Parallel Algorithms

Parallelism

☞ ***Machine parallelism***

- **Processor parallelism**
- **Pipelining**
- **Very-Long Instruction Word (VLIW)**

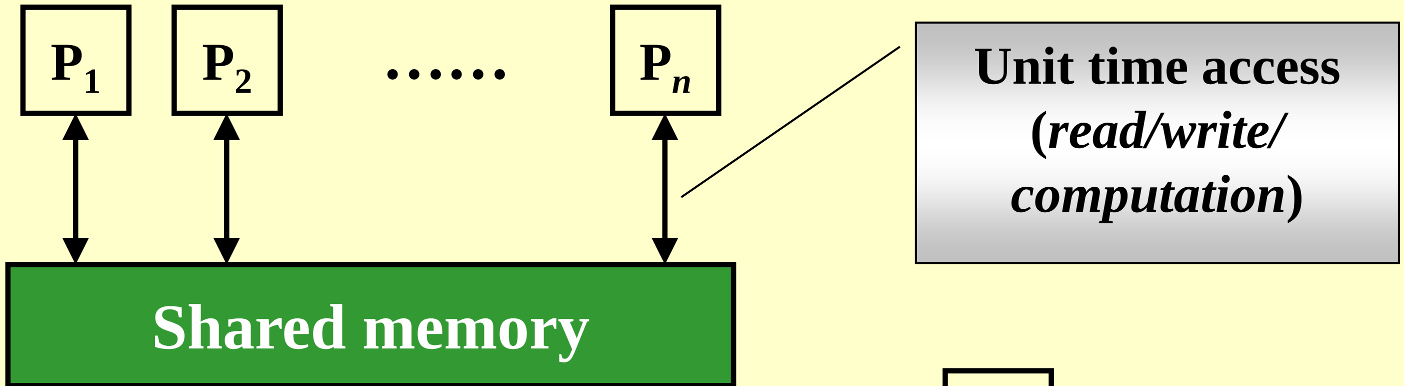
☞ ***Parallel algorithms***

To describe a parallel algorithm

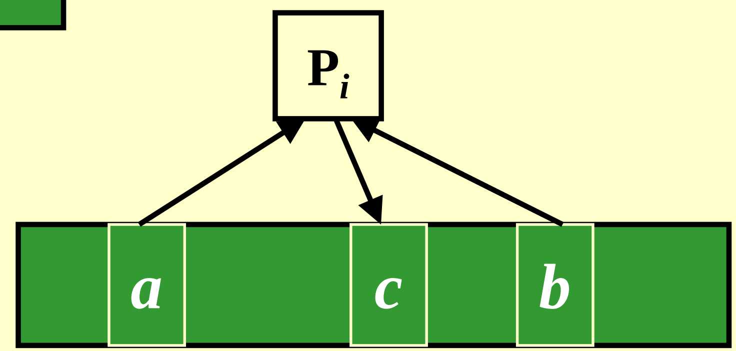
☞ **Parallel Random Access Machine (PRAM)**

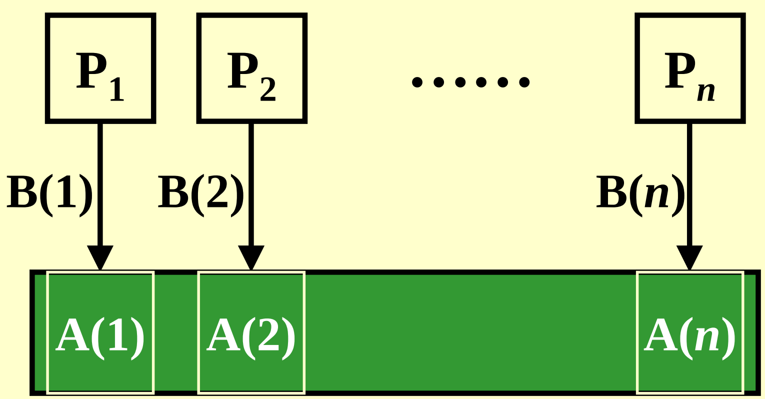
☞ **Work-Depth (WD)**

Parallel Random Access Machine (PRAM)



processor i: $c := a + b$





for P_i , $1 \le i \le n$ **pardo**

$A(i) := B(i)$

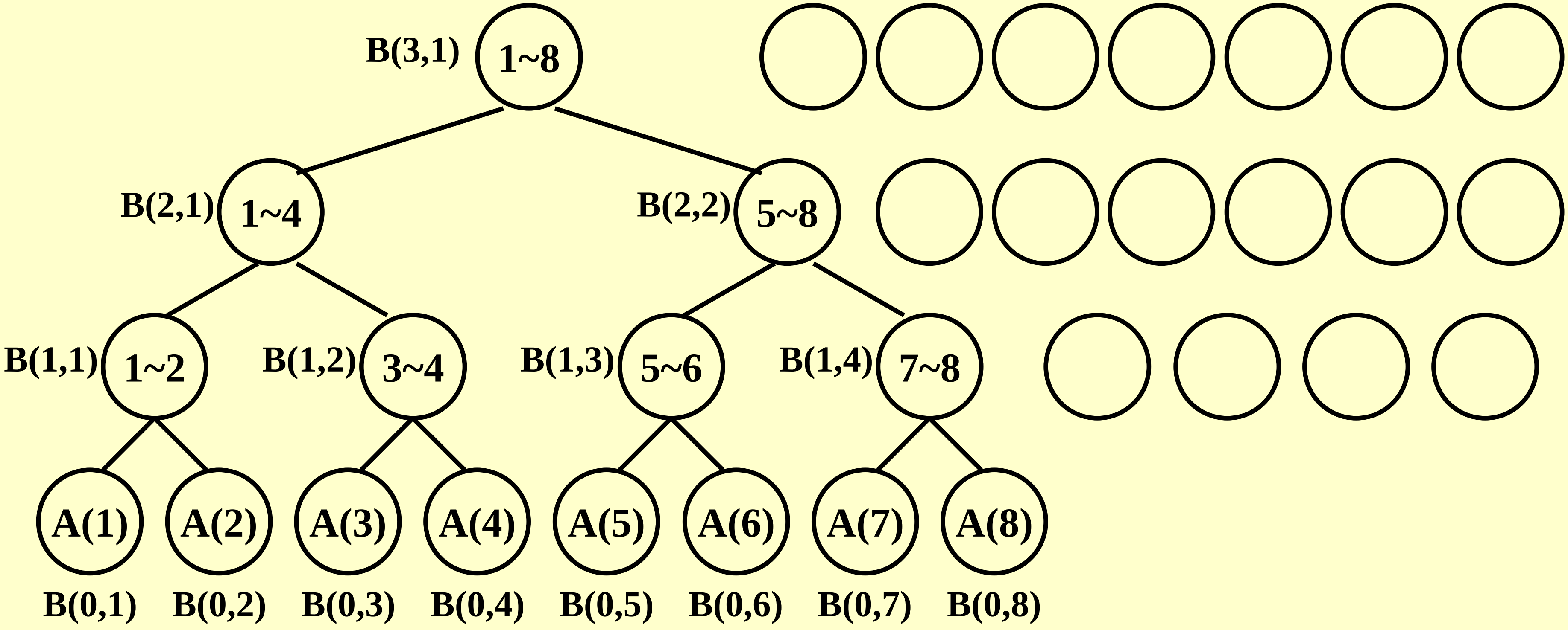
To resolve access conflicts

- ☞ **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)**

〖Example〗 The summation problem.

Input: A(1), A(2), …, A(*n*)

Output: A(1) + A(2) + … +A(*n*)

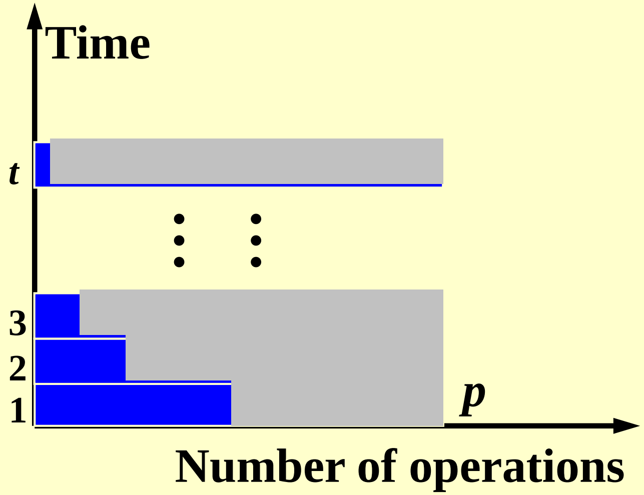


$$B(h, i) = B(h-1, 2i-1) + B(h-1, 2i)$$

PRAM model

for P_i , $1 \le i \le n$ **pardo**
  B(0, i) := A(i)

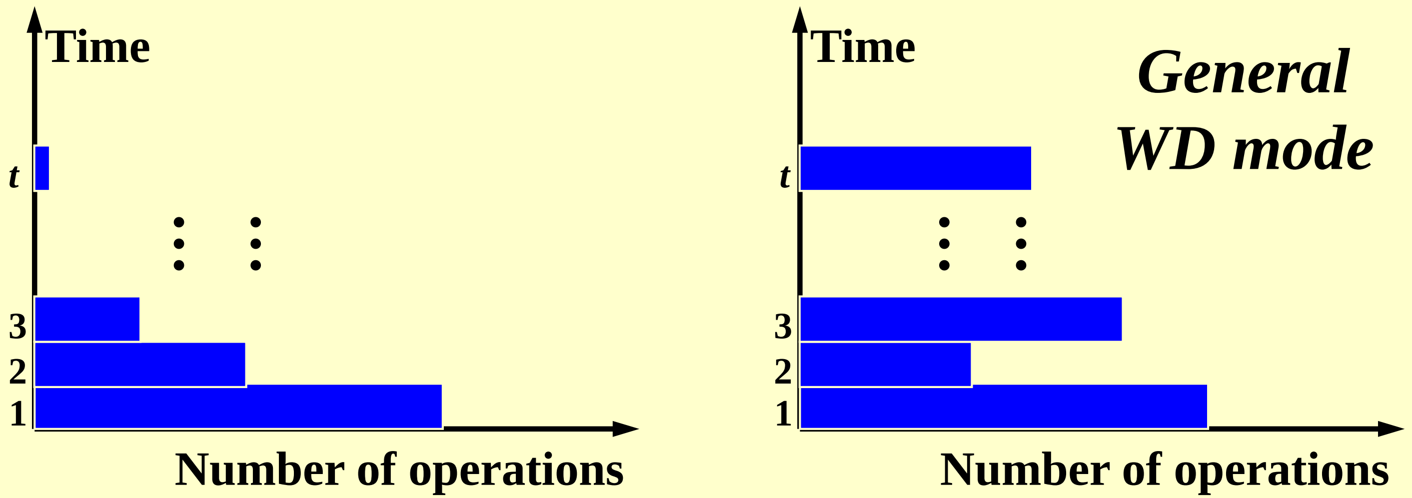
$$T(n) = \log n + 2$$



- Does not reveal how the algorithm will run on PRAMs with different number of processors
- Fully specifying the allocation of instructions to processors requires a level of detail which might be unnecessary

Work-Depth (WD) Presentation

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



General WD mode

Measuring the performance

☞ **Work load – total number of operations: $W(n)$**

☞ **Worst-case running time: $T(n)$**

- **$W(n)$ operations and $T(n)$ time**
- **$P(n) = W(n)/T(n)$ processors and $T(n)$ time (on a PRAM)**
- **$W(n)/p$ time using any number of $p \leq W(n)/T(n)$ processors (on a PRAM)**
- **$W(n)/p + T(n)$ time using any number of p processors (on a PRAM)**

All asymptotically equivalent

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

$$T(n) = \log n + 2$$

$$W(n) = n + n/2 + n/2^2 + \cdots + n/2^k + 1 \quad \text{where } 2^k = n$$
$$= 2n$$

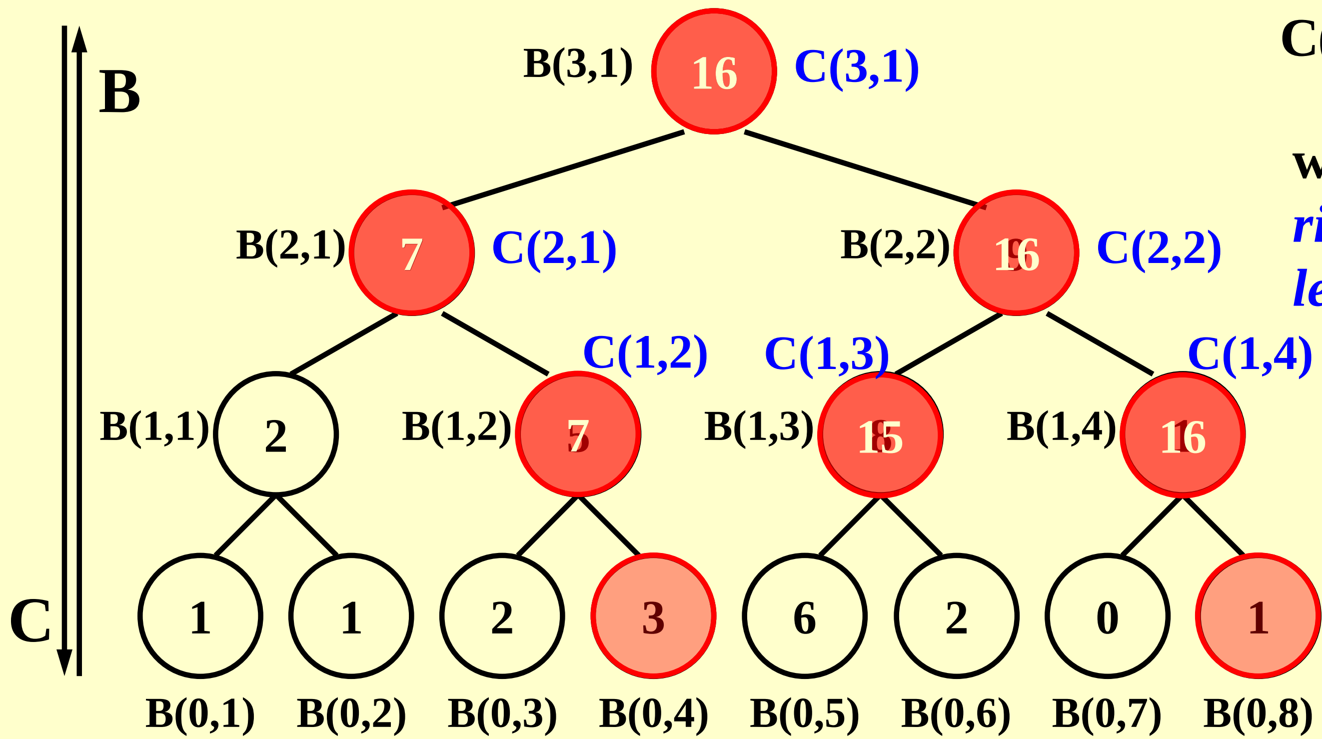
【WD-presentation Sufficiency Theorem】An algorithm in the WD mode can be implemented by *any* $P(n)$ processors within $O(W(n)/P(n) + T(n))$ time, using the same concurrent-write convention as in the WD presentation.

〖Example〗 Prefix-Sums.

Input: A(1), A(2), …, A(n)

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

🛠 Technique: Balanced Binary Trees



$$C(h,i) = \sum_{k=1}^{\alpha} A(k)$$

where (0, α) is the *rightmost descendant leaf* of node (h, i).

```
if ( i==1 )
  C(h, i) := B(h, i)
if ( i%2 == 0 )
  C(h, i) := C(h+1, i/2)
```

```
if (i%2 == 1 && i != 1)  C(h, i) := C(h+1, (i-1)/2) + B(h, i)
```

for P_i , $1 \le i \le n$ **pardo**
  B(0, i) := A(i)

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

☝ ***The operations are charged to nodes of the balanced binary tree***

〖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)**

🛠 **Technique: Partitioning**

To simplify, assume:

1. **the elements of A and B are pairwise distinct**
2. $n = m$
3. **both log n and n/log n are integers**

☞ **Partitioning Paradigm**

• ***partitioning* - partition the input into a large number, say p, of *independent* small jobs, so that the size of the largest small job is roughly n/p**

• ***actual work* - do the small jobs *concurrently*, using a separate (possibly serial) algorithm for each**

Merging ➡ Ranking

$\text{RANK}(j, A) = i$, if $A(i) < B(j) < A(i + 1)$, for $1 \leq i < n$

$\text{RANK}(j, A) = 0$, if $B(j) < A(1)$

$\text{RANK}(j, A) = n$, if $B(j) > A(n)$

The ***ranking problem***, denoted **RANK(A,B)** is to compute:

1. $\text{RANK}(i, B)$ for every $1 \leq i \leq n$, and
2. $\text{RANK}(i, A)$ for every $1 \leq i \leq n$

Claim: Given a solution to the ranking problem, the merging problem can be solved in O(1) time and O(n+m) work.

```
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)
```

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

☞ Binary Search

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

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

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

☞ 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;
}
```

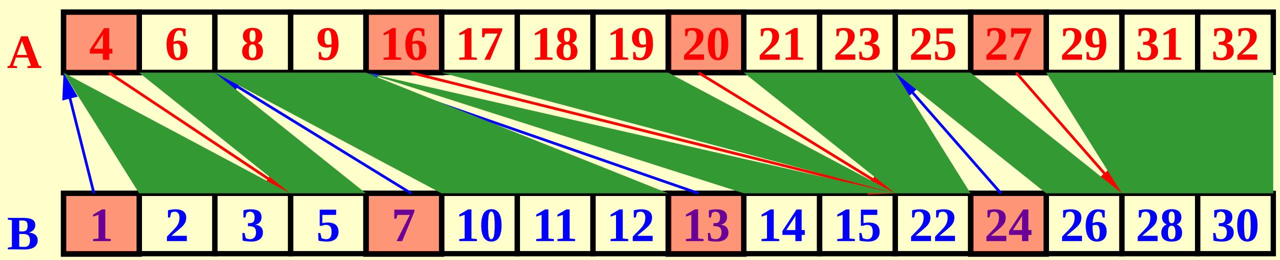
$$T(n) = W(n) = O(n + m)$$

☞ Parallel Ranking

Assume that $n = m$; and that A(n+1) and B(n+1) are each larger than both A(n) and B(n).

Stage 1: Partitioning $\quad p = n / \log n$

- A_Select(i) = A(1+(i-1)log$n$) for $1 \le i \le p$ $\qquad T = O(\log n)$
- B_Select(i) = B(1+(i-1)log$n$) for $1 \le i \le p$ $\qquad W = O(p \log n) = O(n)$
- Compute RANK for each *selected* element



Stage 2: Actual Ranking

At most 2p smaller sized (O(log$n$)) problems.

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

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

〖Example〗 Maximum Finding.

☞ Replace "+" by "max" in the summation algorithm

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

☞ Compare all pairs

```
for Pi , 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
for Pi , 1 ≤ i ≤ n  pardo
   if B(i) == 0
     A(i) is a maximum in A
```

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

Discussion 21: How to resolve access conflicts?

☞ A Doubly-logarithmic Paradigm

Assume that h = log log n is an integer ($n = 2^{2^h}$).

Partition by $\sqrt{n}$:

$$\begin{array}{llll} \mathrm{A}_1 & = & \mathrm{A}(1), \quad \cdots, \quad \mathrm{A}(\sqrt{n}) & \Rightarrow M_1 \sim T(\sqrt{n}), W(\sqrt{n}) \\ \mathrm{A}_2 & = & \mathrm{A}(\sqrt{n}+1), \quad \cdots, \quad \mathrm{A}(2\sqrt{n}) & \Rightarrow M_2 \sim T(\sqrt{n}), W(\sqrt{n}) \\ & \cdots & \cdots & \\ \mathrm{A}_{\sqrt{n}} & = & \mathrm{A}(n-\sqrt{n}+1), \quad \cdots, \quad \mathrm{A}(n) & \Rightarrow M_{\sqrt{n}} \sim T(\sqrt{n}), W(\sqrt{n}) \end{array}$$

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

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

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

☞ A Doubly-logarithmic Paradigm

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

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

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

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

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

☞ Random Sampling $T(n) = O(1),\ W(n) = O(n)$

with *very high probability*, on an Arbitrary CRCW PRAM

A: n elements

random

$\underbrace{n^{1/8} \mid n^{1/8} \mid \text{B: } n^{7/8} \text{ elements of A} \mid n^{1/8}}_{n^{3/4}\ \text{blocks}}$

$T = O(1)$

$W = O(n^{7/8})$

$$M_i^{(1)} \sim T = O(1),\ W_i = O((n^{1/8})^2) = O(n^{1/4})$$

$\Rightarrow$ $T(n) = O(1)$, $W(n) = O(n)$

$\underbrace{n^{1/4} \mid \text{B: } n^{3/4}\, M^{(1)} \text{ elements of A} \mid n^{1/4}}_{n^{1/2}\ \text{blocks}}$

$$M_i^{(2)} \sim T = O(1),\ W_i = O((n^{1/4})^2) = O(n^{1/2})$$

$\Rightarrow$ $T(n) = O(1)$, $W(n) = O(n)$

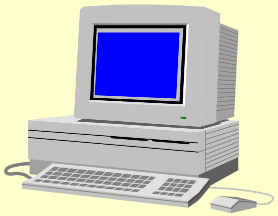
$$M(n^{7/8}) \sim T = O(1),\ W = O(n)$$

☞ Random Sampling

$$M(n^{7/8}) \sim T = O(1),\ W = O(n)$$

```
while (there is an element larger than M) {
    for (each element larger than M)
        Throw it into a random place in a new B(n^{7/8});
    Compute a new M;
}
```

【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*.



Research Project 8

MapReduce (26)

MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. A MapReduce program is composed of a Map( ) procedure and a Reduce( ) procedure.

In this project, you are supposed to briefly introduce the framework of MapReduce, and implement a MapReduce program to count the appearance of each word in a set of documents.

Detailed requirements can be downloaded from https://pintia.cn/

Reference:

Thinking in Parallel：Some Basic Data-Parallel Algorithms and Techniques; ***Uzi Vishkin*. *Class notes, 2010***