



Технології графічного процесінгу

Лекція 3: Вступ до CUDA C II

Кочура Юрій Петрович

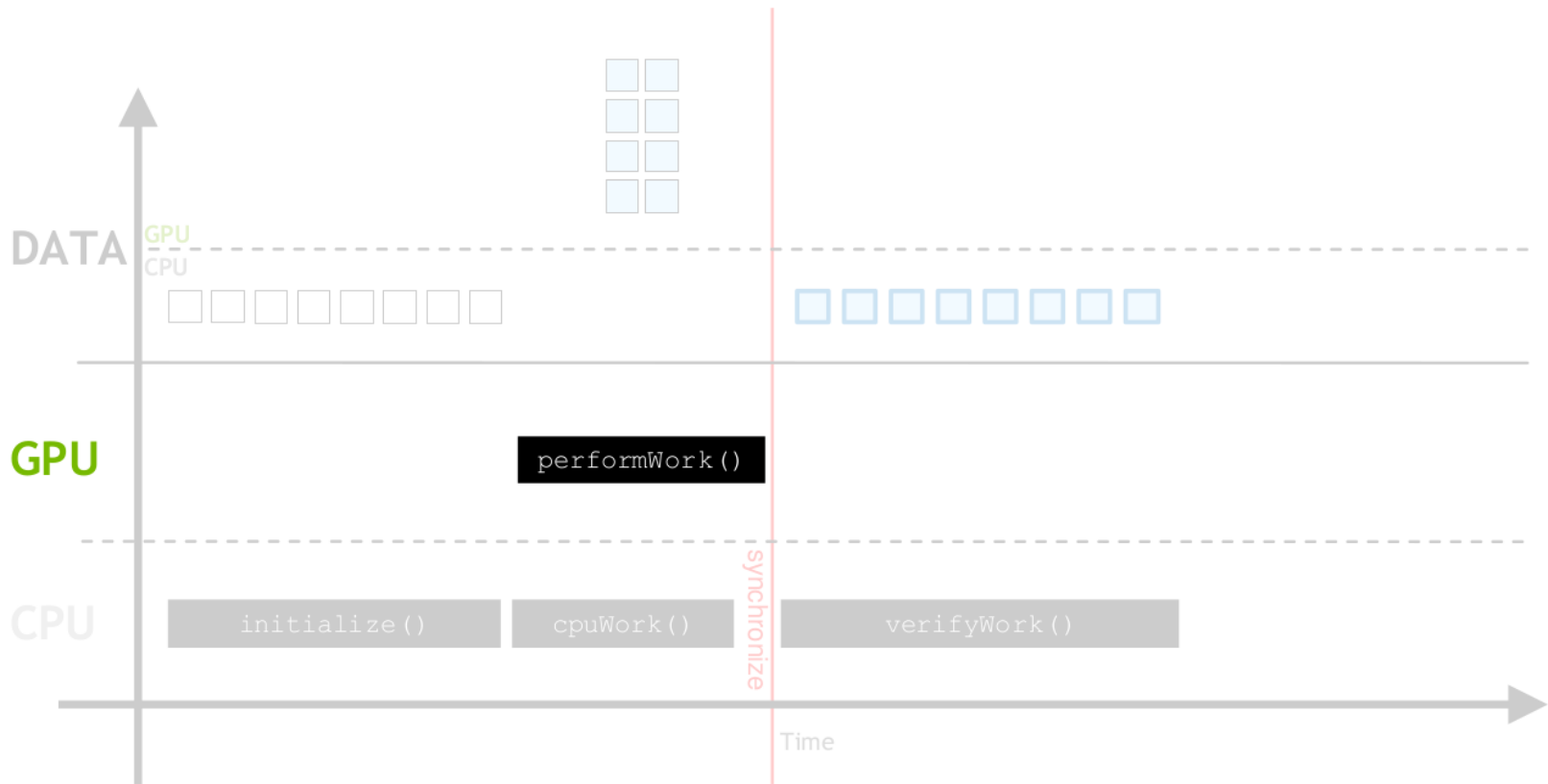
iuriy.kochura@gmail.com

@y_kochura

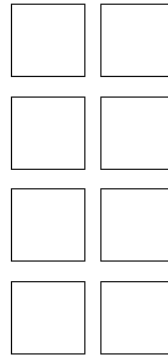
Сьогодні

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

Координація паралельних потоків

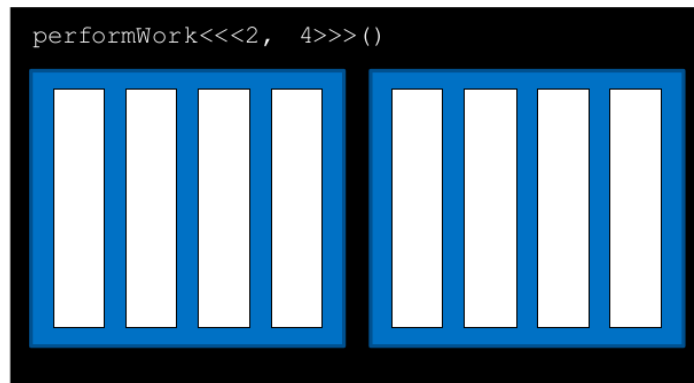


GPU
DATA



Assume data is in a 0 indexed vector

GPU



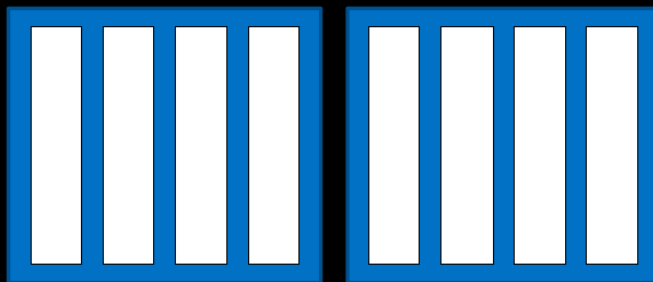
GPU
DATA

0	4
1	5
2	6
3	7

Assume data is in a 0 indexed vector

GPU

```
performWork<<<2, 4>>>()
```

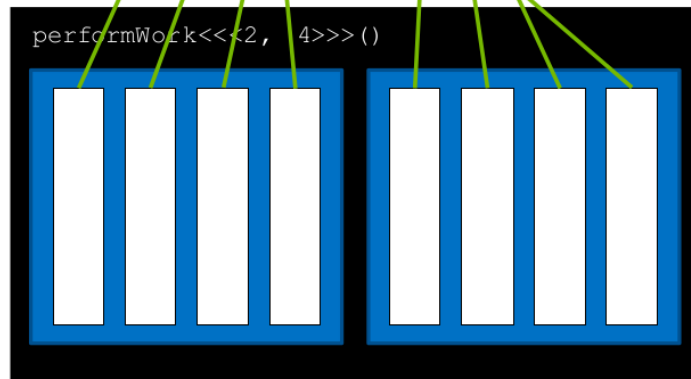


GPU
DATA

0	4
1	5
2	6
3	7

Somehow, each thread must be mapped to work on an element in the vector

GPU

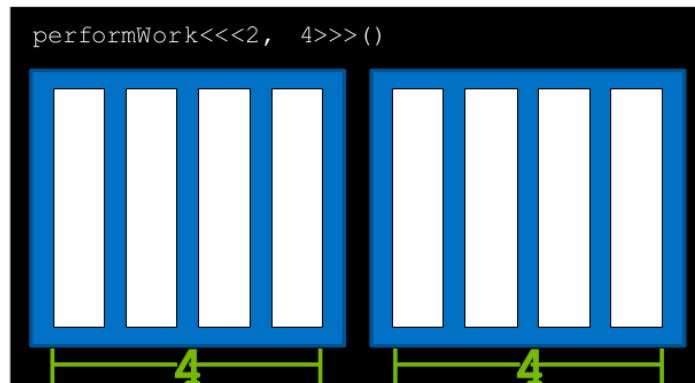


GPU DATA

0	4
1	5
2	6
3	7

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

GPU

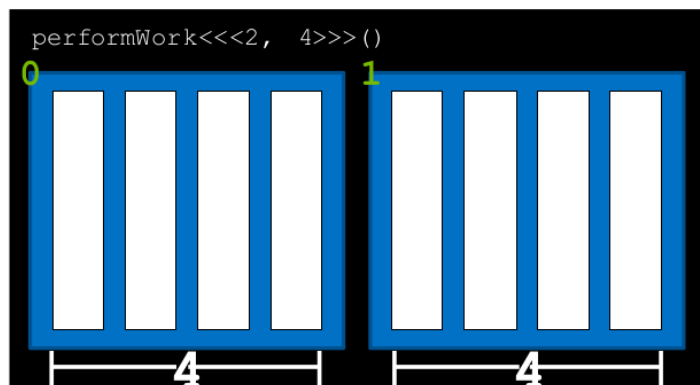


GPU DATA

0	4
1	5
2	6
3	7

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

GPU

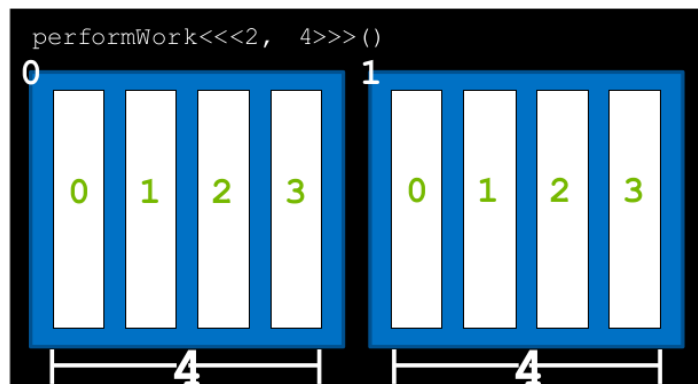


GPU DATA

0	4
1	5
2	6
3	7

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

GPU

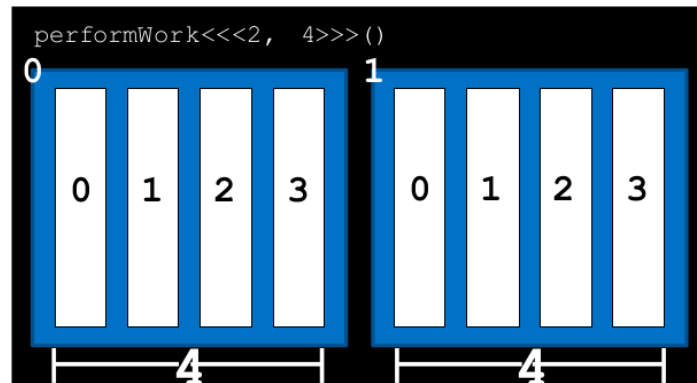


GPU DATA

0	4
1	5
2	6
3	7

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

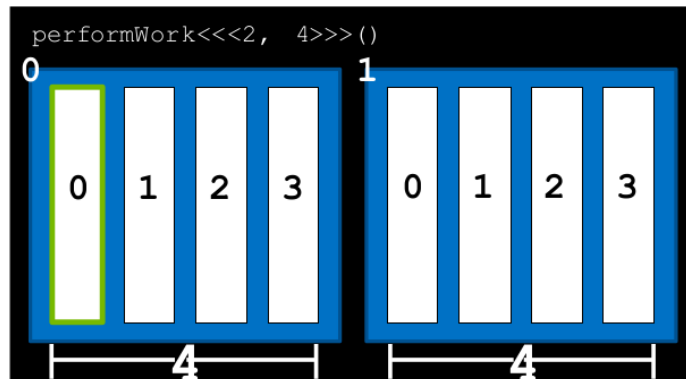
GPU



GPU DATA

0	4	threadIdx.x	+	blockIdx.x	*	blockDim.x
1	5	0		0		4
2	6	dataIndex				
3	7	?				

GPU



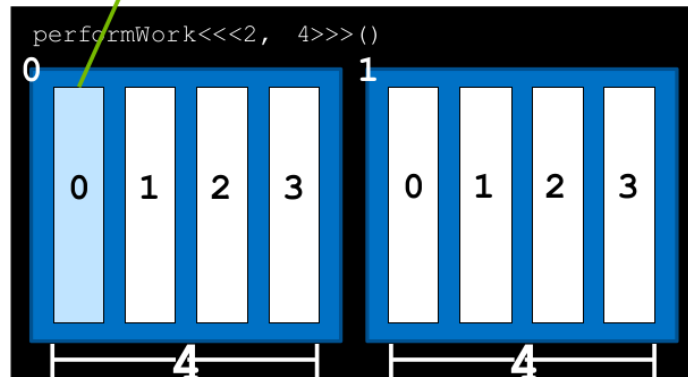
GPU
DATA

0	4
1	5
2	6
3	7

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

dataIndex
0

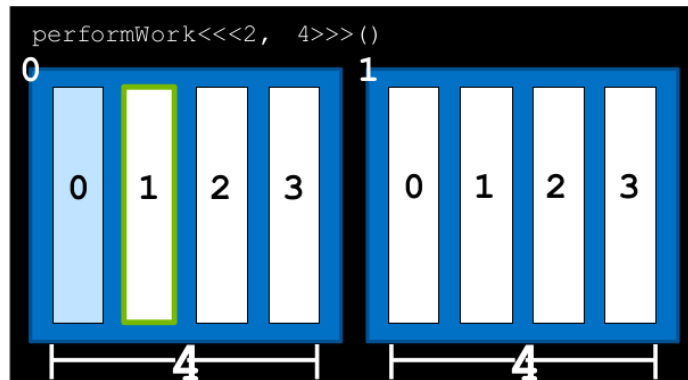
GPU



GPU DATA

0	4	threadIdx.x	+	blockIdx.x	*	blockDim.x
1	5	1		0		4
2	6	dataIndex				
3	7	?				

GPU



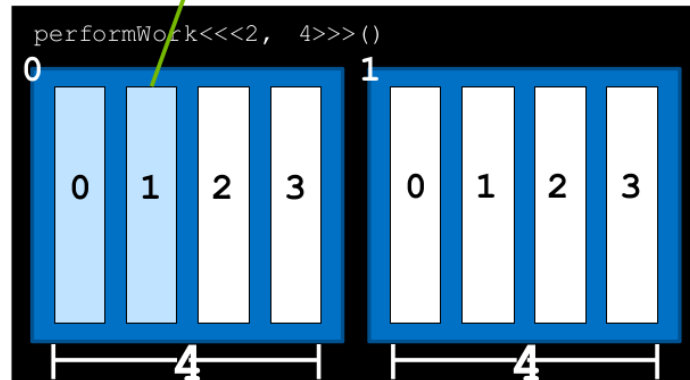
GPU
DATA

0	4
1	5
2	6
3	7

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

dataIndex
1

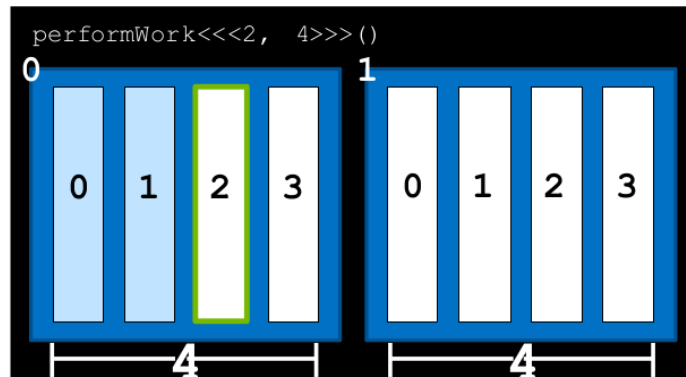
GPU



GPU DATA

0	4	threadIdx.x	+	blockIdx.x	*	blockDim.x
1	5	2		0		4
2	6	dataIndex				
3	7	?				

GPU



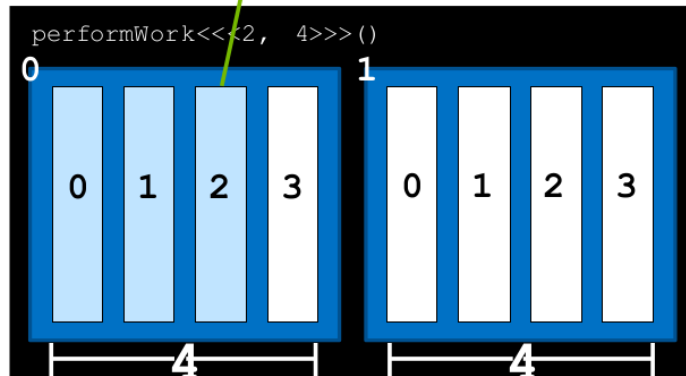
GPU
DATA

0	4
1	5
2	6
3	7

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

dataIndex
2

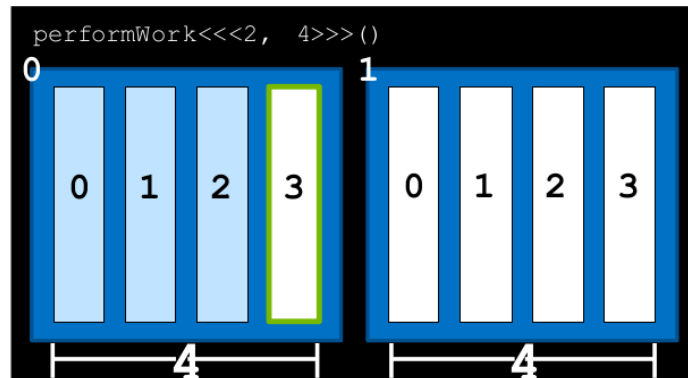
GPU



GPU DATA

0	4	threadIdx.x	+	blockIdx.x	*	blockDim.x
1	5	3		0		4
2	6	dataIndex				
3	7	?				

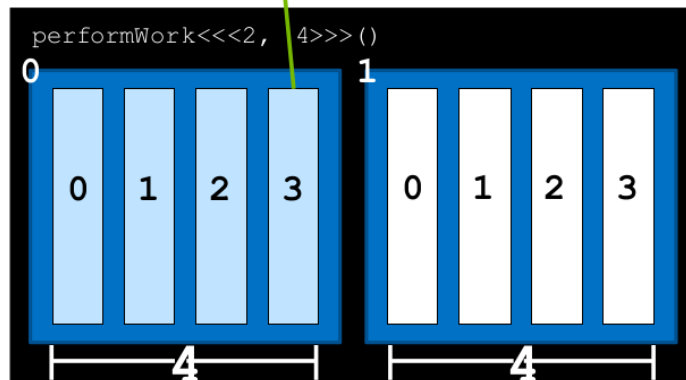
GPU



GPU DATA

0	4	threadIdx.x	+	blockIdx.x	*	blockDim.x
1	5	3		0		4
2	6	dataIndex				
3	7	3				

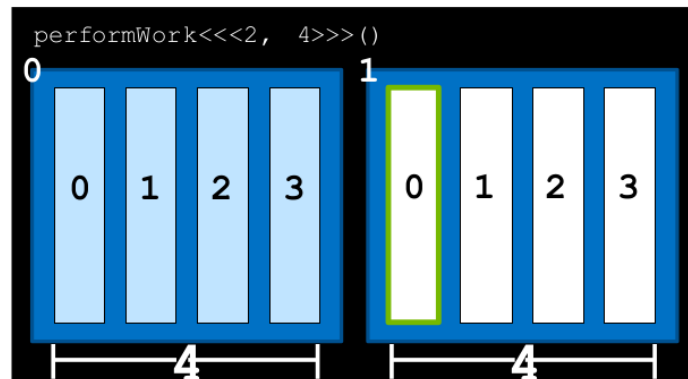
GPU



GPU DATA

0	4	threadIdx.x	+	blockIdx.x	*	blockDim.x
1	5	0		1		4
2	6	dataIndex				
3	7	?				

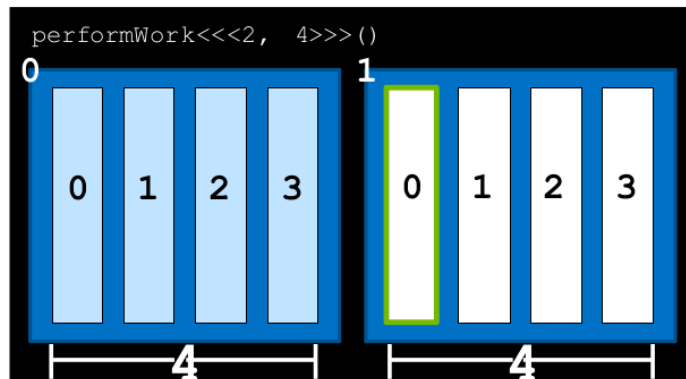
GPU



GPU DATA

0	4	threadIdx.x	+	blockIdx.x	*	blockDim.x
1	5	0		1		4
2	6	dataIndex				
3	7	?				

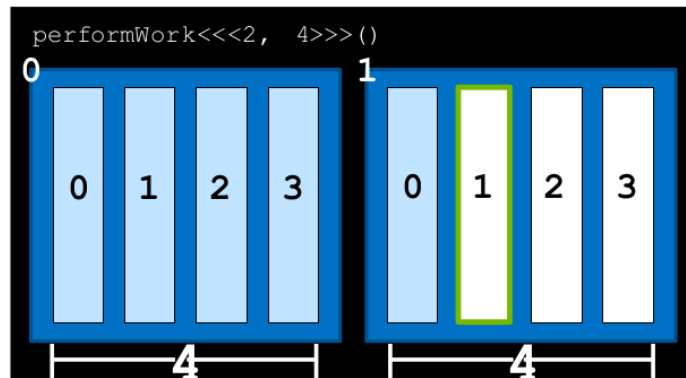
GPU



GPU DATA

0	4	threadIdx.x	+	blockIdx.x	*	blockDim.x
1	5	1		1		4
2	6	dataIndex				
3	7	?				

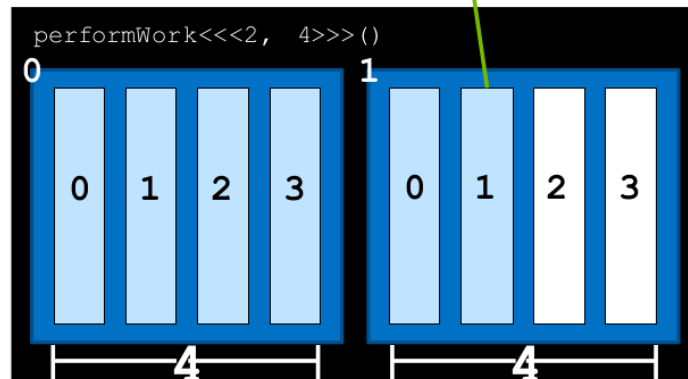
GPU



GPU
DATA

0	4	threadIdx.x	+	blockIdx.x	*	blockDim.x
1	5	1		1		4
2	6	dataIndex				
3	7	5				

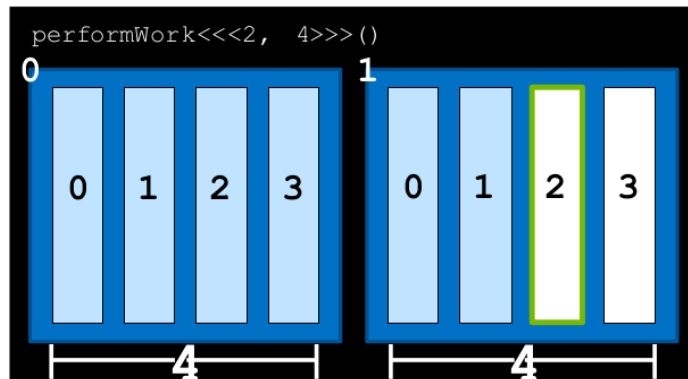
GPU



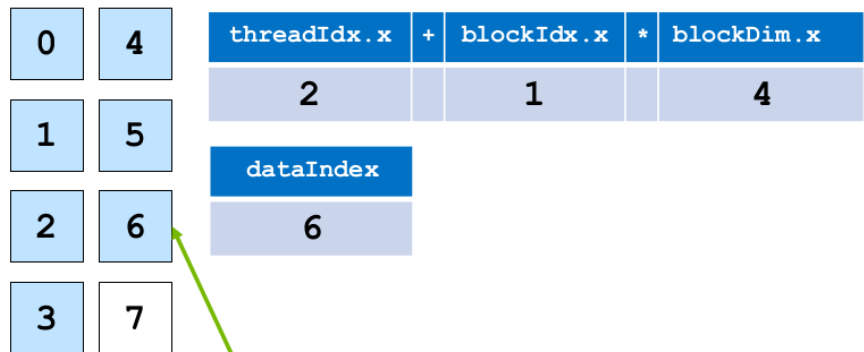
GPU DATA

0	4	threadIdx.x	+	blockIdx.x	*	blockDim.x
1	5	2		1		4
2	6	dataIndex				
3	7	?				

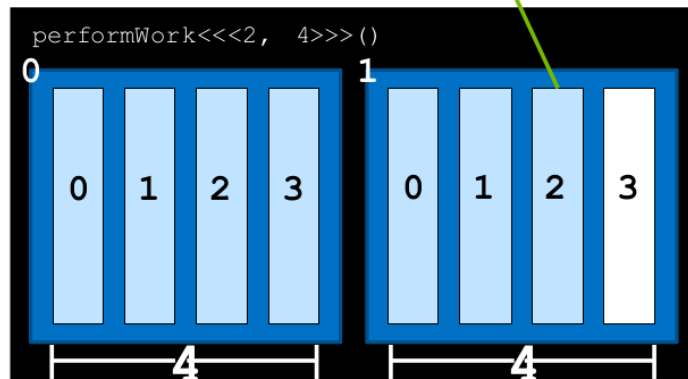
GPU



GPU DATA



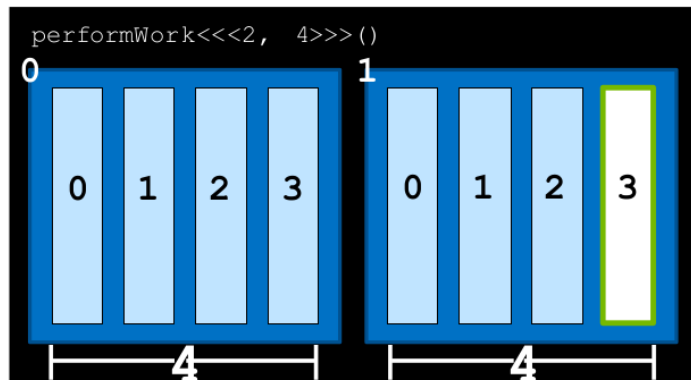
GPU



GPU DATA

0	4	threadIdx.x	+	blockIdx.x	*	blockDim.x
1	5	3		1		4
2	6	dataIndex				
3	7	?				

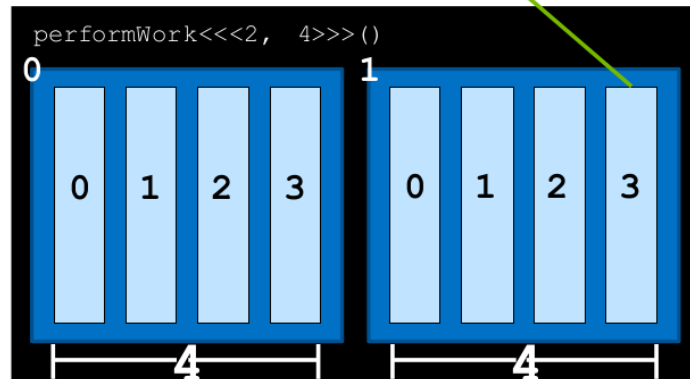
GPU



GPU
DATA

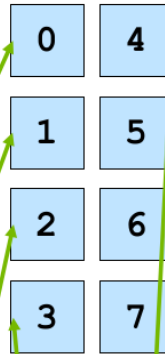
0	4	threadIdx.x	+	blockIdx.x	*	blockDim.x
1	5	3		1		4
2	6	dataIndex				
3	7	7				

GPU



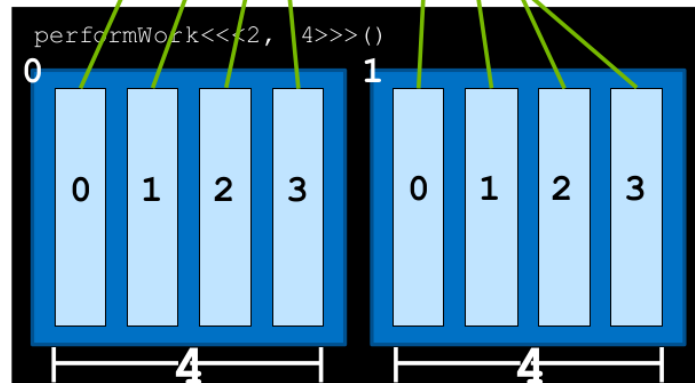
Невідповідність розміру сітки

GPU
DATA

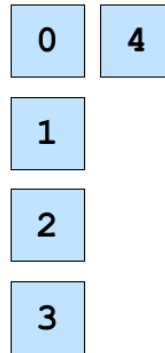


In previous scenarios, the number of threads in the grid matched the number of elements exactly

GPU

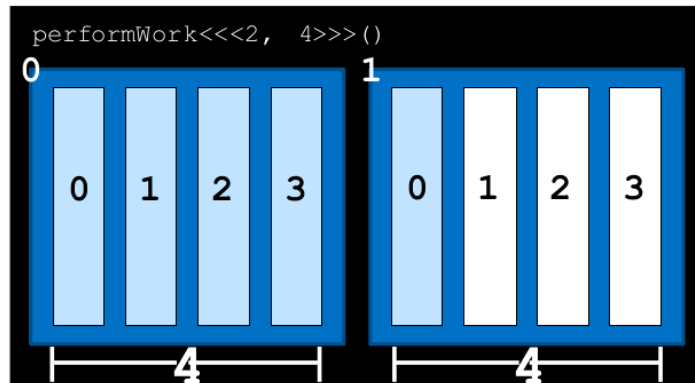


GPU DATA

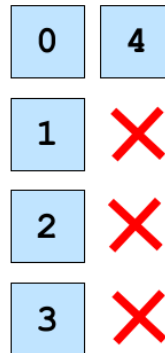


What if there are more threads than work to be done?

GPU

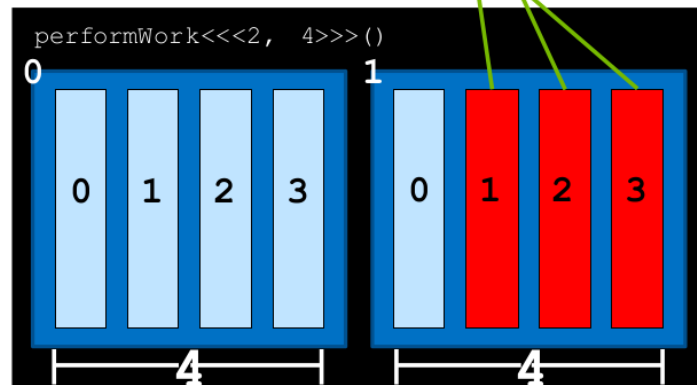


GPU
DATA



Attempting to access non-existent elements can result in a runtime error

GPU

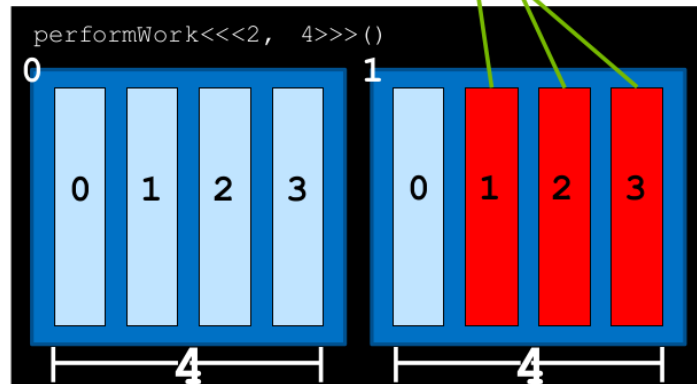


GPU DATA

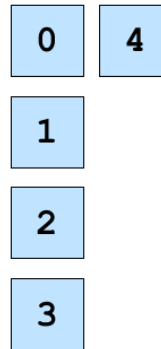
0	4
1	×
2	×
3	×

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

GPU



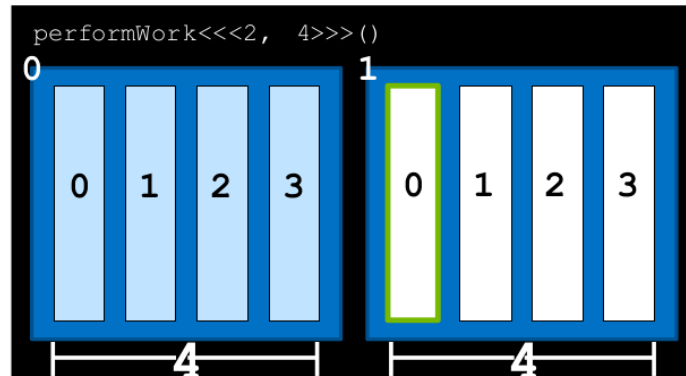
GPU DATA



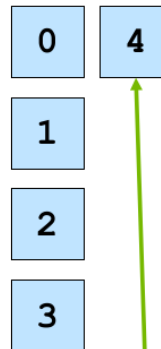
threadIdx.x	+	blockIdx.x	*	blockDim.x
0		1		4

dataIndex	<	N	=	Can work
4		5		?

GPU



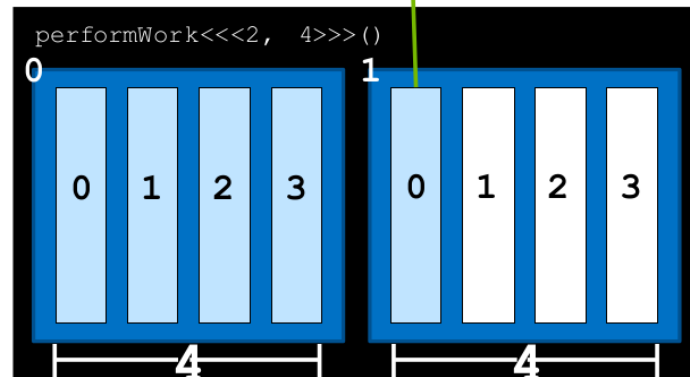
GPU
DATA



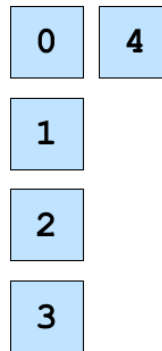
threadIdx.x	+	blockIdx.x	*	blockDim.x
0		1		4

dataIndex	<	N	=	Can work
4		5		true

GPU



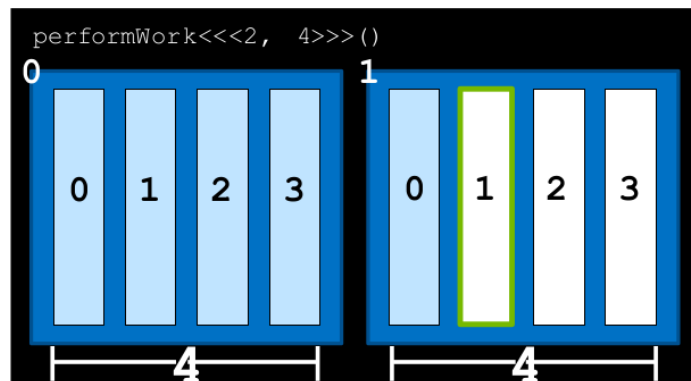
GPU DATA



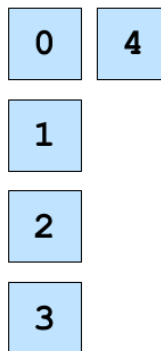
threadIdx.x	+	blockIdx.x	*	blockDim.x
1		1		4

dataIndex	<	N	=	Can work
5		5		?

GPU



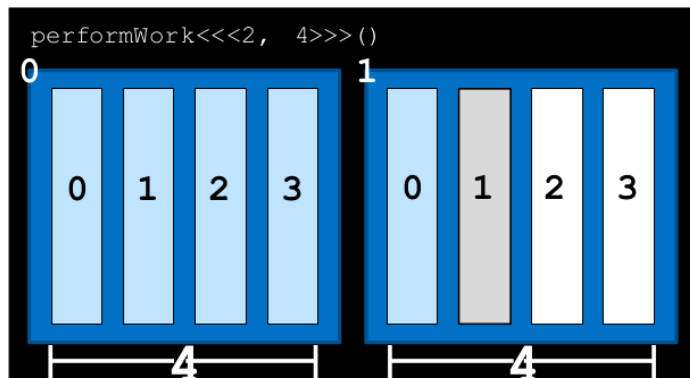
GPU DATA



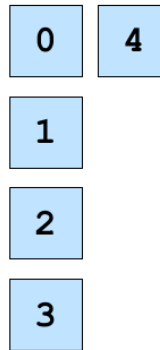
<code>threadIdx.x</code>	+	<code>blockIdx.x</code>	*	<code>blockDim.x</code>
1		1		4

<code>dataIndex</code>	<	N	=	Can work
5		5		false

GPU



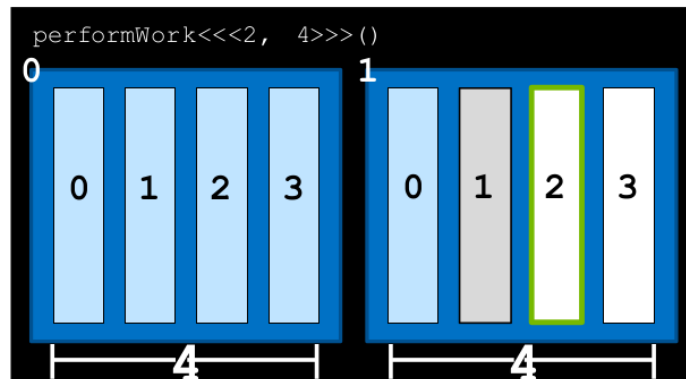
GPU DATA



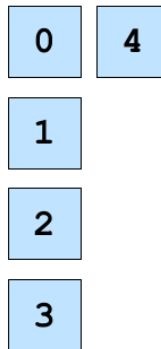
threadIdx.x	+	blockIdx.x	*	blockDim.x
2		1		4

dataIndex	<	N	=	Can work
6		5		?

GPU



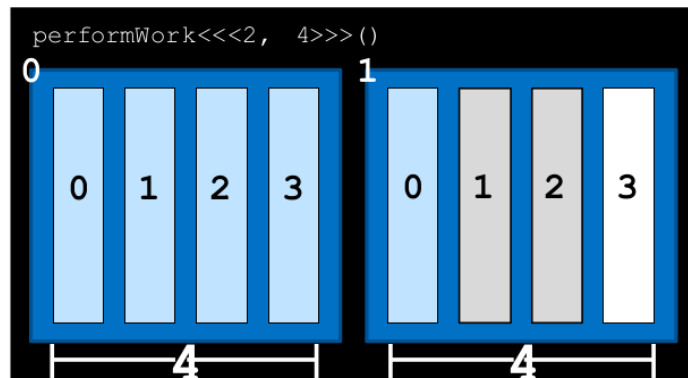
GPU DATA



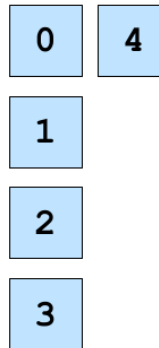
threadIdx.x	+	blockIdx.x	*	blockDim.x
2		1		4

dataIndex	<	N	=	Can work
6		5		false

GPU



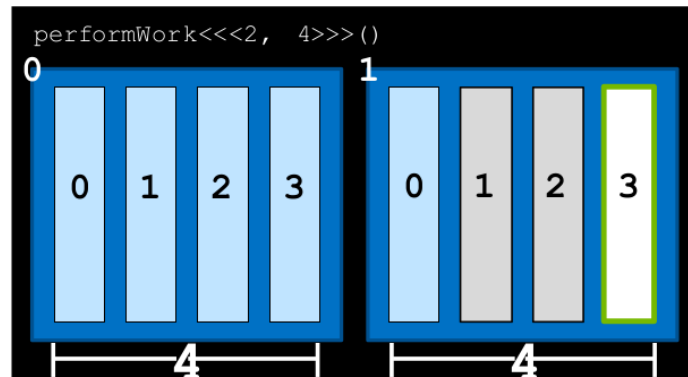
GPU DATA



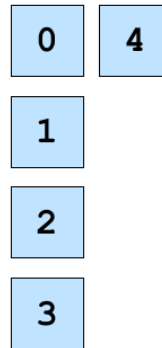
threadIdx.x	+	blockIdx.x	*	blockDim.x
3		1		4

dataIndex	<	N	=	Can work
7		5		?

GPU

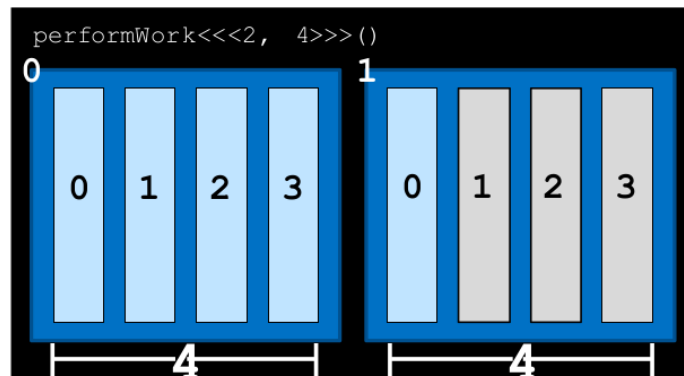


GPU DATA



threadIdx.x	+	blockIdx.x	*	blockDim.x
3		1		4
dataIndex	<	N	=	Can work
7		5		false

GPU



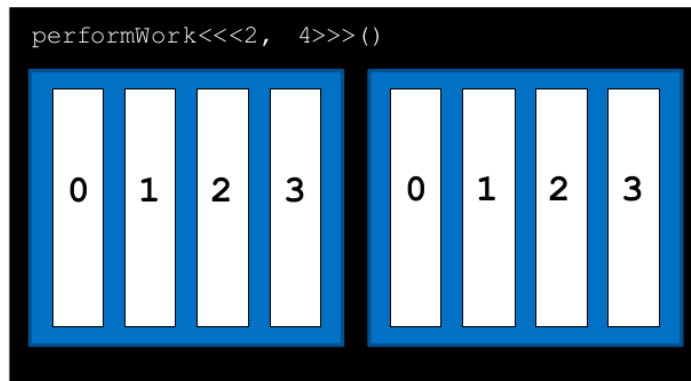
Цикли: крок за сіткою

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

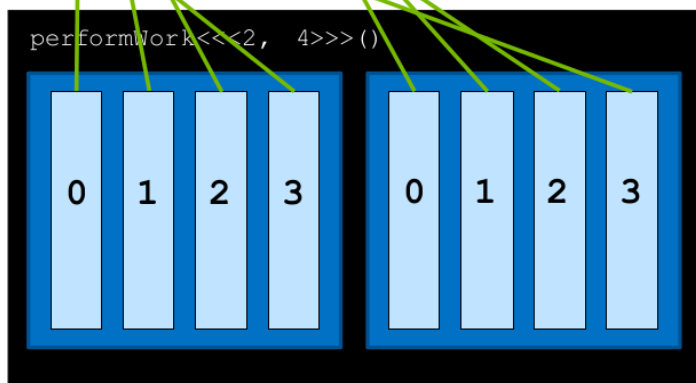


GPU
DATA

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

In such scenarios threads
cannot work on only one
element

GPU

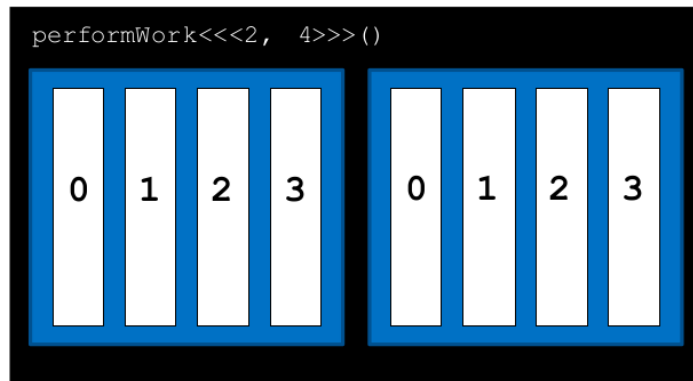


GPU DATA

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

... or else work is left
undone

GPU

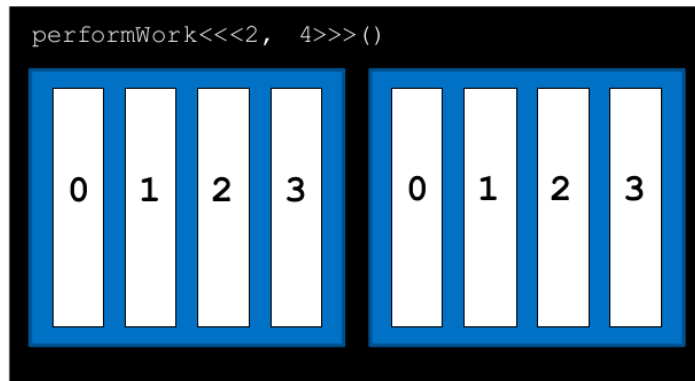


GPU DATA

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

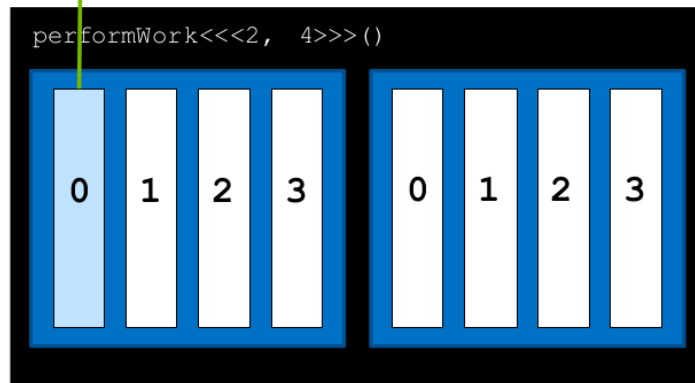


GPU DATA

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

In a grid-stride loop, the thread's first element is calculated as usual, with
 $\text{threadIdx.x} + \text{blockIdx.x} * \text{blockDim.x}$

GPU

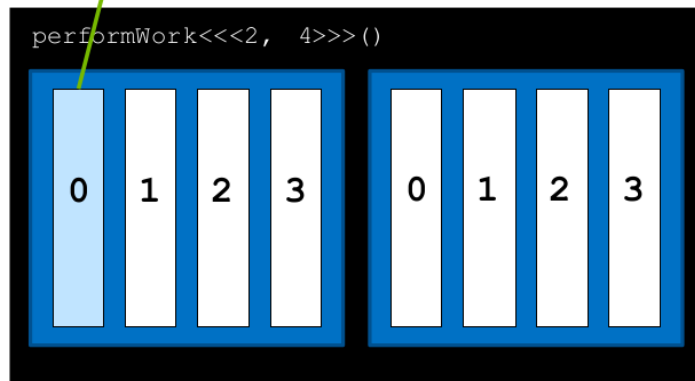


GPU DATA

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

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

GPU

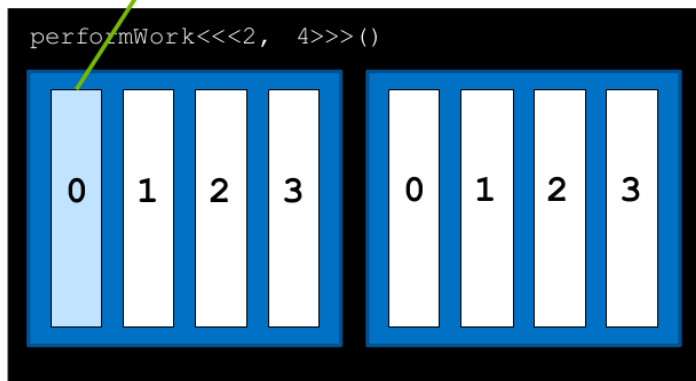


GPU DATA

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

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

GPU

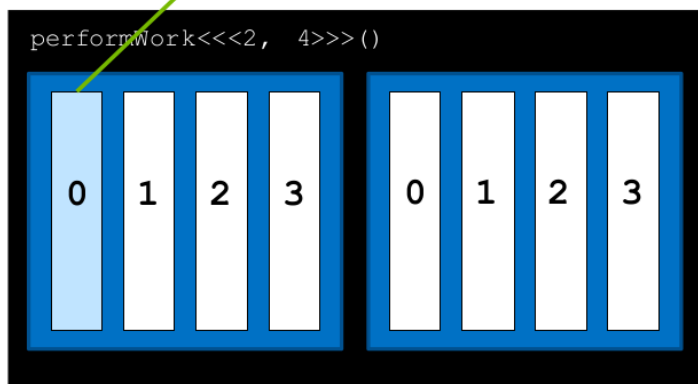


GPU DATA

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

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

GPU

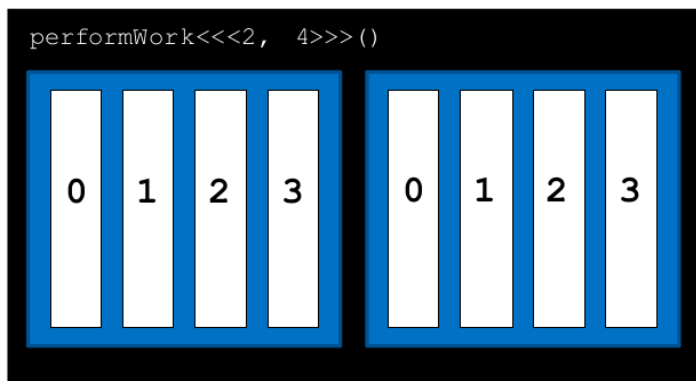


GPU DATA

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

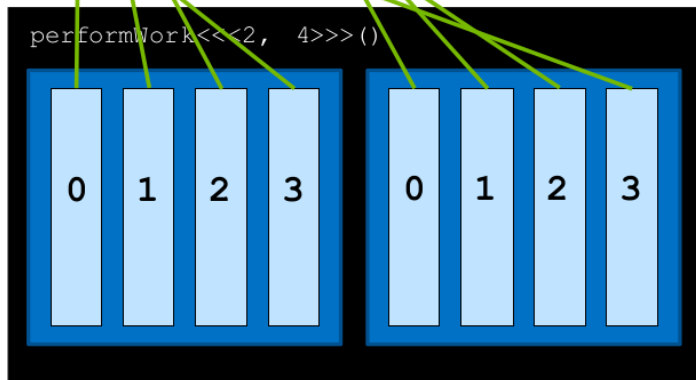


GPU
DATA

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

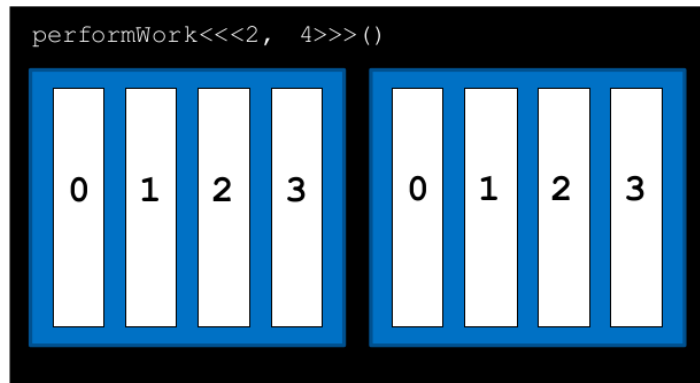


GPU DATA

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

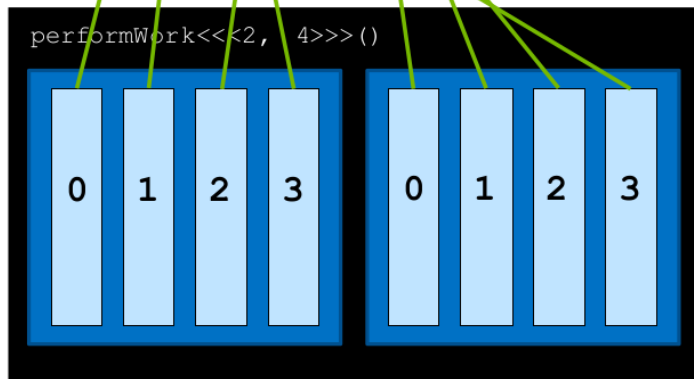


GPU
DATA

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

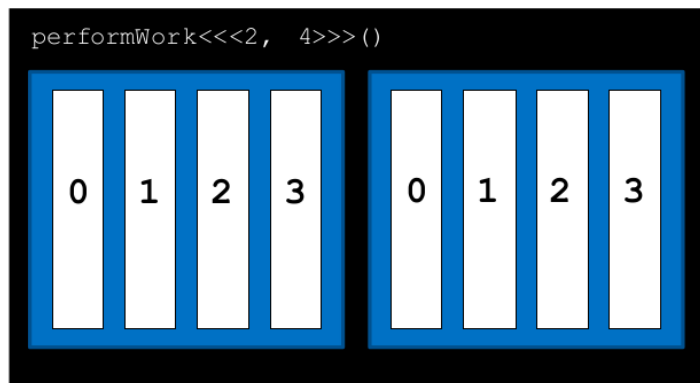


GPU DATA

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

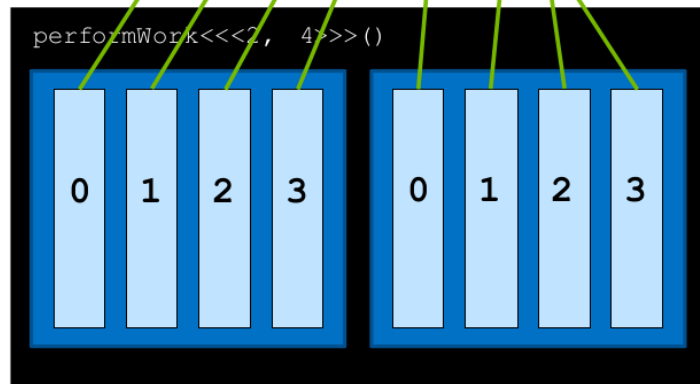


GPU
DATA

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

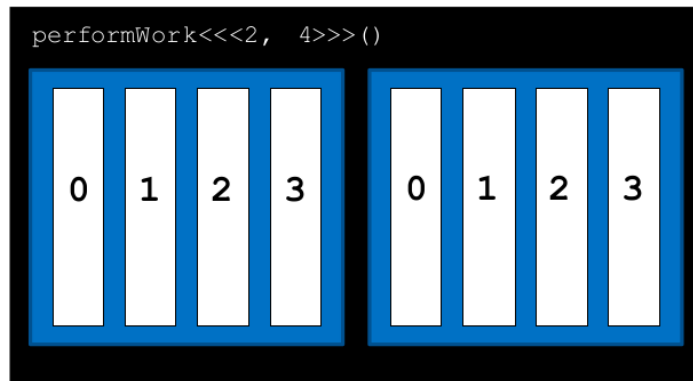


GPU DATA

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

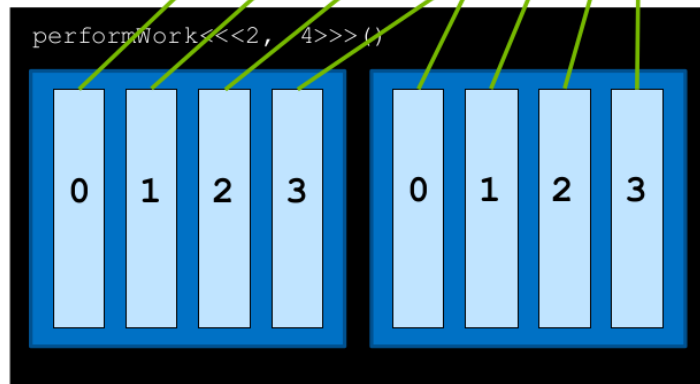


GPU
DATA

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

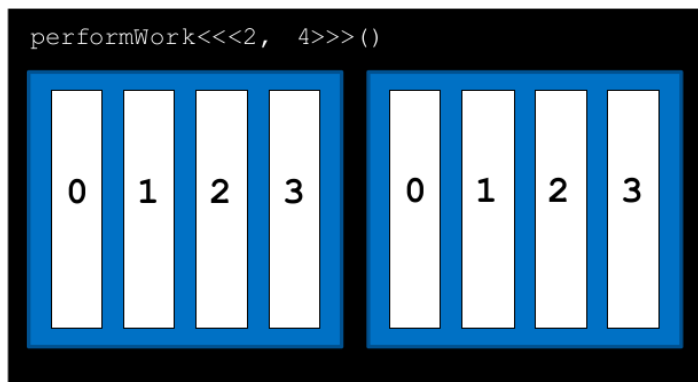


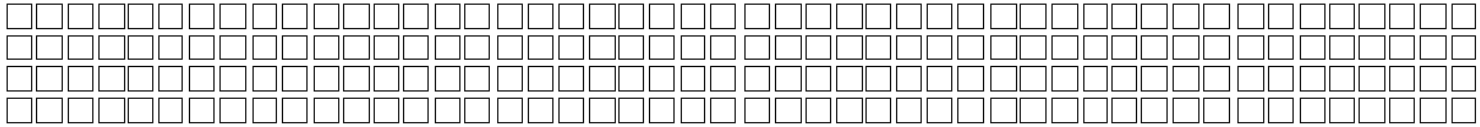
GPU DATA

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

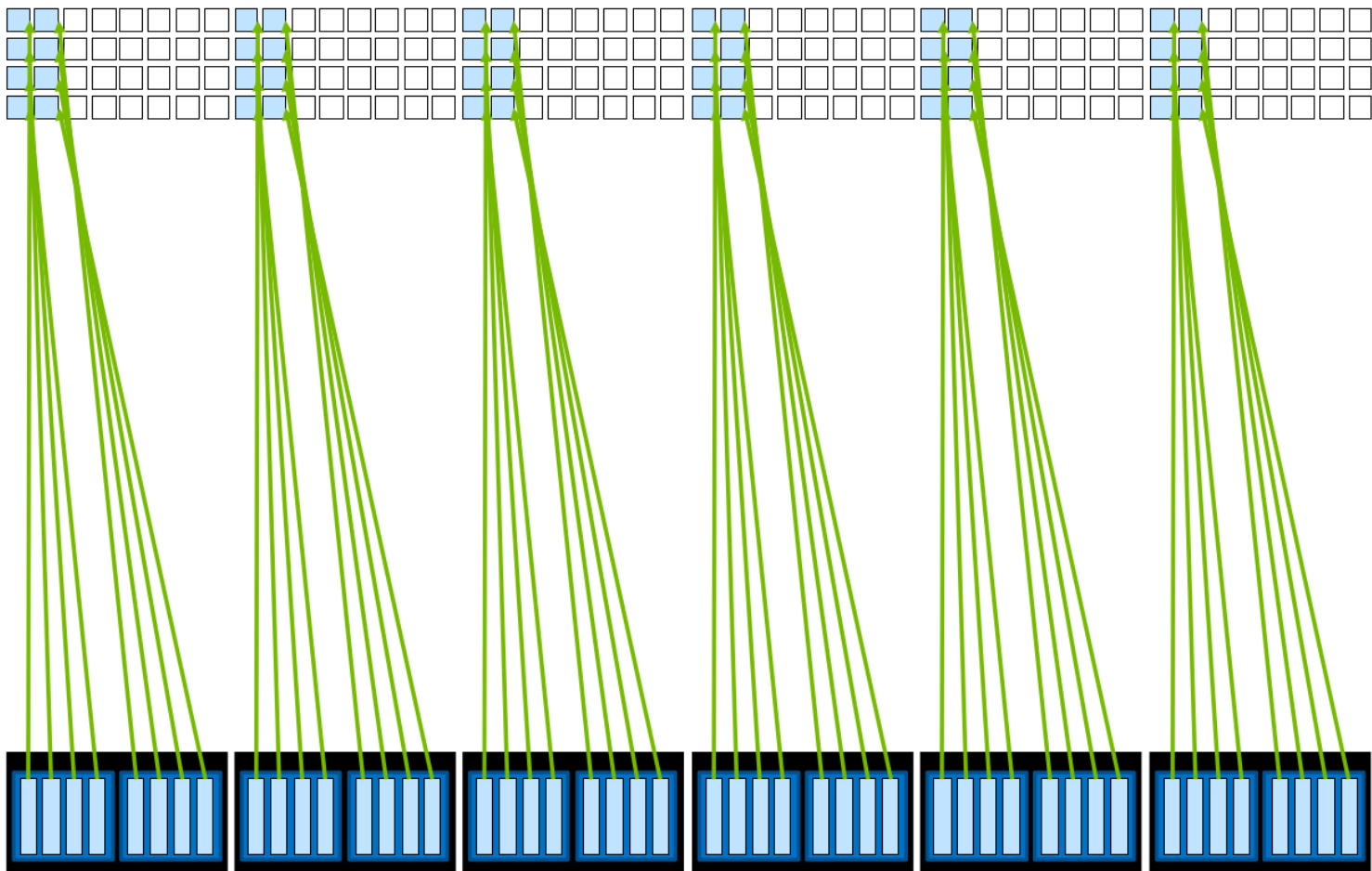
GPU

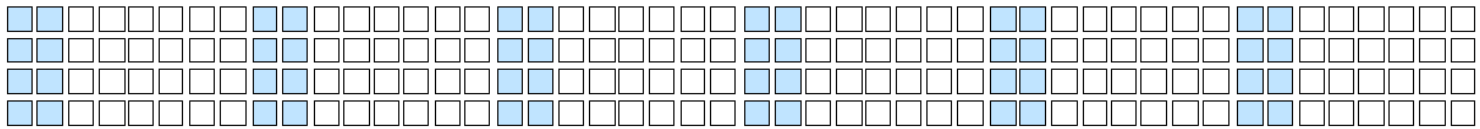


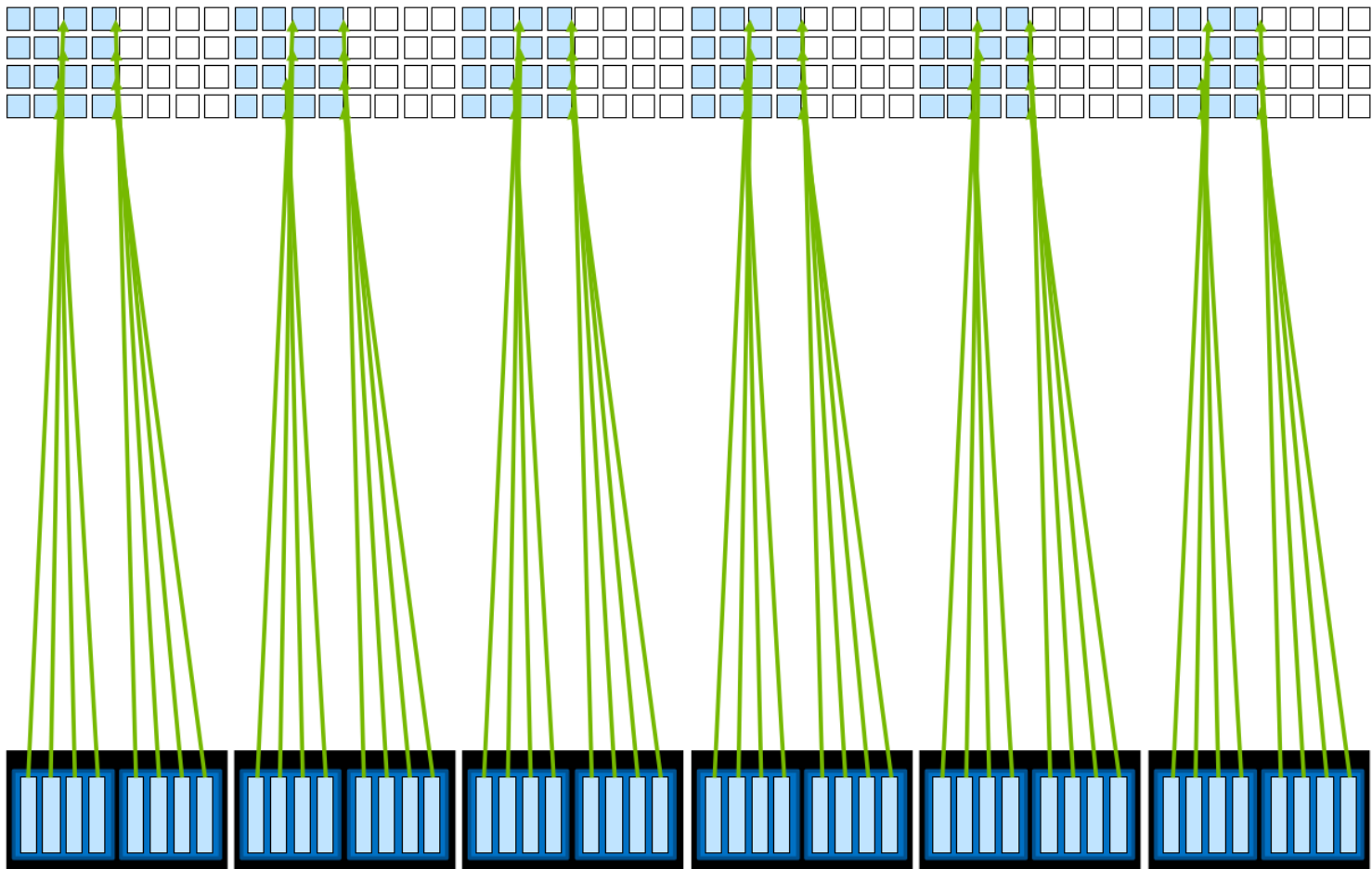


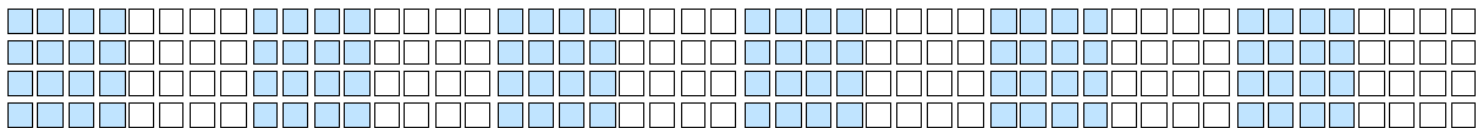
CUDA runs as many blocks
in parallel at once as the
GPU hardware supports, for
massive parallelization

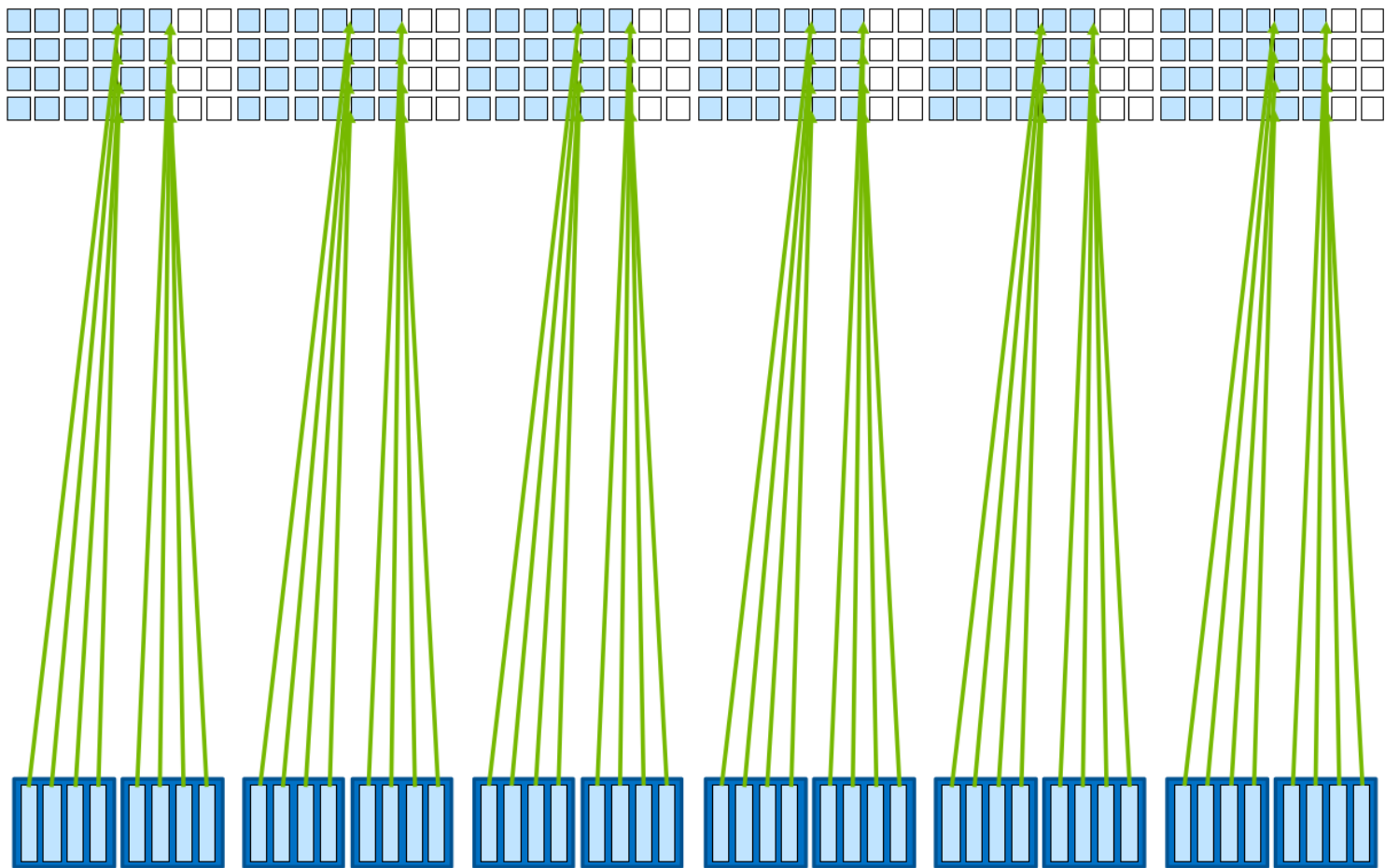


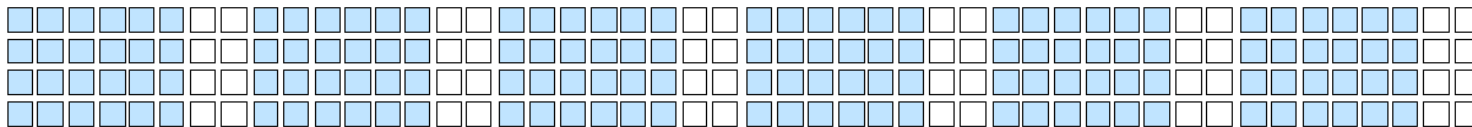


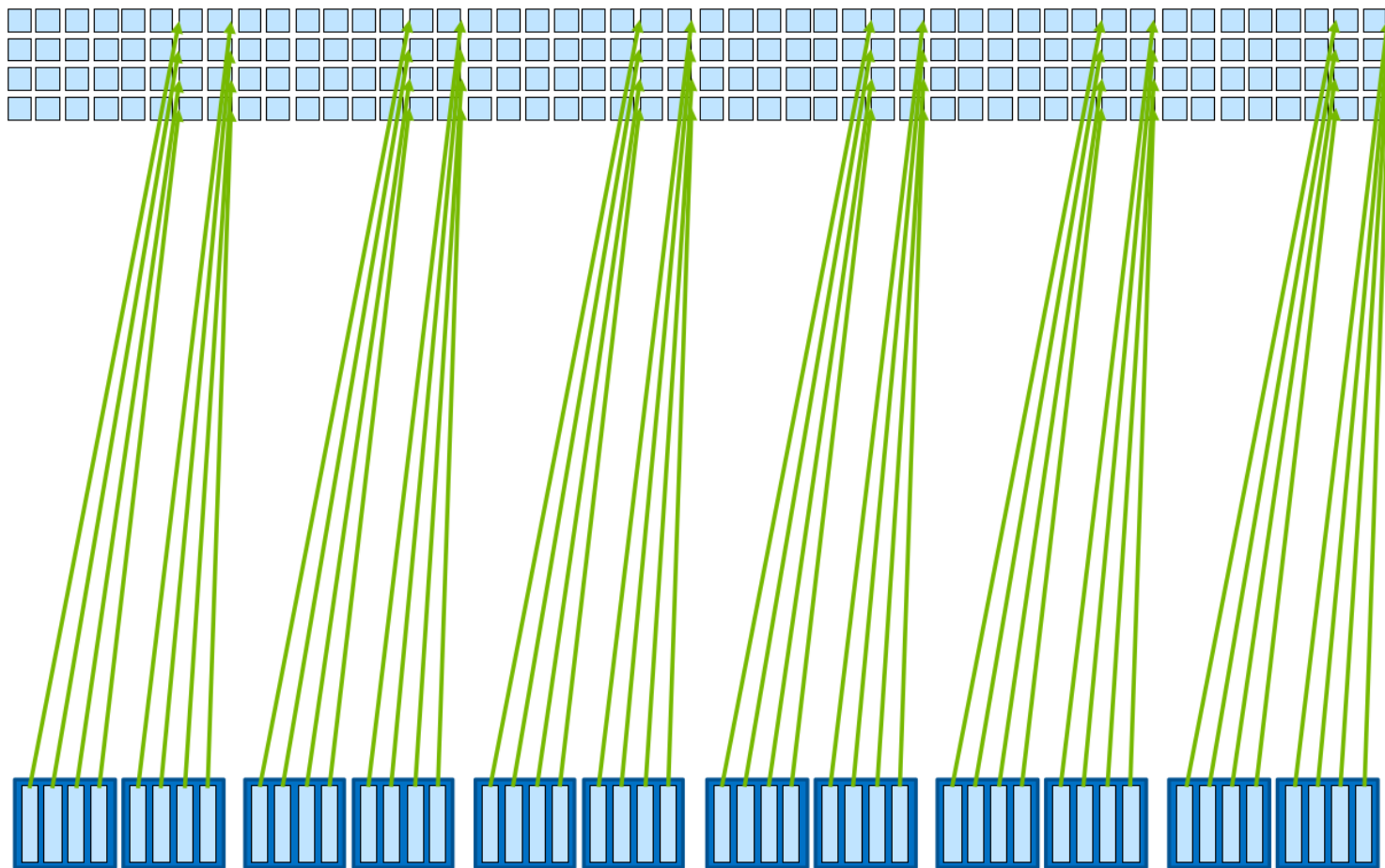


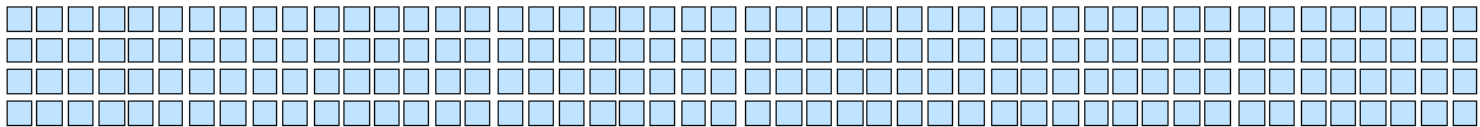












Приклад

```
__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

 Open in Colab

 Open in Colab

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 🏁