PRAM Algorithms

Why do we need a PRAM model?

- to make it easy to reason about algorithms
- to achieve complexity bounds
- to analyze the maximum parallelism

PRAM Complexity Measures

- for each individual processor
  - ***time***: number of instructions executed
  - ***space***: number of memory cells accessed

- PRAM machine
  - ***time***: time taken by the longest running processor
  - ***hardware***: maximum number of active processors

Two Technical Issues

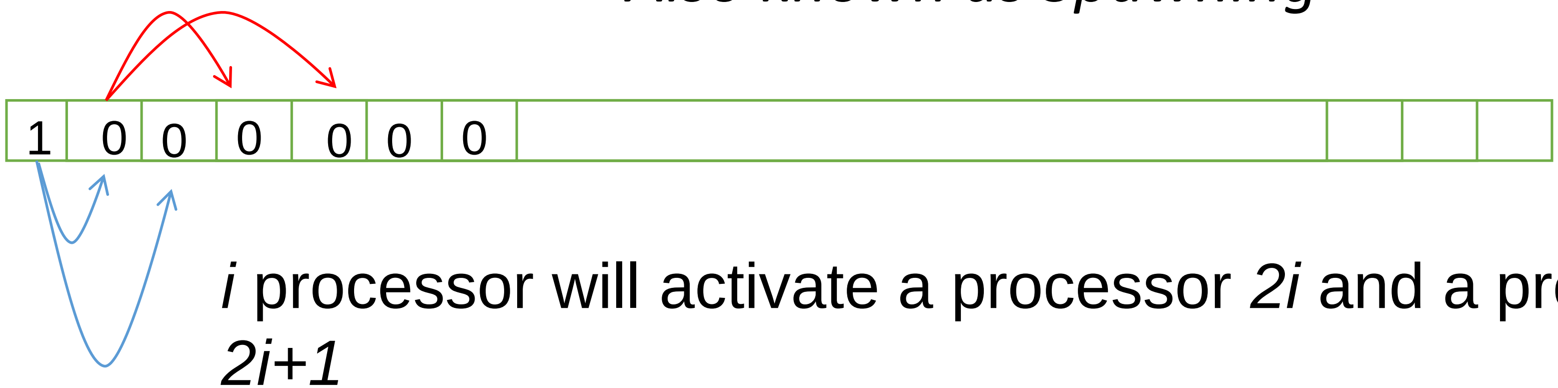
- How processors are activated
- How shared memory is accessed

Processor Activation

P_0 places the number of processors (p) in the designated shared-memory cell

- each active P_i, where $i < p$, starts executing
- $O(1)$ time to activate
- all processors halt when P_0 halts

- Active processors explicitly activate additional processors via FORK instructions
  - tree-like activation
  - $O(\log p)$ time to activate
- *Also known as Spawning*



i processor will activate a processor *2i* and a processor *2i+1*

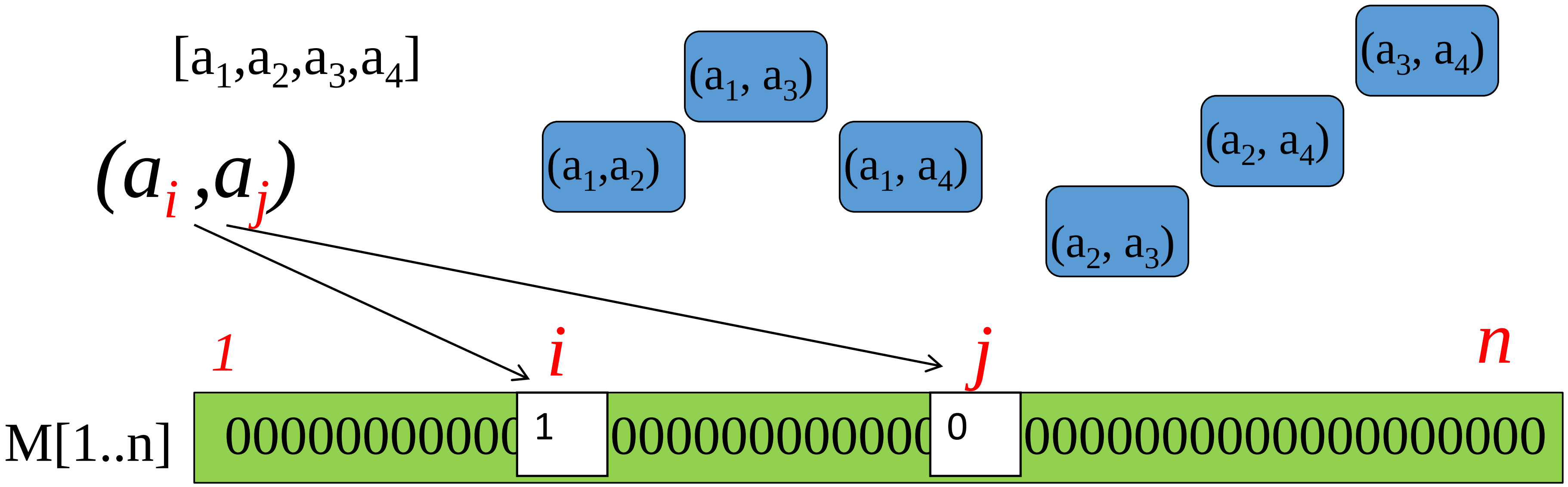
Computing the "Boolean OR" of A[1], A[2], A[3], A[4], A[5]

- Using CRCW PRAM
- Initially
  - table **A** contains values 0 and 1
  - **output** contains value 0

$$\text{for each } 1 \leq i \leq 5 \text{ do in parallel}$$
$$\quad \text{if } A[i] = 1 \text{ then output=1;}$$

Minimum of n numbers

Comparisons between numbers can be done independently
The second part is to find the result using concurrent write mode
For n numbers ----> we have ~ n^2 pairs



If $a_i > a_j$ *then* a_i *cannot be the minimal number*

Minimum of n numbers

for each $1 \leq i \leq n$ *do in parallel*

    $M[i]:=0$

for each $1 \leq i,j \leq n$ *do in parallel*

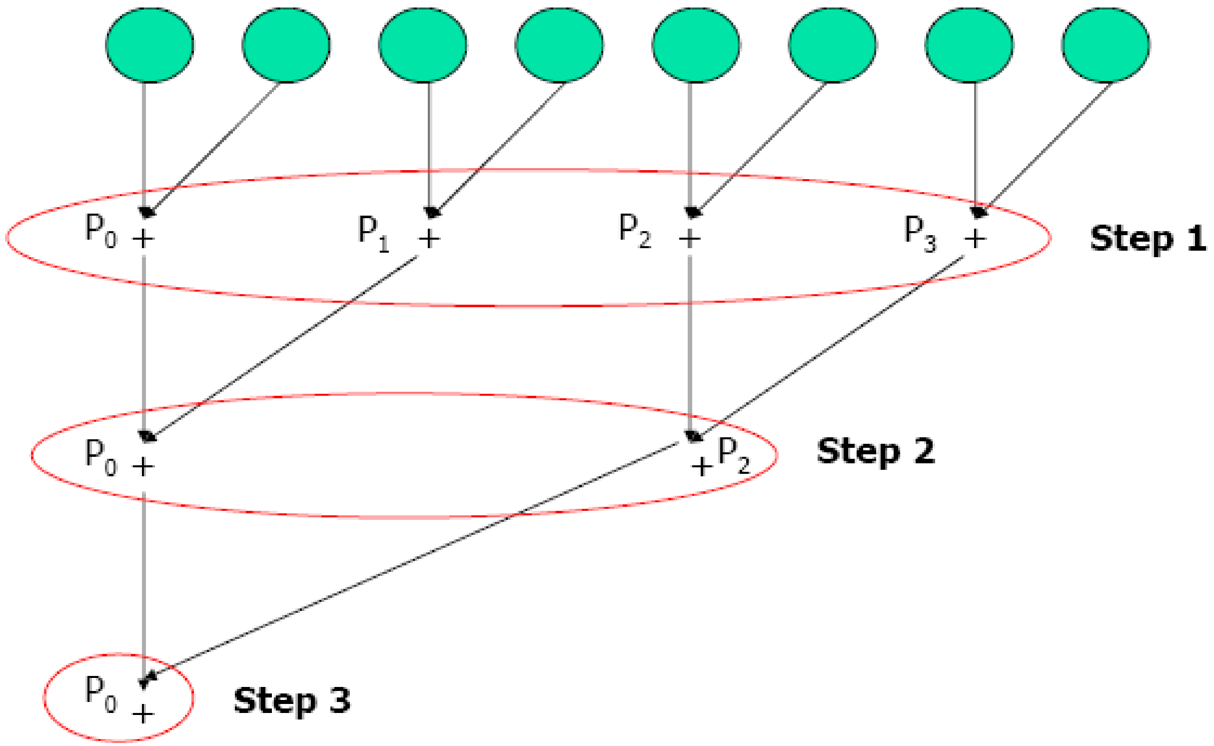
    if $i \neq j$ $C[i] \leq C[j]$ *then* $M[j]:=1$

for each $1 \leq i \leq n$ *do in parallel*

    if $M[i]=0$ *then output*$:=i$

computes MIN of n numbers stored in the array C[1..n] in O(1) time with n^2 processors.

Sum of *n* elements



Reduction:

Given a set of n values *a1, a2, … an* and an associative binary operator +, reduction is the process of computing *a1+a2+ … +an*

log(n) steps
n/2 processors
Speed-up = n/log(n)
Applicable for other operations too

EREW PRAM algorithm

SUM (EREW PRAM)
Input: A[0 … (n-1)]
Output: sum stored in A[0]
Begin

```
        spawn (P0, P1, P2, … P_{n/2 – 1})
        for all P_i where 0 ≤ i ≤ n/2 -1 do in parallel
                for j = 0 to log n -1 do
                        if i modulo 2^j = 0 and 2i + 2^j < n then
                        A[2i] = A[2i] + A[2i + 2^j]
                        endif
                endfor
        endfor
```

end

Time complexity
Spawning log n/2
Sequential for loop executes in log n time

Overall
Θ (log n) on n/2 processors

Sorting using CRCW PRAM

Spawn n^2 Processors

```
for i = 1 to n do in parallel
        for j = 1 to n do in parallel
                if Si > Sj or (Si = Sj and i > j ) then
                        P_{i, j}  writes 1 to r_i
                endif
        endfor
endfor
for i = 1 to n do in parallel
        P_{i, 1}  puts Si in (r_i + 1) position of S
endfor
```