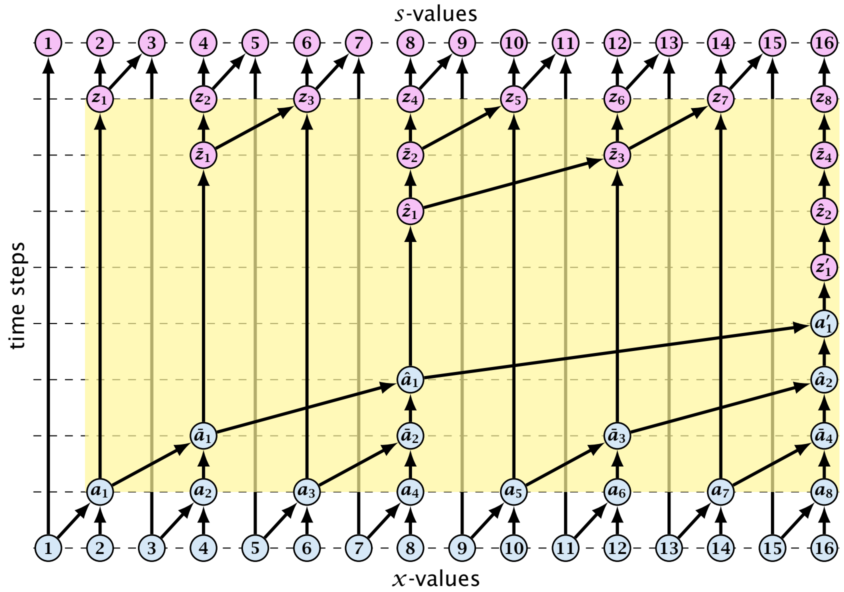
Prefix Sum

input: $x[1]\ldots x[n]$
output: $s[1]\ldots s[n]$ with $s[i] = \sum_{j=1}^{i} x[i]$ (w.r.t. operator $*$)

Algorithm 6 PrefixSum$(n, x[1]\ldots x[n])$

// compute prefixsums; $n = 2^k$
if $n = 1$ **then** $s[1] \leftarrow x[1]$; **return**
for $1 \le i \le n/2$ **pardo**
    $a[i] \leftarrow x[2i-1] * x[2i]$
$z[1], \ldots, z[n/2] \leftarrow$ PrefixSum$(n/2, a[1]\ldots a[n/2])$
for $1 \le i \le n$ **pardo**
    i even : $s[i] \leftarrow z[i/2]$
    $i = 1$: $s[1] = x[1]$
    i odd : $s[i] \leftarrow z[(i-1)/2] * x[i]$

Prefix Sum



Prefix Sum

The algorithm uses work $\mathcal{O}(n)$ and time $\mathcal{O}(\log n)$ for solving Prefix Sum on an EREW-PRAM with n processors.

It is clearly work-optimal.

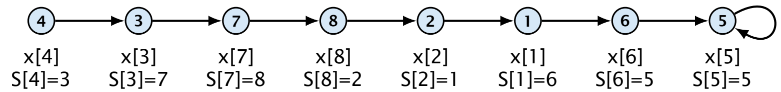
Theorem 1

On a CREW PRAM a Prefix Sum requires running time $\Omega(\log n)$ regardless of the number of processors.

Parallel Prefix

Input: a linked list given by successor pointers; a value $x[i]$ for every list element; an operator $*$;

Output: for every list position ℓ the sum (w.r.t. $*$) of elements after ℓ in the list (including ℓ)



Parallel Prefix

Algorithm 7 ParallelPrefix

for $1 \le i \le n$ **pardo**
  $P[i] \leftarrow S[i]$
  while $S[i] \neq S[S[i]]$ **do**
    $x[i] \leftarrow x[i] * x[S[i]]$
    $S[i] \leftarrow S[S[i]]$
  if $P[i] \neq i$ **then** $S[i] \leftarrow x[S(i)]$

The algorithm runs in time $\mathcal{O}(\log n)$.

It has work requirement $\mathcal{O}(n \log n)$. non-optimal

This technique is also known as pointer jumping

4.3 Divide & Conquer — Merging

Given two sorted sequences $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_n)$, compute the sorted sequence $C = (c_1, \dots, c_n)$.

Definition 2

Let $X = (x_1, \dots, x_t)$ be a sequence. The rank $\operatorname{rank}(y : X)$ of y in X is

$$\operatorname{rank}(y : X) = |\{x \in X \mid x \le y\}|$$

For a sequence $Y = (y_1, \dots, y_s)$ we define $\operatorname{rank}(Y : X) := (r_1, \dots, r_s)$ with $r_i = \operatorname{rank}(y_i : X)$.

4.3 Divide & Conquer — Merging

Given two sorted sequences $A = (a_1 \dots a_n)$ and $B = (b_1 \dots b_n)$, compute the sorted squence $C = (c_1 \dots c_n)$.

Observation:

We can assume wlog. that elements in A and B are different.

Then for $c_i \in C$ we have $i = \operatorname{rank}(c_i : A \cup B)$.

This means we just need to determine $\operatorname{rank}(x : A \cup B)$ for all elements!

Observe, that $\operatorname{rank}(x : A \cup B) = \operatorname{rank}(x : A) + \operatorname{rank}(x : B)$.

Clearly, for $x \in A$ we already know $\operatorname{rank}(x : A)$, and for $x \in B$ we know $\operatorname{rank}(x : B)$.

4.3 Divide & Conquer — Merging

Compute $\operatorname{rank}(x : A)$ for all $x \in B$ and $\operatorname{rank}(x : B)$ for all $x \in A$.
can be done in $\mathcal{O}(\log n)$ time with $2n$ processors by binary search

Lemma 3

On a CREW PRAM, Merging can be done in $\mathcal{O}(\log n)$ time and $\mathcal{O}(n \log n)$ work.

not optimal