: NVIDIA, DLI

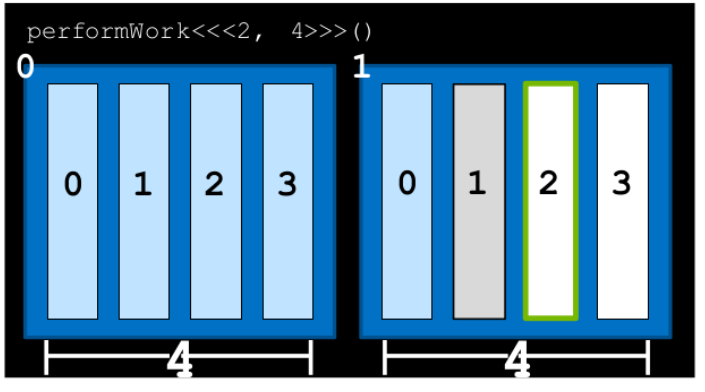
GPU
DATA



threadIdx.x	+	blockIdx.x	*	blockDim.x
2		1		4

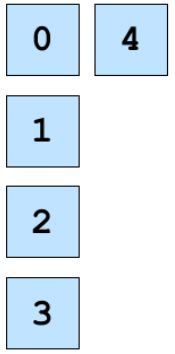
dataIndex	<	N	=	Can work
6		5		?

GPU



—
Джерело слайду: NVIDIA, DLI

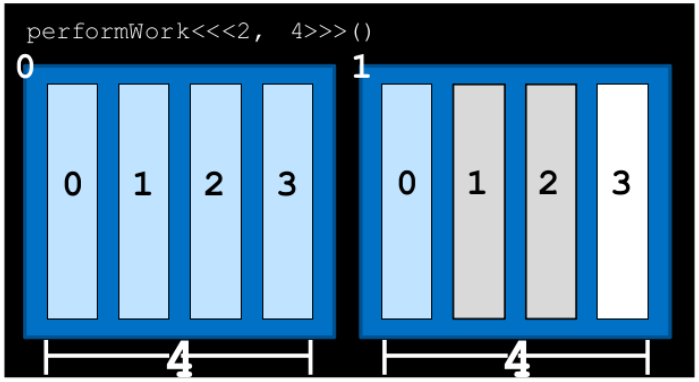
GPU
DATA



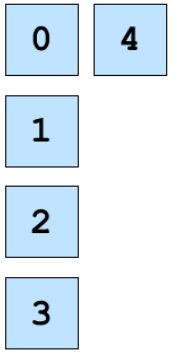
threadIdx.x	+	blockIdx.x	*	blockDim.x
2		1		4

dataIndex	<	N	=	Can work
6		5		false

GPU



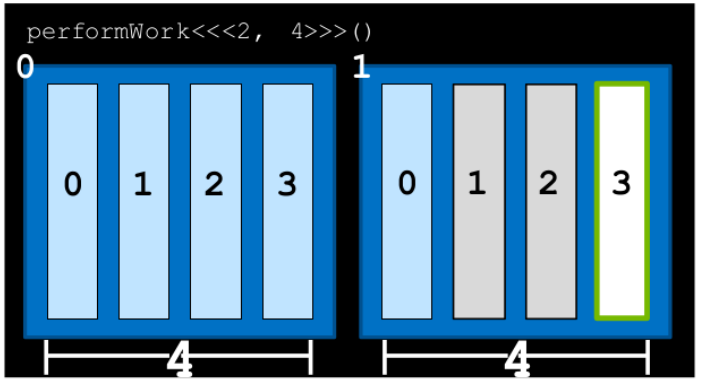
GPU
DATA



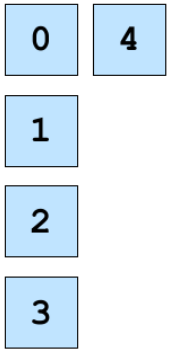
threadIdx.x	+	blockIdx.x	*	blockDim.x
3		1		4

dataIndex	<	N	=	Can work
7		5		?

GPU



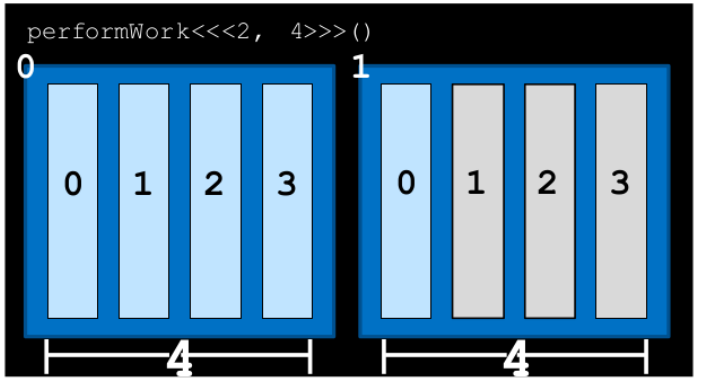
GPU
DATA



threadIdx.x	+	blockIdx.x	*	blockDim.x
3		1		4

dataIndex	<	N	=	Can work
7		5		false

GPU



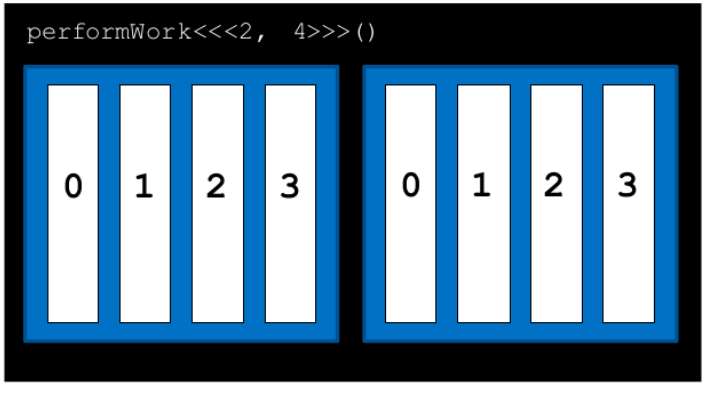
Цикли: крок за сіткою

GPU
DATA

0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

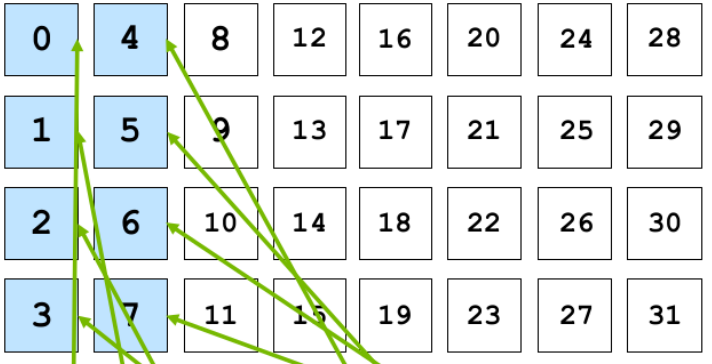
Often there are more data elements than there are threads in the grid

GPU



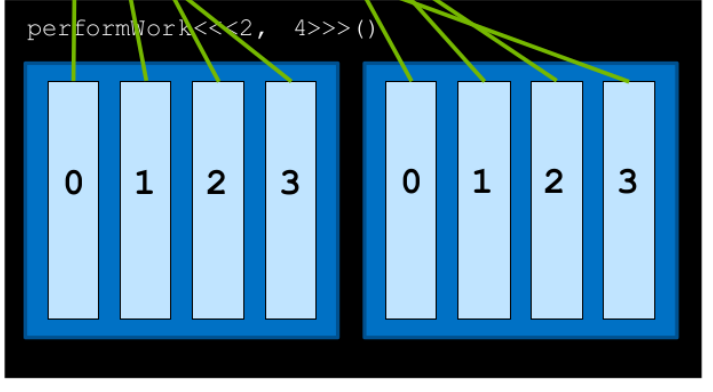
—
Джерело слайду: NVIDIA, DLI

GPU
DATA



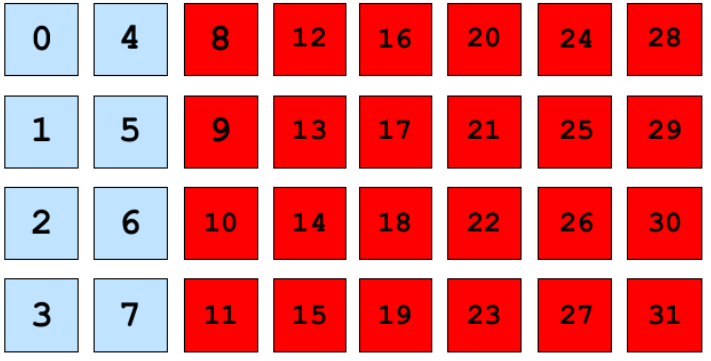
In such scenarios threads cannot work on only one element

GPU



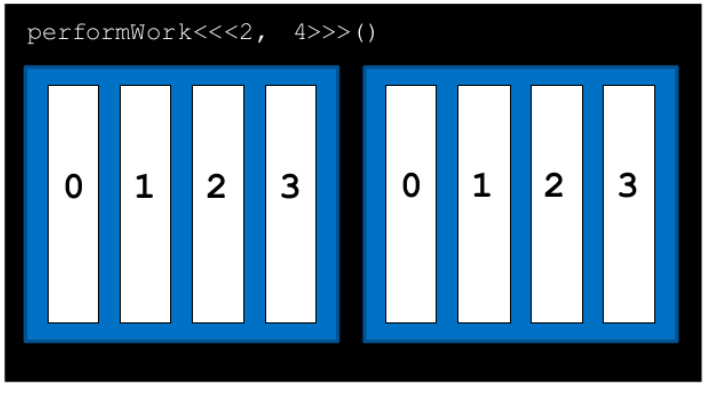
—
Джерело слайду: NVIDIA, DLI

GPU
DATA



… or else work is left undone

GPU



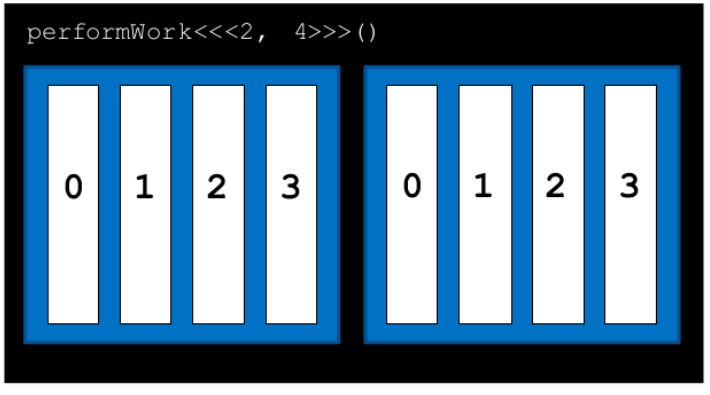
—
Джерело слайду: NVIDIA, DLI

GPU
DATA

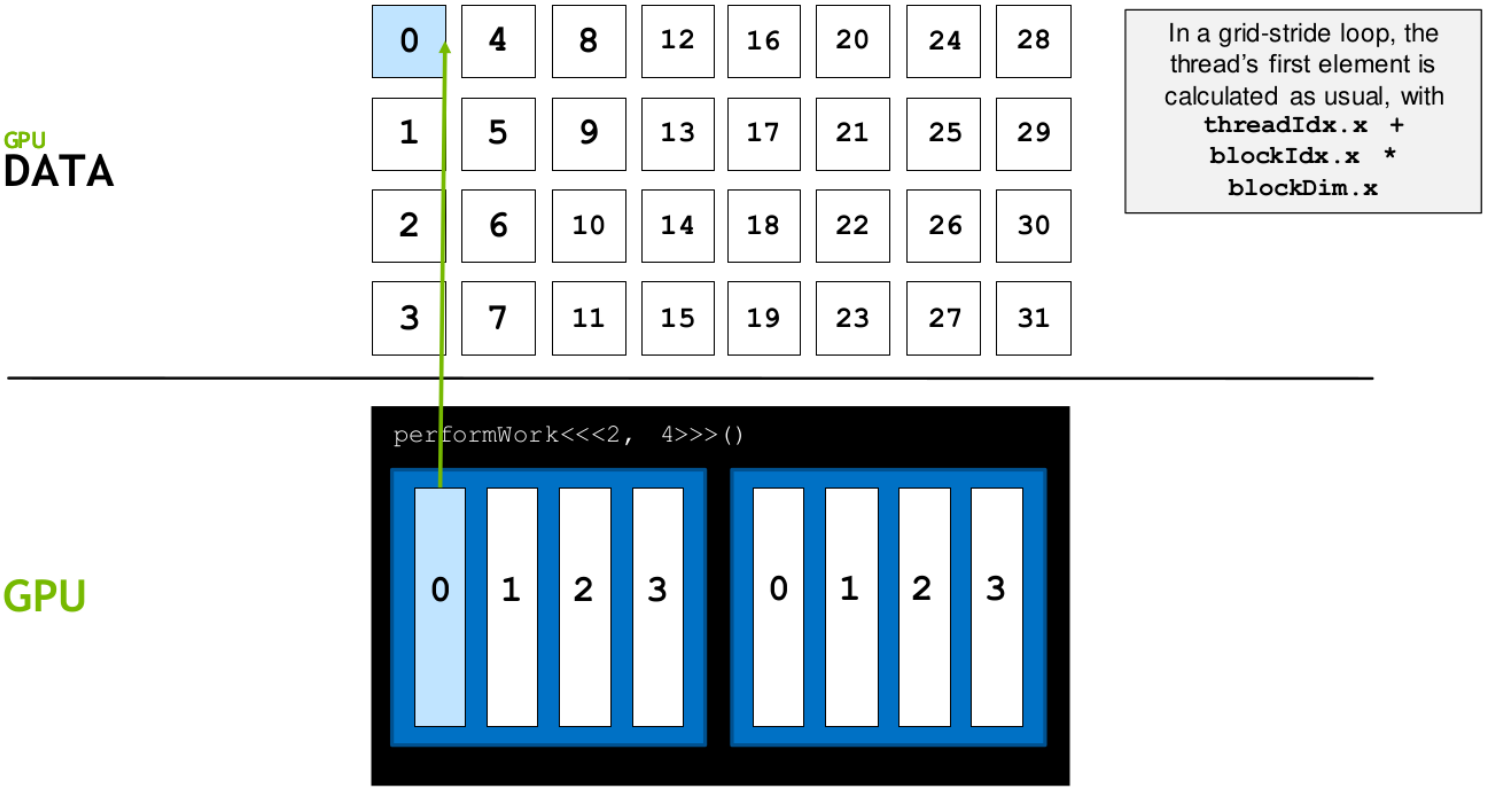
0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

One way to address this programmatically is with a **grid-stride loop**

GPU

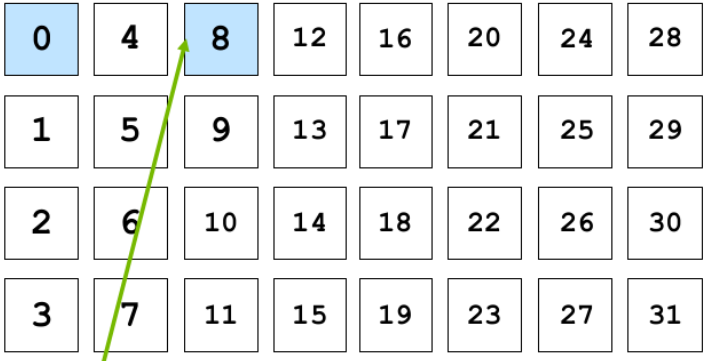


—
Джерело слайду: NVIDIA, DLI



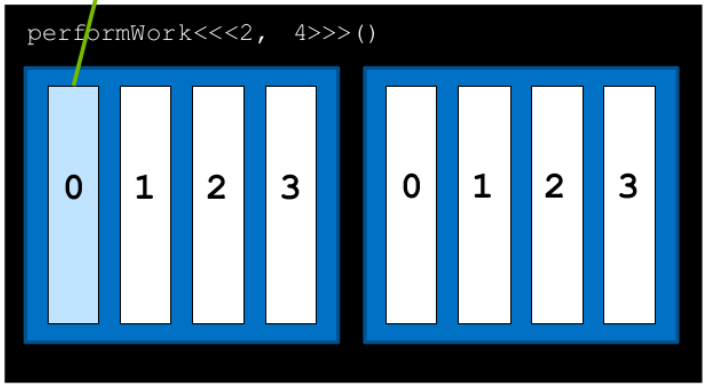
Джерело слайду: NVIDIA, DLI

GPU
DATA

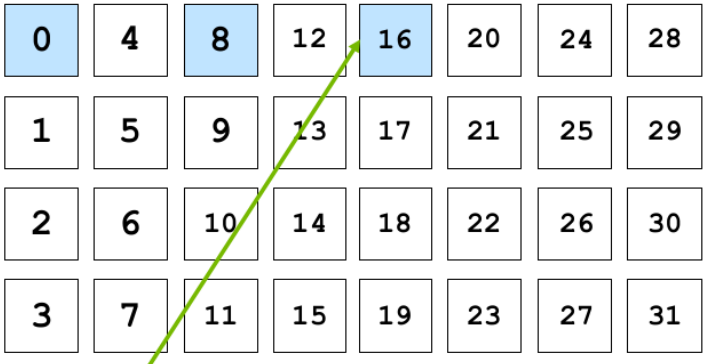


The thread then strides forward by the number of threads in the grid (`blockDim.x * gridDim.x`), in this case `8`

GPU

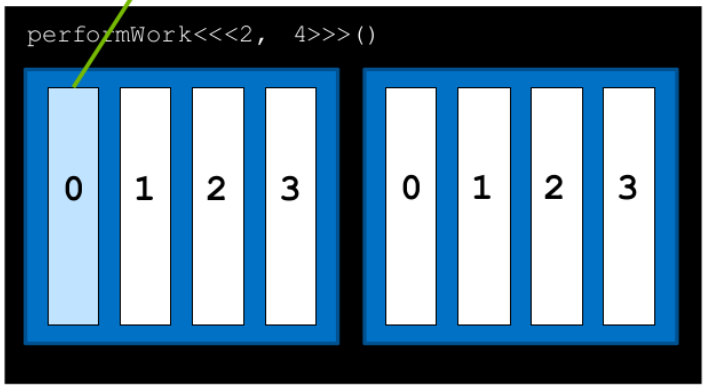


GPU
DATA



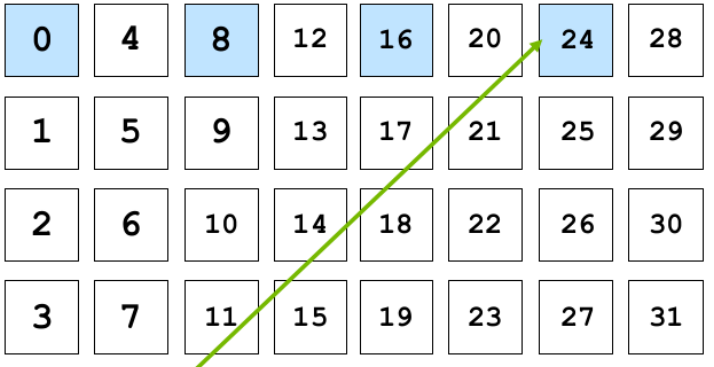
It continues in this way until its data index is greater than the number of data elements

GPU



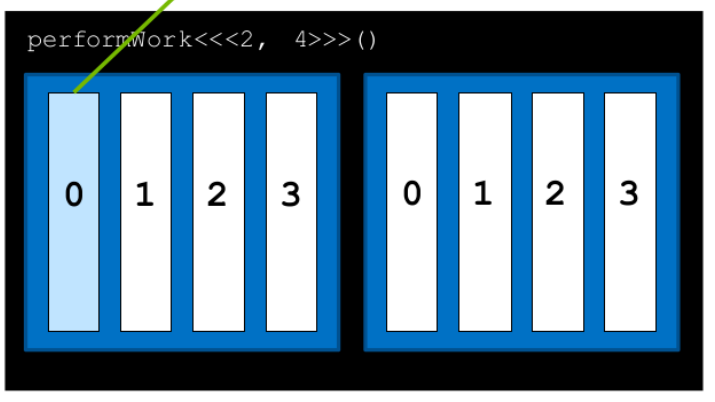
—
Джерело слайду: NVIDIA, DLI

GPU
DATA



It continues in this way until its data index is greater than the number of data elements

GPU



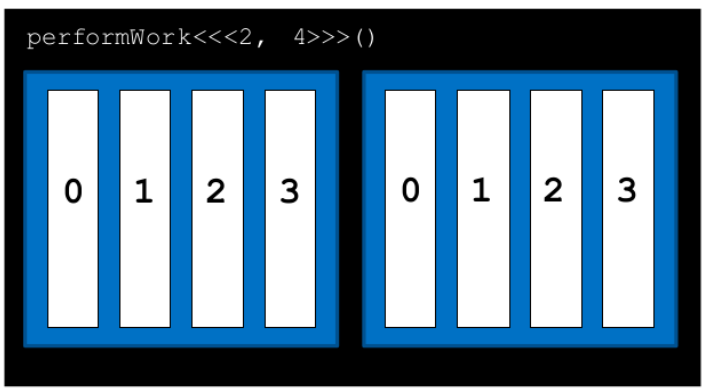
—
Джерело слайду: NVIDIA, DLI

GPU
DATA

0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

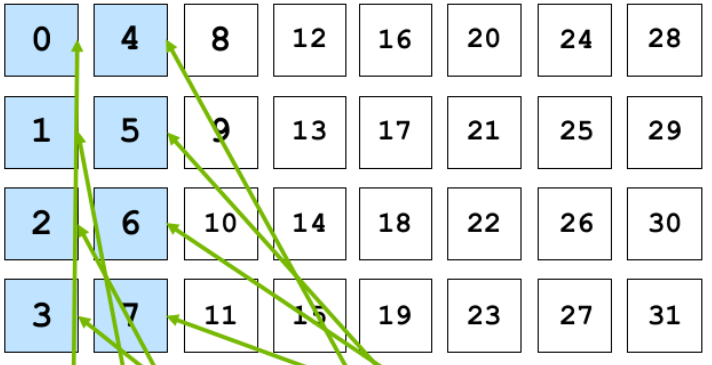
With all threads working in this way, all elements are covered

GPU



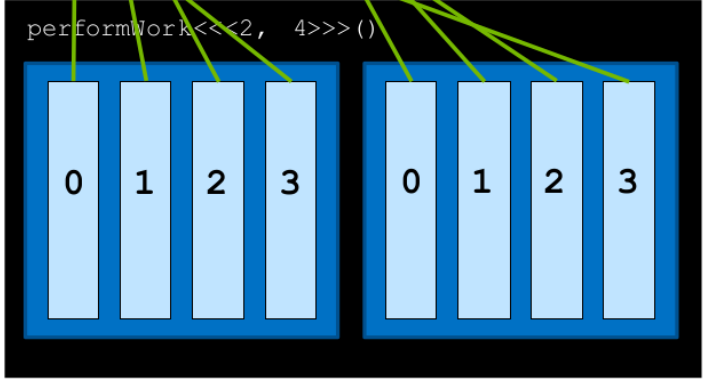
—
Джерело слайду: NVIDIA, DLI

GPU
DATA



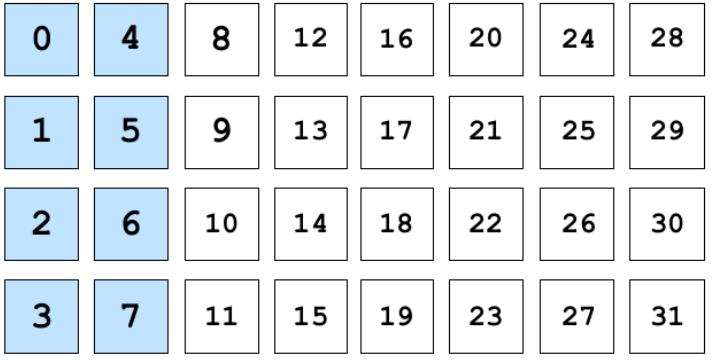
With all threads working in this way, all elements are covered

GPU



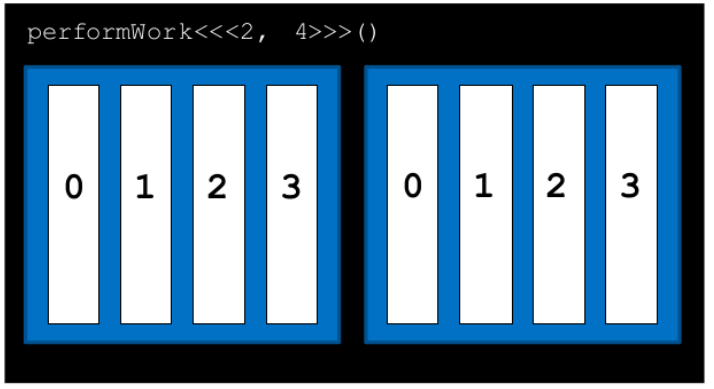
—
Джерело слайду: NVIDIA, DLI

GPU
DATA



With all threads working in this way, all elements are covered

GPU

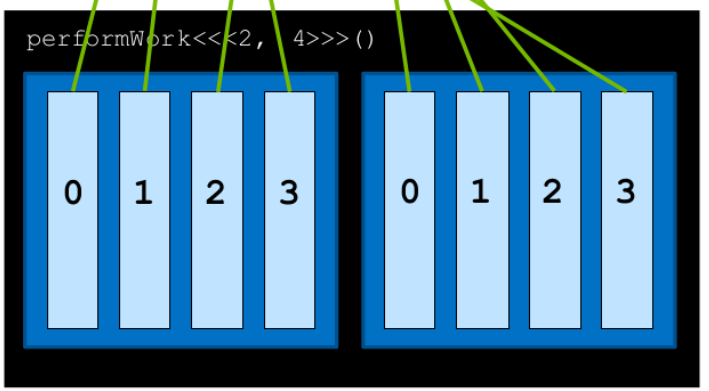


—
Джерело слайду: NVIDIA, DLI

GPU
DATA

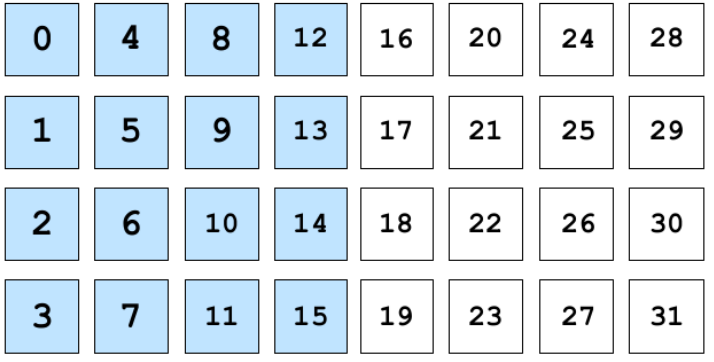
With all threads working in this way, all elements are covered

GPU



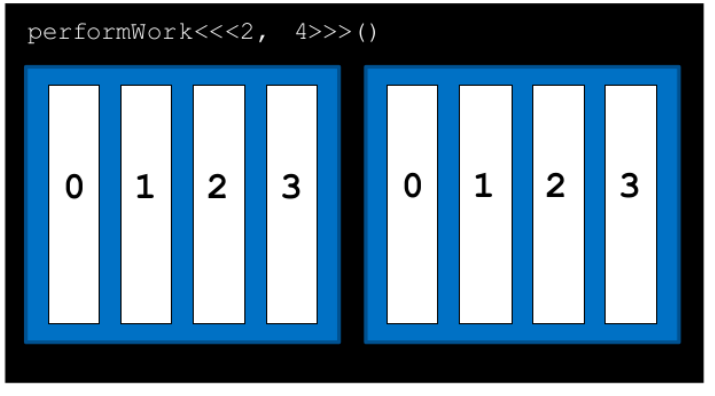
—
Джерело слайду: NVIDIA, DLI

GPU
DATA



With all threads working in this way, all elements are covered

GPU

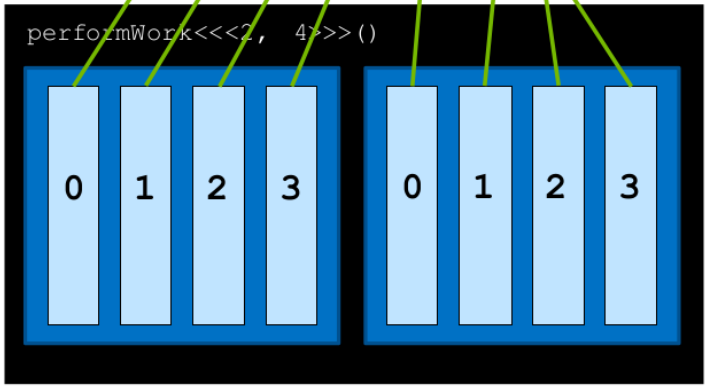


—
Джерело слайду: NVIDIA, DLI

GPU
DATA

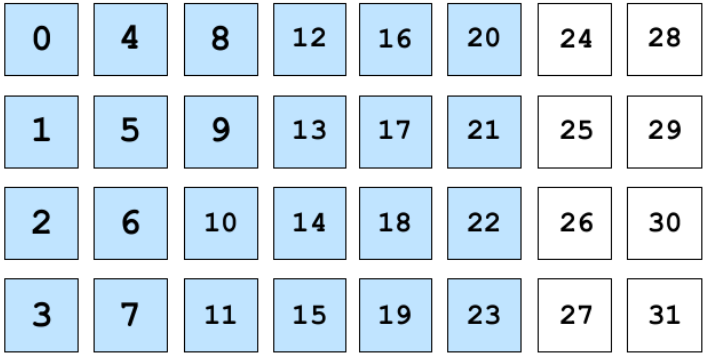
With all threads working in this way, all elements are covered

GPU



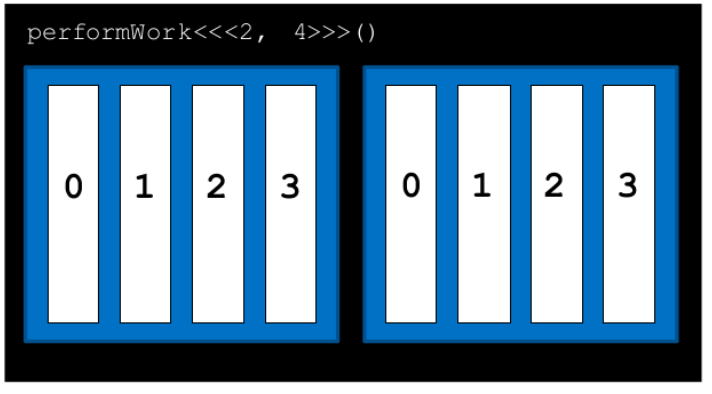
—
Джерело слайду: NVIDIA, DLI

GPU
DATA

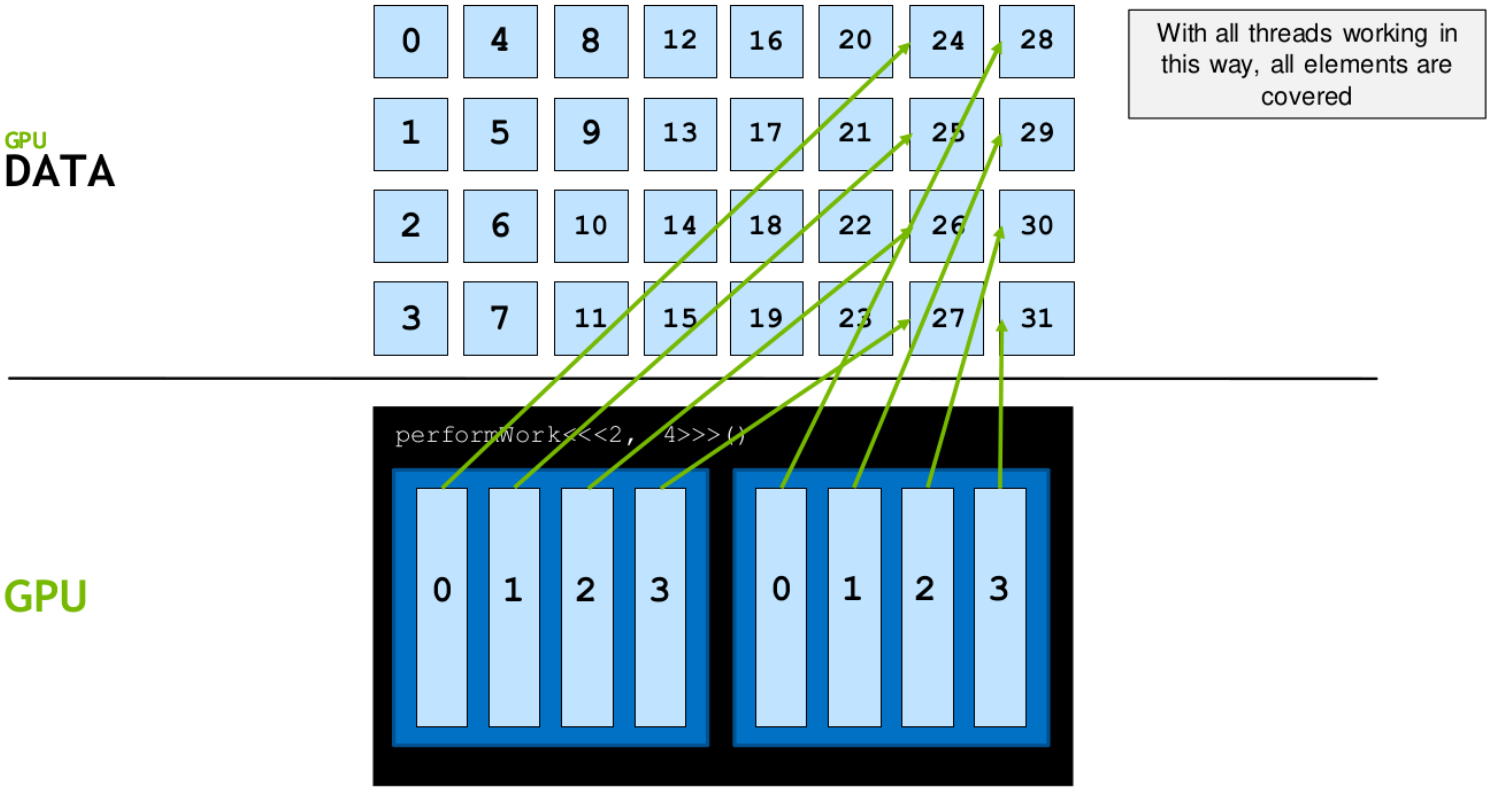


With all threads working in this way, all elements are covered

GPU



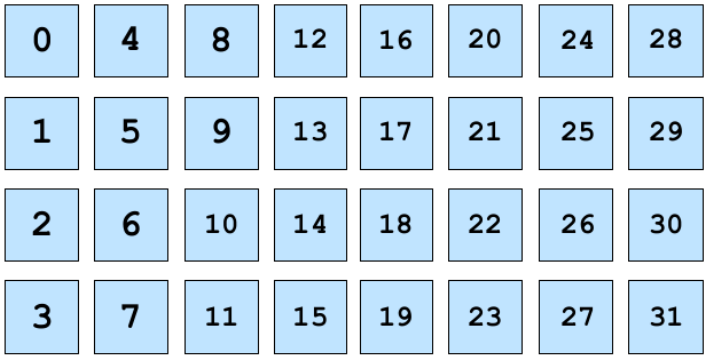
—
Джерело слайду: NVIDIA, DLI



—

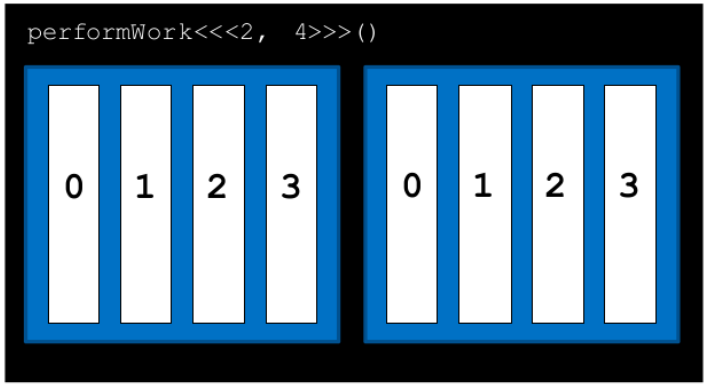
Джерело слайду: NVIDIA, DLI

GPU
DATA

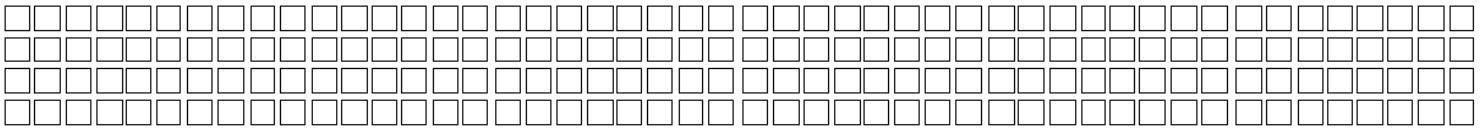


With all threads working in this way, all elements are covered

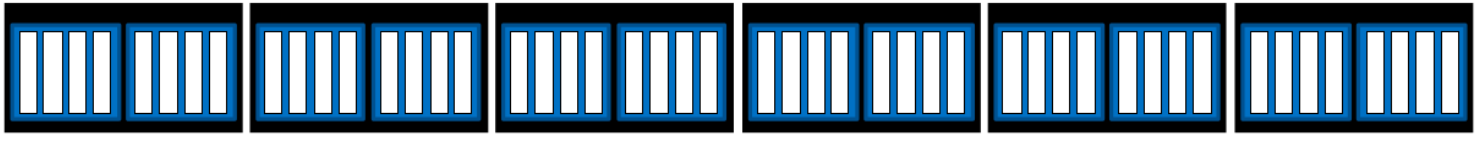
GPU



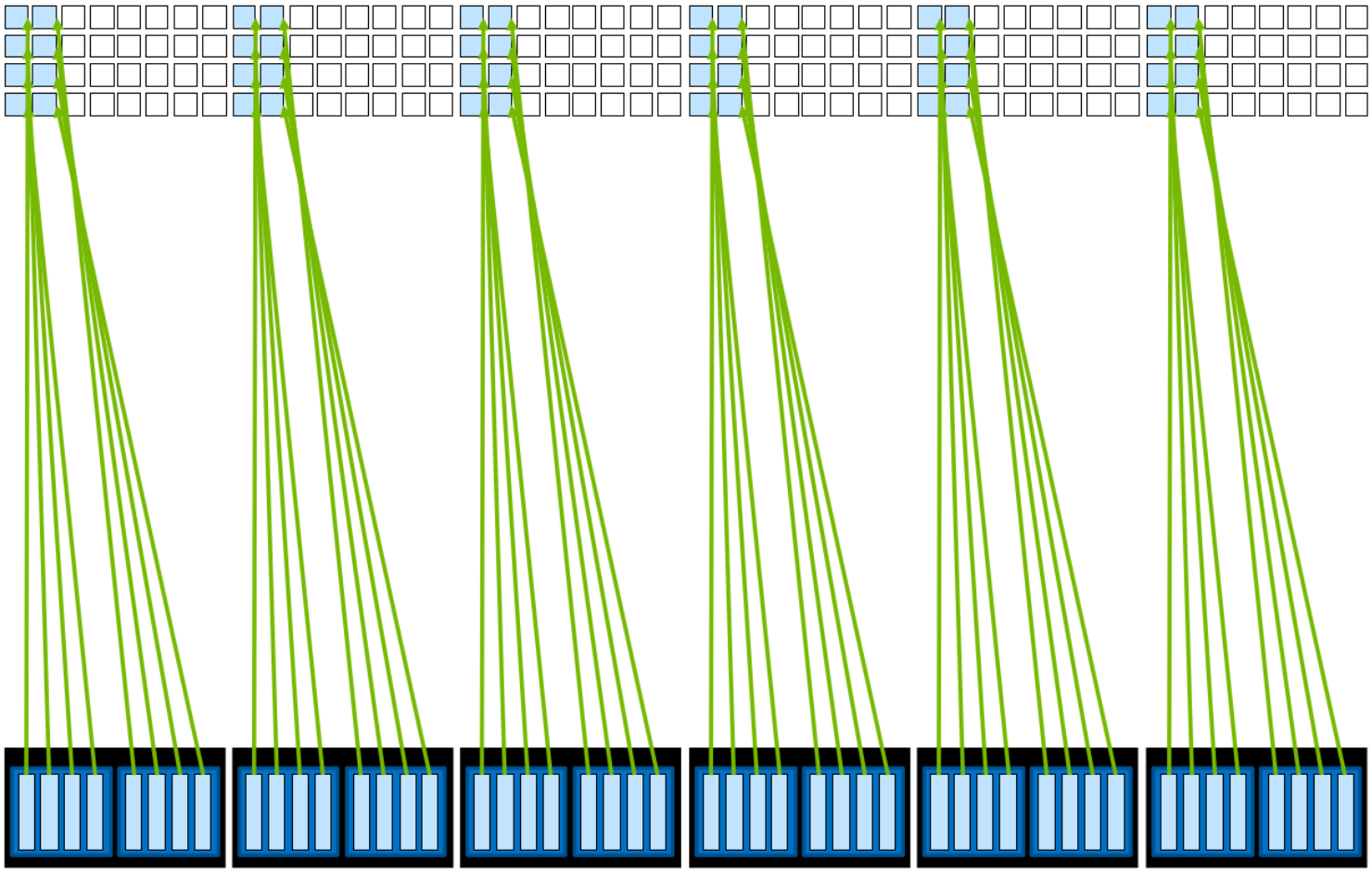
—
Джерело слайду: NVIDIA, DLI



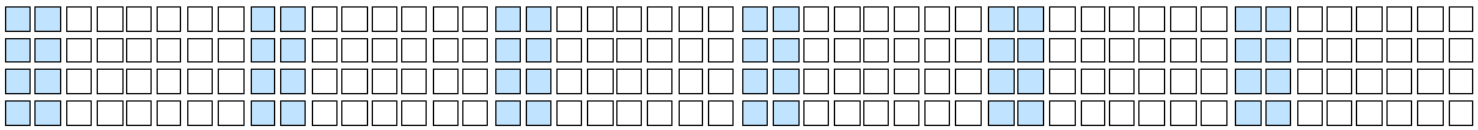
CUDA runs as many blocks in parallel at once as the GPU hardware supports, for massive parallelization



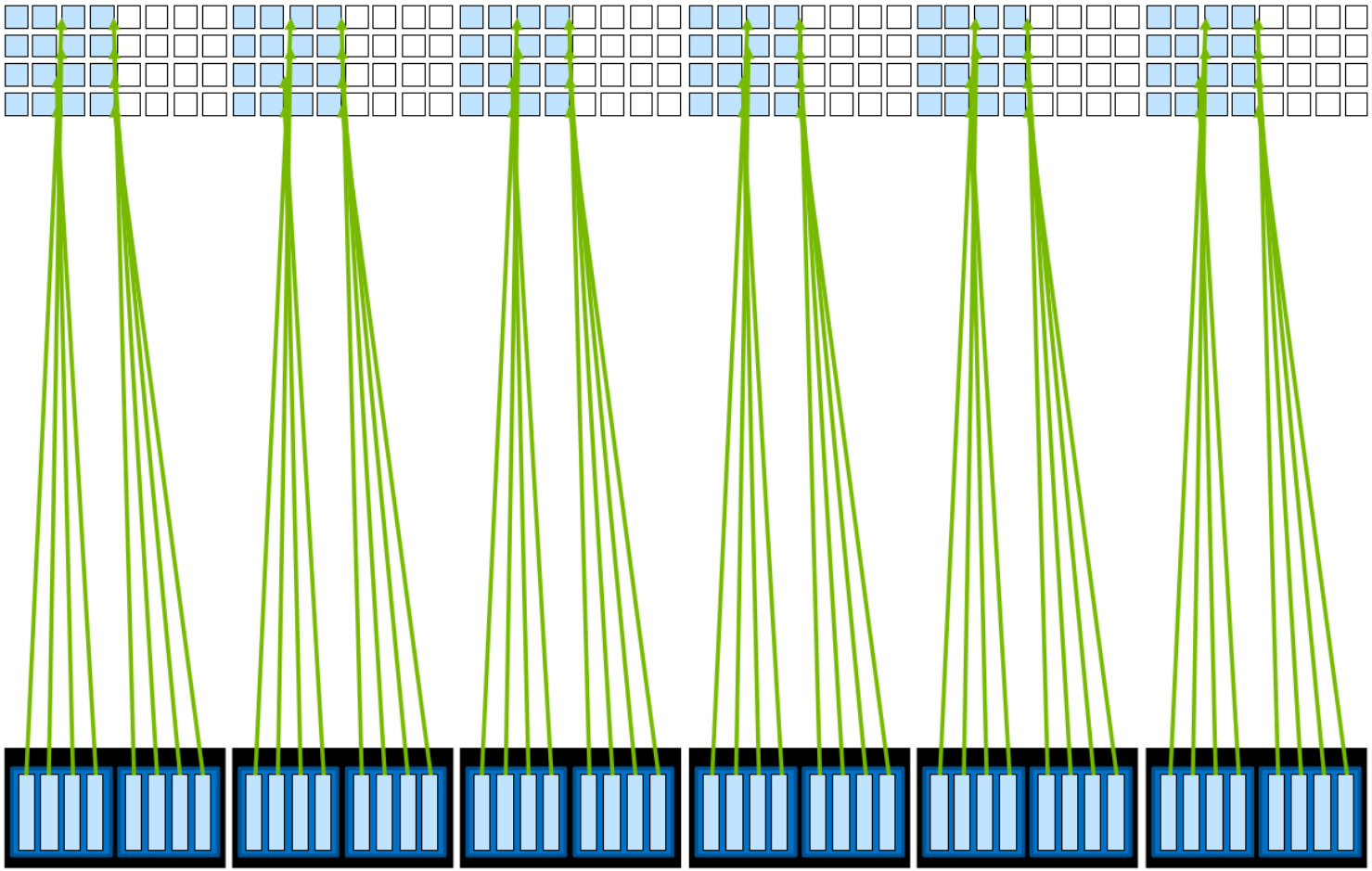
—
Джерело слайду: NVIDIA, DLI



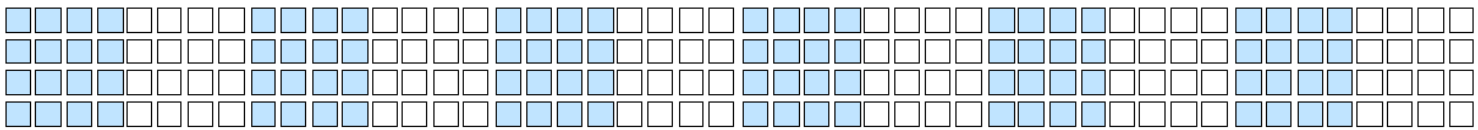
—
Джерело слайду: NVIDIA, DLI

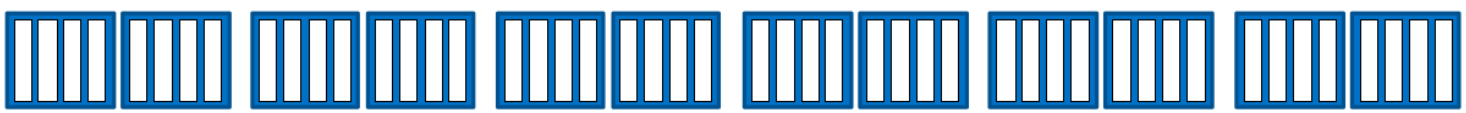


—
Джерело слайду: NVIDIA, DLI



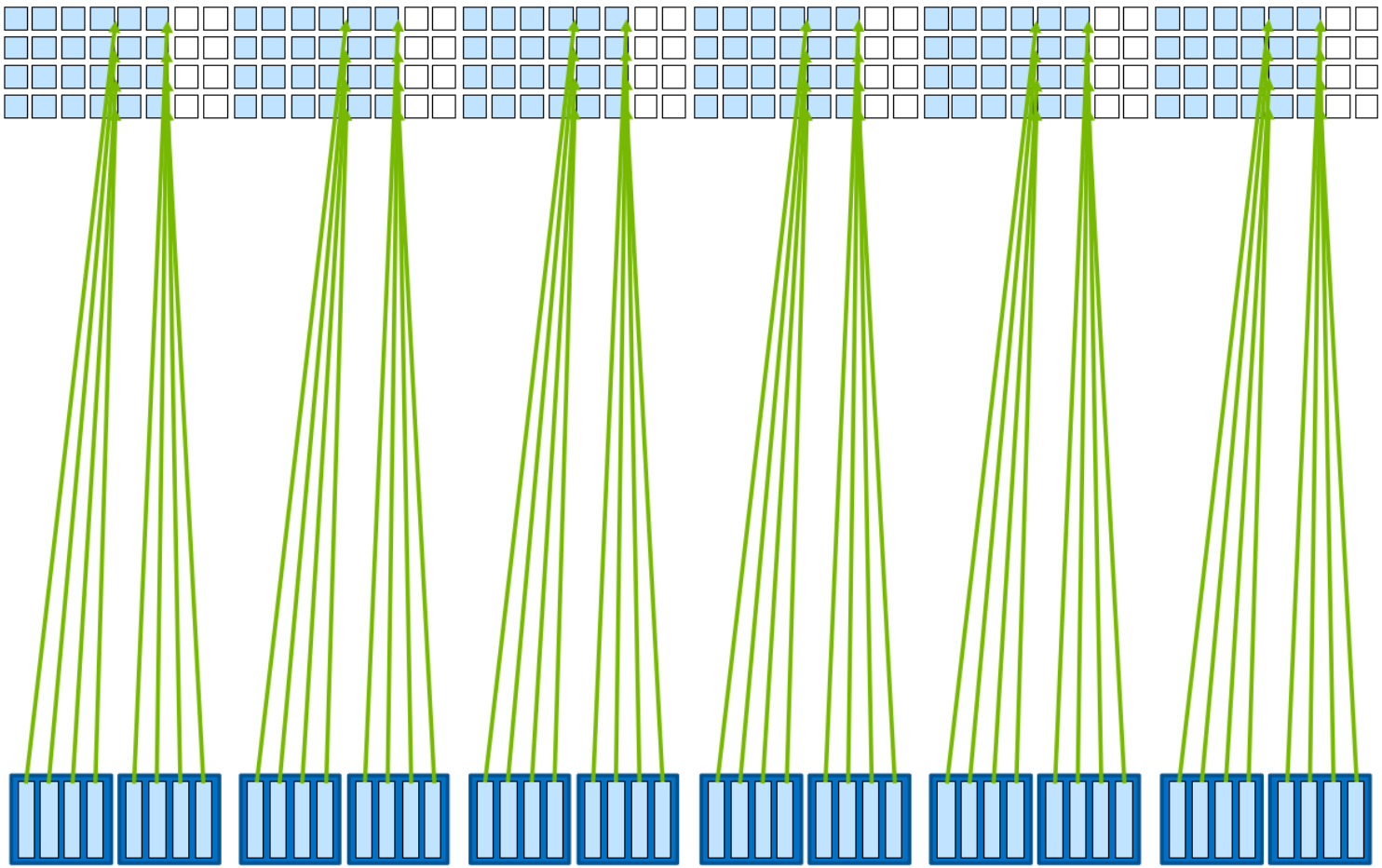
—
Джерело слайду: NVIDIA, DLI



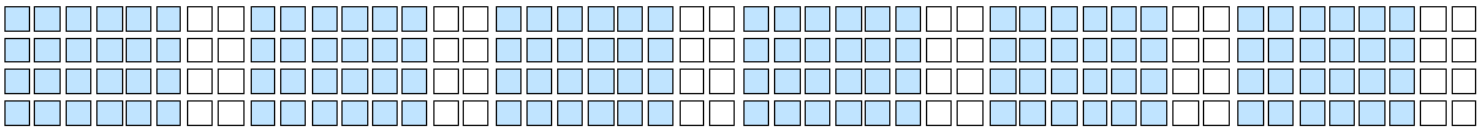


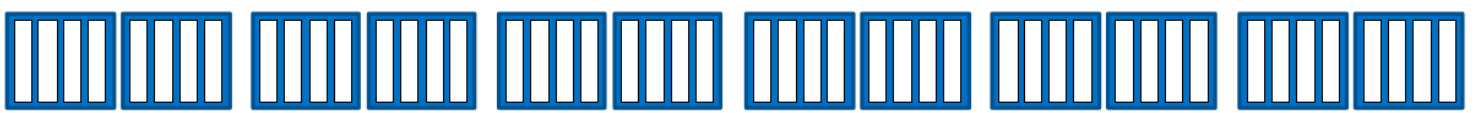
—

Джерело слайду: NVIDIA, DLI



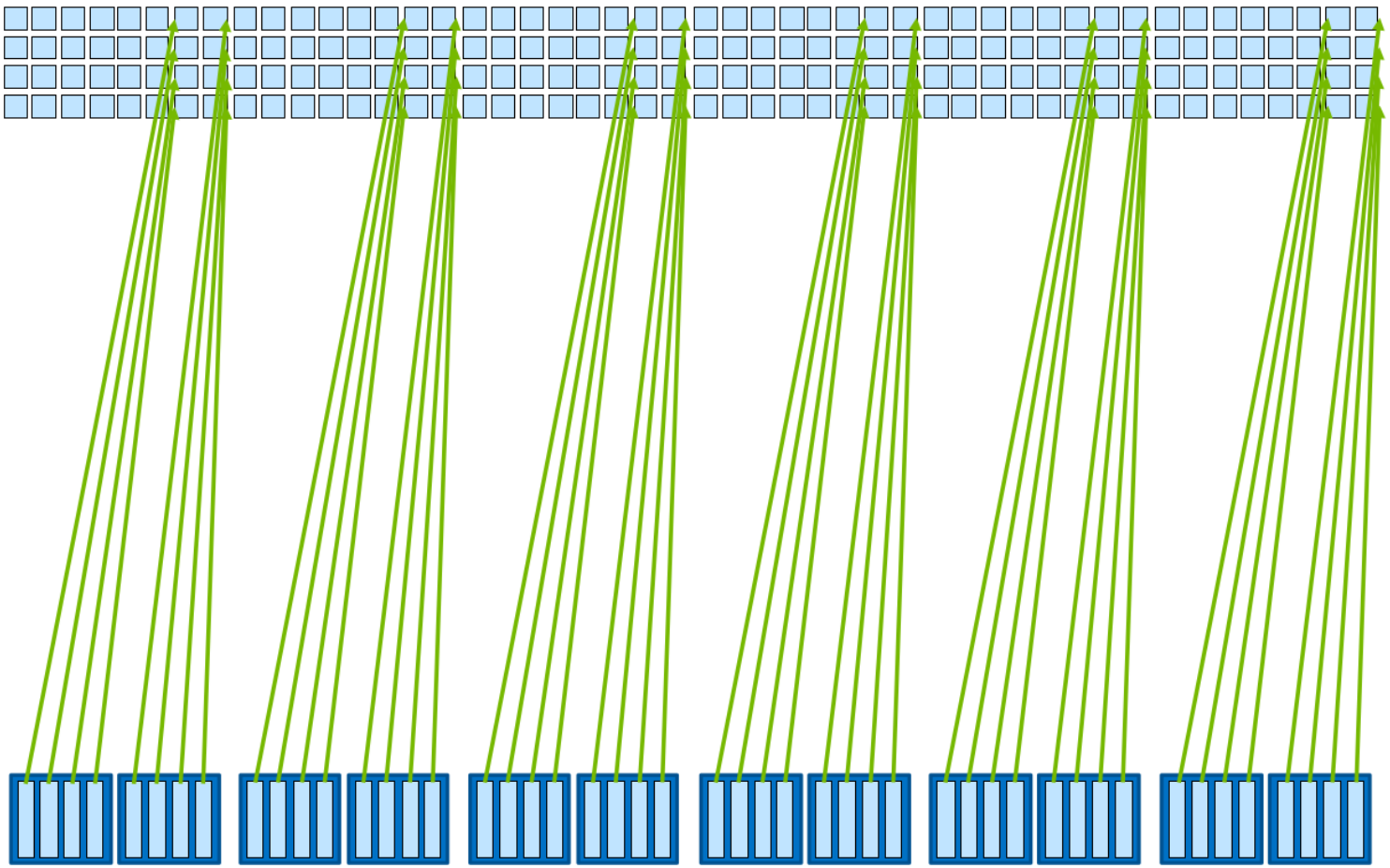
—
Джерело слайду: NVIDIA, DLI



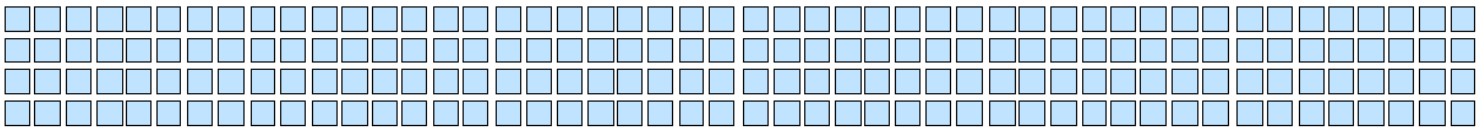


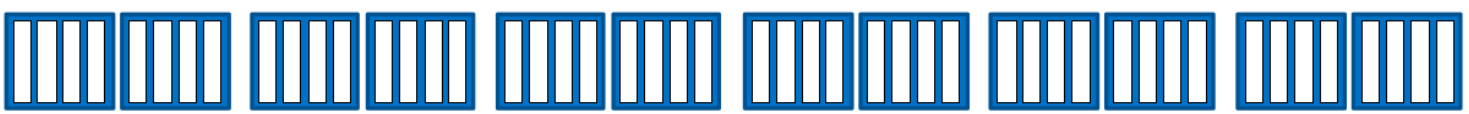
—

Джерело слайду: NVIDIA, DLI



—
Джерело слайду: NVIDIA, DLI





—

Джерело слайду: NVIDIA, DLI

Приклад

```
__global__
void add(int n, float *x, float *y)
{
  int index = blockIdx.x * blockDim.x + threadIdx.x;
  int stride = blockDim.x * gridDim.x;
  for (int i = index; i < n; i += stride)
    y[i] = x[i] + y[i];
}
```

Tesla K80 vs Tesla T4

Глосарій

Глосарій

- `cudaMallocManaged()`: Функція CUDA для виділення пам'яті, доступної як для CPU, так і для GPUs. Пам’ять, виділена таким чином, називається уніфікованою пам’яттю (unified memory) і за потреби автоматично переміщується між CPU та GPUs.
- `cudaDeviceSynchronize()`: Функція CUDA, яка змушує CPU чекати, поки GPU не закінчить роботу.
- `Ядро (Kernel)`: Функція CUDA, що виконується на GPU.
- `Потік (Thread)`: Одиниця виконання для ядер CUDA.
- `Блок (Block)`: Набір потоків.
- `Сітка (Grid)`: Набір блоків.

Кінець