Scan

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

Cập nhật lần cuối: 08/12/2021

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

fit@hcmus

Nội dung

- Bài toán “scan”
- Cài đặt tuần tự
- Cài đặt song song
  - Hàm kernel 1 - “work-inefficient”
  - Hàm kernel 2 - “work-efficient”

Bài toán "scan"

- **Inclusive scan**: $out[i] = \sum_{j=0}^{i} in[j]$

in	**1**	**9**	**5**	**1**	**6**	**4**	**7**	**2**
out								

Bài toán "scan"

- **Inclusive scan**: $out[i] = \sum_{j=0}^{i} in[j]$

in	**1**	**9**	**5**	**1**	**6**	**4**	**7**	**2**
out	**1**							

Bài toán "scan"

- **Inclusive scan**: $out[i] = \sum_{j=0}^{i} in[j]$

in	**1**	**9**	**5**	**1**	**6**	**4**	**7**	**2**
out	**1**	**10**						

Bài toán "scan"

- **Inclusive scan**: $out[i] = \sum_{j=0}^{i} in[j]$

in	1	9	5	1	6	4	7	2
out	1	10	15					

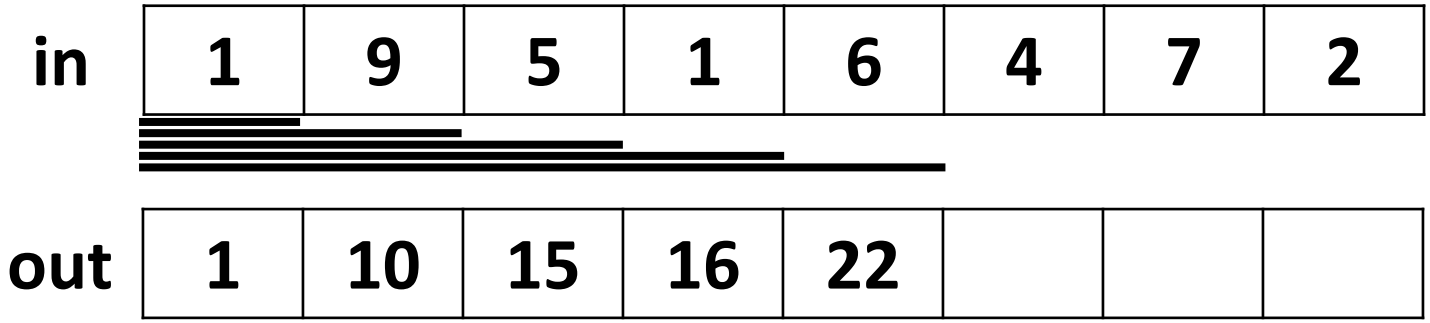
Bài toán "scan"

- **Inclusive scan**: $out[i] = \sum_{j=0}^{i} in[j]$

in	**1**	**9**	**5**	**1**	**6**	**4**	**7**	**2**
out	**1**	**10**	**15**	**16**				

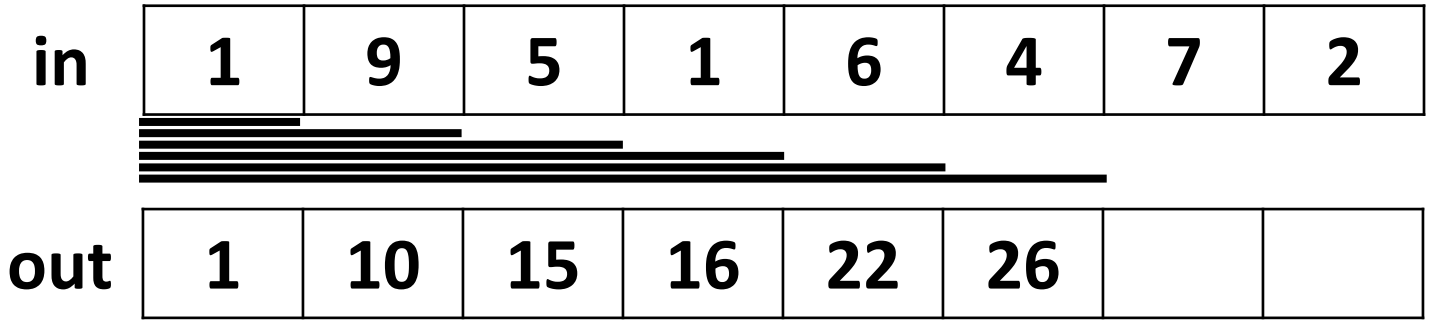
Bài toán "scan"

- **Inclusive scan**: $out[i] = \sum_{j=0}^{i} in[j]$



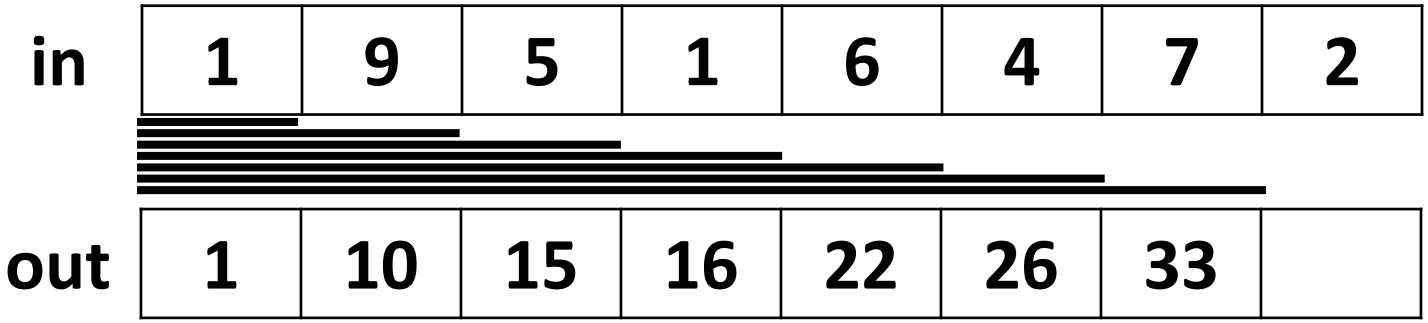
Bài toán "scan"

- **Inclusive scan**: $out[i] = \sum_{j=0}^{i} in[j]$



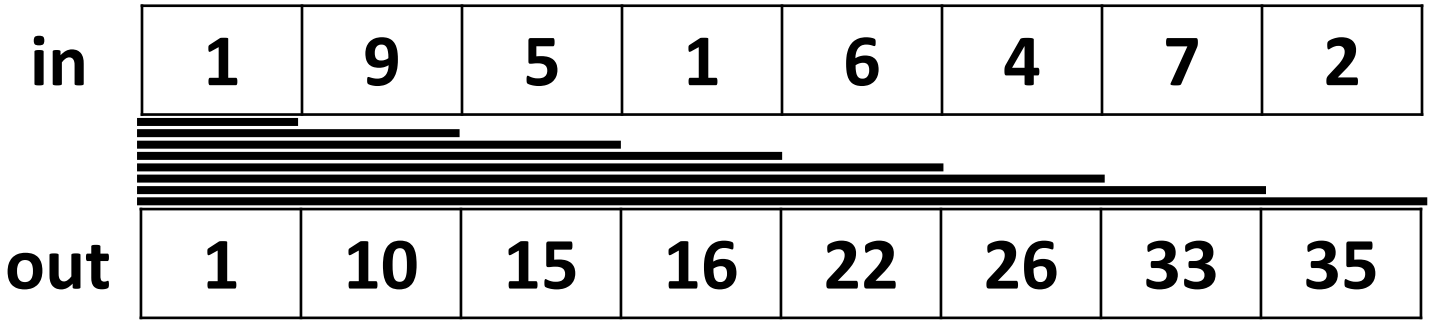
Bài toán "scan"

- **Inclusive scan**: $out[i] = \sum_{j=0}^{i} in[j]$



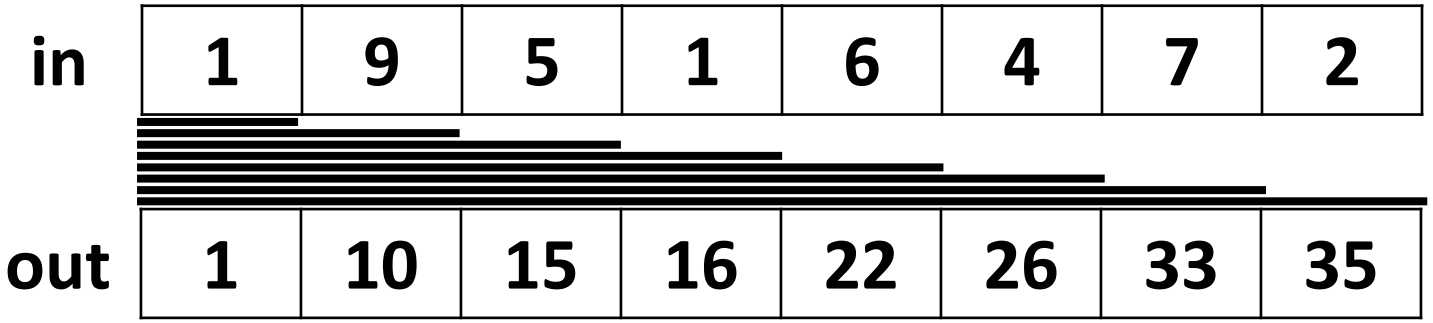
Bài toán "scan"

- **Inclusive scan**: $out[i] = \sum_{j=0}^{i} in[j]$

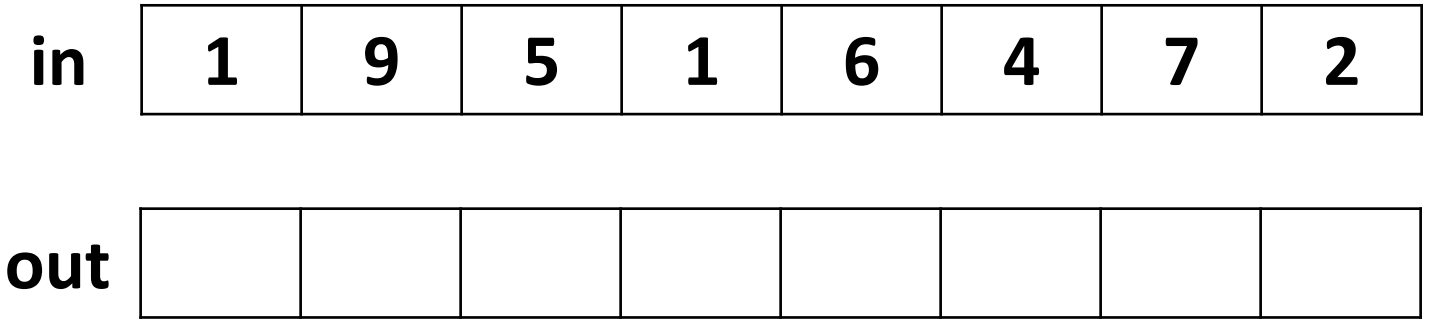


Bài toán "scan"

- **Inclusive scan**: $out[i] = \sum_{j=0}^{i} in[j]$

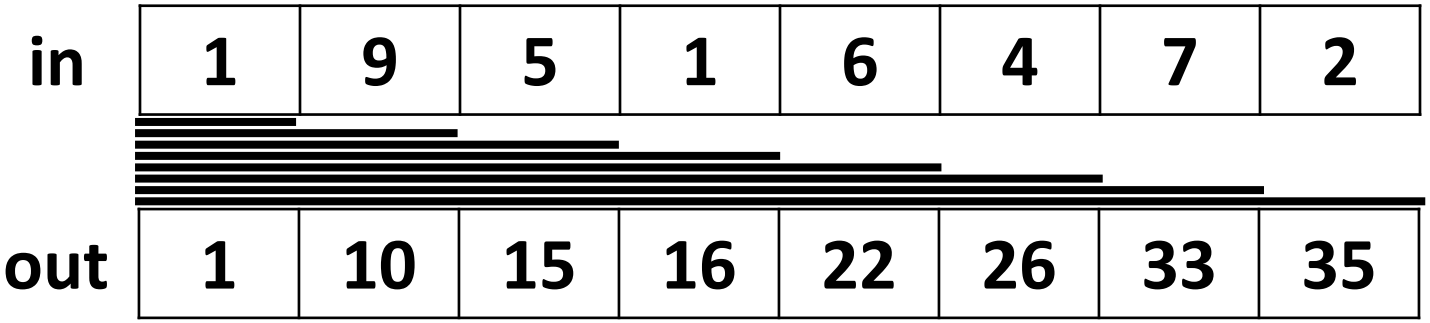


- **Exclusive scan**: $out[0] = 0, out[i] = \sum_{j=0}^{i-1} in[j] \; \forall i > 0$

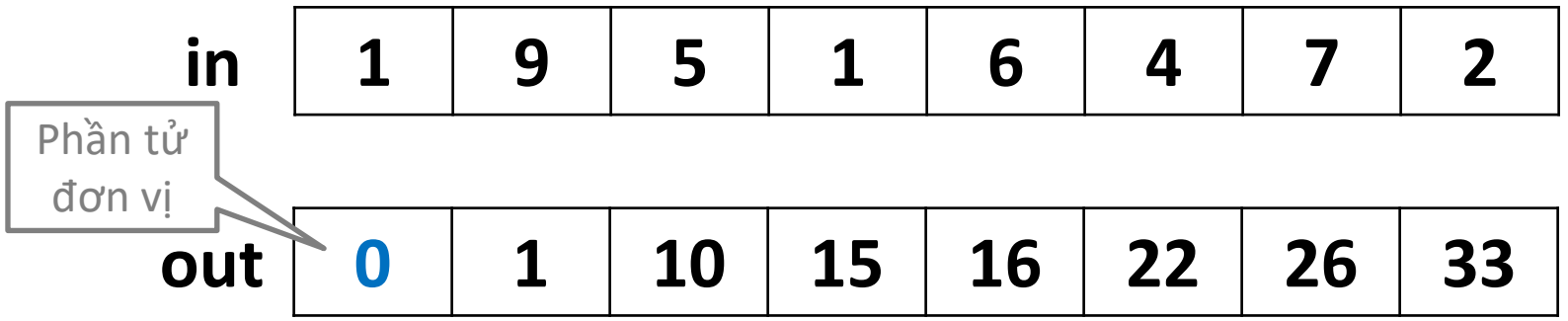


Bài toán "scan"

- **Inclusive scan**: $out[i] = \sum_{j=0}^{i} in[j]$



- **Exclusive scan**: $out[0] = 0, out[i] = \sum_{j=0}^{i-1} in[j]\ \forall i > 0$



- Ngoài phép cộng, cũng có thể áp dụng cho phép nhân, max, min, …
- Ở đây, ta sẽ tập trung làm inclusive scan với phép cộng

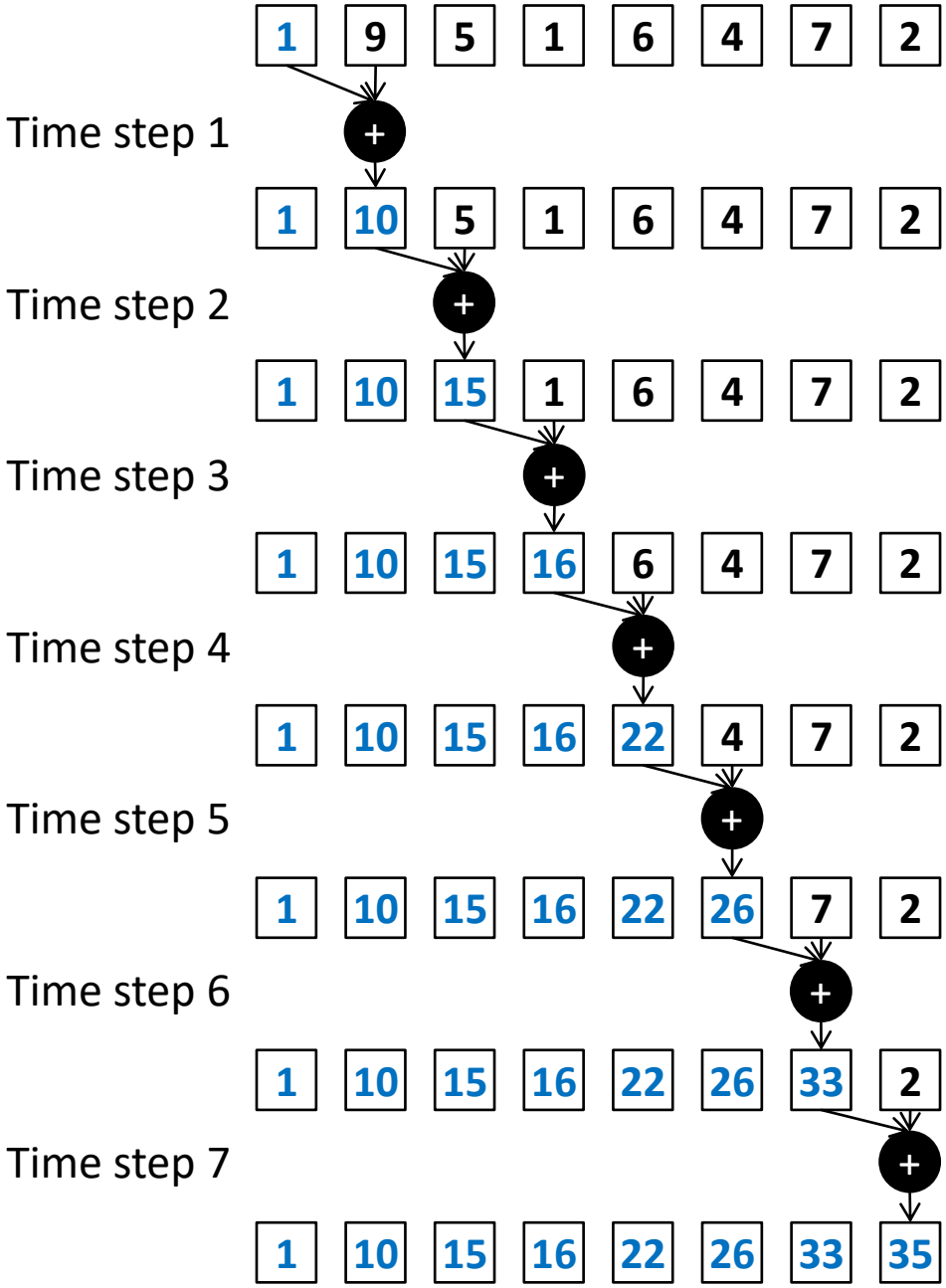
Scan là một dạng tính toán thường gặp trong nhiều ứng dụng ở ngoài kia

Ở buổi tới, ta sẽ thấy scan xuất hiện trong sort

Cài đặt tuần tự

```
void scanOnHost(int *in, int *out, int n)
{
    out[0] = in[0];
    for (int i = 1; i < n; i++)
    {
        out[i] = out[i - 1] + in[i];
    }
}
```

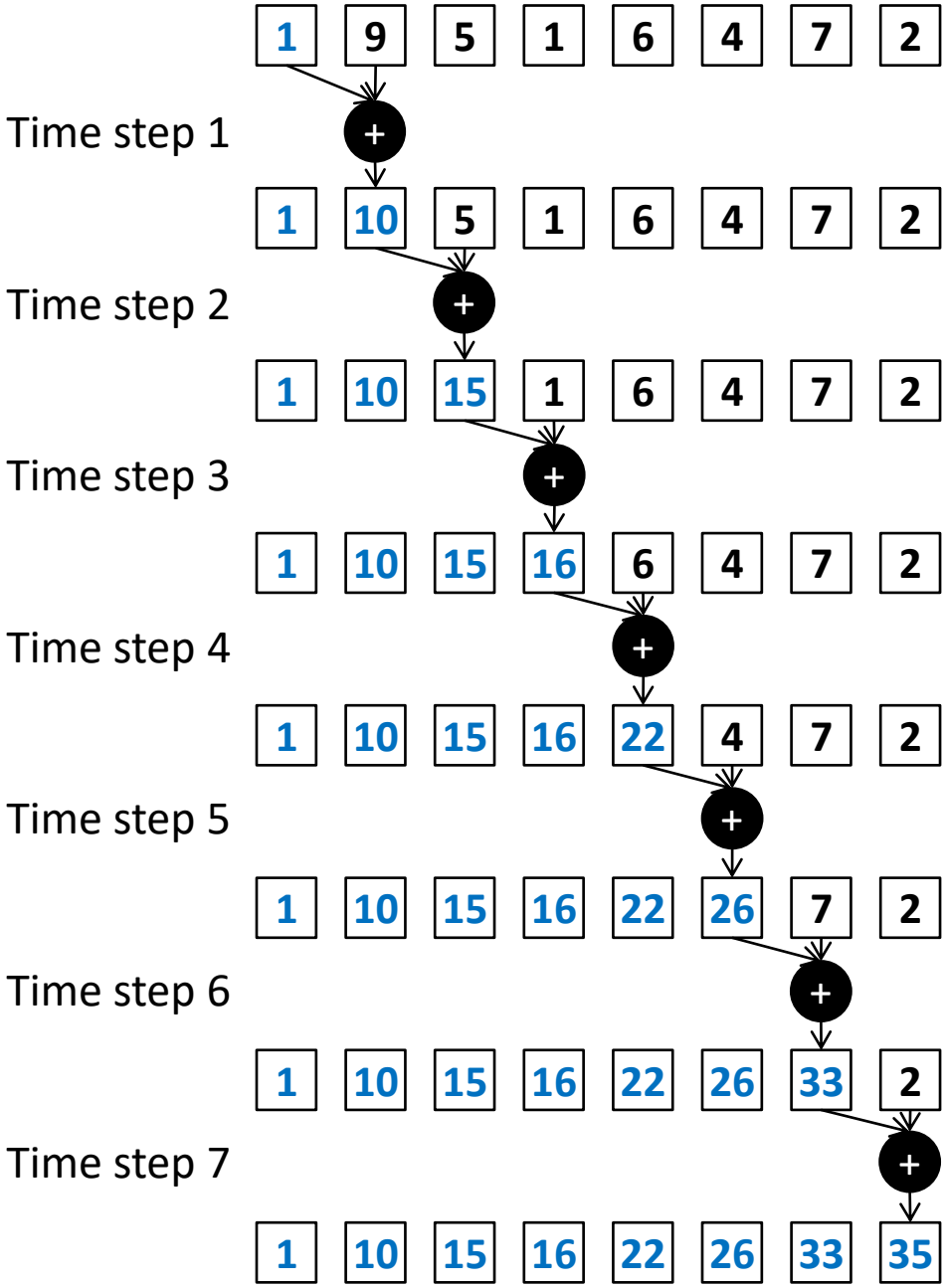
Time (số time step):
Work (số phép cộng):



Cài đặt tuần tự

```
void scanOnHost(int *in, int *out, int n)
{
    out[0] = in[0];
    for (int i = 1; i < n; i++)
    {
        out[i] = out[i - 1] + in[i];
    }
}
```

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

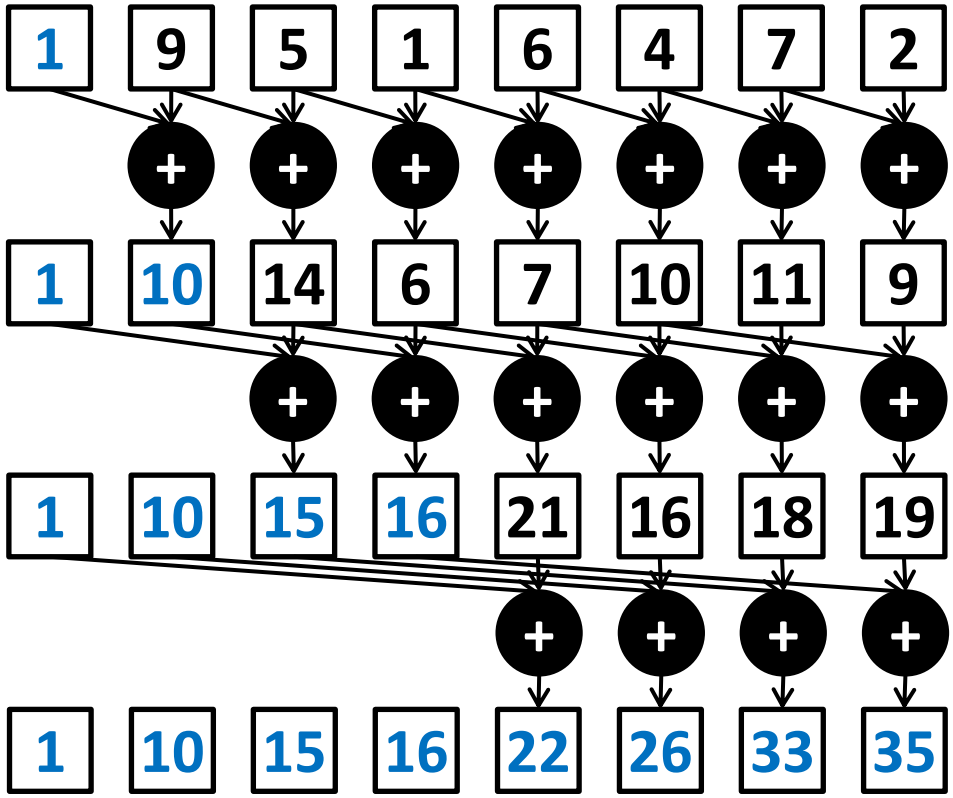


Cài đặt song song

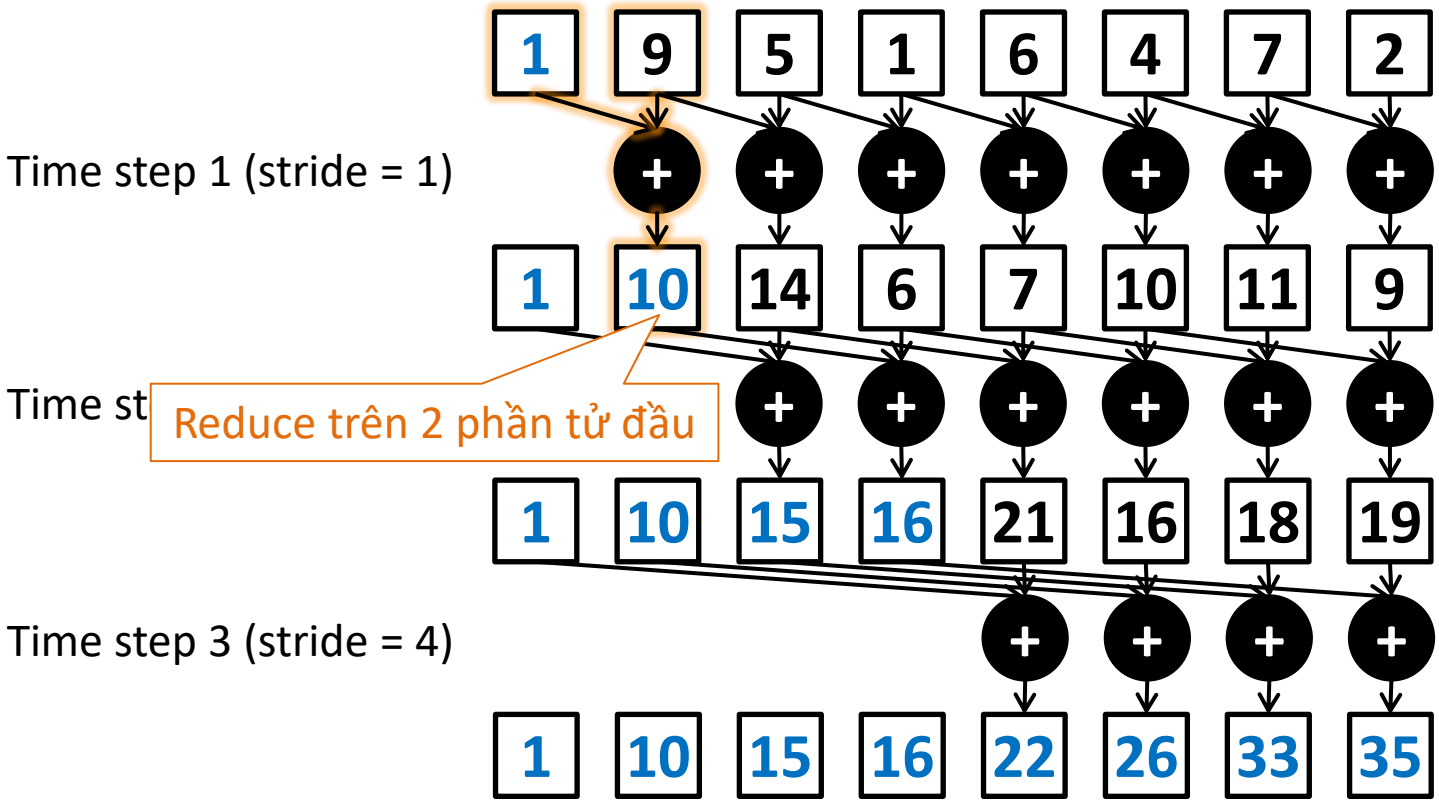
Time step 1 (stride = 1)

Time step 2 (stride = 2)

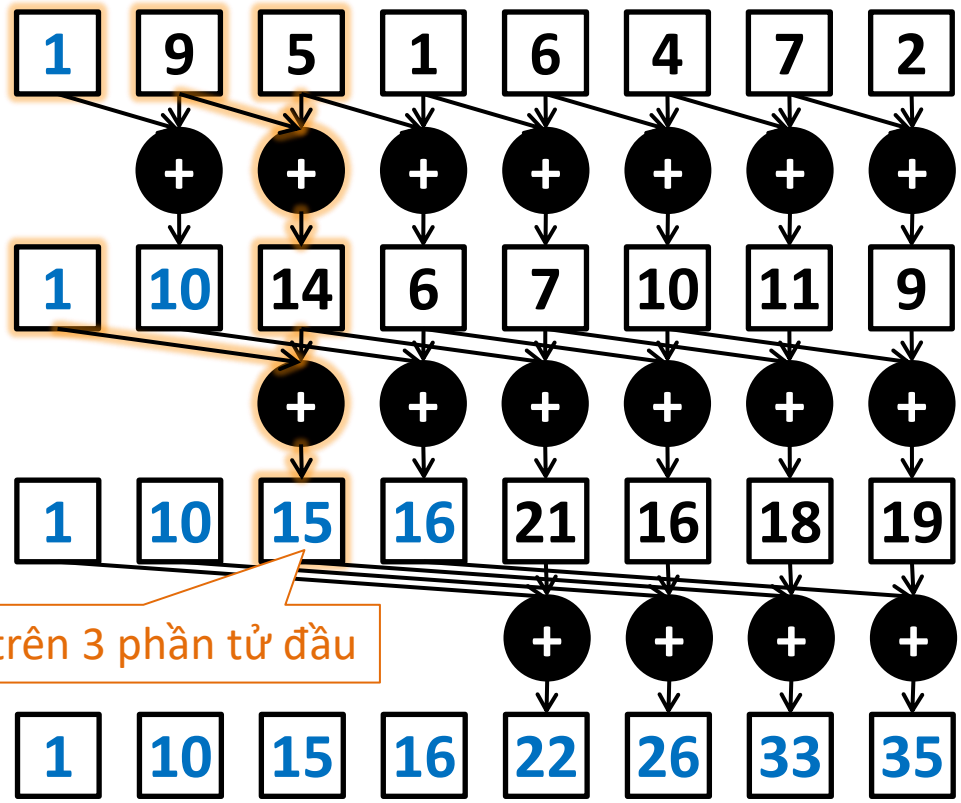
Time step 3 (stride = 4)



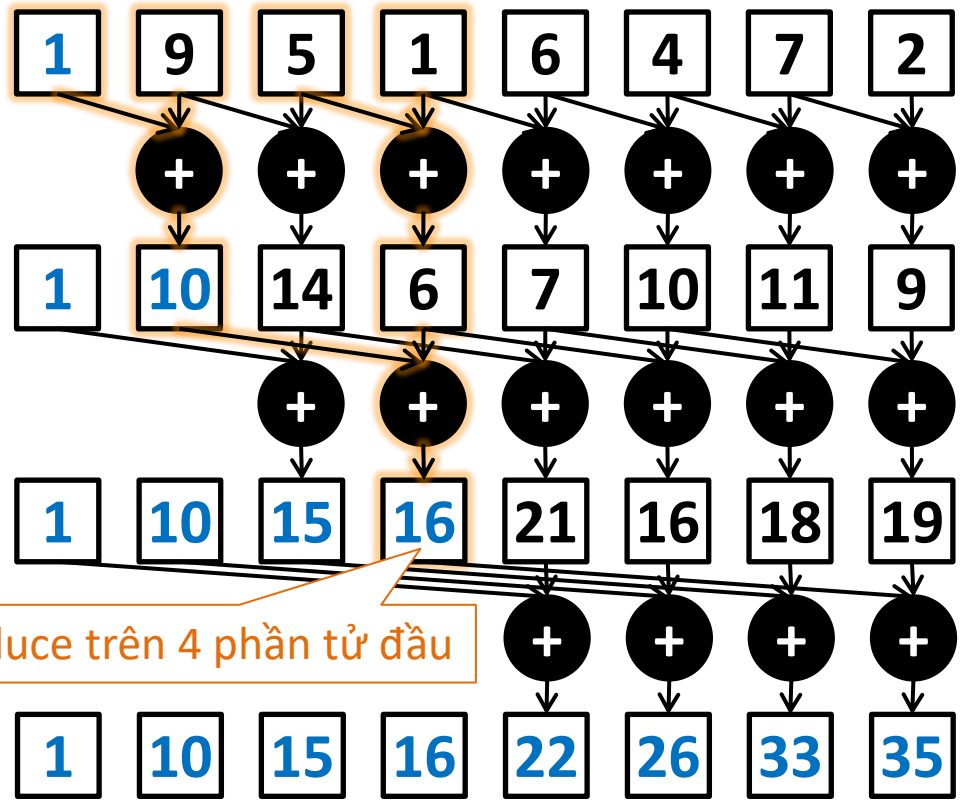
Cài đặt song song



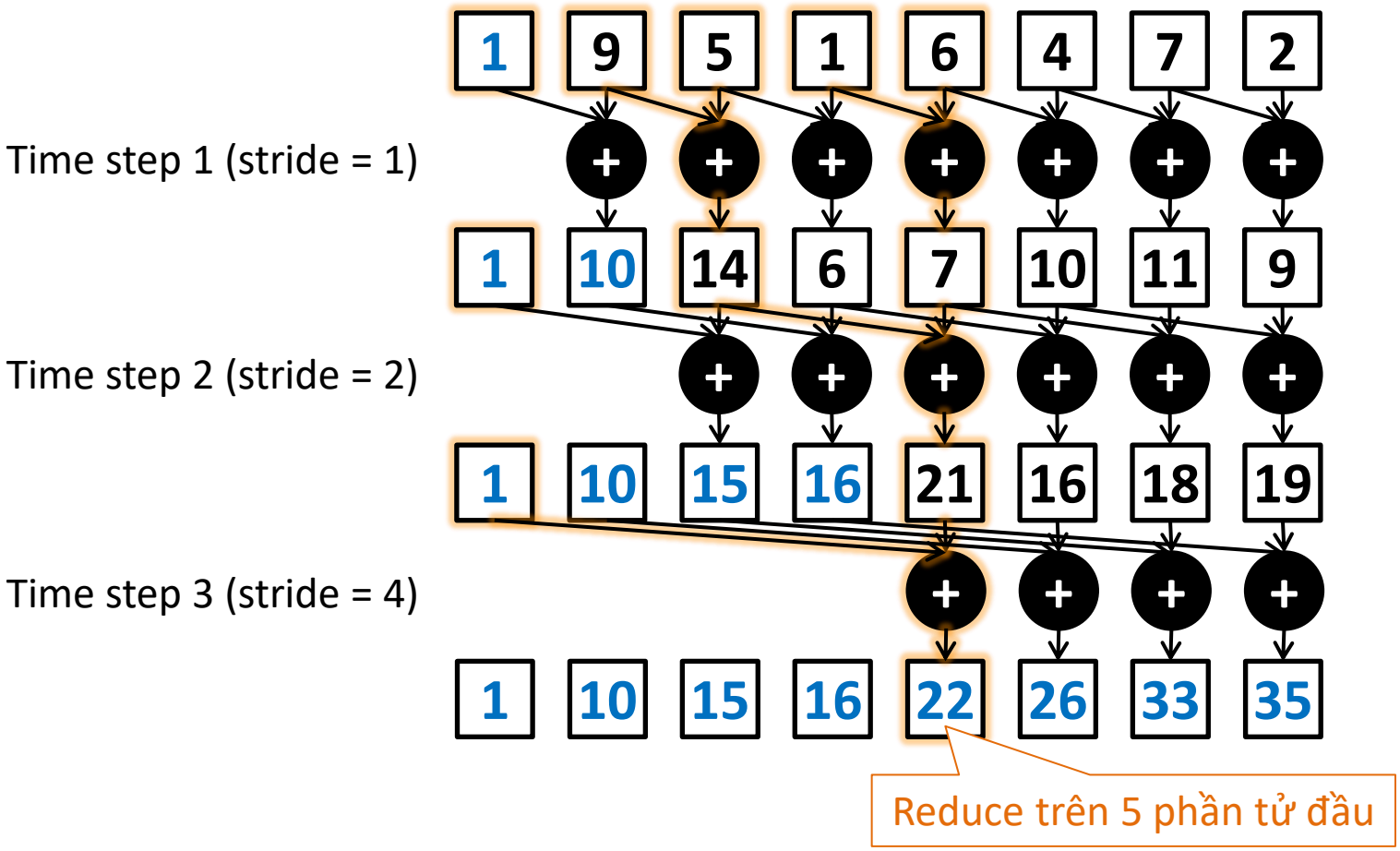
Cài đặt song song



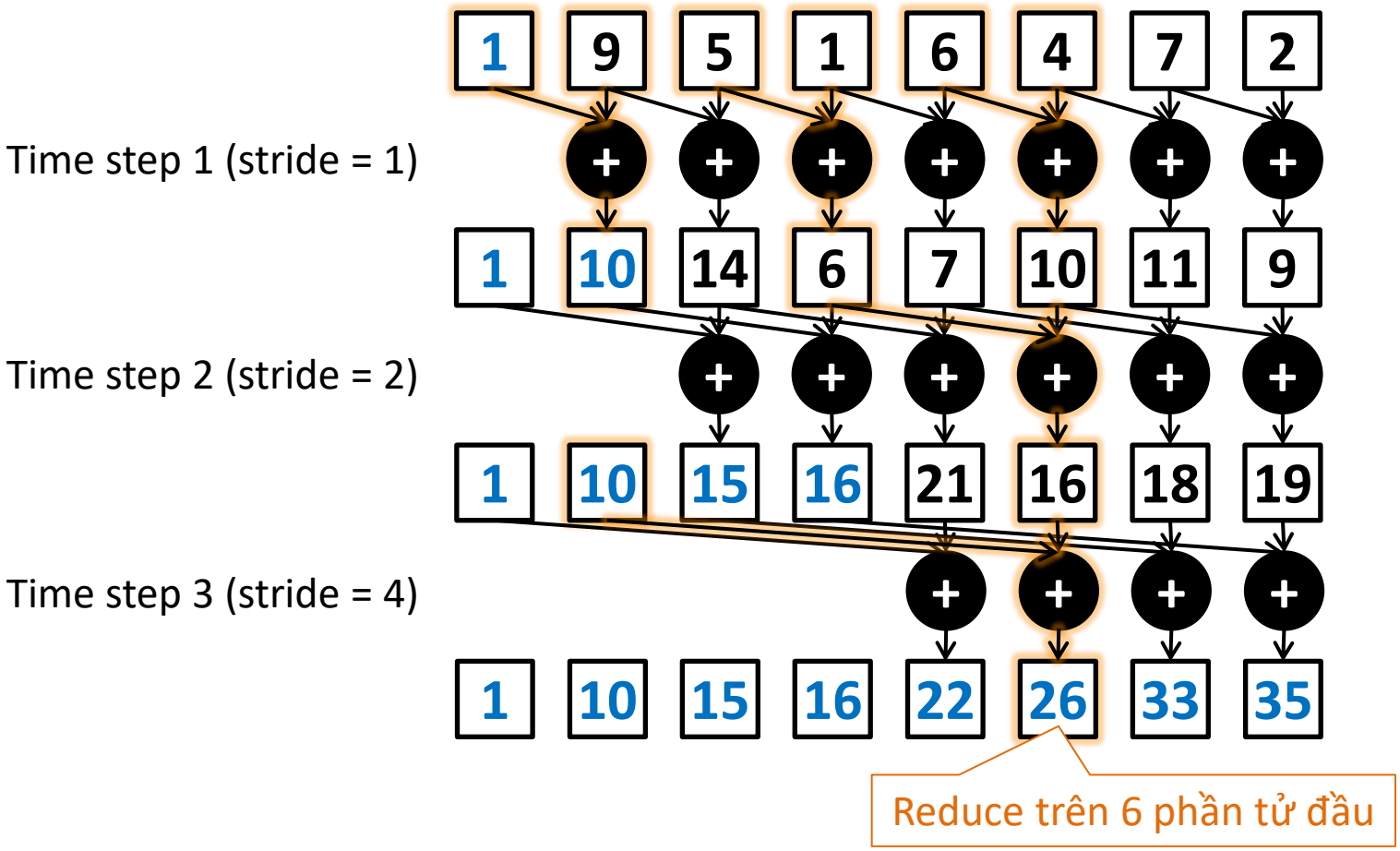
Cài đặt song song



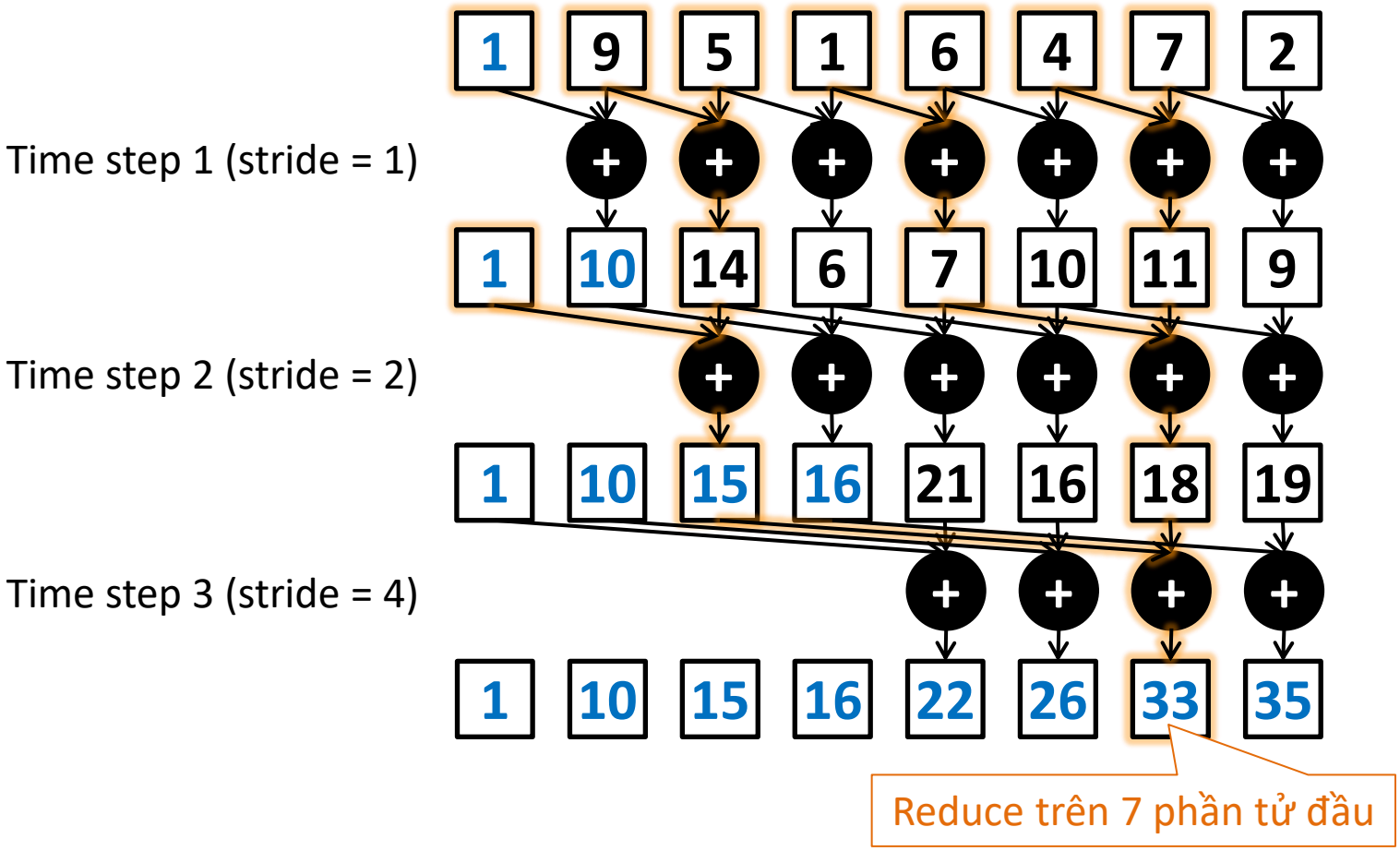
Cài đặt song song



Cài đặt song song



Cài đặt song song

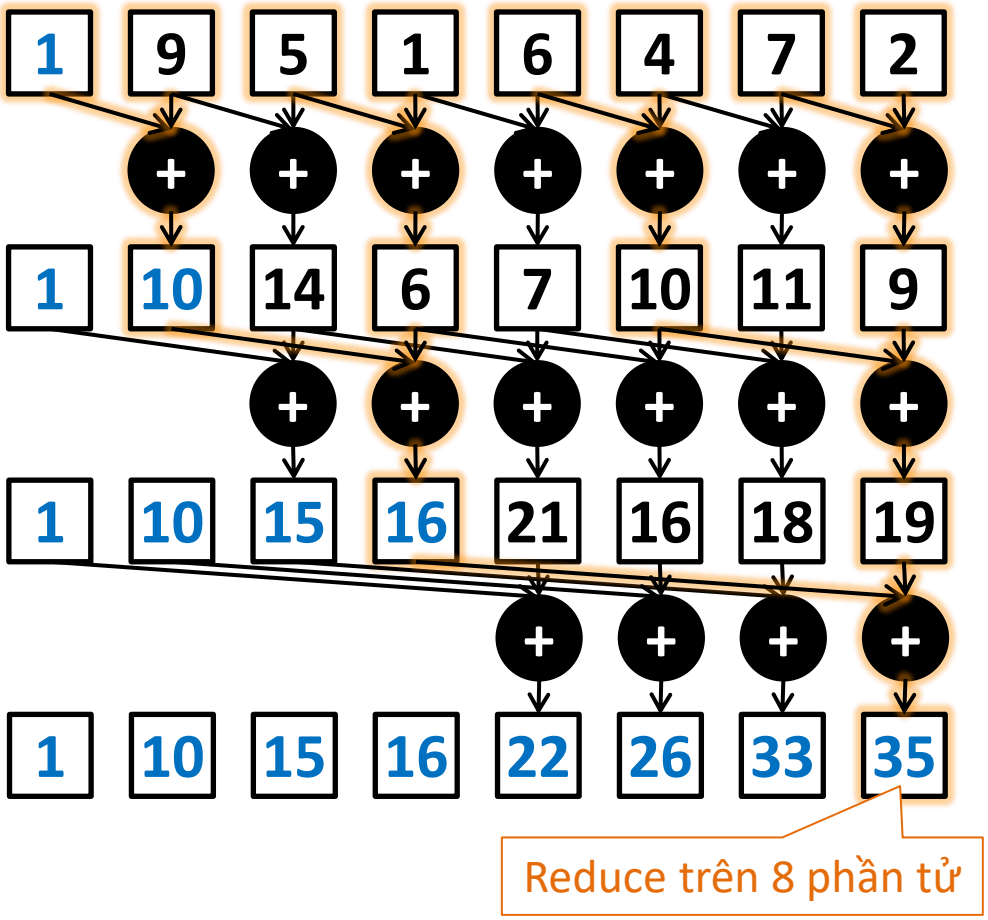


Cài đặt song song

Time step 1 (stride = 1)

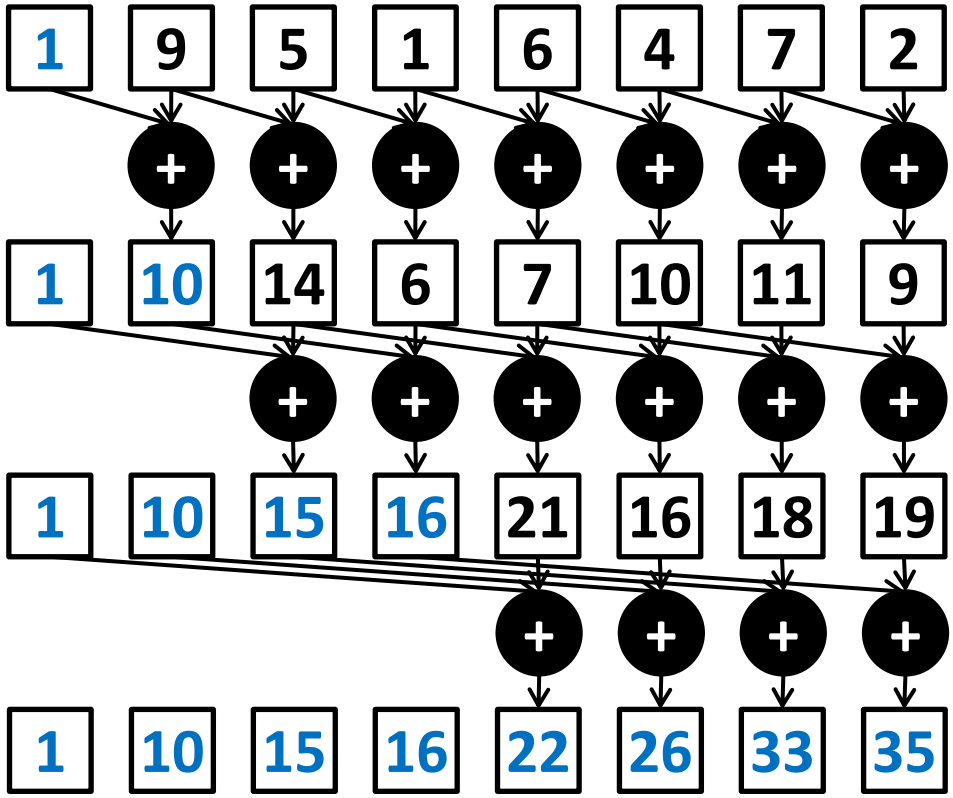
Time step 2 (stride = 2)

Time step 3 (stride = 4)



Điểm hiệu quả ở đây là ta không dùng 8 cây reduce riêng biệt mà 8 cây reduce này dùng chung các kết quả với nhau

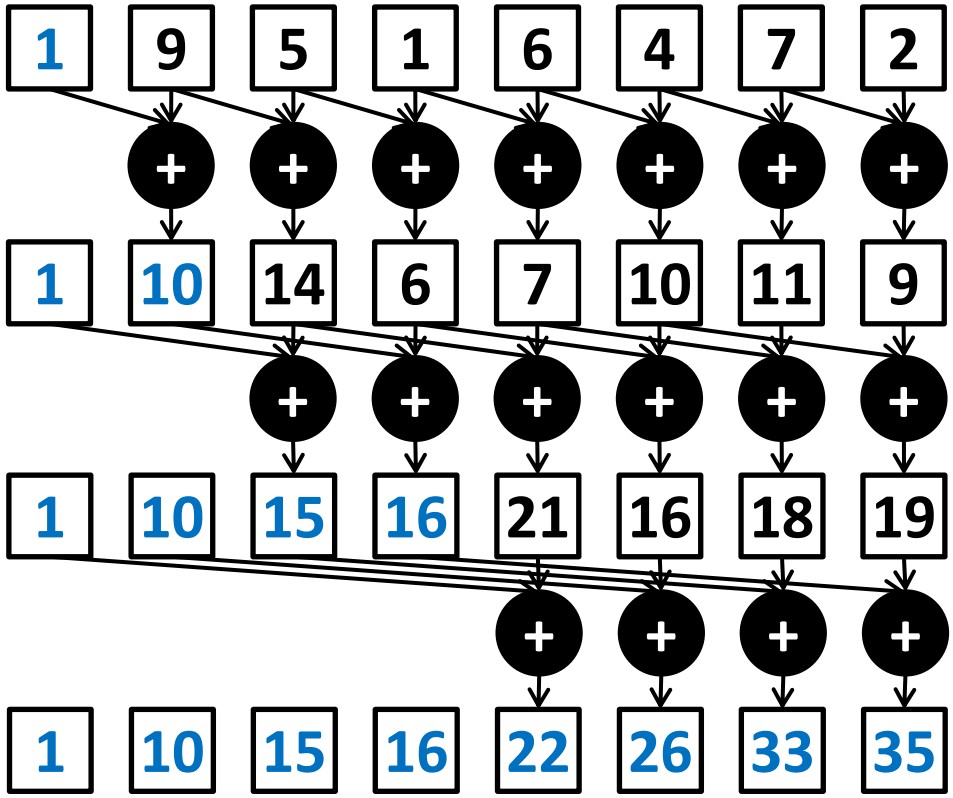
Cài đặt song song



Time:

Work:

Cài đặt song song



Time: 3 = $\log_2 n$ = $O(\log_2 n)$

Work: 17 = (n-1) + (n-2) + (n-4) + …+ (n-n/2)

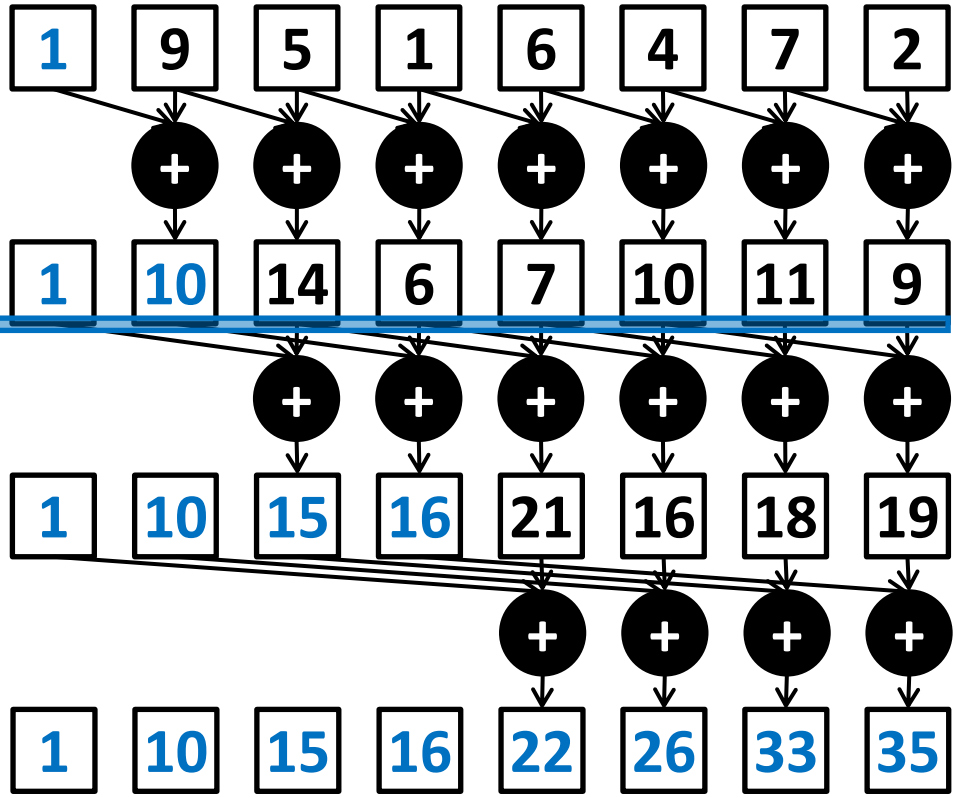
= $n\log_2 n$ - (1 + 2 + … + n/2) = $O(n\log_2 n)$

n - 1

Work-inefficient

Cài đặt song song

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



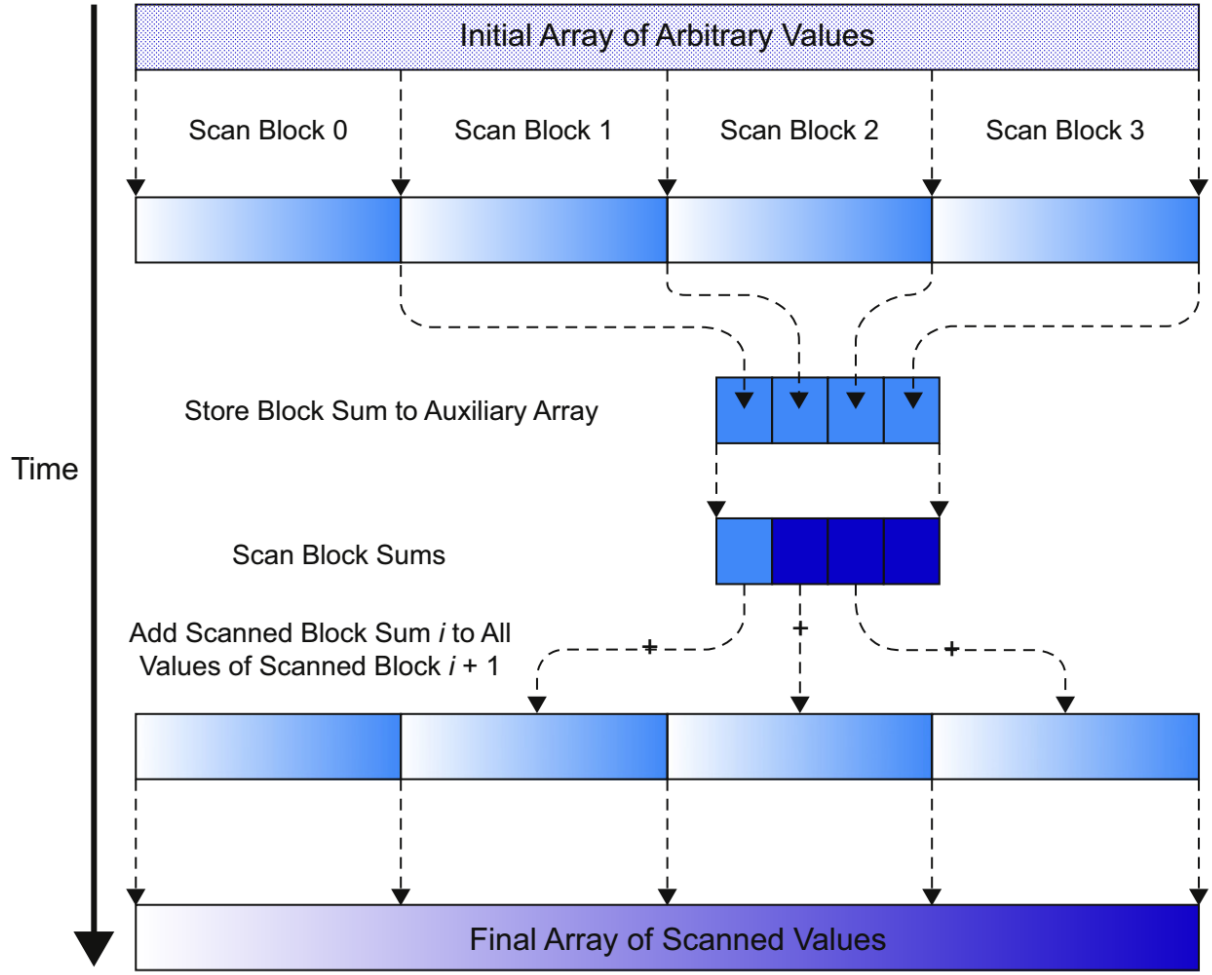
Nhưng trong hàm kernel ta chỉ có thể đồng bộ hóa các thread thuộc cùng một block

Nếu n ≤ kích-thước-khối-dữ-liệu-mà-một-block-có-thể-phụ-trách thì ta có thể dùng hàm kernel với một block để thực hiện

Nếu n > kích-thước-khối-dữ-liệu-mà-một-block-có-thể-phụ-trách thì phải làm sao?

Cài đặt song song

Nếu n > kích-thước-khối-dữ-liệu-mà-một-block-có-thể-phụ-trách thì phải làm sao?

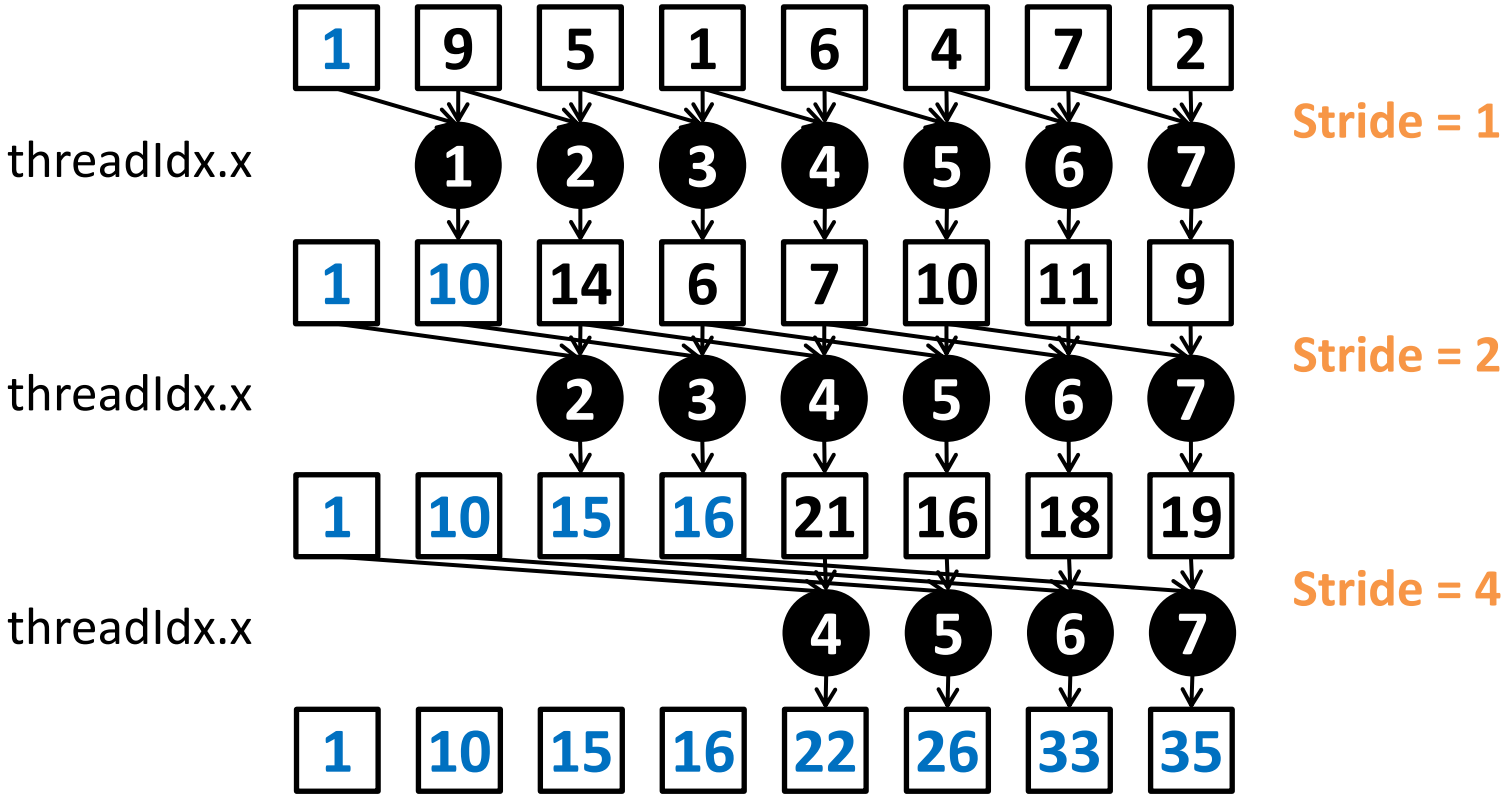


Nguồn ảnh: David B. Kirk et al. Programming Massively Parallel Processors

Scan trong từng block

Xét block gồm 8 thread

1. Block đọc dữ liệu từ GMEM vào SMEM
2. Block thực hiện scan với dữ liệu trên SMEM

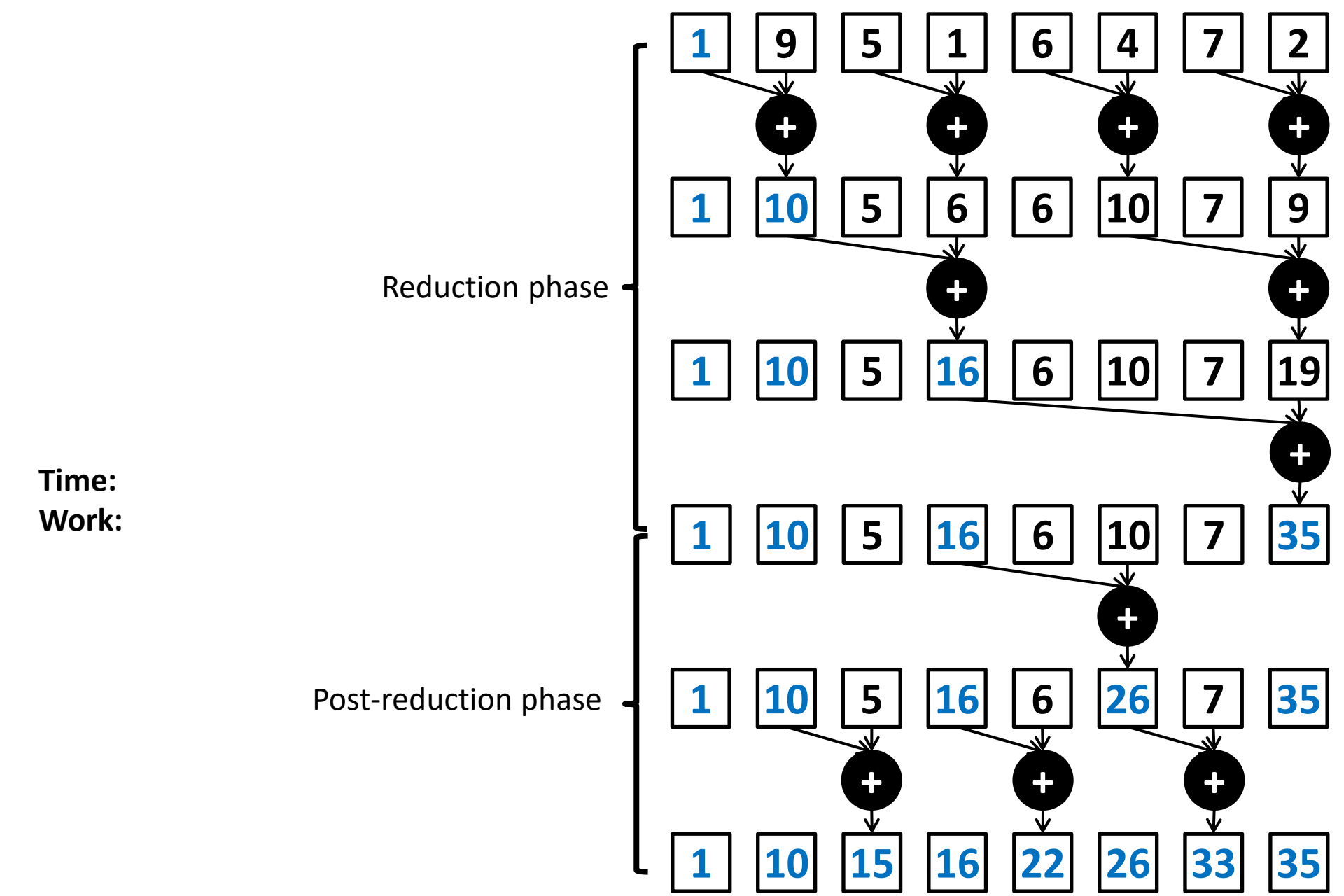


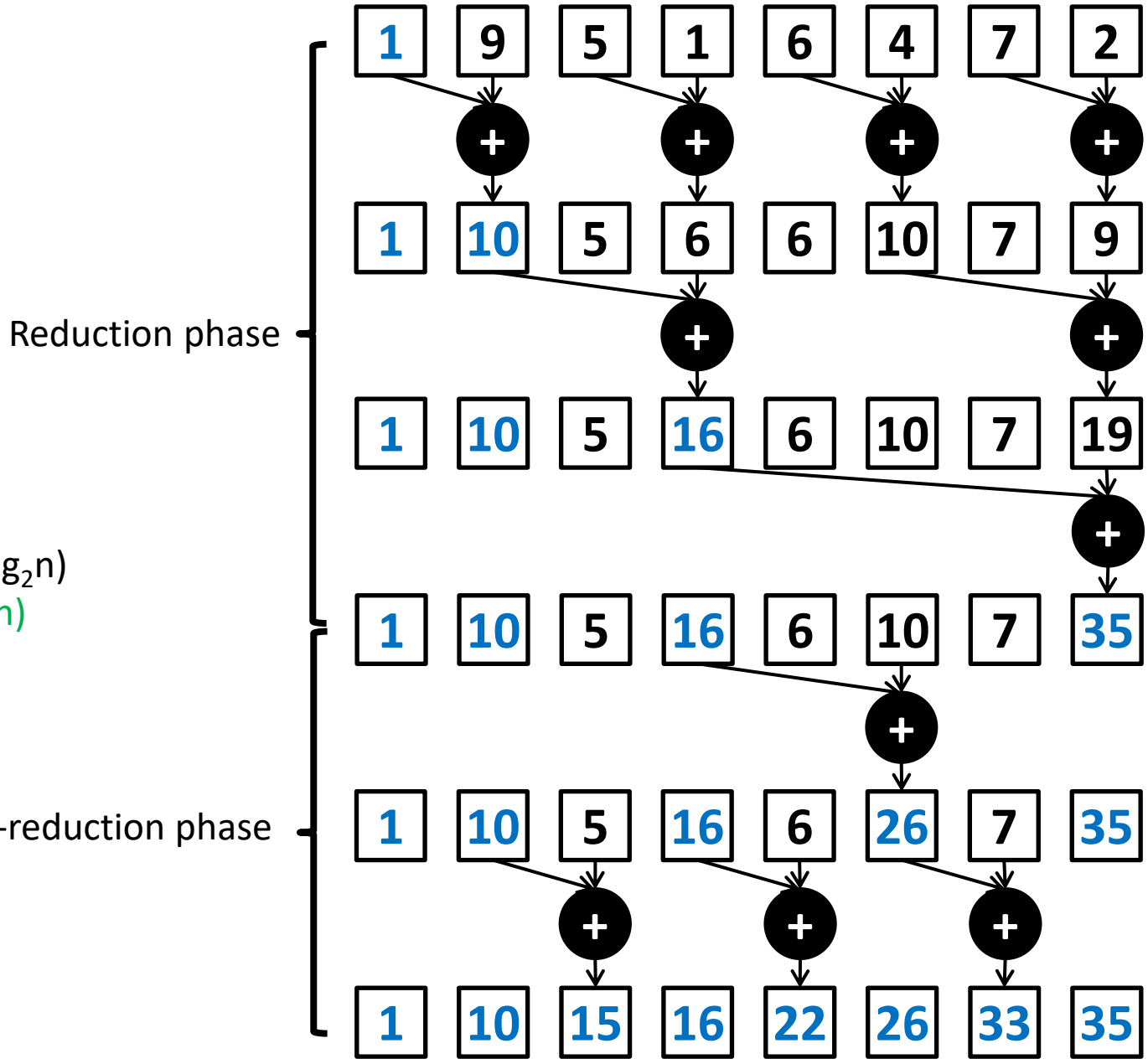
3. Block ghi kết quả từ SMEM xuống GMEM

Bài tập code: file “12-Scan.cu”

Hàm kernel 2 – “work-efficient”

Ý tưởng: giảm số lượng phép cộng xuống bằng cách tận dụng hơn nữa các kết quả đã được tính

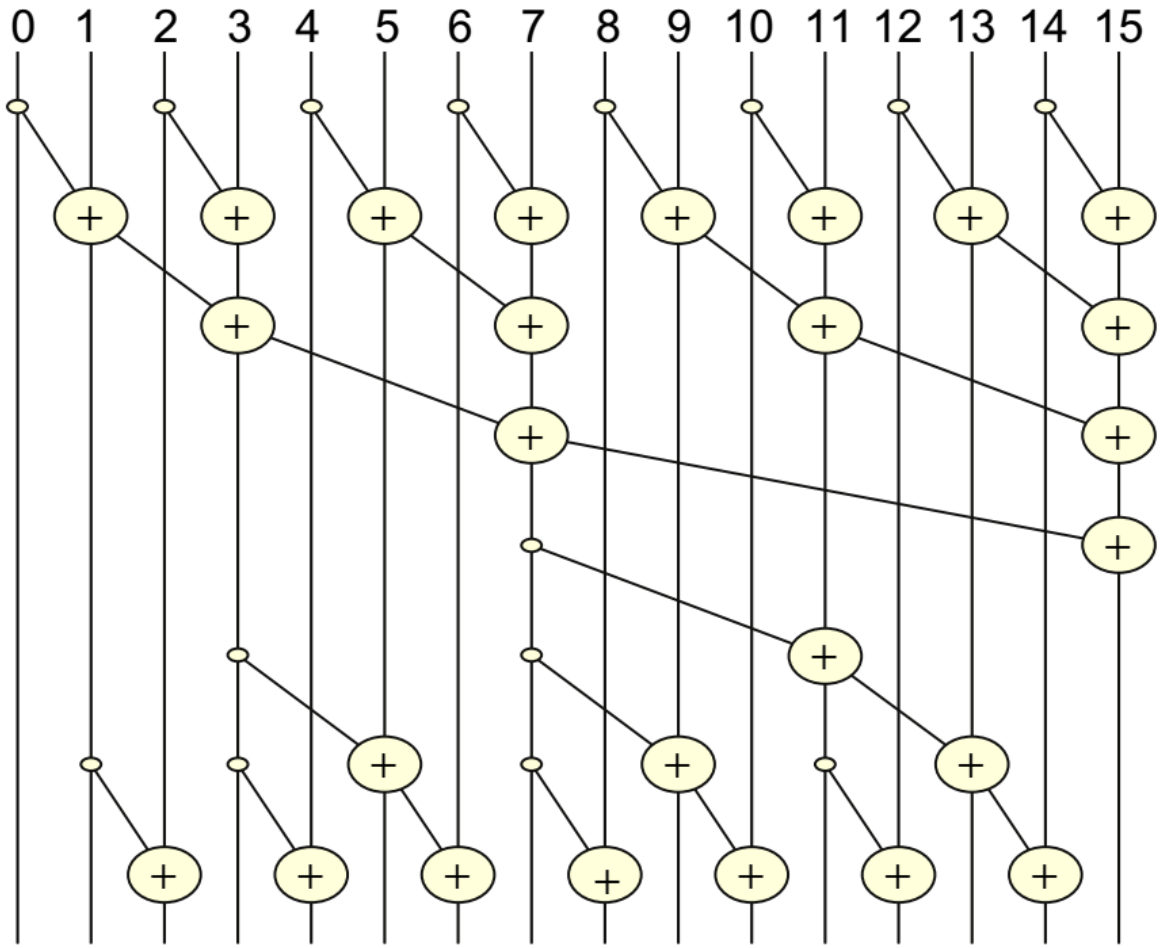

Reduction phase
Post-reduction phase
Time:
Work:
1 9 5 1 6 4 7 2
1 10 5 6 6 10 7 9
1 10 5 16 6 10 7 19
1 10 5 16 6 10 7 35
1 10 5 16 6 26 7 35
1 10 15 16 22 26 33 35



Time: 5 ≈ 2*$\log_2$n = O($\log_2$n)
Work: 11 ≈ 2*(n-1) = O(n)

Work-efficient

Minh họa trường hợp 16 phần tử



Nguồn ảnh: David B. Kirk et al. Programming Massively Parallel Processors

Phân kỳ warp

Giải pháp: cho các thread cạnh nhau hoạt động (tương tự hàm kernel 2 ở bài toán reduction)

Truy xuất GMEM không hiệu quả

Giải pháp: dùng SMEM thì không có vấn đề này