COL380

Introduction to
Parallel & Distributed Programming

Shared-Memory Access Models

- EREW (Exclusive Read Exclusive Write)
  - ➡ Only one processors may read or write any given location in a step
- CREW (Concurrent Read Exclusive Write)
  - ➡ Many processors can simultaneously read a location, but only one may write
- CRCW (Concurrent Read Concurrent Write)
  - ➡ Many processors can read/write the same memory location
- ERCW (Exclusive Read Concurrent Write)
  - ➡ Not commonly used

Concurrent Write (CW)

- Priority CW
  - ➡ Higher priority processor (normally lower index) wins
- Common CW
  - ➡ Succeeds only if all writes have the same value
- Arbitrary/Random CW
  - ➡ One of the values is randomly chosen

Concurrent Write (CW)

- Priority CW
  - ➡ Higher priority processor (normally lower index) wins
- Common CW
  - ➡ Succeeds only if all writes have the same value
- Arbitrary/Random CW
  - ➡ One of the values is randomly chosen

EREW ≤ CREW ≤ Common ≤ Arbitrary ≤ Priority

Less powerful — More powerful

Subodh Kumar

Parallel Addition

p = n; B[i] = A[i]

p0 p1 p2 p3 p4 p5 p6 p7

p = p/2; if(i<p) B[i] = B[2i]+B[2i+1]

p0 p1 p2 p3

p = p/2; if(i<p) B[i] = B[2i]+B[2i+1]

p0 p1

p = p/2; if(i<p) B[i] = B[2i]+B[2i+1]

p0

```
p = n/2
forall i < n
   B[i] = A[i]
while(p > 0) {
   forall i < p
      B[i] = B[2i]+B[2i+1]
   p = p/2;
}
```

(assumes n is a power of 2)

- processors: n
- time: O(log n)
- Speed-up: n/(log n)
- Efficiency: 1/log(n)
- Cost: n log n
- Work: n

Subodh Kumar

Linear Search
$p<n$

- n input integers in n memory cells
- Does *x* exist in the input?
  - ➡ *x* is initially stored in shared memory

Algorithm

step1: If p0, broadcast x

step2: Processor pi: search in ith [n/p] block and {set flag fi}

step3: If p0, check if 'any' flag is 1 and print answer

EREW	CREW	CRCW
• log(p)	• 1	• 1
• n/p	• n/p	• n/p
• log(p)	• log(p)	• 1

Small Processor Count

- **Lemma**
  - ➡ Any problem that can be solved on a p-processor PRAM in t steps can be solved on a p'-processor PRAM in $t' = O(t*p/p')$ steps (assuming the same size of shared memory).

- **Proof**
  - ➡ Partition p *simulated* processors into p' groups of size p/p' each.
  - ➡ Allocate each group to one *available* processor
  - ➡ An available processor executes on behalf of its group, one step at a time
    - Execute all READ substeps
    - Execute all LOCAL computation substeps
    - Execute all WRITE substeps
  - ➡ Assumes local register space

Small Memory PRAM

- **Lemma**
  - ➡ Any problem that can be solved on a p-processor and m-cell PRAM in t steps can be solved on a $\max(p,m')$-processor and m'-cell PRAM in $O(tm/m')$ steps.

- **Proof:**
  - ➡ Partition m simulated cells into m' segments S_i, each of size m/m'
  - ➡ Each available processor P'_i simulates processor P_i, $i = [1..p]$
  - ➡ Each available processor P'_i, $i= [1..m']$, stores S_i in its local memory and uses $M'[i]$ as a public cell to simulate accesses to cells of S_i
  - ➡ To simulate **READ**, P'_i, $i=[1..\max(p,m')]$, repeats, for $k=[1..m/m']$:
    - write the value of the k-th cell of S_i into $M'[i]$, $i=[1..m']$
    - read the value if the original processor P_i, $i=[1..p]$, would read that cell
  - ➡ P'_i then simulates the computation substep of P_i, $i=[1..p]$, in one step

- **Lemma**
  - ➡ Any problem that can be solved on a *p*-processor and *m*-cell CREW PRAM in *t* steps can be solved on a *p*-processor and *m*-cell EREW PRAM in O(*tp*) steps.
- **Proof:**
  - ➡ Complete each READ substep in *p* rounds
  - ➡ In round i, processor P_i executes its READ

Possible in fewer steps?

Performance Evaluation

- Two parameters
  - ➡ $p(n)$, $t(n)$
- Generally, use work, $W(n)$
- If $W(n)$ similar, use $t(n)$
- Speedup/Scalability
  - ➡ Absolute: over best sequential algorithm
  - ➡ Relative: over the 1-processor implementation of the same algorithm

➡ Work-optimal => work = O(serial complexity)

➡ $p(n)$ is hidden but important

- $W_1(n) = O(n)$; $t_1(n) = O(n)$
- $W_2(n) = O(n \log n)$ and $t_2(n) = O(\log n)$

Work Time Scheduling Principle

- Design algorithm in terms of
  - Total work done per ‘time step’: $W_i(n)$
  - $t(n)$ steps
- Total work done $W(n) = \sum W_i(n)$
- For each time step i:
  - divide the work $W_i(n)$ among p processors
    - Time <= $\sum \lceil (W_i(n)/p \rceil$ <= $\lfloor W(n)/p) \rfloor + t(n)$
- Cost = $t(n,p) * p$

Work <= Cost. Cost optimality is more stringent.

Subodh Kumar

Brent's Theorem

- Time taken by p processors:
  - ➡ $t(n,p) = O(W(n)/p + t(n))$
- Cost = $p * t(n,p) = O(W(n) + p * t(n))$
- Work = Cost if:
  - ➡ $W(n) + p * t(n) = O(W(n))$
  - ➡ Or, $p = O(W(n)/t(n))$

Optimal Summation

- Using n processors for parallel sum
  - ➡ Work = O(n), Cost = O(n log n)
- Suppose, we use n/log n processors
  - ➡ forall: Compute sum of its share of log n local elements
  - ➡ Compute sum of n/log n partial sums
    - Using previous parallel sum algorithm (with n/log n processors)
- Time = log n + log(n/log n) = log(n)
- Cost = n/log n * log n = n

Favor long "sequential" sections

Notions of Optimality

- If sequentially optimal algorithm is $O(t'(n))$
  - ➡ Work done by Work-optimal parallel algorithm:
    - $O(t'(n))$ (with time $t(n)$).
  - ➡ Work-scheduling on p processors takes time:
    - $t(n,p) = O(t'(n)/p + t(n))$
  - ➡ Optimal speed-up: $t'(n)/t(n,p) = \theta(p)$, if
    - $[p{*}t'(n)] / [t'(n)+p{*}t(n)] = \theta(p)$
- Work-time optimal if:
  - ➡ $t(n)$ cannot be improved