MI3.22
Advanced Programming for HPC
Master ICT, USTH, 2nd year

Aveneau Lilian

lilian.aveneau@univ-poitiers.fr
XLIM/SIC/IG, CNRS, Computer Science Department
University of Poitiers

Year 2018/2019

Lecture 3 – Scan & other Patterns

- Scan
- Segmented Scan
- Others Parallel Patterns

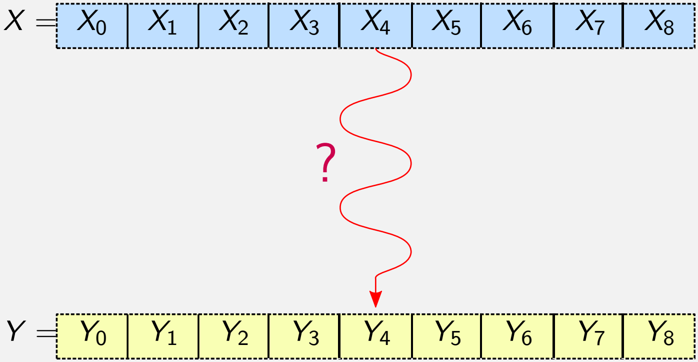
Overview

- Scan
- Segmented Scan
- Others Parallel Patterns

SCAN

It consists to apply a binary, associative and commutative operator to all elements of a given input X of size n, to compute a list of new values:

$$\text{SCAN}(X) = \{Y_i\}_{i=0}^{n-1} = \left\{ \bigoplus_{k=0}^{i} X_k \right\}_{i=0}^{n-1} \qquad \text{where } \bigoplus_{k=i}^{i} = X_i$$

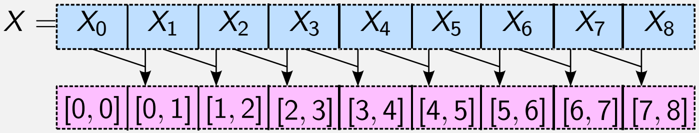


$X =$
X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
?
$Y =$
Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7 Y_8

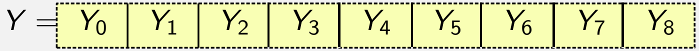
SCAN

It consists to apply a binary, associative and commutative operator to all elements of a given input X of size n, to compute a list of new values:

$$\text{SCAN}(X) = \{Y_i\}_{i=0}^{n-1} = \left\{\bigoplus_{k=0}^{i} X_k\right\}_{i=0}^{n-1} \qquad \text{where } \bigoplus_{k=i}^{i} = X_i$$



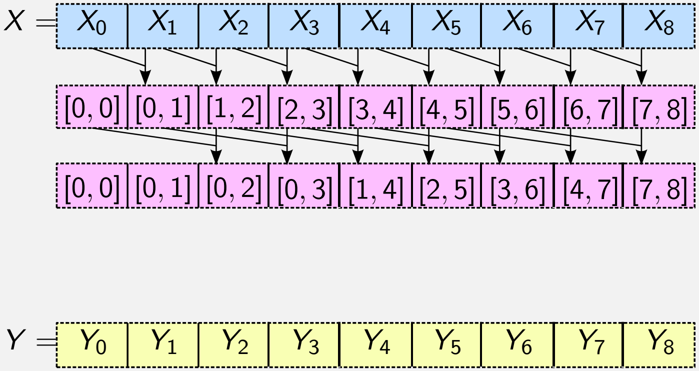
$$\text{with } [i,j] = X_i \oplus X_{i+1} \oplus \ldots \oplus X_j$$



SCAN

It consists to apply a binary, associative and commutative operator to all elements of a given input X of size n, to compute a list of new values:

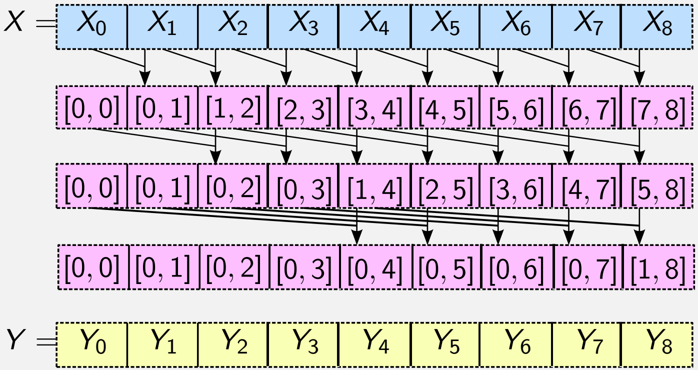
$$\mathrm{SCAN}(X) = \{Y_i\}_{i=0}^{n-1} = \left\{ \bigoplus_{k=0}^{i} X_k \right\}_{i=0}^{n-1} \qquad \text{where } \bigoplus_{k=i}^{i} = X_i$$



SCAN

It consists to apply a binary, associative and commutative operator to all elements of a given input X of size n, to compute a list of new values:

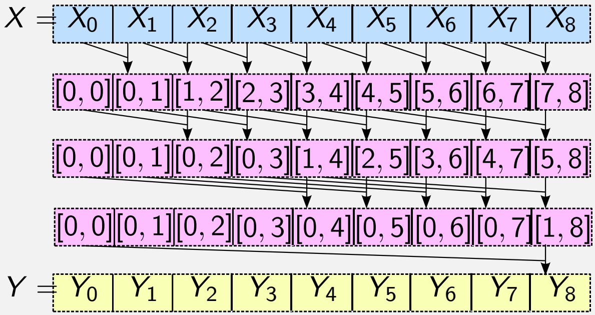
$$\text{SCAN}(X) = \{Y_i\}_{i=0}^{n-1} = \left\{ \bigoplus_{k=0}^{i} X_k \right\}_{i=0}^{n-1} \qquad \text{where } \bigoplus_{k=i}^{i} = X_i$$



SCAN

It consists to apply a binary, associative and commutative operator to all elements of a given input X of size n, to compute a list of new values:

$$\text{SCAN}(X) = \{Y_i\}_{i=0}^{n-1} = \left\{ \bigoplus_{k=0}^{i} X_k \right\}_{i=0}^{n-1} \qquad \text{where } \bigoplus_{k=i}^{i} = X_i$$



- The swiss-knife of any parallel programmer!
- In thrust: `thrust::inclusive_scan` with different possibilities

PREFIX-SCAN

It consists to apply a binary associative commutative operator to all elements of a given input X of size n, to compute a list of new values:

$$\text{PREFIX-SCAN}(X) = \left\{ \bigoplus_{k=0}^{i-1} X_k \right\}_{i=0}^{n-1}$$

where $\forall j < i$, $\bigoplus_i^j = 0$ or any other initialization value

- A *shifted* SCAN, in fact!
  Example with $\oplus = +$

X_i	1	1	1	1	1	1	1	1
SCAN	1	2	3	4	5	6	7	8
PRESCAN	0	1	2	3	4	5	6	7

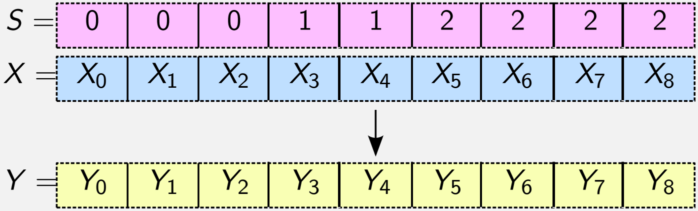
- It is also the swiss-knife of any parallel programmer!
- In thrust: `thrust:exclusive_scan` with different possibilities

Overview

- Scan
- Segmented Scan
- Others Parallel Patterns

SEGMENTED SCAN I

It is a SCAN or a PREFIX-SCAN done per segment of a given array …
Example:


$S =$ 0 0 0 1 1 2 2 2 2
$X =$ X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
$Y =$ Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7 Y_8

where S defines the segments, and so:

- $Y_0 = X_0$, $Y_1 = X_0 \oplus X_1$, and $Y_2 = X_0 \oplus X_1 \oplus X_2$
- $Y_3 = X_3$, $Y_4 = X_3 \oplus X_4$
- $Y_5 = X_5$, $Y_6 = X_5 \oplus X_6$, $Y_7 = X_5 \oplus X_6 \oplus X_7$ and $Y_8 = X_5 \oplus X_6 \oplus X_7 \oplus X_8$

SEGMENTED SCAN II

Obviously, for PREFIX-SEG-SCAN, we have

- $Y_0 = 0,\ Y_1 = X_0$, and $Y_2 = X_0 \oplus X_1$
- $Y_3 = 0,\ Y_4 = X_3$
- $Y_5 = 0,\ Y_6 = X_5,\ Y_7 = X_5 \oplus X_6$ and $Y_8 = X_5 \oplus X_6 \oplus X_7$

Notice that it is a SCAN (or PREFIX-SCAN) using a particular operator:

$$\otimes : \{\mathbb{N} \times T\}^2 \rightarrow T$$
$$\otimes(\{S_i, X_i\}, \{S_j, X_j\}) = \begin{cases} X_i \oplus X_j & \text{if } S_i = S_j, \\ X_j & \text{otherwise.} \end{cases}$$

In practice, computation cost is a little higher:

- Need to read more data (the segment id)
- Avoid it when possible (prefer SCAN)

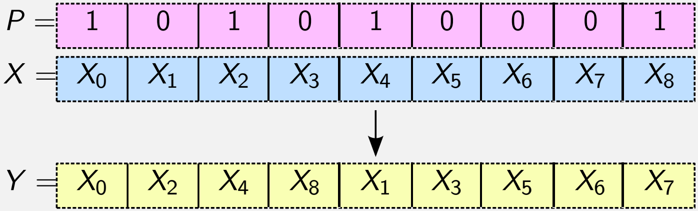
With Thrust, implemented in `thrust::inclusive_scan_by_key` and `thrust::exclusive_scan_by_key`

Overview

- Scan
- Segmented Scan
- Others Parallel Patterns

SPLIT or PARTITION

Allows to separate an array into two segments using a given predicate:

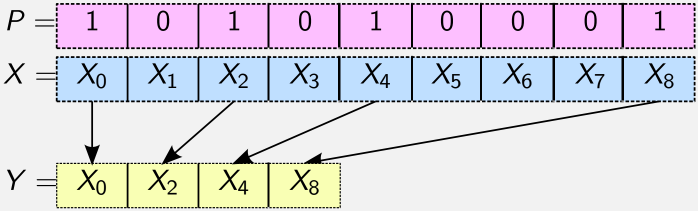

$P =$ 1 0 1 0 1 0 0 0 1
$X =$ X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
$Y =$ X_0 X_2 X_4 X_8 X_1 X_3 X_5 X_6 X_7

Using Thrust, you can use `thrust::partition` and variants ...
Notice that it can be written using others patterns:

```
PARTITION($X$, $P$)
    { $X$ and $P$ are arrays of size $N$, $X$ being data and $P_i = \{0, 1\}$ a boolean array }
    $Y$, $Z$: arrays of size $N$
    FOR each PE $i$ in parallel
        $Y_i \leftarrow P_i$
        $Z_i \leftarrow P_i - 1$ { so contains 0 or $-1$ }
    $Y \leftarrow$ PREFIX−SCAN($Y$)
    $Z \leftarrow$ REVERSE−SCAN($Z$) { scan in reverse order }
    FOR each PE $i$ in parallel
        IF NOT $P_i$ THEN $Y_i \leftarrow N + Z_i$
    return SCATTER($X$, $Y$)
```

COMPACT

COMPACT is similar to PARTITION, but forgetting the second part (for which predicate $P_i = 0$) ...



$P =$ 1 0 1 0 1 0 0 0 1
$X =$ X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
$Y =$ X_0 X_2 X_4 X_8

- Here, destination is not completely modified ...
- Result has variable length!

With Thrust, take a look at *stream compaction* at
`http://thrust.github.io/doc/group__stream__compaction.html`

SORT

SORT is a generic pattern for sorting data, given an order

- Many algorithms exist, for parallel computers
- An old one: RICHARD COLE's sorting machine 1986, "Parallel Merge Sort"
- Bitonic sort
- Parallel Quicksort
- Probably the most efficient: radix sort (see labwork)

In thrust, you can use `thrust::sort` which implements a fast radix-sort with bitonic sort into blocks to accelerate the whole process ...