PARALLEL COMPUTATION AND LOG SPACE

- Parallel algorithms
- Parallel models of computation
- The class **NC**
- The $\mathbf{L} \stackrel{?}{=} \mathbf{NL}$ problem
- Alternation

(C. Papadimitriou: *Computational complexity*, Chapters 15 and 16)

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 *polylogarithmic*, i.e. the length of parallel computation is $O(\log^k n)$ for some k.
- However, the executions of parallel algorithms should not require inordinately large (superpolynomial) numbers of processors.
- Let us study the effect of parallelism in two concrete cases: *matrix multiplication* and *graph reachability*.

Matrix Multiplication

- The goal is to compute the product of two $n \times n$ matrices A and B.
- The product $C = A \cdot B$ is defined by
$$C_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj}$$
for indices i and j ranging from 1 to n.
- There is a sequential algorithm with $O(n^3)$ arithmetic operations.
- The same can be achieved in $\log n$ parallel steps by n^3 processors.
- However, the number of processors required by the algorithm can be brought down to $\frac{n^3}{\log n}$ using *Brent's principle*.

Graph Reachability

- It is suspected that depth-first search is inherently sequential so that it cannot be parallelized in polylogarithmic time.
- A completely different approach is based on the adjacency matrix A of a graph $G = (V, E)$ with self-loops ($A_{ii} = 1$ for all i).
- The Boolean product of A with itself $A^2 = A \cdot A$ is defined by $A^2_{ij} = \bigvee_{k=1}^{n} (A_{ik} \wedge A_{kj})$ for all $1 \leq i, j \leq n = |V|$.
- A parallel algorithm is obtained by computing the *transitive closure* A^* of A by the sequence A, A^2, A^4, ..., $A^{2^{\lceil \log n \rceil}}$.
- The computation involves $O(\log^2 n)$ parallel steps with $O(n^3 \log n)$ total work so that the number of processors required is $O(\frac{n^3}{\log n})$.

Other Problems Summarized

1. **Arithmetic Operations**

   Using the *prefix sum technique*, the sum of two n-bit binary integers can be computed in $\mathrm{O}(\log n)$ parallel time and $\mathrm{O}(n)$ work. For products of n-bit integers, the work becomes $\mathrm{O}(n^2 \log n)$.

2. **Maximum Flow**

   A prime example of a polynomial-time solvable problem that seems to be inherently sequential.

3. **The Traveling Salesperson Problem**

   Parallelism is not sufficient alone to conquer **NP**-completeness.

4. **Determinants and Inverses**

   There is a polylog parallel time & polynomial work algorithm.

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 **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

- 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.
- In the sequel, *uniform* families of Boolean circuits will be 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 $\mathbf{PT}/\mathbf{WK}(f(n), g(n))$ consists of languages $L \subseteq \{0,1\}^*$ for which there is a uniform family of circuits C deciding L with $\mathrm{O}(f(n))$ parallel time and $\mathrm{O}(g(n))$ parallel work.

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

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 Π_i executes its own program, has its own program counter and accumulator, i.e. the ith 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 CWRW 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 $q(n,m)$ of the number m of input integers in $I = (i_1, \ldots, i_m)$ and their total length $n = \mathrm{l}(I)$.
- A family of PRAMs $\mathcal{P} = \{P_{m,n} \mid m,n \geq 0\}$ is *uniform* iff there is a TM which given $1^m 0 1^n$ 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 $\mathcal{P} = \{P_{m,n} \mid m,n \geq 0\}$ be a uniform family of PRAMs.

Definition. The family $\mathcal{P}$ *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 $\mathbf{PT}/\mathbf{WK}(f(n), g(n))$, then there is a uniform PRAM that computes the corresponding function F_L in parallel time $\mathrm{O}(f(n))$ using $\mathrm{O}(\frac{g(n)}{f(n)})$ processors.

- Circuits can simulate PRAMs:

  Let F be computed by a uniform PRAM in parallel time $f(n)$ using $g(n)$ processors ($f(n), g(n)$ comp. from 1^n in log space).

  Then there is a uniform family of circuits of depth

  $$\mathrm{O}(f(n)(f(n) + \log n)) \text{ and size } \mathrm{O}(g(n)f(n)(n^k f(n) + g(n)))$$

  which computes the binary representation of F.

  (Here n^k is the time bound of the log space TM computing the nth 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 e.g. $\log^3 n$ and $\sqrt{n}$ is seen only for big n: $\log^3 10^8 > 18000$ and $\sqrt{10^8} = 10000$.
- One possiblity is to consider subclasses of **NC** for $j = 1, 2, \ldots$: $\mathbf{NC}_j = \mathbf{PT/WK}(\log^j n, n^k)$ — a potential *hierarchy* of classes.
- The class $\mathbf{NC}_2$ provides an alternative (more conservative) notion of "efficient parallel computation".

Relang NC with P

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

☞ **Conjecture:** $\mathbf{NC} \neq \mathbf{P}$.

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

Theorem. ODD MAX FLOW is **P**-complete.

(So are MAX FLOW(D), HORNSAT, adn CIRCUIT VALUE.)

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

We may relate logarithmic space classes and parallel complexity classes:

Theorem. $\mathbf{NC}_1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{NC}_2$.

Proof.

1. The last inclusion follows by reachablity method, since REACHABILITY belongs to $\mathbf{NC}_2$.
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 $\mathbf{NC}_1 \subseteq \mathbf{L}$ — 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.

☞ The composition operates in logarithmic space. □

Parallel computation thesis

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

$$\begin{aligned}\mathbf{PT/WK}(f(n), k^{f(n)}) &\subseteq \mathbf{SPACE}(f(n)) \\ &\subseteq \mathbf{NSPACE}(f(n)) \\ &\subseteq \mathbf{PT/WK}(f(n)^2, k^{f(n)^2}).\end{aligned}$$

Theorem. REACHABILITY is $\mathbf{NL}$-complete.

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

Actually, all languages in $\mathbf{L}$ are $\mathbf{L}$-complete!

Theorem. $\mathbf{NL}$ is precisely the class of all graph-theoretic properties expressible in Krom existential second-order logic with successor.

5. Alternation

- Alternation is an important generalization of nondeterminism.
- In a nondeterministic computation each configuration is an implicit *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 *OR* and *AND* configurations in a tree of configurations generated by a NTM N computing on input x.

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 $K = K_{\mathrm{AND}} \cup K_{\mathrm{OR}}$.

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_{AND} is eventually accepting iff all its successors are.
3. A configuration with state in K_{OR} is eventually accepting iff at least one of its successors is.

☞ N *accepts* x iff its initial configuration is eventually accepting.

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

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

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

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