parallel algorithm

1.并行算法

解决读写冲突的一些rule

To resolve access conflicts

- Exclusive-Read Exclusive-Write (EREW)
- Concurrent-Read Exclusive-Write (CREW)
- Concurrent-Read Concurrent-Write (CRCW)
  - *Arbitrary rule*
  - *Priority rule* (P with the smallest number)
  - *Common rule* (if all the processors are trying to write the same value)

2.summation 栗子1

The summation problem.

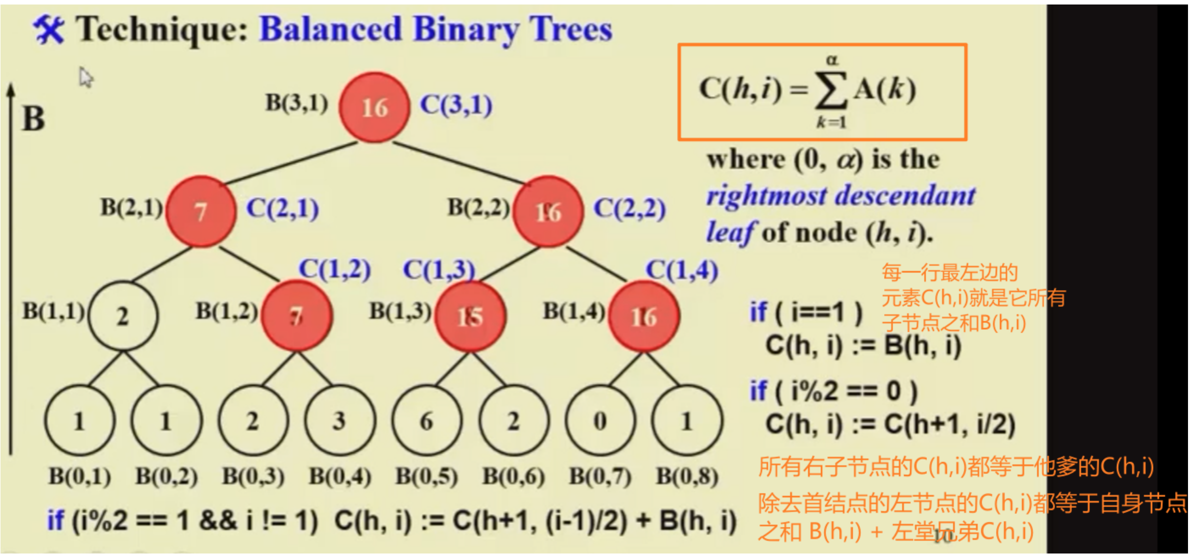
Input: A(1), A(2), …, A(n)

Output: A(1) + A(2) + … +A(n)

3.pre-fix栗子2

Input: A(1), A(2), …, A(n)

Output: $\sum_{i=1}^{1} A(i), \sum_{i=1}^{2} A(i), \cdots, \sum_{i=1}^{n} A(i)$



伪代码来咯!

```
for P_i , 1 ≤ i ≤ n pardo
  B(0, i) := A(i)
for h = 1 to log n
  for i , 1 ≤ i ≤ n/2^h pardo
    B(h, i) := B(h - 1, 2i - 1) + B(h - 1, 2i)
for h = log n to 0
  for i even, 1 ≤ i ≤ n/2^h pardo
    C(h, i) := C(h + 1, i/2)
  for i = 1 pardo
    C(h, 1) := B(h, 1)
  for i odd, 3 ≤ i ≤ n/2^h pardo
    C(h, i) := C(h + 1, (i - 1)/2) + B(h, i)
for P_i , 1 ≤ i ≤ n pardo
  Output C(0, i)
```

初始化 T:O(1) W:O(n)

自底向上计算B(h,i)
T:O(logn) W:(n/2+n/4+..) = O(n)

自顶向下计算C(h,i),结点分三种情况讨论
T:O(logn) W:(n/2+n/4+..) = O(n)

输出结果 T:O(1) W:O(1)

$$T(n) = O(\log n) \qquad W(n) = O(n)$$

The operations are charged to nodes of the balanced binary tree

4.merging

〖Example〗 Merging – merge two *non-decreasing* arrays A(1), A(2), …, A(n) and B(1), B(2), …, B(m) into another *non-decreasing* array C(1), C(2), …, C(n+m)

partition paradigm (分组范式)

partition：n/p

merging to ranking

1）获得rank：

$$\text{RANK}(j, A) = i, \quad \text{if } A(i) < B(j) < A(i+1), \text{ for } 1 \le i < n$$ 一般情况

$$\boxed{\begin{aligned}&\text{RANK}(j, A) = 0, \quad \text{if } B(j) < A(1)\\ &\text{RANK}(j, A) = n, \quad \text{if } B(j) > A(n)\end{aligned}}$$ 边界条件

怎么找rank?

方法1：binary search：

```
for P_i , 1 ≤ i ≤ n  pardo
    RANK(i, B) := BS(A(i), B)
    RANK(i, A) := BS(B(i), A)
```

$$T(n) = O(\log n)$$
$$W(n) = O(n \log n)$$

方法2：serial ranking:

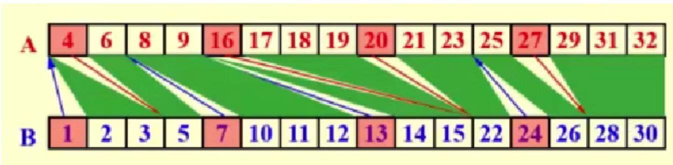
☞ Serial Ranking

```
i = j = 0;
while ( i ≤ n || j ≤ m ) {
   if ( A(i+1) < B(j+1) )
      RANK(++i, B) = j;
   else RANK(++j, A) = i;
}
```

$$T(n) = W(n) = O(n + m)$$

方法3：parallel ranking：

用n/ logn 个处理器并行运算，A,B中各选择p个元素，计算每个选中元素的rank



- T=O（logn）（每一次binary search时间）
- W = O(p logn) = O(n) (p个processor，每一个任务量是 logn)

最多有2p个子问题,size为O（logn）

- T=O（logn）（子问题大小logn）
- W = O(p logn) = O(n) (p个processor，每一个任务量是 logn)

2） 找到元素在C的位置：

```
for P_i , 1 ≤ i ≤ n  pardo
    C(i + RANK(i, B)) := A(i)
for P_i , 1 ≤ i ≤ n  pardo
    C(i + RANK(i, A)) := B(i)
```

栗子：

i	1	2	3	4	5	6	7	8
A	11	12	15	17				
RANK(*i*, B)	0	0	2	3				
B	13	14	16	18				
RANK(*i*, A)	2	2	3	4				
C	11	12	13	14	15	16	17	18

5.maxium finding

1)用summation方法，max 替代 +

T(n)=O(logn),W(n)=0(n)

2）把时间复杂度降到O(1)：

```
for P_i , 1 ≤ i ≤ n  pardo
    B(i) := 0
for i and j, 1 ≤ i, j ≤ n  pardo
    if ( (A(i) < A(j)) || ((A(i) = A(j)) && (i < j)) )
        B(i) = 1
    else B(j) = 1        标记为loser
for P_i , 1 ≤ i ≤ n  pardo
    if B(i) == 0
      A(i) is a maximum in A
```

$$T(n)=O(1), \quad W(n)=O(n^2)$$

存在access conflicts? CRCW

如何降低 W(n) ? ——partition！！！

3）a doubly-logarithmic paradigm (loglogn 范式)

$\boxed{\text{Partition by } \sqrt{n} :}$

$$\begin{array}{llllll}
A_1 & = & A(1), & \cdots, & A(\sqrt{n}) & \Rightarrow M_1 \sim T(\sqrt{n}), W(\sqrt{n}) \\
A_2 & = & A(\sqrt{n}+1), & \cdots, & A(2\sqrt{n}) & \Rightarrow M_2 \sim T(\sqrt{n}), W(\sqrt{n}) \\
 & \cdots & \cdots & & & \\
A_{\sqrt{n}} & = & A(n-\sqrt{n}+1), & \cdots, & A(n) & \Rightarrow M_{\sqrt{n}} \sim T(\sqrt{n}), W(\sqrt{n})
\end{array}$$

$$M_1, M_2, \cdots, M_{\sqrt{n}} \Rightarrow A_{\max} \quad \sim \boxed{T = O(1),\ W = O(\sqrt{n}^2) = O(n)}$$

$$\boxed{T(n) \le T(\sqrt{n}) + c_1, \quad W(n) \le \sqrt{n}\, W(\sqrt{n}) + c_2 n}$$

$$\Rightarrow \boxed{T(n) = O(\log\log n), \quad W(n) = O(n \log\log n)}$$

W(n)增加了？可恶

换一个partition：

Partition by $h = \log\log n$:

$$\begin{array}{llllll}
A_1 & = & A(1), & \cdots, & A(h) & \Rightarrow M_1 \sim O(h) \\
A_2 & = & A(h+1), & \cdots, & A(2h) & \Rightarrow M_2 \sim O(h) \\
 & \cdots & \cdots & & & \\
A_{n/h} & = & A(n-h+1), & \cdots, & A(n) & \Rightarrow M_{n/h} \sim O(h)
\end{array}$$

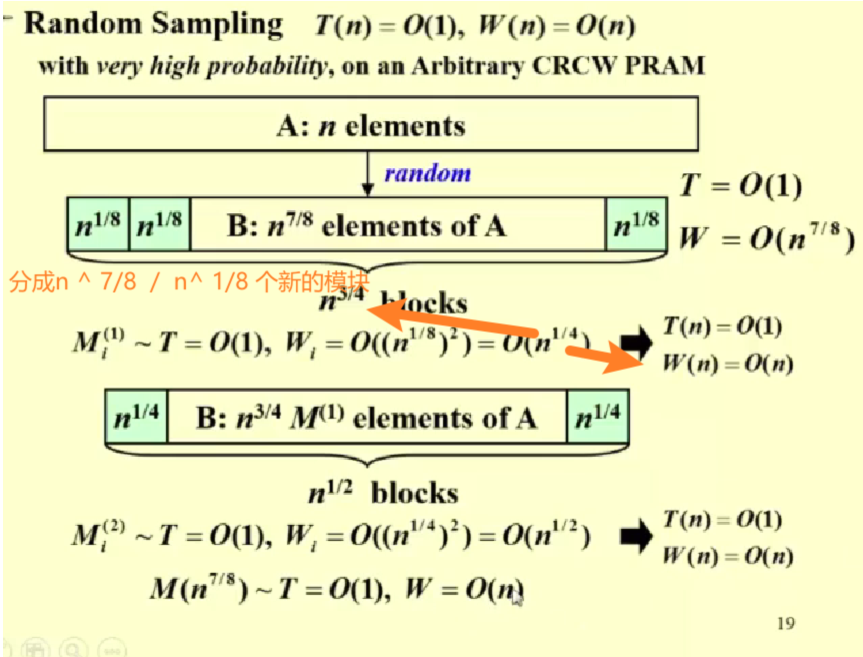
$$M_1, M_2, \cdots, M_{n/h} \Rightarrow A_{\max}$$

size of problem

$$T(n) = O(h + \log\log(n/h)) = O(\log\log n)$$

$$W(n) = O(h \times (n/h) + (n/h)\log\log(n/h)) = O(n)$$

random sampling： T(n) = O(1) , W(n) = O(n)



但是万一最大值不在M里面，扔进来~

【Theorem】The algorithm finds the maximum among n elements. With very high probability it runs in O(1) time and O(n) work. The probability of *not finishing* within this time and work complexity is O($1/n^c$) for some positive constant c.