UNIVERSITÀ di VERONA

Dipartimento di INFORMATICA

PARCO Parallel Computing Lab

Part IV

Laurea magistrale in Ingegneria e Scienze Informatiche
Laurea Magistrale in Medical Bioinformatics

Nicola Bombieri – Federico Busato

Agenda

- Inclusive/Exclusive Prefix Scan
- Sequential Implementation
- Naive Parallel Implementation
- Work-Efficient Parallel Implementation
  - UpSweep
  - DownSweep
- Exercises
- References

PREFIX-SCAN

- The prefix scan operation takes
  - a binary associative operator $\oplus$
  - an array of N elements $[\, x_0, x_1, \ldots, x_{N-1} \,]$
- It returns the following array
  - Inclusive: $[\, x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \oplus \ldots \oplus x_{N-1}) \,]$
  - Exclusive: $[\, 0, x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \oplus \ldots \oplus x_{N-2}) \,]$
- Example:
  - $\oplus$ = addition (prefix-sum)
  - Inclusive: $[3\ 1\ 7\ 4\ 6] \rightarrow [3\ \ 4\ \ 11\ \ 15\ \ 21]$
  - Exclusive: $[3\ 1\ 7\ 4\ 6] \rightarrow [0\ \ 3\ \ \ 4\ \ \ 11\ \ 15]$

INCLUSIVE ←→ EXCLUSIVE PREFIX SCAN

- Exclusive scan → inclusive scan
  - Shift the resulting array left by one element
  - Insert at the end the sum of the last element of the scan and the last element of the input array
- Inclusive scan → exclusive scan
  - Shift the resulting array right by one element
  - Insert the identity at the beginning

PREFIX SCAN Uses

Prefix scan is a key primitive in many parallel algorithms to convert serial computation into parallel computation

- Sorting (counting sort/radix sort)
- Histogram
- Bulk Queue Insertion
- Stream Compaction/Partition
- Sparse matrix multiplication
- Building data structures in parallel
- …

Sequential Implementation

Inclusive

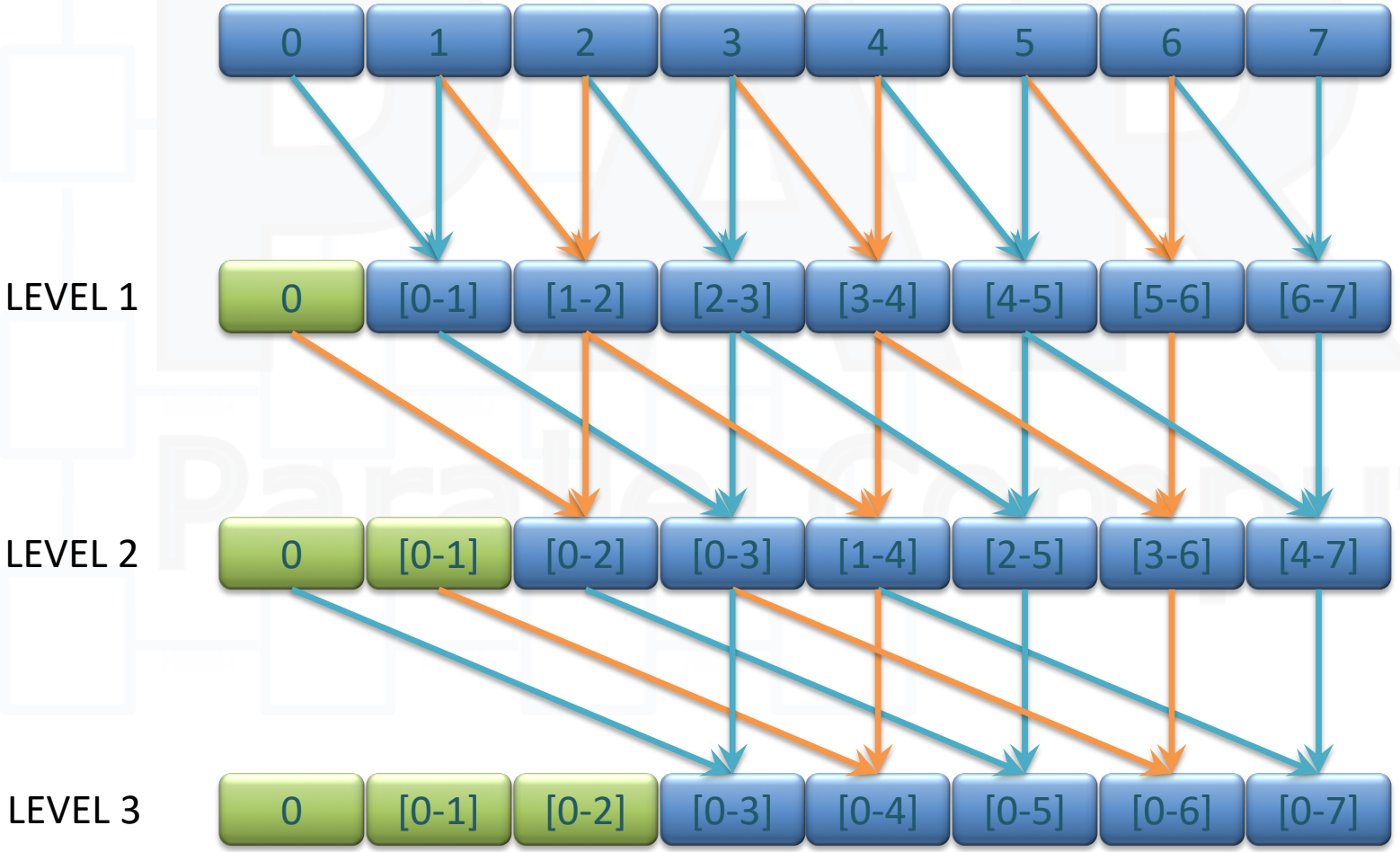
```
Out[0] = In[0];
for (int i = 1; i < N; ++i)
    Out[i] = Out[i – 1] + In[i];
```

Exclusive

```
Out[0] = 0;
for (int i = 1; i < N; ++i)
    Out[i] = Out[i – 1] + In[i – 1];
```

- *N* additions needed for *n* elements
  - Complexity is O(*n*)
- Parallel implementation is not as straightforward
  - The computation for each element requires the result for the previous element → data dependency

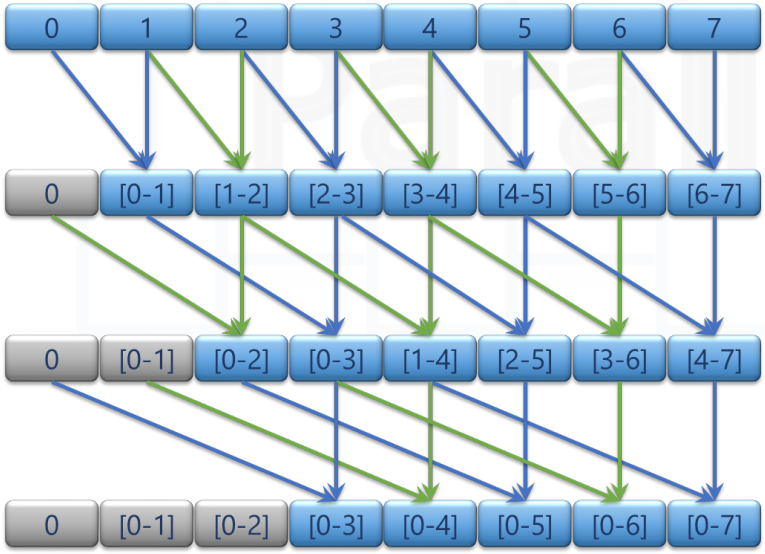
A Naive Parallel Scan (I)



A Naive Parallel Scan (II)

Hillis-Steele PRAM algorithm (1986)

```
for (int level = 0; level < log2(N); ++level) {
    parallel_for ( ∀i ∈ N ) {
        offset = 2^LEVEL
    if (i >= offset)
            X[i] = X[i – offset] + X[i]
    }
}
```

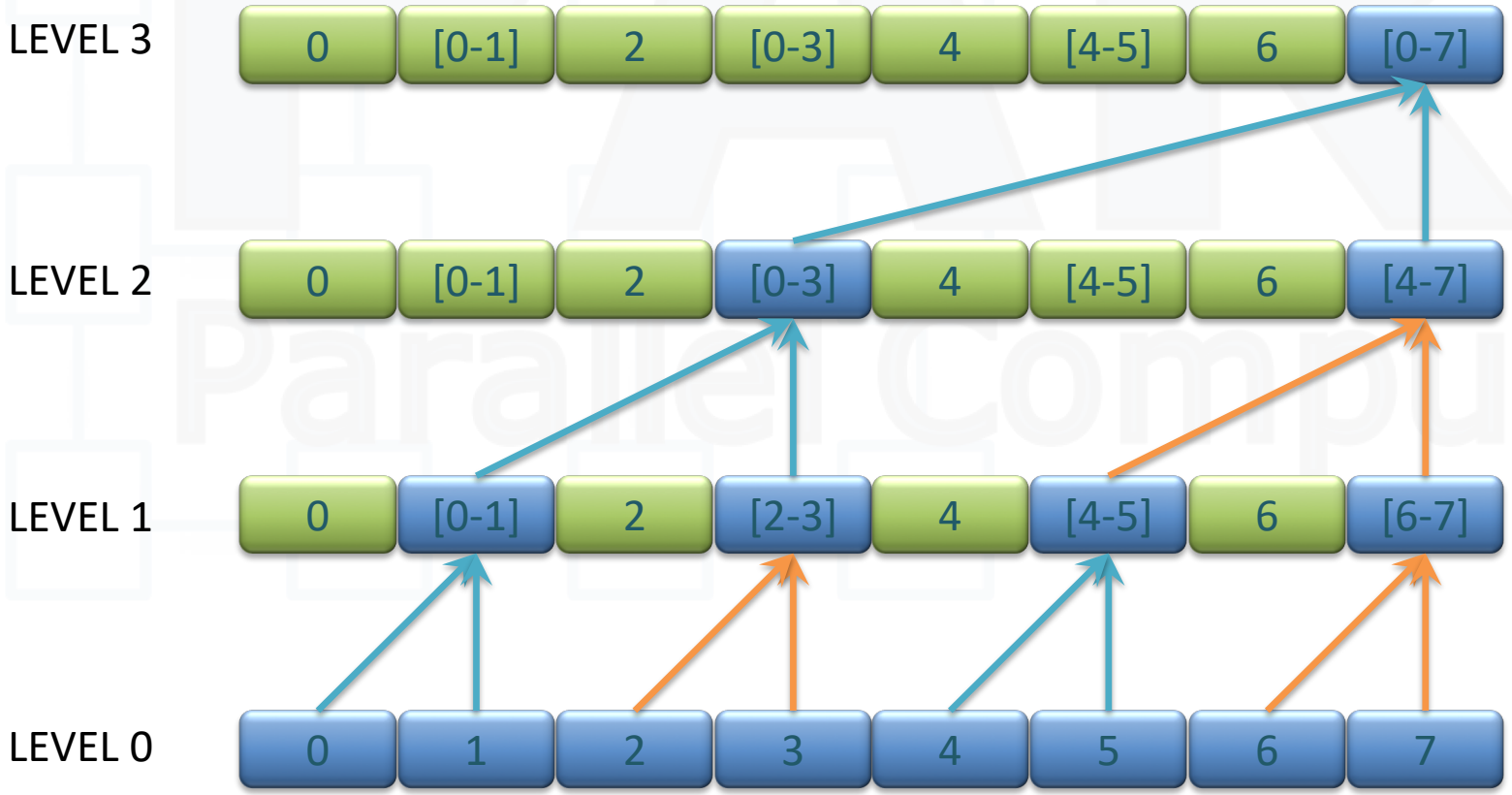


Sequential Complexity	$O(n)$
Parallel Complexity (Span)	$O(log(n))$
Sequential Complexity (Work)	$O(n \cdot log(n))$
Step Efficiency	$log(n)$

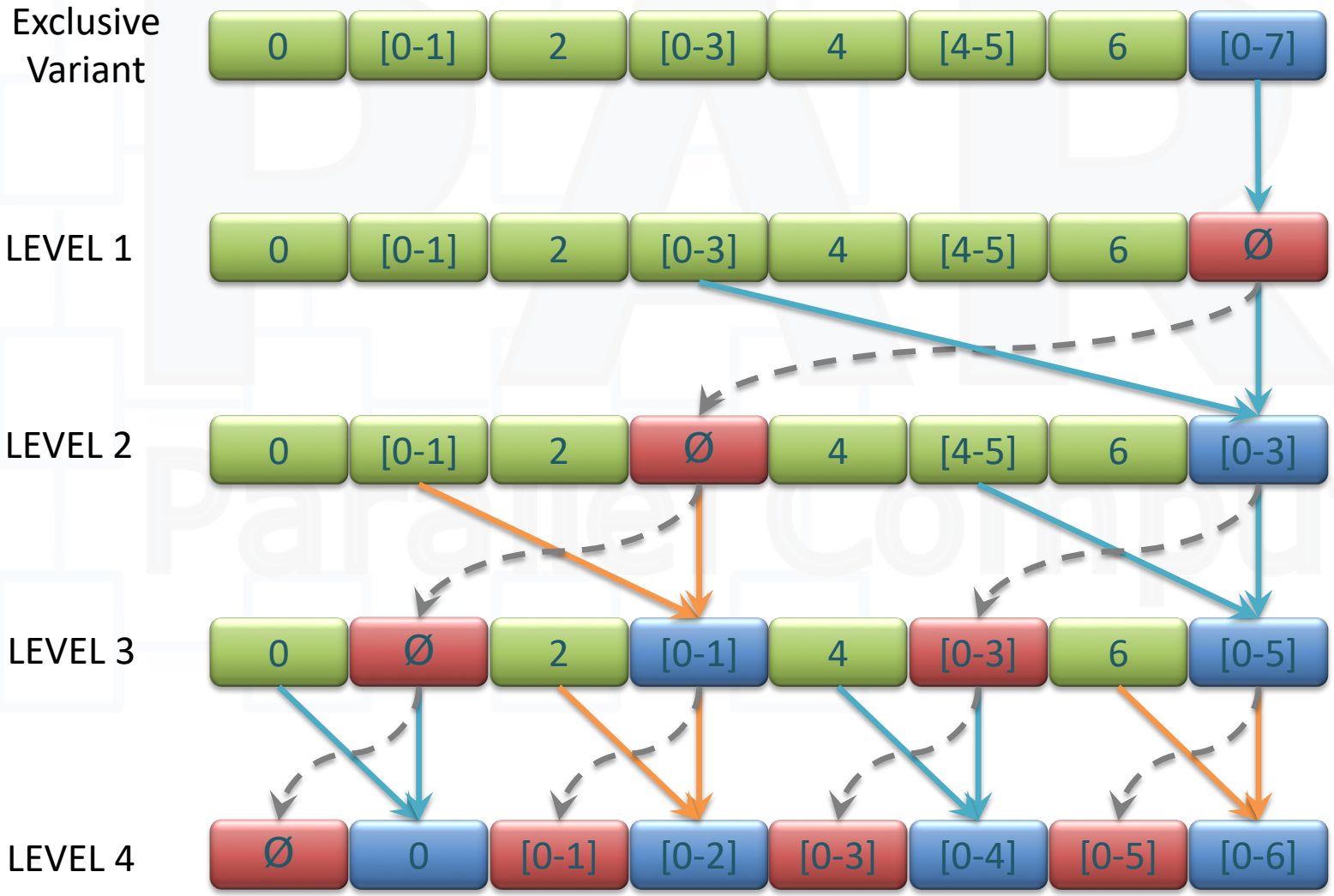
Work-Efficient Parallel Scan (I)

Blelloch (1989) (Brent-Kung Scan Circuit family)(GPU Gems 3, 2007)

- The algorithm consists of two phases:
  - up-sweep phase (reduce phase) (below)
  - down-sweep phase (inverse reduce phase) (next page)

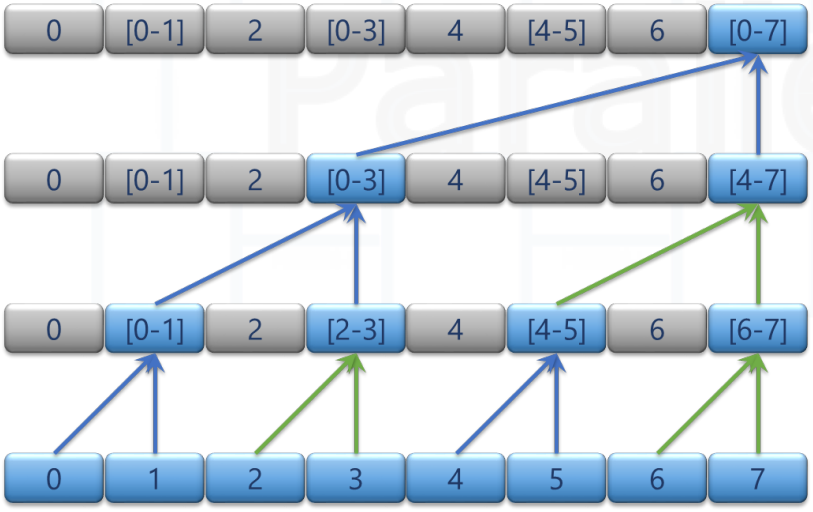


Work-Efficient Parallel Scan (II)



Work-Efficient Parallel Scan (III)

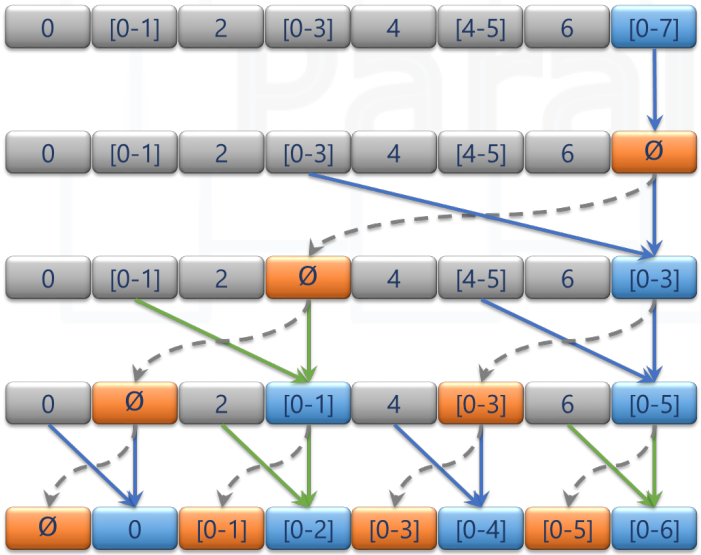
```
for (level = 0; level < log2(N); ++level) {
    step = 2^LEVEL
    parallel_for ( ∀i ∈ N | i % (step * 2) == 0 )
        valueRight = (i + 1) * (step * 2) – 1
        valueLeft = valueRight – step
        X[valueRight] = X[valueRight] + X[valueLeft]
}
```



Sequential Complexity	$O(n)$
Parallel Complexity (Span)	$O(log(n))$
Sequential Complexity (Work)	$O(n)$

Work-Efficient Parallel Scan (IV)

```
X[N - 1] = 0; K = 1;
for (level = log2(N) - 1; level ≥ 0; --level, K *= 2 ) {
    step = 2^LEVEL
    parallel_for ( ∀i ∈ N | i < K )
        valueLeft = (i * 2 + 1) * step – 1
        valueRight = valueLeft + step
        Tmp = X[valueLeft]
        X[valueLeft] = X[valueRight]
        X[valueRight] = Tmp + X[valueRight]
}
```



Sequential Complexity	$O(n)$
Parallel Complexity (Span)	$O(log(n))$
Sequential Complexity (Work)	$O(n)$

Work-Efficient Parallel Scan (V)

Near the real implementation…

```
step = 1
for (limit = blockDim / 2; limit > 0; limit /= 2) {
     if (ThreadID < limit)
          valueRight = (ThreadID + 1) * (step * 2) – 1
          valueLeft = valueRight – step
          X[valueRight] = X[valueRight] + X[valueLeft]
     step *= 2
     sync()
}
if (threadIdx.x == 0)
     X[blockDim.x - 1] = 0
sync()
limit = 1;
for (step = blockDim / 2; step > 0; step /= 2) {
     if (threadID < limit)
          valueRight = (threadID * 2 + 1) * step - 1
          valueLeft = valueRight - step
          tmp = X[valueLeft]
          X[valueLeft] = X[valueRight]
          X[valueRight] = tmp + X[valueRight]
     limit *= 2
     sync()
}
```

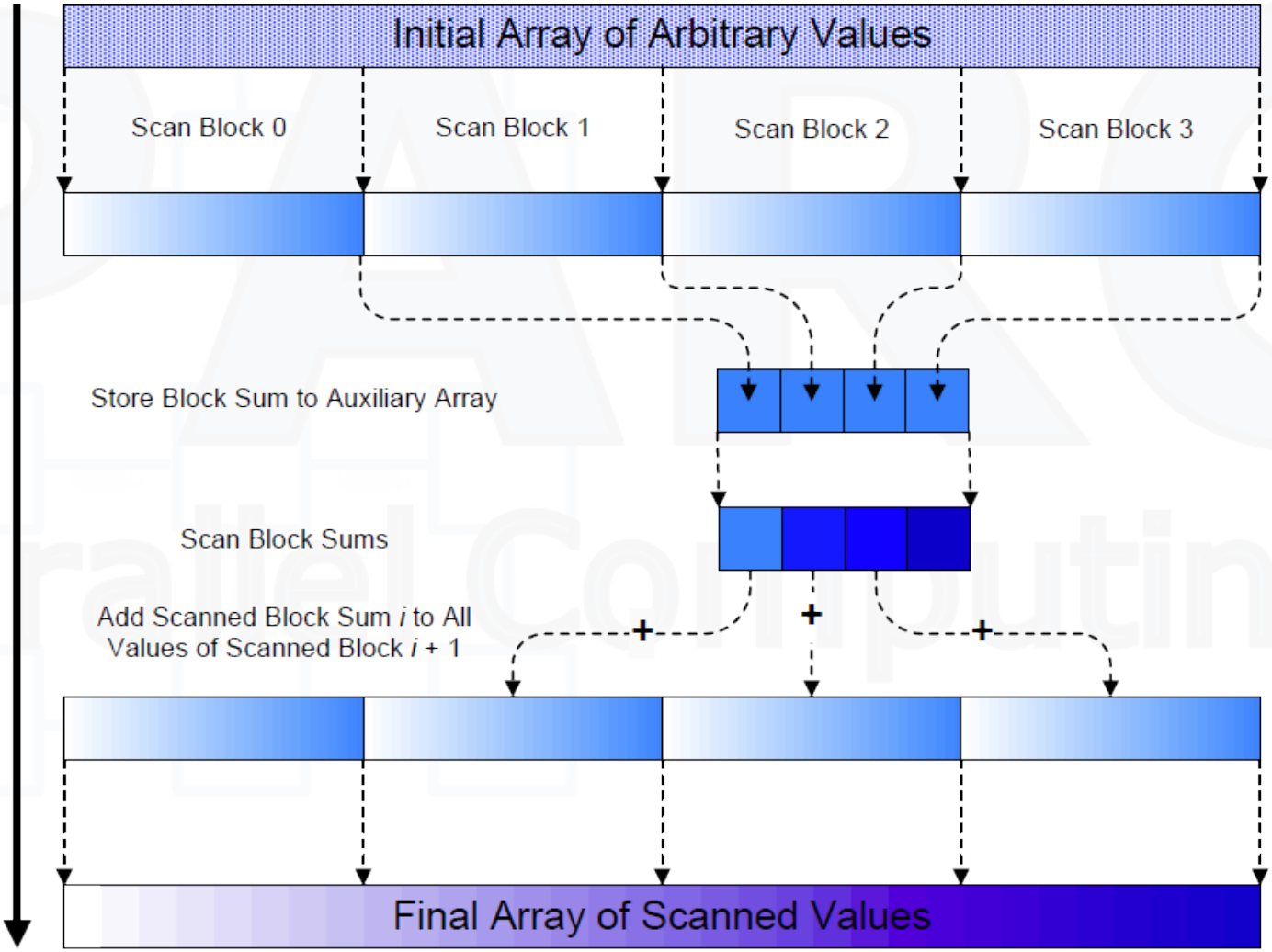
Work-Efficient Parallel Scan (VI)

- Total number of iterations: 2 * log(n)
  - No Step Efficient $2log(n)$
- Total number of additions: 2 * (n – 1)
- Total number of operations: O(n) work
- The total number of additions is no more than twice that done in the efficient sequential algorithm
  - Parallelism can easily overcome this increase in work (especially for large arrays)

Device-wide Parallel Scan (I)

Scan-then-Fan (GPU Gems 3, 2007) (4x GMem access)

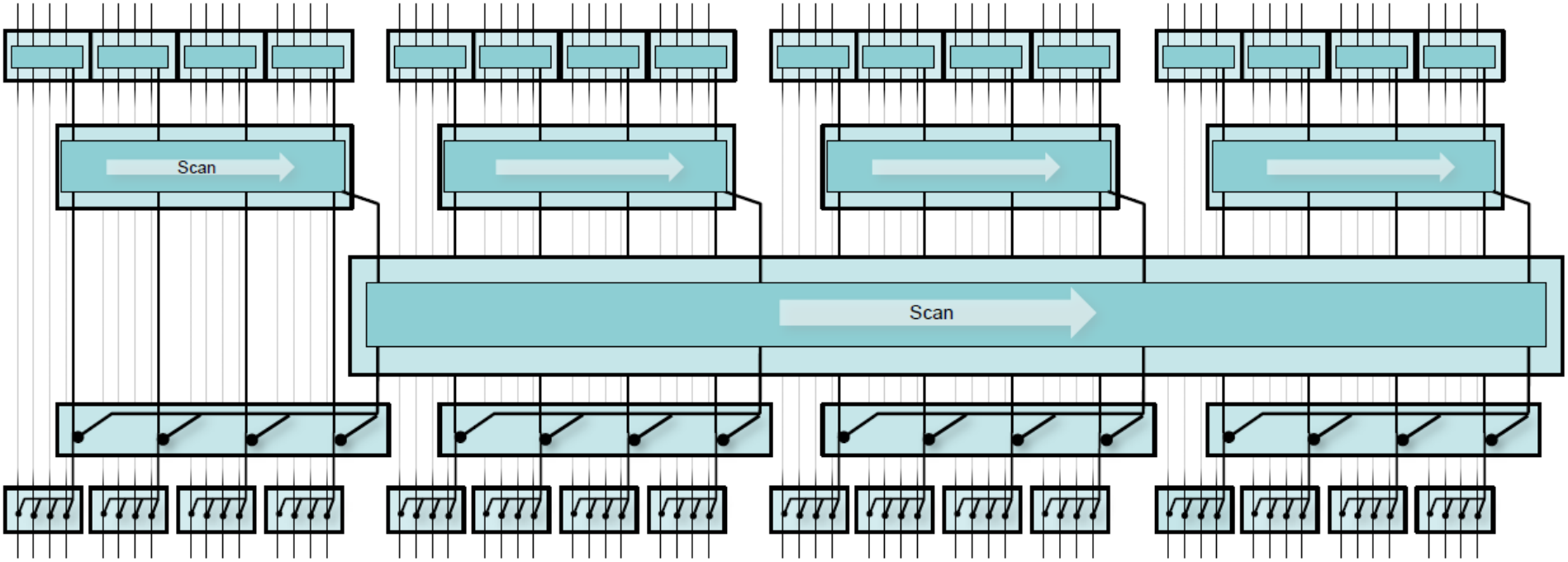
1) Each block writes the sum of its section into a sum array indexed by blockIdx.x
2) Run scan kernel code on the sum array
3) Add the scanned sum array values to each element of the corresponding section



Device-wide Parallel Scan (II)

Scan-then-Propagate (CUDPP, 2008)

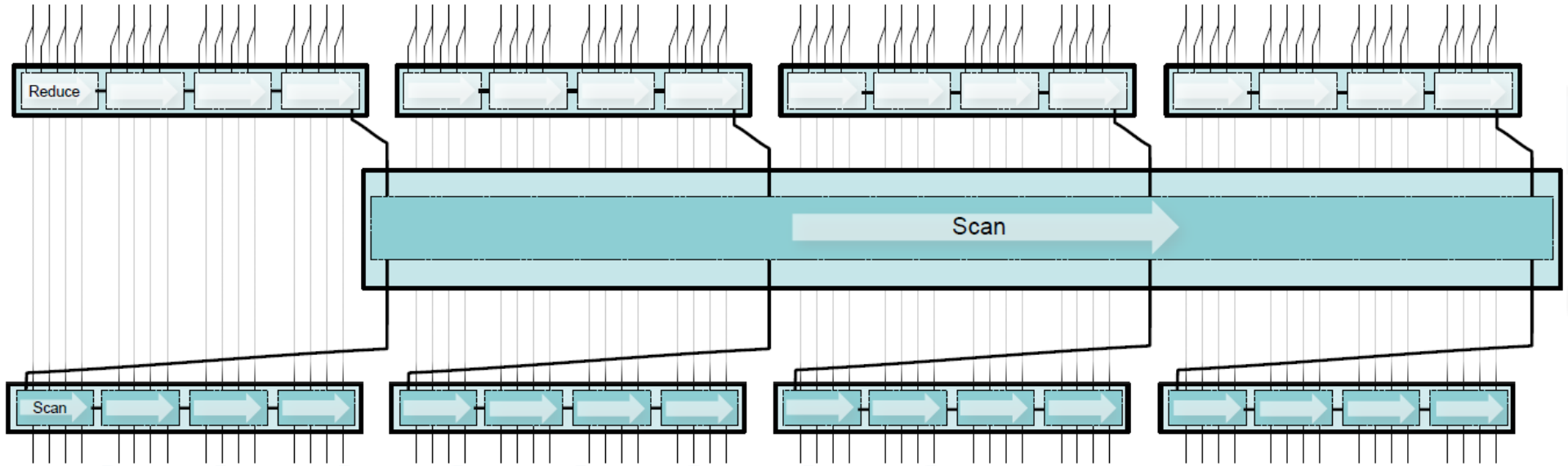
- Log(N) levels upsweep/downsweep
- 4x GMem access



Device-wide Parallel Scan (III)

Reduce-then-Scan (Merrill, 2009)

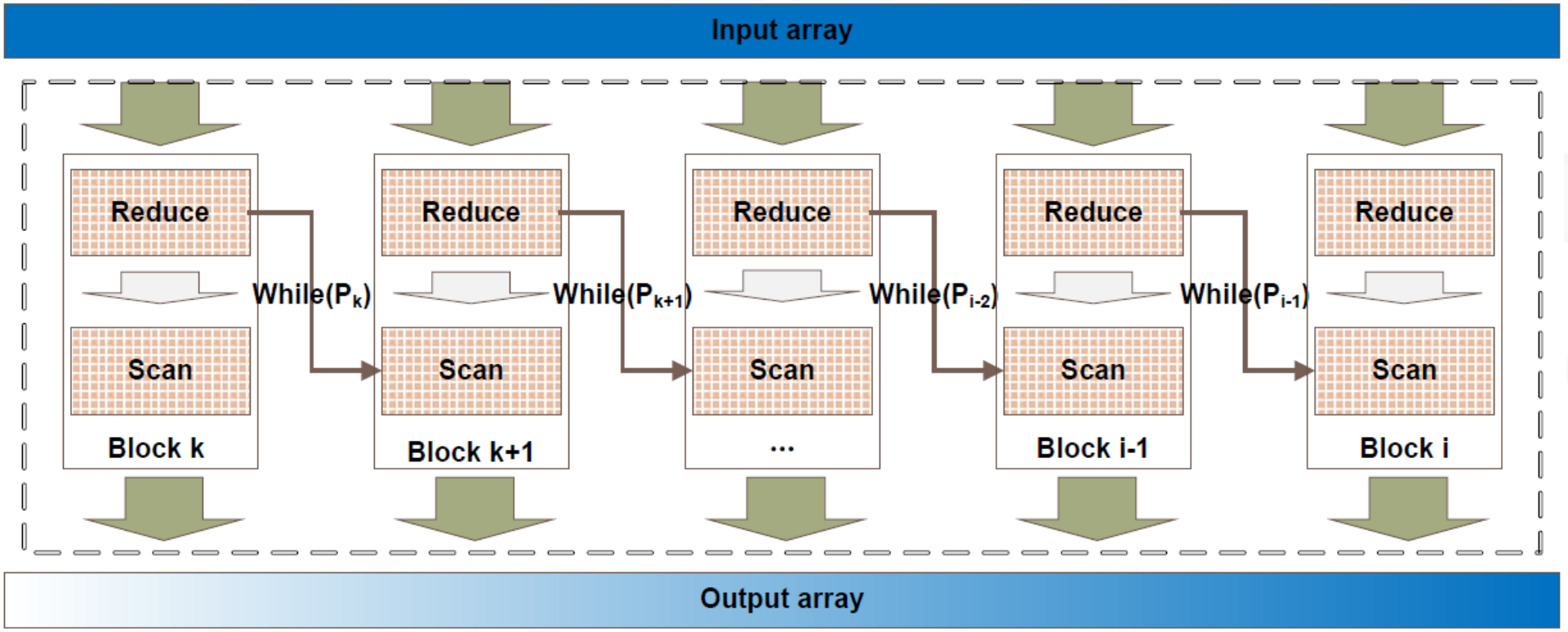
- Requires only 3 kernel launches vs. Log(N)
- 3x GMem access



Device-wide Parallel Scan (III)

StreamScan (Yan, 2013)

- Serialize reduce chain
- 2x GMem access



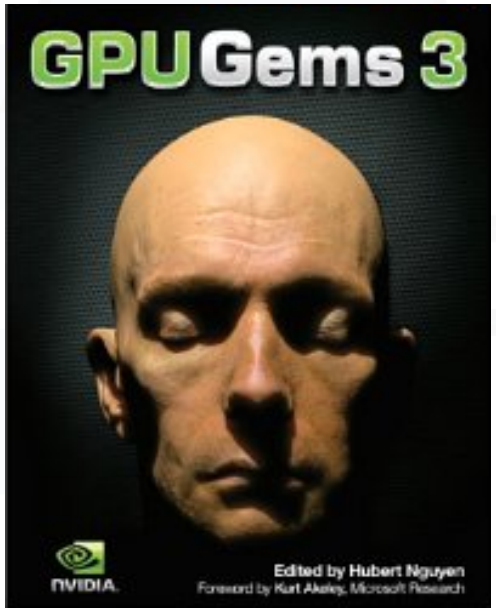
Exercises

- Implement prefix sum:
  - Sequential
  - Parallel v1 (not work efficient)
  - Parallel v2 (work efficient)
  - Measure the speedups

References

Parallel Prefix Sum (Scan) with CUDA

http://http.developer.nvidia.com/GPUGems3/gpugems3_ch39.html



References

- CUDA C Programming Guide

  http://docs.nvidia.com/cuda/cuda-c-programming-guide/

- CUDA Runtime API

  http://docs.nvidia.com/cuda/cuda-runtime-api/

  http://developer.download.nvidia.com/compute/cuda/4_1/rel/toolkit/docs/online/modules.html

  (Memory Management)

- CUDA Math API

  http://docs.nvidia.com/cuda/cuda-math-api/