Cách thực thi song song trong CUDA (phần 3)

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

Cập nhật: 10/11/2021

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

fit@hcmus

Tổng thể

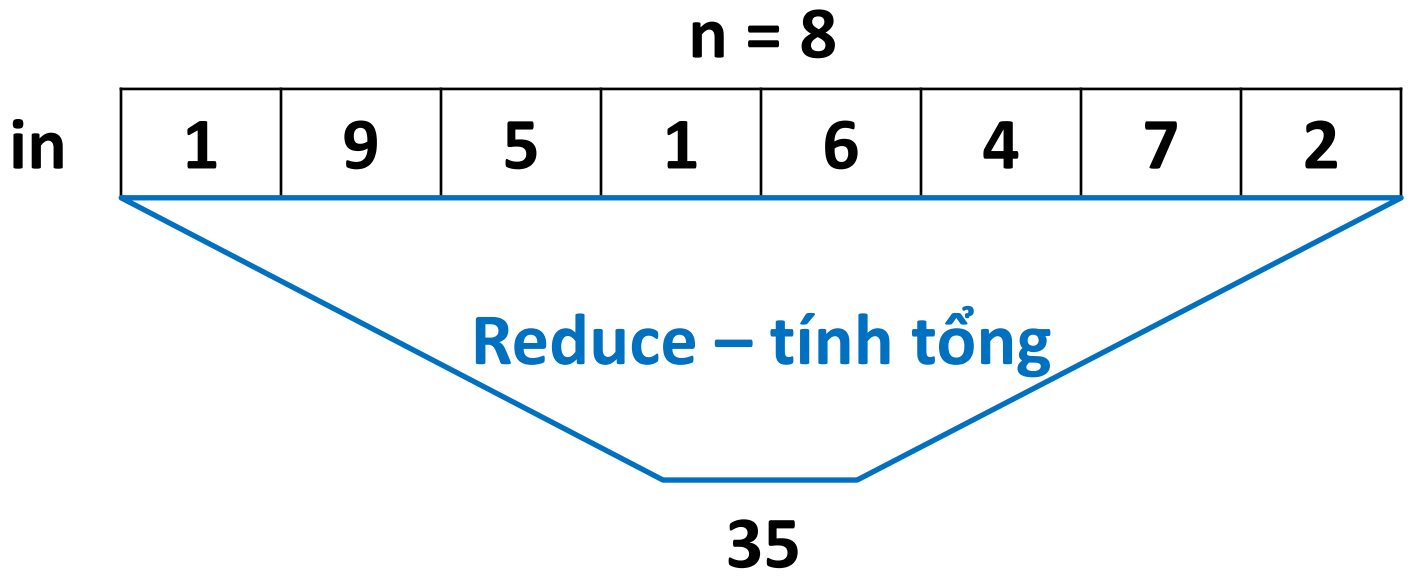
Áp dụng hiểu biết về cách thực thi song song trong CUDA để tối ưu hóa chương trình thực hiện "reduction"

- Bài toán "reduction"
- Cài đặt tuần tự
- Cài đặt song song
  - Hàm kernel 1
  - Hàm kernel 2: giảm số lượng warp bị phân kỳ
  - Hàm kernel 3: giảm số lượng warp bị phân kỳ + …

Bài toán "reduction"

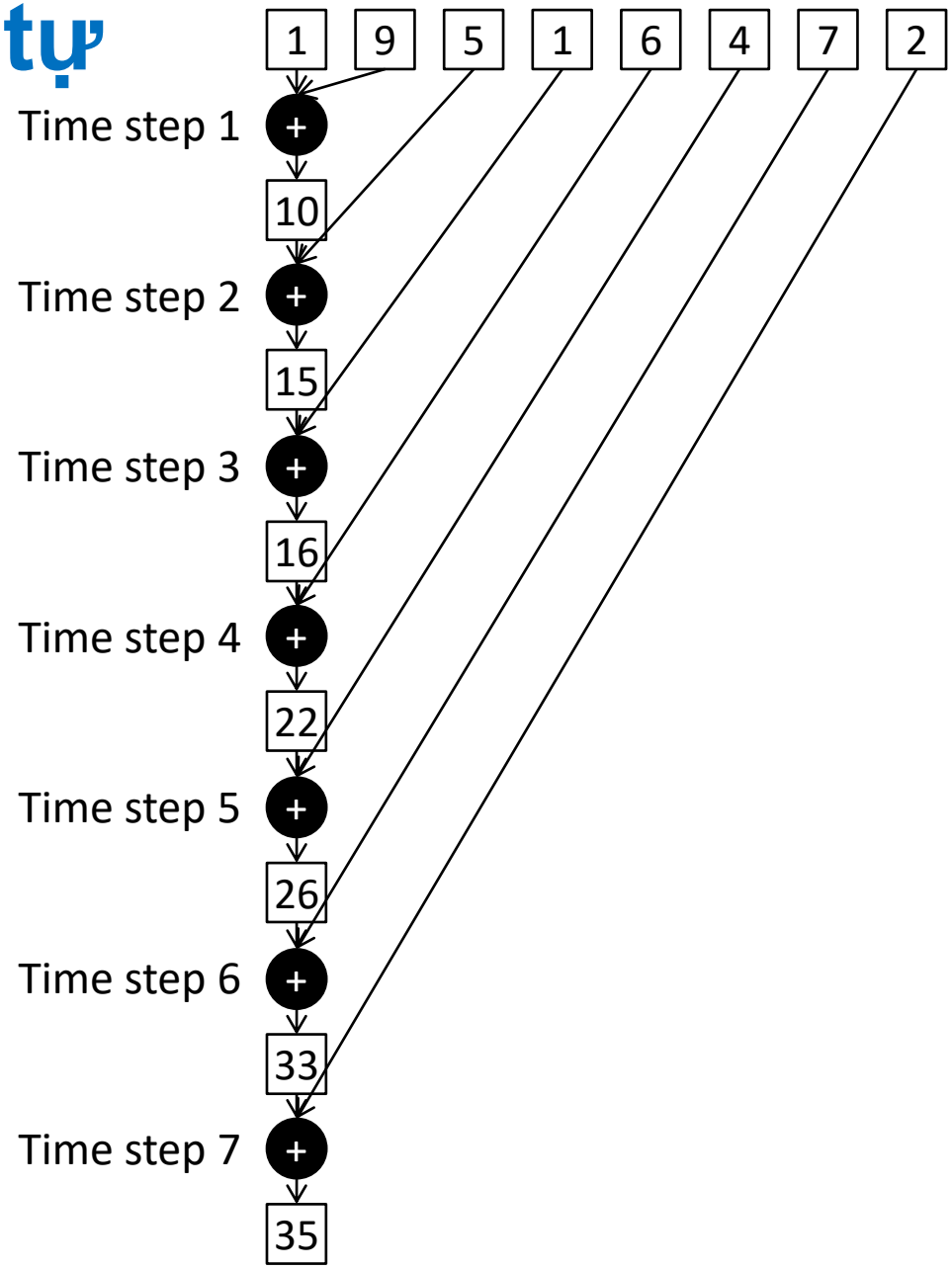
Cho mảng đầu vào **in** gồm **n** phần tử

Tính tổng (hoặc tích, max, min, …) của mảng này



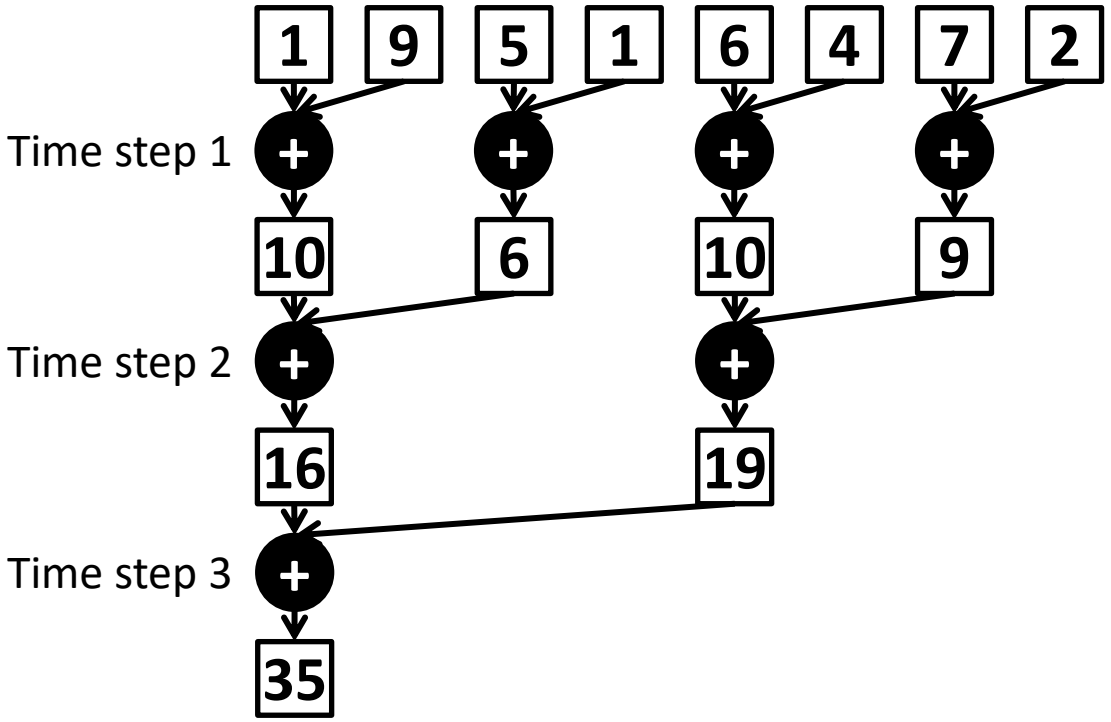
Cài đặt tuần tự

```
int reduceOnHost(int *in, int n)
{
   int s = in[0];
   for (int i = 1; i < n; i++)
      s += in[i];
   return s;
}
```



Time (số time step): 7 = n-1 = O(n)
Work (số phép cộng): 7 = n-1 = O(n)

Cài đặt song song – ý tưởng



Time: ?

Work: ?

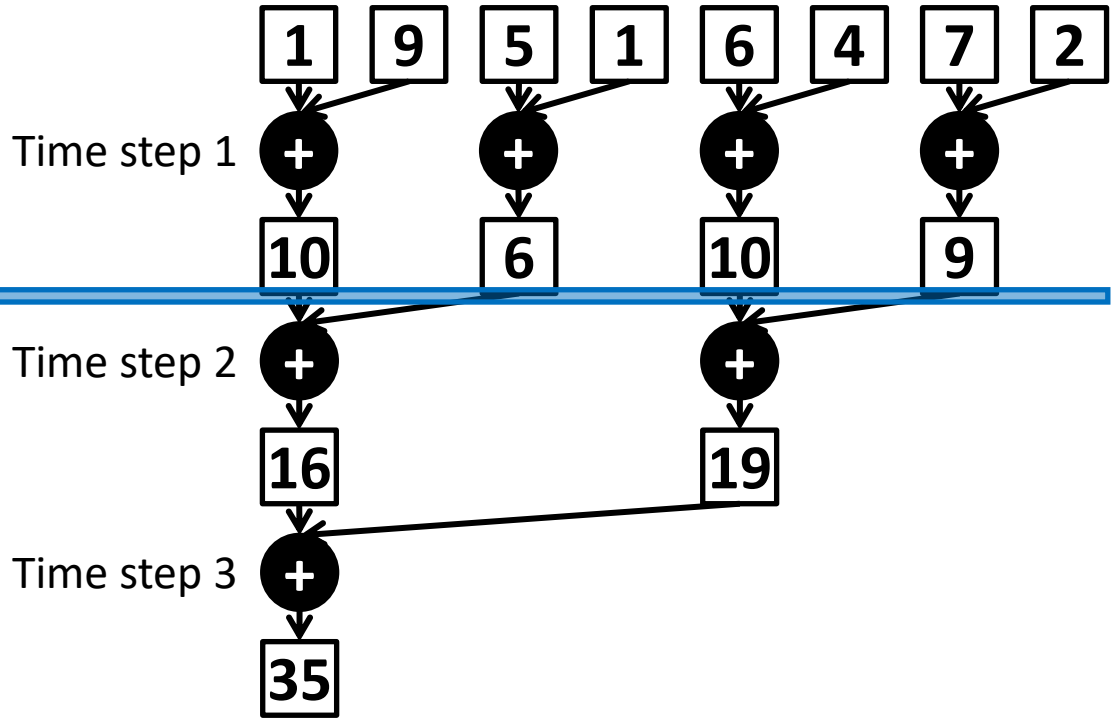
Cài đặt song song – ý tưởng

Có cần **đồng bộ hóa** trước khi qua bước tiếp theo?

Nhưng: trong hàm kernel ta chỉ có thể đồng bộ hóa các thread thuộc cùng một block

Nếu n <= 2*kích-thước-block thì ta có thể gọi hàm kernel với một block để thực hiện (nhưng nếu n nhỏ vậy thì không cần đến GPU)

Nếu n > 2*kích-thước-block thì phải làm sao?



Time: 3 = $\log_2 n$ = $O(\log_2 n)$
Work: 7 = n-1 = O(n) = work của cài đặt tuần tự
(Lúc sau, sẽ học những thuật toán mà để cài đặt song song được thì phải tốn nhiều work hơn cài đặt tuần tự)

Cài đặt song song – ý tưởng

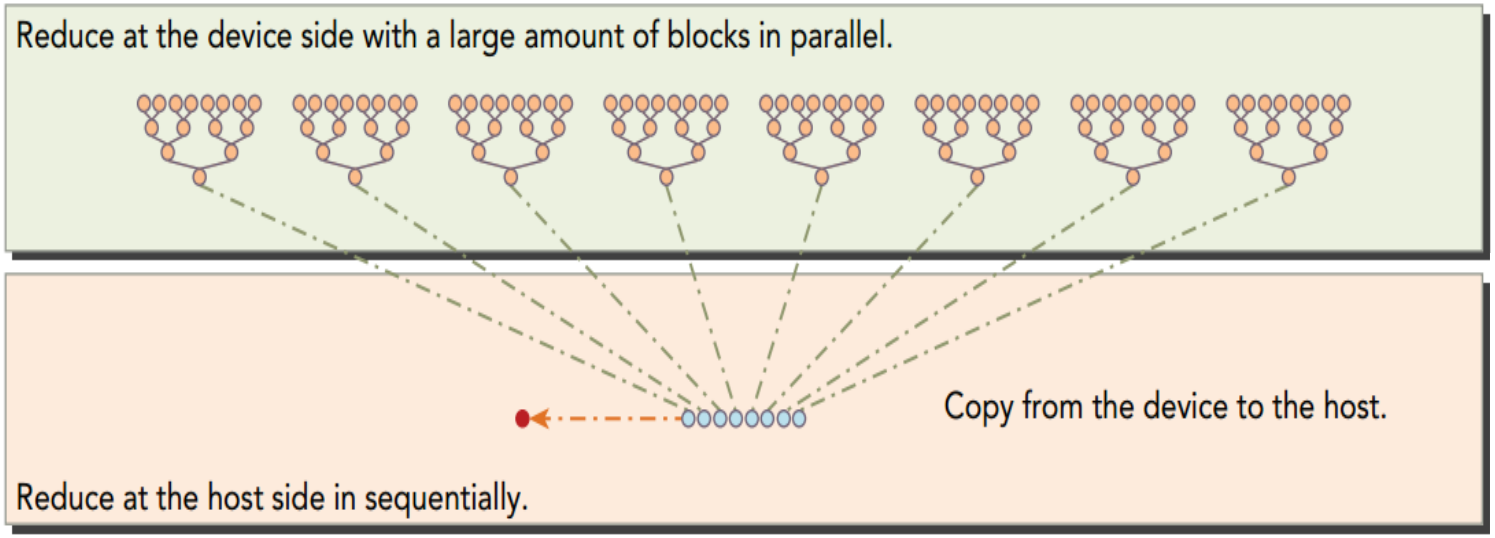
Nếu n > 2*kích-thước-block thì phải làm sao?

- Với một lần gọi hàm kernel thì có thể làm được gì?

  Reduce cục bộ trong từng block
- Sau khi reduce trong từng block thì được mảng output có bao nhiêu phần tử?

  n / (2*kích-thước-block)
- Và phải làm gì tiếp với mảng output này để ra được kết quả cuối cùng?

  Tính tổng của mảng output này

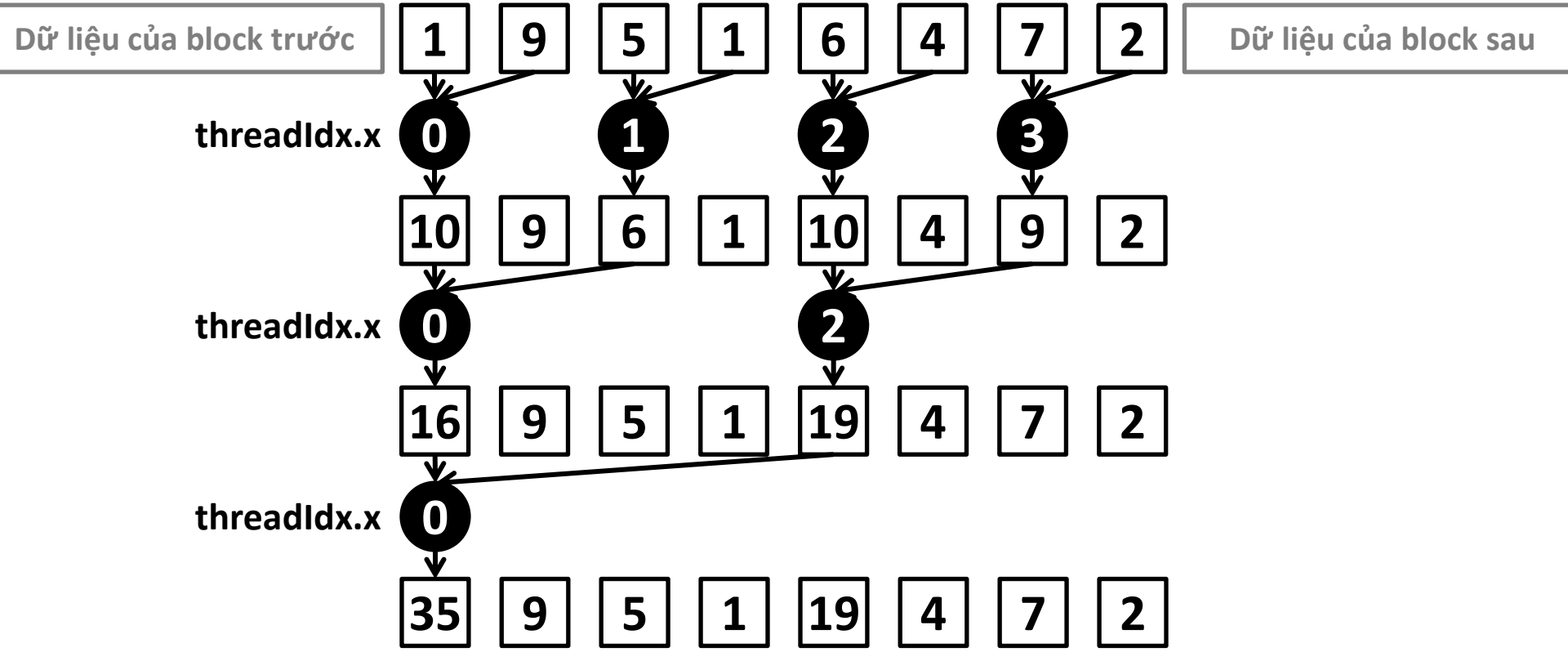


Hoặc cũng có thể gọi tiếp hàm kernel nhiều lần với input là output của lần gọi hàm kernel trước đó

Nguồn ảnh: John Cheng et al. Professional CUDA C Programming. 2014

Cài đặt song song – ý tưởng reduce trong từng block

Xét 1 block gồm 4 thread



Cài đặt song song – hàm kernel reduce trong từng block

Giả sử: 2*kích-thước-block = 2^k

Mảng đầu ra gồm số-block phần tử, mỗi phần tử sẽ chứa tổng của khối dữ liệu mà block tương ứng phụ trách

```
__global__ void reduceOnDevice1(int *in, int *out, int n)
{
    int i = blockIdx.x * blockDim.x * 2 + threadIdx.x * 2;

    for (int stride = 1; stride < 2 * blockDim.x; stride *= 2)
    {
        if ((threadIdx.x % stride) == 0)
            if (i + stride < n)
                in[i] += in[i + stride];
        __syncthreads(); // Synchronize within each block
    }

    if (threadIdx.x == 0)
        out[blockIdx.x] = in[blockIdx.x * blockDim.x * 2];
}
```

Thí nghiệm

- Kích thước mảng đầu vào: $2^{14} + 1$
- Phát sinh ngẫu nhiên giá trị số nguyên của mảng đầu vào trong [0, 255]
- So sánh thời gian chạy hàm kernel + thời gian post-kernel (thời gian host tính tổng các kết quả của từng block) với các block size khác nhau
- GPU CC 2.0

  Mỗi SM có thể chứa tối đa 8 block và 1536 thread (48 warp)

Thí nghiệm – kết quả

Block size	Grid size	Occupancy (%)	Num blocks / SM	Kernel time (ms)	Post-kernel time (ms)	Total time (ms)
1024	8193					
512	16385					
256	32769					
128	65537					

Thí nghiệm – kết quả

Block size	Grid size	Occupancy (%)	Num blocks / SM	Kernel time (ms)	Post-kernel time (ms)	Total time (ms)
1024	8193	67	1			
512	16385					
256	32769	100	6			
128	65537					

Thí nghiệm – kết quả

Block size	Grid size	Occupancy (%)	Num blocks / SM	Kernel time (ms)	Post-kernel time (ms)	Total time (ms)
1024	8193	67	1			
512	16385	100	3			
256	32769	100	6			
128	65537	67	8			

Thí nghiệm – kết quả

Block size	Grid size	Occupancy (%)	Num blocks / SM	Kernel time (ms)	Post-kernel time (ms)	Total time (ms)
1024	8193	67	1	11.996	**0.030**	12.026
512	16385	100	3	7.772	0.059	7.831
256	32769	100	6	**6.937**	0.113	**7.050**
128	65537	67	8			

Trong mỗi block có bao nhiêu warp bị phân kỳ? (không xét block ở biên)

- **Stride = 1:**

  Tất cả các thread đều "bật"

  → không có warp nào bị phân kỳ

- **Stride = 2:**

  Chỉ có các thread có `threadIdx.x` chia hết cho 2 "bật"

  → tất cả các warp đều bị phân kỳ

- **Stride = 4, 8, …, 32:**

  Tất cả các warp vẫn đều bị phân kỳ

- **Stride = 64, 128, …:**

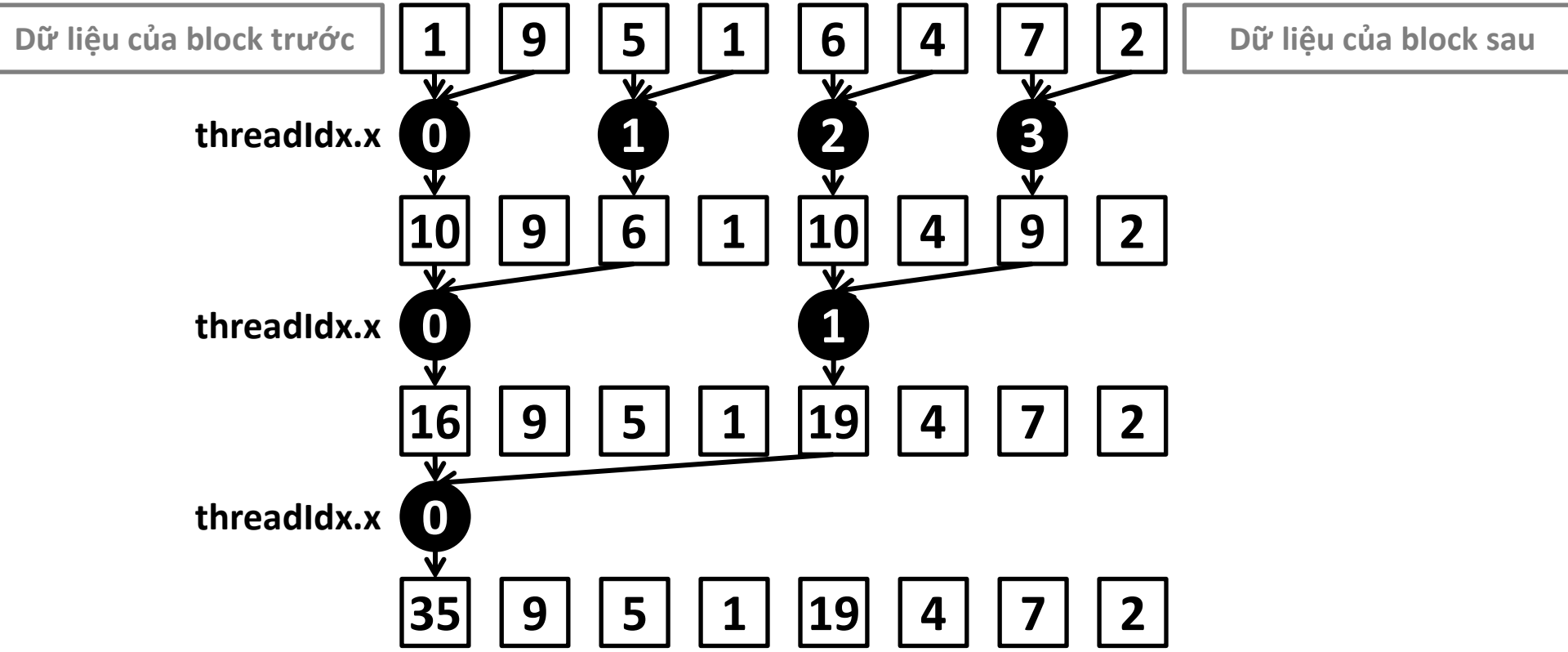
  Số lượng warp bị phân kỳ giảm dần về 1

Hàm kernel 2: giảm số lượng warp bị phân kỳ

- **Ý tưởng:** giảm số lượng warp bị phân kỳ ở mỗi bước bằng cách dồn các thread "bật" về đầu
- **Ví dụ:** xét block gồm 128 thread
  - Với stride = 1: tất cả 128 thread đều "bật"
  - Với stride = 2: 64 thread đầu sẽ "bật", còn lại sẽ "tắt"
  - Với stride = 4: 32 thread đầu sẽ "bật", còn lại sẽ "tắt"
  - …

Hàm kernel 2: giảm số lượng warp bị phân kỳ

Xét 1 block gồm 4 thread



Hàm kernel 2: giảm số lượng warp bị phân kỳ

```
__global__ void reduceOnDevice2(int *in, int *out, int n)
{
    int numElemsBeforeBlk = blockIdx.x * blockDim.x * 2;

    for (int stride = 1; stride < 2 * blockDim.x; stride *= 2)
    {
        int i = numElemsBeforeBlk + ...;
        if (threadIdx.x ...)
            if (i + stride < n)
                in[i] += in[i + stride];
        __syncthreads(); // Synchronize within each block
    }

    if (threadIdx.x == 0)
        out[blockIdx.x] = in[numElemsBeforeBlk];
}
```

Hàm kernel 2: giảm số lượng warp bị phân kỳ

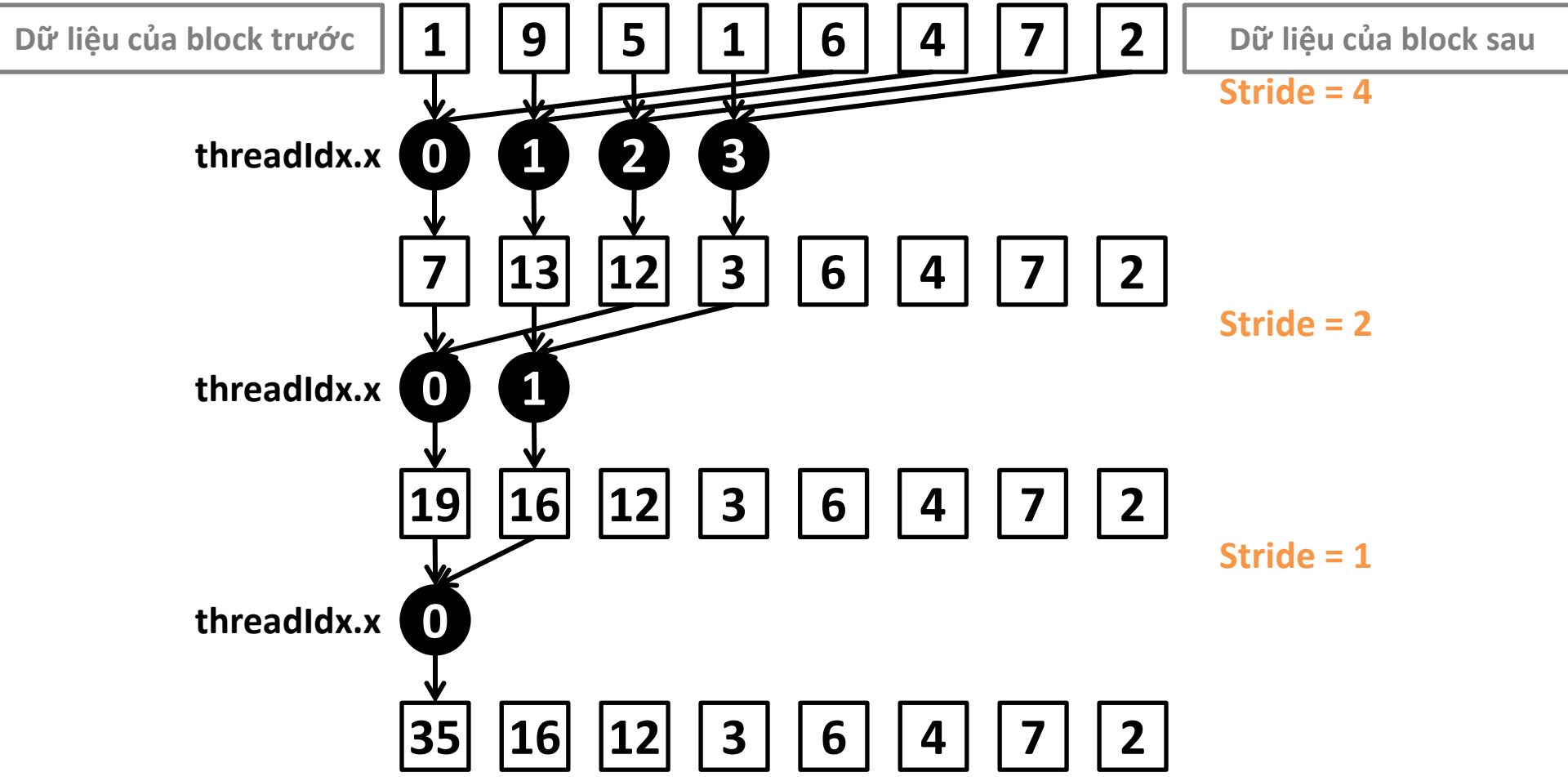
```
__global__ void reduceOnDevice2(int *in, int *out, int n)
{
    int numElemsBeforeBlk = blockIdx.x * blockDim.x * 2;

    for (int stride = 1; stride < 2 * blockDim.x; stride *= 2)
    {
        int i = numElemsBeforeBlk + threadIdx.x * 2 * stride;
        if (threadIdx.x < blockDim.x / stride)
            if (i + stride < n)
                in[i] += in[i + stride];
        __syncthreads(); // Synchronize within each block
    }

    if (threadIdx.x == 0)
        out[blockIdx.x] = in[numElemsBeforeBlk];
}
```

Thí nghiệm: so sánh 2 hàm kernel (với cùng block size là 256)

Function	Kernel time (ms)
reduceOnDevice1	6.937
reduceOnDevice2	**4.968**

Hàm kernel 3: giảm số lượng warp bị phân kỳ + ?

Xét 1 block gồm 4 thread



Hàm kernel 3: giảm số lượng warp bị phân kỳ + ?

Code hàm kernel 3: bài tập sắp tới ;-)

Thí nghiệm: so sánh 3 hàm kernel (với cùng block size là 256)

Function	Kernel time (ms)
reduceOnDevice1	6.937
reduceOnDevice2	4.968
reduceOnDevice3	**4.250**