ASSIGNMENT- 4

Advanced Algorithms

HARSHITA SHARMA
(20171099)

① Prefix sum in EREW PRAM Model.

Upward Traversal can be carried out as in but for EREW, since no two processors are allowed to read/write simultaneously to same shared memory cell – we can carry out the odd/even indices separately like so –

upward traversal:

```
prefix( a[1:n], n) {
    if (n==1) { s[0] = a[0];          → 1 , 1
                return s;
    }
    for( i=0 to n/2-1) {
          x[i] = a[2i] + a[2i+1]      → 1 , n/2
    }
    y= prefix(x, n/2-1);              → T(n/2) , W(n/2)
```

downward traversal:

```
    for odd j in {0 : n-1}
          s[j] = y[j/2]  // parent   → 1 , n/2
    for even j in {0:n-1}
          s[j] = y[j/2] + a[j]; //self + uncle → 1, n/2
    return S;
```

Time Complexity: $T(n) = 1 + 1 + T(n/2) + 1 + 1$

$$\boxed{T(n) = O(\log n)}$$

Work Complexity: $W(n) = 1 + \frac{n}{2} + W(n/2) + n/2 + n/2$

$$\boxed{W(n) = O(n)}$$

(2) Optimal $O(\log \log n)$ time merging algorithm in CREW PRAM model.

a. If we take subsections of size k from array A and B such that $k \leq \lceil \log \log n \rceil$

Hence, $A = A_1, A_2 \dots A_n$

$B = B_1, B_2 \dots B_m$

b. Then, we merge

$\bar{A} = (a_1, a_2, a_3 \dots a_n)$ and

$\bar{B} = (b_1, b_2 \dots b_m)$

where a_i is first element of A_i and
b_i is first element of B_i

This can be done using the following algorithm:

- Rank ~~the~~ a set of $\sqrt{m}$ elements of $\bar{B}$ into parts of almost equal lengths in sorted sequence ~~of~~ $\bar{A}$. (by applying parallel search)
- Hence, the computed ranks of the chosen elements will create a partition on $\bar{A}$ into ~~$\sqrt{}$~~ sections s.t. each section of $\bar{A}$ has to fit between two chosen elements of $\bar{B}$.
- Now, the overall problem is reduced to ranking elements of each part of $\bar{B}$ into a corresponding section of $\bar{A}$.

— $O(\log \log n)$ time, $O(n)$ operations

c. Since each a_i now can be located in a part B_{ci} hence its rank in B can be found since $|B_{ci}| \leq \lceil \log \log n \rceil$ — ~~wh~~ this is done concurrently. — $O(\log \log n)$ time $O(n)$ op

Similarily Ranks of b_i in A are also found concurrently. — $O(\log \log n)$ time, $O(n)$ op.

d. Finally, For each i, ranks of elements (remaining) in each A_i and B_j. (for each j) are found.
As the merging problem is now reduced to a set of non-overlapping merging subproblems — and each corresponding pair has $O(\log \log n)$ elements.
Hence, these subproblems can be solved concurrently in $O(\log \log n)$ time. $O(n)$ operations

③ For a parallel search algorithm,

Time Complexity $= O(\log_p n)$

$\because$ given time complexity $= O(\log \log n)$

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

$$\log_p n = \log \log n$$

$$\frac{\log n}{\log p} = \log \log n$$

$$\log p = \frac{\log n}{\log \log n}$$

$$\Rightarrow \quad p = n^{\frac{1}{\log \log n}}$$

$$\Rightarrow \boxed{\text{no. of processors} = n^{\frac{1}{\log \log n}}}$$

<u>Work complexity</u> : for a parallel search algorithm is $O(p \log_p n)$

$$\Rightarrow \boxed{\text{Work Complexity} = O\left(n^{\frac{1}{\log \log n}} \log_{n^{\frac{1}{\log \log n}}} n\right)}$$

$$= O\left(n^{\frac{1}{\ln \ln n}} \frac{\ln n}{\ln n^{\frac{1}{\log \log n}}}\right)$$

$$= O\left(n^{\frac{1}{\ln \ln n}} \frac{\ln n}{\frac{1}{\ln \ln n} \ln n}\right)$$

$$= O\left(n^{\frac{1}{\ln \ln n}} \times \ln \ln n\right)$$

④ N inputs, each of 'k' bits

So, for each of k bits if each input is checked –

```
for ( i = 1 to k ) {
    for ( j = 1 to N ) {   — O(1) in parallel c N processors
        if ( ith bit of jth input is set )
            write/set ith bit of ans to 1.
    }
}
```

~~So, if ... processors are involved,~~

So, if we use N processors, each to find i^{th} bit of answer, ~~our time~~ in parallel

Time Complexity : $O(\#\text{ bits}) = O(1)$

~~Work Complexity : $O(N) * O(1) = O(N)$~~

Work complexity : $O(N) * O(\#\text{ bits}) = O(N)$

(5)

We have n elements but only p processors. to perform the prefix-sum algorithm

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7 \quad a_8$

(1) Then each processor can sum an n/p section of the array to generate sum ~~(assume n/p is integer)~~.

(2) Then, using the last prefix sum on each processor a p-element parallel prefix algo is run

(3) On each processor, the results from step 1 & local prefix sum is combined.

Work / Time Complexity :

Step 1 :	Local prefix sum of $n/2$ elements	$- O(n/p)$ $- O(n/p * p) = O(n)$
step 2 :	Last prefix sum (parallely)	$- O(\log p)$ $- O(p)$
Step 3 :	updating n/p prefix sum	$- O(n/p)$ $- O(n)$

Time Complexity = $T(n, p) = O(n/p + \log p)$

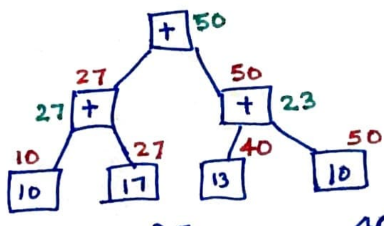
Work Complexity = $W(n, p) = O(n + p)$

Example : $n = 11, \ p = 4$

	2 3 5	7 1 9	3 4 6	8 2	← array
	$\lceil \frac{n}{p} \rceil$	$\lceil \frac{n}{p} \rceil$	$\lceil \frac{n}{p} \rceil$	$\lfloor \frac{n}{p} \rfloor$	
STEP 1:	2 5 10	7 8 17	3 7 13	8 10	← step 1 result
STEP 2:	10	17	13	10	

upward

downward



	10	27	40	50	← step2 result
STEP3: a)	10 − 10 = 0	27−17=10 ~~27−10=17~~	40−13=27	50−10=40	
~~STEP3:~~ b)	2 5 10	17 18 27	30 34 40	48 50	← step 3 & final result