18. Parallel Complexity

Goal:
- Take a complexity-theoretic view to parallel computation.
- Model of computation: Uniform families of Boolean circuits (good for algebraic problems).
- Alternative (but not in this lecture): Parallel random access machines (PRAMs) (good for graph algorithms and combinatorial problems).

Class:
- NC = Nick's class, named after Nicholas Pippenger (by Steve Cook).
  = circuits of poly-logarithmic depth.
- L is between L and P.
- It is to parallel computation what P is to sequential computation:
  A robust and natural approximation to the notion of efficiently parallelizable computation.

Examples:
- Many linear algebra problems can be done in NC.
- There is no efficient parallel algorithm for circuit value (otherwise NL = P, which is believed unlikely).
- Integer greatest common divisor is not known to be in NC, polynomial gcd is in NC.

Status:
- P = NP asks whether important combinatorial problems have efficient sequential solutions.
- NC = P asks whether problems that are known to have efficient sequential solutions have efficient parallel solutions (refer to P-complete later).

18.1 NC

Goal: Study the class NC
↳ Relate it to logarithmic space
↳ Relate it to efficient parallel algorithms.

Technically: We are interested in the simultaneous size and depth complexity of a circuit family:
How many processors do we need in order to achieve a particular parallel time complexity, or vice versa.

Definition:
The simultaneous size-depth complexity of a circuit family is the pair of functions (f, g),
- where $f(n)$ measures the size and
- $g(n)$ measures the depth of the circuits.

Observation:
- Many problems have size-depth complexity $(O(n^k), O(\log^k n))$.
- We consider them highly parallelizable with a moderate degree of processors.

Goal: Understand what can be computed with such circuit families.

Definition:
- For $i \geq 1$, let NC^i be the class of languages that can be decided by
  - a logspace-uniform family of circuits
  - of size-depth complexity $(O(n^c), O((\log n)^i))$.

- Let $NC := \bigcup_{i \geq 1} NC^i$.
- Functions that are computable by such circuit families are called $\underline{NC^i\text{-computable}}$ or $\underline{NC\text{-computable}}$.

Note: Some books refer to the above definition as uniform NC and also have non-uniform variants.

Result 1: Problems solvable with logarithmic depth are also solvable with logarithmic space.

Theorem: $NC^1 \subseteq L$.

Proof:
Let A be a language in NC^1.
We construct a logspace algorithm to decide A.

Idea:
- On input w of length n we first construct a description of the n-th circuit in the uniform family of circuits for A.
- The key idea is then to evaluate the circuit using a depth-first search from the output gate
- We have to store the path to the currently explored gate and partial results that have been obtained along the path

  (one can actually skip the latter:
  - order the predecessors as left, right,
  - in an $\vee$-gate, only enter right if left yields 0
  - in an $\wedge$-gate, only enter right if left yields 1.)
- The circuit has logarithmic depth, hence the simulation only needs logarithmic space.

There is a catch:

- One cannot compute in logspace
  a description of the n-th circuit:
  It will have linear size and hence
  does not fit into memory.
- Instead, we use the logspace function that exists
  by the uniformity requirement to compute
  the left or right predecessor of a given gate.
- So we can still do the depth-first search
  without an explicit representation of the circuit. □

Note:

- The result essentially shows that
  evaluation of a circuit of depth d (with bounded fan-in)
  can be done in space $O(d)$.
- Now we have:
  ↳ The computation of the circuit works in $O(\log n)$ space.
  ↳ The evaluation of the circuit works in $O(\log n)$ space.
  ↳ The composition of two logspace functions
  is again a logspace-computable function.

Result 2: Problems solvable with logarithmic space, even non-deterministically,
can be solved in $O((\log n)^2)$ depth.

Theorem: $NL \subseteq NC^2$.

Idea: • Compute the reflexive transitive closure
of the configuration graph of an NL-machine.

- Output the position in the matrix
  that corresponds to the presence of a path
  from the start to the (single) accept configuration.

Proof: - Let $L \in NL$ be a language that is decided by the TM M
  and assume L has been encoded into $\{0,1\}^*$.
  We construct a log-space-uniform family $(C_n)_{n \in \mathbb{N}}$ for M.

- To get C_n, we construct a graph G
  that is similar to the configuration graph
  for M on input w of length n.
  But we <u>do not know</u> w when we construct C_n,
  only its <u>length</u>.
  The inputs to C_n will be variables $w_1, \dots, w_n$ with $w = w_1 \dots w_n$.
- If configuration of M on input w describes
  ↳ the state
  ↳ the content of the work tape
  ↳ the positions of the input and the work tape heads.
  Notably, the configuration does <u>not</u> depend on w itself,
  only on w's length.

  These <u>polynomially many</u> configurations form the graph G

- The edges of G are labelled with the input variables w_i.
  Let c_1 and c_2 be two configurations of G,
  and assume c_1 indicates input head position i.

↳ There is an edge $c_1 \xrightarrow{w_i} c_2$
if c_1 can yield c_2 according to M's transition relation,
provided the input head is reading 1.

↳ There is an edge $c_1 \xrightarrow{\overline{w_i}} c_2$
if c_1 can yield c_2 when the input head reads 0.

↳ If c_1 can yield c_2 whatever the input head is reading,
we put the unlabelled edge $c_1 \to c_2$.

- Assume we set the edges according to a string w of length n.
This instantiates G to $G(w)$.
Then
there is a path from the start to the accept configuration in $G(w)$
iff M accepts w.

- Setting the edges and hence instantiating G to $G(w)$ is the first step in our circuit.
It results in
the Boolean incidence matrix of the instantiated graph $G(w)$.
We then compute the reflexive transitive closure of $G(w)$.
Our output is the position indicating the presence of a path.
The circuit has polynomial size and $O((\log n)^2)$ depth.

- The construction can be conducted in logarithmic space.
The point is to construct G:
We already did this when showing PATH to be NL-hard
(enumerate nodes and edges).
We know how to do the reflexive transitive closure. □

Note: Every problem in NC is solvable in polynomial time.

Theorem: $NC \subseteq P$.

Proof: Consider a problem $\Pi \in NC$ and an input w.
We run the logspace algorithm to compute $C_{|w|}$.
This takes polynomial time.
Now we compute CVP on $(C_{|w|}, w)$,
again in polynomial time. □

The class NC is well-behaved wrt. reductions.
The following theorem indicates that probably $P \neq NC$,
because circuits do not seem to be compressible (CVP is P-complete).

Theorem:
Let Π be P-complete wrt. $\leq_m^{\log}$.
We have $\Pi \in NC$ iff $P = NC$.

The idea is as follows. Let B be a problem in P.
We already know that NC-circuits can compute
the logspace reduction from B to Π.
To be precise, we use an NC-circuit for every bit
of the result $f(w)$ of the reduction
(w the input to B).

We then wire the $|f(w)|$-bits to the circuit $C_{|f(w)|}$ for Π.
The circuits for each bit are logspace constructible.
The enumeration of bits can also be done in $\log |f(w)| \in O(\log |w|)$,
since f is a polynomial.
Similarly, the circuit $C_{|f(w)|}$ is logspace constructible.

Lemma: If $B \leq_m^{\log} \Pi$ and $\Pi \in NC$, then $B \in NC$.

- NC characterizes the languages with efficient parallel algorithms
  ↳ Polylogarithmic parallel compute time.
- Here we are rough in the model of parallel computation (PRAM-like).

Theorem:
A language has an efficient parallel algorithm if and only if it is in NC.

Argument (rather a sketch of the idea than a real proof):

"⇐" Suppose A is decidable by the circuit family $(C_n)_{n\in\mathbb{N}}$ of size $N = O(n^c)$ and depth $D = O((\log n)^c)$.
We take a general-purpose computer with N nodes and configure it to decide A as follows.
  ↳ Compute a description of C_n and allocate each gate to a processor.
  ↳ Each processor, after having computed the output of its node, sends it to all processors that need it.
  ↳ Assuming the interconnect network sends messages in $O(\log N)$ time, the total (parallel) running time is $O((\log n)^{c+1})$.

"⇒" Let the computer have $N = \text{poly}(n)$ nodes and let the computation take $D = (\log n)^c$ time.
We construct a circuit where the gates are arranged in D layers.

On each layer, we have N gates.

Idea:
- The i-th node in the j-th layer performs the computation of node i at time step j.
- The interconnect is mimicked by the wires. □

18.2 AC

Goal: Understand the role of the fan-in for the power of the computational model.

Definition:
- For $i \in \mathbb{N}$, let AC^i be defined like NC^i but admit gates of unbounded fan-in.
- Let $AC := \bigcup_{i \in \mathbb{N}} AC^i$.

Remark:
- Note that NC^0 does not make sense because we can only have a constant number of input bits.
- AC^0 does not have this problem.

Theorem: $NC^i \subseteq AC^i \subseteq NC^{i+1}$, and hence $NC = AC$.

Proof: The first inclusion is by definition, the second is homework.

Status:
- NC^1 vs. PH is not known.
- $AC^0 \subsetneq NC^1$ and $NC^0 \subsetneq AC^0$ are known.
- $NC^i \subsetneq NC^{i+1}$ is not known and a major issue.
- $NC^i \subsetneq NC^{i+1}$ is known for <u>monotone</u> circuits.