Parallel Computation

- Goal: parallel algorithms should be dramatically better that sequential ones (for a polynomial sequential one, parallel with polylogarithmic ($\log^k n$) step count) but without requiring inordinately large numbers of processors.
- The amount of work done by a parallel algorithm can be no smaller that the time complexity of the best sequential algorithm.

Models of Parallel Computation

- TMs and RAMs are sequential because of the *von Neumann property*: at each instant only a bounded amount of computational activity can occur.
- Boolean circuits are genuinely parallel.
- *Uniform families of Boolean circuits* are used as the basic model of parallel algorithms.
- Complexity measures for parallel computation: *parallel time and parallel work*

- Let $C = (C_0, C_1, \ldots)$ be a uniform family of Boolean circuits and let $f(n)$ and $g(n)$ be functions from integers to integers.

  Parallel time of C is at most $f(n)$ if for all n the *depth* of C_n is at most $f(n)$.

  Parallel work of C is at most $g(n)$ if for all n the *size* of C_n is at most $g(n)$.
- $\mathbf{PT}/\mathbf{WK}(f(n), g(n))$ is the class of languages $L \subseteq \{0,1\}^*$ such that there is a uniform family of circuits C deciding L with $\mathcal{O}(f(n))$ parallel time and $\mathcal{O}(g(n))$ parallel work.

Example. REACHABILITY $\in \mathbf{PT}/\mathbf{WK}(\log^2 n, n^3 \log n)$

Parallel Random Access Machines

- How a realistic model of parallel computation is a circuit? Correspond to parallel random access machines.
- A PRAM program is a set of RAM programs $P = (\Pi_1, \ldots, \Pi_q)$, one for each q RAMs.
- Each RAM executes its own program and has its own program counter and accumulator but shares all registers.
  For concurrent writes the RAM with the smallest index prevails: *PRIORITY CRCW PRAM*
- The number q of RAMs and their program depend on the number m of input integers and their total length n.

- $\mathcal{P} = \{P_{mn} \mid m, n \geq 0\}$ is a *uniform family of PRAMs* if there is a TM which given $1^m 0 1^n$ generates $q(m,n)$ and programs $P_{mn} = (\Pi_{m,n,0}, \Pi_{m,n,1}, \ldots, \Pi_{m,n,q(m,n)})$ in logarithmic space.
- Consider F a function from finite sequence of strings of integers to finite sequence of integers;
  Let $\mathcal{P} = \{P_{mn} \mid m, n \geq 0\}$ be a uniform family of PRAMs and $f(n)$ and $g(n)$ functions from pos. integers to pos. integers.
- $\mathcal{P}$ *computes F in parallel time f with g processors* if for each $m, n \leq 0$, (i) P_{mn} has $q(m,n) \leq g(n)$ processors and (ii) if P_{mn} is executed on input I of m integers with total length n, than all $q(m,n)$ RAMs reach a HALT instruction after at most $f(n)$ steps and the $k \leq q(m,n)$ first registers contain the output $F(I)$.

- (PRIORITY CWRW) PRAM is a most idealized and powerful model (unrealistic: instantaneous communication; conc. writing)
- *PRAMs can simulate circuits*:
  If $L \subseteq \{0,1\}^*$ is in $\mathbf{PT/WK}(f(n), g(n))$, then there is a uniform PRAM that computes the corresponding function F_L in parallel time $\mathcal{O}(f(n))$ using $\mathcal{O}(\frac{g(n)}{f(n)})$ processors.
- *Circuits can simulate PRAMS*:
  Let F be computed by a uniform PRAM in parallel time $f(n)$ using $g(n)$ processors ($f(n), g(n)$ from 1^n in log. space).
  Then there is a uniform family of circuits of depth $\mathcal{O}(f(n)(f(n) + \log(n))$ and size $\mathcal{O}(g(n)f(n)(n^k f(n) + g(n))$ which computes F (binary representation) where n^k is the time bound of the log-space TM computing the nth PRAM given 1^n.

The Class NC

- What is the class of problems satisfactorily solved by parallel computers?
  $$\mathbf{NC} = \mathbf{PT/WK}(\log^k n, n^k)$$
- $\mathbf{NC}$ is the class of languages decided by PRAMs in polylog parallel time and with polynomially many processors.
- Difference between polylog and poly is seen sometimes only for big inputs: $\log^3 10^8 > 18000$ and $(10^8)^{1/2} = 10000$
- Consider more refined classes: $\mathbf{NC}_j = \mathbf{PT/WK}(\log^j n, n^k)$
- Another candidate class for satisfactorily solved problems: $\mathbf{NC}_2$

- Clearly $\mathbf{NC} \subseteq \mathbf{P}$ but is $\mathbf{NC} = \mathbf{P}$?
- Conjecture: $\mathbf{NC} \neq \mathbf{P}$
  (There seem to be problems that are inherently sequential.)
- $\mathbf{P}$-complete problems are the least likely to be in $\mathbf{NC}$:
  $\mathbf{NC}$ is closed under reductions and so is $\mathbf{NC}_2$.
- ODD MAX FLOW (given a network, is the maximum flow value odd) is $\mathbf{P}$-complete.
  (So are MAX FLOW(D), HORNSAT, CIRCUIT VALUE)

The $\mathbf{L} \stackrel{?}{=} \mathbf{NL}$ problem

- Theorem: $\mathbf{NC}_1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{NC}_2$
- The *parallel computation thesis*:
  space and parallel time are polynomially related
- $\mathbf{PT/WK}(f(n), k^{f(n)}) \subseteq \mathbf{SPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n)) \subseteq \mathbf{PT/WK}(f(n)^2, k^{f(n)^2})$
- REACHABILITY is $\mathbf{NL}$-complete.
- 2SAT is $\mathbf{NL}$-complete.
- All languages in $\mathbf{L}$ are $\mathbf{L}$-complete!

Alternation

- A generalization of nondeterminism
- In a nondeterministic computation each configuration is an implicit *or* of its successors:
  a configuration "leads to acceptance" if at least one of its successors leads to acceptance.
- The generalization: allow both *OR* and *AND* configurations

Alternation — cont'd

Definition. An *alternating* Turing machine N is a nondeterministic Turing machine where the set of states K is partitioned into two sets $K = K_{\mathrm{AND}} \cup K_{\mathrm{OR}}$.

- Consider the tree of computations of N on input x.
- *Eventually accepting configurations*:
  (i) all leafs with state "yes" are eventually accepting.
  (ii) A configuration with state in K_{AND} is eventually accepting iff all its successors are.
  (iii) A configuration with state in K_{OR} is eventually accepting iff at least one of its successors is.

Alternation — cont'd

- N accepts x if its initial configuration is eventually accepting.
- N decides a language L if N accepts all strings $x \in L$ and rejects all strings $x \notin L$.
- Straightforward to define $\mathbf{ATIME}(f(n))$, $\mathbf{ASPACE}(f(n))$
- $\mathbf{AP} = \mathbf{ATIME}(n^k)$; $\mathbf{AL} = \mathbf{ASPACE}(\log n)$
- Alternating space is determ. time but one exponential higher.

Theorem. MONOTONIC CIRCUIT VALUE is $\mathbf{AL}$-complete

Corollary. $\mathbf{AL} = \mathbf{P}$

Corollary. $\mathbf{ASPACE}(f(n)) = \mathbf{TIME}(k^{f(n)})$