1. Parallel Algorithms

- A synchronous architecture with shared memory is assumed.
- The goal of parallel algorithms is to be dramatically better than sequential ones, preferably \( \text{polylogarithmic} \), i.e. the length of parallel computation is \( O(\log^k n) \) for some \( k \).
  (We use the notation: \( \log^k n = (\log n)^k \))
- However, the executions of parallel algorithms should not require inordinately large (superpolynomial) numbers of processors.

Observations

- The amount of work done by a parallel algorithm can be no smaller than the time complexity of the best sequential algorithm.
- Parallel computation is not the answer to \( \text{NP} \)-completeness:
  \[ \text{work} = \text{parallel time} \times \text{number of processors} \]
- If the amount of work is exponential, then either the number of parallel steps or the number of processors (or both) is exponential.
2. Parallel Models of Computation

- TM 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 and computation.
- The primary complexity measures for parallel computation are parallel time and parallel work.

Parallel time and 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. 
  - The parallel time of \( C \) is at most \( f(n) \) iff for all \( n \) the depth of \( C_n \) is at most \( f(n) \).
  - The parallel work of \( C \) is at most \( g(n) \) iff for all \( n \) the size of \( C_n \) is at most \( g(n) \).
- The class \( \text{PT/WK}(f(n), g(n)) \) consists of languages \( L \subseteq \{0, 1\}^* \) for which there is a uniform family of circuits \( C \) deciding \( L \) with \( O(f(n)) \) parallel time and \( O(g(n)) \) parallel work.

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

Parallel random access machines

- How realistic models of parallel computation are circuits? They correspond to parallel random access machines (PRAMs)!
- A PRAM program is a set of RAM programs \( P = (\Pi_1, \ldots, \Pi_q) \), one for each of the \( q \) RAMs.
- Each RAM \( \Pi_i \) executes its own program, has its own program counter and accumulator, i.e. the \( i \)th register, but shares all registers (including accumulators and input).
- For concurrent writes the RAM with the smallest index prevails: i.e. the PRIORITY CRCW PRAM is assumed (see note 15.5.7).

Uniform PRAM families

- PRAMs (under PRIORITY CRCW scheme) form a very idealized and powerful model which is also rather unrealistic due to instantaneous communication and concurrent writing.
- The number \( q \) of RAMs is a function \( q(m, n) \) of the number \( m \) of input integers in \( I = (i_1, \ldots, i_m) \) and their total length \( n = l(I) \).
- A family of PRAMs \( \mathcal{P} = \{P_{m,n} \mid m, n \geq 0 \} \) is uniform iff there is a TM which given \( 1^m0^n1 \) generates \( q(m, n) \) and the programs \( P_{m,n} = (\Pi_{m,n,0}, \Pi_{m,n,1}, \ldots, \Pi_{m,n,q(m,n)}) \) all in logarithmic space.
### Computing functions with PRAMs

- Let $F$ be a function from finite sequences of integers to finite sequences of integers; and $f(n)$ and $g(n)$ functions from positive integers to positive integers.
- Let $P = \{P_{m,n} \mid m,n \geq 0\}$ be a uniform family of PRAMs.

**Definition.** The family $P$ computes $F$ in parallel time $f$ with $g$ processors iff for each $m,n \geq 0$, for $P_{m,n}$ it holds that

(i) it has $q(m,n) \leq g(n)$ processors and

(ii) if $P_{m,n}$ is executed on input $I$ of $m$ integers with total length $n$, then 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 $F(I)$.

---

### Simulation results

- PRAMs can simulate circuits:
  - If $L \subseteq \{0,1\}^*$ is in $PT/WK(f(n),g(n))$, then there is a uniform PRAM that computes the corresponding function $F_L$ in parallel time $O(f(n))$ using $O(g(n))$ processors.
- Circuits can simulate PRAMs:
  - Let $F$ be computed by a uniform PRAM in parallel time $f(n)$ using $g(n)$ processors ($f(n),g(n)$ can be computed from $1^n$ in log space). Then there is a uniform family of circuits of depth $O(f(n)(f(n) + \log n))$ and size $O(g(n)f(n)(n^k f(n) + g(n)))$ which computes the binary representation of $F$.

(Here $n^k$ is the time bound of the log space TM computing the $n$th PRAM in the family given $1^n$ as its input.)

---

### 3. The Class NC

- What would be the class of problems that is satisfactorily solved by parallel computers? A candidate definition (Nick’s class):
  
  \[
  \mathbf{NC} = \mathbf{PT}/WK(\log^k n, n^k).
  \]

- NC is the class of languages decided by PRAMs in polylogarithmic parallel time and with polynomially many processors.
- However, the difference between polylog and polynomial is seen sometimes only for big $n$. For example, consider $\log^3 n$ and $\sqrt{n}$: $18000 < \sqrt{18000} < 10000$.
- One possibility is to consider subclasses of NC for $j = 1, 2, \ldots$:
  
  \[
  \mathbf{NC}_j = \mathbf{PT}/WK(\log^j n, n^j)
  \]
  — a potential hierarchy of classes.
- The class $\mathbf{NC}_2$ provides an alternative (more conservative) notion of “efficient parallel computation”.

---

### NC vs P

- Clearly $\mathbf{NC} \subseteq \mathbf{P}$ but is $\mathbf{NC} = \mathbf{P}$?
- There seem to be problems in $\mathbf{P}$ that are inherently sequential.
  - Conjecture: $\mathbf{NC} \neq \mathbf{P}$.
- Since $\mathbf{NC}$ (and $\mathbf{NC}_2$) is closed under log space reductions, $\mathbf{P}$-complete problems are the least likely to be in $\mathbf{NC}$.

**Example.** ODD MAX FLOW:
Given a network $N = (V,E,s,t,c)$, is the maximum flow value odd?

**Theorem.** ODD MAX FLOW is $\mathbf{P}$-complete.
(So are $\mathbf{MAX FLOW}(D)$, $\mathbf{HORNSAT}$, and $\mathbf{CIRCUIT VALUE}$.)
4. The $L = NL$ problem

We may relate logarithmic space classes and parallel complexity classes:

**Theorem.** $\text{NC}_1 \subset L \subset NL \subset \text{NC}_2$.

**Proof.**

1. The last inclusion follows by reachability method, since $\text{REACHABILITY}$ belongs to $\text{NC}_2$.
2. The inclusion in the middle is trivial.
3. For the first inclusion, we have to compose three algorithms that operate in logarithmic space (recall Proposition 8.2).

**Proof of $\text{NC}_1 \subset L$ - continued**

The following logspace algorithms are needed:

1. The first generates a circuit $C$ from the given uniform family.
2. The second transforms $C$ into an equivalent circuit/expression $E$ whose gates have all outdegree one (no shared subexpressions).
   - Each path in $C$ identifies a gate in $E$.
3. The third evaluates the output gate of the tree-like circuit $E$.
   - During the recursive evaluation, it is sufficient to remember the label of the gate being evaluated and its truth value.

$\square$ The composition operates in logarithmic space.

---

Parallel computation thesis

- Space and parallel time are polynomially related!
- This can be generalized beyond logarithmic space:

$$\text{PT/ WK}(f(n), k^f(n)) \subset \text{SPACE}(f(n)) \subset \text{NSPACE}(f(n)) \subset \text{PT/ WK}(f(n)^2, k^f(n)^2).$$

**Theorem.** $\text{REACHABILITY}$ is $\text{NL}$-complete.

**Theorem.** $2\text{SAT}$ is $\text{NL}$-complete.

Actually, all languages in $L$ are $L$-complete!

---

5. Alternation

- Alternation is an important generalization of nondeterminism.
- In a nondeterministic computation each configuration is an implicit $\text{OR}$ of its successor configurations: i.e.
  it “leads to acceptance” if at least one of its successors does.
- The idea is to allow both $\text{OR}$ and $\text{AND}$ configurations in a tree of configurations generated by a NTM $N$ computing on input $x$. 

© 2007 TKK, Laboratory for Theoretical Computer Science
### Alternating Turing machines

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

Given the tree of configurations of $N$ on input $x$, the eventually accepting configurations of $N$ are defined recursively:

1. Any leaf configuration with state “yes” is eventually accepting.
2. A configuration with state in $K_{\text{AND}}$ is eventually accepting iff all its successors are.
3. A configuration with state in $K_{\text{OR}}$ is eventually accepting iff at least one of its successors is.

$N$ accepts $x$ iff its initial configuration is eventually accepting.

\[
\begin{align*}
\text{Learning Objectives} & \\
\text{Parallel models of computation: uniform circuits and PRAMs} & \\
\text{The classes } & \mathsf{NC}, \mathsf{NC}_1, \mathsf{NC}_2 \text{ and their relationship to } \mathsf{L}, \mathsf{NL}, \mathsf{P}. & \\
\text{Alternating Turing machines} & \\
\end{align*}
\]

### Alternation-based complexity classes

**Definition.** An alternating Turing machine $N$ decides a language $L$ iff $N$ accepts all strings $x \in L$ and rejects all strings $x \notin L$.

- It is straightforward to define $\mathsf{ATIME}(f(n))$ and $\mathsf{ASPACE}(f(n))$; and using them, $\mathsf{AP} = \mathsf{ATIME}(n^k)$ and $\mathsf{AL} = \mathsf{ASPACE}(\log n)$.
- Roughly speaking, alternating space classes correspond to deterministic time but one exponential higher.

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

**Corollary.** $\mathsf{AL} = \mathsf{P}$.

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

\[
\begin{align*}
\text{Learning Objectives} & \\
\text{Parallel models of computation: uniform circuits and PRAMs} & \\
\text{The classes } & \mathsf{NC}, \mathsf{NC}_1, \mathsf{NC}_2 \text{ and their relationship to } \mathsf{L}, \mathsf{NL}, \mathsf{P}. & \\
\text{Alternating Turing machines} & \\
\end{align*}
\]