Limitations of PRAMS (Ch. 10)

Relationship between PRAM Models

\[ \text{EREW} \leq \text{CREW} \leq \text{CRCW} \text{ (arbitrary} \leq \text{common} \leq \text{priority)} \]

Theorem (10.1) A concurrent read or write instruction of a \( p \) processor priority CRCW PRAM can be implemented on a \( p \) processor EREW PRAM in \( O(\log p) \) time.

Proof (sketch)

Read Operation (Write is similar + easier)

On CRCW: processor \( Q_i \) reads loc \( M_{j_i} \), \( 1 \leq i \leq p \)

On EREW: processor \( P_i \) simulates \( Q_i \)

- reserve mem locs \( M_1, M_2, \ldots, M_p \) for simulation
- \( O(1) \) 1. \( P_i \) writes pair \( <j_i, i> \) to \( M_i \)
- \( O(\log p) \) 2. in parallel, lexicographically sort pairs in \( M_1, M_2, \ldots, M_p \)
  \[ \text{now } M_1, M_2, \ldots, M_p \text{ organized in blocks where each } x/ \]
  \[ \text{block repres. procs wanting to read same value } x/ \]
- \( O(1) \) 3. First elt in each block performs read
- \( O(\log p) \) 4. Propagate value read to all els in block w/ segmented prefix sums
- \( O(1) \) 5. Write values back to correct \( M_i \) locs
- \( O(1) \) 6. \( P_i \) reads its desired value from \( M_i \)

\[ e.g. \quad p=4 \quad Q_i \quad \begin{array}{c|cccc}
  & 1 & 2 & 3 & 4 \\
\hline
M_{ji} & 6 & 5 & 5 & 6
\end{array} \]

\[ M_5 = x \quad M_6 = y \]

\[ \begin{array}{cccc}
  1 \quad M_1: & <6,1> & <5,2> & <5,3> & <6,4> \\
  2 \quad <5,2> & <5,3> & <6,1> & <6,4> \\
  3 \quad x & X & Y \\
  4 \quad x & X & Y & y \\
  5 \quad M: & Y & X & X & Y
\end{array} \]
Corollary (10.1) Let A be an alg that runs on a p-processor priority CRCW PRAM in time T. Then A can be implemented on a p-processor EREW PRAM in time $O(T \log p)$.

Note:
- CREW PRAM can simulate priority CRCW PRAM in same time ($O(\log p)$ slowdown)
- EREW PRAM can simulate CREW PRAM with $O(\log p)$ slowdown

⇒ in general, these results cannot be improved
⇒ relationship among CRCW PRAMs more subtle (see text)

Lowerbounds
- sequential lowerbound results give lowerbounds on the work for a parallel algorithm
- most interesting to examine lowerbounds on the time required to solve a problem in parallel

Most basic lowerbound result:
Lemma (Cor 10.2 in text): A CREW PRAM requires $\Omega(\log n)$ time to compute the Boolean OR of n variables, regardless of the number of processors available.

Corollary The following problems require $\Omega(\log n)$ time on a CREW PRAM w/ any # of processors
⇒ Sorting a sequence $x_1, x_2, ..., x_n$ where $x_i \in \{0, 1\}^n$
⇒ Computing $\text{sum} x_1, x_2, ..., x_n$ where $x_i \in \{0, 1\}^n$
⇒ Computing max of n inputs
The Class NC and P-Completeness

The Class NC (Nick's Class for Nick Pippenger @ UBC)
the set of problems that can be solved "efficiently" in parallel on a PRAM
⇒ fast w/ a reasonable number of processors
  • fast = time polylogarithmic in input size
    $O(\log^k n)$ for constant $k$ independent of $n$
  • reasonable # procs = # procs polynomial in $n$
    $O(n^c)$ for constant $c$ independent of $n$

Note: Similar to class P in that a very "bad" algorithm can put problem in NC, i.e., in reality not efficient e.g. $O(\log^{59} n)$ time + $O(n^{100})$ procs

Sometimes we further subdivide class NC

NC'$ = $ problems solvable in time $O(\log n)$
NC'' = problems solvable in time $O(\log^2 n)$

NC$k$ = problems solvable in time $O(\log^k n)$

Note:
(1) Thus far all problems we've studied are in NC
(2) Class RNC is analogous for randomized PRAM (w/ random # gen.)

Importance of class NC
• if a problem is not in NC then it is "hard to parallelize"
• similar to class P, i.e., if a problem is not in P, then we think it is hard to solve efficiently.
P-Completeness

P - class of probs solvable in polynomial time on sequential machine

NC \subseteq P?
- Yes - convert parallel alg to sequential alg
  - sequential running time \( O(n^c \log^k n) = O(n^{1+c}) \)

P \subseteq NC?
- i.e. can every problem solvable "efficiently" on uniprocessor
  be solved "efficiently" in parallel?
- Don't know... but we think not.

The class P-Complete consists of the problems in P
that we think are not in NC.

NC-Reductions

As for NP-Complete problems, we use reductions to build up
our class of P-Complete problems.

To reduce \( P_1 \) to \( P_2 \)
1. transform input \( x \) for \( P_1 \) to input \( x' \) for \( P_2 \)
2. run (assumed) algorithm for \( P_2 \) on \( x' \)
3. transform answer from 2 to get answer for \( P_1 \) on \( x \)

We say \( P_1 \) is NC-Reducible to \( P_2 \) if transformations
in steps 1 + 3 can be done w/ NC-algorithms

\implies \text{implys NC-algorithm for } P_2 \text{ can be used to obtain NC-algorithm for } P_1 \text{ (} P_2 \text{ is at least as hard as } P_1 \text{)}
Facts:
1. Suppose $P_1 \leq_{nc} P_2$
   Then, $P_2 \in NC$ implies $P_1 \in NC$
   /* $P_2$ is at least as hard as $P_1$ */

2. Suppose $P_1 \leq_{nc} P_2$ and $P_2 \leq_{nc} P_3$
   Then $P_1 \leq_{nc} P_3$
   /* $\leq_{nc}$ is transitive */

P-Completeness:
A problem $P_i$ is P-Complete if
(1) $P_i \in P$ and
(2) every problem $P' \in P$, $P' \leq_{nc} P_i$

Lemma Let $P'$ be a P-Complete Problem.
If $P' \in NC$, then $NC = P$.
/* if $P \neq NC$, as we believe, then no P-complete */
/* problem is in NC */

As w/ NP-Completeness, we need to start with one
P-Complete Problem. $P_i^*$
⇒ Need to show directly that every problem $P' \in P$
is NC-reducible to $P_i$.

Circuit Value Problem (CVP):
input: a Boolean Circuit $C$ represented by a $C = (g_1, g_2, \ldots, g_n)$
of gates (NOT, two input AND + OR gates) + set of inputs
question: does value of circuit = one?
Theorem (10.12) CVP is P-Complete

Now, we can show other problems are P-complete by reducing these to CVP to them. 
I actually can reduce any known P-complete problem to this.

As w/ NP-Completeness, we study decision problems? (why?)

1. Ordered DFS:
   - input: digraph \( G = (V,E) \) specified by adj lists and 
     \( S, u, v \in V \)
   - question: Is \( u \) visited before \( v \) in DFS of \( G \) starting at \( s \)?

2. Max Flow
   - input: network w/ integer valued capacities
     source \( s \) + sink \( t \)
   - question: Is the value of the max \( s-t \) flow odd?

3. Linear Inequalities (LI)
   - input: \( n \times n \) matrix \( A \) + \( n \) dimensional vector \( b \) (integers)
   - question: Is there a rational \( n \)-dim vector \( x \) s.t. \( Ax \leq b \)?

Book shows all 3 above are P-Complete.
Theorem (10.17) LI is P-Complete

Proof

(1) LI ∈ P

(2) CVP ≤_P LI /* So NC-alg for LI gives NCalg for CVP */

• let C = ⟨g₁, g₂, ..., gₙ⟩ w/ inputs be instance of CVP.

• with each node gᵢ of C we associate a variable xᵢ and a set of inequalities as follows:

  (idea: force xᵢ to be 0 or 1 whenever inputs are 0 or 1)

  (i) if gᵢ is an input node:
      xᵢ = 1 if gᵢ = 1
      xᵢ = 0 if gᵢ = 0

  (ii) if gᵢ = gⱼ ∧ gₖ:
      \[-xᵢ ≤ 0 (xᵢ ≥ 0)\] if xⱼ, xₖ ∈ {0, 1}
      \[xᵢ - xⱼ ≤ 0 \quad \text{if } xᵢ - xⱼ ≤ 0\] if xᵢ - xⱼ ≤ 0
      \[xᵢ - xₖ ≤ 0 \quad \text{if } xᵢ - xₖ ≤ 0\] if xᵢ - xₖ ≤ 0
      \[xⱼ + xₖ - xᵢ ≤ 1\] proper value for AND

      e.g. if xⱼ = xₖ = 1 then xᵢ = 1 (xᵢ = 0)

  (iii) if gᵢ = gⱼ ∨ gₖ:
      \[xᵢ ≤ 1 \quad \text{if } xᵢ ≤ 1\] if xⱼ, xₖ ∈ {0, 1}
      \[xⱼ - xᵢ ≤ 0 \quad \text{if } xⱼ - xᵢ ≤ 0\] if xⱼ - xᵢ ≤ 0
      \[xₖ - xᵢ ≤ 0 \quad \text{if } xₖ - xᵢ ≤ 0\] if xₖ - xᵢ ≤ 0
      \[xᵢ - xⱼ - xₖ ≤ 0\] proper value for OR

      e.g. if xⱼ = xₖ = 0 then xᵢ = 0 (xᵢ = 1)

• Also add:

  \[-xₙ ≤ -1\]

  \[xₙ ≤ 1\]

⇒ output of circuit is 1 ⇔ xₙ = 1

⇒ there (corresponding linear system has a feasible solution)

⇒