Paradigms for Parallel Algorithms

Reference: C. Xavier and S. S. Iyengar, *Introduction to Parallel Algorithms*

Binary Tree Paradigm

- A binary tree with $n$ nodes is of height $\lceil \log n \rceil$
  - Can use this property in the design of parallel algorithms
  - Suppose there are $n$ data items, corresponding to the $n$ leaf vertices of a complete binary tree
  - Process them in parallel and get partial results at the non-leaf nodes
  - Proceed bottom up, and reach the root in $\lceil \log n \rceil$ time
Binary Tree Paradigm

- Sum of n Numbers
  - $O(n)$ time for a single processor
- Assume $n = 2^k$
- Example of finding the sum of 8 numbers

Algorithm SUM

Input: Array $A(1:n)$ where $n = 2^k$
Output: The sum of the values of the array stored in $A(1)$

BEGIN
1. $P = n/2$
2. 2. While $p > 0$ do
3.   For $i = 1$ to $p$ do in parallel
4.     $A(i) = A(2i-1) + A(2i)$
5.   End parallel
6. $p = \lceil p/2 \rceil$
7. End While
END

Complexity:
- $O(\log n)$ time using $O(n)$ processors
- EREW PRAM model
Growing by Doubling Paradigm

- If n entities are to be covered for processing
  - At the initial step, each processor covers a single entity and waits for consolidation
  - At the first step, the number of entities covered by each processor will be 2, and the second stage will be 4, and the third stage it is 8, and so on.
  - At every step the number of entities covered is double the entities covered at the previous step.
  - Summation of n numbers can be viewed as an example for growing by doubling.

Growing by Doubling Paradigm

- Another example is Link Ranking
  - The link list representation is extensively used as data structure for most of the classical problems.
  - If we want to reach the end of the linked list of n entries from the head, $O(n)$ time is needed using one processor.
  - Algorithm Link Ranking provides a method to reach the end in $O(\log n)$ time using $O(n)$ processors
  - $\text{LINK}(i)$ - the index of the number next to $A(i)$
  - $\text{LINK}(i) = 0$ if $A(i)$ is the last entry in the linked list
  - $\text{HEAD}$ contains the index of the first element
  - The rank of the number is defined to be its distance from the end. The last number in the linked list is 1 and the first entry of the linked list is of rank n.
Growing by Doubling Paradigm

**Algorithm List ranking**

**Input:** Array A(1:n), LINK(1:n) HEAD where n = 2^k

**Output:** RANK(1:n)

**BEGIN**

1. For i = 1 to n do in parallel
   - RANK(i) = 1
   - NEXT(i) = LINK(i)
   End Parallel

2. For j = 1 to k do
   For i = 1 to n do in parallel
     If NEXT(i) is not zero
       RANK(i) = RANK(i) + RANK(NEXT(i))
       NEXT(i) = NEXT(NEXT(i))
     Endif
   End Parallel

Endfor

**END**

**Complexity:**

O(log n) time using O(n) processors

**EREW PRAM model**

---

**Growing by Doubling Paradigm**

**Input:** Array A(1:n), LINK(1:n) HEAD where n = 2^k

**Output:** RANK(1:n)

**BEGIN**

1. For i = 1 to n do in parallel
   - RANK(i) = 1
   - NEXT(i) = LINK(i)
   End Parallel

2. For j = 1 to k do
   For i = 1 to n do in parallel
     If NEXT(i) is not zero
       RANK(i) = RANK(i) + RANK(NEXT(i))
       NEXT(i) = NEXT(NEXT(i))
     Endif
   End Parallel

Endfor

**END**

**Complexity:**

O(log n) time using O(n) processors

**EREW PRAM model**
Divide and Conquer Paradigm

- Problem is divided into smaller subproblems of the same kind
- The solution of these subproblems are found first.
- Then these are processed further, to get the solution of the complete problem.
- Illustrated in the following summation algorithms

An array of $n$ numbers is partitioned into $(n/\log n)$ groups, each containing $\log n$ entries.

Assign each group a processor. We need $n/\log n$ processors and each processor gets $\log n$ elements. Then each processor adds these elements sequentially in $O(\log n)$ time.

Using the algorithm given earlier, we then add these $n/\log n$ partial sums using $O(n/\log n)$ processor in $O(\log(n/\log n)) = O(\log n)$ time.

So this method solves the summation problem in $O(\log n)$ time, using $O(n/\log n)$ processors.

It is a work optimal algorithm.

---

### Algorithm Optimal-SUM

**Input:** Array $A(1:n)$ where $n = 2^k$

**Output:** The sum of these numbers in a variable called SUM

BEGIN

1. For $i = 1$ to $n/\log n$ do in parallel
   - Using the sequential method to find the sum of $A((i-1)\log n + 1), A((i-1)\log n + 2), \ldots, A(i\log n)$ and store the result in variable $B_i$
   - End parallel

2. Find the sum of $B_1, B_1, \ldots, B_{n/\log n}$ and store in the variable SUM

END

**Complexity:**

- $O(\log n)$ time
- $O(n/\log n)$ processors
- EREW PRAM model
Partitioning Paradigm

- Pay more attention to the process of dividing the problem into subproblems
  - We carefully divide the problem into subproblems $P_1, P_1, \ldots, P_S$, in such a way that, when subproblems $P_1, P_1, \ldots, P_S$, are solved concurrently, the solution of the original problem is already available.
  - Illustrated in the Merging problem covered earlier in the class.
  - This method can be solved in $O(\log n)$ time, using $O(n/\log n)$ processors in EREW PRAM.