ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY

SOICT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Advanced Parallel Algorithms

References

- Michael J. Quinn. **Parallel Computing. Theory and Practice**. McGraw-Hill
- Albert Y. Zomaya. **Parallel and Distributed Computing Handbook**. McGraw-Hill
- Ian Foster. **Designing and Building Parallel Programs**. Addison-Wesley.
- Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar **. Introduction to Parallel Computing, Second Edition.** Addison Wesley.
- Joseph Jaja**. An Introduction to Parallel Algorithm.** Addison Wesley.
- Nguyễn Đức Nghĩa**. Tính toán song song.** Hà Nội 2003.

8.1 Parallel Recursive

Divide and Conquer

- The principle of "divide and conquer" as follows:
  - B1: Divide the original problem into smaller problems.
  - B2: Recursive implementation with small problems.
  - B3: Combine results from small problems to obtain original problem results.
  - Small problems are independent of each other so they can be done in parallel.
- The problem is how to do steps 1 and 3 the most effectively ???

Devide and Conquer

General strategy:

Divide-and-conquer(Problem P of size n):

If P is trivial (i.e., n very small), solve P directly.
otherwise:

divide P into $q \geq 1$ *independent* subproblems $P_1,\ldots,P_q$ of smaller size $n_1,\ldots,n_q$

solve these recursively:

$S_1 \leftarrow$ Divide-and-conquer(P_1, n_1),

$\vdots$

$S_q \leftarrow$ Divide-and-conquer(P_q, n_q)

combine the subsolutions $S_1,\ldots,S_q$ to a solution S for P.

Complexity

- Considering the problem P having n-length, divided into q child problems of n/k-length (each problem has k elements, k>1), executed in parallel with p processors

$$
\text{t\_divide\_conquer}(n,p) = \begin{cases} \text{t\_run\_trivial}(n) \text{ if } n \text{ is small enough} \\ \text{t\_run\_serial}(n) \text{ if } p = 1 \\ \text{t\_divide}(n,p) + \text{t\_combine}(n,p) + q/p \\ \quad \text{t\_divide\_conquer}(k,1) \text{ if } 1 < p < q \\ \text{t\_divide}(n,p) + \text{t\_combine}(n,p) + \\ \quad \text{t\_divide\_conquer}(k,p/q) \text{ if } p > q \text{ or } p=q \end{cases}
$$

For Example (1)

- Sum of n numbers A[1..n] with p processors.
- Idea:
  - If $n = 1 \rightarrow$ return value A[1].
  - If $p = 1 \rightarrow$ run in serial mode.
  - Divide array A into 2 parts A1 and A2, each containing n/2 elements, executed in parallel:
    - Calculate recursively S1: sum of all A1's elements with p/2 processors.
    - Calculate recursively S2: sum of all A2's elements with p/2 processors.
  - Get the total S = S1+S2.

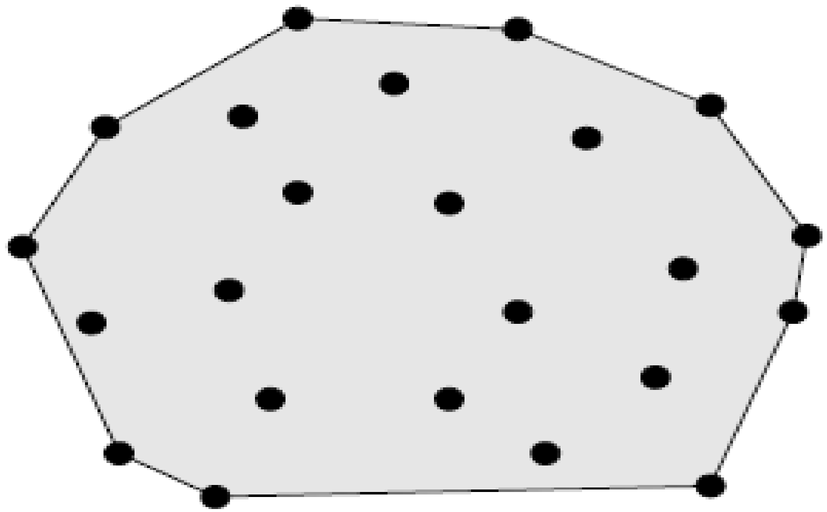
Sum of n numbers A[1..n] with p processors

```
INPUT        :    A[1..n], p bộ xử lý;
OUTPUT       :    SUM = ∑ A[i];
FUNCTION S = SUM(A,n,m,p)  // n,m la chi so dau tien va cuoi cung
BEGIN
    IF p = 1 THEN
        S = SEQUENCE_SUM(A,n,m);
    END IF.
    DO IN PARALLEL
        S1  =   SUM(A1,n,(n+m)/2,p/2);
        S2  =   SUM(A2,(n+m)/2,m,p/2);
    END DO
    S   =   S1 + S2;
END;
```

- Recursive equation: $T(n) = T(n/2) + O(1)$ (considering $p \approx n$)
- → Complexity: $O(\log n)$.
- Machine PRAM EREW.

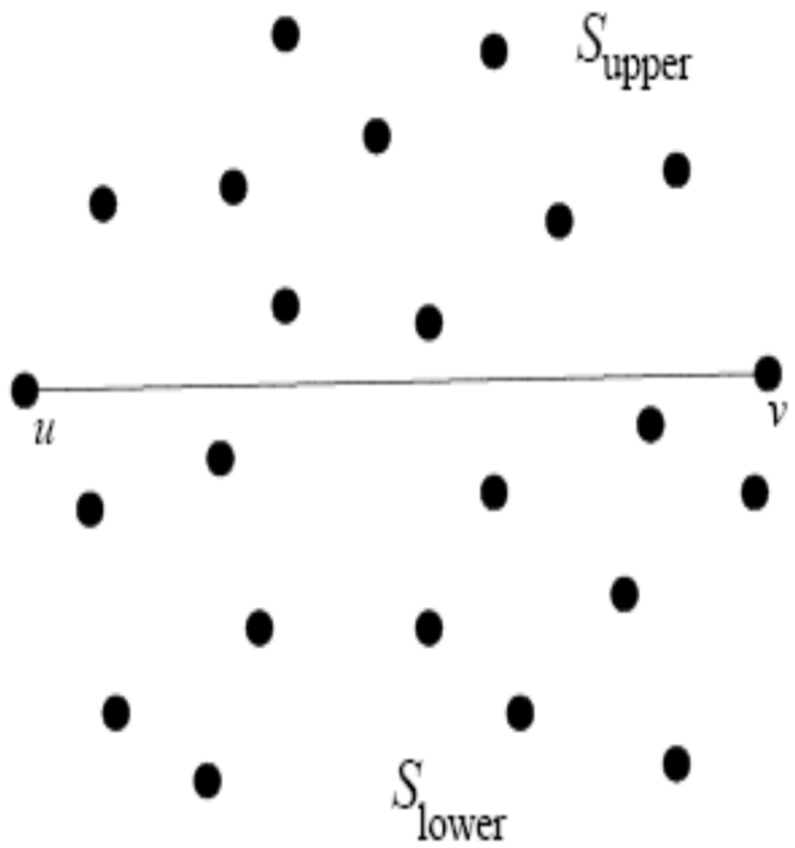
Example : convex hull

- The problem of determining the convex envelope of a set of vertices in the plane.
  - Input: n vertex (xk, yk) in the plane.
  - Output: Set of vertices that form the smallest convex polygon containing all the remaining vertices.



Parallel QuickHull

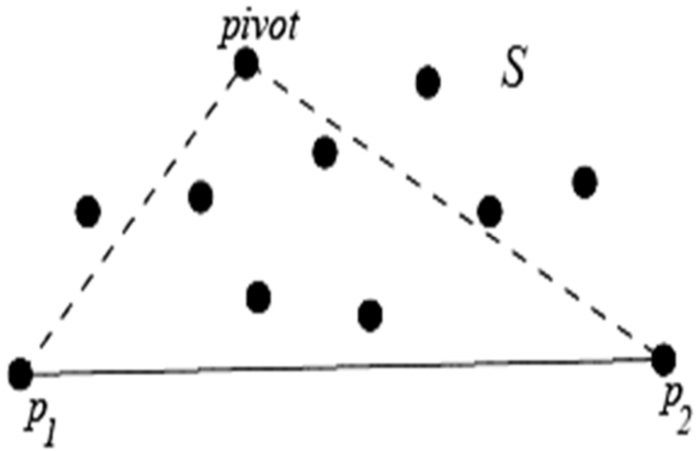
- Idea:
  - Initial: Define u,v as 2 vertices with x coordinate values being the smallest and largest → hence u, v are in convex envelope.
  - Segment (u,v) divides the initial set S into 2 upper and lower regions: S_upper and S_lower
  - Treating S_upper and S_lower in parallel.

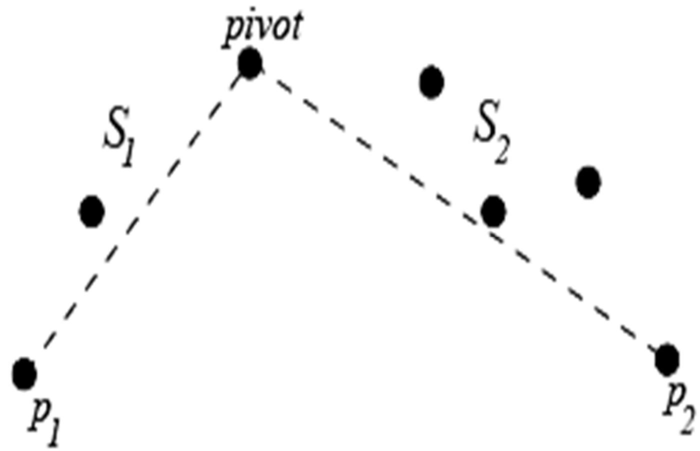


Parallel QuickHull

- Both upper hull and lower hull can be treated in the same way.
- Division step:
  - Select the pivot p as the point that has the longest distance from the (p_1 , p_2).
  - The pivot point will be on the convex envelope → points in the triangle (p, p1, p2) are eliminated.
  - The remaining points are divided into 2 parts outside the edges: (p, p_1) and $(p_2\ p)$.
  - Recursive implementation with these two parts with the steps as above.

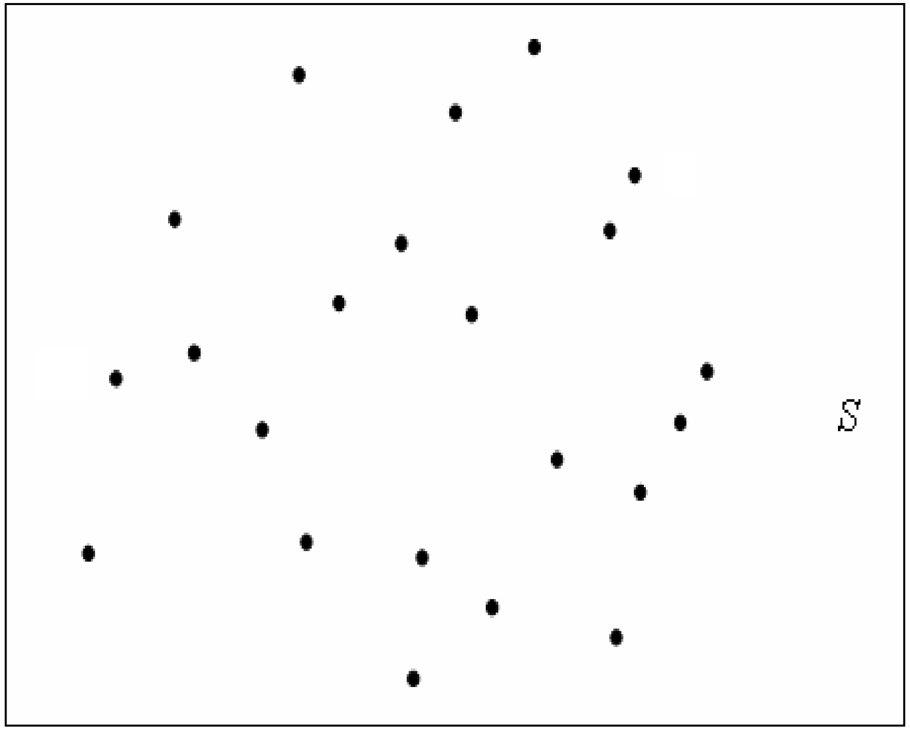
Parallel QuickHull



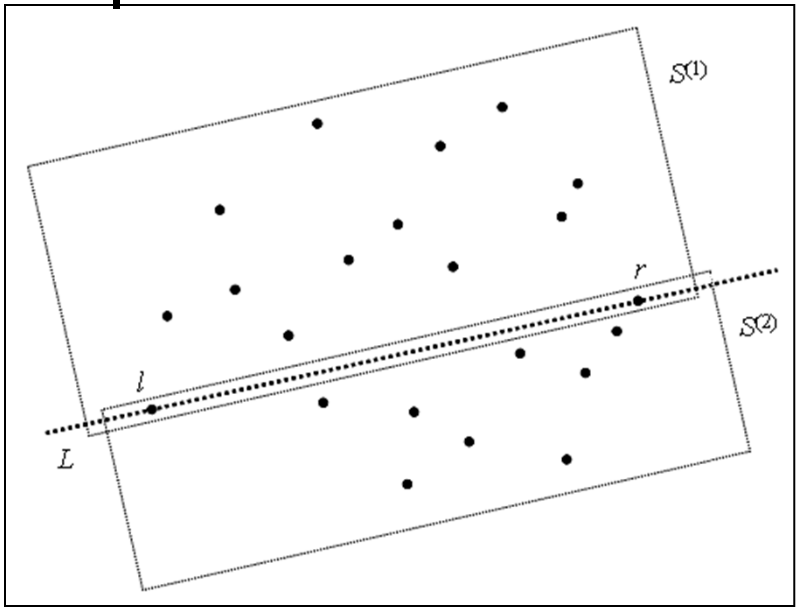


- Signs:
  - Pivot point p: max $|(p_1 - p) \times (p_2 - p)|$;
  - The vertexes are in the triangle if the total angles from that vertex are equal to 2π.
  - Angle between 2 vectors: $\cos(a,b) = (a \times b)/(|a||b|)$

Illustration steps



Uper and Lower Hull



Set of vertexes in S

Illustration steps

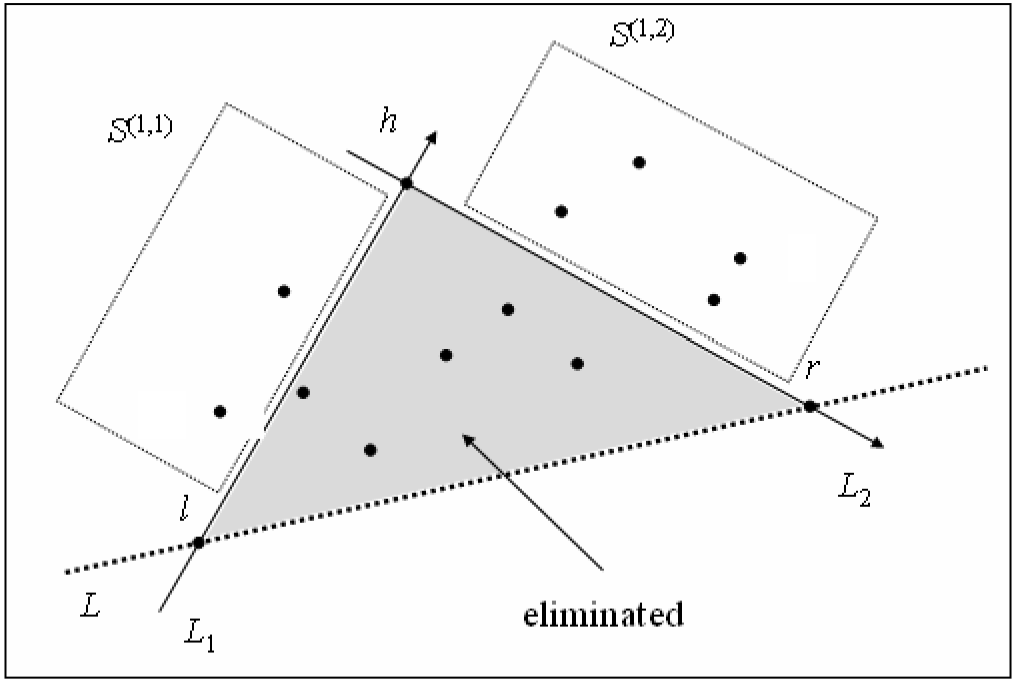
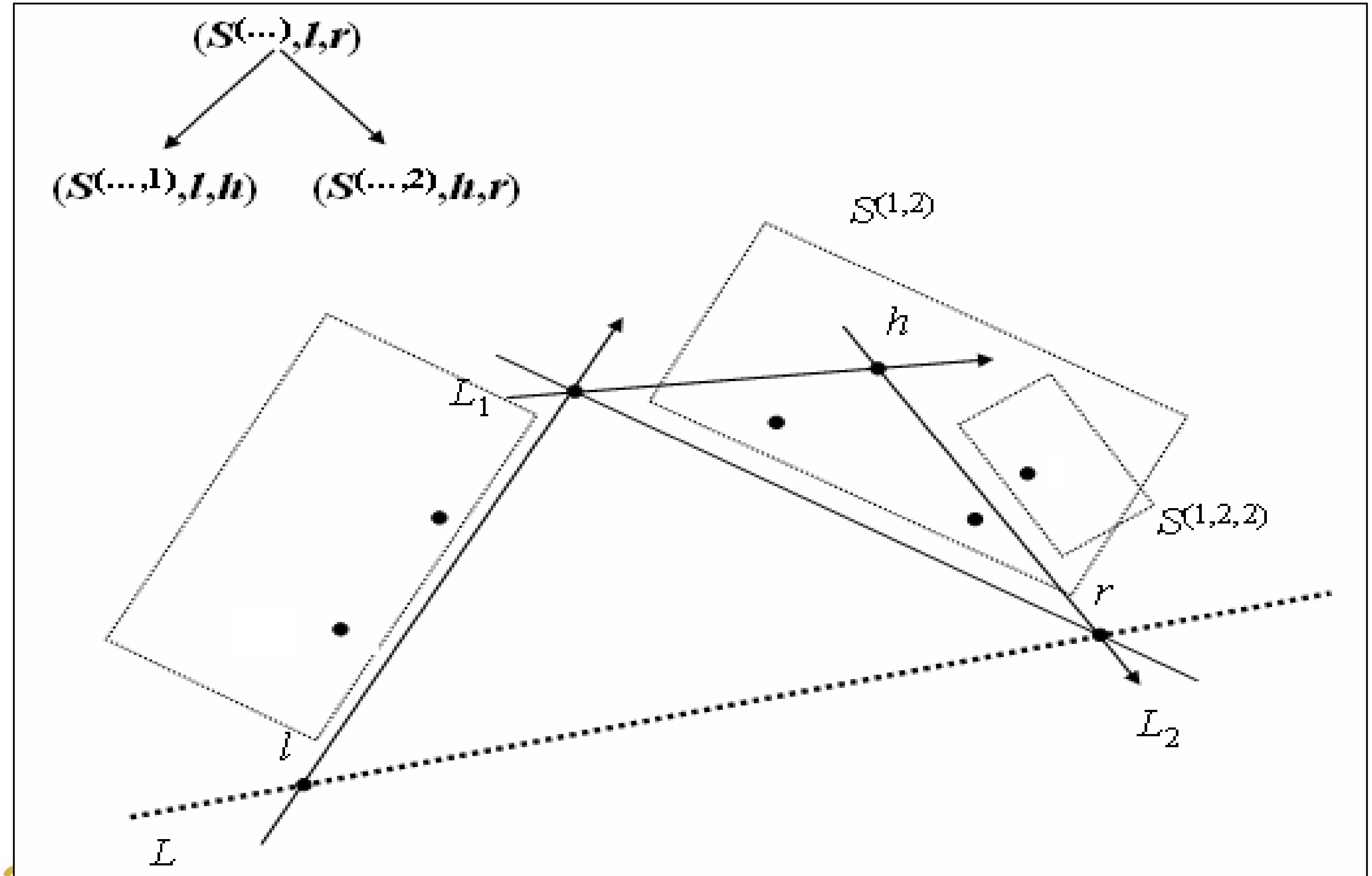


Illustration steps



Procedure QUICKHULL

```
procedure QUICKHULL(S, l, r)
begin
    if  S = {l, r} then
        return (l, r)  /* lr is an edge of H(S) */
    else
        h = FURTHEST(S, l, r)
        S(1) = p ∈ S ∋ p is on or left of line lh
        S(2) = p ∈ S ∋ p is on or left of line hr
        return QUICKHULL(S(1), l, h) ||
            (QUICKHULL(S(2), h, r) - h)
    end
end

Initial call
begin
    l0 = (x0,y0)  /* point of S with smallest abscissa */
    r0 = (x0, y0 - ε)
    result = QUICKHULL(S, l0, r0) - r0
/* The point r0 is eliminated from the final list*/
end
```

Recursive in PRAM

- If we represent the sub-problem levels of a recursive algorithm as a tree → get the k-level tree (with k = 2 → binary tree).
- The recursive idea in PRAM is to divide the set of processors into groups, each of which will correspond to a sub-tree in the tree.
- Executing in parallel with all processors, in each group of processors, doing the work corresponding to the sub-tree.

Recursive with UperHull

- Variables used:
  - Each point i corresponds to 1 variable F[i]:
    - Initial: F[i] = 1;
    - Eliminated for being in: F[i] = 0;
    - Marked as the point in the convex envelope: F[i] = 2.
  - Since each vertex set will be assigned to two bottom points, each vertex determines the current 2 bottom points through the variable: P[i] và Q[i].
- Recursive steps:
  - All processors perform in parallel to identify the T[i].
  - Update peaks P[i] and Q[i]:
    - Left vertexes (P[i],T[i]) assigned Q[i] = T[i].
    - Left vertexes (T[i],Q[i]) assigned P[i] = T[i].
  - Update again values F[i].
  - Repeat the above work until all F[i] $\neq$ 1.

8.2 Accelerated Cascading

Concepts of complexity

- In serial calculation, there is only one concept of complexity = number of steps to implement the algorithm (≈ duration): Called S(n)
- In parallel calculations there is additional concept number of operations performed on all processors: Called W(n).
- If $W_i(n)$ is the number of operations performed simultaneously at step i → we have a formula:

$$W(n) = \sum_{i=1}^{S(n)} W_i(n)$$

Example of S(n) and W(n)

- The combined problem with $n = 2^k$ values using math operation $\oplus$. Balancing tree algorithm as follows:

```
T REDUCE(sequence⟨T⟩ a, ⊕ : T × T → T)

T B[1..n]
forall i ∈ 1 : n do
    B[i] ← a_i
enddo
for h = 1  to k do
    forall i ∈ 1 : n/2^h do
        B[i] ← B[2i − 1] ⊕ B[2i]
    enddo
enddo
S ← B[1]
return S
```

Examples of S(n) and W(n)

- Define S(n) and W(n) values according to algorithm's segment codes as follows:

$$
\begin{gathered}
S_{2\text{--}4}(n) = \Theta(1) \\
S_{6\text{--}8}(n) = \Theta(1) \\
S_{5\text{--}9}(n) = kS_{6\text{--}8}(n) = \Theta(\lg n) \\
S_{10}(n) = \Theta(1) \\
S(n) = S_{2\text{--}4}(n) + S_{5\text{--}9}(n) + S_{10}(n) = \Theta(\lg n)
\end{gathered}
$$

$$
\begin{gathered}
W_{2\text{--}4}(n) = \Theta(n) \\
W_{6\text{--}8}(n,h) = \Theta(\tfrac{n}{2^h}) \\
W_{5\text{--}9}(n) = \sum_{h=1}^{k} W_{6\text{--}8}(n,h) = \Theta(n) \\
W_{10}(n) = \Theta(1) \\
W(n) = W_{2\text{--}4}(n) + W_{5\text{--}9}(n) + W_{10}(n) = \Theta(n)
\end{gathered}
$$

Accelerated Cascading Technique

- The cost of an algorithm is the number of operations that the system must perform.
- Some algorithms are called optimal if: $W(n) = \Theta(T_s(n))$. where:
  - $W(n)$: cost of parallel algorithm.
  - $T_s(n)$: the best execution time of serial algorithm.
- Accelerated Cascading technique combines non-optimal algorithm but faster execution time with optimal algorithm but slow execution time.

Example (1)

- An array L[1..n] receives integer values from 1..k with $k = O(\log_2 n)$. Determine the number of occurrences of integers appeared in array L.
- Let R[i] be the number of occurrences of value i.
- Optimal serial algorithm $T_s(n) = \Theta(n)$:

$$
\begin{array}{l}
R[1:k] \leftarrow 0 \\
\textbf{for } i = 1 \textbf{ to } n \textbf{ do} \\
\quad R[L[i]] \leftarrow R[L[i]] + 1 \\
\textbf{enddo}
\end{array}
$$

First parallel approach

- Use two-dimensional arrays: C[1..n,1..k] $C_{i,j} = \begin{cases} 1 & \text{if } L[i] = j \\ 0 & \text{otherwise} \end{cases}$

- The the number of occurrences of integer i equals the total of C[1:n,j].

$$
\begin{array}{l}
\textbf{forall } i \in 1:n, j \in 1:k \textbf{ do} \\
\quad C[i,j] \leftarrow 0 \\
\textbf{enddo} \\
\textbf{forall } i \in 1:n \textbf{ do} \\
\quad C[i, L[i]] \leftarrow 1 \\
\textbf{enddo} \\
\textbf{forall } j \in 1:k \textbf{ do} \\
\quad R[j] \leftarrow \textsc{Reduce}(C[1:n,j], +) \\
\textbf{enddo}
\end{array}
$$

Example with n = 8 & k = 4

L[1..8] = 1 2 2 3 2 1 4 2 (k = 4)

	1	2	3	4
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
	R1	R2	R3	R4

	1	2	3	4
1	1			
2		1		
3		1		
4			1	
5		1		
6	1			
7				1
8		1		
	R1	R2	R3	R4

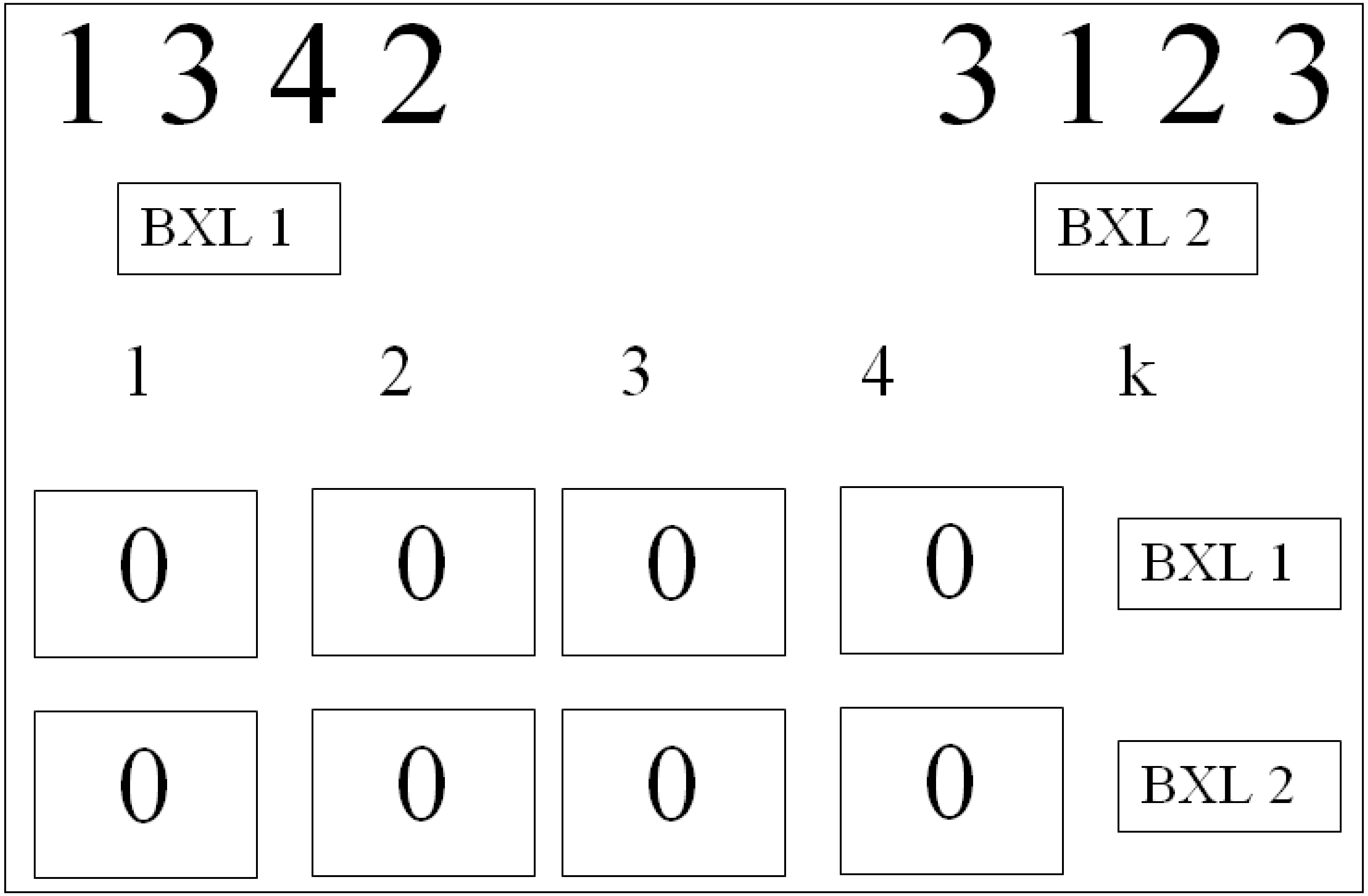
Comments …

- The number of operations performed by the first two parallel loops is $\Theta(nk)$ with $\Theta(1)$ step.
- Execution time in paragraph 3 according to binary tree model: $\Theta(\log_2 n)$.
- The real number of calculations in paragraph 3 is: $\Theta(nk)$
- This algorithm is not cost effective → not optimal.

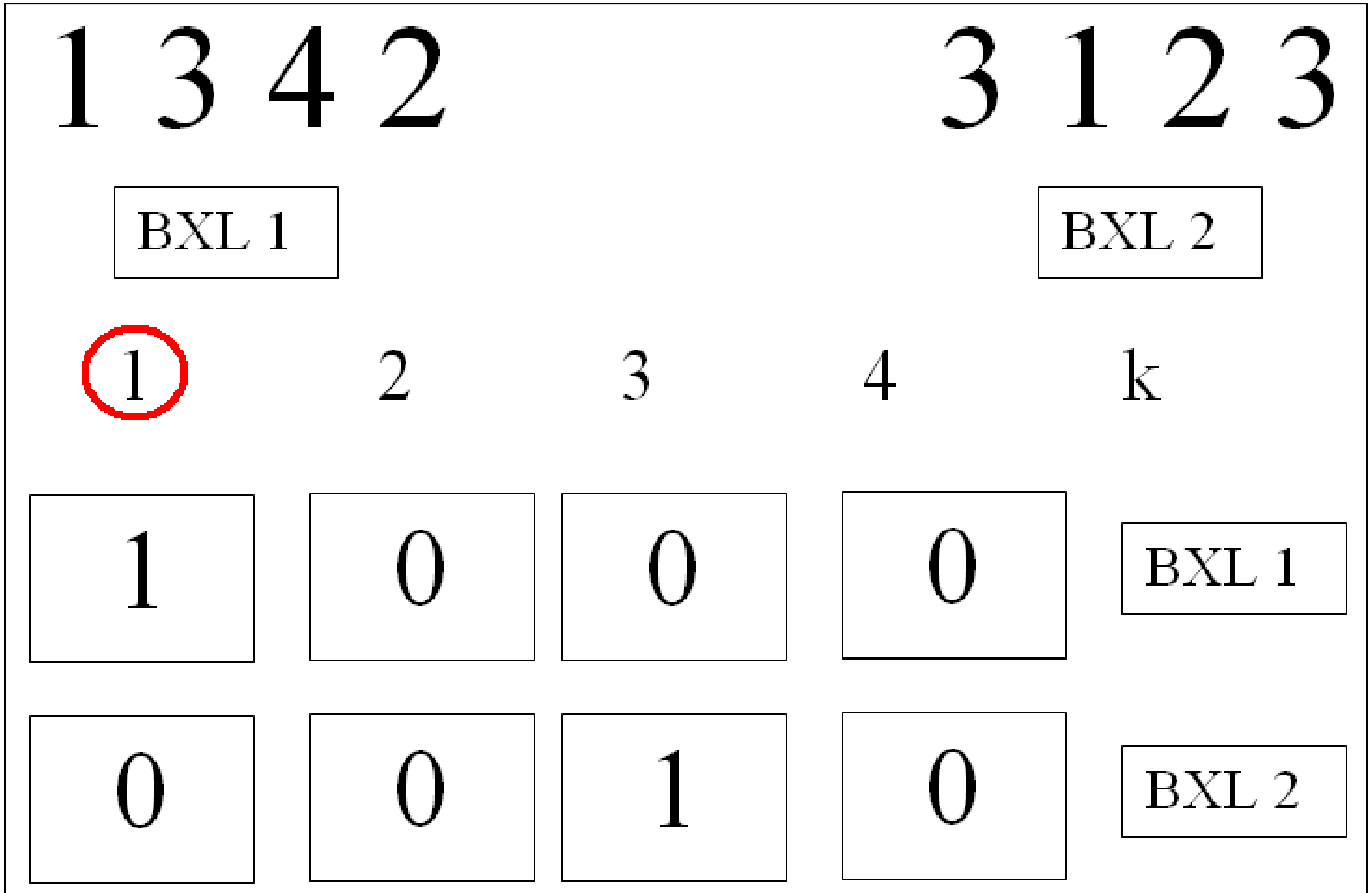
Using Accelerated Cascading

- With m=n/k, setup array Ĉ[1..m.1.k] corresponding to array L. That is, dividing L into m sub-array of k elements.
- Using m processor in order to scan from 1:k, i.e. each processor performs 1 optimal serial algorithm to determine the number of occurrentce of j $\in$ 1:k.
    - The number of steps executed is $\Theta(k) = \Theta(\log_2 n)$,
    - The cost of execution is $\Theta(mk) = \Theta(n)$.
- R[j] determined by summing each column Ĉ[1:m,j]:
    - Using balancing tree algorithm: cost $\Theta(m)$
    - Total cost is $\Theta(mk) = \Theta(n)$. → cost optimization

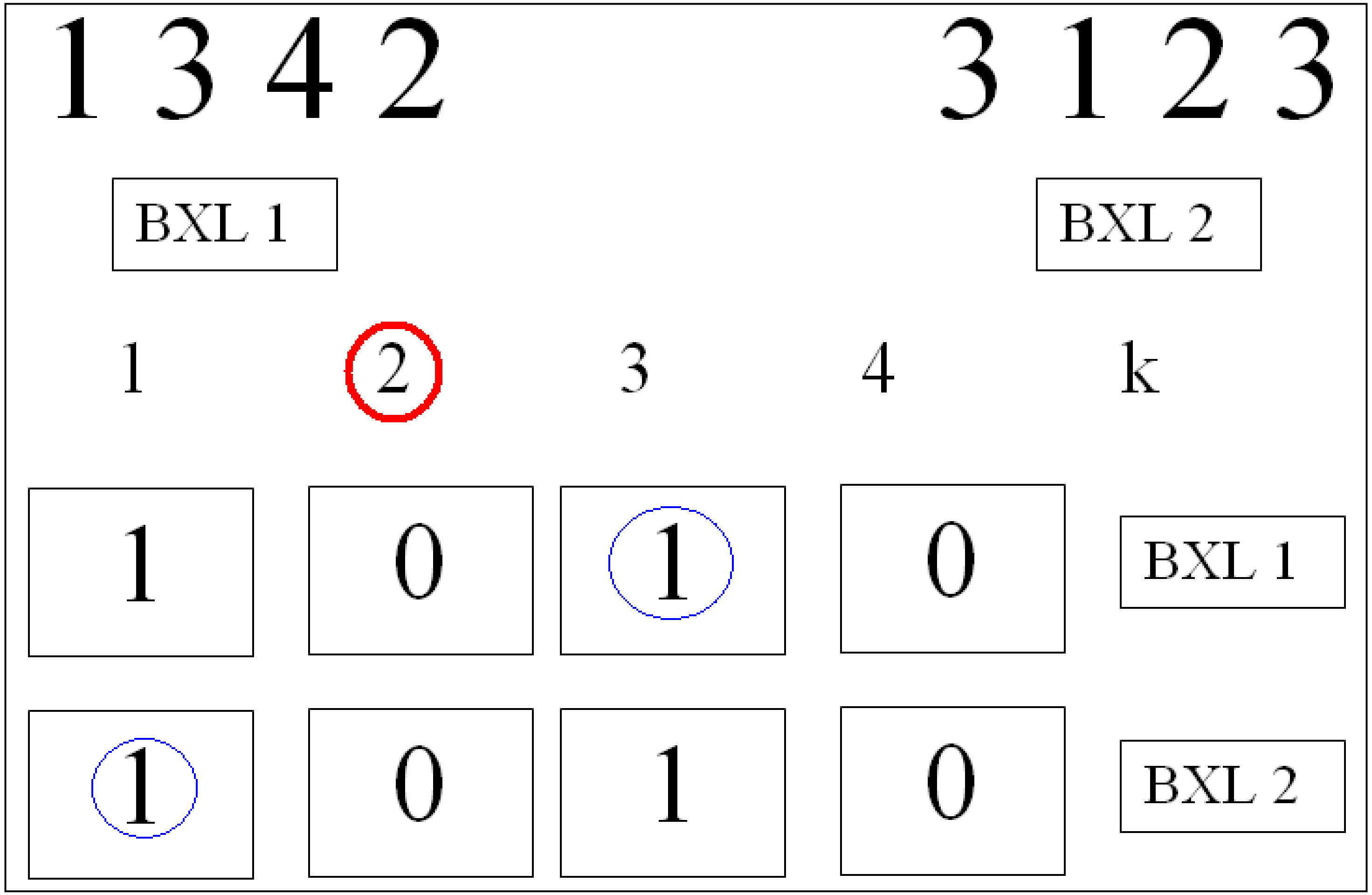
Using Accelerated Cascading



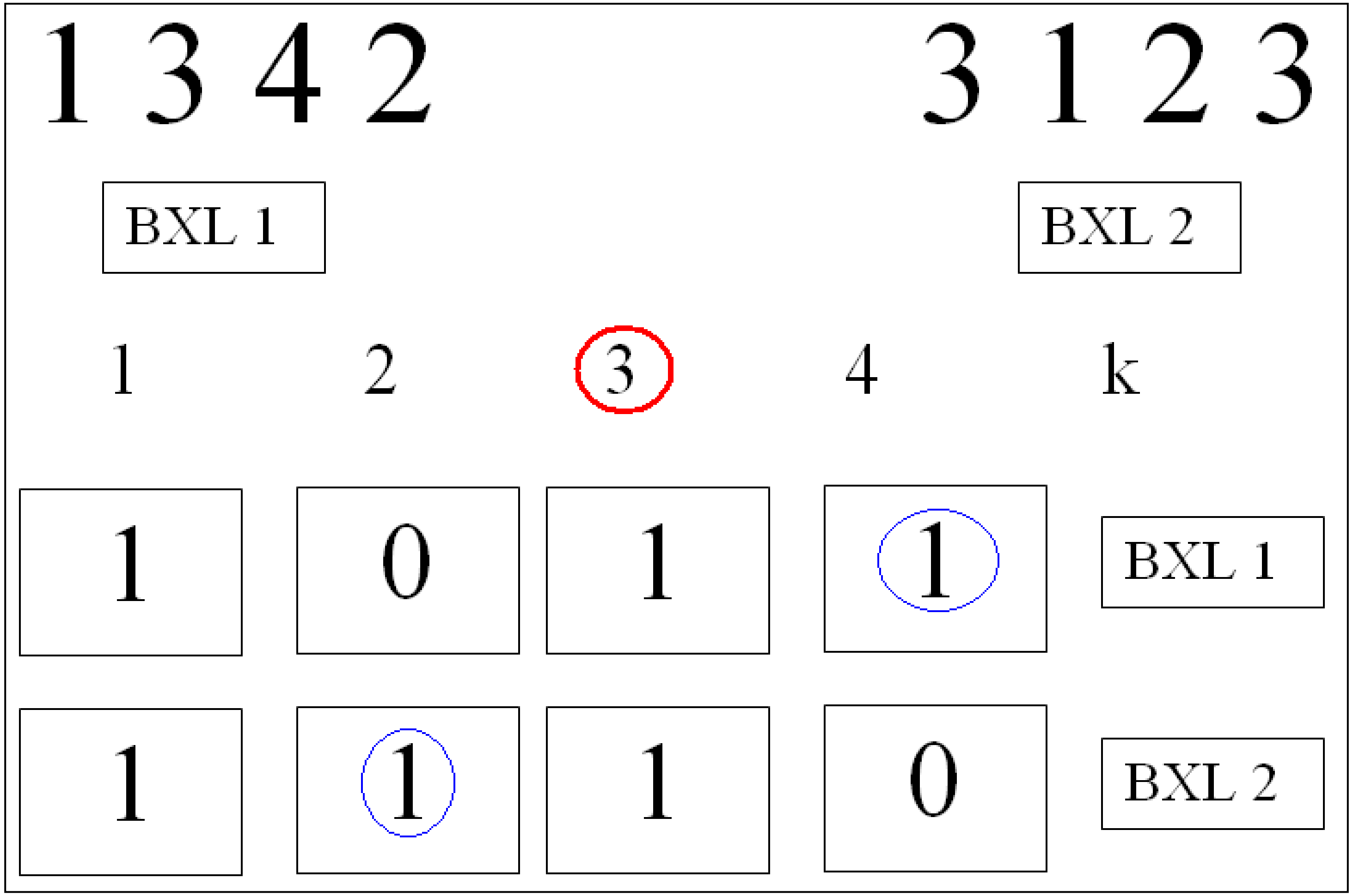
Using Accelerated Cascading



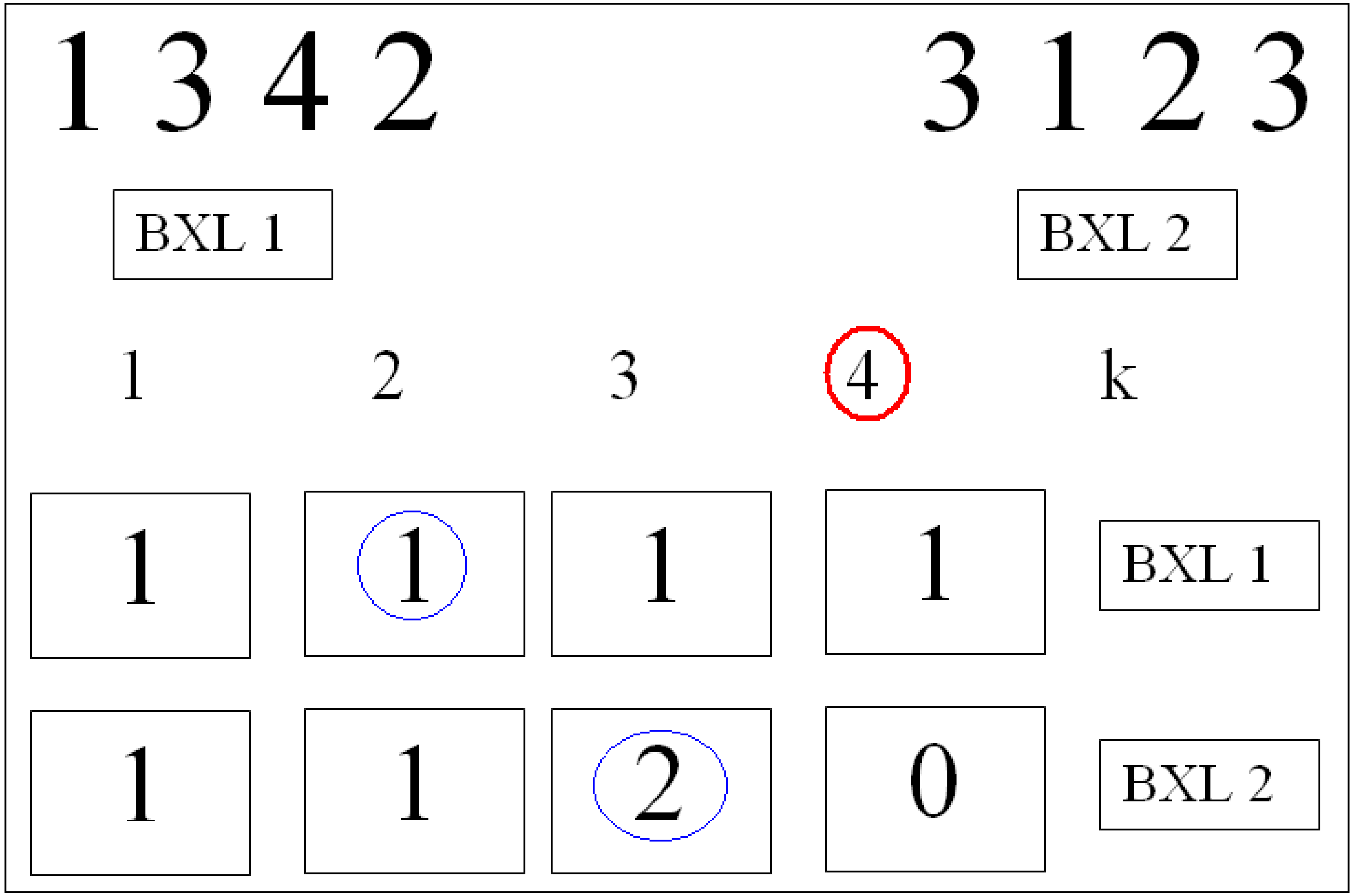
Using Accelerated Cascading



Using Accelerated Cascading



Using Accelerated Cascading



Optimal algorithm

Input: Sequence $L[1..n]$ with values in the range $1:k$

Output: Sequence $R[1..k]$ the occurrence counts for the values in L

integer $\hat{C}[1..m, 1..k]$
forall $i \in 1:m, j \in 1:k$ **do**
    $\hat{C}[i, j] \leftarrow 0$
enddo
forall $i \in 1:m$ **do**
    for $j = 1$ **to** k **do**
        $\hat{C}[i, L[(i-1)k + j]] \leftarrow \hat{C}[i, L[(i-1)k + j]] + 1$
    enddo
enddo
forall $j \in 1:k$ **do**
    $R[j] \leftarrow \text{REDUCE}(\hat{C}[1:m, j], +)$
enddo

Example (2): find Max

- Determine $X_i = \max \{ X_1, X_2, ..X_n \}: X_i \geq X_j \ \forall\ j \in 1{:}n$.
- Algorithm with PRAM EREW: $O(\log_2 n)$ step with the cost of O(n), using O(n) processors and balanced tree model.
- Consider the following algorithm with PRAM CRCW:

Input: *An array A of p distinct elements*

Output: *Boolean Array of M, M(i) =1 if and only if A(i) is the maximum element.*

begin

1. **for** $1 \leq i,j \leq p$ **pardo**
   if $A(i) \geq A(j)$ **then** *Set* $B(i,j):=1$
   else *Set* $B(i,j):=0$
2. **for** $1 \leq i \leq p$ **pardo**
   Set $M(i):=B(i,1) \cap ... \cap B(i,p)$

end.

Find Max with PRAM CRCW

- The algorithm above has 2 parts:
  - Part 1 can be done in parallel using n^2 processor with O(1) steps → Costs $O(n^2)$.
  - Part 2 can be done with PRAM CRCW: determine the value M[i] needs O(1) step, cost O(n) → total cost: O(n).
  - If PRAM CRCW is selected, the problem is quickly resolved with O(1) steps with $O(n^2)$ processors, hence the total cost is $O(n^2)$.

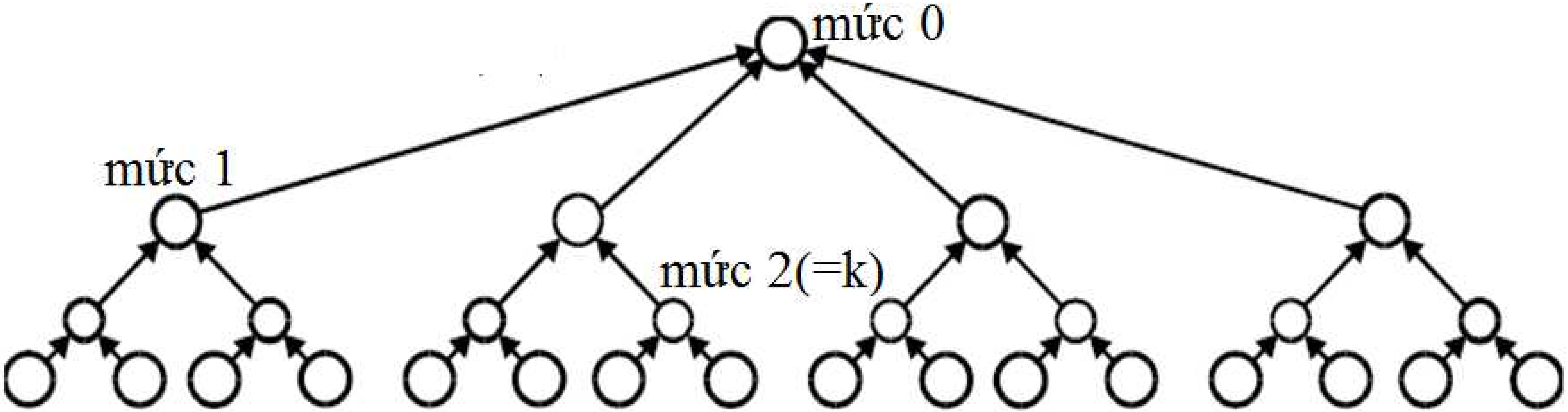
Find Max: Accelerated Cascading

- Let (W(n),T(n)) be the number of operation and time duration of the algorithm.
- Find-Max problem:
  - (1) Balance-Tree with EREW: (O(n), O(log2n)): optimal but slow
  - (2) Use CRCW with n2 processors: (O(n2), O(1)) : not optimal but fast.
- (3) Build a new algorithms on the DLDT tree with $(O(n.\log_2\log_2 n),O(\log_2\log_2 n))$;
- Apply (1) and (3) with Accelerated Cascading technique → new algorithm : $(O(n), O(\log_2\log_2 n))$.

Tree DLDT

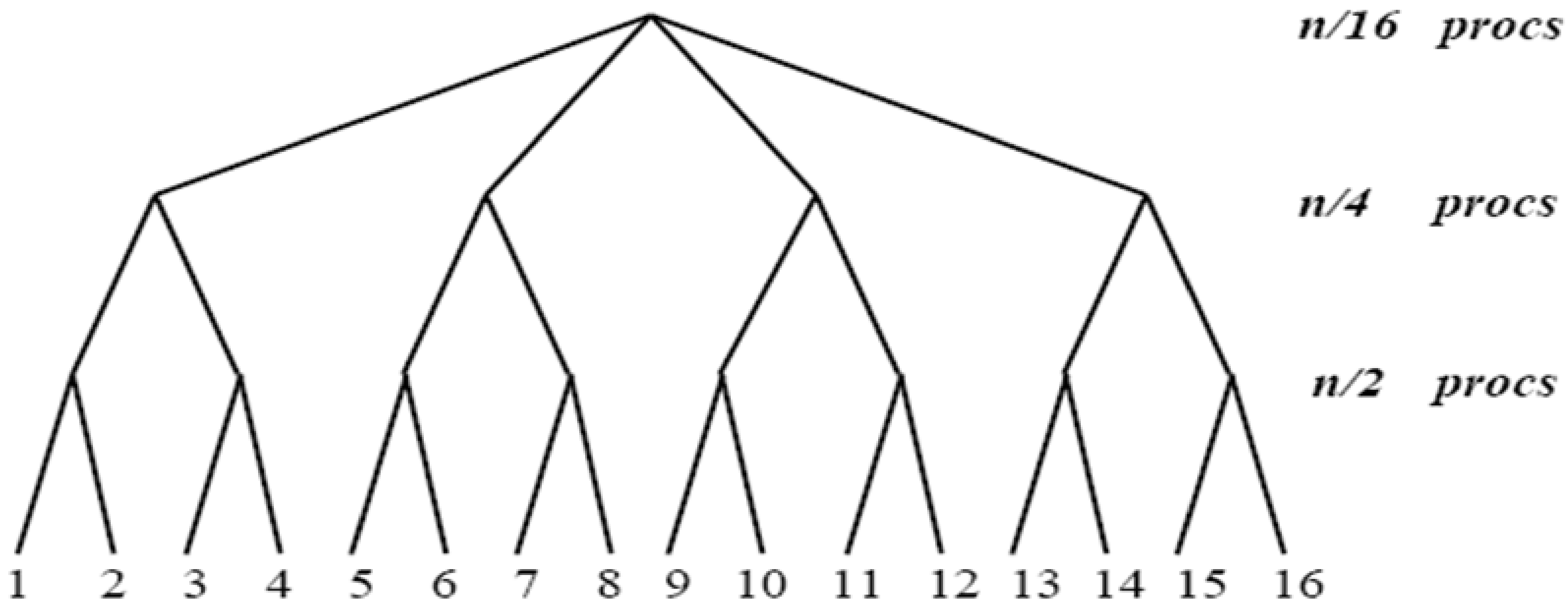
- DLDT = Doubly Logarithmic Depth Tree.
- This is recursive tree.
  - DLDT(n) is a tree with n leaves. ($n = 2^{2^k}$).
  - With k = 0 → n = 2 → tree has 1 root and 2 leaves.
  - With k > 0, tree is recursively constructed as follows:
    - Root has $\sqrt{n} = 2^{2^{k-1}}$ sub-trees.
    - Each sub-tree has $\sqrt{n}$ leaves : DLDT($\sqrt{n}$).
- Comment: The number of leaves in the tree with the root node at level i equals the number of leaves in the tree with the root node at level i+1.

Tree DLDT



- Degree of node u is the number of child nodes of u.
- Thanks to $n = 2^{2^k}$ then the root has degree of $2^{2^{k-1}} = \sqrt{n}$
- Node at level i has degree of $2^{2^{k-i-1}}$ với $0 \leq i < k$.
- Node at level k-1 has 2 child nodes
- Node at level k has 2 leaves

Tree DLDT



- The depth of the tree is : $k+1 = \log_2 \log_2 n + 1$.
- Let n be the number of the leave of the DLDT tree

Find-Max: DLDT Tree

- Comments:
  - At level 0 (root) we have n processors used to determine the Maximum value of obtained results returned from the child node ($\sqrt{n}$) at level 1. The algorithm can be applied using PRAM CRCW with (O(n),O(1)).
  - At level 1, dividing n processors into $m = \sqrt{n}$ groups. Each group corresponds to 1 node; we determine the Maximum value from $\sqrt{m}$ child node at level 2. The algorithm can be applied using PRAM CRCW to each node with (O(m),O(1));
  - At level i, each node corresponds to 1 group $2^{2^{k-i}}$ processors. Each node is the father of $2^{2^{k-i-1}}$ child nodes. The algorithm can be applied using PRAM CRCW to each node with $(O(2^{2^{k-i}}),O(1))$.

Find-Max: DLDT Tree

- Algorithm's idea:
  - Done with k repeating steps (from level k-1 = loglogn -1 to level 0 (root node)).
  - Executing the algorithm from the bottom to up.
  - At the i_th iteration step:
    - Perform in parallel with n processors divided into: $2^{2^k-2^{k-i}}$ groups, each group contains $2^{2^{k-i}}$ processors (because each child node is assigned to 1 processor =>processor group's number = child node group's number).
    - Using algorithms with CRCW for each node, the largest value of child nodes is stored at parent nodes.

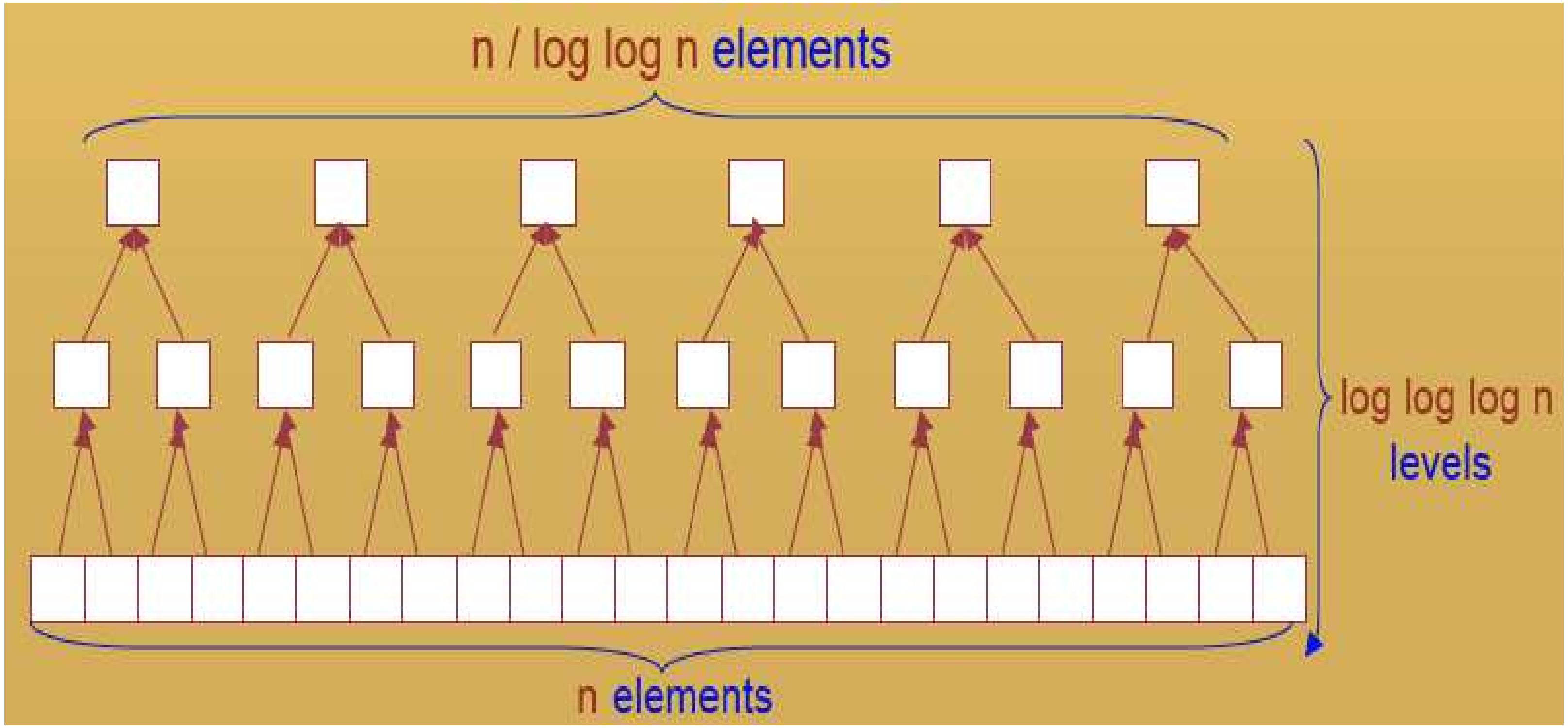
Find-Max: DLDT tree

- Performance evaluation:
  - Time duration: $O(k)=O(\log_2\log_2 n)$ repeating steps.
  - Cost evaluation:
    - Each node in i_th iteration step performs $O(2^{2^{k-i}})$ operations or $O(n^{2^{-1}})$
    - At i_th step, there is $2^{2^{k}-2^{k-i}} = n^{1-2^{-i}}$ nodes → total cost at each i_th iteration step: $W_i(n) = O(n^{1-2^{-1}} \text{x } n^{2^{-1}}) = O(n)$.
    - The cost of the entire algorithm is:

$W(n) = k*W_i(n)= O(n.k) = O(n.\log_2\log_2 n)$.

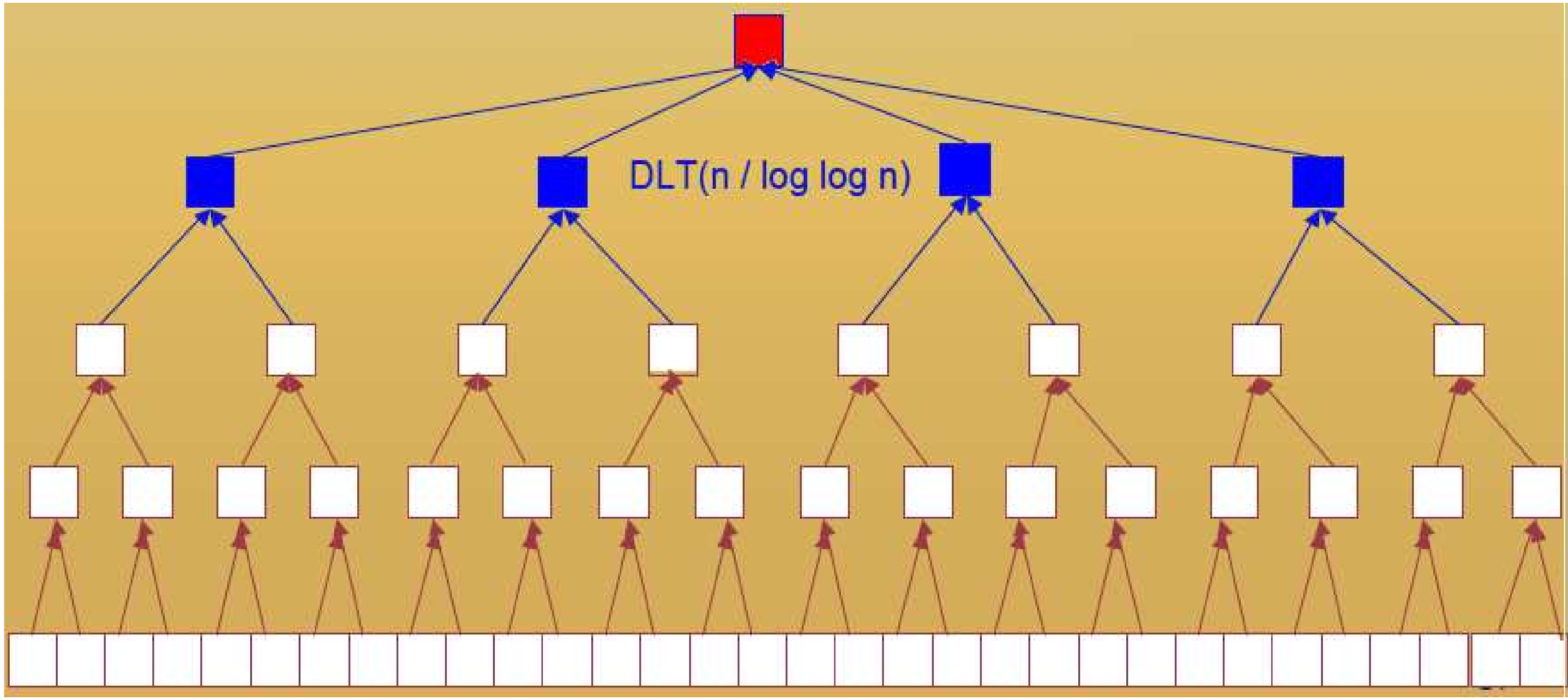
Using Accelerated Cascading

- Step 1: Using balancing tree technique with [logloglogn] level from the bottom.



Using Accelerated Cascading

- Step 2. Performing the algorithm on the DLDT tree with the number of nodes $m = n/\log_2\log_2 n$.



Using Accelerated Cascading

- Step 1. Using balanced tree:
  - After each step of the balancing tree, the number of nodes decreases by 1/2 (done with $\log_2\log_2\log_2 n$ serial steps =>T(n) = $\log_2\log_2\log_2 n$).
  - With k= $\log_2\log_2\log_2 n$, after this step the remaining number of nodes equals $m = n/2^k = n/\log_2\log_2 n$.
  - Cost of step 1: W(n) = O(n).
- Step 2. Using DLDT algorithm to the m other nodes:
  - $T(n) = O(\log_2\log_2 m) = O(\log_2\log_2 n)$.
  - $W(n) = O(m \text{ x } \log_2\log_2 m) = O(m \text{ x } \log_2\log_2 n) = O(n)$.
- Conclusion: New algorithm (O(n), $O(\log_2\log_2 n)$).

8.3 Pipeline

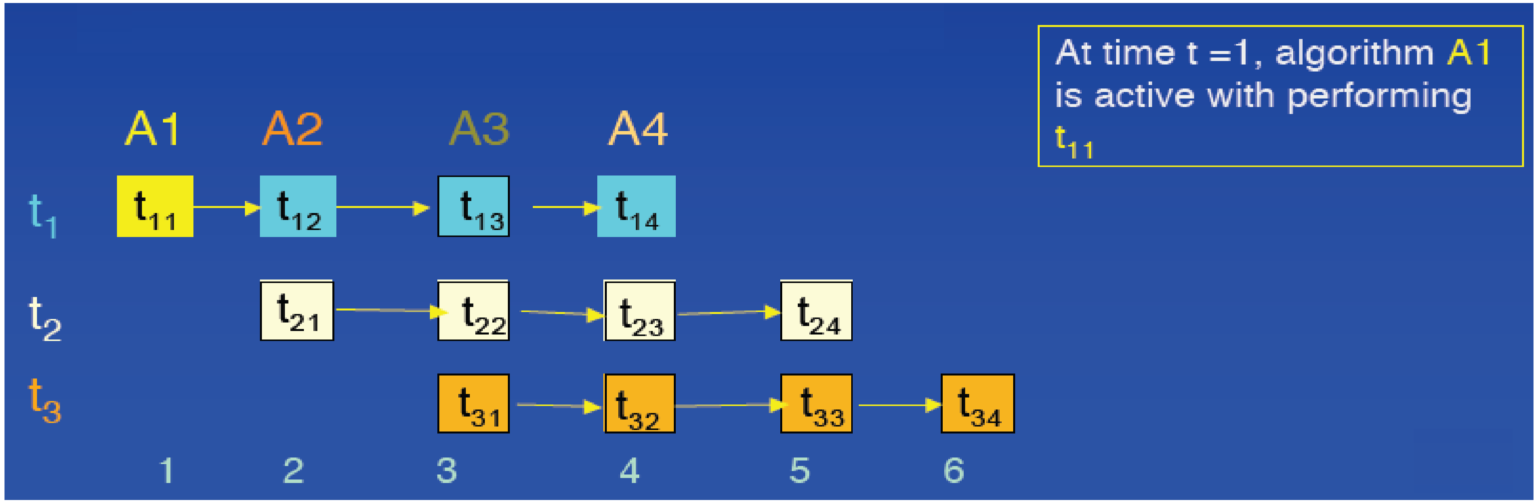
Pipeline Technique

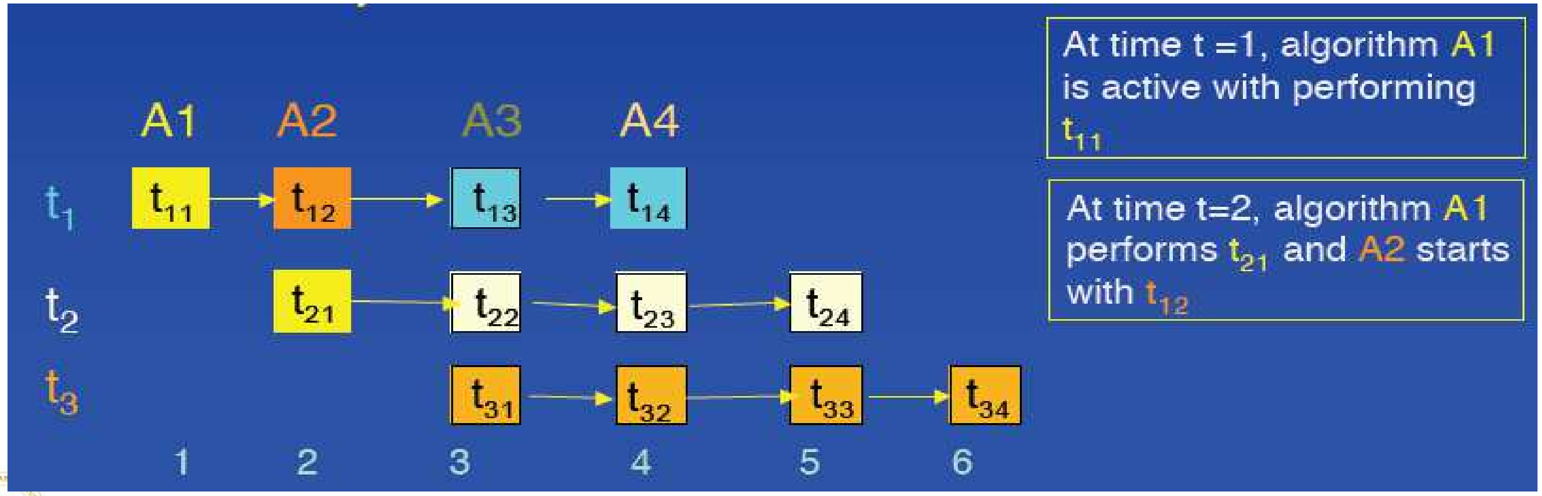
- Widely used to accelerate the executing time of many problems, including:
  - Each main problem can be divided into several child problems.
  - These sub-problems can depend on each other in an order of execution.
  - At each moment, the processors perform an algorithm for sub-problems of each main problem in parallel (make sure the execution order is constant).

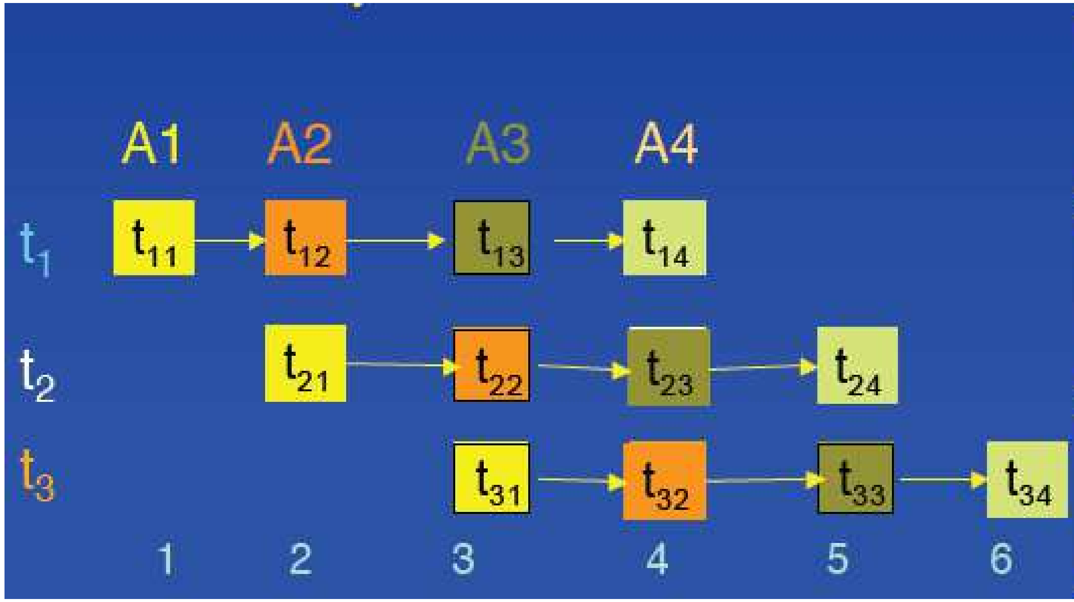
Pipeline Mechanism

- Considering n problems: $t_1, t_2, \ldots, t_n$ need to do.
- Each t_i can be divided into a set of sub-problems: $\{t_{i,1}, t_{i,2}, \ldots, t_{i,m}\}$ so that $t_{i,k}$ must be terminated before starting $t_{i,k+1}$.
- Assuming that at each step j = 1..m, algorithm step A_j will be done with sub-problems: $t_{1,j}, t_{2,j}, \ldots, t_{n,j}$.

Pipeline Mechanism



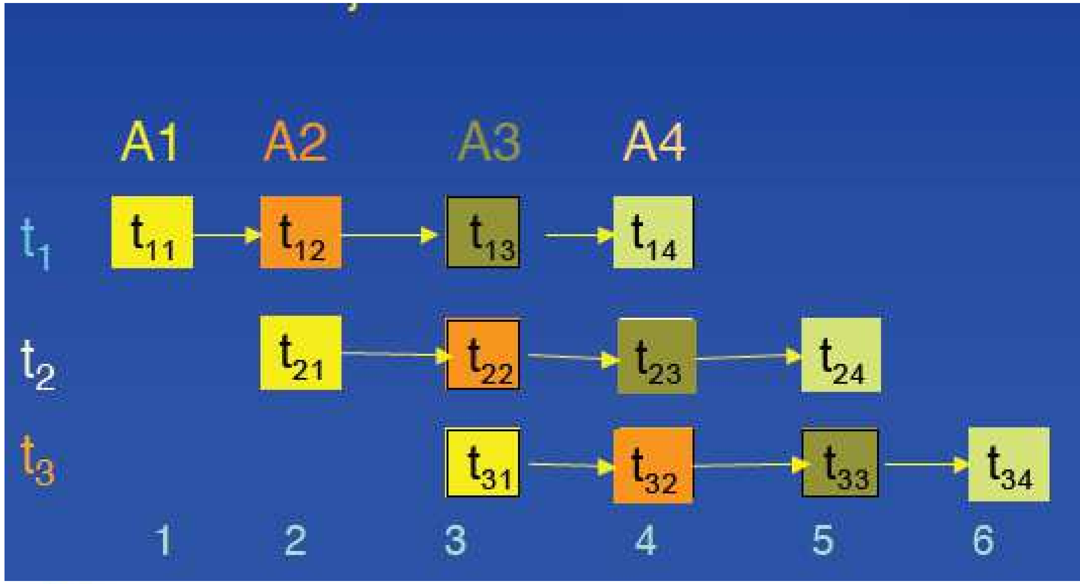




At time t=1, algorithm A1 is active with performing t_{11}

At time t=2, algorithm A1 performs t_{21} and A2 starts with t_{12}

Last subtask completed after execution of m + (n -1) subtasks



Suppose that there are m subtasks per task, n tasks and time O(t) per subtask.

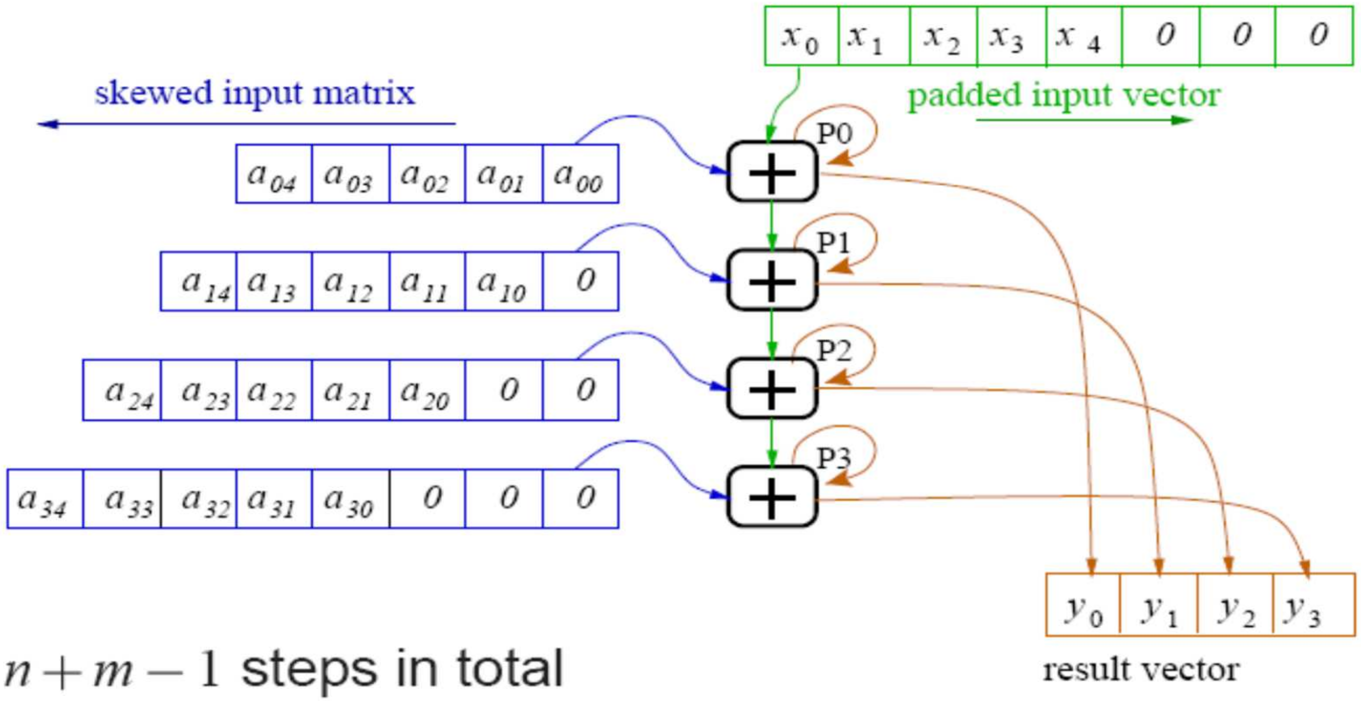
without pipelining:
$T = O(m.n.t)$

with pipelining:
$T = O((n+m) t)$

Example: matrix by vector

multiply matrix $A \in \mathbb{R}^{n,m}$ by vector $x \in \mathbb{R}^{m}$ $\rightarrow$ vector $y \in \mathbb{R}^{n}$

$$y_i = \sum_{j=0}^{m-1} a_{ij} x_j \qquad i = 0, \ldots, n-1$$



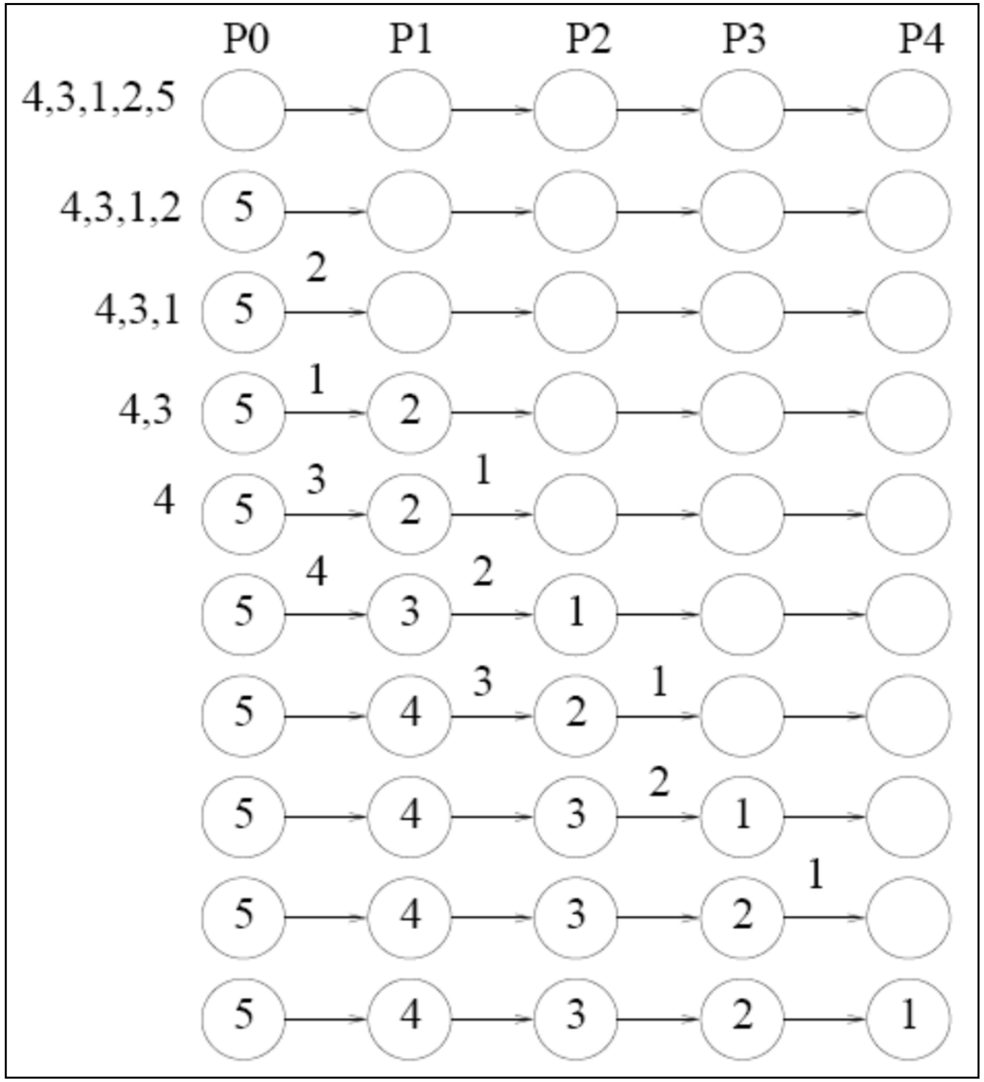
$n + m - 1$ steps in total

Example: multiply the matrix by the vector

- Illustrating with: matrix 4x4 and vector 4x1.
- Step1:
  - P_0 received (X_0 and A_{00});
  - Calculate the product of X_0 and A_{00}, then saved to Y_0 .
  - Push X_0 down to P_1 and move X_1 to the top of the range.
- Step 2:
  - P_0 received (X_1 and A_{01});P_1 received (X_0 and A_{10});
  - Calculate the corresponding products and save to Y_0 and Y_1 .
  - Push X_0, X_1 down, move X_2 to the top of the range, ...repeat until Step 8.

Parallel Insertion Sort

- Algorithm idea:
  - Values that will be sorted go into the range of processors one by one.
  - At each processor:
    - Read the value just received.
    - Compare with current value.
    - Move a value to the next processor.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions!