BÀI 6. CÁC KỸ THUẬT CƠ BẢN

NỘI DUNG BÀI HỌC

- Binary Tree Paradigm
- Growing by Doubling
- Pointer Jumping
- Partitioning
- Devide and Conquer
- Pipelining
- Accelerating Cascading
- Symmetry Breaking

1. BINARY TREE PARADIGM

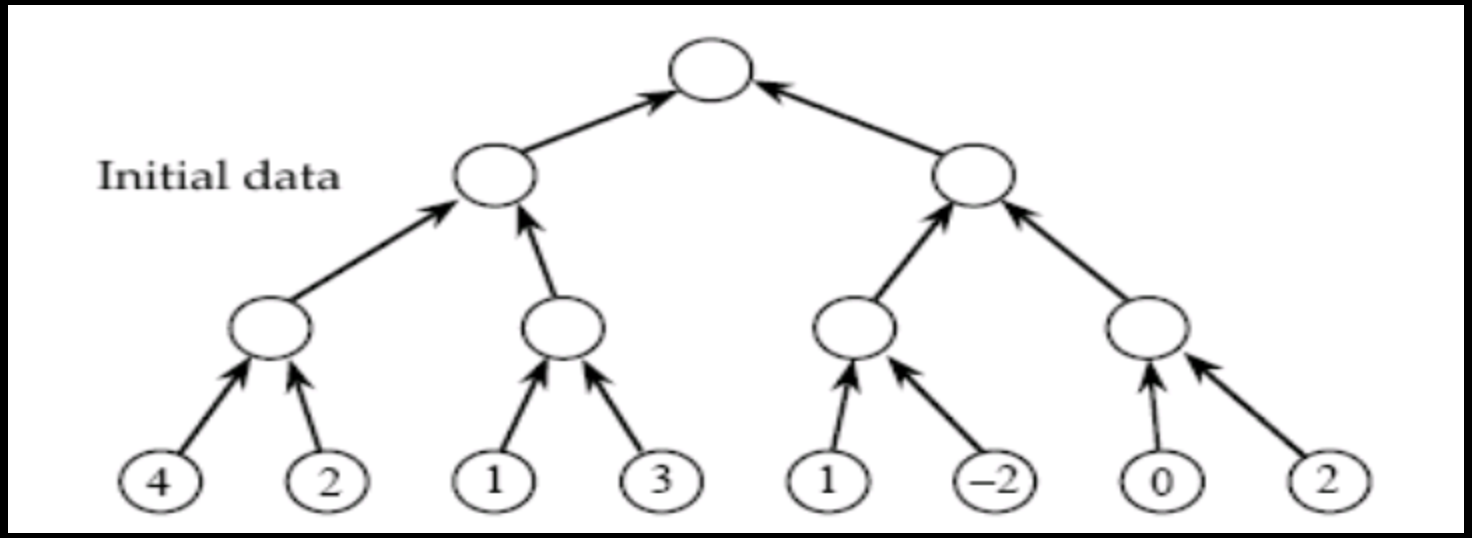
BINARY TREE PARADIGM

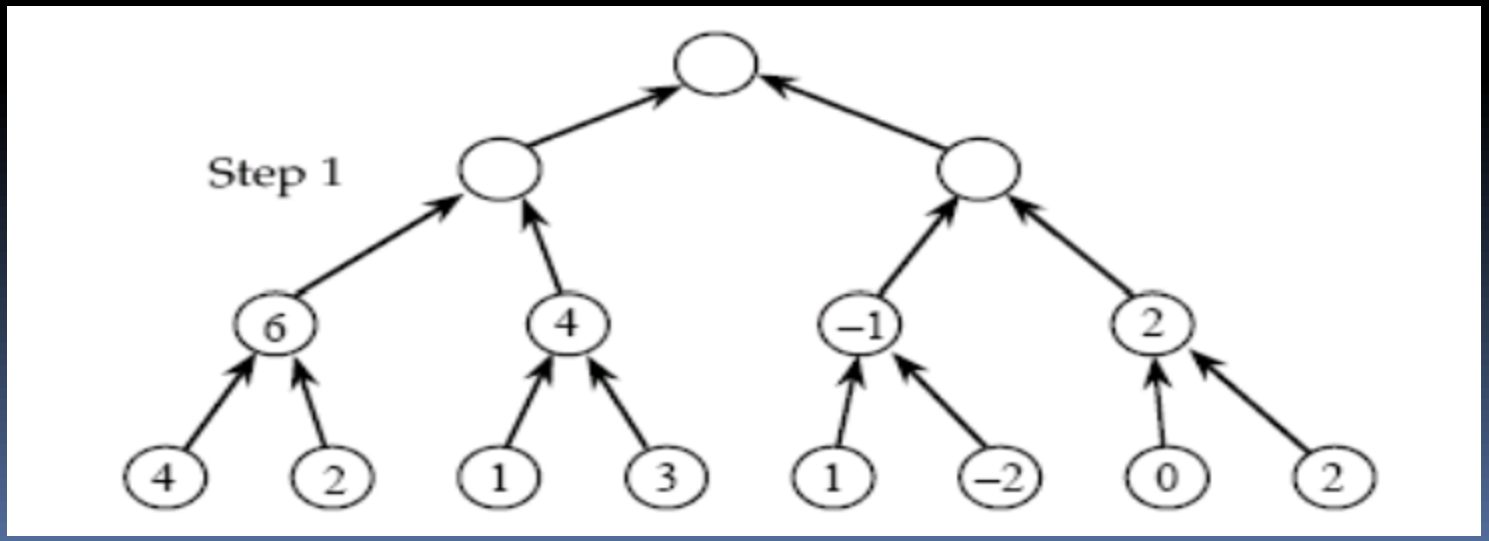
- Còn được gọi là mô hình cây cân bằng.
- Đặc điểm:
  - Các nút biểu diễn một thao tác (phép toán)
  - Các nút trên cùng mức thực hiện song song.
  - Dữ liệu đầu vào của các nút là kết quả của các phép toán ở mức thấp hơn và được lấy từ bộ nhớ dùng chung.
  - Sau khi thực hiện mỗi nút sẽ ghi kết quả ra bộ nhớ dùng chung.

VÍ DỤ MINH HỌA

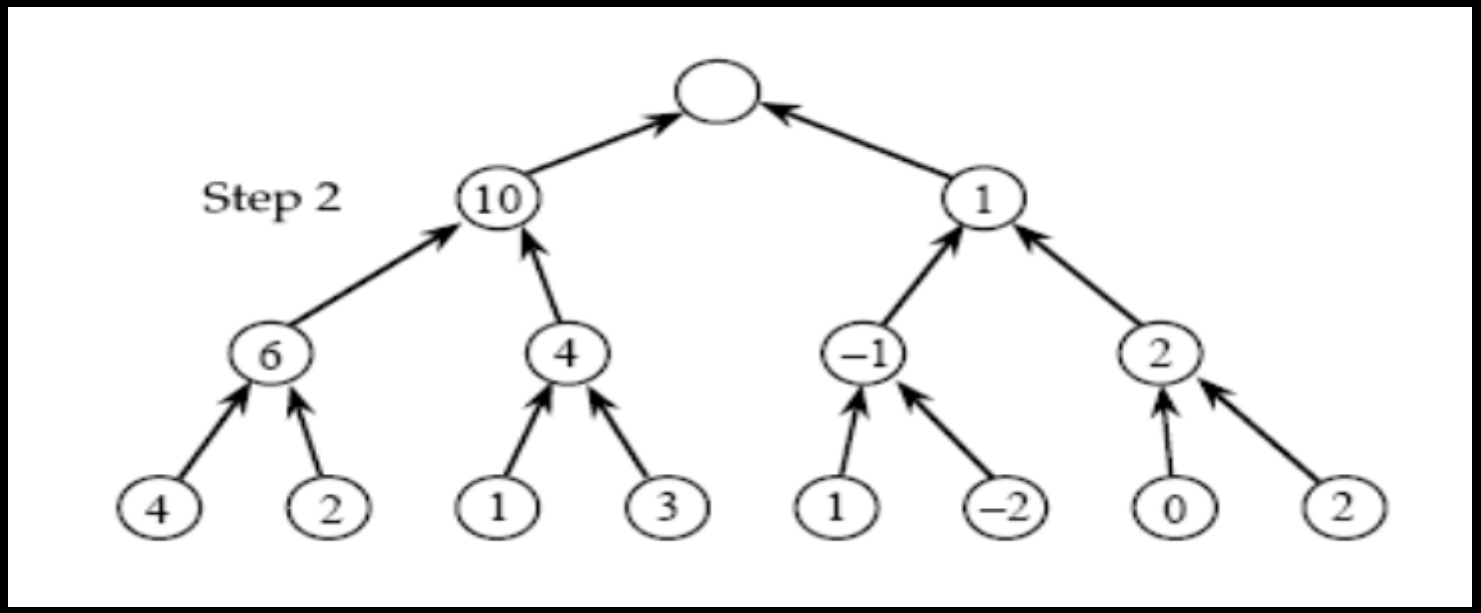
- Bài toán tính tổng n số $n = 2^k$.
- Phân tích:
  - Thực hiện tuần tự ???
  - Song song hóa với nhiều BXL:
    - Dữ liệu độc lập
    - Sử dụng tối đa n/2 BXL để thực hiện cộng các cặp song song.

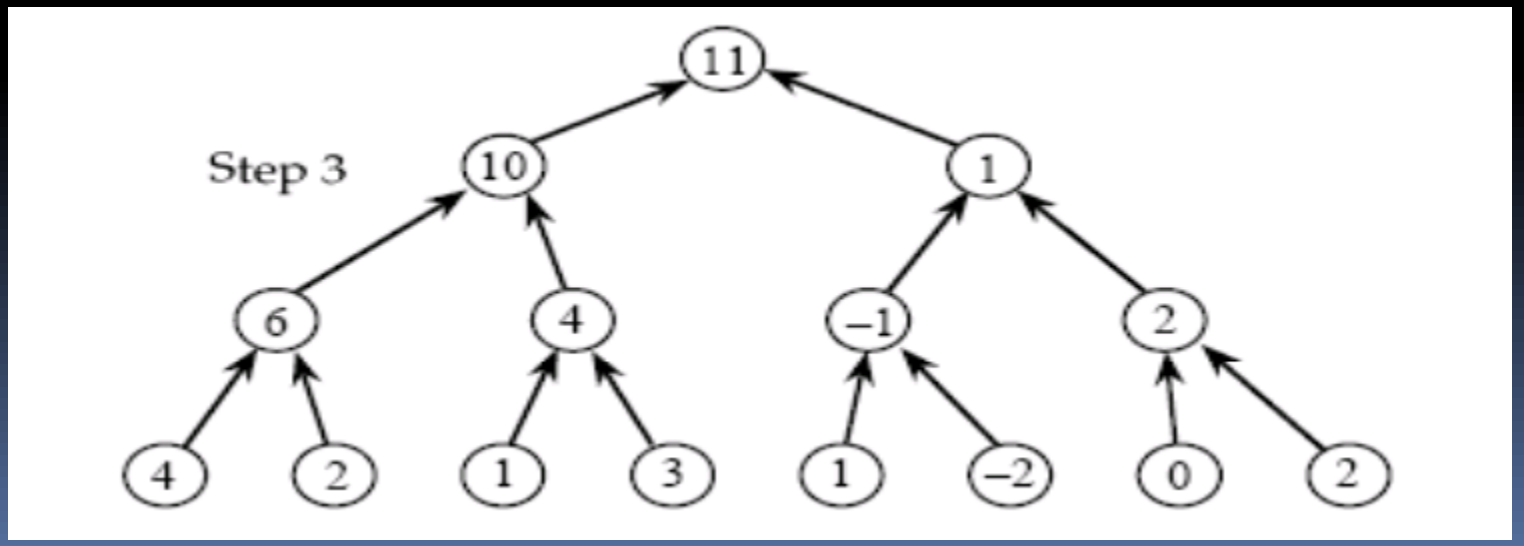
TÍNH TỔNG 8 SỐ





TÍNH TỔNG 8 SỐ



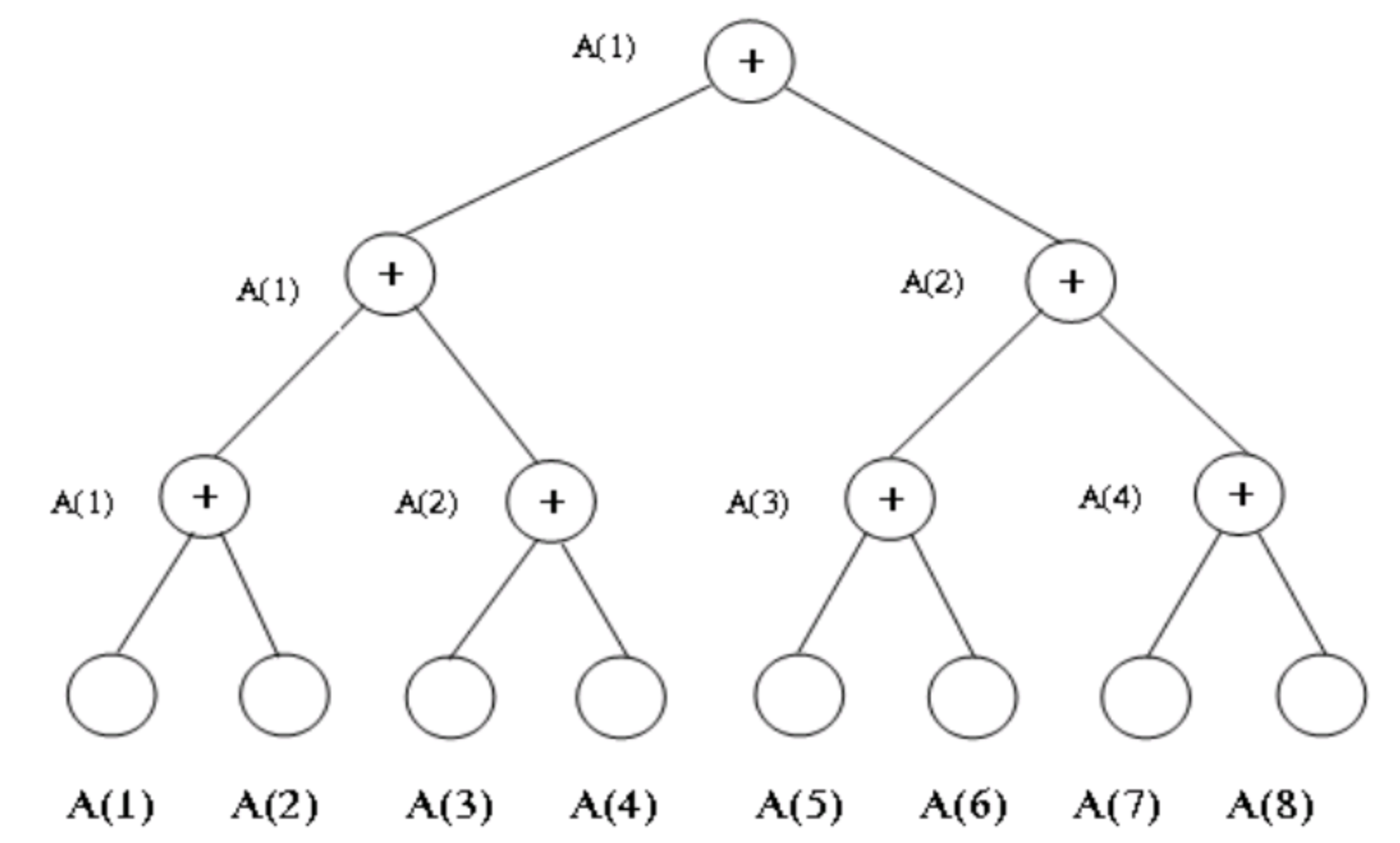


TÍNH TỔNG 8 SỐ

- Ý tưởng: A[i] = A[2*i-1] + A[2*i]
- Lập lịch thực hiện cho các BXL theo thời gian
- Số bước tuần tự ?

SCH	Ý Nghĩa
SCH(1) = (1,1)	Bộ xử lý 1 thực hiện tại thời điểm 1
SCH(2) = (2,1)	Bộ xử lý 2 thực hiện tại thời điểm 1
SCH(3) = (3,1)	Bộ xử lý 3 thực hiện tại thời điểm 1
SCH(4) = (4,1)	Bộ xử lý 4 thực hiện tại thời điểm 1
SCH(5) = (1,2)	Bộ xử lý 1 thực hiện tại thời điểm 2
SCH(6) = (2,2)	Bộ xử lý 2 thực hiện tại thời điểm 2
SCH(7) = (1,3)	Bộ xử lý 1 thực hiện tại thời điểm 3

TÍNH TỔNG 8 SỐ



TÍNH TỔNG 8 SỐ

- Thuật toán

```
INPUT          :        A[1…n];
OUTPUT         :        SUM = ∑ A[i];
BEGIN
        p      =        n/2;
        WHILE p > 0 DO
                FOR i = 1 TO p DO IN PARALLEL
                        A[i]  =  A[2i-1] + A[2i];
                END PARALLEL;
                p      =        p/2;
        END WHILE;
END.
```

- Đánh giá hiệu quả: $O(Log_2 n)$ so với O(n) 1 BXL
- Máy thực hiện PRAM EREW

MỘT SỐ VÍ DỤ KHÁC

- Bài toán Boolean – AND:
  - Chỉ thay phép toán + bằng phép toán AND.
  - Code ví dụ bài 3
- Bài toán tính tích vô hướng 2 vector:
  - Có 2 bước song song:
    - Nhân từng cặp song song với n BXL
    - Cộng các tích con theo mô hình cây cân bằng

TÍNH TÍCH VÔ HƯỚNG 2 VECTO

- Thuật toán:
  - Độ phức tạp: $O(Log_2 n)$ với n BXL
  - Thực hiện trên máy PRAM EREW

```
INPUT          :        A[1..n], B[1..n];
OUTPUT         :        RESULT=          ∑(A[i]*B[i]);
BEGIN
        FOR i = 1 TO n DO IN PARALLEL
                C[i]     =        A[i] * B[i];
        END PARALLEL;
        FOR i = 1 TO log(n) DO
                FOR j = 1 TO n/2^l DO IN PARALLEL
                        C[j]     =        C[j] + C[j +  n/2^l];
                END PARALLEL;
        END FOR ;
END;
```

2. GROWING BY DOUBLING

GROWING BY DOUBLING

- Trình bày ngược lại với kỹ thuật trước
- Công thức chung cho cả 2 bước:
  - Xác định số bước lặp tuần tự
  - Xác định số BXL và các chỉ số cụ thể trên từng bước lặp.
  - Xác định công việc từng BXL trong từng bước tuần tự.

VÍ DỤ MINH HỌA

- Bài toán "Broadcast" trong PRAM
- Phát biểu bài toán:
  - Máy PRAM EREW với n BXL
  - P1 đang chứa giá trị x.
  - Viết thuật toán truyền giá trị x đến các BXL còn lại
- Khái niệm truyền thông tin trong PRAM
  - Bước 1: BXL a ghi dữ liệu vào ô nhớ M
  - Bước 2: BXL b đọc dữ liệu từ ô nhớ M

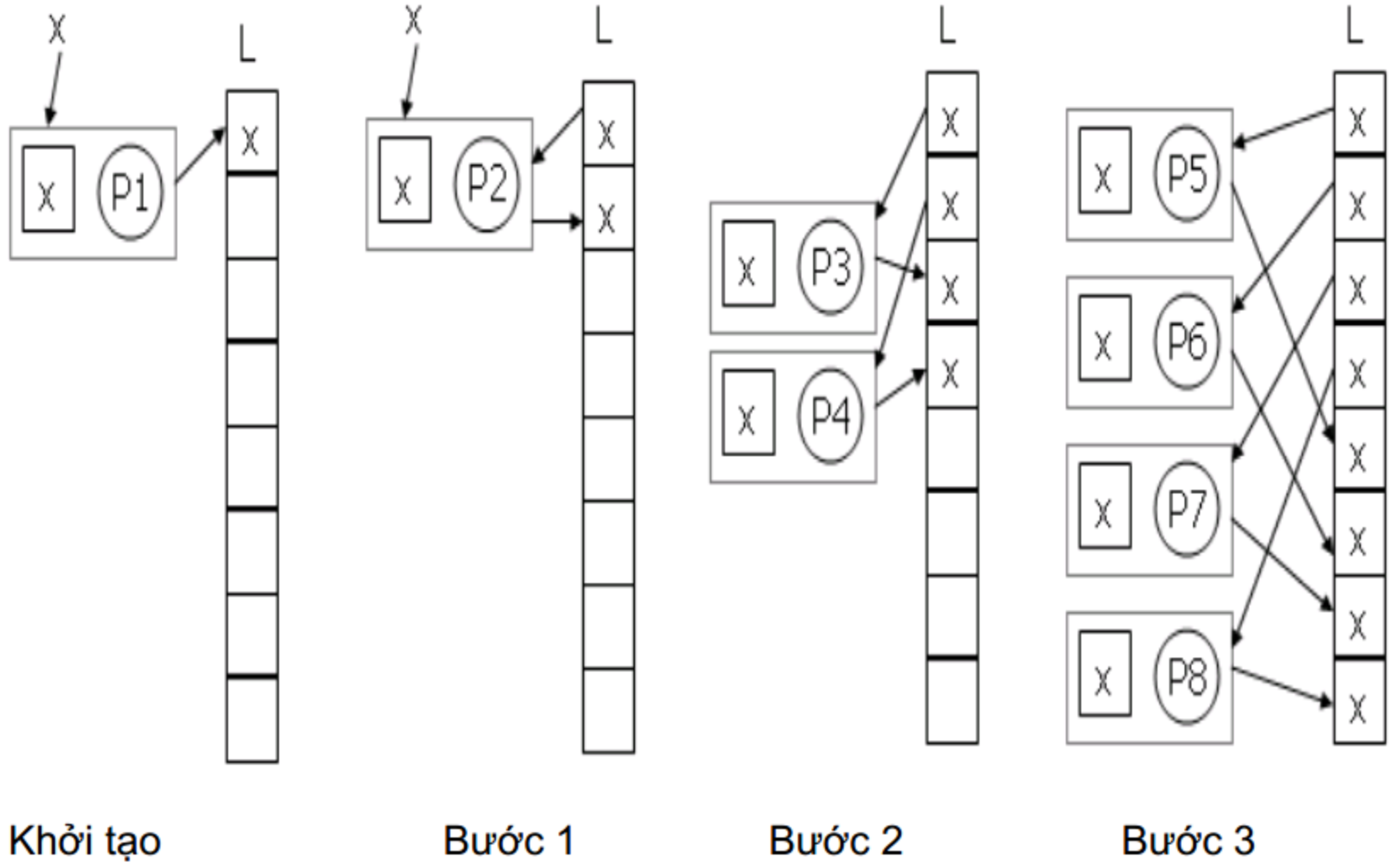
BROADCAST TRONG PRAM

- Thuật toán đơn giản nhất:
  - BXL P1 ghi giá trị x vào 1 ô nhớ M
  - PRAM EREW -> tại 1 thời điểm chỉ 1 BXL được đọc dữ liệu từ 1 ô nhớ.
  - Do đó các BXL đọc lần lượt là O(n)

BROADCAST TRONG PRAM

- Ý tưởng giải thuật song song:
  - B1: P1 ghi giá trị x vào ô nhớ m1
  - B2: Chia làm 2 giai đoạn:
    - P2 đọc dữ liệu từ m1 do đó P2 cũng có x.
    - P2 ghi x vào các ô nhớ m2
  - B3: Chia làm 2 giai đoạn:
    - P3, P4 đọc dữ liệu từ m1, m2
    - P3, P4 ghi dữ liệu vào m3, m4
  - B4: Chia làm 2 giai đoạn:
    - P5,…P8 đọc dữ liệu từ m1,..m4
    - P5… P8 đọc dữ liệu từ m5..m8
- Sau mỗi bước BXL tăng gấp 2 lần

BROADCAST TRONG PRAM



BROADCAST TRONG PRAM

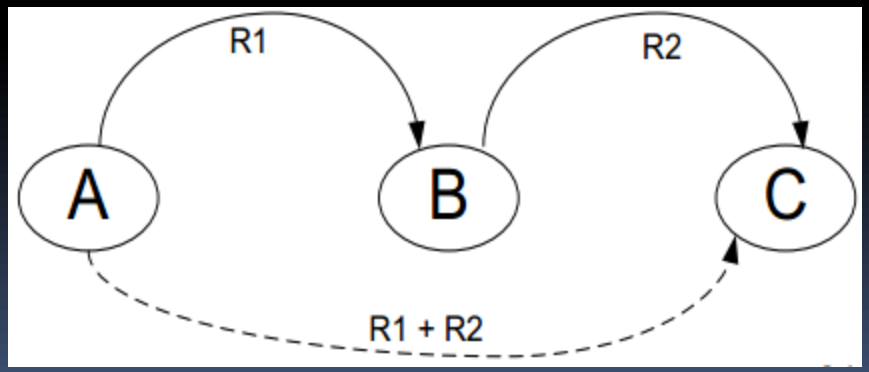
- Thuật toán song song:
  - Độ phức tạp: $O(Log_2 n)$ với n BXL
  - Máy PRAM EREW

```
INPUT          :      P1 được kích hoạt.
OUTPUT         :      P1,P2,.. Pn đều chứa giá trị x.
BEGIN
        P1:    y      =      x;
               L[1]   =      y;
        FOR  k = 0 TO log(n) -1 DO
               FOR i = 2^k + 1 TO 2^(k+1) DO IN PARALLEL
                      Pi:    y      =      L[i – 2^k];
                             L[i]   =      y;
               END PARALLEL;
        END FOR;
END.
```

3. POINTER JUMPING

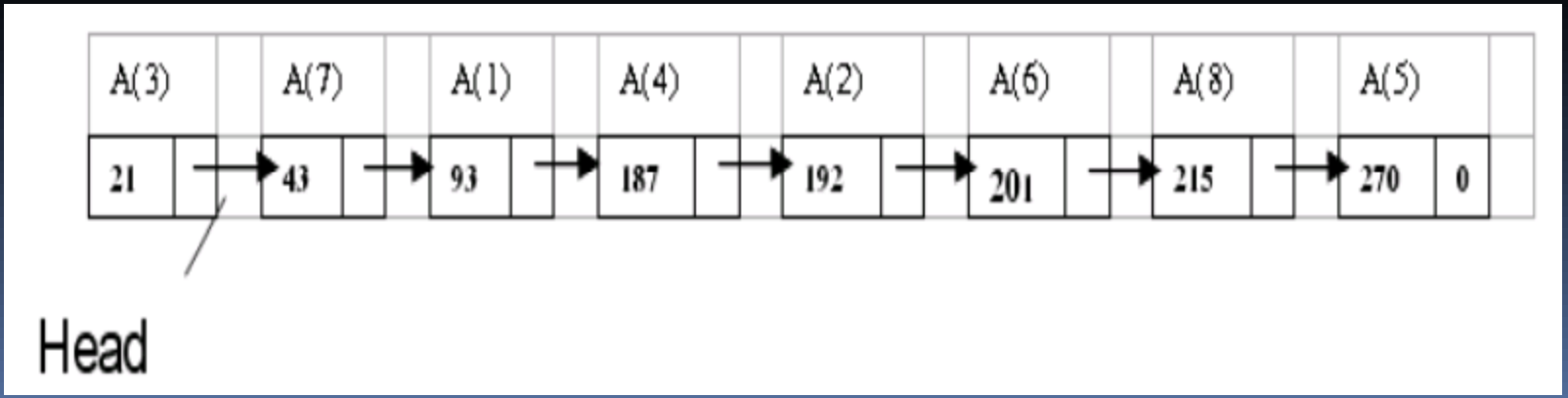
POINTER JUMPING

- Ý tưởng phương pháp:
  - Xét 3 nút trong 1 danh sách: A->B->C.
  - Gọi R1, R2 là hàm công việc từ A->B và từ B->C.
  - Khi đó từ A-> C là R3 = R1 + R2



POINTER JUMPING

- Giả sử dữ liệu đầu vào là 1 danh sách các phần tử liên kết theo 1 trật tự nào đó. Cần tính hạng của các phần tử trong mảng.
- Biểu diễn dữ liệu dưới dạng các mảng
  - Giá trị được lưu tại nút
  - Chỉ số của phần tử kế tiếp mà nút trỏ đến

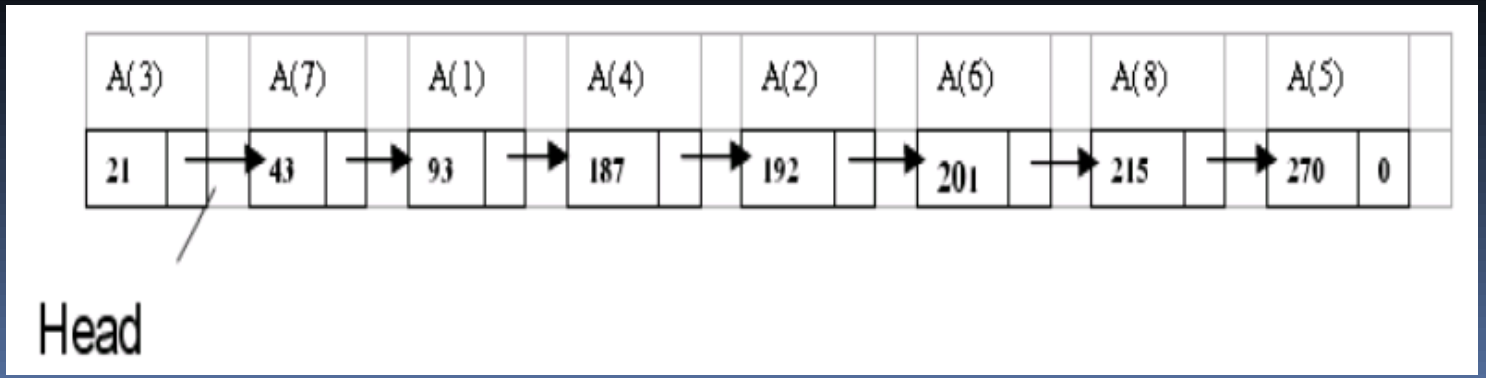


BÀI TOÁN XÁC ĐỊNH HẠNG

- Xét ví dụ với 8 phần tử:
  - Ta gọi LINK[i] là chỉ số của phần tử kế tiếp A[i].
  - LINK[i] = 0 có nghĩa là A[i] là phần tử cuối cùng trong danh sách liên kết.
  - Biến HEAD chứa chỉ số của phần tử đầu tiên.
  - Ta gọi hạng của một phần tử mảng trong danh sách là khoảng cách từ nó đến cuối.

BÀI TOÁN XÁC ĐỊNH HẠNG

HEAD = 3		
i	A	LINK
1	93	4
2	192	6
3	21	7
4	187	2
5	270	0
6	201	8
7	43	1
8	215	5



BÀI TOÁN XÁC ĐỊNH HẠNG

- Thuật toán tuần tự:
  - Độ phức tạp O(n) với 1 BXL

```
INPUT           :       A[1..n], LINK[1..n], HEAD.
OUTPUT          :       RANK[1..n].
BEGIN
        p       =       HEAD;
        r       =       n;
        RANK[p]         =       r;
        REPEAT
                p       =       LINK(p);
                r       =       r – 1;
                RANK[p]         =       r;
        UNTIL LINK(p) = 0;
END.
```

BÀI TOÁN XÁC ĐỊNH HẠNG

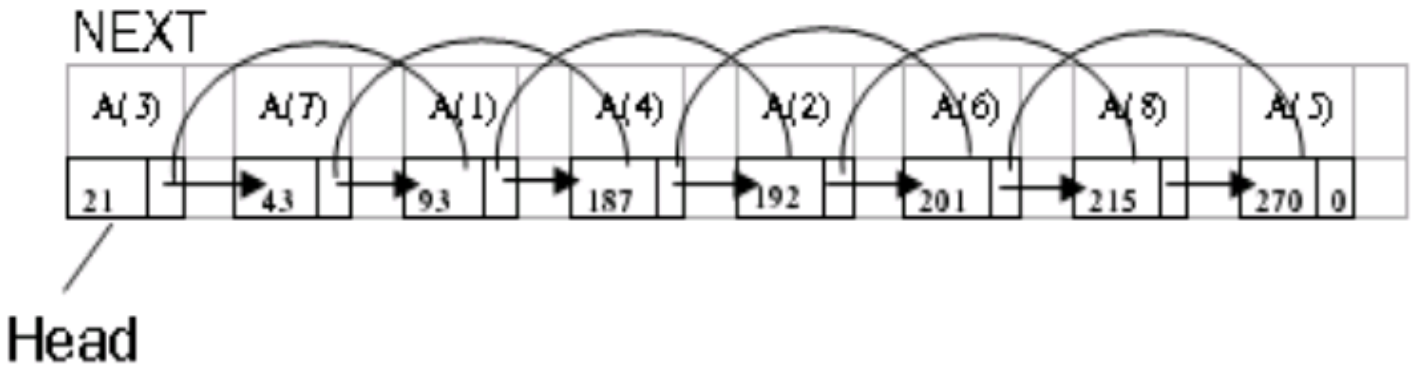
- Giải thuật song song:
  - Đặt NEXT[i] = LINK[i];
  - Hạng của các nút được xác định bằng khoảng cách mà nó có thể nhảy tới.
  - Tiếp theo NEXT[i] = NEXT[NEXT[i]]
  - Cập nhập giá trị hạng của các phần tử
  - Đồng thời với nó là khoảng cách nhảy tăng lên gấp đôi. Sau $Log_2(n)$ lần thì NEXT[i] sẽ đạt đến đích là phần tử cuối.
  - Khi tất cả NEXT[i] đạt đến phần tử cuối thì chương trình kết thúc

BÀI TOÁN XÁC ĐỊNH HẠNG

- Giải thuật song song:

i	3	7	1	4	2	6	8	5
LINK	7	1	4	2	6	8	5	0
NEXT	7	1	4	2	6	8	5	0
RANK	1	1	1	1	1	1	1	1

(a) Trạng thái khởi tạo

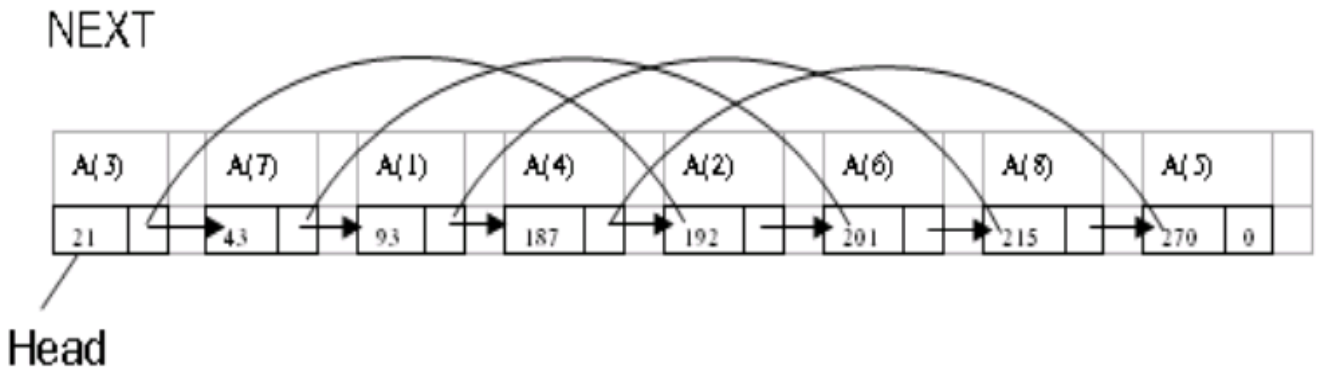


BÀI TOÁN XÁC ĐỊNH HẠNG

- Giải thuật song song:

i	3	7	1	4	2	6	8	5
LINK	7	1	4	2	6	8	5	0
NEXT	1	4	2	6	8	5	0	0
RANK	2	2	2	2	2	2	2	1

(b) Giai đoạn 1

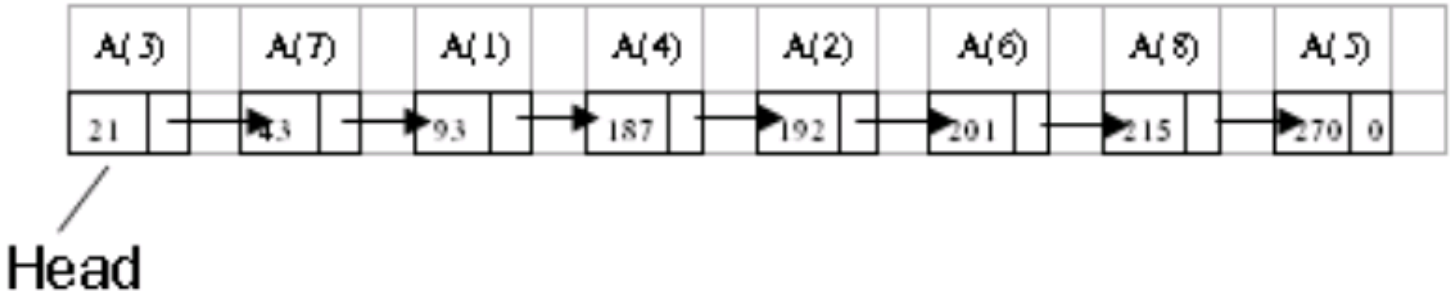


BÀI TOÁN XÁC ĐỊNH HẠNG

- Giải thuật song song:

i	3	7	1	4	2	6	8	5
LINK	7	1	4	2	6	8	5	0
NEXT	2	6	8	5	0	0	0	0
RANK	4	4	4	4	4	3	2	1

(c) Giai đoạn 2



BÀI TOÁN XÁC ĐỊNH HẠNG

i	3	7	1	4	2	6	8	5
LINK	7	1	4	2	6	8	5	0
NEXT	0	0	0	0	0	0	0	0
RANK	8	7	6	5	4	3	2	1

(d) Giai đoạn 3

- Bước thứ 3 kết thúc: các giá trị NEXT[i] = 0
- Bảng RANK[i] xác định hạng các phần tử trong danh sách liên kết.

GIẢI THUẬT SONG SONG

```
INPUT           :       A[1..n], LINK[1..n], HEAD;
OUTPUT          :       RANK[1..n];
BEGIN
        FOR i = 1 TO n DO IN PARALLEL
                RANK[i]         =       1;
                NEXT[i] =       LINK[i];
        END PARALLEL;
        FOR k = 1 TO log(n) DO
                FOR i = 1 TO n DO IN PARALLEL
                        IF NEXT[i] <> 0 THEN
                                RANK[i] = RANK[i] + RANK[NEXT[i]];
                                NEXT[i] = NEXT[NEXT[i]];
                        END IF;
                END PARALLEL;
        END FOR;
END;
```

GIẢI THUẬT SONG SONG

- Đánh giá độ phức tạp:
  - Thuật toán chia thành 2 phần tuần tự
    - Phần thứ 1: O(1) đơn vị thời gian
    - Phần thứ 2: $O(\log_2 n)$ đơn vị thời gian
- Kiến trúc máy tính PRAM EREW
  - RANK[i] = RANK[i] + RANK[NEXT[i]]
    - B1: X = RANK[i]
    - B2: Y = RANK[NEXT[i]]
    - B3: RANK[i] = X + Y

4. PARTITIONING

PARTITIONING

- Bài toán trộn : Cho A[1..n] và B[1..n] là hai mảng đã được sắp xếp. Thực hiện trộn hai mảng này thành một mảng C[1..2n] được sắp xếp.

THUẬT GIẢI TUẦN TỰ

- Tư tưởng của thuật toán tuần tự:
  - 3 chỉ số i, j, k trên 3 mảng A, B, C tương ứng.
  - Các giá trị của mảng C[k] = min{ A_i ; B_j }
  - Lướt hết các phần tử của A, B

THUẬT GIẢI TUẦN TỰ

```
INPUT          :    A1 ≤ A2 ≤ .... ≤ An  và  B1 ≤ B2 ≤ ... ≤ Bn .
OUTPUT         :    C[1..2n] = A[1..n] U B[1..n] : C1 ≤ C2 ≤ ...≤ C2n
BEGIN
        A[n+1] =  ∞; B[n+1] = ∞ ;
        i = 1; j = 1; k = 1;
        WHILE k ≤ 2n DO
                IF A[i] < B[j] THEN
                        C[k]    =       A[i];
                        i       =       i + 1;
                ELSE
                        C[k]    =       B[j] ;
                        j       =       j + 1;
                END IF;
                k       =       k + 1;
        END WHILE;
END.
```

THUẬT GIẢI SONG SONG

- Sử dụng kỹ thuật phân chia:
  - Phân chia mảng A thành r = n / $\log_2(n)$;
  - Mỗi nhóm có k = $\log_2(n)$ phần tử. , (k, r thuộc N)
- Như vậy ta có các nhóm như sau:
  - Nhóm NA_1: $A_1, A_2, \ldots, A_k$;
  - Nhóm NA_2: $A_{k+1}, A_{k+2}, \ldots A_{2k}$
  - …..
  - Nhóm NA_r: $A_{(r-1)k+1}, A_{(r-2)k+2}, \ldots, A_n$

THUẬT GIẢI SONG SONG

- Tìm r số nguyên j[1], j[2], … j[r] sao cho:
  - j[1] là chỉ số lớn nhất thỏa mãn $A_k >= B_{j[1]}$;
  - j[2] là chỉ số lớn nhất thỏa mãn $A_{2k} >= B_{j[2]}$;
  - ……
  - j[r] là chỉ số lớn nhất thỏa mãn $A_n >= B_{j[r]}$;
- Chia mảng B[1..n] ra thành các nhóm
  - Nhóm NB_1: B_1, B_2, ……, $B_{j[1]}$;
  - Nhóm NB_2: $B_{j[1]+1}$, $B_{j[1]+2}$, ……, $B_{j[2]}$;
  - …..
  - Nhóm NB_{r+1}:$B_{j[r]+1}$,……B_n

THUẬT GIẢI SONG SONG

A = (1, 5, 15, 18, 19, 21, 23, 24, 27, 29, 30, 31, 32, 37, 42, 49),

B = (2, 3, 4, 13, 15, 19, 20, 22, 28, 29, 38, 41, 42, 43, 48, 49).

A	1, 5, 15, 18	19, 21, 23, 24	27, 29, 30, 31	32, 37, 42, 49
Nhóm	Nhóm 1	Nhóm 2	Nhóm 3	Nhóm 4

Nhóm	Nhóm 1	Nhóm 2	Nhóm 3	Nhóm 4
A	1, 5, 15, 18	19, 21, 23, 24	27, 29, 30, 31	32, 37, 42, 49
B	2, 3, 4, 13, 15	19, 20, 22	28, 29	38, 41, 42, 43, 48, 49

THUẬT GIẢI SONG SONG

- Nhóm 1 cho C[1..9] = (1, 2, 3, 4, 5, 13, 15, 15, 18)
- Nhóm 2 cho C[10..16] = (19, 19, 20, 21, 22, 23, 24)
- Nhóm 3 cho C[17..22] = (27, 28, 29, 30, 31)
- Nhóm 4 cho C[23..32] = (32, 37, 38, 41, 42, 43, 48, 49, 49)

THUẬT GIẢI SONG SONG

```
INPUT          :    A1 ≤ A2 ≤ .... ≤ An  và  B1 ≤ B2 ≤ ... ≤ Bn .
OUTPUT         :    C[1..2n] = A[1..n] U B[1..n] : C1 ≤ C2 ≤ ...≤ C2n
BEGIN
        FOR i = 1 TO r DO IN PARALLEL
                Pi :
                        READ(A(i-1)k+1, ... Aik);
                        j[i] = MAX{ t : Aik ≥ Bt ) : BINARY_SEARCH.
                        S_MERGE(A(i-1)k+1, ... Aik , Bj[i-1]+1,...Bj[i]);
        END PARALLEL;
END.
```

- Hàm BINARY_SEARCH: tìm kiếm nhị phân
- Hàm S_MERGE: trộn 2 mảng tuần tự

ĐÁNH GIÁ ĐỘ PHỨC TẠP

- 2 chương trình con được sử dụng:
  - Tìm kiếm nhị phân :$O(\log_2 n)$ đơn vị thời gian.
  - Nối hai mảng NA và NB
    - Nếu kích thước mảng NB_i cũng là k thì bước này có thể thực hiện với thời gian là O(logn)
    - Nếu kích thước của mảng NB_i lớn hơn k thì ta có thể lặp lại một cách đệ quy thao tác chia một vài lần với NB trước NB sau. Khi đó bước 3 cũng có thể thực hiện với O(logn) đơn vị thời gian.

5. DIVIDE AND CONQUER

DIVIDE AND CONQUER

- Nguyên tắc chia để trị như sau:
  - B1: Chia bài toán ban đầu thành các bài toán nhỏ hơn.
  - B2: Thực hiện đệ quy với các bài toán nhỏ.
  - B3: Kết hợp kết quả từ các bài toán nhỏ để đưa ra kết quả bài toán gốc.

Divide and Conquer

Tổng quát

Divide-and-conquer(Vấn đề P với kích thước n):

Nếu P đơn giản(n nhỏ), giải quyết trực tiếp P

Ngược lại:

Phân chia P thành $q \geq 1$ tập con độc lập $P_1, \dots, P_q$ với kích thước nhỏ hơn $n_1, \dots, n_q$

Thực hiện đệ quy:

$s_1 <-$ Divide-and-conquer (P_1, n_1),

…

$s_q <-$ Divide-and-conquer(P_q, n_q)

Nhóm các kết quả $S_1 \dots S_1$ lại

Divide-and-Conquer

Xét bài toán P với kích thước n, chia thành q bài toán nhỏ hơn với kích thước n/k $(k > 1)$, thực hiện song song với p bộ xử lý.

Khi đó $t_{divide_{conquer}}(n, p) =$

1) $t_{solvetrivial}(n)$ nếu n bé
2) $t_{solveseq}(n)$ nếu $p = 1$
3) $t_{divide}(n, p) + t_{conquer}(n, p) + \left\lceil \frac{q}{p} \right\rceil t_{divid-conquere\backslash}\left(\left\lceil \frac{n}{k} \right\rceil, 1\right)$ nếu $1 < p < q$
4) $t_{divide}(n, p) + t_{conquer}(n, p) + t_{divid-conquer}\left(\left\lceil \frac{n}{k} \right\rceil, \left\lfloor \frac{p}{k} \right\rfloor\right)$ nếu $p \geq q$

VÍ DỤ MINH HỌA

- Tính tổng n số A[1..n] với p BXL
- Ý tưởng chia để trị:
  - Nếu n = 1 trả lại giá trị A[1]
  - Nếu p = 1 tính tổng tuần tự.
  - Chia mảng A thành 2 phần A1 và A2, mỗi phần n/2 phần tử thực hiện song song.
    - Thực hiện đệ quy tính tổng S1 của A1 với p/2 BXL
    - Thực hiện đệ quy tính tổng S2 của A2 với p/2 BXl
  - Tính tổng S = S1 + S2

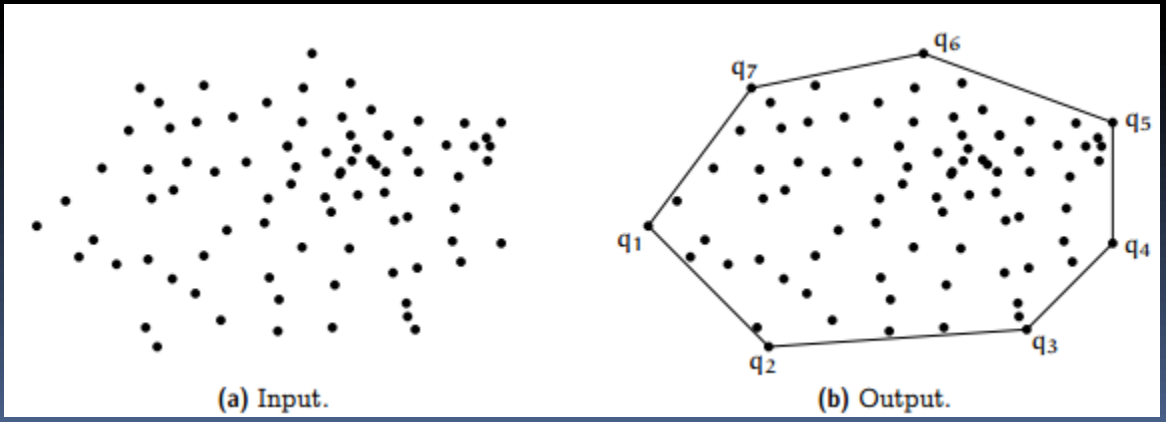
VÍ DỤ MINH HỌA

```
INPUT           :       A[1..n], p BXL;
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;
```

- Công thức đệ quy: T(n) = T(n/2) + O(1) (khi p~n)
- Độ phức tạp O(logn)
- Máy PRAM EREW

VÍ DỤ MINH HỌA CONVEX HULL

- Bài toán xác định bao lồi của 1 tập đỉnh trong mặt phẳng:
  - Đầu vào: n đỉnh trong mặt phẳng có tọa độ (x_k, y_k)
  - Đầu ra: tập các đỉnh tạo thành một đa giác lồi nhỏ nhất chứa tất cả cá đỉnh còn lại



SONG SONG CONVEX HULL

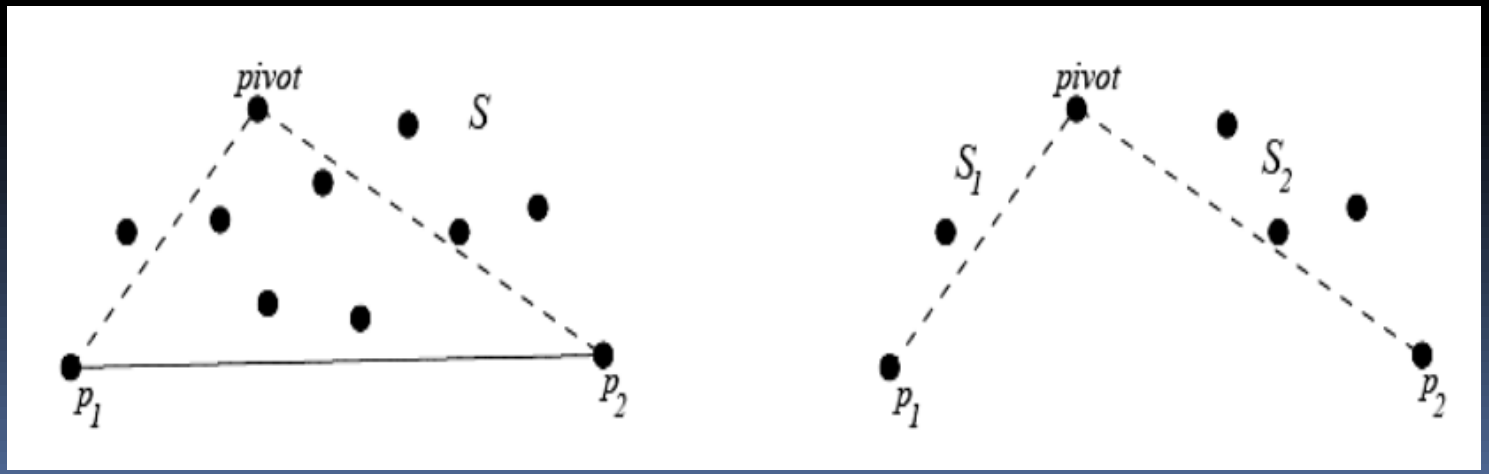
- Ý tưởng:
  - Khởi tạo: xác định u, v là 2 đỉnh có giá trị tọa độ x là nhỏ nhất và lớn nhất
  - Đoạn (u,v) sẽ chia tập đỉnh S thành 2 miền trên và dưới ký hiệu là S_upper và S_lower
  - Tiến hành xử lý song song 2 miền này

Parallel QuickHull

- Thực hiện với nửa trên,(nửa dưới làm tương tự)
- Chia để trị:
  - Chọn điểm trụ p (pivot) là điểm có khoảng cách xa nhất đối vưới đường (p_1, p_2) (ban đầu là đường u,v)
  - Các điểm còn lại chia làm 2 phần nằm bên ngoài các cạnh (p, p_1) và (p_2, p)
  - Thực hiện đệ quy với phần còn lại

Parallel QuickHull

- Dấu hiệu nhận biết:
  - Đỉnh trụ p: max $|(p_1 - p)(p_2 - p)|$
  - Các đỉnh trụ nằm trong nếu tổng các góc từ đỉnh đó bằng 2π
  - Góc giữa 2 véc tơ: $\cos(a, b) = (a.b)/(|a||b|)$



QuickHull

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

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

Đệ quy với UpperHull

- Các biến sử dụng:
  - Mỗi điểm i ứng với 1 biến Boolean $F[i]$:
    - Khởi tạo $F[i] = 1$
    - Bị loại vì nằm trong: $F[i] = 0$
    - Đánh dấu là đỉnh của bao lồi: $F[i] = 2$
  - Mỗi đỉnh xác định giá trị 2 đáy $P[i]$ và $Q[i]$

Đệ quy với UpperHull

- Các bước đệ quy:
  - Tất cả BXL thực hiện song song để xác định trụ $T[i]$
  - Cập nhật lại đỉnh $P[i]$ và $Q[i]$:
    - Các đỉnh bên trái của cạnh $(P[i], T[i])$ gán $Q[i] = T[i]$
    - Các đỉnh bên phải của cạnh $(T[i], Q[i])$ gán $P[i] = T[i]$
  - Cập nhật lại giá trị của $F[i]$
  - Lặp lại cho đến khi $F[i] \neq 1$

6. ACCELERATED CASCADING

CÁC KHÁI NIỆM VỀ ĐỘ PHỨC TẠP

- Trong tính toán tuần tự: độ phức tạp = số bước thực hiện thuật toán . Ký hiệu S(n).
- Trong tính toán song song : số thao tác được thực hiện trên tất cả các BXL. Ký hiệu W(n).
- Nếu Wi(n) là số thao tác thực hiện đồng thời tại bước thứ i ta có có công thức

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

Kỹ thuật Accelerated Cascading

- Chi phí cho một giải thuật là số thao tác mà hệ thống phải thực hiện.
- Một số giải thuật được gọi là tối ưu nếu như : $W(n) = \Theta(T_s(n))$. Trong đó:
  - $W(n)$: chi phí của giải thuật song song
  - $T_S(n)$: thời gian của thuật giải tuần tự tốt nhất

Ví dụ minh họa

- Cho dãy $L[1 \ldots n]$ nhận các giá trị nguyên từ $1..k$ với $k = \log_2 n$. Hãy xác định số lần xuất hiện các số trong dãy L
- Gọi $R[i]$là số lần xuất hiện của giá trị i.
- Giải thuật tuần tự tối ưu $T_s(n) = \Theta(n)$ như sau:

```
R[1 : k] ← 0
for i = 1 to n do
    R[L[i]] ← R[L[i]] + 1
enddo
```

Song song thứ I

- Dùng mảng 2 chiều: $C[1 \ldots n, 1 \ldots n]$ với:

$$C_{ij} = \begin{cases} 1 \; if \; L[i] = j \\ 0 \; otherwise \end{cases}$$

Số lần xuất hiện i bằng tổng của $C[1{:}\, n, j]$

```
forall i ∈ 1 : n, j ∈ 1 : k do
   C[i, j] ← 0
enddo
forall i ∈ 1 : n do
   C[i, L[i]] ← 1
enddo
forall j ∈ 1 : k do
   R[j] ← REDUCE(C[1 : n, j], +)
enddo
```

7. PIPELINING

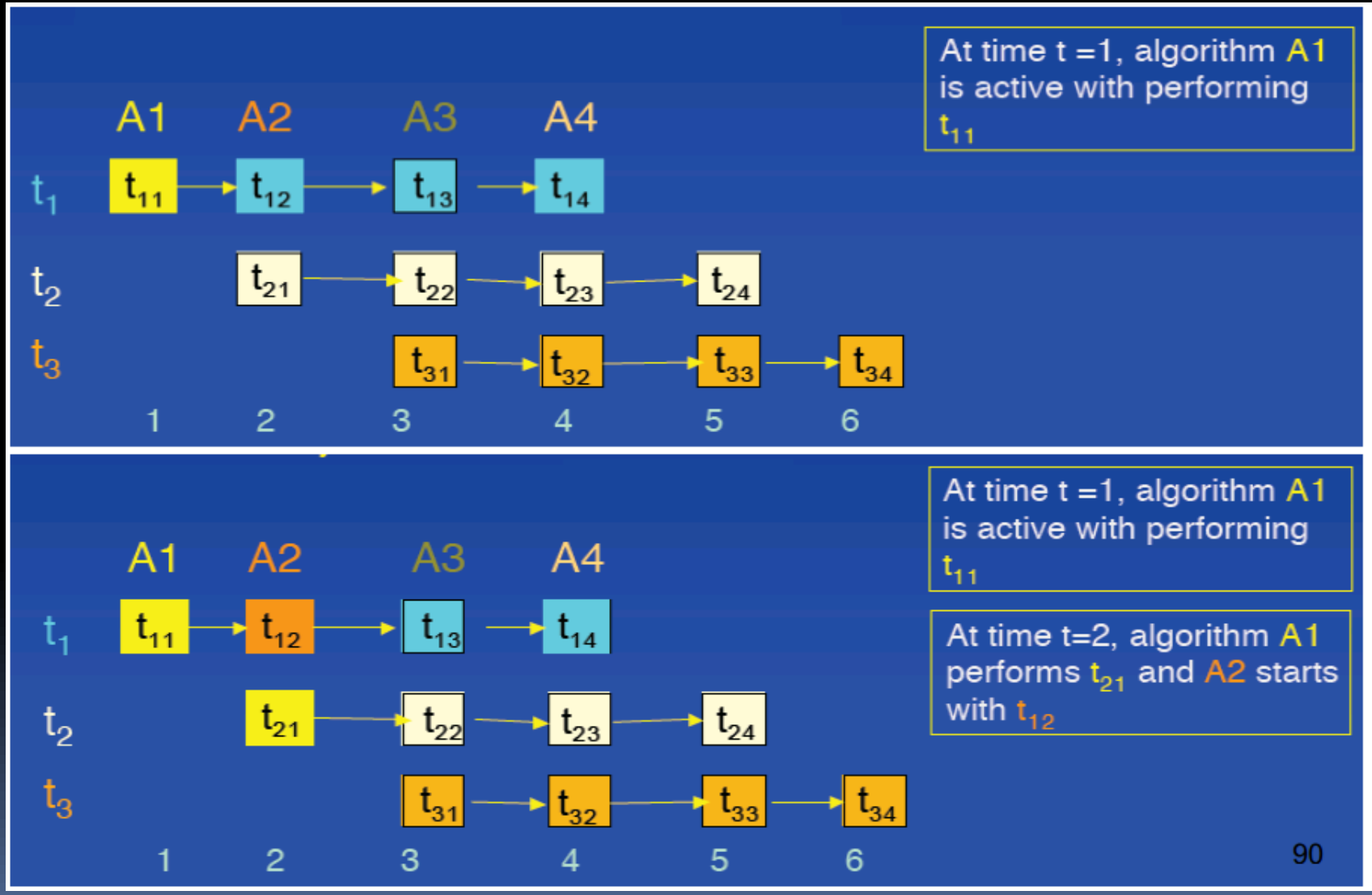
PIPELINING

- Được sử dụng rộng rãi để tăng tốc thực hiện nhiều bài toán, trong đó:
  - Mỗi bài toán lớn có thể chia thành nhiều bài toán con.
  - Các bài toán con này có thể phụ thuộc với nhau theo trình tự thực hiện.
  - Tại mỗi thời điểm, các BXL thực hiện các bài toán con của mỗi bài toán lớn song song (đảm bảo trình tự thực hiện không đổi)

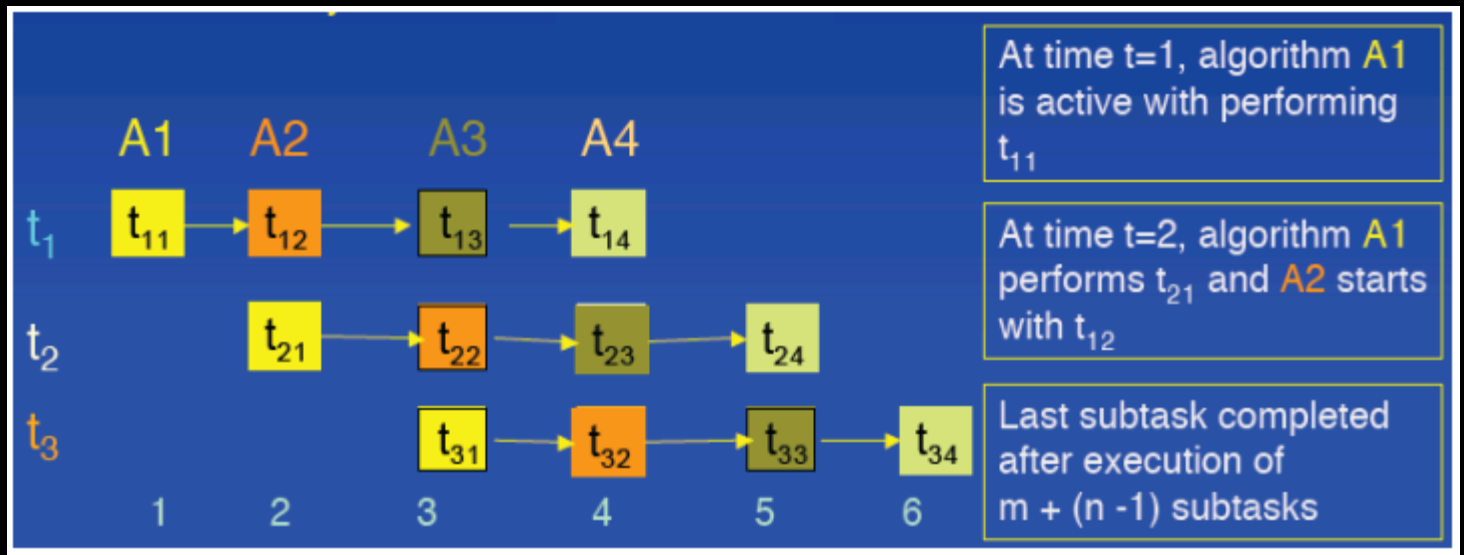
CƠ CHẾ PIPELINING

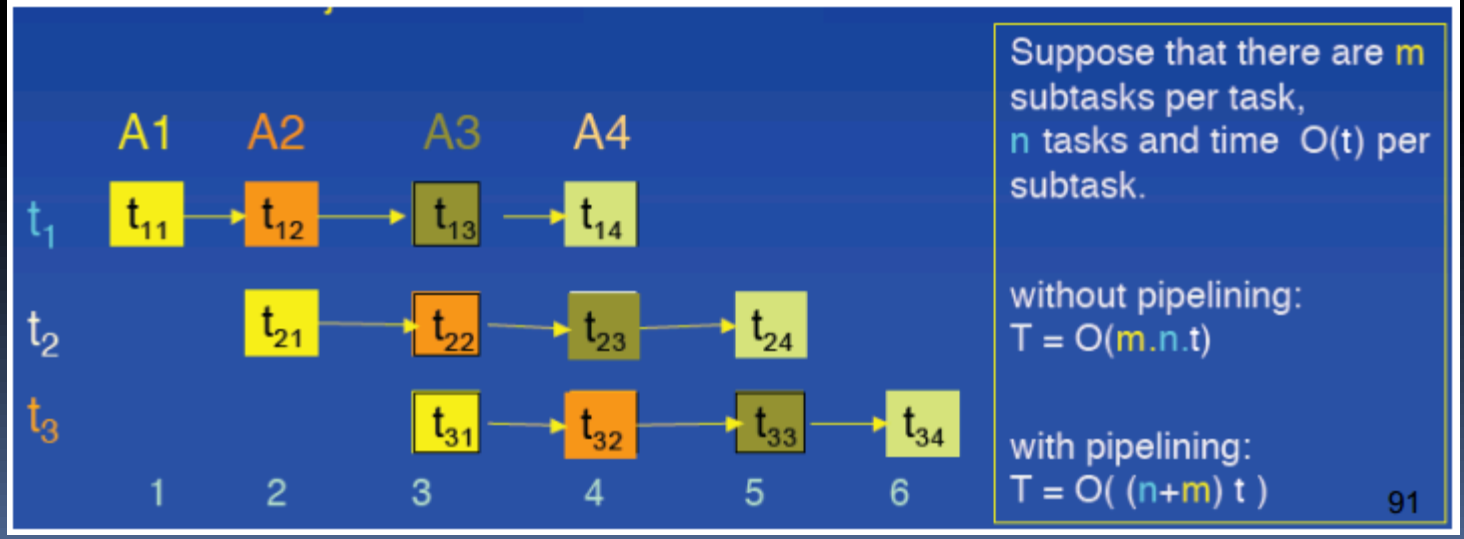
- Xét tập n bài toán: t_1, t_2, …., t_n cần thực hiện
- Mỗi t_i có thể chia thành tập m các bài toán con $\{t_{i,1}$, $t_{i,2}$, ….., $t_{i,m}\}$ sao cho $t_{i,k}$ phải kết thúc trước khi bắt đầu $t_{i,k+1}$.

CƠ CHẾ PIPELINING



CƠ CHẾ PIPELINING

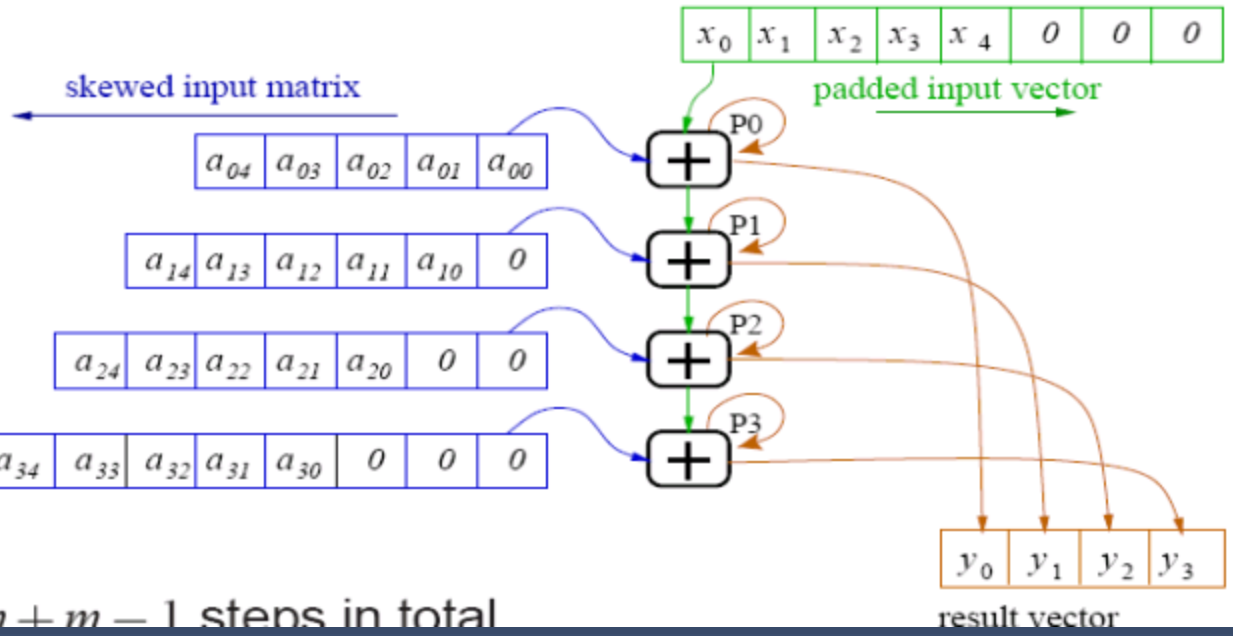




VÍ DỤ NHÂN MATRIX VECTOR

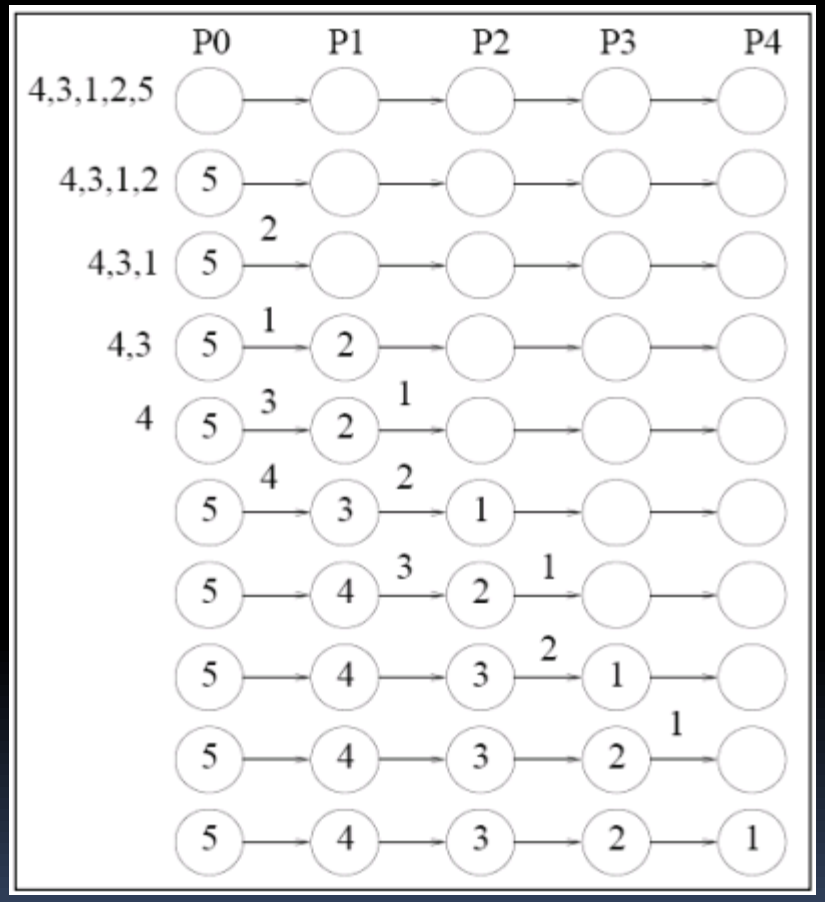
multiply matrix $A \in \mathbb{R}^{n,m}$ by vector $x \in \mathbb{R}^m \rightarrow$ vector $y \in \mathbb{R}^n$

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



$n + m - 1$ steps in total

PARALLEL INSERTION SORT



PARALLEL INSERTION SORT

```
Procedure  zero-time-sorting
    let  local := +∞
    let  temp := +∞
    repeat          /* input phase */
        temp := receive from left port
        if (temp = #) then
            send temp to right port
            exit repeat
        else
            send max(local, temp) to right port
            local := min(local, temp)
        end if
    end repeat
    send local to left port
    repeat          /* output phase */
        local := receive from the right port
        send local to left port
        if local = # then exit repeat
    end repeat
```

8. SYMMETRIC BREAKING

ĐẶC ĐIỂM

- Tính đối xứng của bài toán:
  - Bài toán có thể chia ra thành 1 tập các bài toán con.
  - Các bài toán con có vai trò như nhau

ĐẶC ĐIỂM

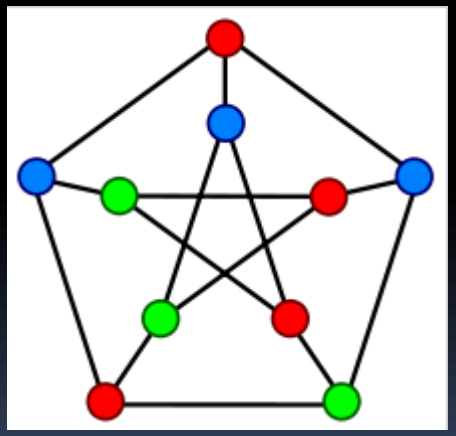
- Vấn đề: Cách lựa chọn công việc để bắt đầu
- Cách giải quyết:
  - Trong tuần tự: Chọn ngẫu nhiên 1 bài toán con
  - Trong song song:
    - Nhiều BXL cùng bắt đầu do đó cách chọn ngẫu nhiên
    - Các công việc con có thể phụ thuộc nhau do đó sự ràng buộc trong công việc của các BXL

ĐẶC ĐIỂM

- Giải quyết vấn đề ngắt tính đối xứng c
- Input hoặc (subtasks) tập các đỉnh có vai trò như nhau (symmetry)
- Symmetry Breaking sẽ phân chia các đỉnh thành các lớp riêng biệt -> ngắt tính đối xứng.

ĐẶC ĐIỂM

- Bài toán k-coloring: tô màu cho các đỉnh của G bởi hàm c: $V(G) \rightarrow \{0..k-1\}$ sao cho nếu $< v_i, v_j > \in E(G)$ thì $c(v_i) \neq c(v_j)$



3-coloring trên đồ thị hình tròn có hướng

- G=(V,E) là đồ thị có **hướng hình tròn**. Đối với mọi đỉnh:
  - Outdegree = 1
  - Indegree = 1
  - Đường đi (u,v) duy nhất

BÀI TOÁN 3-COLORING

- Thuật toán tuần tự bài toán 3 màu (0,1,2)
- Nếu n là số đỉnh của đồ thị thì thuật toán tuần tự thực hiện với O(n) bước lặp

BÀI TOÁN 3-COLORING

- Vấn đề khi thực hiện song song:
  - Vấn đề chọn điểm xuất phát.
  - Giả sử đồ thị có n đỉnh và các đỉnh của đồ thị ban đầu được xác định bởi các giá trị từ 1..n. Ở đây trình tự liên két giữa các đỉnh là không nhất thiết phải theo trình tự tuyến tính. Ví dụ
    - 1->3->4->2->5->6->8->7->1 ngẫu nhiên
    - 1->2->3->4->5->6->7->8->1 tuyến tính

BÀI TOÁN 3-COLORING

- Đỉnh tuyến tính:
  - Mỗi BXL được gán với 1 đỉnh
  - Đọc giá trị của đỉnh:
    - Chẵn = 0
    - Lẻ = 1
    - Đỉnh n lẻ = 2
- n BXL => độ phức tạp thuật toán là O(1)

BÀI TOÁN 3-COLORING

Input : G = (V,E), |V| = n, E = {(V_i V_{i+1})} i = 1..n,
: (V_{n+1} = V_1).
Output : c[i] = {0,1,2} | c[i] ≠ c[i+1] i = 1..n, c[n+1] = c[1].
Begin
for i = 1 to n do in parallel
if (i lẻ và i ≠ n) then
c[i] = 0;
if (i chẵn) then
c[i] = 1;
if (n lẻ) then
c[n] = 2;
end parallel.
End.

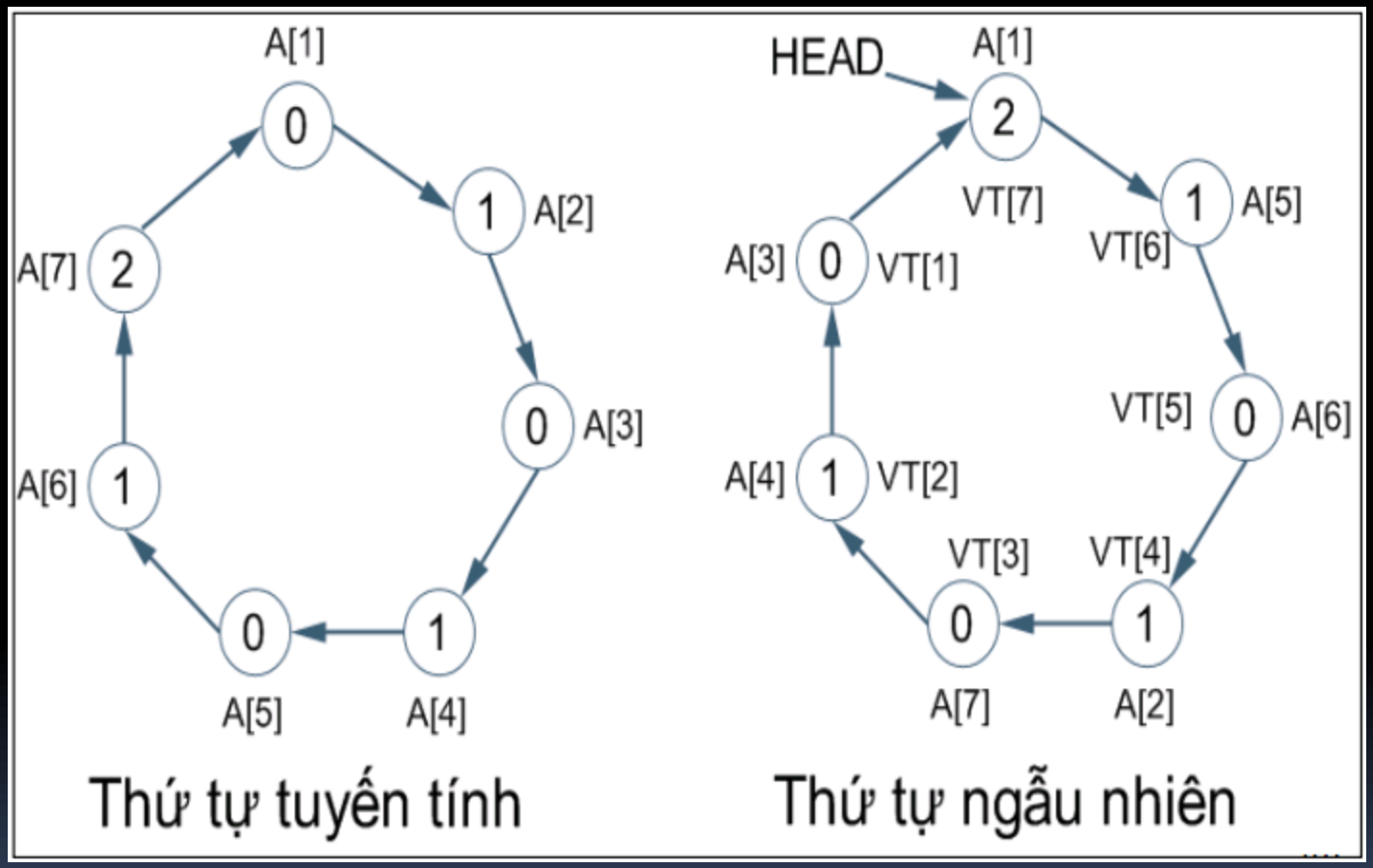
BÀI TOÁN 3-COLORING

- Nếu trình tự các đỉnh không tuyến tính thì cách tiếp cận thứ nhất:
  - 1 đỉnh = (HEAD)
  - B1 Tìm hạng các phần tử
  - B2 Như thuật toán trước
- Đánh giá độ phức tạp:
  - B1: Thực hiện với O(logn) bước tuần tự
  - B2: Thực hiện với O(1) bước

BÀI TOÁN 3-COLORING

```
Input   : G = (V,E), |V| = n, LINK[i] = j if (Vi Vj) ∈ E i,j ∈ 1..n.
Output  : c[i] = {0,1,2} | c[i] ≠ c[j] if (Vi Vj) ∈ E.
Begin
        HEAD    =       1.
        for i = 1 to n do in parallel
                NEXT[i] =       LINK[i];
                VT[i]   =       1;
        end for.
        For k = 1 to log2n do
                For i = 1 to n do in parallel
                        If NEXT[i] ≠ HEAD then
                                VT[i]   = VT[i] + VT[NEXT[i]];
                                NEXT[i] = NEXT[NEXT[i]];
                        End if;
                End parallel;
        End for.
        For i = 1 to n do in parallel
                If (VT[i] lẻ và i ≠ HEAD) then
                        C[i]    =       0;
                If (VT[i] chẵn) then
                        C[i]    =       1;
                If (VT[HEAD] lẻ ) then
                        C[HEAD] = 2;
        End parallel
End.
```

BÀI TOÁN 3-COLORING



BÀI TOÁN 3-COLORING

- Nhận xét:
  - HEAD => Tính đối xứng phá vỡ
- Cách tiếp cận khác: Thuật toán Basic Coloring
- Biểu diễn đồ thị có hướng G=(V,E) bằng mảng:
  - S[i] = j nếu (i,j) thuộc E với 1<= i,j <= n
  - P[i] = j nếu (j,i) thuộc E với 1<= i,j <= n -> P[S[i]] =1

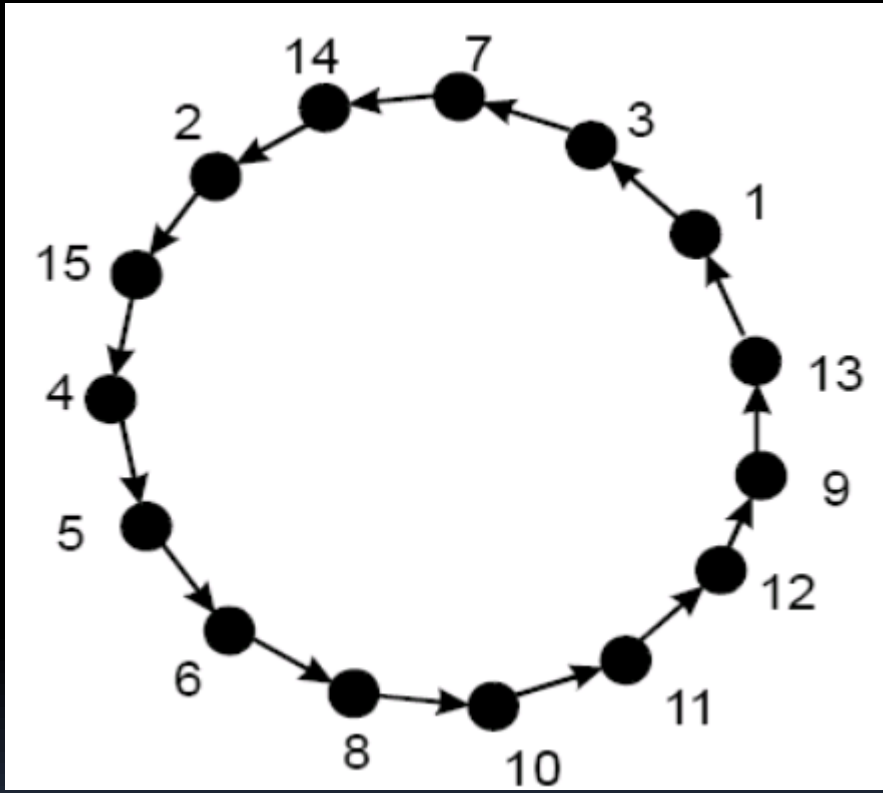
THUẬT TOÁN BASIC COLORING

- C[i] = i với i thuộc 1..n.(Màu thứ i)
- Biểu diễn giá trị i dưới cơ số 2 -> dãy t bits {0, 1}: i= $b_{t-1}.....b_k.....b_1b_0$.
- Thuật toán Basic Coloring giảm từ n màu xuống 2*t màu

BASIC COLORING N=15

- Ý tưởng thuật toán:
  - Thực hiện song song với n BXL
  - Tại BXL thứ i, xét 2 đỉnh kề nhau (i,S[i])
  - Xác định vị trí k nhỏ nhất trên dãy bit biểu diễn 2 màu tương ứng sao cho giá trị bit tại đó khác nhau.
  - Cập nhật giá trị màu mới cho đỉnh thứ i theo công thức: $c'[i] = 2k + c[i]_k$ (trong đó $c[i]_k$ là bit giá trị thứ k trong dãy bit biểu diễn của c[i])

BASIC COLORING N=15



v	c	k	c'
1	0001	1	2
3	0011	2	4
7	0111	0	1
14	1110	2	5
2	0010	0	0
15	1111	0	1
4	0100	0	0
5	0101	0	1
6	0110	1	3
8	1000	1	2
10	1010	0	0
11	1011	0	1
12	1100	0	0
9	1001	2	4
13	1101	2	5

- k là chỉ số LSB – Least Significant bit, c(i) <> c(i+1),
- $c'(i)=2k+c(i)_k$

BASIC COLORING N=15

```
Input     : G = (V,E), |V| = n, S[i] = j if (i,j) ∈E, c[i] = i.
Output    : c'[i] | c'[i] ≠ c'[j] if (i,j) ∈E.
Begin
        for i = 1 to n do in parallel
                B1. xác định k-th least significant (c[i] & c[S[i]]);
                B2. c'[i]       =       2k + c[i]_k .
        end parallel.
End.
```

- Thuật toán với O(1) đơn vị thời gian

BASIC COLORING

- Thuật toán Basic Coloring không giảm đến 3 màu
- Với t1 là số bit biểu diễn tối đa của n thì sau 1 bước số màu còn lại là: n’ = $2*t_1$
- Nếu biểu diễn n’ với t_2 bit thì lặp lại thuật toán ta giảm số màu còn: $2*t_2$
- Cho đến khi t_k=3 thì không giảm thêm được nữa -> Basic coloring giảm tối đa đến 6 màu

BASIC COLORING

- Cách chuyển từ 6 màu thành 3 màu:
  - Mỗi BXL đọc giá trị màu của đỉnh tương ứng
  - Nếu giá trị màu thuộc {0,1,2} thì giữ nguyên
  - Nếu giá trị màu thuộc {3,4,5} gán giá trị màu mới là giá trị nhỏ nhất {0,1,2}
- Như vậy thuật toán thực hiện với thời gian O(log*n) và O(n.log*n) phép toán

Hết bài!!!