Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 11: Boolean Circuits and Parallel Computation

Aalto University
School of Science
Department of Computer Science

Spring 2017

Agenda

- Boolean circuits and circuit complexity
- Parallel algorithms and complexity measures
- Parallel random access machines (PRAMs)
- The class $\mathbf{NC}$
- Sequential space vs. parallel time

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

1. Boolean Circuits and Circuit Complexity

- A Boolean circuit with n inputs *accepts* a string $x = x_1 \ldots x_n$ of length n in $\{0,1\}^*$ if the output of the circuit is **true** given x as its input (i.e., input i is **true** if x_i is 1 and **false** otherwise).
- The *size* of a circuit is the number of gates in it.
- To relate circuits to strings of arbitrary length, families of circuits are introduced.
- A *family of circuits* is an infinite sequence $C = (C_0, C_1, \ldots)$ of Boolean circuits where C_n has n input variables.

Languages with polynomial circuits

- A language $L \subseteq \{0,1\}^*$ has *polynomial circuits* if there is a family $C = (C_0, C_1, \ldots)$ such that
  1. the size of C_n is at most $p(n)$, for some fixed polynomial p; and
  2. for all $x \in \{0,1\}^*$, $x \in L$ iff the output of $C_{|x|}$ is **true** when x is given as input to $C_{|x|}$.

Proposition

All languages in $\mathbf{P}$ *have polynomial circuits.*

Proof

By the $\mathbf{P}$-completeness proof of CIRCUIT VALUE:
Given a Turing machine M, its input x, and running time $p(|x|)$, a variable-free polynomial size circuit $C(M,x,p)$ is constructed such that the output of $C(M,x,p)$ is **true** iff M accepts x.
It is easy to modify input gates of $C(M,x,p)$ to variable gates reflecting the symbols in x.

Proposition

There are undecidable languages that have polynomial circuits.

Proof

- Let $L \subseteq \{0,1\}^*$ be an undecidable language and let $U \subseteq \{1\}^*$ be $U = \{1^n : \text{ the binary expansion of } n \text{ is } 1x, \text{ where } x \in L\}$.
- U is undecidable.
- U has polynomial circuits $(C_0, C_1, \ldots)$ where C_n consists of n input gates and either
  - $n-1$ AND-gates (computing the AND of all input bits) if $1^n \in U$; or
  - a constant **false** output gate if $1^n \notin U$.

Uniformly polynomial circuits

- A family $C = (C_0, C_1, \ldots)$ of circuits is said to be *uniform* if there is a $\log n$-space bounded Turing machine which on input 1^n outputs C_n.
- A language $L \subseteq \{0,1\}^*$ has *uniformly polynomial circuits* if there is a uniform family of polynomial circuits $C = (C_0, C_1, \ldots)$ that decides L.

Theorem

A language L has uniformly polynomial circuits iff $L \in \mathbf{P}$.

Proof

$(\Rightarrow)$ $x \in L$ can be decided in polynomial time by constructing $C_{|x|}$ in $\log |x|$ space (and hence in polynomial time) and evaluating it for input x (in polynomial time).

$(\Leftarrow)$ By the $\mathbf{P}$-completeness proof of CIRCUIT VALUE:

Given a Turing machine M, its input x, and running time $p(|x|)$, the circuit $C(M,x,p)$ can be constructed in $\log |x|$ space.

Polynomial circuits and $\mathbf{P}$ vs $\mathbf{NP}$

- $\mathbf{P} \neq \mathbf{NP}$ is equivalent to
  - **Conjecture A**: $\mathbf{NP}$-complete problems have no uniformly polynomial circuits.
- Related(?) **Conjecture B:** $\mathbf{NP}$-complete problems have no polynomial circuits, uniform or not.
- Shannon (1949): Most Boolean functions do not have polynomial (not even subexponential) circuits.
- An approach to establish $\mathbf{P} \neq \mathbf{NP}$: Prove Conjecture B for some $\mathbf{NP}$-complete problem.

2. Parallel Algorithms and Complexity Measures

- The goal of parallel algorithms is to be dramatically better than sequential ones, preferably *polylogarithmic*, i.e. such that the length of the 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 now consider parallel computations using processors more powerful than single Boolean gates.

Case: Matrix Multiplication

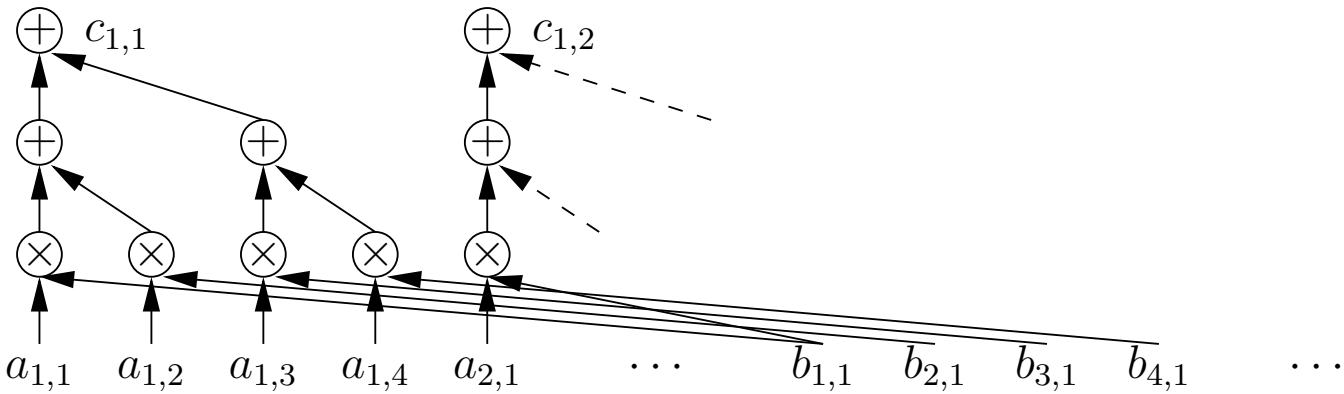
- The task is to compute the product of two $n \times n$ matrices A and B.
- The product $C = AB$ is defined by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

  for indices i and j ranging from 1 to n.
- There is a sequential algorithm with $O(n^3)$ arithmetic operations.
- Example: multiplying two 4×4 matrices

$$\begin{aligned} c_{1,1} &= a_{1,1}b_{1,1} + a_{1,2}b_{2,1} + a_{1,3}b_{3,1} + a_{1,4}b_{4,1} \\ c_{1,2} &= a_{1,1}b_{1,2} + a_{1,2}b_{2,2} + a_{1,3}b_{3,2} + a_{1,4}b_{4,2} \\ &\vdots \end{aligned}$$

- The same can be achieved in $1+\log n$ parallel steps with n^3 processors and $n^3+n^2\times(n-1)=2n^3-n^2$ arithmetic operations.
- Example: multiplying two 4×4 matrices



- The number of processors used by the algorithm can be brought down to any P using *Brent's principle*: The computation is done in $1+\log n$ "shifts" where each shift $t=0,\ldots,\log n$ is completed in $\lceil \frac{n^3/2^t}{P} \rceil$ parallel steps, for a total of $T \leq \frac{2n^3}{P}+\log n+1$ steps.
- With $P \sim \frac{n^3}{\log n}$, one still achieves $T=O(\log n)$ parallel time.

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 $\mathbf{NP}$-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.

Parallel time and work

- The primary complexity measures for parallel computation are *parallel time* and *parallel work*.
- Let $C = (C_0, C_1, \ldots)$ be a uniform family of Boolean circuits.
  - C has *parallel time* $t(n)$ if for all n the *depth* of C_n is at most $t(n)$.
  - C has *parallel work* $s(n)$ if for all n the *size* of C_n is at most $s(n)$.
- The class

$$\mathbf{PT/WK}(t(n), s(n))$$

  consists of languages $L \subseteq \{0,1\}^*$ for which there is a uniform family of circuits C deciding L with $O(t(n))$ parallel time and $O(s(n))$ parallel work.
- *Note:* No speedup theorem for circuits $\Rightarrow$ big-Oh notation in the definition.

Case: REACHABILITY

- By using the circuit representation of Warshall's algorithm for reflexive and transitive closure of graphs (recall Lecure 7 "Reductions and Completeness"), we have

$$\text{REACHABILITY} \in \mathbf{PT}/\mathbf{WK}(n, n^3)$$

- By an adjacency matrix based technique, one achieves

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

  - *Input:* Boolean adjacency matrix A of a graph $G = (V, E)$ with self-loops ($A_{ii} = \top$ for all i).
  - Boolean product of A with itself $A^2 = AA$ is $A^2_{ij} = \bigvee_{k=1}^{n} (A_{ik} \wedge A_{kj})$ for all $1 \leq i, j \leq n = |V|$. $[\Rightarrow \mathbf{PT}/\mathbf{WK}(\log n, n^3)]$
  - Parallel algorithm is obtained by computing the *transitive closure* A^* of A by the sequence $A, A^2, A^4, \ldots, A^{2^{\lceil \log n \rceil}}$.
    $[\Rightarrow \mathbf{PT}/\mathbf{WK}(\log^2 n, n^3 \log n)]$

Some Other Problems Summarised

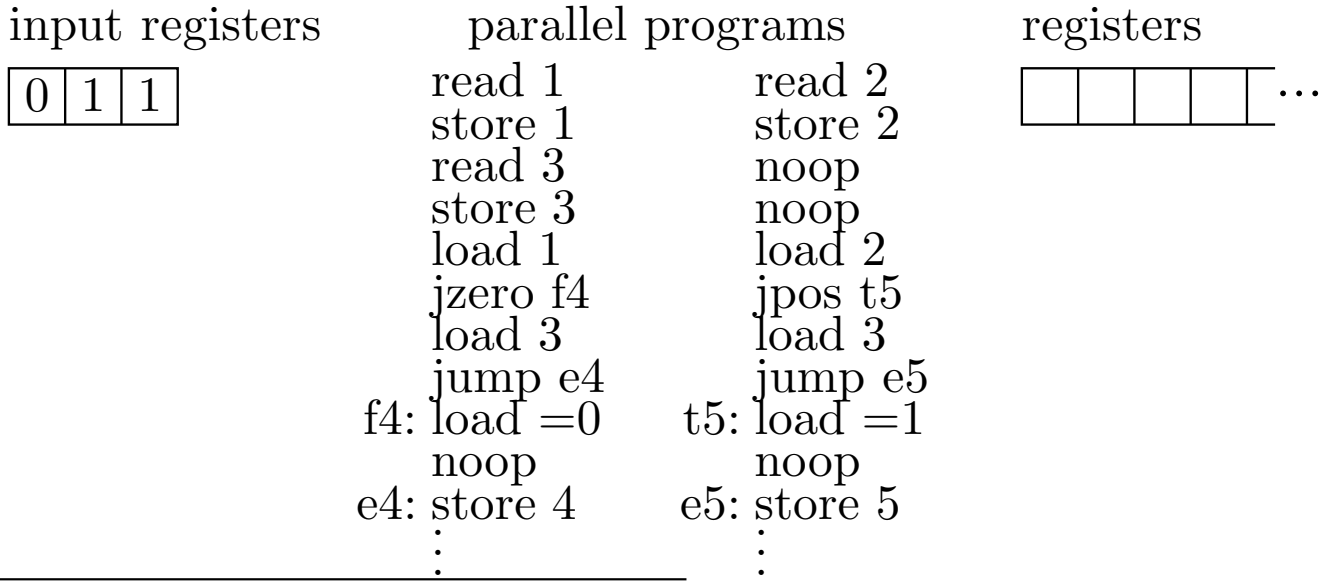
1. **Arithmetic Operations**
   Using the *prefix sum technique*, the sum of two n-bit binary integers can be computed in $O(\log n)$ parallel time and $O(n)$ work.
   For products of n-bit integers, the work becomes $O(n^2 \log n)$.
2. **Maximum Flow**
   A prime example of a polynomial-time solvable problem that seems to be inherently sequential.
3. **The Travelling Salesperson Problem**
   Parallelism is not sufficient alone to conquer $\mathbf{NP}$-completeness.
4. **Matrix Determinants and Inverses**
   There is a polylog parallel time & polynomial work algorithm.

3. Parallel Random Access Machines (PRAMs)

- A formal model of parallel computation by a large number of independent processors. Each processor can execute its own program and can communicate with other processors instantaneously and synchronously through a large shared memory.
- Complexity closely related to Boolean circuits, hence a useful high-level tool for designing parallel algorithms and understanding parallelism.
- From the practical point of view somewhat unrealistic due to the assumptions of instantaneous communication and concurrent write access to the shared memory.

The PRAM model

- 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. we assume the *PRIORITY CRCW PRAM* model.[1]



[1] Cf. Papadimitriou, Note 15.5.7.

Uniform PRAM families

- 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* if 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 $t(n)$ and $q(n)$ 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* t *with* q *processors* if for each $m,n \geq 0$, for $P_{m,n}$ it holds that

(i) it has $q(m,n) \leq q(n)$ 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 $t(n)$ steps and the $k \leq q(m,n)$ first registers contain the output $F(I)$.

PRAMs and circuits

- PRAMs can simulate circuits:
  If $L \subseteq \{0,1\}^*$ is in $\mathbf{PT/WK}(t(n), s(n))$, then there is a uniform PRAM that computes the corresponding function F_L in parallel time $O(t(n))$ using $O(\frac{s(n)}{t(n)})$ processors. [Brent's principle!]
- Circuits can simulate PRAMs:
  Let F be computed by a uniform PRAM in parallel time $t(n)$ using $q(n)$ processors ($t(n), q(n)$ can be computed from 1^n in logarithmic space).
  Then there is a uniform family of circuits of
  depth $O(t(n)(t(n) + \log n))$ and size $O(q(n)t(n)(n^k t(n) + q(n)))$
  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.)

4. The Class NC

- What would be the class of problems that is satisfactorily solved by parallel computers? A candidate definition ("Nick's class"):

$$\mathbf{NC} = \mathbf{PT}/\mathbf{WK}(\log^k n, n^k).$$

- $\mathbf{NC}$ 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 $\sqrt{n}$: $\log^3 10^8 > 18000$ and $\sqrt{10^8} = 10000$.
- One possibility is to consider subclasses of $\mathbf{NC}$ for $j = 1, 2, \ldots$: $\mathbf{NC}_j = \mathbf{PT}/\mathbf{WK}(\log^j n, n^k)$ — a potential *hierarchy* of classes.
- The class $\mathbf{NC}_2$ provides an alternative (more conservative) notion of "efficient parallel computation".

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 MAX FLOW(D), HORNSAT, and CIRCUIT VALUE.)

5. Sequential Space and Parallel Time

Logarithmic space classes and parallel complexity classes are closely related:

Theorem

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

Proof.

1. For the last inclusion note that REACHABILITY is $\mathbf{NL}$-complete (by the "reachability method" of Lecture 6), and belongs to $\mathbf{PT}/\mathbf{WK}(\log^2 n, n^3 \log n) \subseteq \mathbf{NC}_2$.
2. The inclusion in the middle is trivial.
3. For the first inclusion, we have to compose three algorithms that operate in logarithmic space.

Proof of $\mathbf{NC}_1 \subseteq \mathbf{L}$

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 (cf. Lecture 7, Section 3). □

The parallel computation thesis

- The previous result can be generalised beyond logarithmic space:

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

- Sequential space and parallel time complexities are polynomially related!

Theorem

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

Theorem

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

Learning Objectives

- Basic concepts of Boolean circuit complexity: languages with polynomial circuits and uniformly polynomial circuits.
- Parallel models of computation: uniform circuits and PRAMs
- The classes $\mathbf{NC}$, $\mathbf{NC}_1$, $\mathbf{NC}_2$ and their relationship to $\mathbf{L}, \mathbf{NL}, \mathbf{P}$.