

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 2 – Pointer jumping & Reduce

- Pointer Jumping
- Reduce

Overview

- Pointer Jumping
- Reduce

List Ranking

Input: linked list of size n .

Output: for each element i the distance $d[i]$ to end of the list:

$$d[i] = \begin{cases} 0 & \text{if next}[i] = \text{NIL} \\ d[\text{next}[i]] + 1 & \text{if next}[i] \neq \text{NIL} \end{cases}$$

Sequential complexity : $O(n)$

With PRAM: complexity $O(\log(n))$ – base 2 –

Associate one PE to each list element i ...

```

1 RANK_LIST(L)
2 {Initialization}
3 FOR each PE  $i$  in parallel:
4   IF next[ $i$ ] = NIL THEN  $d[i] \leftarrow 0$  ELSE  $d[i] \leftarrow 1$ 
5   { Main loop }
6   WHILE exists a node  $i$  such that next[ $i$ ]  $\neq$  NIL:
7     FOR each PE  $i$  in parallel { with synchronized access }
8       IF next[ $i$ ]  $\neq$  NIL THEN
9          $d[i] \leftarrow d[i] + d[\text{next}[i]]$  {Each PE read, THEN write}
10        next[ $i$ ]  $\leftarrow$  next[next[ $i$ ]] {idem}

```

- CRCW: write in a boolean “ended” + fusion

Line 6 :

- CREW: $O(\log(n))$! Better, with loop FOR $s=1$ TO $\lceil \log n \rceil$

- EREW: idem, writing into temporary variable

SCAN

Let \oplus be a binary associative operation ...

Input: sequence (x_1, x_2, \dots, x_n) known as a linked list

Output: sequence (y_1, y_2, \dots, y_n) where

$$y_k = y_{k-1} \oplus x_k = x_1 \oplus x_2 \oplus \dots \oplus x_k$$

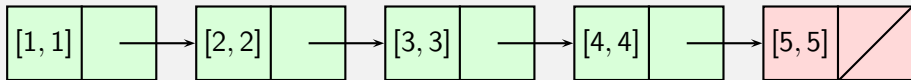
PRAM Algorithm

```

1 SCAN(L)
2 {Initialization}
3 FOR each PE  $i$  in parallel:
4    $y[i] \leftarrow x[i]$ 
5 {Main loop}
6 WHILE exists node  $i$  such that  $\text{next}[i] \neq \text{NIL}$ : {Or FOR loop ...}
7   FOR each PE  $i$  in parallel
8     IF  $\text{next}[i] \neq \text{NIL}$  THEN
9        $y[\text{next}[i]] \leftarrow y[i] \oplus y[\text{next}[i]]$ 
10       $\text{next}[i] \leftarrow \text{next}[\text{next}[i]]$ 

```

$[i, j]$ denotes $x_i \oplus x_{i+1} \oplus \dots \oplus x_j$ for $i \leq j$... Example:



SCAN

Let \oplus be a binary associative operation ...

Input: sequence (x_1, x_2, \dots, x_n) known as a linked list

Output: sequence (y_1, y_2, \dots, y_n) where

$$y_k = y_{k-1} \oplus x_k = x_1 \oplus x_2 \oplus \dots \oplus x_k$$

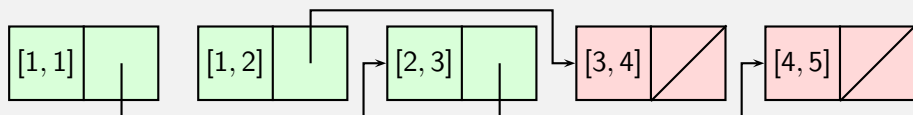
PRAM Algorithm

```

1 SCAN(L)
2 {Initialization}
3 FOR each PE  $i$  in parallel:
4    $y[i] \leftarrow x[i]$ 
5 {Main loop}
6 WHILE exists node  $i$  such that  $\text{next}[i] \neq \text{NIL}$ : {Or FOR loop ...}
7   FOR each PE  $i$  in parallel
8     IF  $\text{next}[i] \neq \text{NIL}$  THEN
9        $y[\text{next}[i]] \leftarrow y[i] \oplus y[\text{next}[i]]$ 
10       $\text{next}[i] \leftarrow \text{next}[\text{next}[i]]$ 

```

$[i, j]$ denotes $x_i \oplus x_{i+1} \oplus \dots \oplus x_j$ for $i \leq j$... Example:



SCAN

Let \oplus be a binary associative operation ...

Input: sequence (x_1, x_2, \dots, x_n) known as a linked list

Output: sequence (y_1, y_2, \dots, y_n) where

$$y_k = y_{k-1} \oplus x_k = x_1 \oplus x_2 \oplus \dots \oplus x_k$$

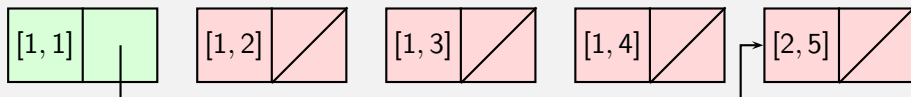
PRAM Algorithm

```

1 SCAN(L)
2 {Initialization}
3 FOR each PE  $i$  in parallel:
4    $y[i] \leftarrow x[i]$ 
5 {Main loop}
6 WHILE exists node  $i$  such that  $\text{next}[i] \neq \text{NIL}$ : {Or FOR loop ...}
7   FOR each PE  $i$  in parallel
8     IF  $\text{next}[i] \neq \text{NIL}$  THEN
9        $y[\text{next}[i]] \leftarrow y[i] \oplus y[\text{next}[i]]$ 
10       $\text{next}[i] \leftarrow \text{next}[\text{next}[i]]$ 

```

$[i, j]$ denotes $x_i \oplus x_{i+1} \oplus \dots \oplus x_j$ for $i \leq j$... Example:



SCAN

Let \oplus be a binary associative operation ...

Input: sequence (x_1, x_2, \dots, x_n) known as a linked list

Output: sequence (y_1, y_2, \dots, y_n) where

$$y_k = y_{k-1} \oplus x_k = x_1 \oplus x_2 \oplus \dots \oplus x_k$$

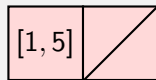
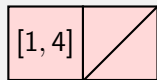
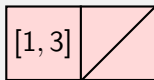
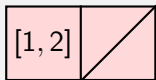
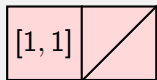
PRAM Algorithm

```

1 SCAN(L)
2 {Initialization}
3 FOR each PE  $i$  in parallel:
4    $y[i] \leftarrow x[i]$ 
5 {Main loop}
6 WHILE exists node  $i$  such that  $\text{next}[i] \neq \text{NIL}$ : {Or FOR loop ...}
7   FOR each PE  $i$  in parallel
8     IF  $\text{next}[i] \neq \text{NIL}$  THEN
9        $y[\text{next}[i]] \leftarrow y[i] \oplus y[\text{next}[i]]$ 
10       $\text{next}[i] \leftarrow \text{next}[\text{next}[i]]$ 

```

$[i, j]$ denotes $x_i \oplus x_{i+1} \oplus \dots \oplus x_j$ for $i \leq j$... Example:



Euler Tower

Input: binary tree with n nodes (leaves included)

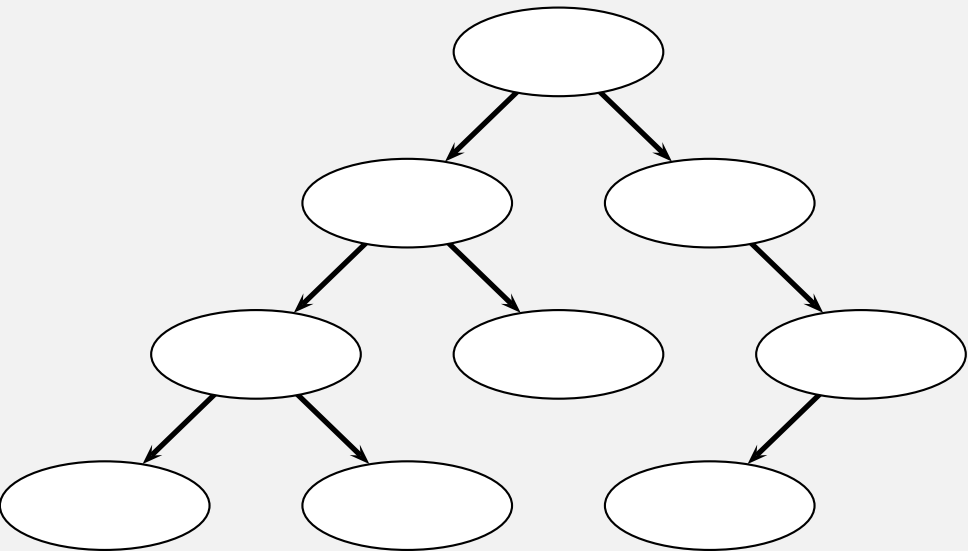
Output: depth of each node

Naive algorithm: breadth-first, with $O(d)$ steps where d is tree depth

$O(\log(n))$ algorithm \forall trees

- Associate 3 PE (A , B , C) to each node, scan or prefix-scan
- Binary operation: addition in \mathbb{Z}
- Linking:
 - A on left child A if it exists, B else
 - B on right child A if it exists, C else
 - C on B –if left– or C –if right– of father (or NIL if root)
- Initialization :
 - $A = 1$, to go down
 - $B = 0$, to go right
 - $C = -1$, to go up

Euler Tower Example



Euler Tower Example

Set the PEs

$$\begin{array}{l} A = 1 \quad C = -1 \\ B = 0 \end{array}$$

$$\begin{array}{l} A = 1 \quad C = -1 \\ B = 0 \end{array}$$

$$\begin{array}{l} A = 1 \quad C = -1 \\ B = 0 \end{array}$$

$$\begin{array}{l} A = 1 \quad C = -1 \\ B = 0 \end{array}$$

$$\begin{array}{l} A = 1 \quad C = -1 \\ B = 0 \end{array}$$

$$\begin{array}{l} A = 1 \quad C = -1 \\ B = 0 \end{array}$$

$$\begin{array}{l} A = 1 \quad C = -1 \\ B = 0 \end{array}$$

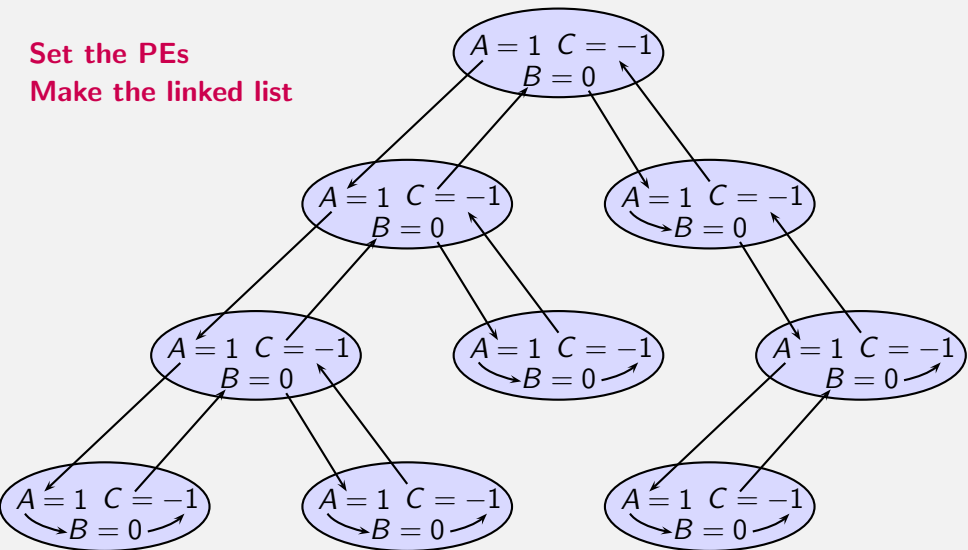
$$\begin{array}{l} A = 1 \quad C = -1 \\ B = 0 \end{array}$$

$$\begin{array}{l} A = 1 \quad C = -1 \\ B = 0 \end{array}$$

Euler Tower Example

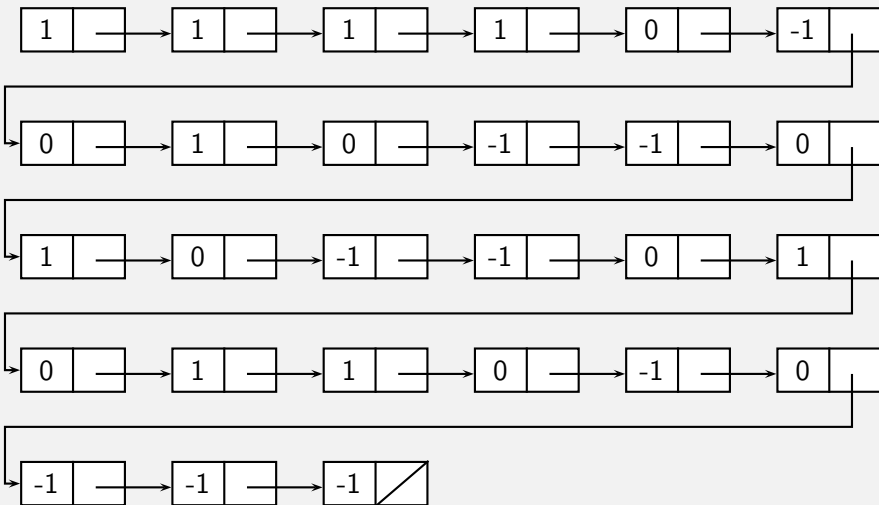
Set the PEs

Make the linked list



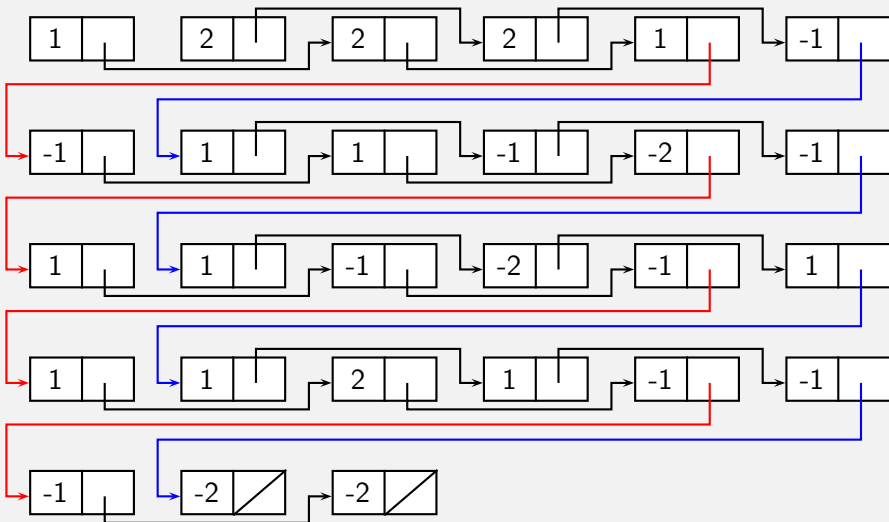
Euler Tower Example

Print as a list: loop 0



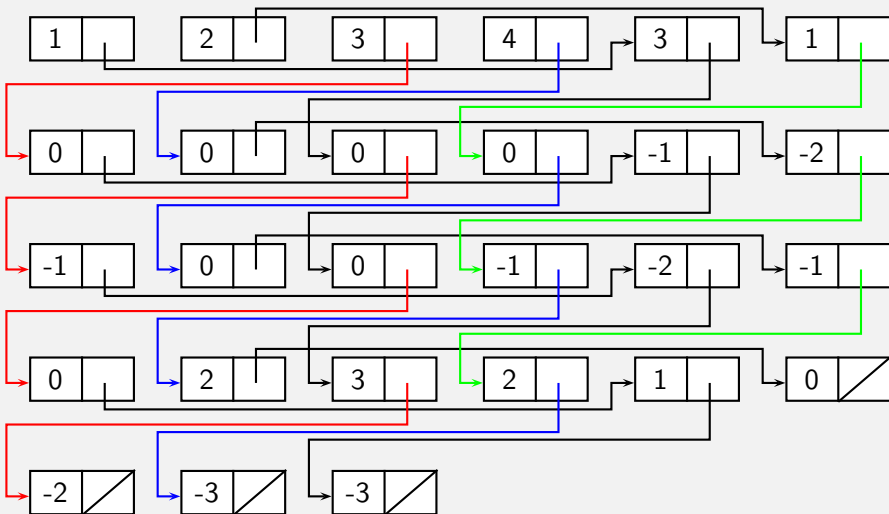
Euler Tower Example

Print as a list: loop 1



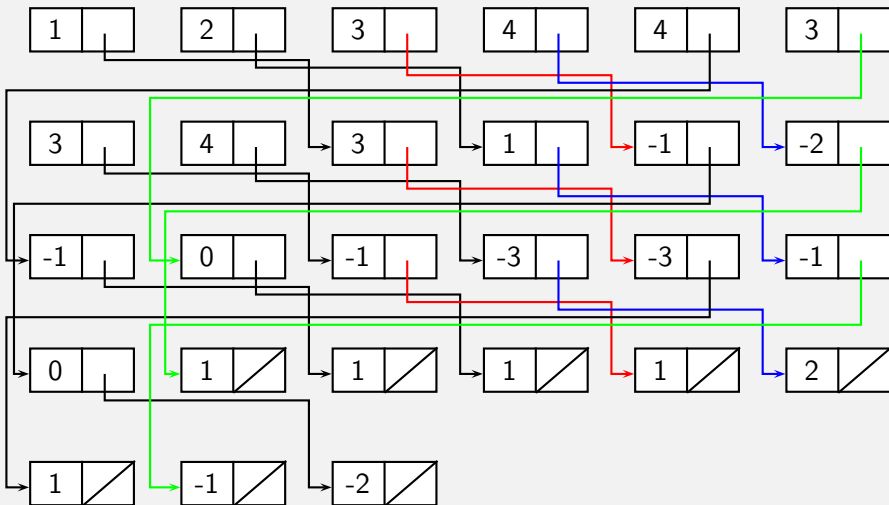
Euler Tower Example

Print as a list: loop 2



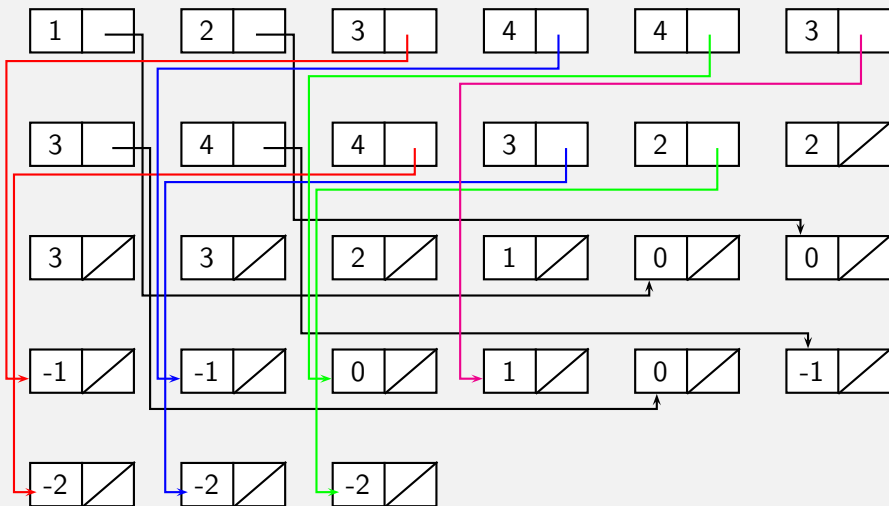
Euler Tower Example

Print as a list: loop 3



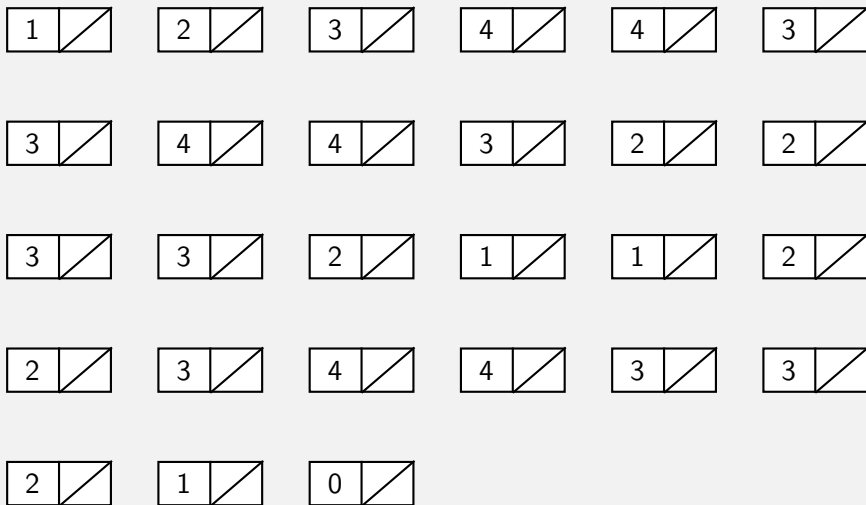
Euler Tower Example

Print as a list: loop 4



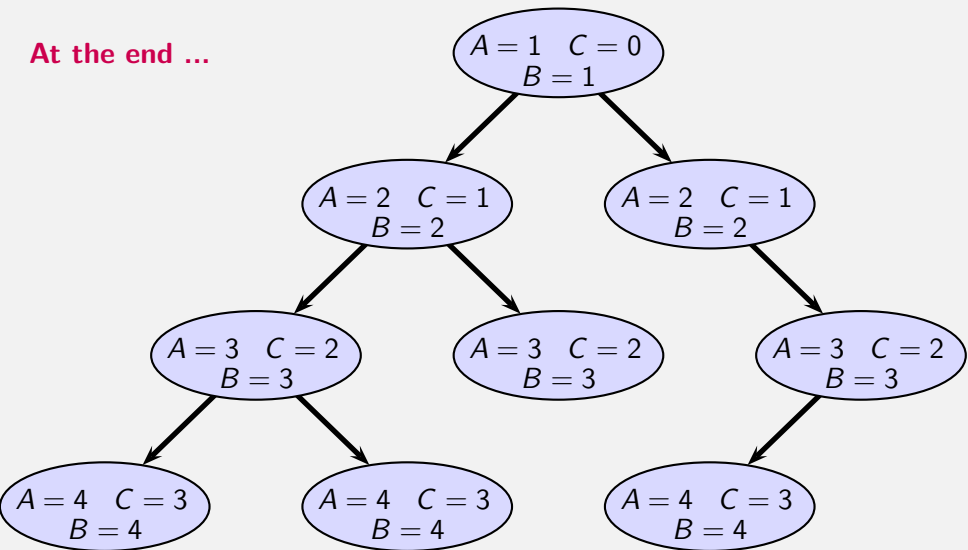
Euler Tower Example

Print as a list: loop 5



Euler Tower Example

At the end ...



Overview

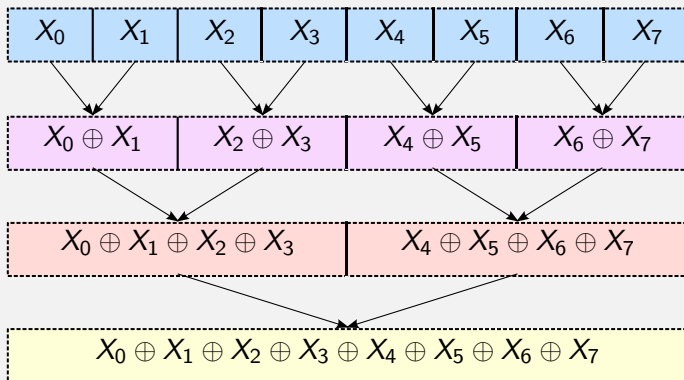
- Pointer Jumping
- Reduce

REDUCE

REDUCE consists to **apply a binary associative commutative operator to all the elements** of a given input of size n :

$$\bigoplus_{i=0}^{n-1} X_i$$

You have already seen the parallel version of this computation:



REDUCE using PRAM

Many solutions exists! The following corresponds to the figure displayed on previous slide ...

```

1 REDUCE( $X, N, op$ )
2   {  $X$  is an array of size  $N$ ,  $op$  a binary operator }
3   jump  $\leftarrow$  1
4   WHILE jump  $<$   $N$  DO:
5     FOR each PE  $i$  in parallel DO:
6       IF ( $i \bmod 2 \times \text{jump}$ )  $=$  0 THEN
7         next  $\leftarrow X_{i+\text{jump}}$ 
8         { implicit barrier }
9          $X_i \leftarrow op(X_i, \text{next})$ 
10      END-IF
11    END-FOR
12    jump  $\leftarrow 2 \times \text{jump}$ 
13  END-WHILE
14  RETURN  $X_0$ 
15 END { REDUCE }

```

Thrust

We have two versions!

Classical REDUCE

```
template<typename InputIterator, typename T, typename BinaryFunction >
T thrust::reduce ( InputIterator first,      // data iterator
                  InputIterator last,       // end of data
                  T init,                   // 0 by default
                  BinaryFunction binary_op   // thrust::plus<T> by default
                ) ;
```

Thrust

We have two versions!

Classical REDUCE

```
template<typename InputIterator, typename T, typename BinaryFunction >
T thrust::reduce ( InputIterator first,      // data iterator
                  InputIterator last,       // end of data
                  T init,                  // 0 by default
                  BinaryFunction binary_op  // thrust::plus<T> by default
                ) ;
```

SEGMENTED-REDUCE (or reduce-by-key)

```
template<typename InputIterator1, typename InputIterator2,
         typename OutputIterator1, typename OutputIterator2,
         typename BinaryPredicate, typename BinaryFunction>
thrust::pair<OutputIterator1, OutputIterator2> thrust::reduce_by_key
( InputIterator1 keys_first,
  InputIterator1 keys_last,
  InputIterator2 values_first,
  OutputIterator1 keys_output,
  OutputIterator2 values_output,
  BinaryPredicate binary_predicate,
  BinaryFunction binary_function
) ;
```


Reduce and reduce-by-key example

```

#include <thrust/reduce.h>
// This example runs on HOST ...
int main() {
    const int N = 10;

    int iV[N] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }; // input values

    thrust::plus<int> binary_op;
    int sum = thrust::reduce(iV, iV+N, 0, binary_op);
    // sum == (10*11)/2 == 55 ...

    int iK[N] = {0, 0, 0, 0, 1, 1, 1, 1, 2, 2 }; // input keys
    int oK[N]; // output keys
    int oV[N]; // output values
    thrust::equal_to<int> binary_pred; // for keys

    thrust::pair<int*,int*> new_end =
        thrust::reduce_by_key(iK, iK + N, iV, oK, oV, binary_pred, binary_op);
    // Now, (new_end.first - oK) == (new_end.second - oV) == 3 !
    // The first three keys in oK are now {0, 1, 2}
    // The first three values in oV are now {10, 26, 19}
    return 0;
}

```

REDUCE-BY-KEY using PRAM

Need to consider the Keys: when they differ, cannot do the binary operation; equality means the operation is valid

```

1 REDUCE-BY-KEY( $X, K, N, op$ )
2   {  $X$  and  $K$  are arrays of size  $N$ ,  $op$  a binary operator }
3   jump  $\leftarrow$  1
4   WHILE jump  $< N$  DO:
5     FOR each PE  $i$  in parallel DO:
6       IF ( $i \bmod 2 \times \text{jump}$ ) = 0 THEN
7         next  $\leftarrow X_{i+\text{jump}}$ 
8         { implicit barrier }
9         IF  $K_i = K_{\text{next}}$  THEN
10           $X_i \leftarrow op(X_i, \text{next})$ 
11        END-IF
12      END-IF
13    END-FOR
14    jump  $\leftarrow 2 \times \text{jump}$ 
15  END-WHILE
16 END { REDUCE-BY-KEY }

```