Algoritmos PRAM

Jones Albuquerque
DFM-UFRPE

2004, Recife - PE.

Algoritmos PRAM

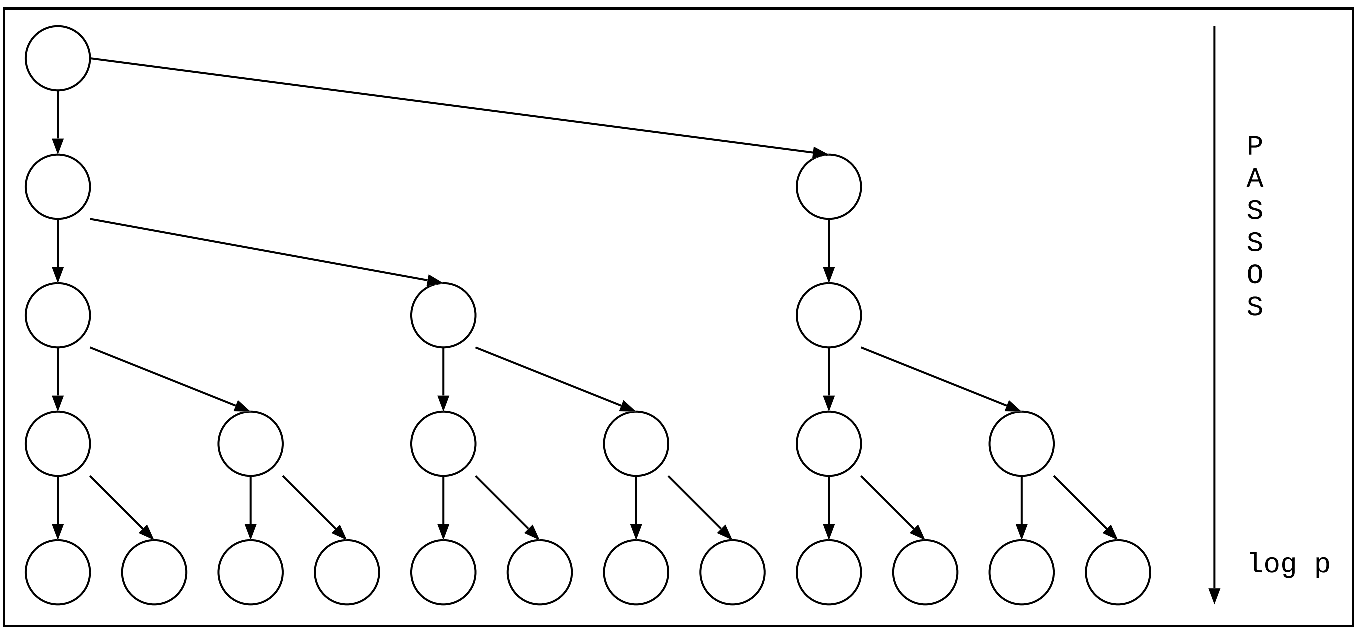
Como um algoritmo PRAM inicia com um único processador, os algoritmos têm duas fases:

- ativação de processadores;
- computação realizada pelos processadores ativos

Ativação. Dado um único processador, são necessários e suficientes $\lceil log\, p \rceil$ passos para ativar p processadores

Ativação

SPAWN (PROCESSADORES)



Cada processador habilita um único processador por vez

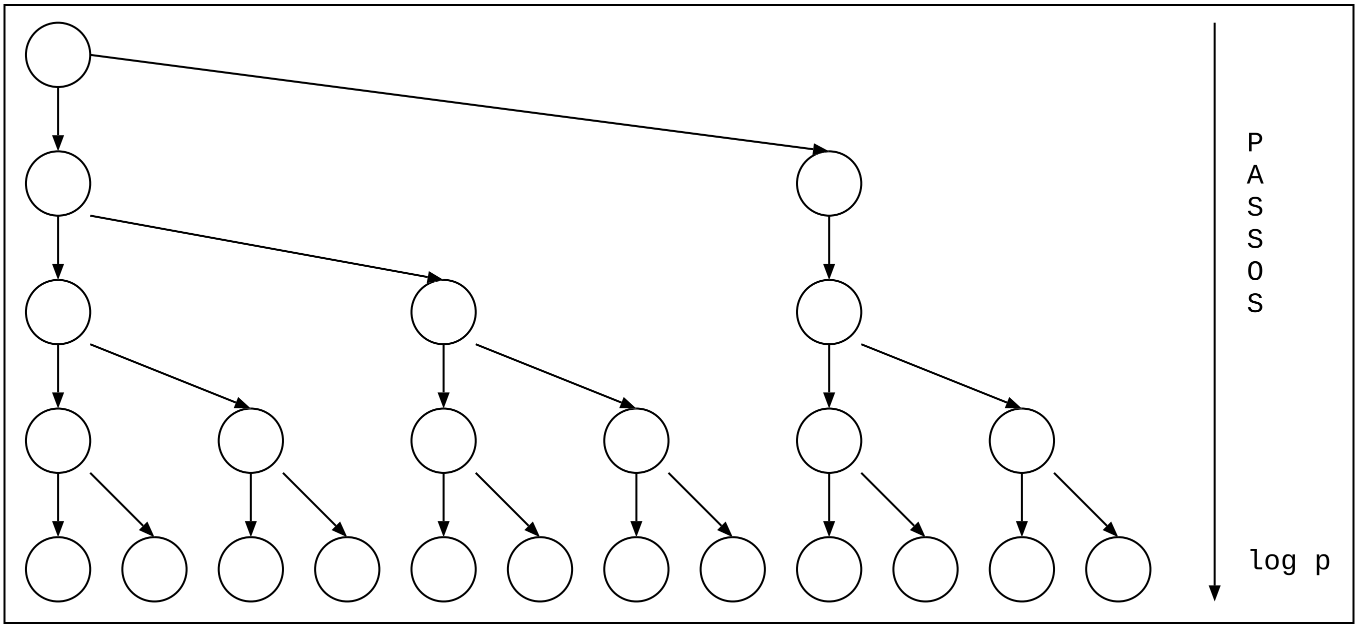
Fases - Computação

```
FOR ALL [lista de processadores] DO
    [lista de sentencas]
ENDFOR
```

Estruturas PRAM

IF [...] THEN [...] ELSE [...] ENDIF
FOR [...] ENDFOR
WHILE [...] ENDWHILE
REPEAT [...] UNTIL
$\leftarrow$ ATRIBUIÇÃO

Redução Paralela

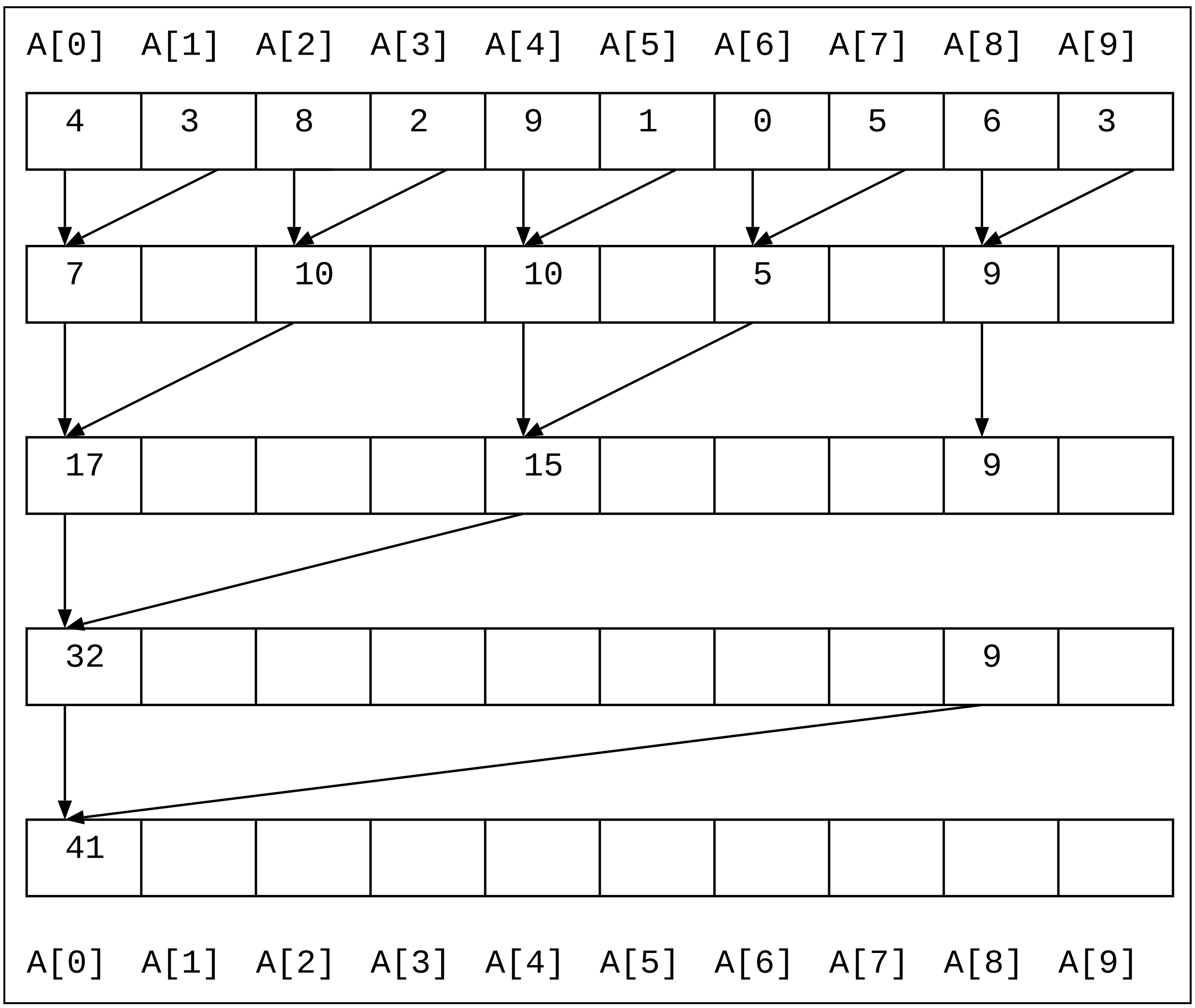


Paradigma da árvore binária (*broadcast, divide-and-conquer*)

Operações de redução ou *fan-in*

Dado um conjunto de n valores $a_1, a_2, \ldots, a_n$ e um operador binário $\oplus$, **redução** é o processo de computar $a_1 \oplus a_2 \oplus \cdots \oplus a_n$.

Exemplo Redução: Soma Paralela



Algoritmo: Soma Paralela

SUM (EREW PRAM)
Initial condition: List of $n \geq 1$ elements stored in $A[0...(n-1)]$
Final condition: Sum of elements stored in $A[0]$
Global variables: n, $A[0...(n-1)], j$

```
begin
   spawn(P_0, P_1, P_2, ..., P_{⌊n/2⌋-1})
   for all P_i where 0 ≤ i ≤ ⌊n/2⌋ - 1 do
      for j ← 0 to ⌈log n⌉ - 1 do
         if i modulo 2^j = 0 and 2^i + 2^j < n then
            A[2i] ← A[2i] + A[2i + 2^j]
         endif
      endfor
   endfor
end
```

Análise de complexidade:

Spawn = $\lceil log \lfloor n/2 \rfloor \rceil$

Loop = $\lceil log\, n \rceil$, cada iteração possui complexidade de tempo constante

Custo total = $\theta(log\ n)$, dado $\lfloor n/2 \rfloor$ processadores

Soma de Prefixos

Dado um conjunto com n valores $a_1, a_2, \ldots, a_n$ e uma operação associacitiva $\oplus$, a soma de prefixos é a computação de n valores

a_1

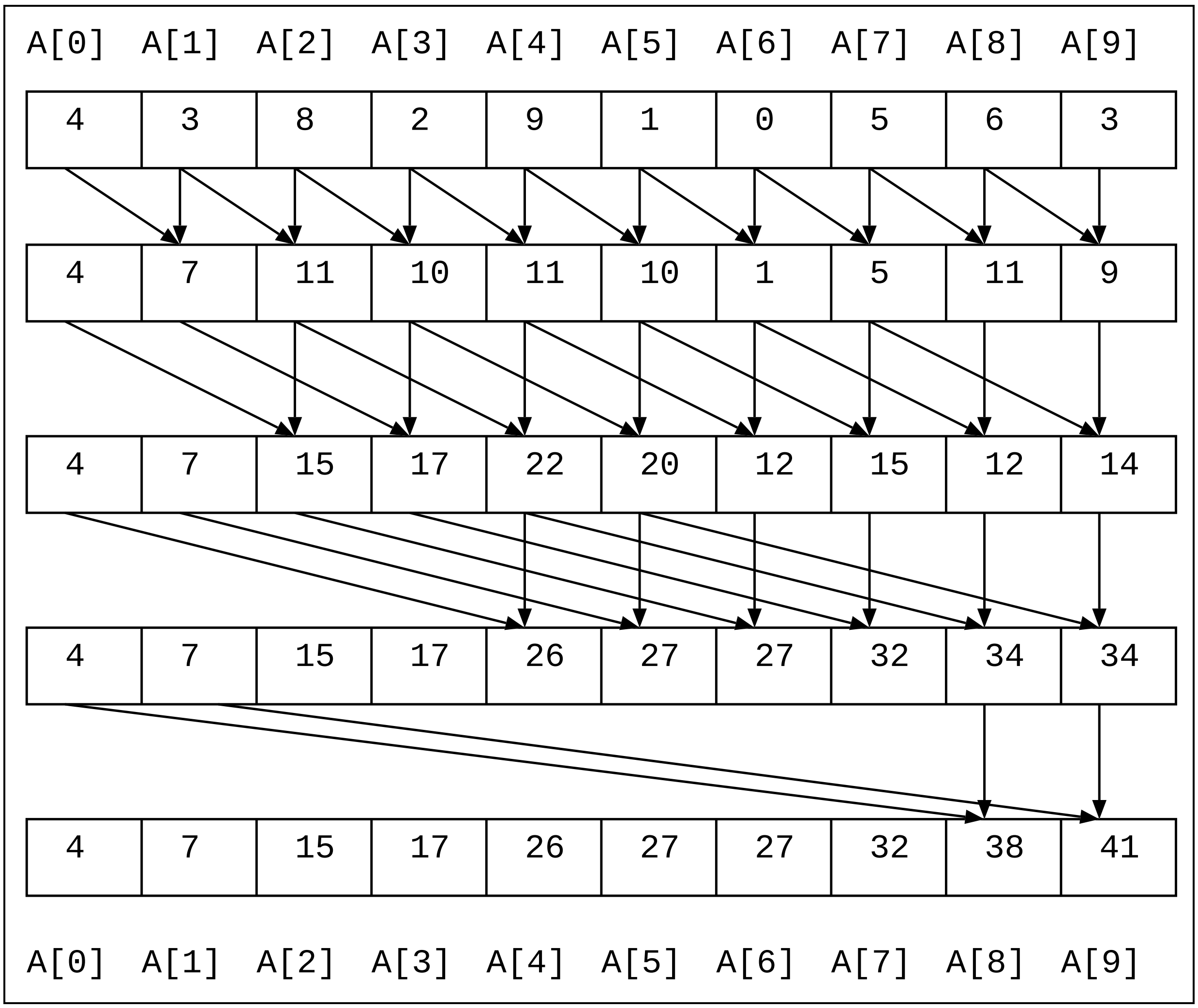
$a_1 \oplus a_2$

$\ldots$

$a_1 \oplus a_2 \oplus a_3 \oplus \cdots \oplus a_n$

Por exemplo, dada a operação $+$ e um vetor de inteiros $\{3, 1, 0, 4, 2\}$ a soma de prefixos do vetor é $\{3, 4, 4, 8, 10\}$

Soma de Prefixos em Paralelo



Algoritmo: Soma de Prefixos

PREFIX.SUMS (CREW PRAM)
Initial condition: List of $n \geq 1$ elements stored in $A[0...(n-1)]$
Final condition: Each element $A[i]$ contains $A[0] \oplus A[1] \oplus \cdots \oplus A[i]$
Global variables: n, $A[0...(n-1)], j$

```
begin
   spawn(P_1, P_2, ..., P_{n-1})
   for all P_i where 1 ≤ i ≤ n − 1 do
      for j ← 0 to ⌈log n⌉ − 1 do
         if i − 2^j ≥ 0 then
            A[i] ← A[i] + A[i − 2^j]
         endif
      endfor
   endfor
end
```

Análise de complexidade:

Spawn = $\lceil log\ n - 1 \rceil$

Loop = $\lceil log\ n \rceil$, cada iteração possui complexidade de tempo constante

Custo total = $\theta(log\ n)$, para $n-1$ processadores

Outros Algoritmos Paralelos

List ranking

variante da soma de prefixos, onde os sufixos sao calculados como uma lista encadeada, na qual os elementos são `1` ou `0` e a operação é adição

Preorder tree traversal

numerar os vértices de uma árvore em pré-ordem (ordem obtida com uma busca em profundidade)

Merging two sorted lists

Graph coloring

determinar se os vértices de um grafo podem ser coloridos com c cores de tal forma que vértices adjacentes possuam cores diferentes