

# Histogram

Trần Trung Kiên  
ttkien@fit.hcmus.edu.vn

Cập nhật lần cuối: 01/12/2021



KHOA CÔNG NGHỆ THÔNG TIN  
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

**fit@hcmus**

# Nội dung

- ☐ Bài toán histogram
- ☐ Cài đặt tuần tự
- ☐ Cài đặt song song
  - ☐ Hàm kernel “ngây thơ”
  - ☐ Hàm kernel dùng toán tử atomic
  - ☐ Tăng tốc hàm-kernel-dùng-toán-tử-atomic

# Bài toán histogram

Input

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

Histogram 4 bin

Bin	chỉ số	0	1	2	3
	miền giá trị	[0, 1]	[2, 3]	[4, 5]	[6, 7]
Count					

# Bài toán histogram

Input

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

Histogram 8 bin

Bin	chỉ số	0	1	2	3	4	5	6	7
	miền giá trị	0	1	2	3	4	5	6	7
Count									

Histogram 4 bin

Bin	chỉ số	0	1	2	3
	miền giá trị	[0, 1]	[2, 3]	[4, 5]	[6, 7]
Count		5	3	2	2

# Bài toán histogram

Input

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

Histogram 8 bin

Bin	chỉ số	0	1	2	3	4	5	6	7
	miền giá trị	0	1	2	3	4	5	6	7
Count		3	2	2	1	1	1	1	1

Histogram 4 bin

Bin	chỉ số	0	1	2	3
	miền giá trị	[0, 1]	[2, 3]	[4, 5]	[6, 7]
Count		5	3	2	2

# Cài đặt tuần tự

```
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;
}
```

Cho biết **in[i]** thuộc **bin** nào

Trong các phần còn lại, ta sẽ giả sử: **in[i]** không âm và **computeBin(in[i]) = in[i]**

# Cài đặt song song: hàm kernel “ngây thơ”

- Để mỗi thread phụ trách một phần tử trong mảng đầu vào
- Thread sẽ xem phần tử mà mình phụ trách thuộc bin nào và cộng 1 vào biến đếm của bin đó

```
__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;
}
```

Cần đồng bộ hóa, nhưng ta chỉ thể đồng bộ hóa trong mỗi block 😞  
Một giải pháp đơn giản: từ host dùng hàm **cudaMemset** 😊

Read-Modify-Write (RMW); nếu có nhiều thread cùng RMW một vị trí thì kết quả sẽ như thế nào? 😞

# Hàm kernel dùng toán tử atomic: giải quyết vấn đề đụng độ khi có nhiều thread cùng RMW một vị trí

```
__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);
}
```

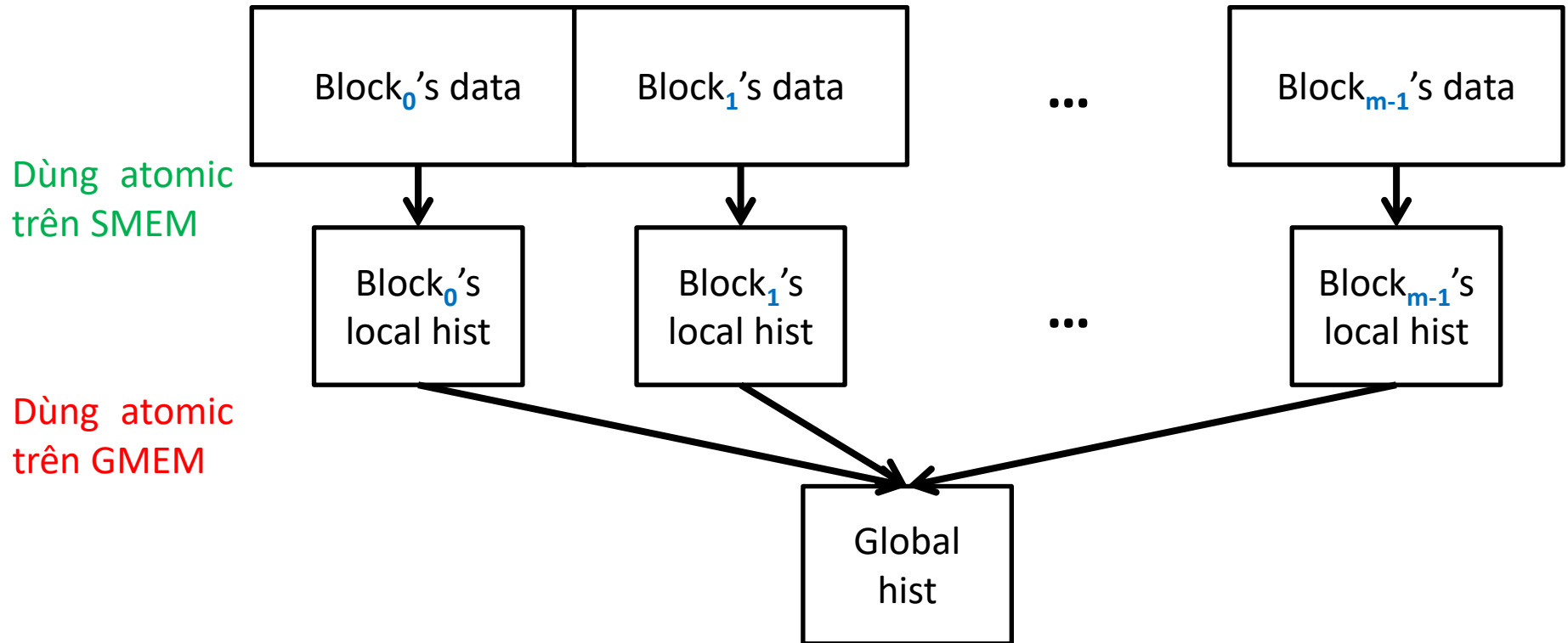


Nếu có nhiều thread cùng RMW một vị trí thì sẽ **tuần tự hóa**:  
lần lượt từng thread sẽ thực hiện RMW  
→ đảm bảo tính đúng đắn, nhưng sẽ ảnh hưởng đến tốc độ



**Bài tập code: file “11-Histogram.cu”**

# Tăng tốc hàm-kernel-dùng-toán-tử-atomic



## Có giúp tăng tốc không?

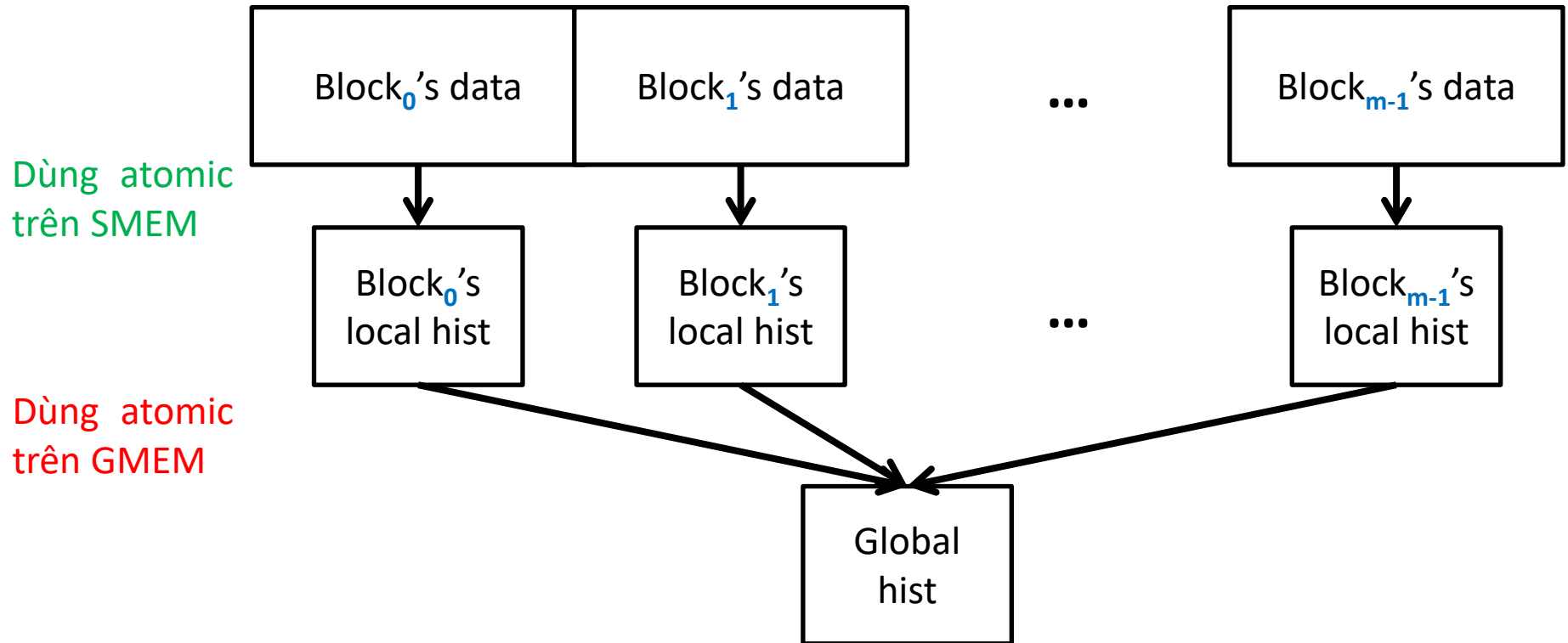
Xét một bin; giả sử trong khối dữ liệu của mỗi block có 2 giá trị thuộc bin này

Cách cũ tốn: ? phép atomicAdd trên GMEM

Cách này tốn: ? phép atomicAdd trên SMEM

? phép atomicAdd trên GMEM

# Tăng tốc hàm-kernel-dùng-toán-tử-atomic



## Có giúp tăng tốc không?

Xét một bin; giả sử trong khối dữ liệu của mỗi block có 2 giá trị thuộc bin này

Cách cũ tốn: **2m** phép atomicAdd trên GMEM

Cách này tốn: **2m** phép atomicAdd trên SMEM, 2 phép/block và m block chạy song song  
**m** phép atomicAdd trên GMEM

**Bài tập code: file “11-Histogram.cu”**

# Thí nghiệm

- ☐ Mạng gồm 10,240,000 phần tử số nguyên không âm, mỗi phần tử được phát sinh ngẫu nhiên trong  $[0, 1023]$ ; 1024 bin
- ☐ Block size 256
- ☐ GPU CC 2.0

# Thí nghiệm

Cách cài đặt	Thời gian (ms)
Dùng atomic	7.970
Dùng atomic + tăng tốc bằng local hist	2.688

# Thí nghiệm

Cách cài đặt	Thời gian (ms)
Dùng atomic	7.970
Dùng atomic + tăng tốc bằng local hist	2.688
Dùng atomic + tăng tốc bằng local hist + 4 phần tử trong mảng in / thread	

# Thí nghiệm

Cách cài đặt	Thời gian (ms)
Dùng atomic	7.970
Dùng atomic + tăng tốc bằng local hist	2.688
Dùng atomic + tăng tốc bằng local hist + 4 phần tử trong mảng in / thread	<b>1.397</b>

Tại sao để mỗi thread phụ trách nhiều phần tử trong mảng in hơn lại giúp ích?

Một thread nên phụ trách các phần tử nào trong mảng in?  
Các phần tử **kề nhau**?