Parallel Prefix Computation

Advanced Algorithms & Data Structures

Lecture Theme 14

Prof. Dr. Th. Ottmann

Summer Semester 2006

Overview

- A simple parallel algorithm for computing parallel prefix.
- A parallel merging algorithm

Definition of prefix computation

- We are given an ordered set *A* of *n* elements and a binary associative operator $\oplus$.

$$A = \{a_0, a_1, a_2, ..., a_{n-1}\}$$

- We have to compute the ordered set

$$\{a_0, (a_0 \oplus a_1), ..., (a_0 \oplus a_1 \oplus ... a_{n-1})\}$$

An example of prefix computation

- For example, if $\oplus$ is + and the input is the ordered set

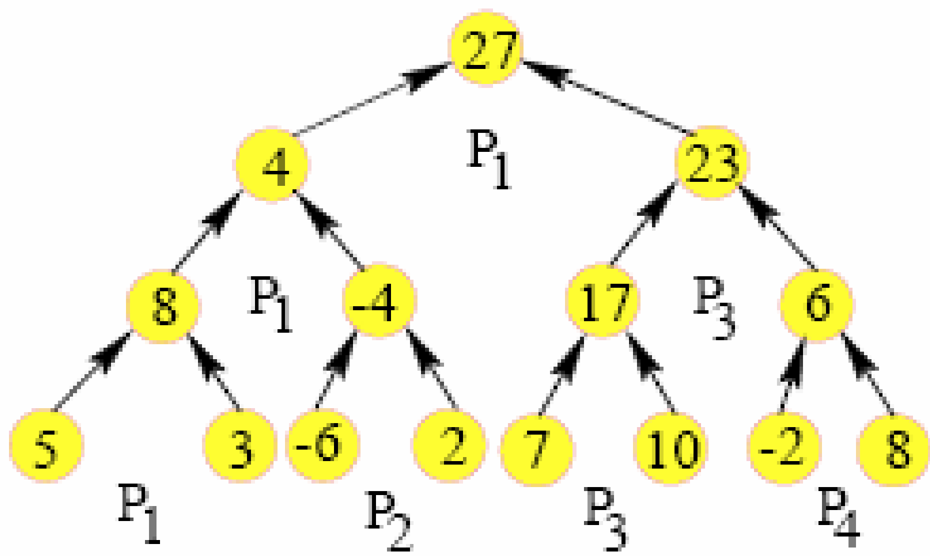
  $$\{5, 3, -6, 2, 7, 10, -2, 8\}$$

  then the output is

  $$\{5, 8, 2, 4, 11, 21, 19, 27\}$$

- Prefix sum can be computed in $O(n)$ time sequentially.

Using a binary tree



First Pass

- For every internal node of the tree, compute the sum of all the leaves in its subtree in a bottom-up fashion.

$$sum[v] := sum[L[v]] + sum[R[v]]$$

Parallel prefix computation

for $d = 0$ to $\log n - 1$ do

for $i = 0$ to $n - 1$ by 2^{d+1} do in parallel

$$a[i + 2^{d+1} - 1] := a[i + 2^{d} - 1] + a[i + 2^{d+1} - 1]$$

- In our example, $n = 8$, hence the outer loop iterates 3 times, $d = 0, 1, 2$.

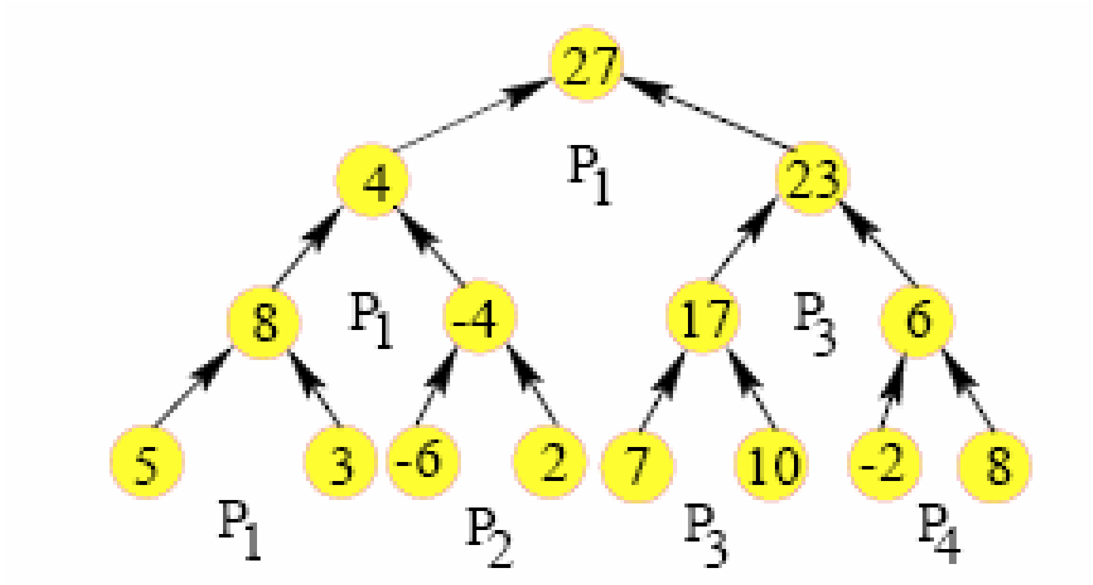
When d= 0

- d = 0: In this case, the increments of 2^{d+1} will be in terms of 2 elements.
- for i = 0,

  $a[0 + 2^{0+1} - 1] := a[0 + 2^{0} - 1] + a[0 + 2^{0+1} - 1]$

  or, $a[1] := a[0] + a[1]$

Using a binary tree



First Pass

- For every internal node of the tree, compute the sum of all the leaves in its subtree in a bottom-up fashion.

$$sum[v] := sum[L[v]] + sum[R[v]]$$

When d = 1

- $d = 1$: In this case, the increments of 2^{d+1} will be in terms of 4 elements.
- for $i = 0$,

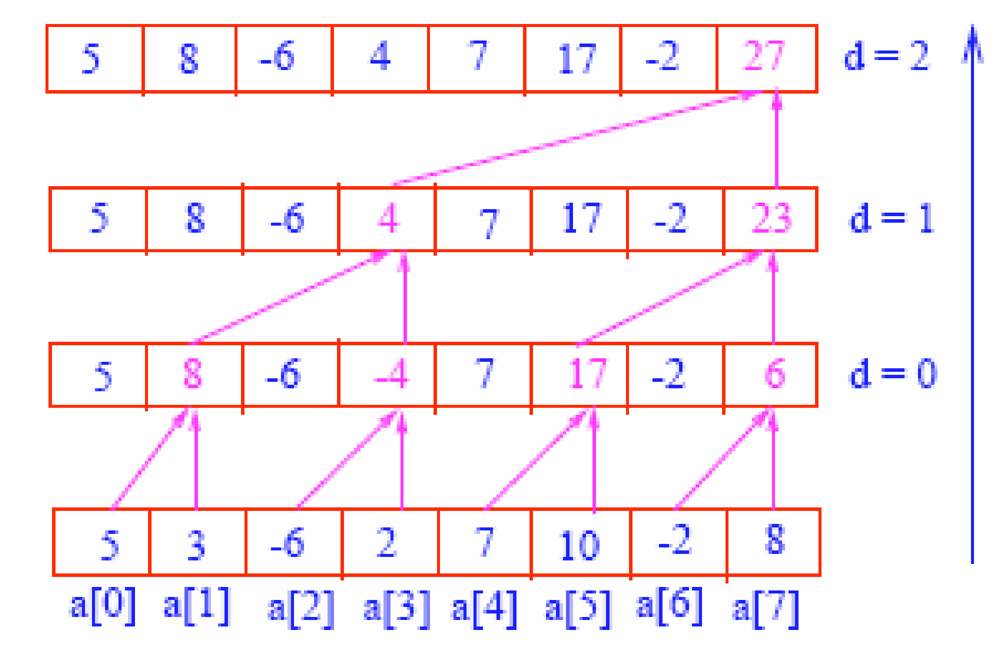
  $a[0 + 2^{1+1} - 1] := a[0 + 2^{1} - 1] + a[0 + 2^{1+1} - 1]$

  or, $a[3] := a[1] + a[3]$
- for $i = 4$,

  $a[4 + 2^{1+1} - 1] := a[4 + 2^{1} - 1] + a[4 + 2^{1+1} - 1]$

  or, $a[7] := a[5] + a[7]$

The First Pass



- blue: no change from last iteration.
- magenta: changed in the current iteration.

The Second Pass

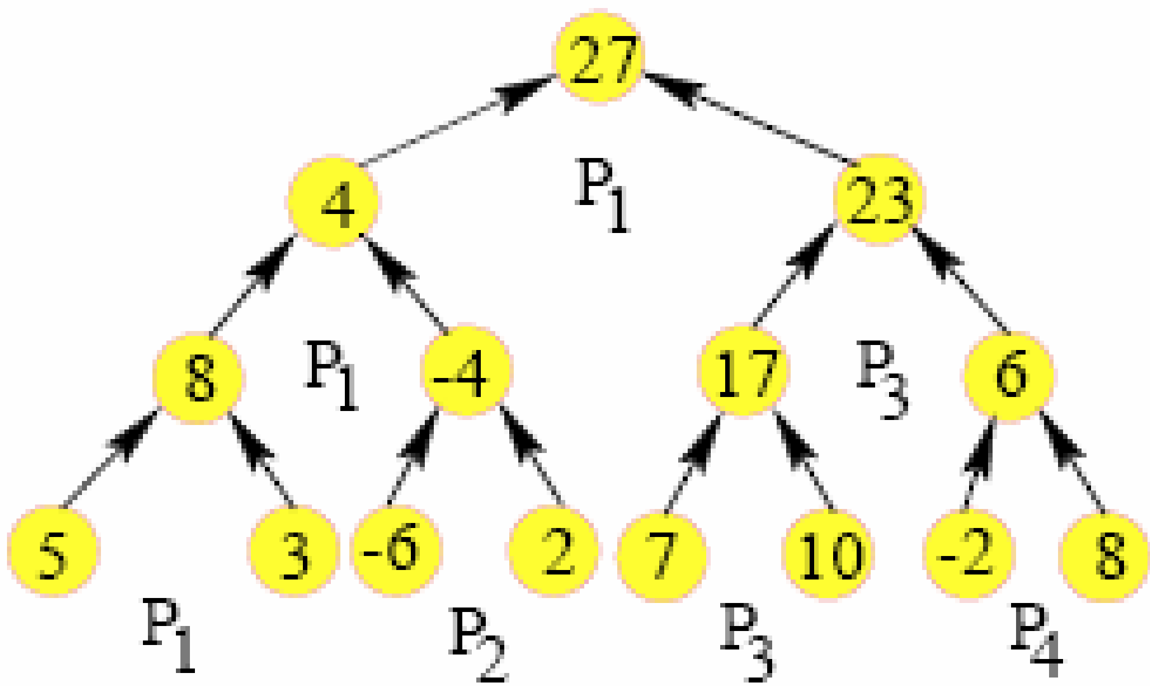
Second Pass

- The idea in the second pass is to do a topdown computation to generate all the prefix sums.
- We use the notation $pre[v]$ to denote the prefix sum at every node.

Computation in the second phase

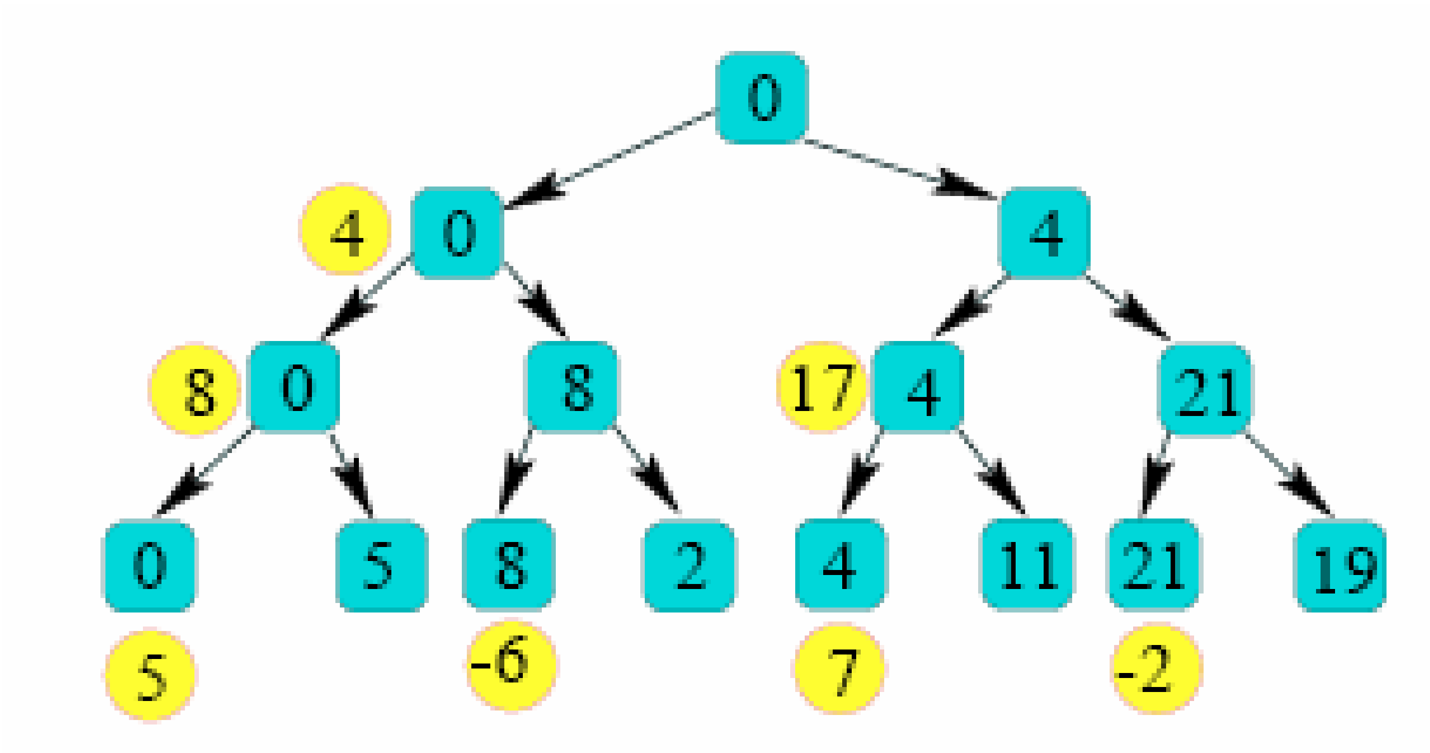
- $pre[root]$:= 0, the identity element for the $\oplus$ operation, since we are considering the $+$ operation.
- If the operation is max, the identity element will be $-\infty$.

Second phase (continued)



$$pre[L[v]] := pre[v]$$
$$pre[R[v]] := sum[L[v]] + pre[v]$$

Example of second phase



$pre[L[v]] := pre[v]$
$pre[R[v]] := sum[L[v]] + pre[v]$

Parallel prefix computation

for $d = (\log n - 1)$ downto 0 do

for $i = 0$ to $n - 1$ by 2^{d+1} do in parallel

$temp := a[i + 2^d - 1]$

$a[i + 2^d - 1] := a[i + 2^{d+1} - 1]$ (left child)

$a[i + 2^{d+1} - 1] := temp + a[i + 2^{d+1} - 1]$ (right child)

a[7] is set to 0

Parallel prefix computation

- We consider the case $d = 2$ and $i = 0$

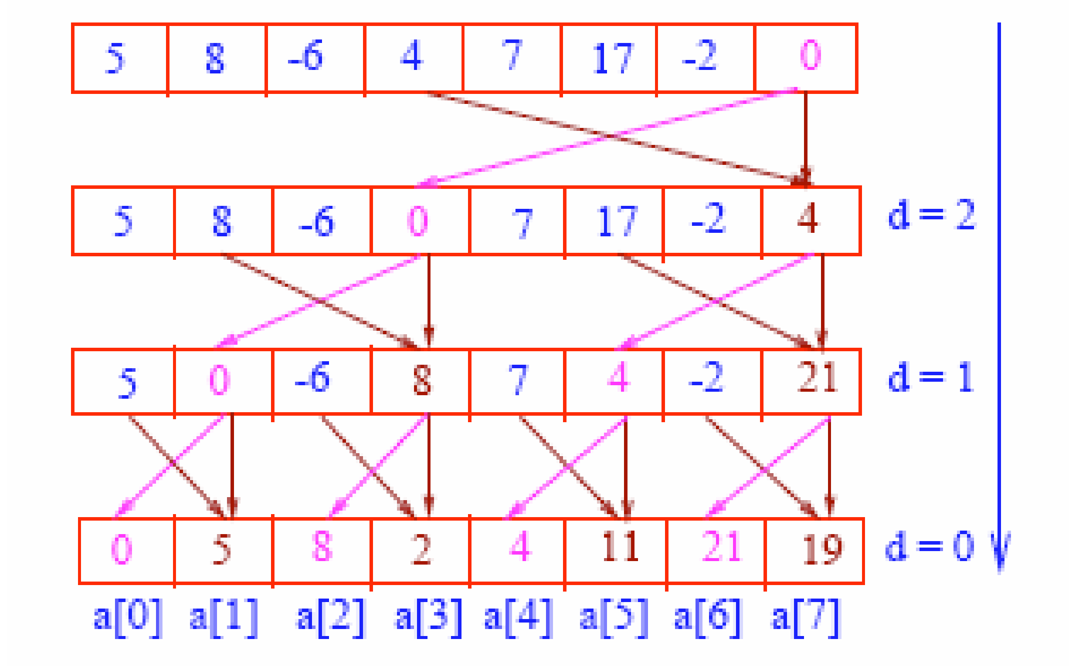
  $temp := a[0 + 2^2 - 1] := a[3]$

  $a[0 + 2^2 - 1] := a[0 + 2^{2+1} - 1]$ or, $a[3] := a[7]$

  $a[0 + 2^{2+1} - 1] := temp + a[0 + 2^{2+1} - 1]$ or,

  $a[7] := a[3] + a[7]$

Parallel prefix computation



- blue: no change from last iteration.
- magenta: left child.
- brown: right child.

Parallel prefix computation

- All the prefix sums except the last one are now in the leaves of the tree from left to right.
- The prefix sums have to be shifted one position to the left. Also, the last prefix sum (the sum of all the elements) should be inserted at the last leaf.
- The complexity is $O(\log n)$ time and $O(n)$ processors.

  Exercise: Reduce the processor complexity to $O(n / \log n)$.

Proof of correctness

- Vertex x precedes vertex y if x appears before y in the preorder (depth first) traversal of the tree.

Lemma: After the second pass, each vertex of the tree contains the sum of all the leaf values that precede it.

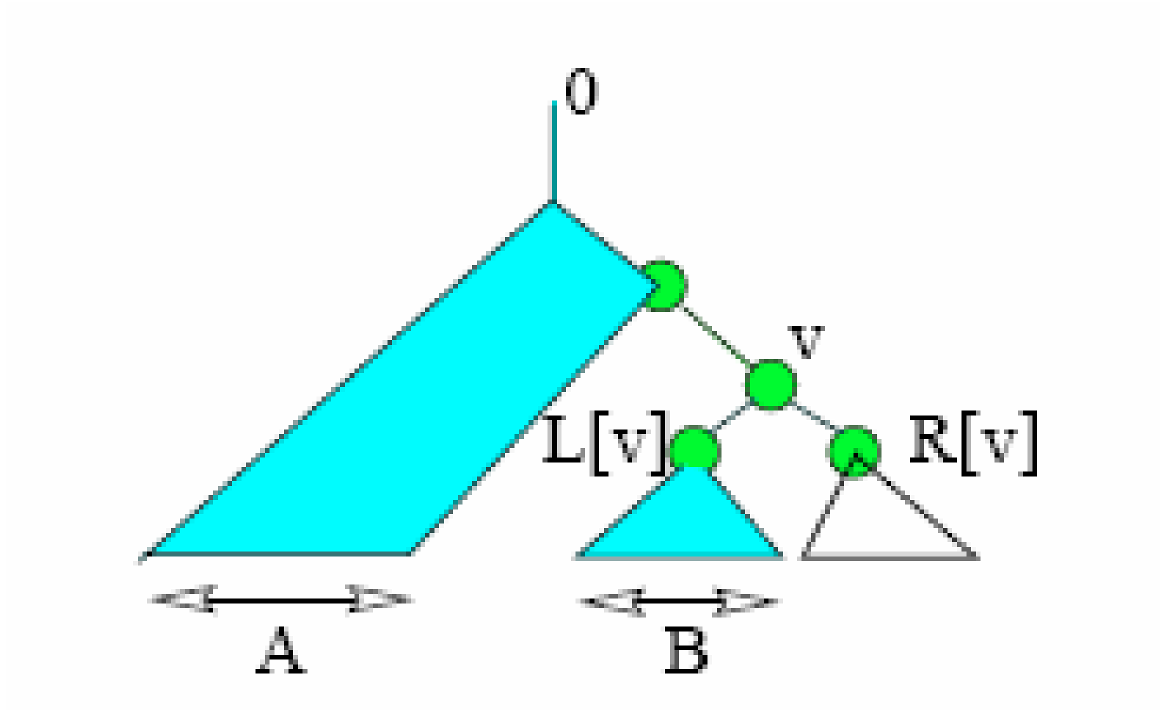
Proof: The proof is inductive starting from the root.

Proof of correctness

Inductive hypothesis: If a parent has the correct sum, both children must have the correct sum.

Base case: This is true for the root since the root does not have any node preceding it.

Proof of correctness



- Left child: The left child $L[v]$ of vertex v has exactly the same leaves preceding it as the vertex itself.

Proof of correctness

- These are the leaves in the region A for vertex $L[v]$.
- Hence for $L[v]$, we can copy $pre(v)$ as the parent's prefix sum is correct from the inductive hypothesis.

Proof of correctness

- Right child: The right child of v has two sets of leaves preceding it.
  - The leaves preceding the parent (region A) for $R[v]$
  - The leaves preceding $L[v]$ (region B).

$pre(v)$ is correct from the inductive hypothesis.

Hence, $pre(R[v]) := pre(v) + sum(L[v])$.