An introductory course on General Purpose Computing on GPUs

Part II

Francesco Lettich

Insight Lab
Universidade Federal do Ceará, Fortaleza, Brasil

Introduction

- Today we are going to introduce an extensively used operation in GPU-based applications.

 This operation is called **scan.**

- **Exercise:** I will ask you to implement a few versions of this operation on GPU.

- The goal is to make you **aware** of some of the **challenges** behind coding GPU algorithms.

Scan

- Definition:

  The all-prefix-sums operation takes a binary associative operator $\oplus$ with identity I, and an array of n elements

  $$[a_0, a_1, \ldots, a_{n-1}]$$

  and returns the ordered set

  $$[I, a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-2})].$$

- Example:
  if $\oplus$ is addition, then scan on the set

  [3 1 7 0 4 1 6 3]

  returns the set

  [0 3 4 11 11 15 16 22]

Exclusive scan: last input element is not included in the result

Scan / 2

- Scan is a simple and useful parallel building block
  - Convert recurrences from sequential …

```
for(j=1;j<n;j++)
    out[j] = out[j-1] + f(j);
```

  - … into parallel:

```
forall(j) in parallel
      temp[j] = f(j);
scan(out, temp);
```

- Useful in implementation of several parallel algorithms:

  - radix sort
  - quicksort
  - String comparison
  - Lexical analysis
  - Stream compaction

  - Polynomial evaluation
  - Solving recurrences
  - Tree operations
  - Histograms
  - Etc.

Exclusive Prefix Sum / 1

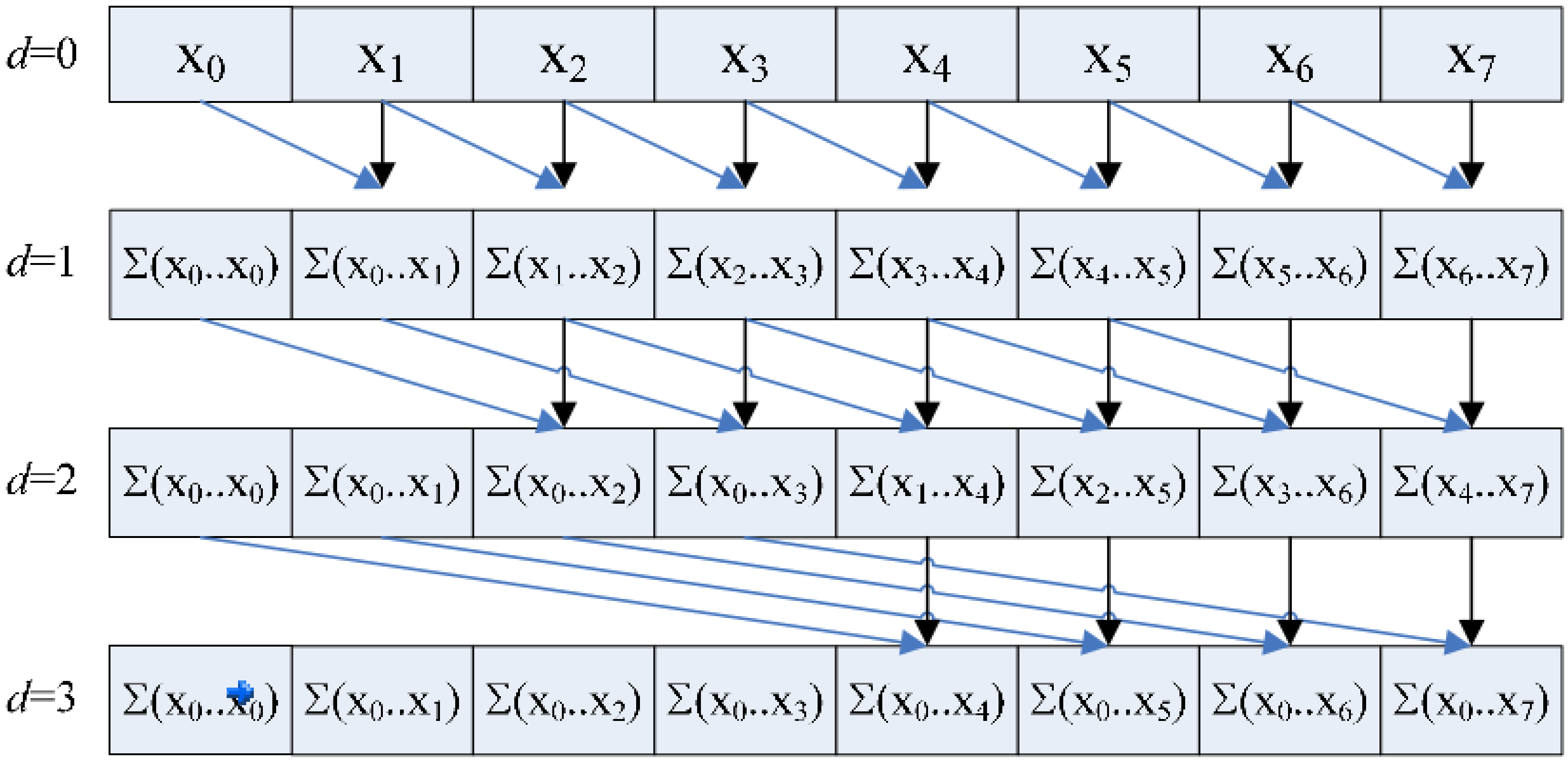
- Implementing the sequential version of this operation is trivial on CPU:

```
void scan(float* scanned, float* input, int length)
{
   scanned[0] = 0;
   for(int i = 1; i < length; ++i)
      scanned[i] = scanned[i-1] + input[i-1];
 }
```

- Complexity is **O(n-1) => O(n)**.

Exclusive Prefix Sum / 2

Hillis and Steele algorithm



- Assume that the number of elements is a power of 2: $n=2^M$.
- Then, at the ***d*-th iteration** the **last "2^M-2^d" elements** have to **find** their **mate**, at a distance equal to 2^d, and **make** the **sum**.

Exclusive Prefix Sum / 3

- How **many iterations** do we have to perform?
- How **many operations** do we perform at each iteration?
- What is the **overall amount** of operations?

Exclusive Prefix Sum / 4

- How **many iterations** do we have to perform?

$\log_2(n) = M$ (3 in the example).

- How **many operations** do we have to perform during each iteration?

$2^M - 2^d$

Exclusive Prefix Sum / 5

- What is the **overall amount** of operations?

Let's make some calculations...

$$M * 2^M - (2^0 + 2^1 + 2^2 + \ldots + 2^{M-1}) =$$

$$M * 2^M - (2^M - 1) =$$

$$\log_2(n) * n - \log_2(n) + 1$$

- Hence, the complexity is in $O(n * \log_2(n))$. **Expensive** when "**n**" **gets large** (example: 1 million of elements); **not good** if compared to the work needed by the **sequential solution O(n)**!!!

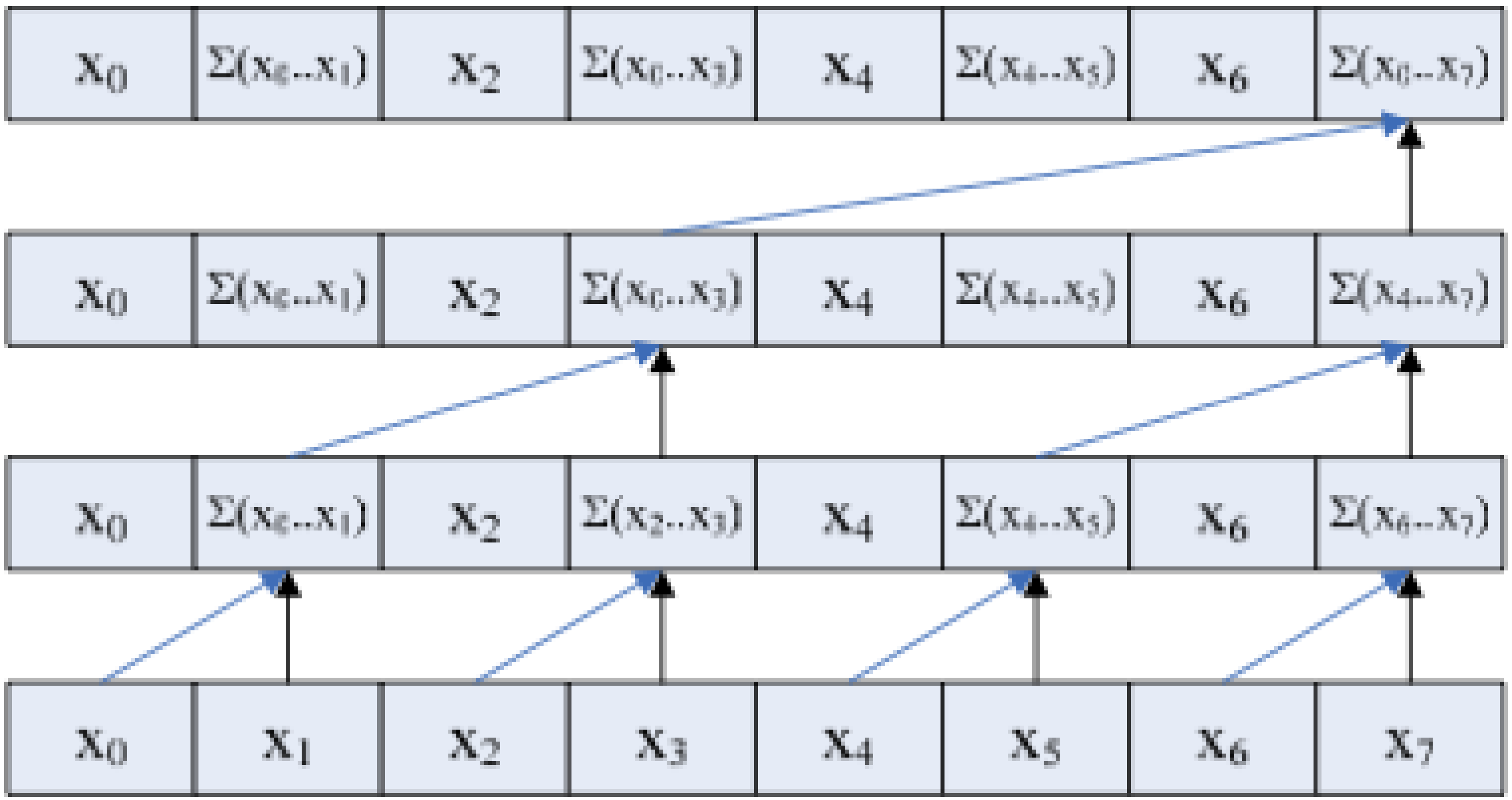
Exclusive Prefix Sum / 6

```
__global__ void scan(float *g_odata, float *g_idata, int n)
{
    extern  __shared__  float temp[]; // allocated on invocation

    int thid = threadIdx.x;
    int pout = 0, pin = 1;

    // load input into shared memory.
    // Exclusive scan: shift right by one and set first element to 0
    temp[thid] = (thid > 0) ? g_idata[thid-1] : 0;
    __syncthreads();

    for( int offset = 1; offset < n; offset <<= 1 )
    {
        pout = 1 - pout; // swap double buffer indices
        pin  = 1 - pout;

        if (thid >= offset)
            temp[pout*n+thid] += temp[pin*n+thid - offset];
          else
            temp[pout*n+thid] = temp[pin*n+thid];

        __syncthreads();
    }

    g_odata[thid] = temp[pout*n+thid1]; // write output
}
```

Blelloch's Scan / 1

- We use the concept of **balanced trees** to implement a cleverer scan algorithm.

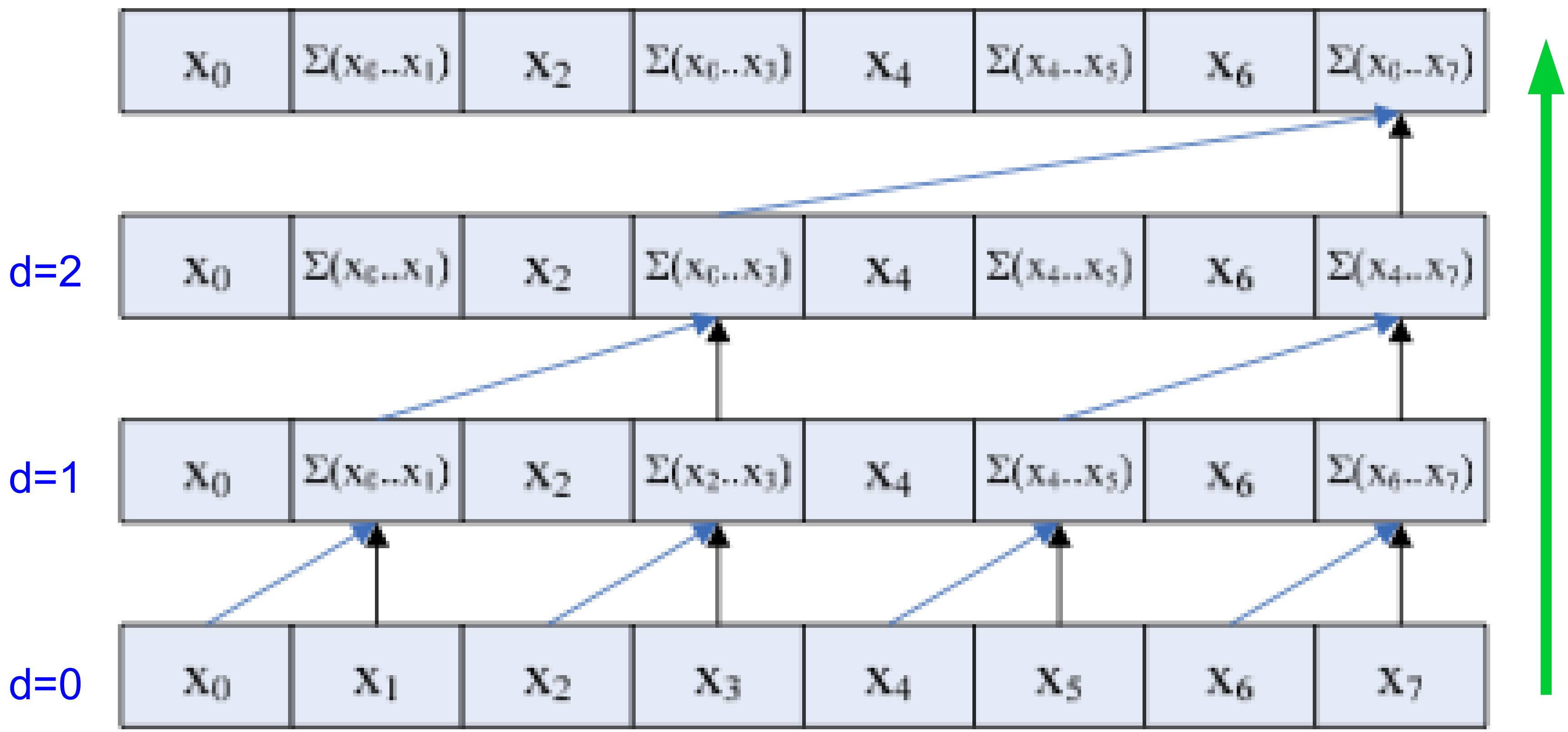
- **Idea**: suppose that the number of elements is $\mathbf{2^M}$, and that the elements represent the leaves of a binary, balanced tree.



Blelloch's Scan / 2

- **<u>First</u>**, "sweep" **from** the **leaves** to the **root** of the tree: *up-sweep*.

- The idea is to **accumulate** and **propagate towards** the **root** the partial **sums** of **increasingly** bigger partitions in the vector.

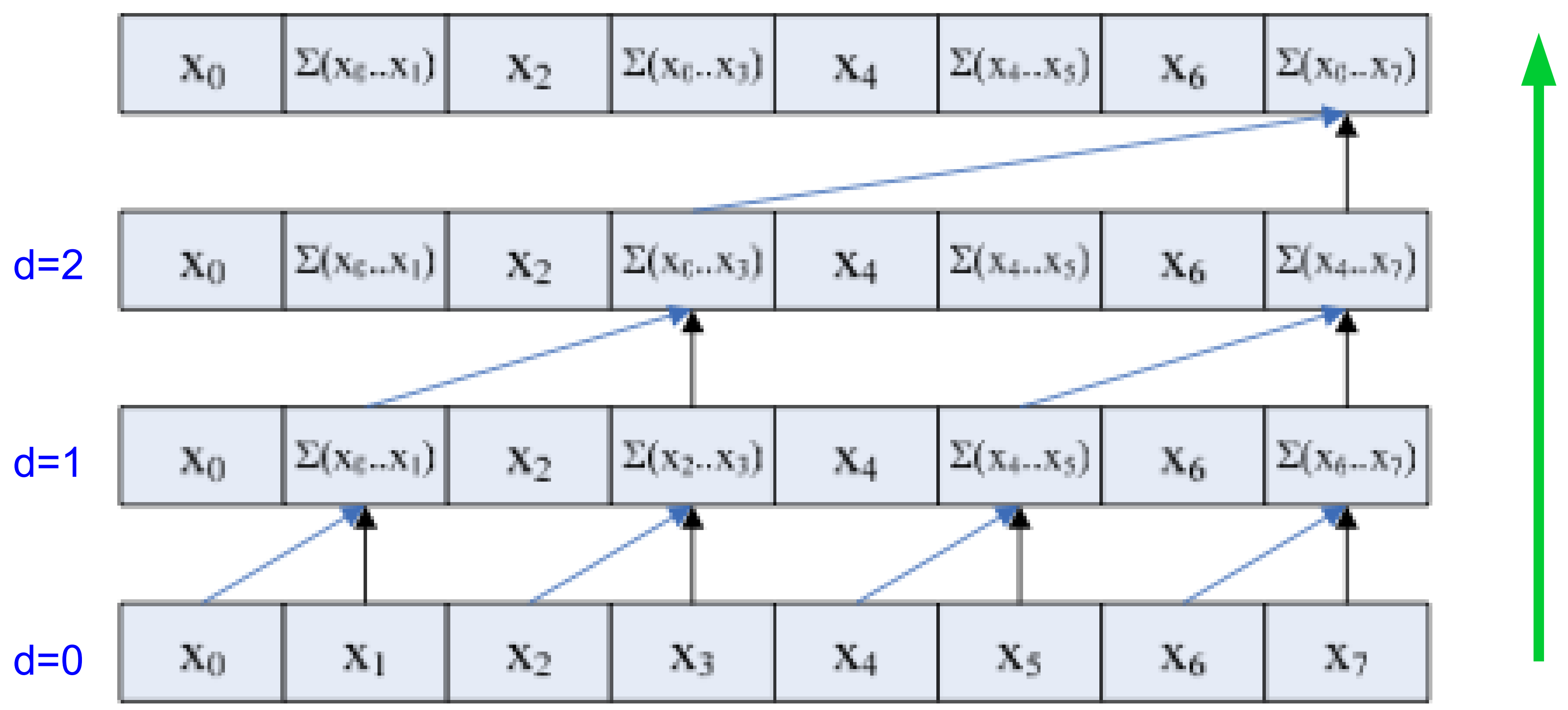


Blelloch's Scan / 3

for d:=0 **to** $\log_2(n)-1$ **do**

  for k **from** 0 **to** $n-1$ **step** 2^{d+1} **in parallel do**

    $x[k + 2^{d+1} - 1] = x[k + 2^{d} - 1] + x[k + 2^{d+1} - 1]$



Blelloch's Scan / 4

- How **many iterations** do we have to perform during the up-sweep?

- How **many operations** do we perform at each iteration?

- What is the **overall amount** of operations during the up-sweep?

Blelloch's Scan / 5

- How **many iterations** do we have to perform during the up-sweep?

$\log_2(n) = M$ (3 in the example).

- How **many sums** do we have to perform during each iteration?

2^{M-d-1}

Blelloch's Scan / 6

- What is the **overall amount** of operations during the up-sweep phase?

Let's make some calculations...

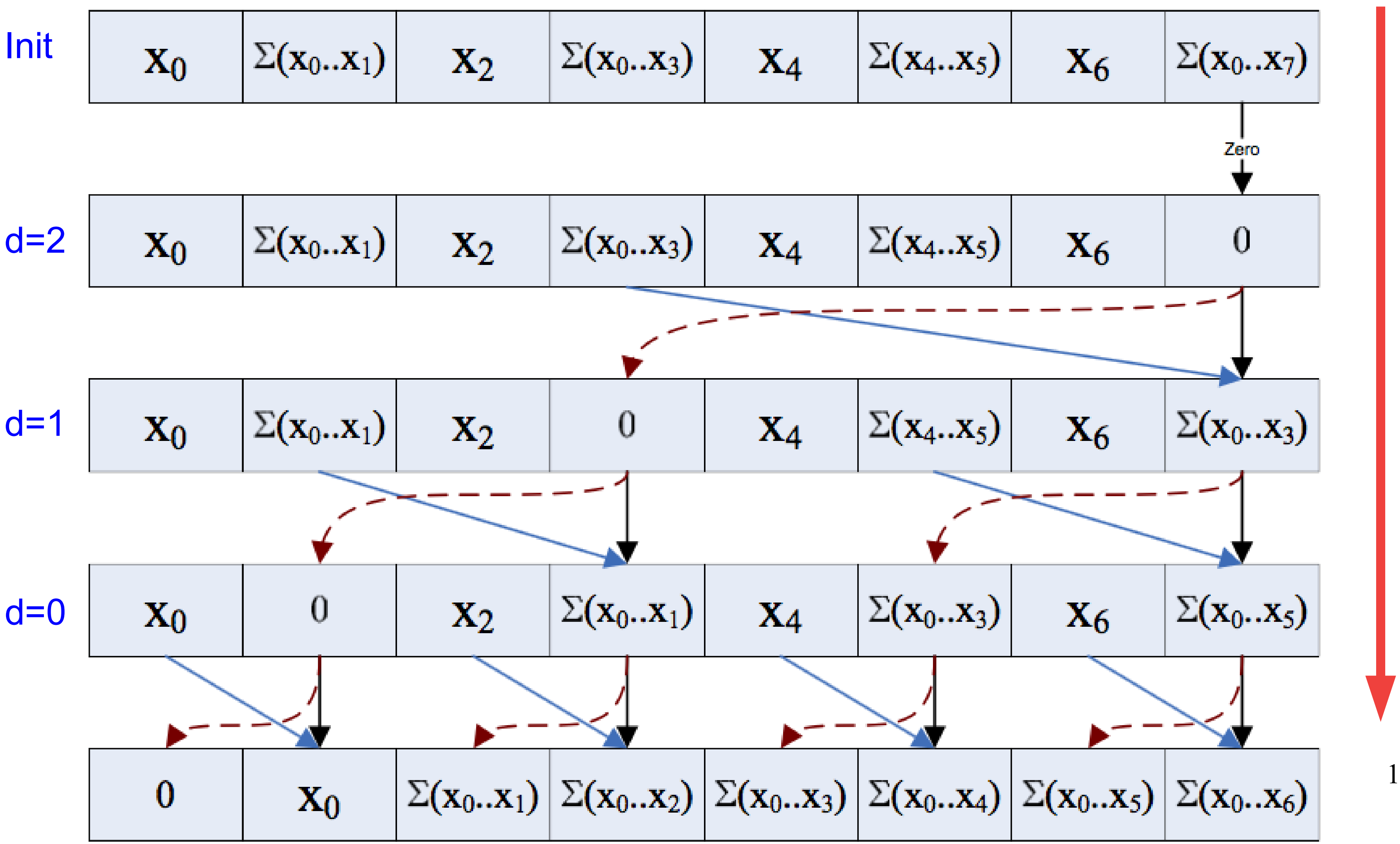
$(2^{M-1} + \ldots + 2^{0}) = 2^{M} - 1 = n - 1$ **adds**

- Hence, the complexity is in $O(n-1)$. **Good**!

- Now we need to **propagate** some of the **partial sums** back in the right places...

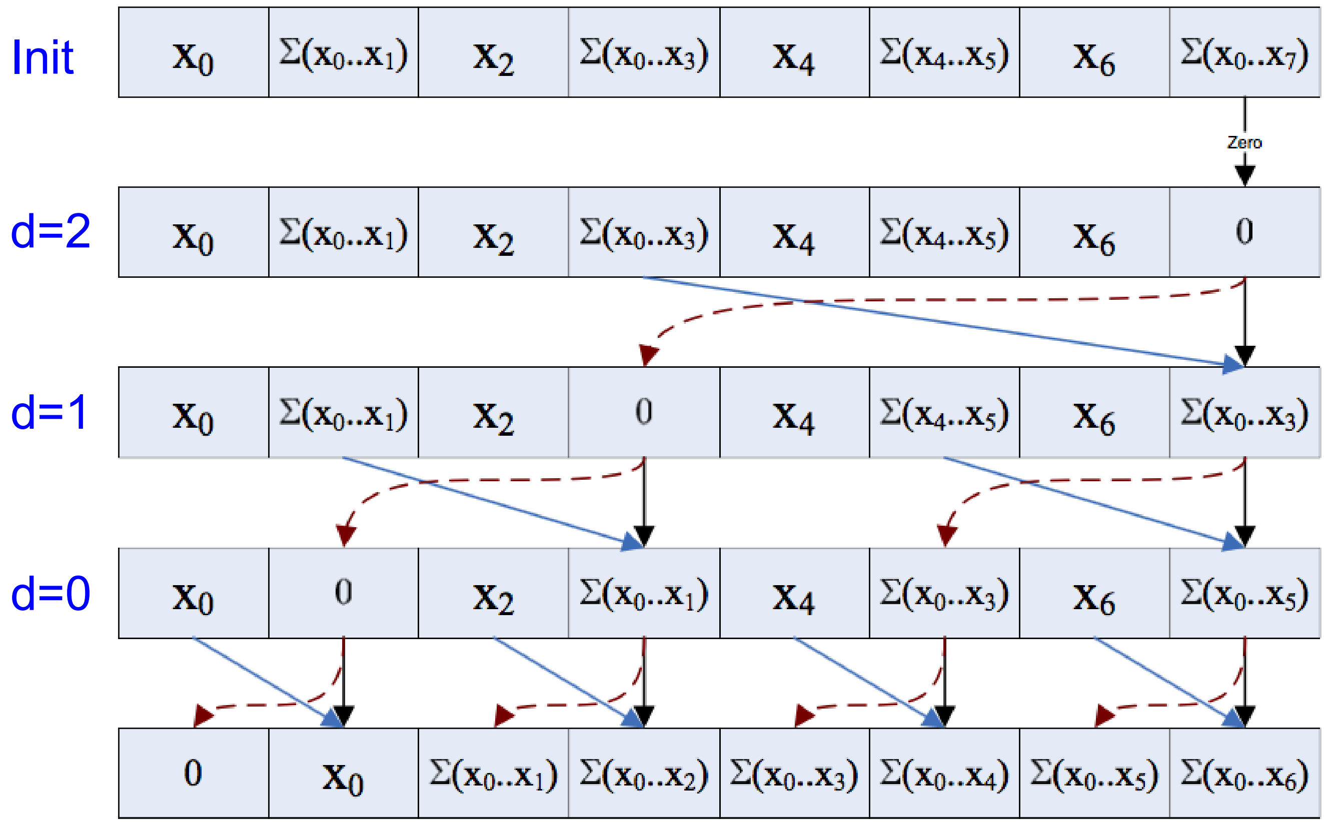
Blelloch's Scan / 7

- **Down-sweep** phase: we **go** from the **root to the leaves**!



Blelloch's Scan / 8

```
x[n - 1] := 0 // Init
for d := log_2(n)-1 down to 0 do
   for k from 0 to n – 1 step 2^(d+1) in parallel do
      t := x[k + 2^d - 1]
      x[k + 2^d - 1] := x [k + 2^(d+1) − 1] // Copy
      x[k + 2^(d+1) - 1] := t + x [k + 2^(d+1) - 1] // Add
```



Blelloch's Scan / 9

- How **many iterations** do we have to perform during the down-sweep?

- How **many operations** do we perform at each iteration?

- What is the **overall amount** of operations during the down-sweep?

Blelloch's Scan / 10

- How **many iterations** do we have to perform during the down-sweep?

$\log_2(n) = M$ (3 in the example).

- How **many operations** do we have to perform during each iteration?

2^{M-d-1} moves $+$ 2^{M-d-1} adds

Blelloch's Scan / 11

- What is the **overall amount** of moves and adds during the up-sweep phase?

Let's make some calculations…

$(2^{M-1} + \ldots + 2^{0}) = 2^{M} - 1 = n - 1$ moves + the same for adds

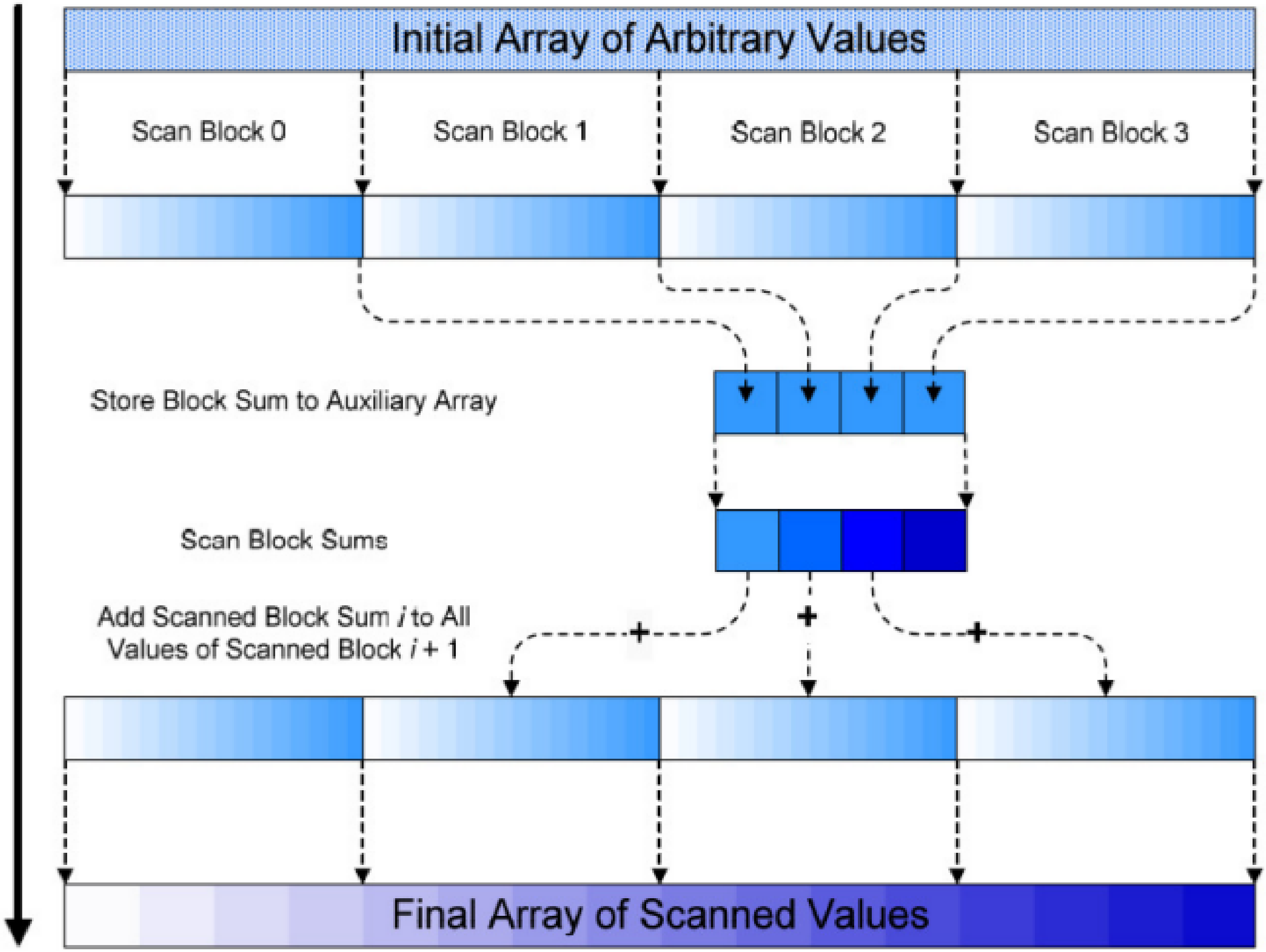
- Hence, the complexity is in **$O(2(n - 1))$.**

- Overall, the complexity of the Blelloch's algorithm is is $O(3(n-1)) \Rightarrow O(n)$. **Good!**

```
__global__ void prescan(float *g_odata, float *g_idata, int n)
{
    extern  __shared__  float temp[];// allocated on invocation

    int thid = threadIdx.x;
    int offset = 1;

A   temp[2*thid]    = g_idata[2*thid]; // load input into shared memory
    temp[2*thid+1] = g_idata[2*thid+1];

    for (int d = n>>1; d > 0; d >>= 1) // build sum in place up the tree
    {
        __syncthreads();

        if  (thid < d)
        {
B           int ai = offset*(2*thid+1)-1;
            int bi = offset*(2*thid+2)-1;

            temp[bi] += temp[ai];
        }
        offset *= 2;
    }

    if (thid == 0) { temp[n - 1] = 0; } // clear the last element
C
    for (int d = 1; d < n; d *= 2) // traverse down tree & build scan
    {
        offset >>= 1;
        __syncthreads();

        if  (thid < d)
        {
D           int ai = offset*(2*thid+1)-1;
            int bi = offset*(2*thid+2)-1;

            float t   = temp[ai];
            temp[ai]  = temp[bi];
            temp[bi] += t;
        }
    }

    __syncthreads();

E   g_odata[2*thid]   = temp[2*thid]; // write results to device memory
    g_odata[2*thid+1] = temp[2*thid+1];
}
```

Using multiple thread-blocks

- What if we want to extend the scan to vectors of arbitrary size?



Exercises / 1

- To complete the exercises you will **need**:
  - Recommended O/S: Linux (e.g., Ubuntu $>=$ 16.04).
  - CUDA 8.5 or 9.
  - A GCC version compatible with CUDA (GCC 6.x is fine).
  - A NVIDIA videocard.
  - Cmake.
  - (optional) Thrust library.
- I will also give you a little zip containing the **source code** of a little application that you will have to **complete** by providing the **required solutions**.

Exercises / 2

- **Exercise 1**: implement the GPU-based version of the **Hill-Steele** algorithm, limitedly to the case of a **single thread-block.**

- **Exercise 2**: implement the Blelloch's (down/up-sweep) algorithm, limitedly to the case of a **single thread-block**.

- **Exercise 3**: Adapt the code written for **exercise 2** to implement the **inclusive prefix sum** (easy!).

- **Exercise 4**: Extend the code of **exercise 2** to make use of **multiple thread-blocks**.

- **Exercise 5** (**optional!**): try to avoid the **bank conflicts** in shared memory.

Questions?