Parallel complexity

Jean-Louis Roch, Grenoble Univ.

Books / Readings

- Parallel algorithms for shared memory machine, RM Karp, V Ramachandran, Chap 17, HTCS, volA "Algorithms and Complexity" pp 871—932
- Limits to parallel computation - *P*-Completeness Theory
  Ray Greenlaw, Jim Hoover, and Larry Ruzzo
- An introduction to Parallel Algorithms, J. Jaja
- Slides from Ray Greenlaw: An Introduction to Parallel Computation and *P*-Completeness Theory,

Outline

- Introduction
- Parallel Models of Computation
- Basic Complexity – NC and Reductions
- *P*-Complete Problems
- Open Problems
- Parallel evaluation of arithmetic circuits

2

Introduction

- Sequential computation: *Feasible* ~ $n^{O(1)}$ time (polynomial time).
- Parallel computation: *Feasible* ~ $n^{O(1)}$ operations (or processors) (polynomial work).
- Goal of parallel computation: to develop fast algorithms: *feasible highly parallel*
  Both *polylog time* ~ $\log^{O(1)} n$ and polynomial work ~$n^{O(1)}$ (procs).
- A problem is *inherently sequential* if it is feasible but has no feasible highly parallel algorithm for its solution.

3

Outline

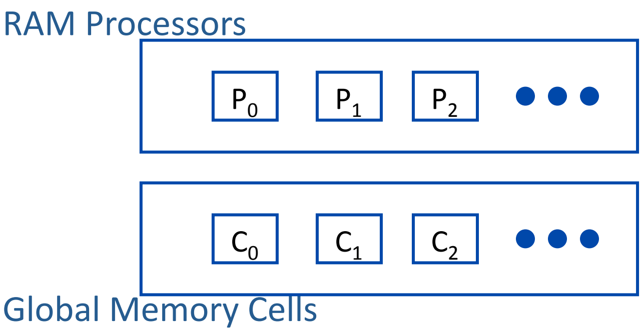
- Introduction
- **Parallel Models of Computation**
- Basic Complexity – NC and Reductions
- *P*-Complete Problems
- Open Problems
- Parallel evaluation of arithmetic circuits

4

Parallel Models of Computation

- Parallel Random Access Machine Model
- Boolean Circuit Model
- Circuits and PRAMs

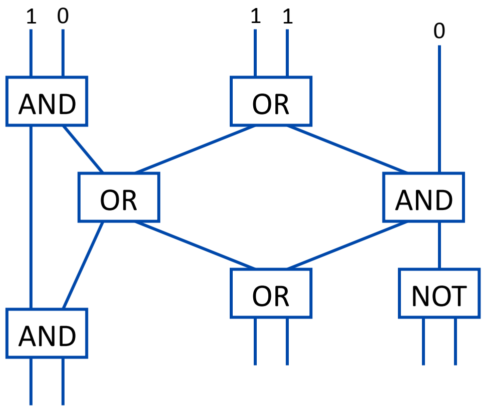
Parallel Random Access Machine = PRAM



Memory Access: EREW / CREW / CRCW [common/arbitrary/priority]

Theorem: A *priority-CRCW* PRAM that runs in time $t(n) = O(\log^k n)$ using $p(n) \in n^{O(1)}$ processors can be simulated by an EREW PRAM in time $t(n) = O(\log^{k+1} n)$ using $n^{O(1)}$ processors.

Boolean Circuit Model



Circuits and PRAMS

Theorem:

A function *f* from {0,1}* to {0,1}* can be computed by a logarithmic space uniform Boolean circuit family $\{\alpha_n\}$ with *depth*$(\alpha_n) \in (\log n)^{O(1)}$ and *size*$(\alpha_n) \in n^{O(1)}$

if and only if

f can be computed by a CREW-PRAM *M* on inputs of length *n* in time $t(n) \in (\log n)^{O(1)}$ using $p(n) \in n^{O(1)}$.

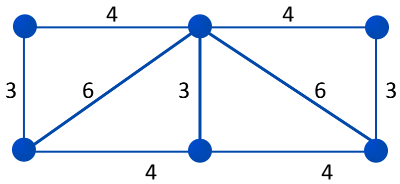
Outline

- Introduction
- Parallel Models of Computation
- **Basic Complexity – NC and Reductions**
- *P*-Complete Problems
- Open Problems
- Parallel evaluation of arithmetic circuits

Basic Complexity

- Decision, Function, and Search Problems
- Complexity Classes
- Reducibility
- Completeness

Decision, Function, and Search Problems



Spanning Tree-D
Given: An undirected graph $G = (V,E)$ with weights from N labeling edges in E and a natural number k.
Problem: Is there a spanning tree of G with cost less than or equal to k ?

Spanning Tree-F
Given: Same (no k).
Problem: Compute the weight of a minimum cost spanning tree.

Spanning Tree-S
Given: Same.
Problem: Find a minimum cost spanning tree.

Complexity Classes

Definitions:

P is the set of all languages L that are decidable in sequential time $n^{O(1)}$.

NC is the set of all languages L that are decidable in parallel time $(\log n)^{O(1)}$ and processors $n^{O(1)}$.

FP is the set of all functions from {0,1}* to {0,1}* that are computable in sequential time $n^{O(1)}$.

FNC is the set of all functions from {0,1}* to {0,1}* that are computable in parallel time $(\log n)^{O(1)}$ and processors $n^{O(1)}$.

NC^k, $k \geq 1$, is the set of all languages L such that L is recognized by a uniform Boolean circuit family $\{\alpha_n\}$ with $size(\alpha_n) = n^{O(1)}$ and $depth(\alpha_n) = O((\log n)^k)$.

NC - Reducibility

Definitions:

A language *L* is *reducible* to a language *L'*, written $L \leq L'$, if there is a function *f* such that: $x \in L$ if and only if $f(x) \in L'$.

L is *P reducible* to *L'*, written $L \leq^{P} L'$, if the function *f* is in *FP*.

For $k \geq 1$, *L* is NC^{k} *reducible* to *L'*, written $L \leq^{NC^k} L'$, if the function *f* is in FNC^{k}.

L is *NC many-one reducible* to *L'*, written $L \leq^{NC} L'$, if the function *f* is in *FNC*.

Turing-Reducibility: *A function f* is NC1-Turing-reducible to a function g, $f \leq_T^{NC1} g$, iff there exists a uniform circuit family $\{\alpha_n\}$ which gates are boolean or oracles for g, with $size(\alpha_n) = n^{O(1)}$ and $depth(\alpha_n) = O((\log n))$.

NB An oracle gate for g with m *inputs has depth* log m

Properties: $\leq^{P}$, $\leq^{NC^k}$ (k>1), $\leq^{NC}$ and $\leq_T^{NC1}$, $\leq_T^{NC}$ are transitive.

Thus: If $L \leq^{NC^k} L'$ and $L' \in NC^k$ (for k >1) then $L \in NC^k$.

13

Outline

- Introduction
- Parallel Models of Computation
- Basic Complexity
- **Example of reduction**
- *P*-Complete Problems
- Open Problems

14

Outline

- Introduction
- Parallel Models of Computation
- Basic Complexity – NC and Reductions
  - **Example of reduction**
- *P*-Complete Problems
- Open Problems
- Parallel evaluation of arithmetic circuits

15

Linear Algebra – DET class

- Triangular Matrix Inversion $\leq_T^{NC1}$ Matrix Power
- Matrix Power $\leq_T^{NC1}$ Triangular Matrix Inversion
- ***Sequential***: MatrixInversion=Θ(MatrixMultiplication)
- ***Parallel***: Matrix Multiplication <<
  MatrixInversion$=_T^{NC1}$ MatrixPower

Outline

- Introduction
- Parallel Models of Computation
- Basic Complexity – NC and Reductions
- ***P*-Complete Problems**
- Open Problems
- Parallel evaluation of arithmetic circuits

17

Completeness

Definitions:

A language *L* is *P-hard under NC reducibility* if $L' \leq_T^{NC} L$ for every $L' \in P$.

A language *L* is *P-complete under NC reducibility* if $L \in P$ and *L* is *P*-hard.

Theorem:

If any *P*-complete problem is in *NC* then *NC* equals *P*.

Remark:

It is conjectured that NC ≠ P (proved with R-arithmetic).

18

P-Complete Problems

There are approximately 175 *P*-complete problems (500 with variations).

<u>Categories</u>:

- Circuit complexity
- Graph theory
- Searching graphs
- Combinatorial optimization and flow
- Local optimality
- Logic
- Formal languages
- Algebra
- Geometry
- Real analysis
- Games
- Miscellaneous

Eg: Gaussian elimination with pivot: P-complete, but MatrixInversion is in NC^2

Circuit Value Problem

<u>Given</u>:
An encoding $\underline{\alpha}$ of a Boolean circuit α, inputs $x_1,\ldots,x_n$, and a designated output *y.*

<u>Problem</u>:
Is output *y* of α TRUE on input $x_1,\ldots,x_n$?

<u>Theorem</u>: [Ladner 75]
The Circuit Value Problem is *P*-complete under $\leq_m^{NC^1}$ reductions.

P-Complete Variations of CVP

- Topologically Ordered [Folklore]
- Monotone [Goldschlager 77]
- Alternating Monotone Fanin 2, Fanout 2 [Folklore]
- NAND [Folklore]
- Topologically Ordered NOR [Folklore]
- Synchronous Alternating Monotone Fanout 2 CVP [Greenlaw, Hoover, and Ruzzo 87]
- Planar [Goldschlager 77]

NAND Circuit Value Problem

Given:

An encoding $\overline{\alpha}$ of a Boolean circuit α that consists solely of NAND gates, inputs $x_1,\ldots,x_n$, and a designated output *y*.

Problem:

Is output *y* of α TRUE on input $x_1,\ldots,x_n$?

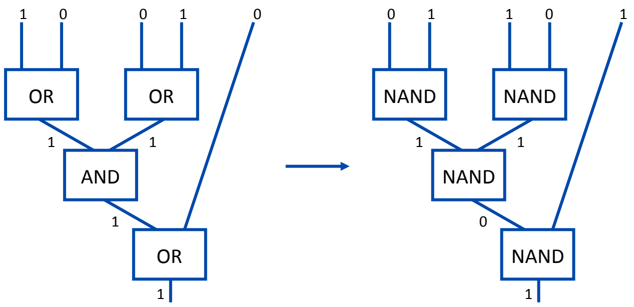
Theorem:

The NAND Circuit Value Problem is *P*-complete.

NAND Circuit Value Problem

Proof:

Reduce AM2CVP to NAND CVP. Complement all inputs. Relabel all gates as NAND.



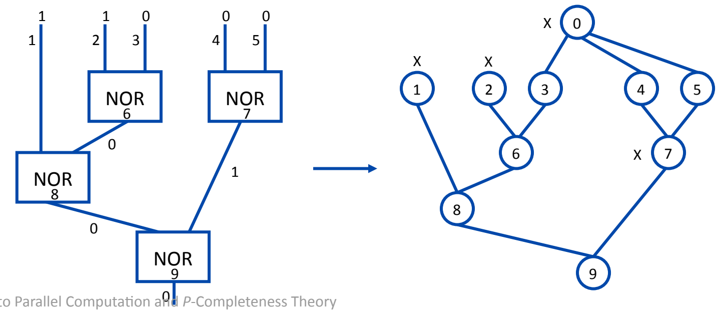
Graph Theory

- Lexicographically First Maximal Independent Set
  [Cook 85]
- Lexicographically First (Δ + 1)-Vertex Coloring
  [Luby 84]
- High Degree Subgraph
  [Anderson and Mayr 84]
- Nearest Neighbor Traveling Salesman Heuristic
  [Kindervater, Lenstra, and Shmoys 89]

Lexicographically First Maximal Independent Set

Theorem: [Cook 85]
LFMIS is *P*-complete.

Proof:
Reduce TopNOR CVP to LFMIS. Add new vertex 0. Connect to all false inputs.



Searching Graphs

– Lexicographically First Depth-First Search Ordering [Reif 85]
– Stack Breadth-First Search [Greenlaw 92]
– Breadth-Depth Search [Greenlaw 93]

Context-Free Grammar Empty

Given: A context-free grammar $G=(N,T,P,S)$.
Problem: Is $L(G)$ empty?

Theorem: [Jones and Laaser 76], [Goldschlager 81], [Tompa 91]
CFGempty is *P*-complete.

Proof: Reduce Monotone CVP to CFGempty. Given α construct $G=(N,T,P,S)$ with N, T, S, and P as follows:

Context-Free Grammar Empty

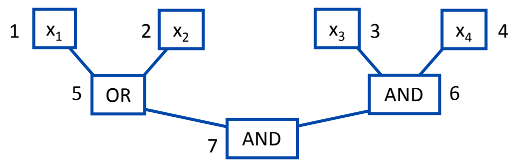
$N = \{i \mid v_i \text{ is a vertex in } \alpha\}$
$T = \{a\}$
$S = n$, where v_n is the output of α.
P as follows:

1. For input v_i, $i \rightarrow a$ if value of v_i is 1,
2. $i \rightarrow jk$ if $v_i \leftarrow v_j \wedge v_k$, and
3. $i \rightarrow j \mid k$ if $v_i \leftarrow v_j \vee v_k$.

Then the value of v_i is 1 if and only if $i \overset{*}{\Rightarrow} \gamma$, where $\gamma \in \{a\}^+$.

CFGempty Example

$x_1 = 0$, $x_2 = 0$, $x_3 = 1$, and $x_4 = 1$.



G = (N, T, S, P), where
N = {1, 2, 3, 4, 5, 6, 7}
T = {a}
S = 7
P = {3 → a, 4 → a, 5 → 1 | 2, 6 → 34, 7 → 56}

Outline

- Introduction
- Parallel Models of Computation
- Basic Complexity – NC and Reductions
- *P*-Complete Problems
- **Parallel evaluation of arithmetic circuits**
- Open Problems

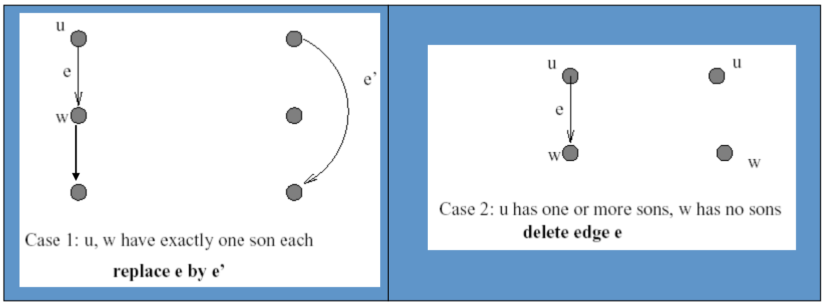
Circuits and parallelism

- General CVP is P-complete.
  - What subset instances are in P ?
- Arithmetic Expression evaluation
- Arithmetic Circuit evaluation

Tree contraction

- Tree-contraction is used in parallel expression evaluation
- Since the structure of a expression is a tree there are different tree-contraction techniques
- Basic operations are:
  - redirecting edges of the tree
  - removing nodes marking (pebbling) nodes
  - creating additional edges
- the final aim is to guarantee that logarithmic number of contractions is sufficient

Basic Tree contraction operations



tree-contraction related to SimSub

repeat
    for each edge e do in parallel
    perform local action on e
until there are no edges

Parallel pebble game on binary tree

- Within the game each node v of the tree has associated with it similar node denoted by cond(v).
- At the outset of the game cond(v)=v, for all v
- During the game the pairs (v,cond(v)) can be thought of as additional edges
- Node v is "active" if and only if cond(v)≠v

Operations: active, square and pebble

Activate

for all non-leaf nodes v **in parallel do**

if v is not active and precisely one of its sons is pebbled **then**

cond(v) becomes the other son

if v is not active and both sons are pebbled **then**

cond(v) becomes one of the sons arbitrarily

Square

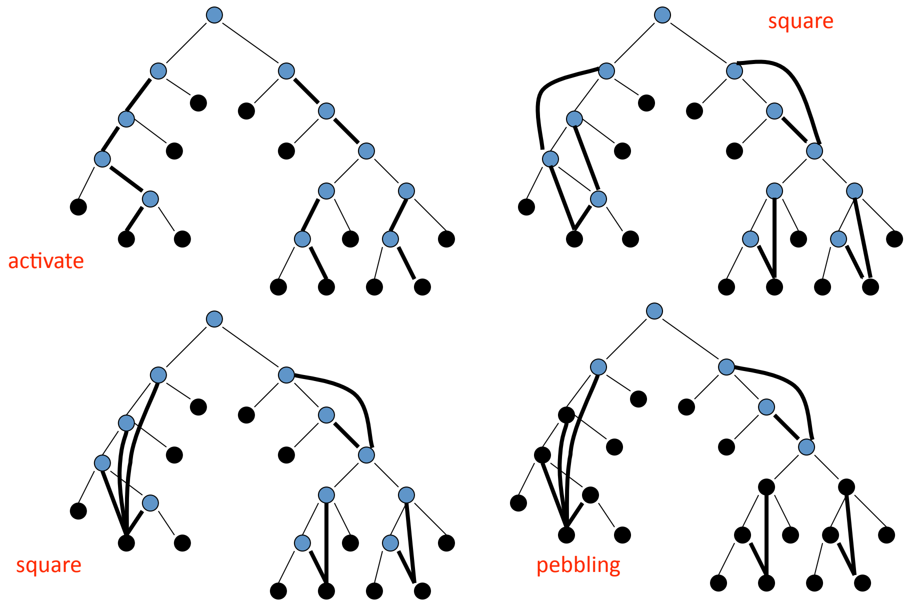
for all nodes v **in parallel do** cond(v)← cond(cond(v))

Pebble

for all nodes v **in parallel do**

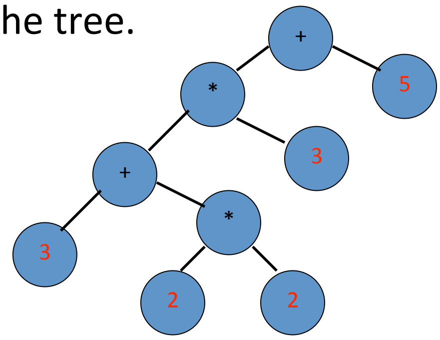
if cond(v) is pebbled **then** pebble v

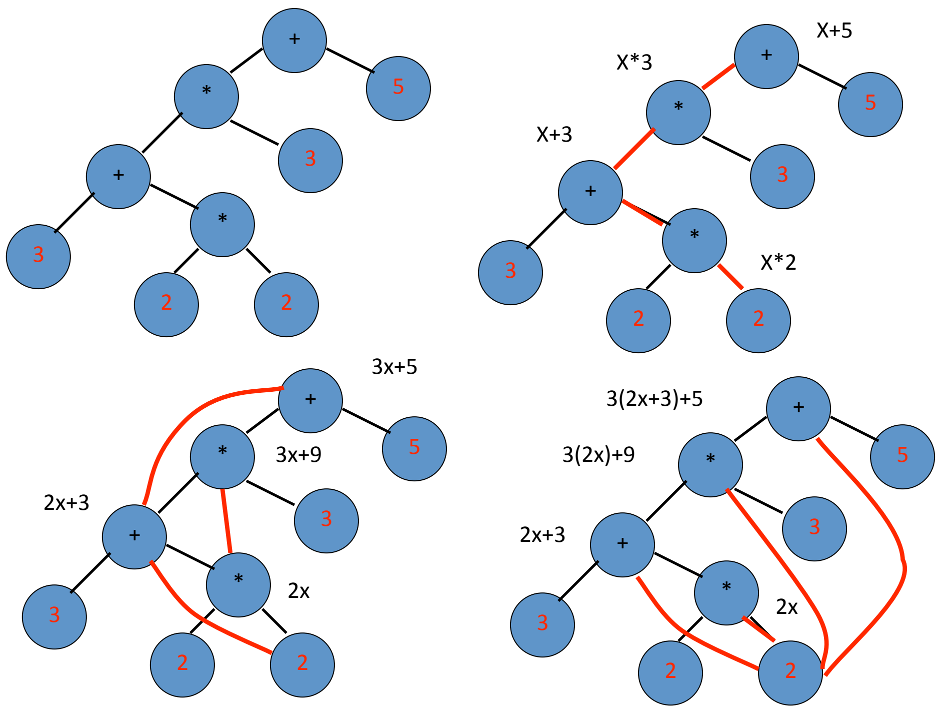
One step: `Activate;square;square; pebbling`



Application of the pebbling game

- Consider the arithmetic expression ((3+(2*2))*3+5)
- We assign a processor to each non-leaf node of the tree.





Expression evaluation

- Algorithm:
  while not(all nodes are evaluated) do
  { activate; square; square; pebble; }

- **Theorem**
  Let T be a binary tree with n leaves. After *$log_2 n$* stepsof the pebbling game, T is evaluated.

=> Arithmetic expressions can be evaluated on a PRAM in O(log n) time using O(n) processors.

Circuit evaluation

- Straight line arithmetic program
  - (+, *) in a semi-ring (extensionq to boolean or to a a field)
  - Circuit with arithmetic gates : n-ary + and binary * (and dummy+ to avoid non consecutive *)
- Algorithm: Loop while not (all nodes evaluated) {
  - 1. MM (gather +nodes)
  - 2. Rake (eval nodes with leaves)
  - 3. Shunt (bypass * nodes with only one son not evaluated)

  }

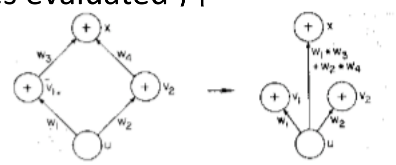


Fig. 4. The MM operation.

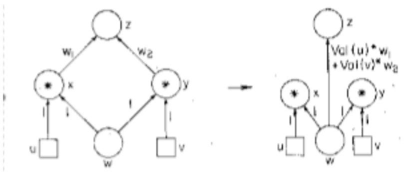


Fig. 5. The Shunt operation.

Circuit evaluation [Miller Ramachandran Kaltofen]

- Consider a straight line arithmetic program
  - (+, *) in a semi-ring
  - Each output can be seen as a polynomial in the input
- Let n = # gates; let d= max. degree of an output gate w.r.t. input gates
- **Theorem**: MRK straight line evaluation evaluates the circuit in Depth = (log n)(log d + log n) and Work = $O(M(n))=O(n^3)$
- **Application**: triangular linear system inversion: k=dim(system)
  - Sequential: $n = k^2$ degree= k
  - => circuit with depth = $O(\log^2 k)$ and work $O(k^6)$

Outline

- Introduction
- Parallel Models of Computation
- Basic Complexity – NC and Reductions
- *P*-Complete Problems
- Parallel evaluation of arithmetic circuits
- **Open Problems**

42

Open Problems

Find an *NC* algorithm or classify as *P*-complete:

- Edge Ranking
- Edge-Weighted Matching
- Integer Greatest Common Divisor
  - Polynomial GCD is in DET, so in NC2.
- Modular Powering

43