Parallel Algorithms

Improve the running time by distributing the processing over several processors.

Issues:

- Accessing and controlling processors
  - Single vs. multiple instructions
  - Local memory
- Accessing global memory
  - Synchronization
  - Conflicts
- Message propagation

Example: Cleaning a house

Machines

- Sequent (8 processors, multiple instructions, shared memory)
- Hypercube (8 processors, distributed memory, hypercube topology)
- CM2 (64,000 processors, single instruction)
- CM5 (32,000 processors, single instruction for each 8000 processors)
- Intel Paragon (1,028 processors, 2D mesh topology)

- nCUBE (128 processors, MIMD distributed memory, hypercube topology)

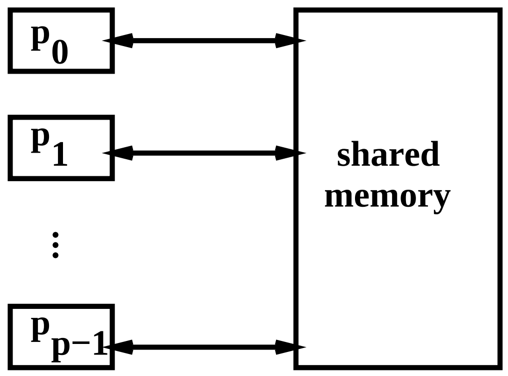
Models

RAM (serially random access machine)

PRAM (parallel RAM) unit time accesses

Spatial delayed accesses

PRAM Model



p serial processors

Shared memory

Unit time memory access

In reality, memory access time increases with number of processors

Running time measured in terms of number of memory accesses

Depends on size of input and number of processors

Each processor executes the same instruction at the same time

---

Memory Access

- ______________________________ — multiple processors read from same memory location at the same time.
- ________________________________ — multiple processors write to same memory location at the same time.
- _____________________________ — no two processors read from same memory location at the same time.
- ______________________________ — no two processors write to same memory location at the same time.
- EREW, CREW, ERCW, and CRCW are all PRAM models
- EREW and CRCW are most prevalent
- EREW is easy to implement (constraint maintained by programmer)
- In the CRCW model, need to determine what to do when two or more processors write to the same location
  - ______________ — select actual written value arbitrarily.
  - ____________ — write value from processor with lowest index.
  - ______________ — write some combination of values (sum, maximum).
  - _________ — processors must all write same value.

Synchronization And Control

Processors execute same instructions at same time

Stopping conditions depend on state of all processors

Processors interconnected somehow

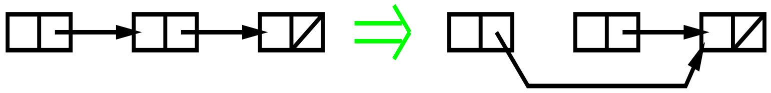
Propagation time

CRCW PRAM can use concurrent writes to detect termination conditions

---

Pointer Jumping

- Parallel operation on linked lists
- next(i) = next(next(i))



- __________ time operations on lists with n elements

List Ranking

For each object in the list, compute distance to the end of the list.

$$d(i) = \begin{cases} 0 & \text{if } next(i) = \text{NIL} \\ d(next(i)) + 1 & \text{if } next(i) \neq \text{NIL} \end{cases}$$

Serial: $\Theta(n)$

Parallel: O(lg n)

Idea:

```
repeat
   pointer jump
   add number of objects jumped to current distance
until all next(i) = NIL
```

Speedup: S = T (best serial algorithm) / T (parallel algorithm)

$S = \frac{O(n)}{O(\lg n)}$

Pseudocode

```
List-Rank(L)
   foreach processor i, in parallel                 ; O(1)
      if next(i) = NIL
      then d(i) = 0
      else d(i) = 1
   while next(i) $\neq$ NIL for some i               ; test
      foreach processor i, in parallel              ; O(1)
         if next(i) $\neq$ NIL                       ; synchronize
         then d(i) = d(i) + d(next(i))
              next(i) = next(next(i))
```

Click mouse to advance to next frame.

List-Rank halves the lengths of the lists each time, thus O(lg n) times through while loop.

The algorithm has run time performance of $\Theta(\lg n)$.

Another Performance Measure

Work = (number of processors) x (running time)

ListRank: Work = $\Theta(n \lg n)$

A PRAM algorithm A is **work-efficient** with respect to another algorithm B if Work(A) = $\Theta(Work(B))$.

Usually B is the best serial algorithm.

ListRank:

Serial: Work = $\Theta(n)$ (one processor needs n time)

PRAM: Work = $\Theta(n \lg n)$

Is the ListRank algorithm work efficient? ___

Efficiency: Speedup / #Processors.

An algorithm is thus work-efficient if Efficiency = $\Theta(1)$.

Prefix Computation

Given the binary, associative operator $\otimes$ and $x_1..x_k$, compute $y_k = x_1 \otimes x_2 \otimes .. \otimes x_k$.

Let

[i,j] = $x_i \otimes .. \otimes x_j$,

[i,k] = [i,j] $\otimes$ [j+1,k], and

[i,i] = x_i.

We can use pointer jumping to change [i,i] to [1,i].

ListPrefix

```
ListPrefix(L)
    foreach processor i, in parallel
        y(i) = x(i)                              ; [i,i]
    while next(i) $\neq$ NIL for some object i
        foreach processor i, in parallel
            if next(i) $\neq$ NIL                       ; [i,j] $\otimes$ [j+1,k] $\rightarrow$ [i,k]
            then y(next(i)) = y(i) $\otimes$ y(next(i))
                next(i) = next(next(i))          ; pointer jump
   Click mouse to advance to next frame.
```

Analysis (same as List Rank)

Serial: $\Theta(n)$

PRAM: ________

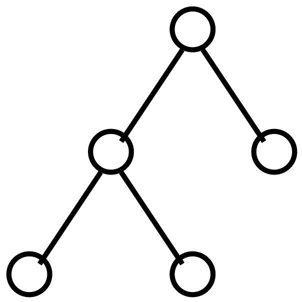
Speedup: S = _________

Efficiency: E = _________

Work: W = $\Theta(n \lg n)$

Is ListPrefix work efficient? ___

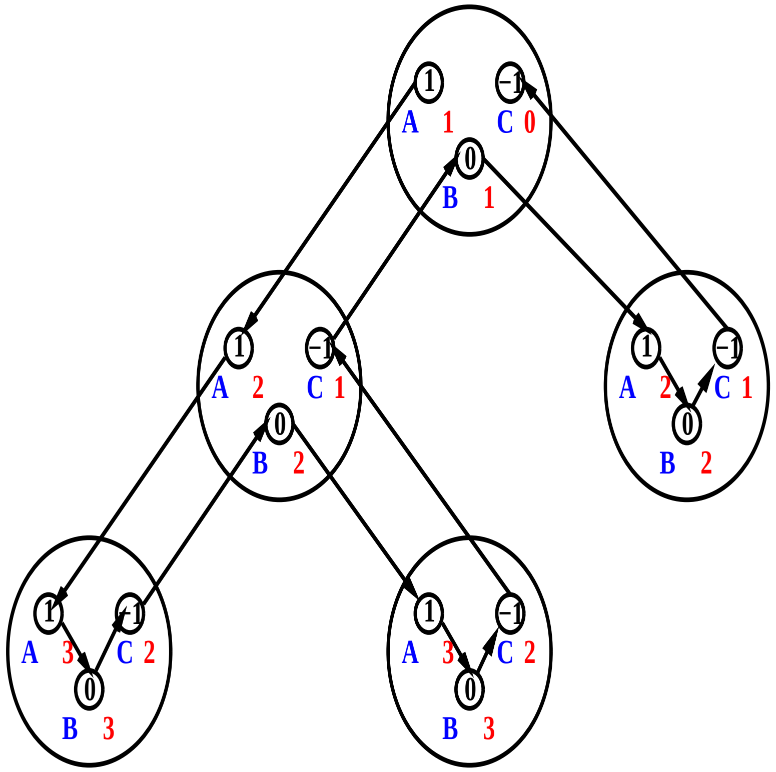
Computing Binary Tree Node Depth



Let each node be represented by three processors, A, B and C, which are part of a singly-linked list.

- A points to A of left child (if exists)
  or B (go down)
- B points to A of right child (if exists)
  or C (bottom out / backtrack)
- C points to B of parent (if left child)
  or C of parent (if right child)
  or NIL if root

Example



- Assign 1 to A (going down)
- Assign 0 to B (bottom out)
- Assign -1 to C (going up)
- These steps are O(1) given initial tree
- The depth of the node is stored in C
- A + B + C = 0
- The net effect of the subtree is 0
- Serial: $\Theta(n)$
- Run ListPrefix: lg 3n = O(lg n)

- $\otimes$ here is addition
- Work: W = $\Theta(nlgn)$

CRCW vs. EREW

1. CREW helps when finding roots of a forest of trees.

```
FindRoots(F)
    foreach processor i, in parallel
        if parent(i) = NIL
        then root(i) = i
    while some parent(i) $\neq$ NIL
        foreach processor i, in parallel
            if parent(i) $\neq$ NIL
            then root(i) = root(parent(i))
                parent(i) = parent(parent(i))
```

Click mouse to advance to next frame.

---

4 steps

- Multiple concurrent reads since multiple nodes have same parent
- If CR, then O(lg d), d = maximum depth of trees
  O(lg d) = O(lg lg n) for balanced trees.
- If ER, then the number of nodes that contain the correct root at most doubles each iteration, thus $\Omega(lgn)$.

CRCW vs. EREW

2. Common-CRCW helps find maximum value of array

Common-CRCW: concurrent writes write same value.

Given A[0, .., n-1], define n^2 processors to compare A[i] with A[j], n processors store m(i) = TRUE if A[i] $\geq$ every A[j]

Pseudocode

```
Fast-Max(A)
    n = length(A)
    for i = 0 to n-1, in parallel
        m(i) = true
    for i = 0 to n-1, j = 0 to n-1, in parallel      ; $n^2$ processors
        if A[i] < A[j]
        then m(i) = false                   ; CW
    for i = 0 to n-1, in parallel
        if m(i) = true
        then max = A[i]                     ; CW
    return max
```

```
Find Max of array containing 1, 2, 3, 4

Processor     Compares        m             max

   1            1, 1          false           4
   2            1, 2
```

3	1, 3	
4	1, 4	
5	2, 1	false
6	2, 2	
7	2, 3	
8	2, 4	
9	3, 1	false
9	3, 2	
9	3, 3	
9	3, 4	
9	4, 1	true
9	4, 2	
9	4, 3	
9	4, 4	

Analysis

O(1) FastMax, but $\Theta(n^2)$ work

EREW PRAM, __________ Again, the number of elements that know they are not max at most doubles each iteration.

Serial: $\Theta(n)$

Theorem 30.1

A p-processor CRCW can be no more than __________ times faster than the best p-processor EREW algorithm for the same problem.

Proof: Simulate each step of CRCW algorithm with a O(lg p) time EREW computation.

Theorem 30.4

What if we do not have enough processors?

Theorem: If a p-processor PRAM algorithm runs in time t, then for any p' < p, there is a p'-processor PRAM algorithm A' for the same problem that runs in time O(pt/p').

Example: p=10, p'=5, A = O(t)

A' = O(10/5 t) = O(t)

Work(A) = Work(A')

Empirical Results

- Compute sum of numbers (16,000,000 integers)
- MIMD distributed network of workstations using PVM
- Divide numbers over processors, host collects and combines partial sums
- Insert delay (10 units) to create readable results
- S = P
- One processor: 8.75 seconds
- Should be: (1, 8.750), (2, 4.375), (4, 2.187), (8, 1.094), (16, 0.547)
- Collected timings: (1, 8.750), (2, 4.460), (4, 3.360), (8, 2.840), (16, 2.830)

- Delay of 100 units should be: (1, 63.790), (2, 31.895), (4, 15.948), (8, 7.974), (16, 3.987)
- Collected timings: (1, 63.790), (2, 31.700), (4, 21.300), (8, 11.640), (16, 7.400)