# Scan

# What is a Scan?

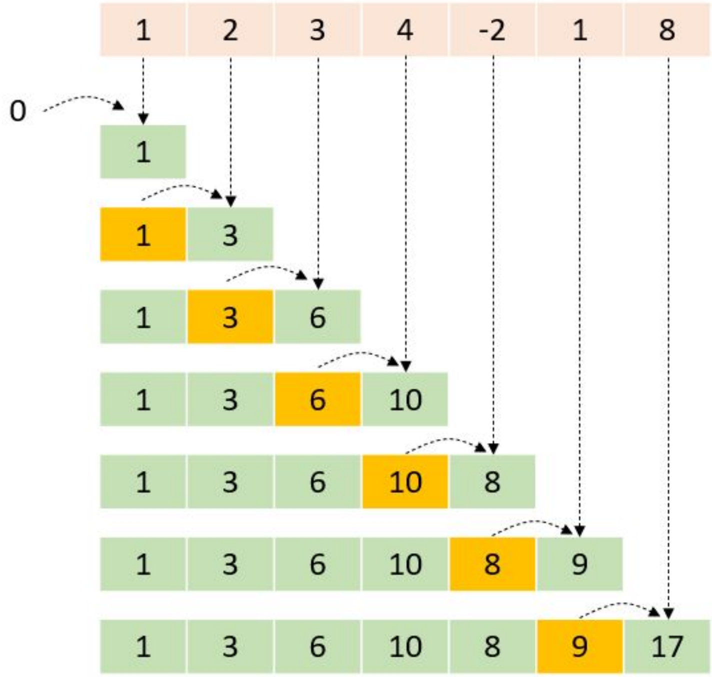
- Takes as input $[x_0,\ x_1,\ \ldots,\ x_{n-1}]$
- Based on the return value, it could be
  - **Inclusive** each element includes the effect of the corresponding input elements
  - **Exclusive** each element does not include the effect of the corresponding input elements

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

    where is i, the identity $[i, x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \oplus \ldots \oplus x_{n-2})]$
- Also called Prefix Sum
- The computational complexity of the sequential algorithm is $O(N)$

# Usage Examples

- Example of where the work performed by some parallel algorithms can have higher complexity
- Used a primitive algorithm for different sorting algorithms
  - Radix sort
  - Quicksort
- Used to perform regex
  - For example the `grep` command



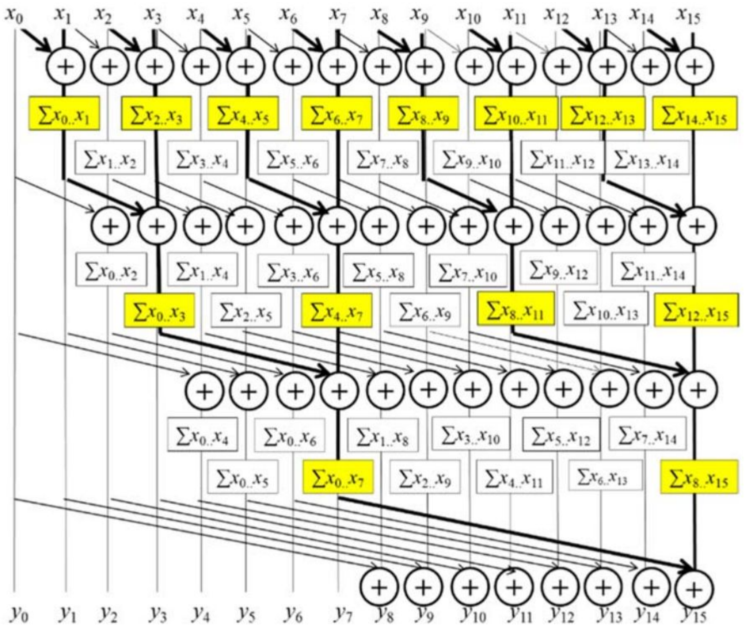
# Naive Implementation

- Assumes that there are as many processors as data element
  - Each threads do the reduction for each output element
- Threads are doing redundant work
  - In the end, the bottleneck will be the longest path
    - The time required to get the last element
- It is not work efficient, the complexity is $O(N^2)$

$$\sum_{i=0}^{n-1} i = \frac{n \cdot (n-1)}{2}$$

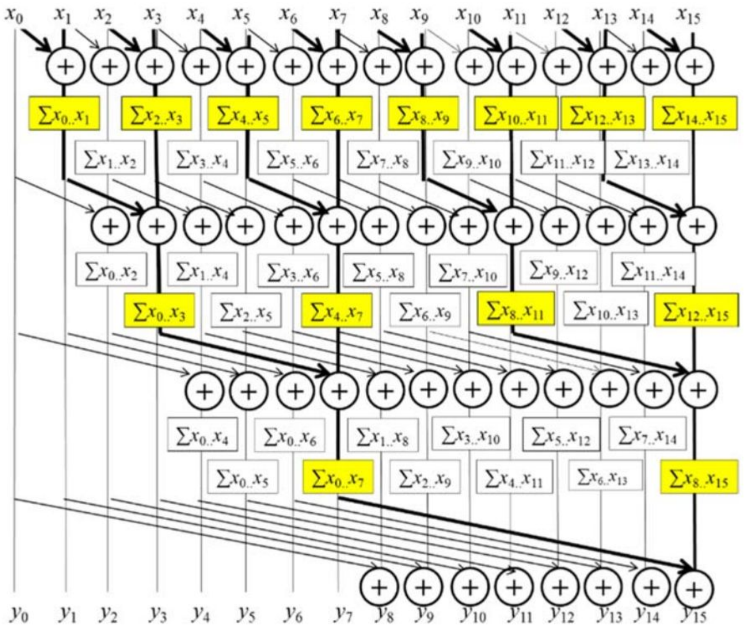
# Kogge-Stone

- Before the algorithm begins, the output[i] contains the input element $x_i$
- After k iterations, output[i] will contain the sum of the $2^k$ input elements before
- Write-after read hazards
  - Two `__syncthreads` are required
- The second `__syncthreads`
  - ensures that all active threads have completed their read of the old values
  - before any of them can move forward and perform a write.



# Kogge-Stone: Double Buffering

- Write-after read hazards
- To avoid the second `__synchthreads` is to use double buffering
  - Each iteration you can swap the two
- It has a complexity of $O(N \log N)$



# Work Efficiency

- It measures the extent to which the work performed is close to the minimum amount of work needed for the computation
- For example
  - The Kogge-Stone algorithm is not work efficient as the sequential one
  - The same applies to the naive one, which is also worse than the Kogge-Stone
- Kogge-Stone algorithm performs more computations
  - But it does so in fewer steps because of parallel execution
- With unlimited execution resources, the reduction would be approximately

  $$N/\log(N)$$

  For $N=512$ would be $512/9=56.93x$
- 

# Code Hands-on

# Bonus

- The Brent-Kung algorithm is another approach to this problem
- Performance-guided optimization in depth can be found here
- It is available in the Thrust Library
  - Documentation here

# Thank you for your attention!

**Gianmarco Accordi**
*gianmarco.accordi@polimi.it*