CS 6023 - GPU Programming

Parallel Scan Operation

17/10/2018

Quiz

- Scheduled on Mon 22nd
- Interchange of A and G slots => **Quiz at 8am**
- Different format this time
  - Subjective questions
  - Write code, give explanations, derive formulae etc.
  - Syllabus: Lectures 7, 8, 9, 10, 11 (today)
  - Marks: 3x3 + 1 (for neatness/concise answers)
  - Allowed one sheet of paper with notes

Setting and Agenda

- We are looking at some common parallel programming patterns and how to optimize them for GPUs
    - Parallel scan
- Work-efficiency in parallel algorithms

Scan

- Given a binary associative $\otimes$ operation and an array of n elements

  $[x_0, x_1, \ldots, x_{n-1}]$

  the **scan operation** computes the array

  $[x_0, x_0 \otimes x_1, \ldots, x_0 \otimes x_1 \otimes \ldots x_n]$

Scan

- Given a binary associative ⊗ operation and an array of n elements

  $[x_0, x_1, \ldots, x_{n-1}]$

  the **scan operation** computes the array

  $[x_0, x_0 \otimes x_1, \ldots, x_0 \otimes x_1 \otimes \ldots x_n]$

- When the operation is addition, the scan operation computes the prefix sum or the cumulative sum
- Example: [5 2 1 3 6 7 0 4] -> [5 7 8 11 17 24 24 28]
- It has a wide set of applications in many algorithms from sorting, string comparisons, to statistics

Exclusive and Inclusive Scans

- Inclusive scan
  - `X: [5 2 1  3  6  7  0  4]`
  - `Y: [5 7 8 11 17 24 24 28]`
- Exclusive scan
  - `X: [5 2 1 3  6  7  0  4]`
  - `Y: [0 5 7 8 11 17 24 24]`

- Exercise: Given a kernel for inclusive scan, how do you compute exclusive scan? Vice versa?

Very simple sequential algorithm

Very simple sequential algorithm

```
y[0] = x[0];
for (int k = 1; k < N; ++k)
  y[k] = y[k - 1] + x[k];
```

- Complexity: O(N) operations
- For sequential programs: Parallel reduce and scan have almost same amount of work

- We have a loop (like in matrix multiplication / convolution / parallel reduce). Can we use loop-parallelism here?

The simplest parallel program

The simplest parallel program

```
In thread i, compute y[i] as
y[i] = x[0] + ... + x[i]
```

The simplest parallel program

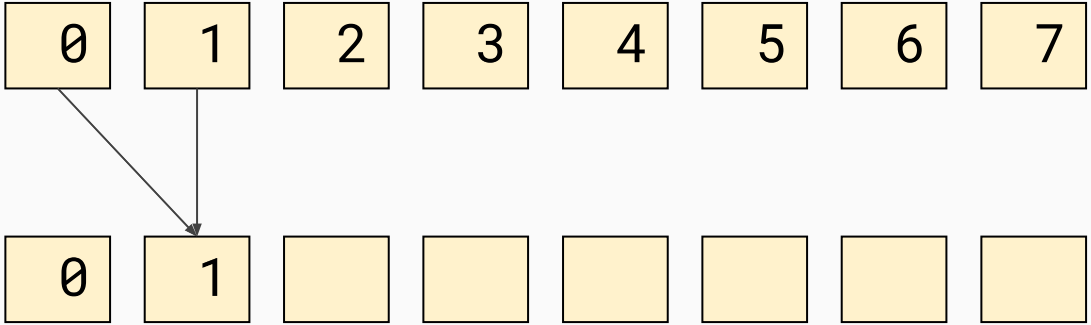
```
In thread i, compute y[i] as
y[i] = x[0] + ... + x[i]
```

- Complexity:
    - $O(N^2)$ operations
    - $O(N)$ time
    - $O(N)$ threads

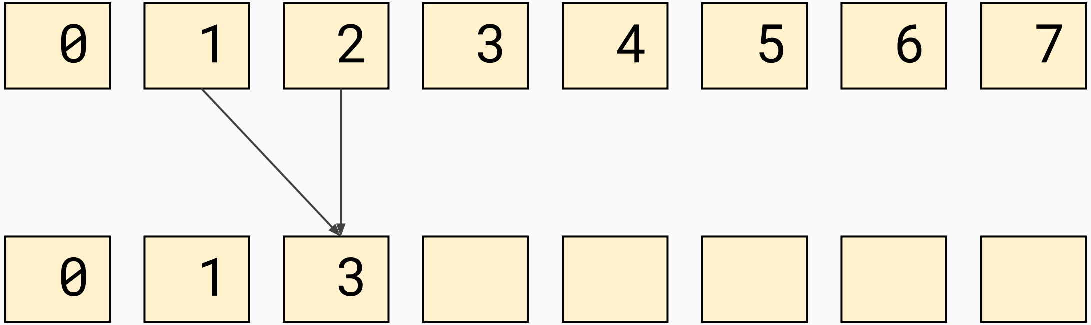
Next best solution

0	1	2	3	4	5	6	7

Next best solution



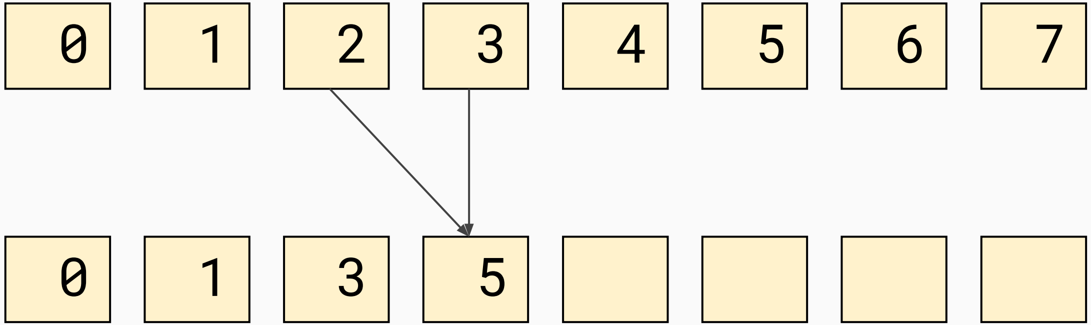
Next best solution



0 1 2 3 4 5 6 7

0 1 3

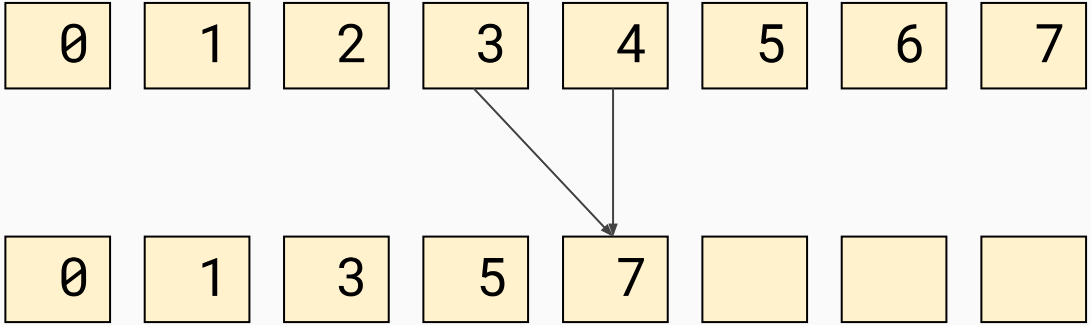
Next best solution



0 1 2 3 4 5 6 7

0 1 3 5

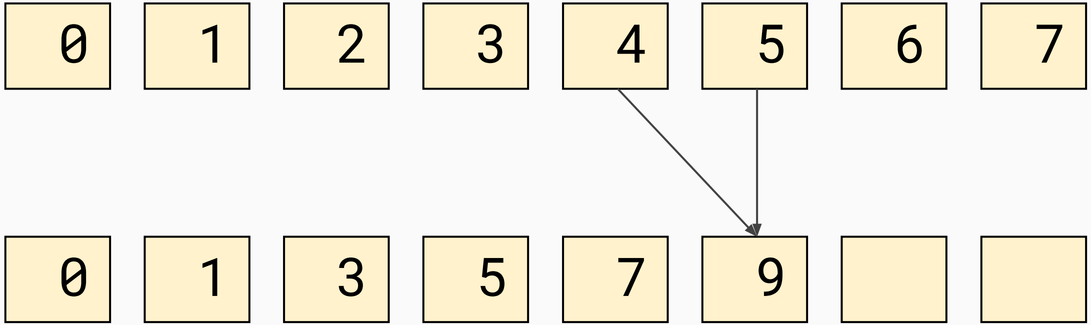
Next best solution



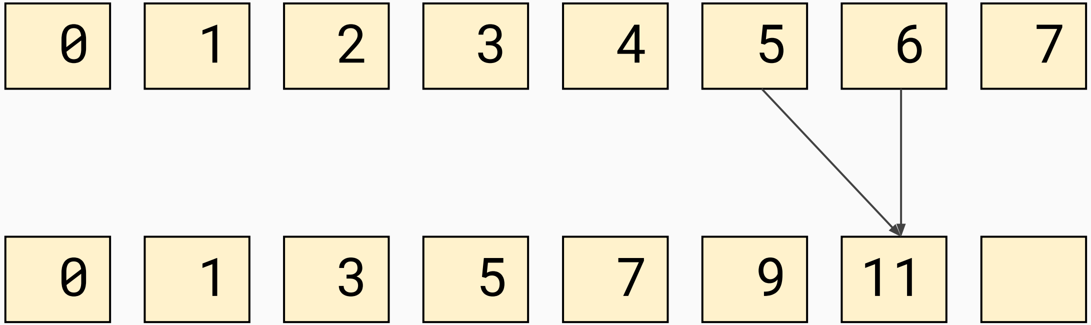
0 1 2 3 4 5 6 7

0 1 3 5 7

Next best solution



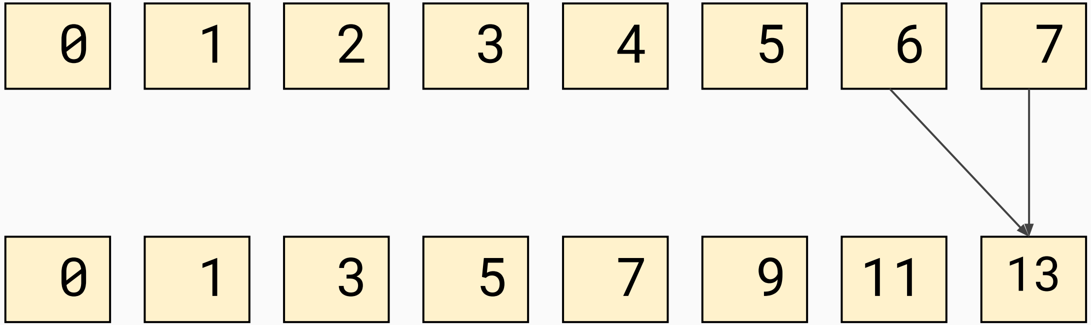
Next best solution



0 1 2 3 4 5 6 7

0 1 3 5 7 9 11

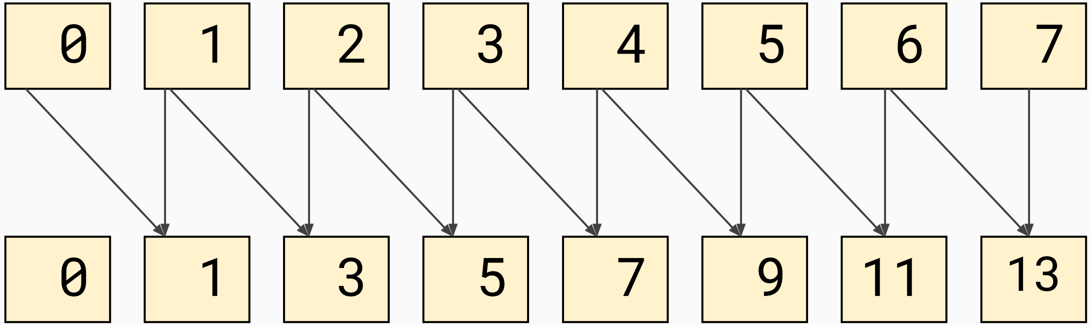
Next best solution



0	1	2	3	4	5	6	7
0	1	3	5	7	9	11	13

Next best solution

End of iteration 1 running in parallel



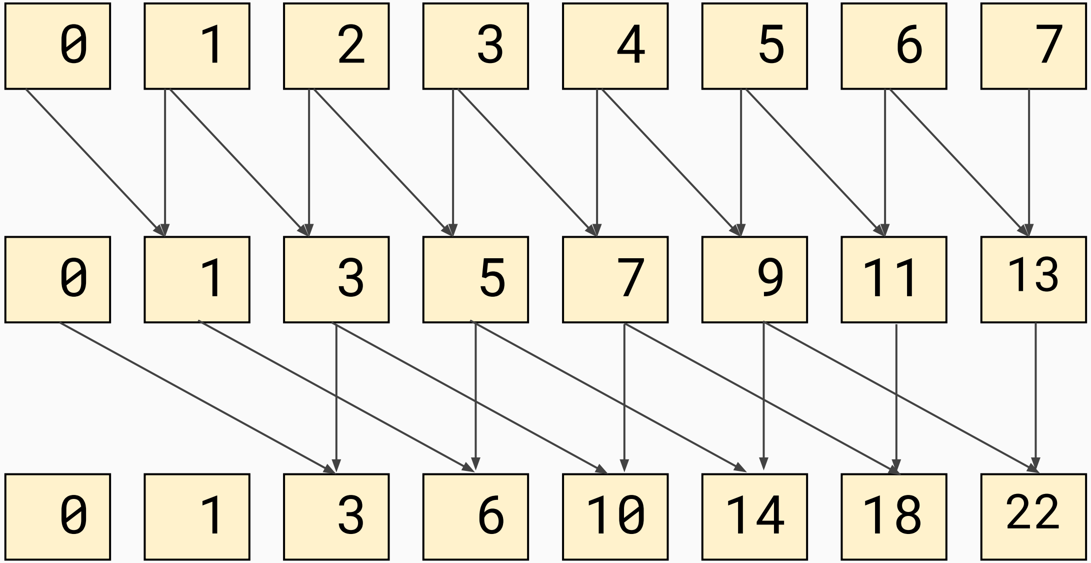
0 1 2 3 4 5 6 7

0 1 3 5 7 9 11 13

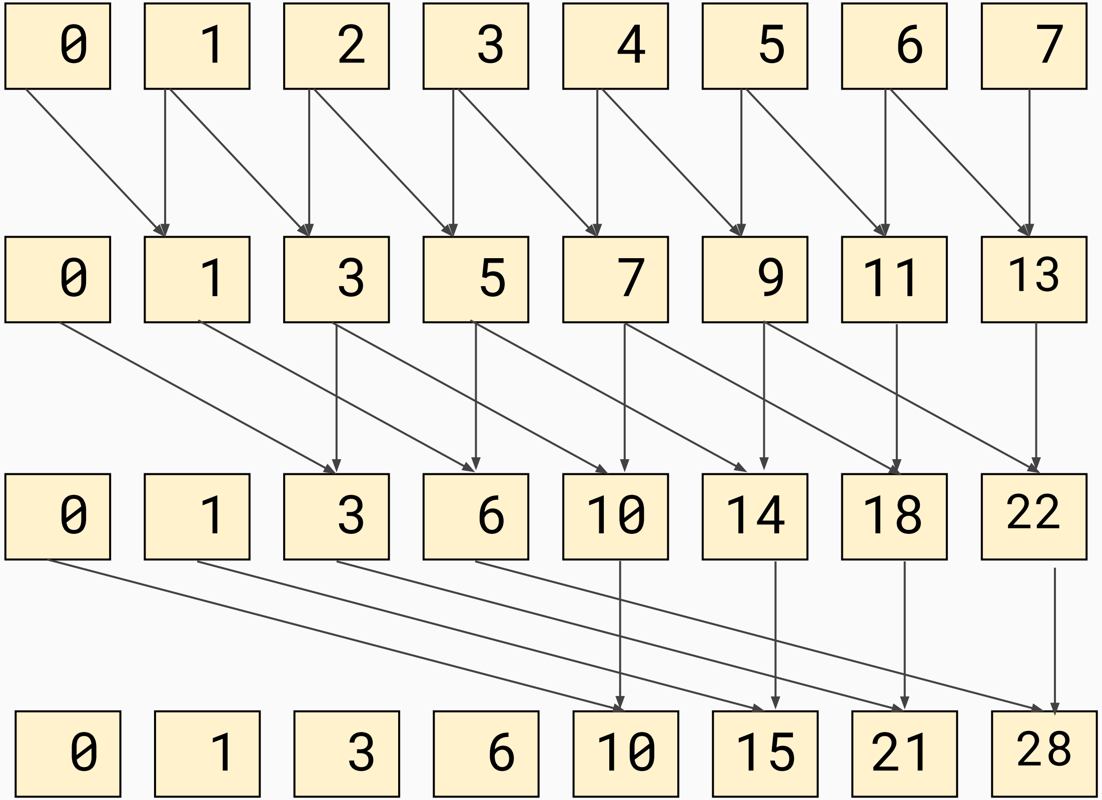
Do we have a problem with in-place edit?

Next best solution

Iteration 2



Next best solution



Iteration 3

Prove that this process computes the prefix sum

What is the complexity?

What is the complexity?

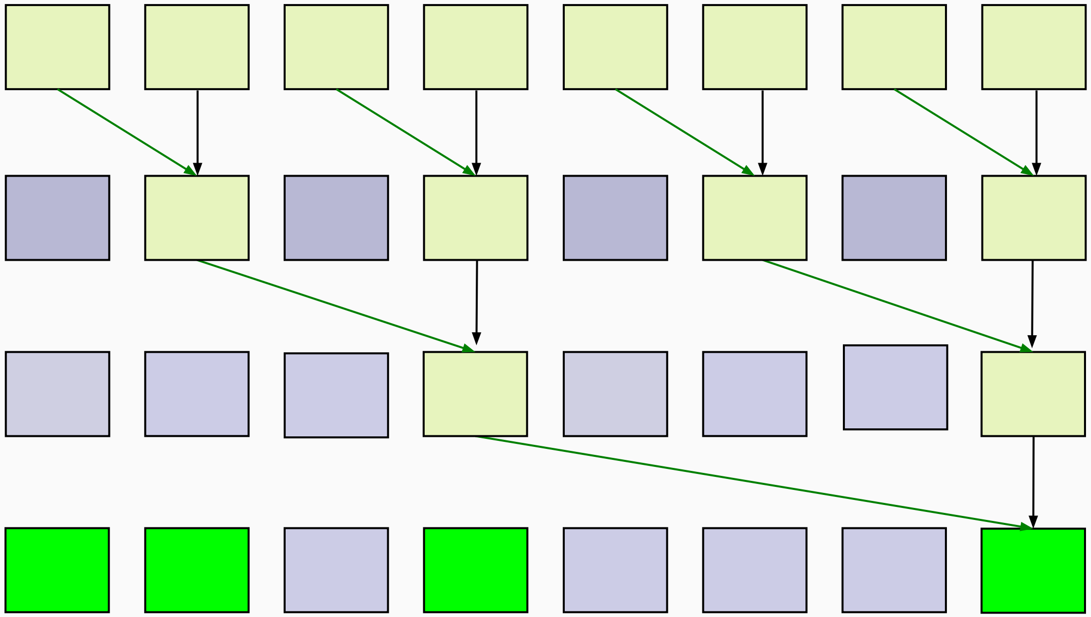
- Complexity:
  - O(log(N)) time
  - O(N) resources
  - Operations?

What is the complexity?

- Complexity:
  - O(log(N)) time
  - O(N) resources
  - Operations?
    - Iteration i does (N - 2^i) operations
    - log(N) number of iterations
    - Total operations: N*log(N) - (N - 1) => O(N*log(N))

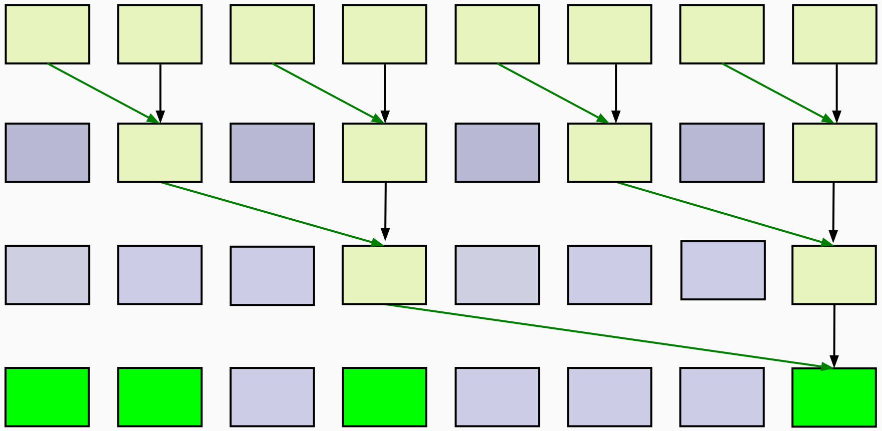
- O(N*log(N)) is an improvement, but it is clearly not work efficient

Can we do even better?

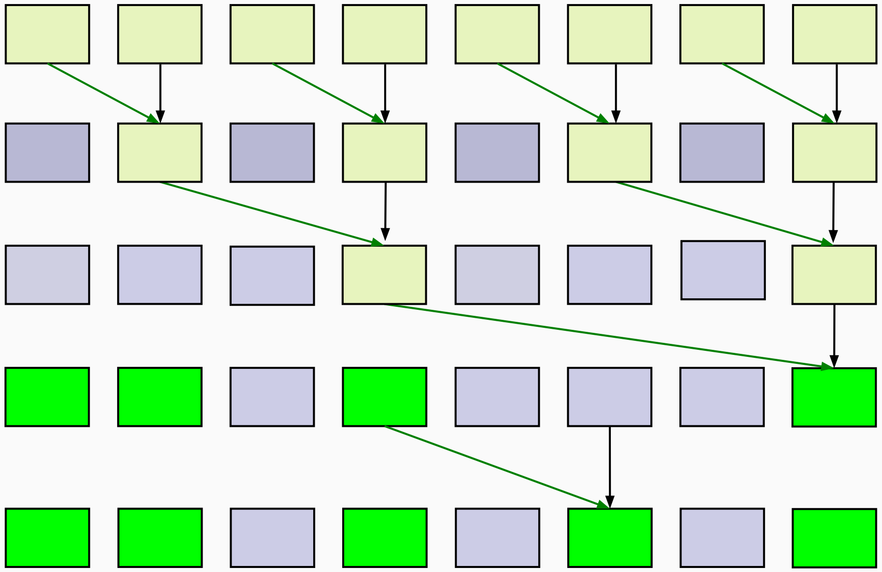


- Consider parallel reduce: At the end some indices have the correct prefix sum
- It has an efficient O(N) complexity of operations
- Formally, what is the value at each index?

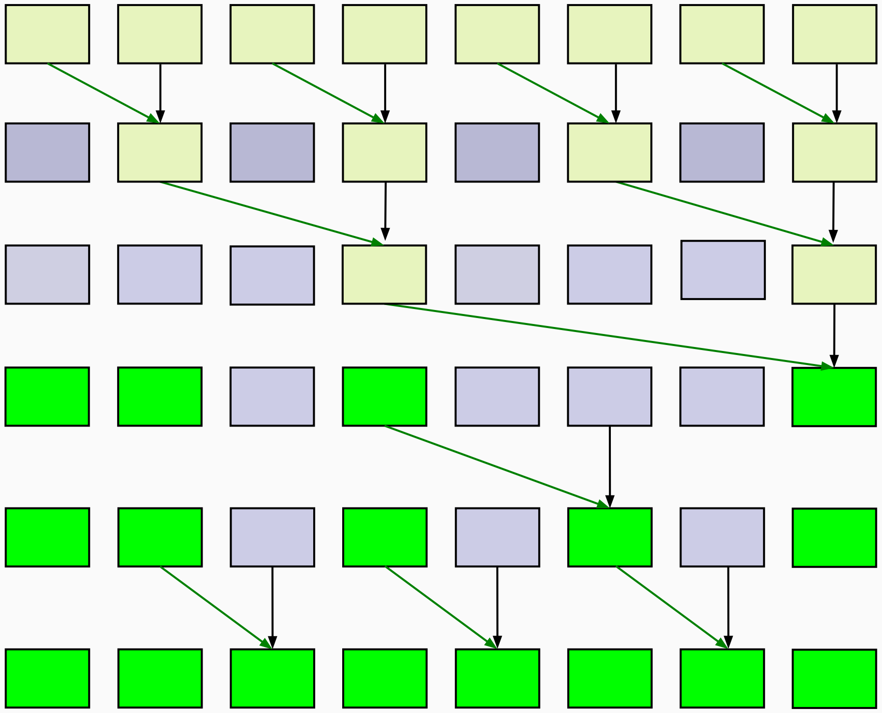
Second phase



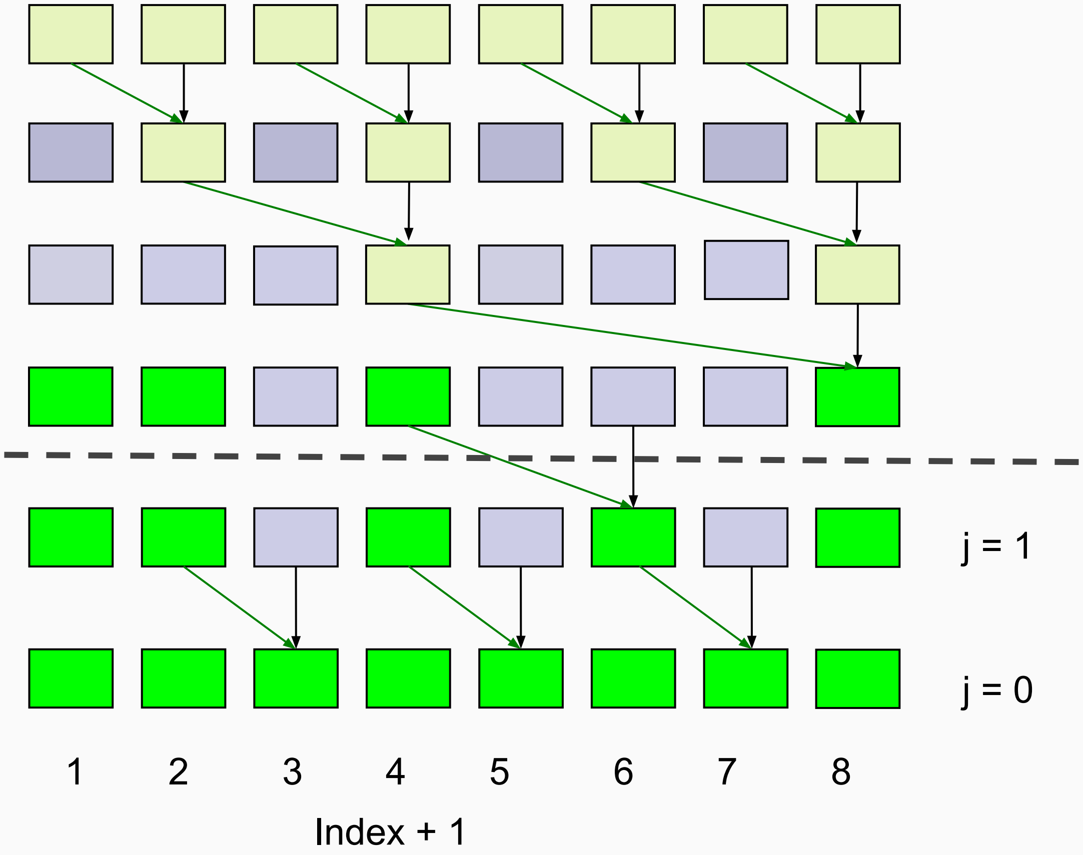
Second phase



Second phase

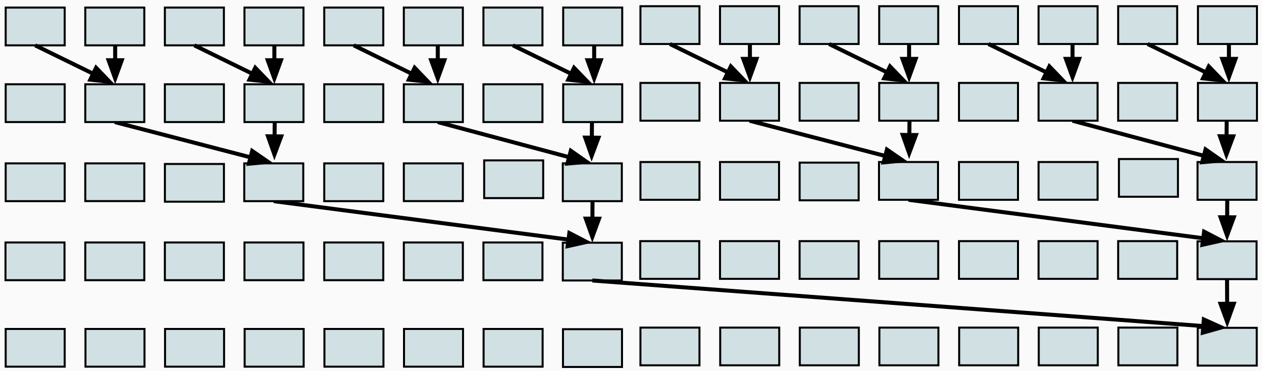


Second phase



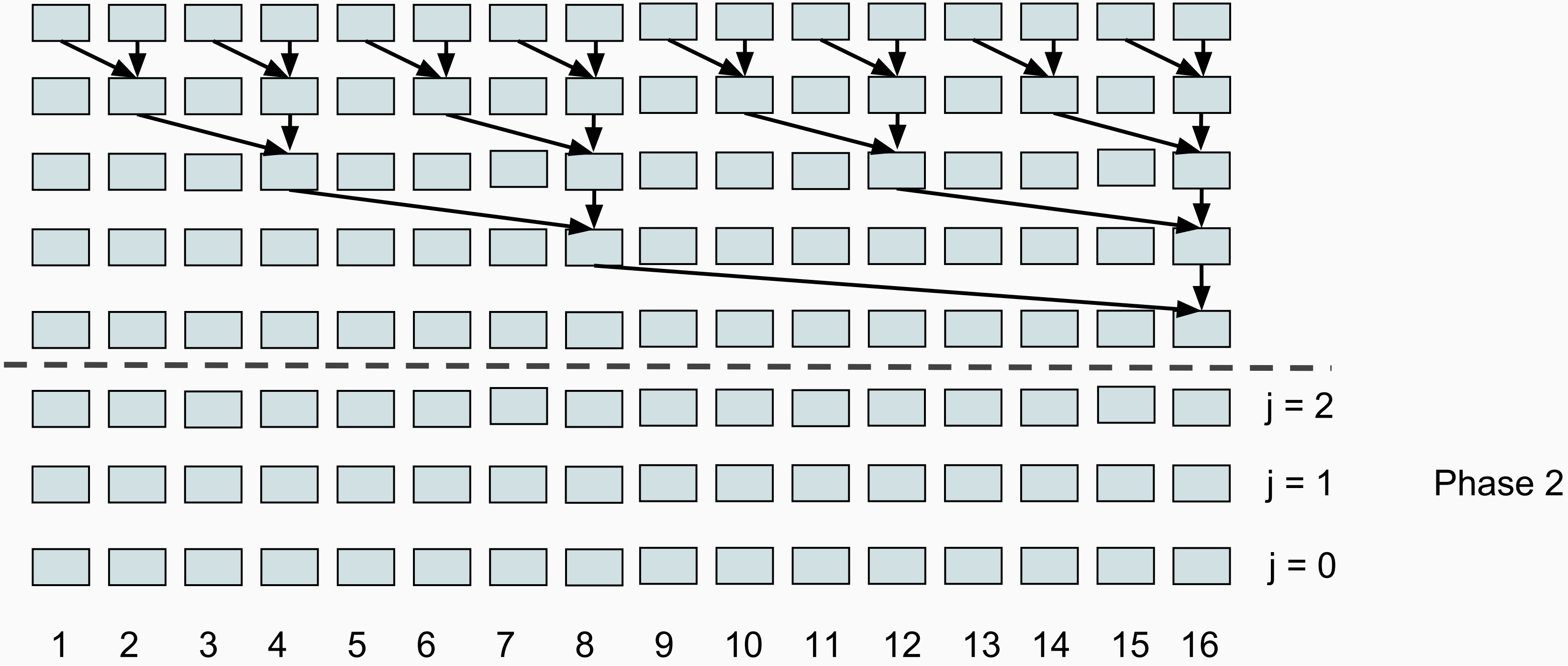
- Need to update only certain values in downward pass
- Which ones?
- Value at (index+1) is updated at step j, if its largest factor which is a power of 2 is 2^j and it is not a power of 2

Consider N = 16

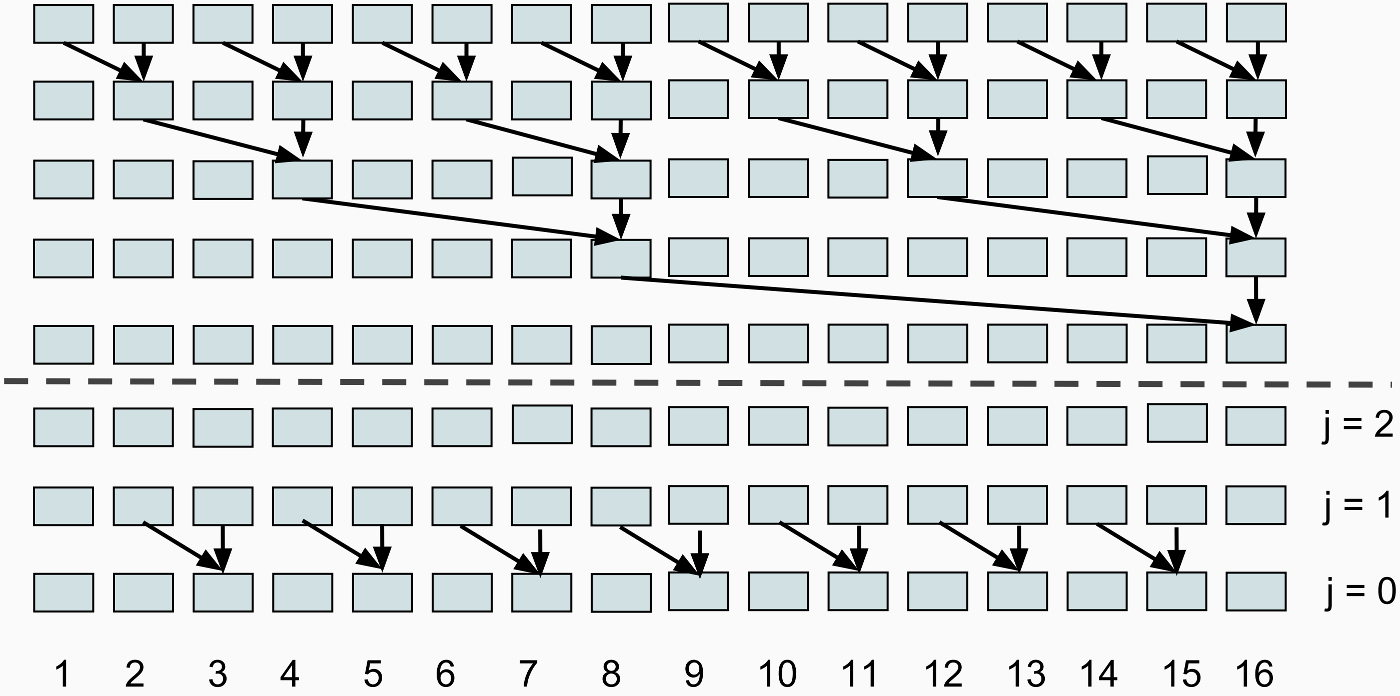


Phase 1: parallel reduce

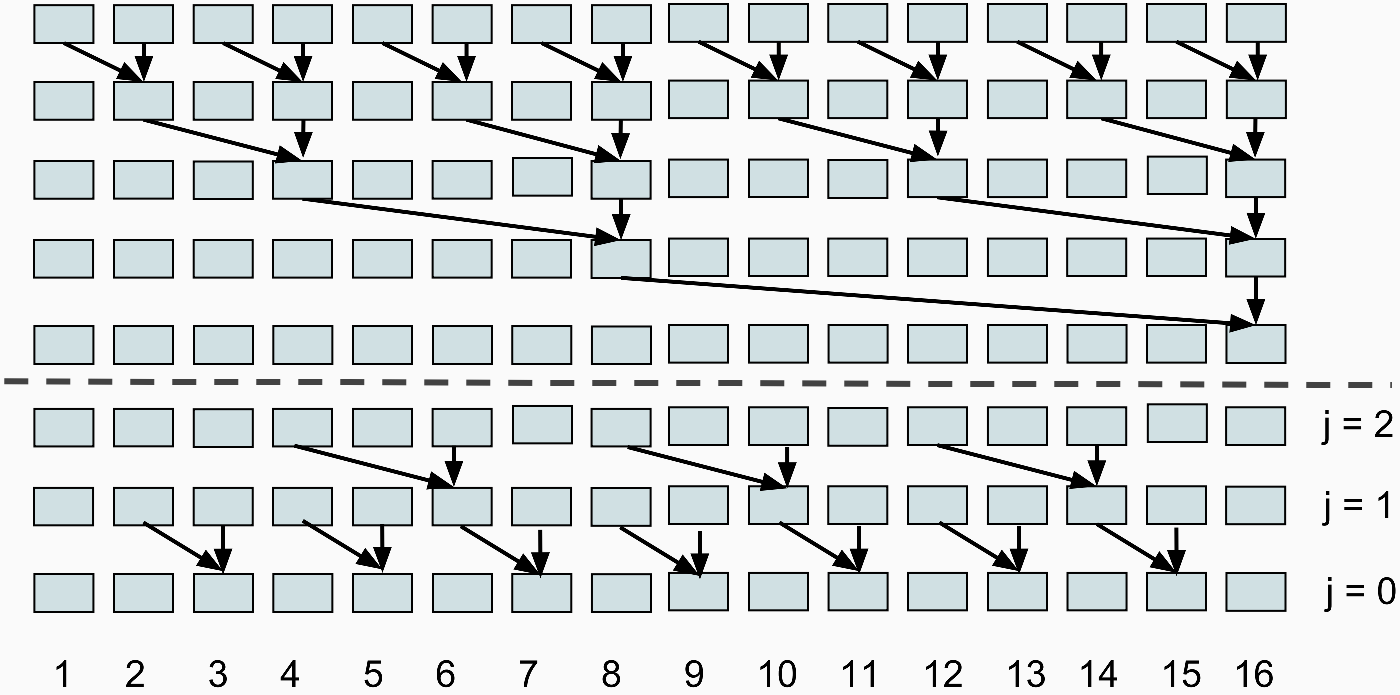
Consider N = 16



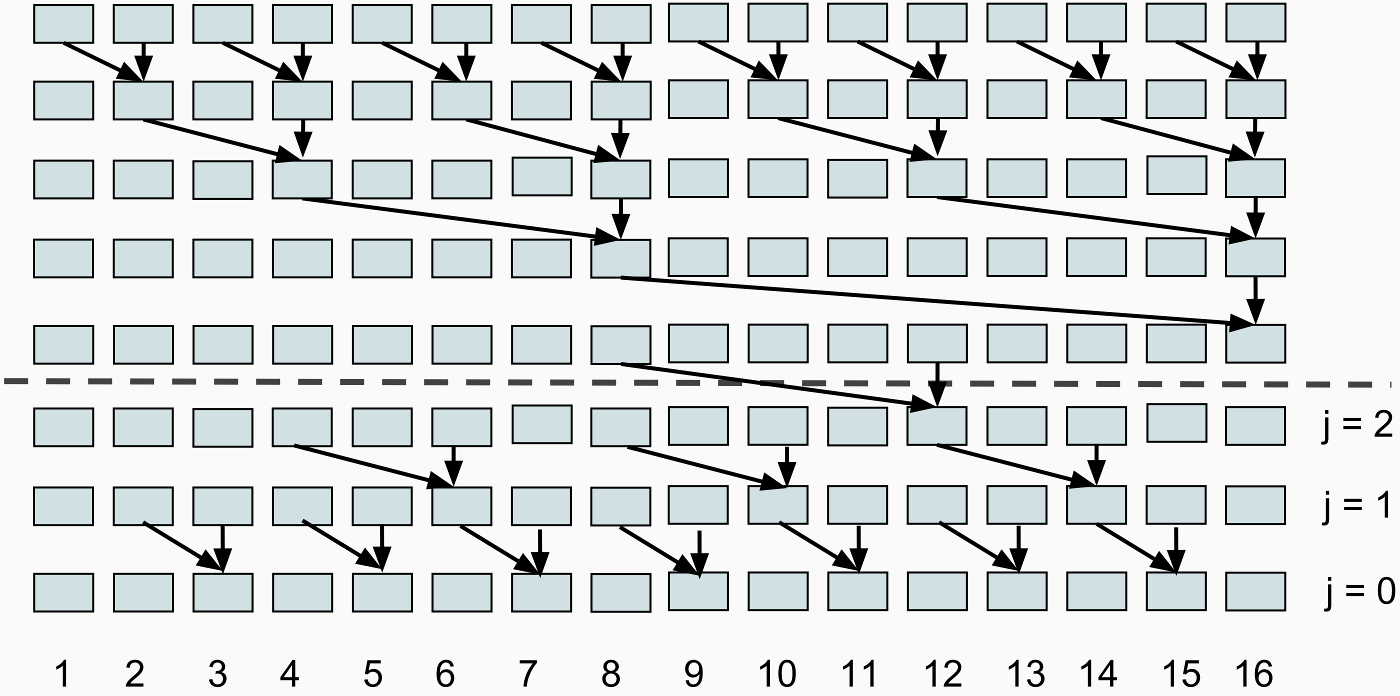
Consider N = 16



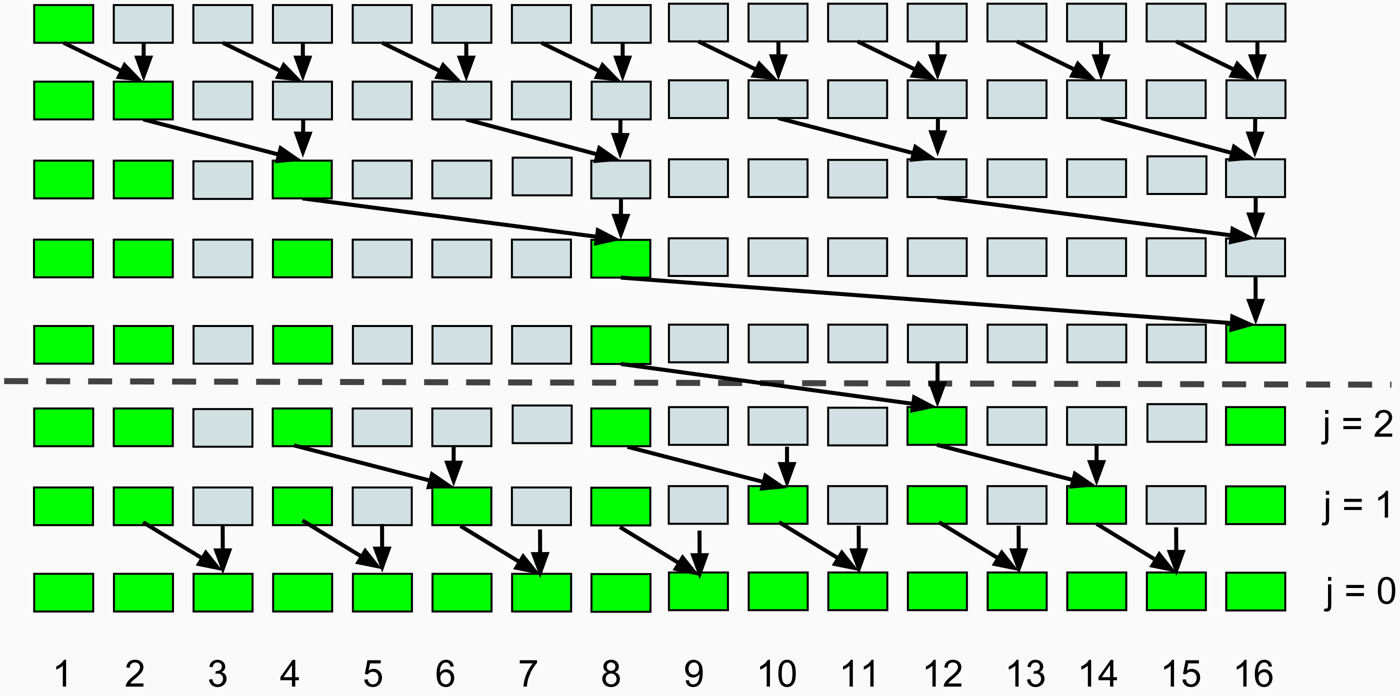
Consider N = 16



Consider N = 16



Consider N = 16



Elegant CUDA code for second phase

```
for (unsigned int stride = N/4; stride > 0; stride /= 2) {
    __syncthreads();
    int t = (threadIdx.x + 1) * stride * 2 - 1;
    if(t + stride < N) {
        partialSum[t + stride] += partialSum[t];
    }
}
```

Complexity

- Complexity:
    - O(log(N)) time
    - O(N) resources
    - Operations?

Complexity

- Complexity:
    - O(log(N)) time
    - O(N) resources
    - Operations
        - Phase 1 parallel reduce is O(N)
        - Phase 2 has log(N)-1 steps
            - Iterations do (2 - 1), (4 - 1), … (N/2 - 1) operations
            - Total = (N - log(N) + 1) -> O(N) operations
        - Total O(N) operations

Is it work efficient

- Recall sequential code had O(N) complexity for scan
- Now, we have O(N) for parallel scan
- But not quite work efficient
    - Phase 1 and 2 can do at most 2x(N - 1) adds
    - Thus, upto twice the number of operations as sequential
- But constant factors do not matter because we are interested in asymptotic trend for highly parallel implementations