

The background of the slide is a dark blue field filled with a complex, interconnected network of thin, light blue lines. These lines form a web-like structure with numerous small, glowing blue dots at the nodes. Some areas of the network are more dense than others, creating a sense of depth and complexity. The overall aesthetic is technological and modern.

Parallel Programming

Parallel Histogram

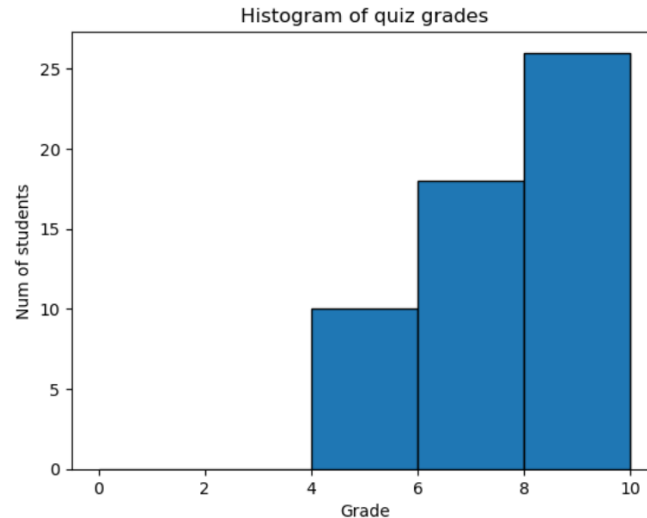
Phạm Trọng Nghĩa
ptnghia@fit.hcmus.edu.vn

Overview

- The “histogram” task
- Sequential implementation
- Parallel implementation
 - A naïve kernel
 - A kernel using atomic operation
 - A fast kernel using atomic operation

Histogram

- A method for extracting notable features and patterns from large data sets
 - *Feature extraction* for object recognition in images
 - *Fraud detection* in credit card transactions
 - *Correlating* heavenly object movements in astrophysics
- **Basic histograms**: for each element in the data set, use the value to identify a “bin counter” to increment



The “histogram” task

Input

0	5	4	0
7	1	1	2
2	3	0	6

Histogram 4 bin

Bin	Index	0	1	2	3
	Range	[0, 1]	[2, 3]	[4, 5]	[6, 7]
Count					

The “histogram” task

Input

0	5	4	0
7	1	1	2
2	3	0	6

Histogram 8 bin

Bin	Index	0	1	2	3	4	5	6	7
	Range	0	1	2	3	4	5	6	7
Count									

Histogram 4 bin

Bin	Index	0	1	2	3
	Range	[0, 1]	[2, 3]	[4, 5]	[6, 7]
Count		5	3	2	2

The “histogram” task

Input

0	5	4	0
7	1	1	2
2	3	0	6

Histogram 8 bin

Bin	Index	0	1	2	3	4	5	6	7
	Range	0	1	2	3	4	5	6	7
Count		3	2	2	1	1	1	1	1

Histogram 4 bin

Bin	Index	0	1	2	3
	Range	[0, 1]	[2, 3]	[4, 5]	[6, 7]
Count		5	3	2	2

Sequential implementation

```
void computeHistOnHost(int *in, int *hist, int n, int nBins)
{
    for (int i = 0; i < nBins; i++)
        hist[i] = 0;
    for (int i = 0; i < n; i++)
        hist[computeBin(in[i])] += 1;
}
```

Compute **bin** which **in[i]** falls into

In the rest of this lecture, we'll assume: **in[i] ≥ 0** and **computeBin(in[i]) = in[i]**

Parallel implementation: a naïve kernel

- Let each thread take care of one element in the input array
- Thread will identify the bin of its element and increase the bin counter by 1

```
__global__ void computeHistOnDevice1(int *in, int *hist, int n, int nBins)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    if (i < nBins)
        hist[i] = 0;

    if (i < n)
        hist[in[i]] += 1;
}
```

Need synchronize globally, but we can only synchronize within each block 😞
One simple solution: from host, call **cudaMemset** function 😊

Read-Modify-Write (RMW); what if more than one threads RMW the same location? 😞

Parallel implementation: a naïve kernel

- A kernel using atomic operation: solve the conflict problem when many threads RMW the same location

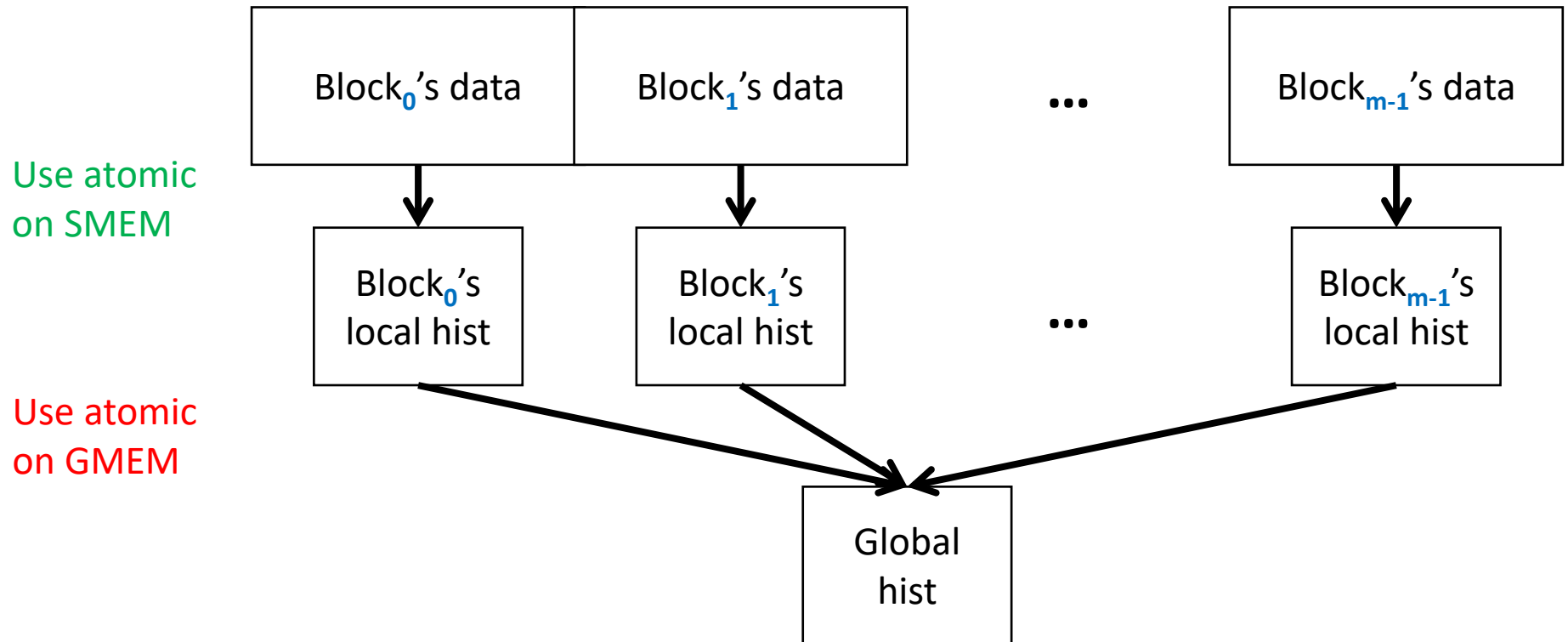
```
__global__ void computeHistOnDevice2(int *in, int *hist, int n, int nBins)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n)
        hist[in[i]] += 1;
        atomicAdd(&hist[in[i]], 1);
}
```

If many threads RMW the same location, they will be **serialized**:
threads will RMW in turn
→ Guarantee correctness, but can hurt speed

Coding exercise:

file “10-Histogram.cu”

A fast kernel using atomic operation



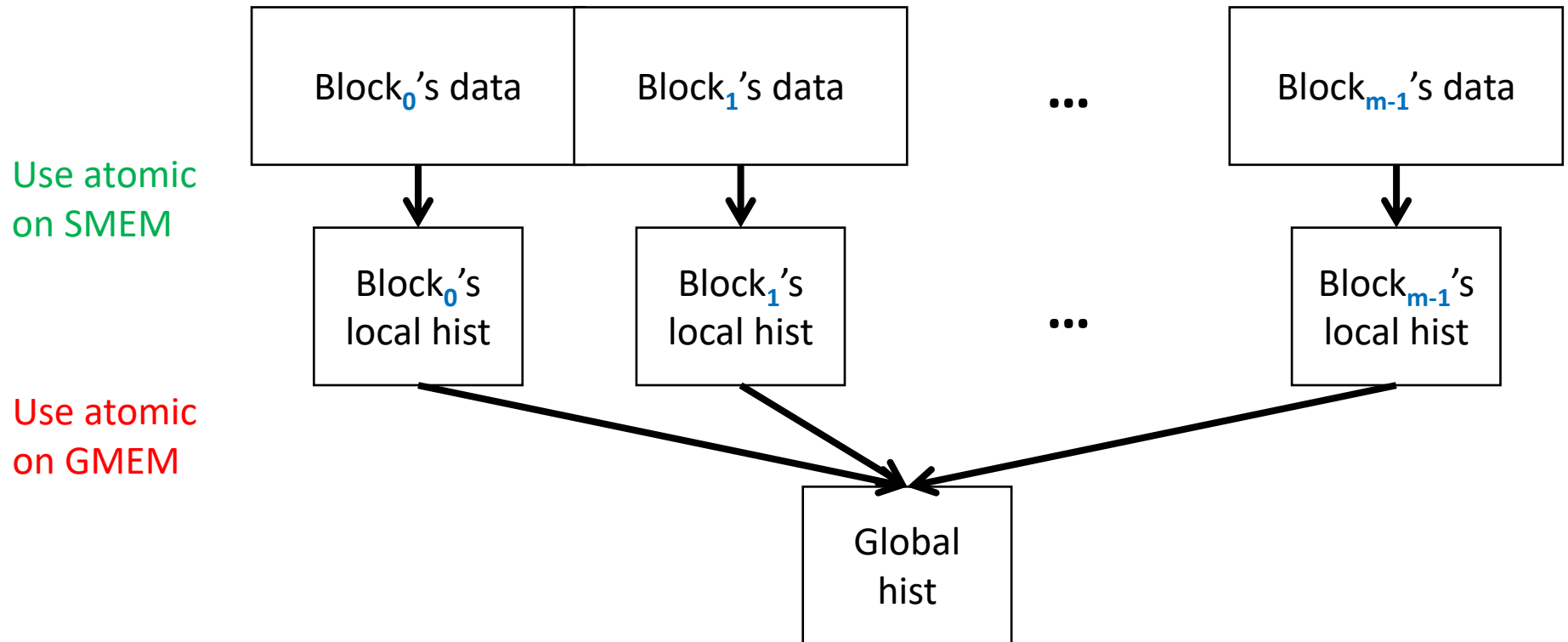
Will it run faster?

Consider a bin; assume in each block's data chunk there are 2 values falling into this bin

The old way costs: ? **atomicAdd's on GMEM**

This way costs: ?

A fast kernel using atomic operation



Will it run faster?

Consider a bin; assume in each block's data chunk there are 2 values falling into this bin

The old way costs: **2m** atomicAdd's on GMEM

This way costs: **2m** atomicAdd's on SMEM, 2 atomicAdd's/block and m blocks run in parallel
+ **m** atomicAdd's on GMEM

Coding exercise:

file “10-Histogram.cu”

Reference

- [1] Wen-Mei, W. Hwu, David B. Kirk, and Izzat El Hajj. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 2022
- [2] Cheng John, Max Grossman, and Ty McKercher. *Professional Cuda C Programming*. John Wiley & Sons, 2014
- [3] Illinois GPU course

<https://wiki.illinois.edu/wiki/display/ECE408/ECE408+Home>



THE END