CMPSCI 691AD – General Purpose Computation on the GPU

Spring 2009

Lecture 6: Prefix Sum (Scan) I

Rui Wang

(cont. from last lecture)

- Serial time: T_S
- Parallel time (p processors): T_P
- Overhead: $T_O = pT_P - T_S$
- Speedup:

$$S = \frac{T_S}{T_P}$$

- Efficiency:

$$E = \frac{S}{p} = \frac{T_S}{pT_P} = \frac{1}{1 + T_O/T_S}$$

- Cost Optimal: $pT_P = O(T_S)$

(cont. from last lecture)

- Parallel Reduction:
  - The iterative version:

$$T_P = O\left(\frac{n}{p}\log p\right) \longrightarrow p\,T_P = O(n \log p)$$

  - The serial sum version:

$$T_P = O\left(\frac{n}{p} + \log p\right) \longrightarrow p\,T_P = O(n + p \log p)$$

Scalability

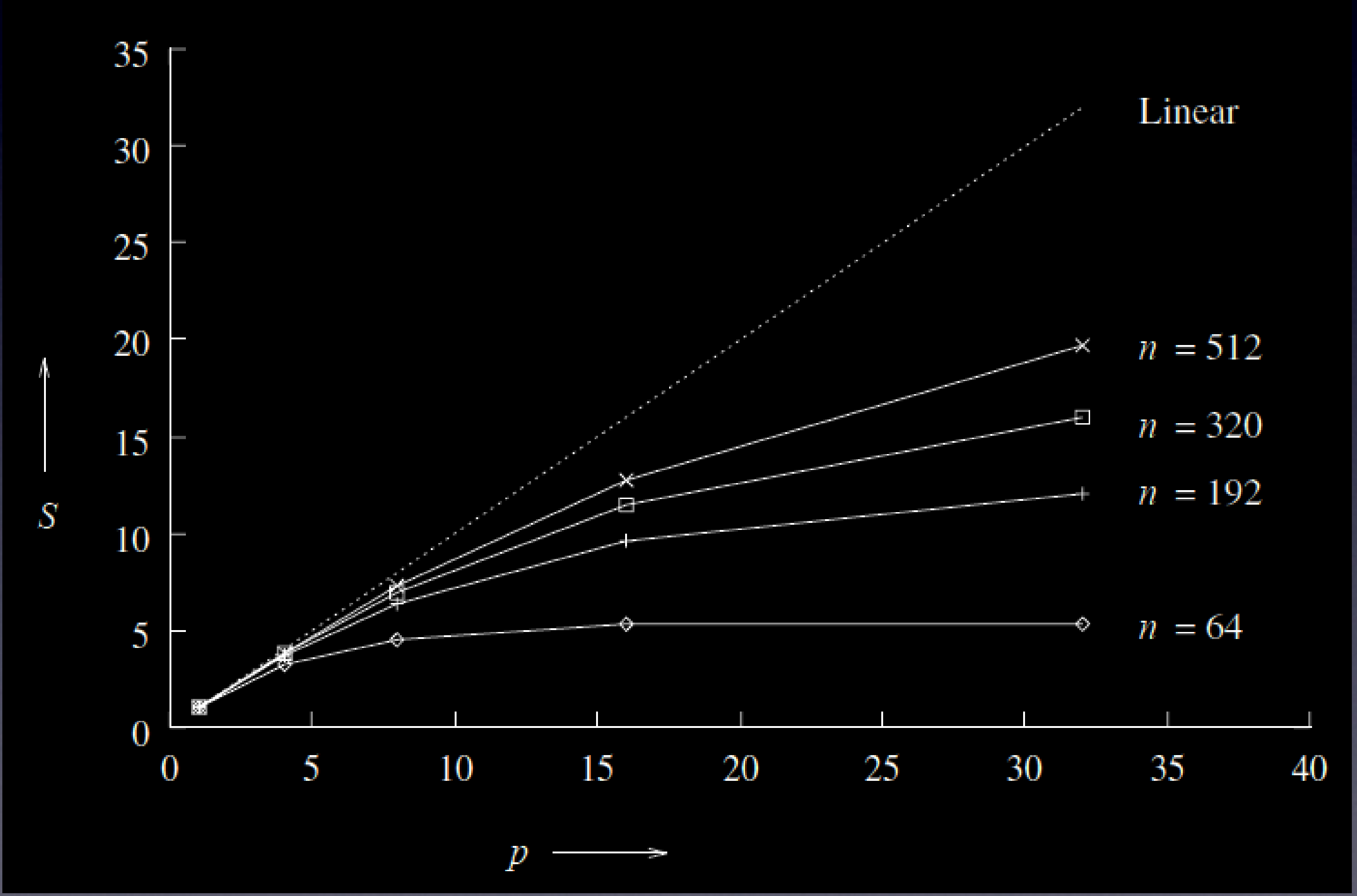
- How does the system scale up and down with different number of input size (work size)?
- Let's pick parallel reduction as an example, and make up the actual cost:

$$T_P = \frac{n}{p} + 2\log p$$

$$S = \frac{n}{\frac{n}{p} + 2\log p} \qquad T_O = 2p\log p$$

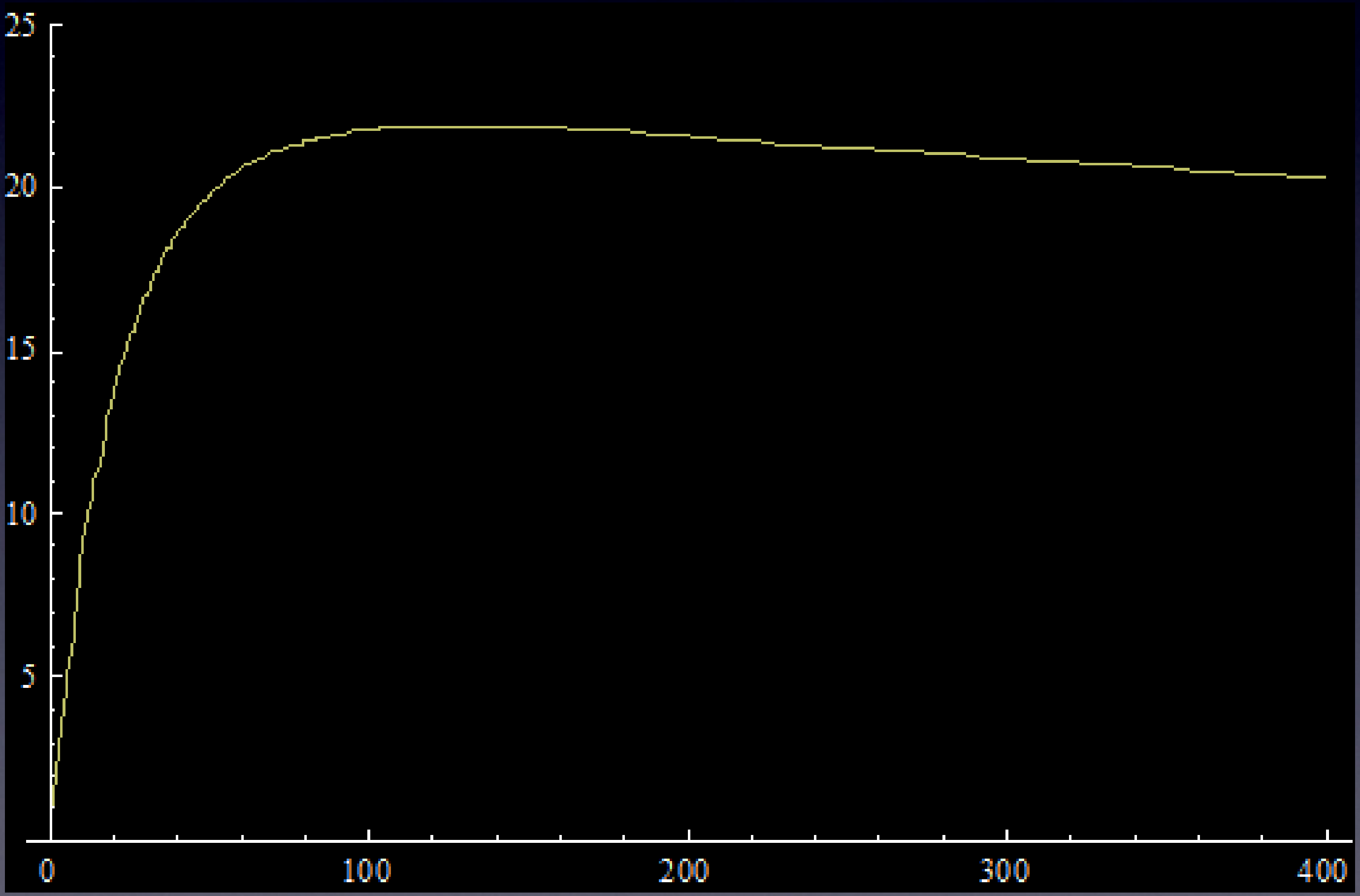
Scalability

- Speedup:



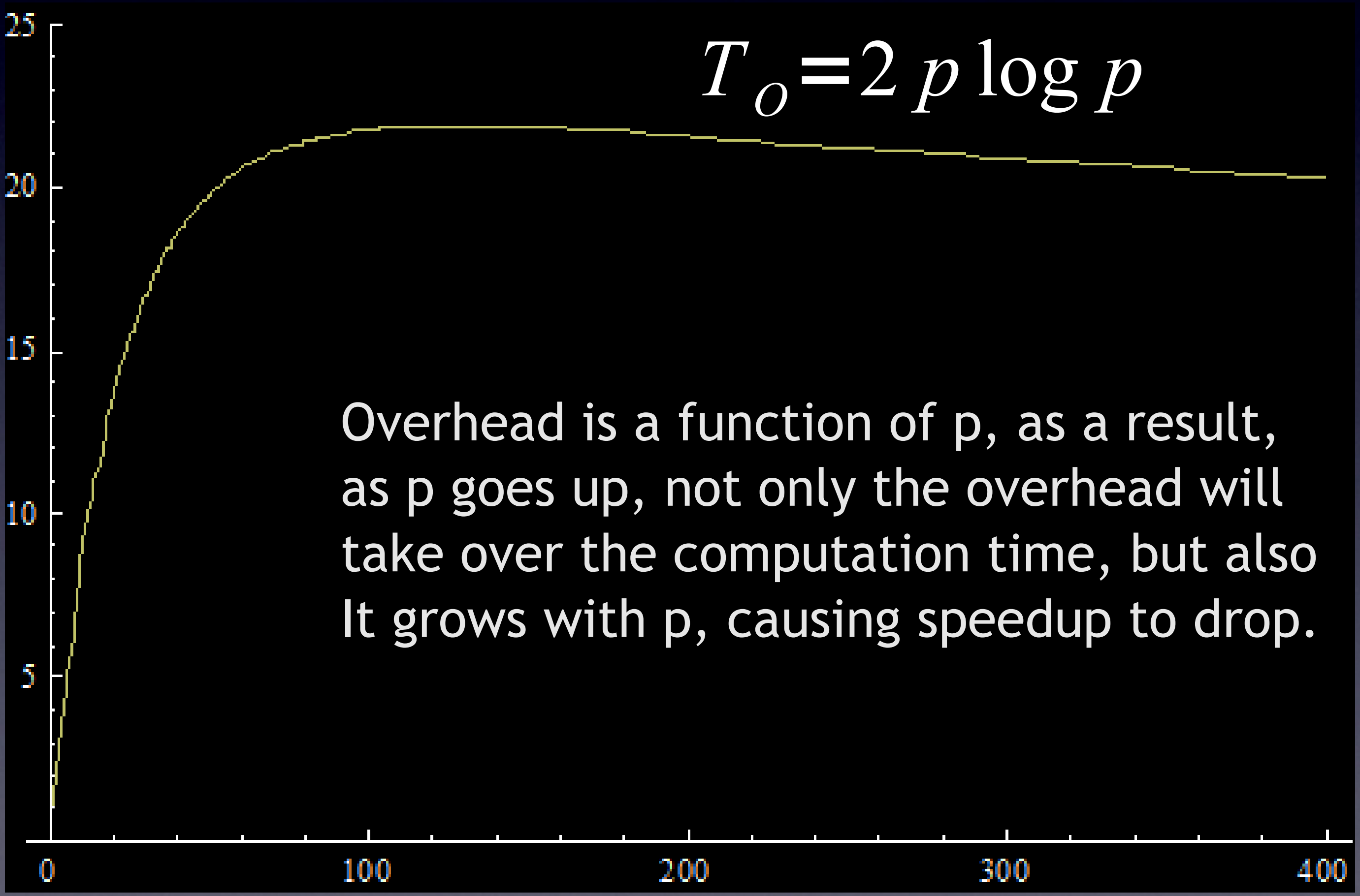
Scalability

- Speedup as a function of p (n = 256)



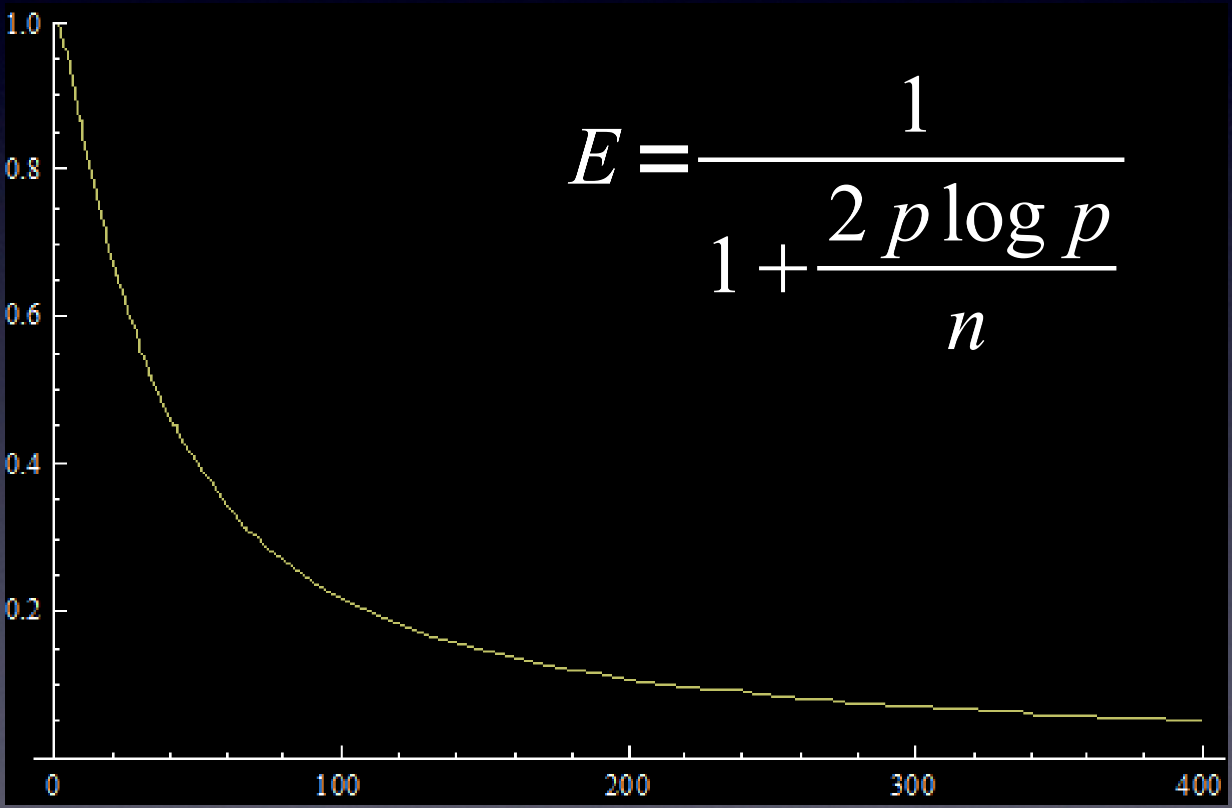
Scalability

- Speedup as a function of p (n = 256)



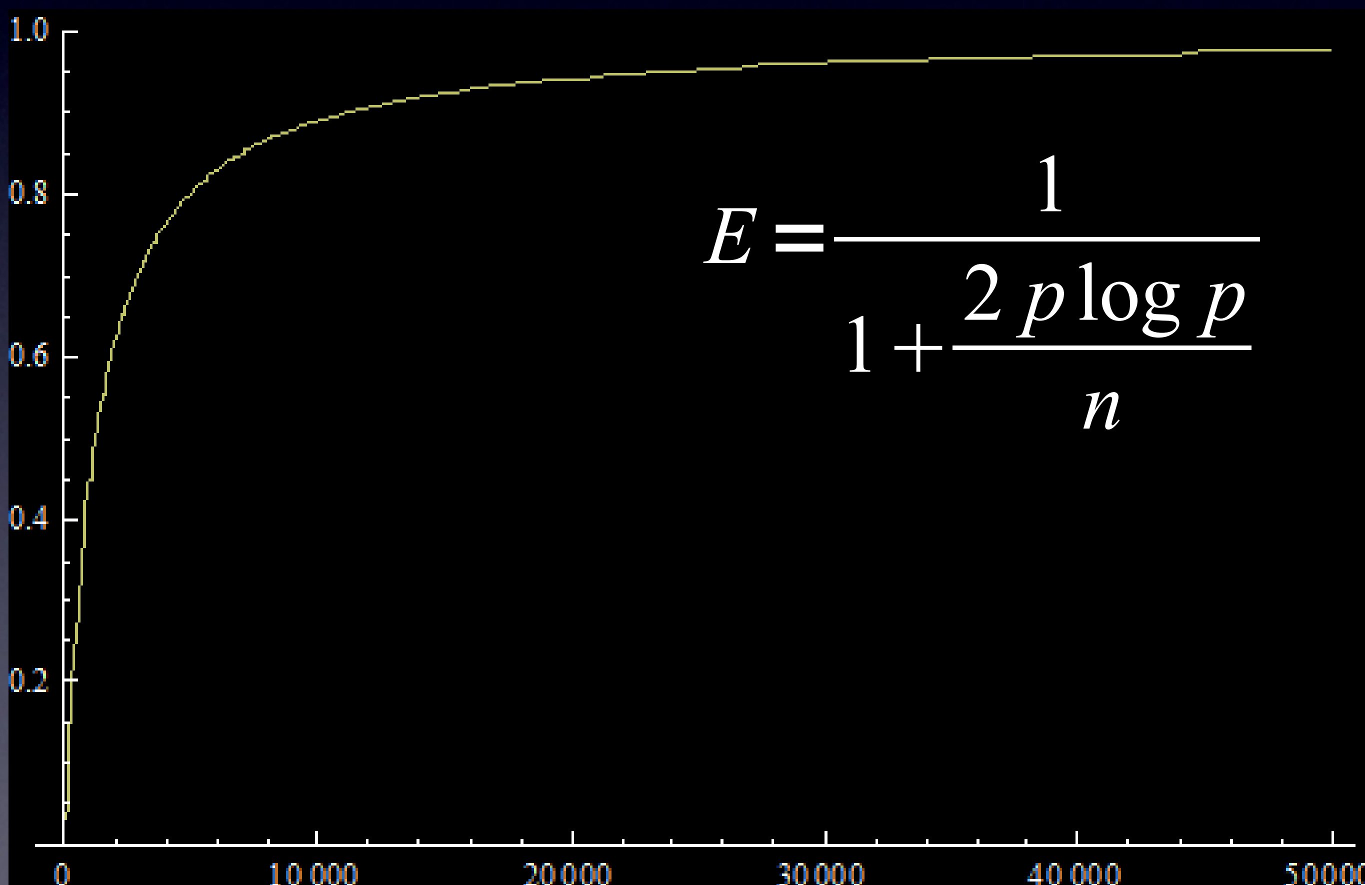
Efficiency

- Efficiency as a function of p (n = 256)



Efficiency

- Efficiency as a function of n (p = 128)



Scalability

- In many cases, overhead grows sublinearly with respect to the work size.
- As a result, we can increase efficiency by simultaneously increase the work size and the number of processors to keep efficiency constant.
- Such systems are called scalable systems.
- Scalability reflects a parallel system's ability to utilize increasing resources effectively.

The Isoefficiency Function

- How much should the work size grow to maintain the same efficiency?

$$E = \frac{1}{1 + \frac{2p\log p}{n}} \longrightarrow n = \frac{E}{1-E}(2p\log p)$$

The Isoefficiency Function

- How much should the work size grow to maintain the same efficiency?

$$E = \frac{1}{1 + \frac{2p\log p}{n}} \longrightarrow n = \frac{E}{1-E}(2p\log p)$$

- Generally:

$$W = \frac{E}{1-E} T_O$$

Isoefficiency function

Problem size should grow proportionally to the overhead function, to maintain the same efficiency.

Minimum Execution Time

- Assume the work size is n, how many processors should I throw in to get the minimum execution time?

$$T_P = \frac{n}{p} + 2 \log p$$

Minimum Execution Time

- Assume the work size is n, how many processors should I throw in to get the minimum execution time?

$$T_P = \frac{n}{p} + 2\log p$$

- Differentiate T_P and solve the minimization problem

Turns out: $p = \frac{n}{2}$ and $T_P^{min} = 2\log n$

Minimum Execution Time

- Assume the work size is n, how many processors should I throw in to get the minimum execution time?

$$T_P = \frac{n}{p} + 2\log p$$

- Differentiate T_P and solve the minimization problem

Turns out: $p = \dfrac{n}{2}$ and $T_P^{min} = 2\log n$

- However, by coupling p and n, the algorithm is no longer cost-optimal!

Minimum Cost-Optimal Execution Time

- In order for the system to remain cost-optimal:

$$f(p) = p \log p = O(n)$$

- Hence, for a problem size of n, we can throw in as much as $f^{-1}(n)$ processors to remain cost-optimal.
- Therefore, the minimum cost-optimal execution time we can achieve is:

$$O\left(\frac{n}{f^{-1}(n)}\right)$$

Minimum Cost-Optimal Execution Time

- Let's take: $p = \dfrac{n}{\log n}$
- Since T_P reduces monotonically up till $p = \dfrac{n}{2}$ we get:

$$T_P^{cost\ optimal\, min} = \frac{n}{p} + 2\log p = 3\log n - \log\log n$$

Prefix Sum (Scan) and Applications

- Problem Definition:
  Given sequence A of n elements and an associative operator $\oplus$ with identity I

  Inclusive scan:
  scan(A, $\oplus$) = $[A_{0,}(A_0 \oplus A_1), \ldots, (A_0 \oplus A_1 \oplus \ldots A_{n-1})]$

  Exclusive scan:
  prescan(A, $\oplus$) =$[I, A_{0,}(A_0 \oplus A_1), \ldots, (A_0 \oplus A_1 \oplus \ldots A_{n-2})]$

- Example: A = [3 1 7 0 4 1 6 3]

Prefix Sum (Scan) and Applications

- Why is this importance?

  Lots of applications: split, radix sort, building trees, subdivision, string comparison, line of sight, histogram, run-length encoding......

- Naïve serial algorithm: linear sum in O(n) time
- Important to study data-parallel version of the algorithm!

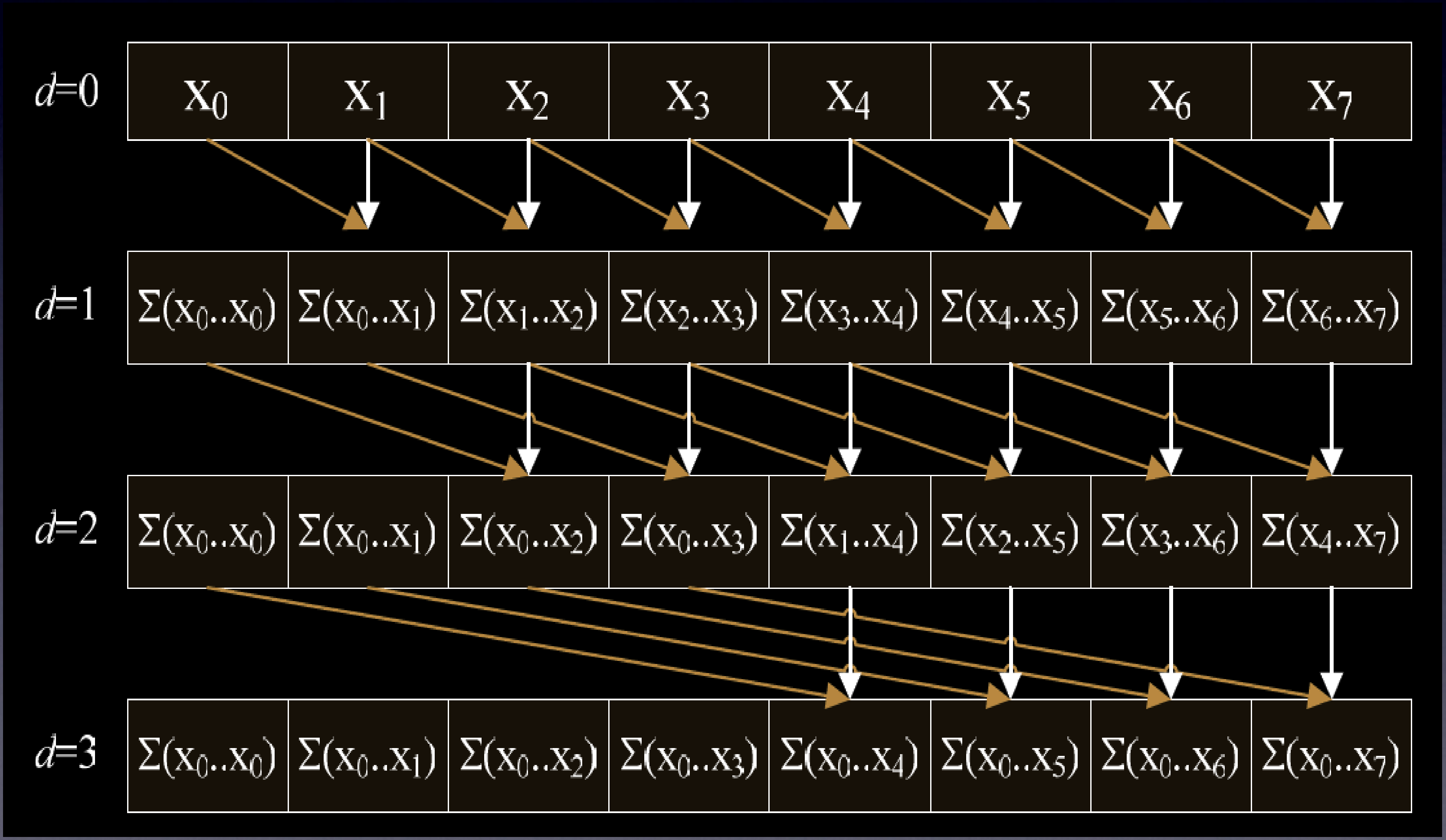
Prefix Sum (Scan) and Applications

- Reading List:
  - *“Parallel Prefix Sum (Scan)” whitepaper in CUDA SDK*
  - *“Prefix Sums and Their Applications”:*

    http://sbel.wisc.edu/Courses/ME964/2008/Documents/CMU-CS-90-190.pdf

Prefix Sum (Scan) and Applications

- First try:



Prefix Sum (Scan) and Applications

- First try:

```
for d := 1 to log2n do
    forall k in parallel do
        if k ≥ 2^d then
            x[out][k] := x[in][k – 2^(d-1)] + x[in][k]
        else
            x[out][k] := x[in][k]
    swap(in, out)
```

- What's the number of operations?

Prefix Sum (Scan) and Applications

- First try:

```
for d := 1 to log₂n do
    forall k in parallel do
        if k ≥ 2^d then
            x[out][k] := x[in][k – 2^(d-1)] + x[in][k]
        else
            x[out][k] := x[in][k]
    swap(in, out)
```

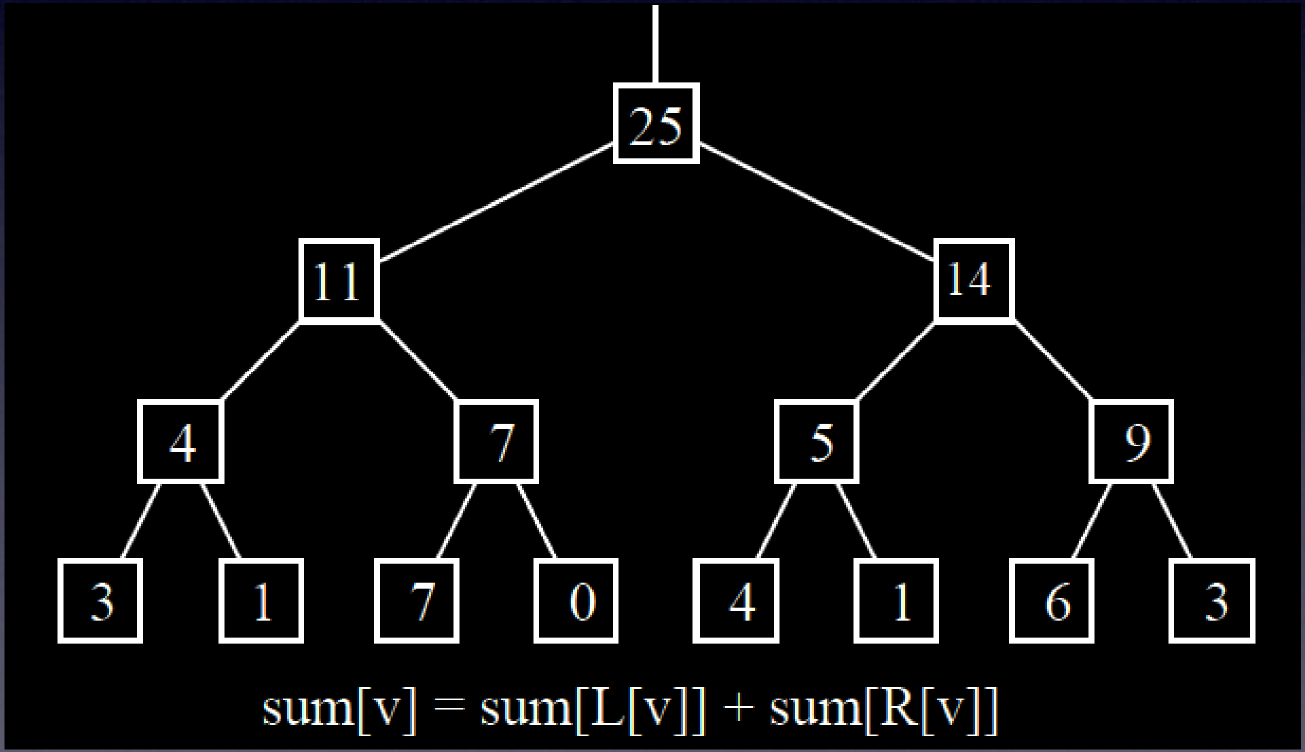
- What's the number of operations?

$$\sum_{d=1}^{\log n} (n - 2^{d-1}) = O(n \log n)$$

More than serial !

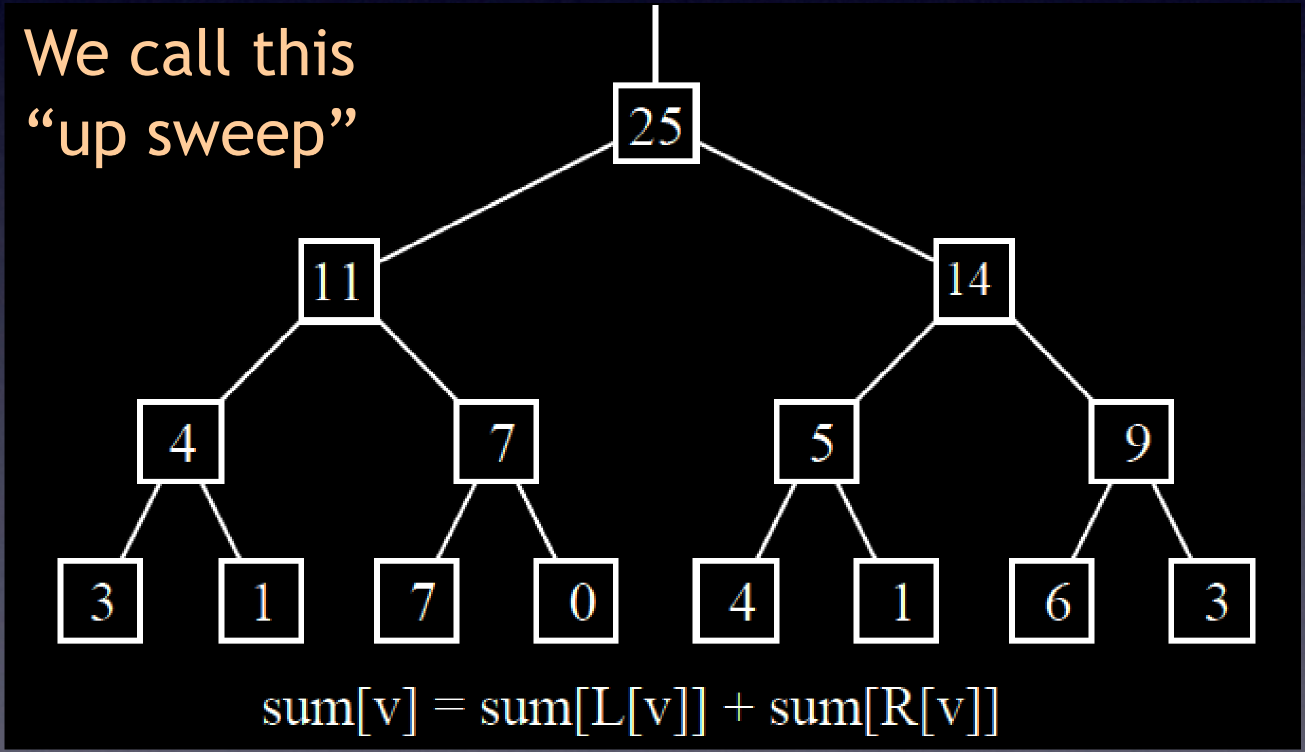
Prefix Sum (Scan) and Applications

- Second try: think about how to utilize parallel reduction results:



Prefix Sum (Scan) and Applications

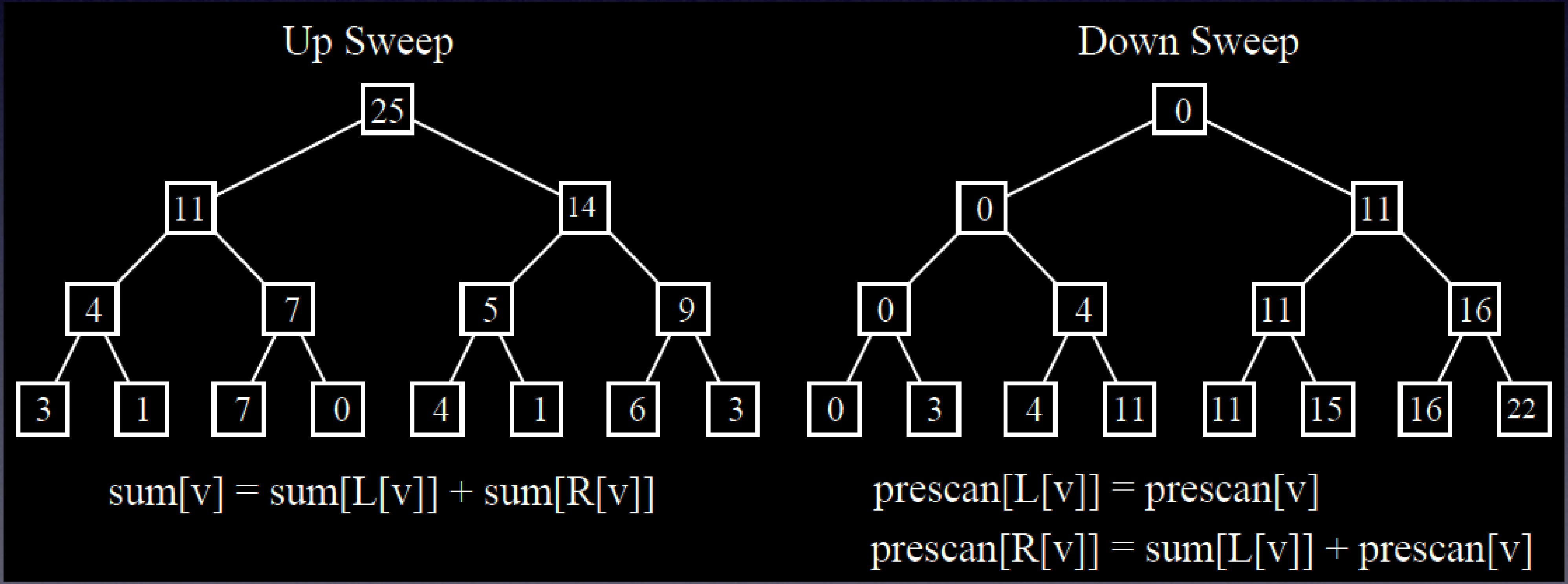
- Second try: think about how to utilize parallel reduction results:



Prefix Sum (Scan) and Applications

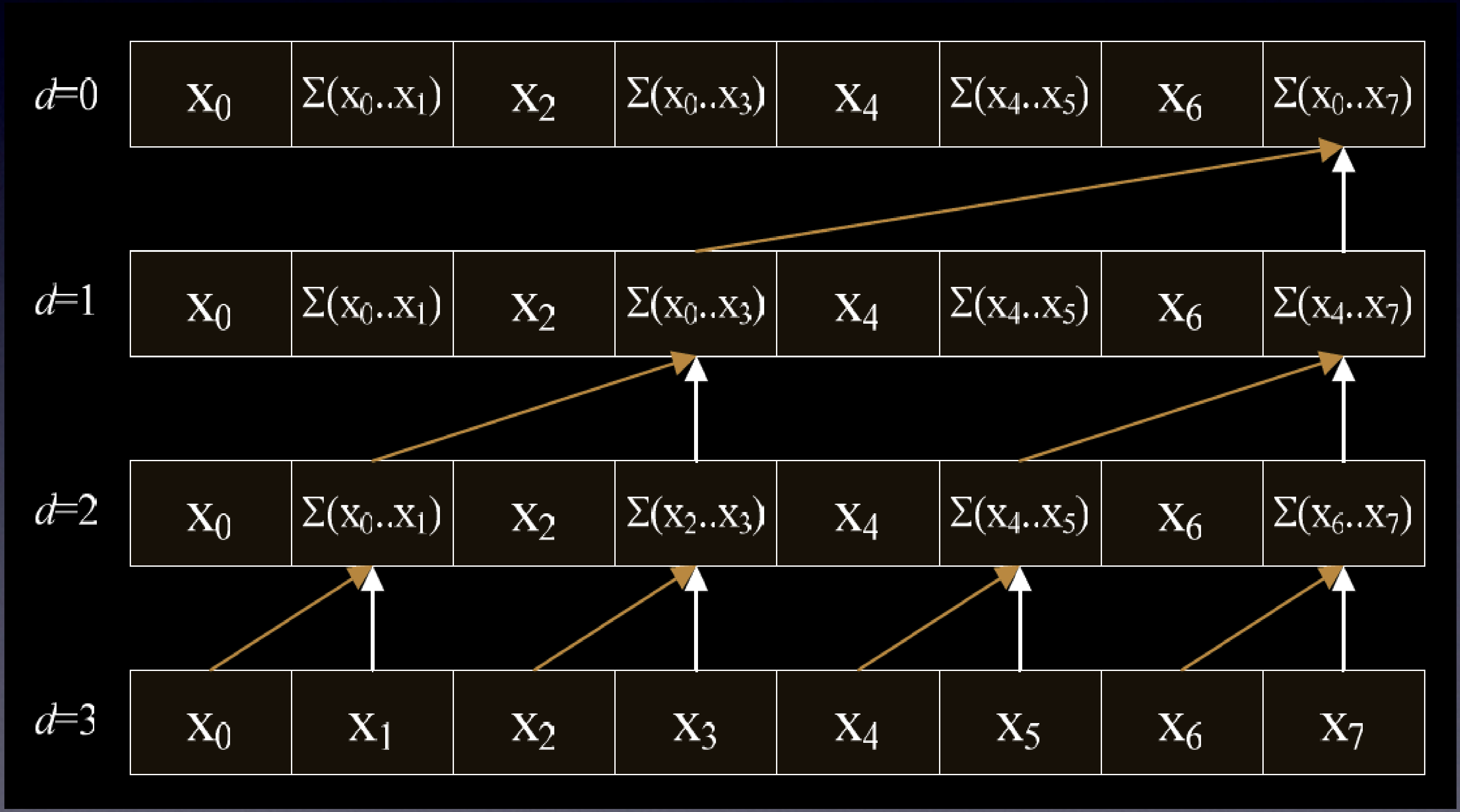
- Second try: think about how to utilize parallel reduction results:

Now let's do "down sweep"!



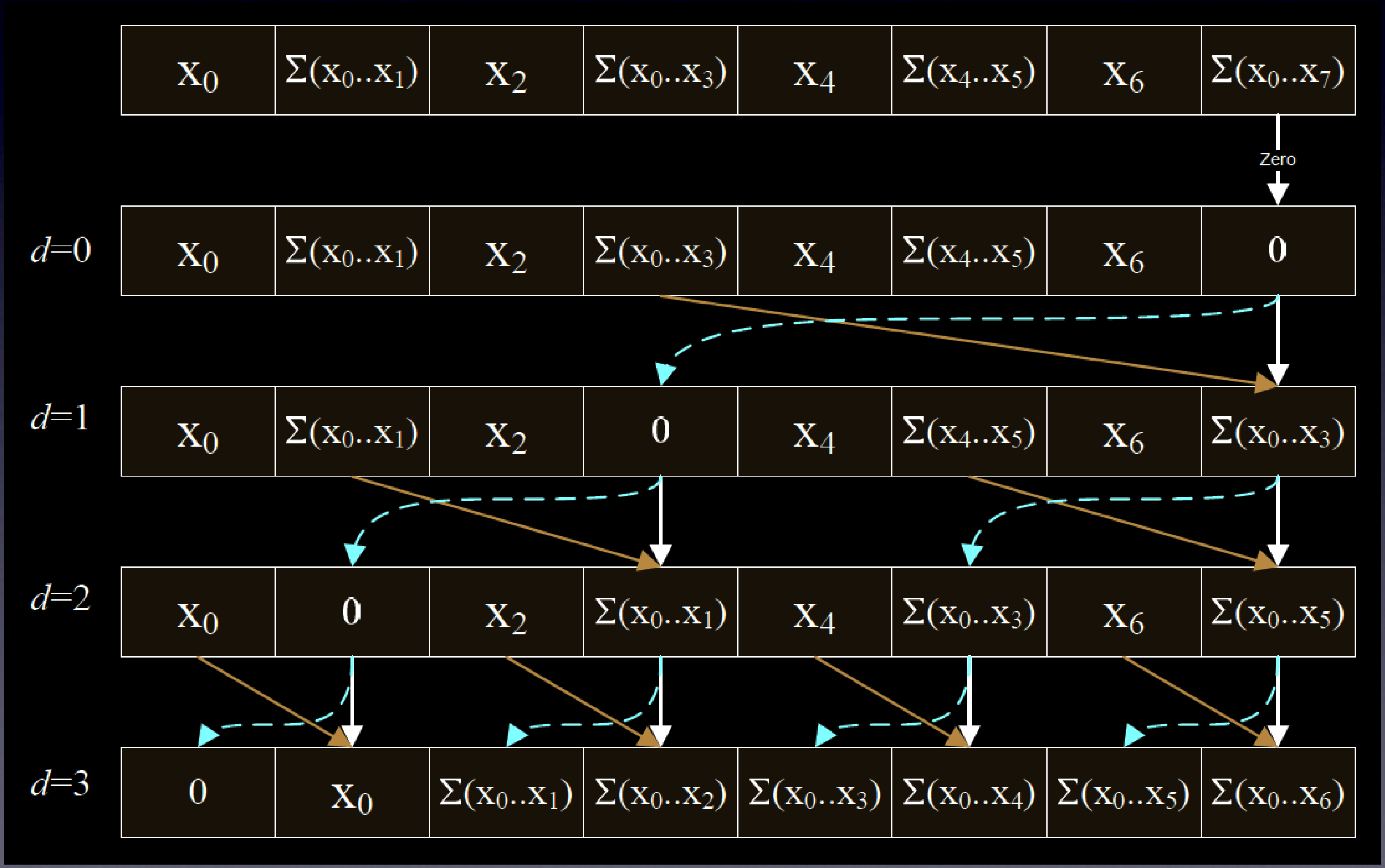
Prefix Sum (Scan) and Applications

- Up-sweep



Prefix Sum (Scan) and Applications

- Down-sweep:



Prefix Sum (Scan) and Applications

- Down sweep algorithm:

$x[n - 1] := 0$
for $d := \log_2 n$ **down to** 0 **do**
    for k **from** 0 **to** $n - 1$ **by** 2^{d+1} **in parallel do**
        $t := x[k + 2^d - 1]$
        $x[k + 2^d - 1] := x[k + 2^{d+1} - 1]$
        $x[k + 2^{d+1} - 1] := t + x[k + 2^{d+1} - 1]$

- Number of operations?

Prefix Sum (Scan) and Applications

- Down sweep algorithm:

```
x[n - 1] := 0
for d := log2n down to 0 do
    for k from 0 to n – 1 by 2^(d+1) in parallel do
        t := x[k + 2^d - 1]
        x[k + 2^d - 1] := x [k + 2^(d+1) - 1]
        x[k + 2^(d+1) - 1] := t + x [k + 2^(d+1) - 1]
```

- Number of operations?

$$\sum_{d=1}^{\log n} (2^{d-1}) = O(n)$$

Same with serial!

Prefix Sum (Scan) and Applications

- Advantages:

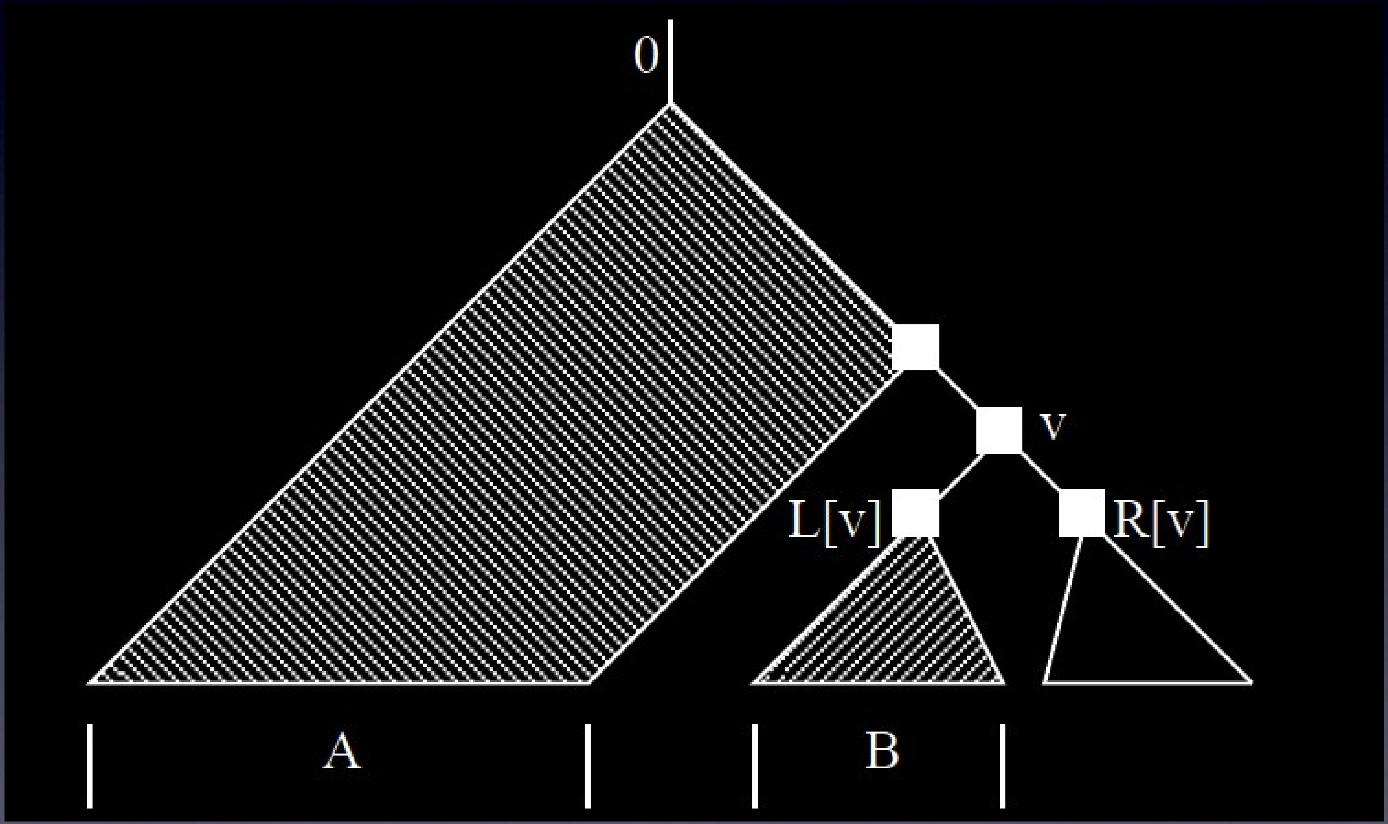
  Although involves the same number of passes as the first try version, this version does involve less memory reads, and compute “in place”, which eliminates the need for double buffers.

- For Large Arrays:

  Similar to parallel reduction: split the input into p partitions → each processor sequentially scans each partition → then a single thread block processes the p partial sums (up-down sweep algorithm) → finally distributes the results back to compute the final scan.

Prefix Sum (Scan) and Applications

- Proof of correctness:



- *"Prefix Sums and Their Applications":*

http://sbel.wisc.edu/Courses/ME964/2008/Documents/CMU-CS-90-190.pdf