Complexity Theory

Jan Křetínský

Technical University of Munich

Summer 2019

June 4, 2019

Lecture 22

Models of Parallel Computation

Goal and plan

Goal

- introduce two models of parallel computation
- understand why they are equivalent

Plan

- PRAM: parallel random access machine
- circuits
- some complexity class definitions

Random access machine

RAM: more realistic model of sequential computation, which can be simulated by standard TMs with polynomial overhead.

- computation unit with user-defined program
- read-only input tape, write-only output tape, unbounded number of local memory cells
- memory cells can hold unbounded integers
- instructions include
  - moving data between memory cells
  - comparisons and branches
  - simple arithmetic operations
- all operations take unit time

Parallel random access machine

PRAM: parallel extension of RAM

- unbounded collection of RAM processors without tapes: $P_0, P_1, P_2, \ldots$
- unbounded collection of shared memory cells: $M[0], M[1], M[2], \ldots$
- each P_i has its own local memory (registers)
- input: n items stored in $M[0], \ldots, M[n-1]$
- output stored on some designated part of memory
- instructions execute in 3-phase cycles
  - read from shared memory
  - local computation
  - write to shared memory
- processors execute cycles synchronously
- P_0 starts and halts execution

Read/write conflicts

It may happen that several processors want to read from or write to the same memory cell in one cycle.

Read/write conflicts

It may happen that several processors want to read from or write to the same memory cell in one cycle.

Three policies:

- **EREW** : exclusive read/exclusive write
- **CREW** : concurrent read/exclusive write allows for simultaneous reads
- **CRCW** : simultaneous read and write allowed

Practical concerns

- idealized: PRAMs are an abstract, idealized formalism
  - unbounded integers
  - communication between any two processors in constant time due to shared memory (in reality: interconnection networks)
  - too many processors
- CRCW and CREW hard to build technically but easier to design algorithms
- still useful as benchmark
  - if there is no good PRAM algorithm, probably the problem is hard to parallelize

Time and space complexity

- time complexity: number of steps of P_0
- space complexity: number of shared memory cells accessed
- one can show that the weakest PRAM (EREW) can simulate the strongest with logarithmic overhead; cf. search-example
- efficient parallel computation
  - polynomially many processors
  - polylogarithmic time, where $polylog(n) = \bigcup_{k \geq 1} \log^k n$
- problems with efficient parallel algorithms are said to be in **NC**
- **NC** is robust wrt different PRAM models (and circuits)

Example: Search

Example

Given n items on the shared memory tape and $p + 1 < n$ processors. For some $x \in \mathbb{N}$ P_0 wants to know, whether there exists an $0 \leq i < n$ such that $M[i] = x$.

Example: Search

Example

Given n items on the shared memory tape and $p + 1 < n$ processors. For some $x \in \mathbb{N}$ P_0 wants to know, whether there exists an $0 \leq i < n$ such that $M[i] = x$.

Solution (high level):

1. P_0 publishes x
2. for $1 \leq i \leq p$ each P_i searches through $M[\lceil \frac{n}{p} \rceil (i-1)], \ldots, M[\lceil \frac{n}{p} \rceil i - 1]$
3. each P_i announces its search result

Analysis

Step 2 need n/p parallel time independently of PRAM model.

Analysis

Step 2 need n/p parallel time independently of PRAM model.

Step 1

- needs $\mathcal{O}(1)$ time in CRCW and CREW since P_0 can simply write x on the shared tape which everybody can read simultaneously
- needs $\log p$ steps in EREW by binary broadcast tree

Analysis

Step 2 need n/p parallel time independently of PRAM model.

Step 1

- needs $\mathcal{O}(1)$ time in CRCW and CREW since P_0 can simply write x on the shared tape which everybody can read simultaneously
- needs $\log p$ steps in EREW by binary broadcast tree

Step 3

- needs $\mathcal{O}(1)$ time in CRCW only, where all successful processors indicate success in the same memory cell
- otherwise, we need $\log p$ time to perform a parallel reduction

Other problems in NC

Many practical problems are known to be in **NC**, for details, take some class on parallel algorithms.

- sorting
- matrix multiplication
- expression evaluation
- connected components of graphs
- string matching

Signpost

Just seen:

- RAMs and PRAMs
- CRCW, CREW, EREW
- simulations between models have at most logarithmic overhead
- efficient parallel $\sim$ polylogarithmic (stable under different PRAM models)

Next:

- Boolean circuits as parallel model of computation
- equivalence with respect to efficient parallel algorithms of PRAM and circuits

Boolean Circuits

Definition

A Boolean circuit, C, is a directed acyclic graph with labeled nodes.

- the input nodes are labeled with a variable x_i or with a constant 0 or 1
- the gate nodes have fan-in $k > 0$ are labeled with one of the Boolean functions
  - $\wedge$ (fan-in k)
  - $\vee$ (fan-in k)
  - $\neg$ (fan-in 1)
- the output nodes are labeled *output* and have fan-out 0

Boolean Circuits

Definition

A Boolean circuit, C, is a directed acyclic graph with labeled nodes.

- the input nodes are labeled with a variable x_i or with a constant 0 or 1
- the gate nodes have fan-in $k > 0$ are labeled with one of the Boolean functions
  - $\wedge$ (fan-in k)
  - $\vee$ (fan-in k)
  - $\neg$ (fan-in 1)
- the output nodes are labeled *output* and have fan-out 0

Given an assignment $\sigma : \{0, 1\}^m \to \{0, 1\}$ to the m variables, $C(\sigma)$ denotes the value of the o output nodes. We denote by $size(C)$ the number of gates and by $depth(C)$ the maximum distance from an input to an output.

Boolean Circuits

Definition

A Boolean circuit, C, is a directed acyclic graph with labeled nodes.

- the input nodes are labeled with a variable x_i or with a constant 0 or 1
- the gate nodes have fan-in $k > 0$ are labeled with one of the Boolean functions
  - $\wedge$ (fan-in k)
  - $\vee$ (fan-in k)
  - $\neg$ (fan-in 1)
- the output nodes are labeled *output* and have fan-out 0

Given an assignment $\sigma : \{0,1\}^m \to \{0,1\}$ to the m variables, $C(\sigma)$ denotes the value of the o output nodes. We denote by $size(C)$ the number of gates and by $depth(C)$ the maximum distance from an input to an output.

We distinguish circuits with and without a-priori bounds on fan-in.

Boolean Circuits

Definition

A Boolean circuit, C, is a directed acyclic graph with labeled nodes.

- the input nodes are labeled with a variable x_i or with a constant 0 or 1
- the gate nodes have fan-in $k > 0$ are labeled with one of the Boolean functions
  - $\wedge$ (fan-in k)
  - $\vee$ (fan-in k)
  - $\neg$ (fan-in 1)
- the output nodes are labeled *output* and have fan-out 0

Given an assignment $\sigma : \{0,1\}^m \rightarrow \{0,1\}$ to the m variables, $C(\sigma)$ denotes the value of the o output nodes. We denote by $size(C)$ the number of gates and by $depth(C)$ the maximum distance from an input to an output.

We distinguish circuits with and without a-priori bounds on fan-in.

Wlog we assume that all negations appear in the input layer only.

Example: addition

Assume we want to add two n-bit integers, that is, we want circuits to compute $+ : \{0,1\}^{2n} \to \{0,1\}^{n+1}$

Example: addition

Assume we want to add two n-bit integers, that is, we want circuits to compute $+ : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{n+1}$

Ripple carry adder

- n sequential full adder
- depth: $\mathcal{O}(n)$
- size: $\mathcal{O}(n)$

Example: addition

Assume we want to add two n-bit integers, that is, we want circuits to compute $+ : \{0, 1\}^{2n} \to \{0, 1\}^{n+1}$

Ripple carry adder

- n sequential full adder
- depth: $\mathcal{O}(n)$
- size: $\mathcal{O}(n)$

Conditional sum adder

- depth: $\mathcal{O}(\log n)$
- size: $\mathcal{O}(n \log n)$

Example: addition

Assume we want to add two n-bit integers, that is, we want circuits to compute $+ : \{0,1\}^{2n} \to \{0,1\}^{n+1}$

Ripple carry adder

- n sequential full adder
- depth: $\mathcal{O}(n)$
- size: $\mathcal{O}(n)$

Conditional sum adder

- depth: $\mathcal{O}(\log n)$
- size: $\mathcal{O}(n \log n)$

Carry lookahead adder

- depth: $\mathcal{O}(\log n)$
- size: $\mathcal{O}(n)$

Deciding languages with circuits

Definition

A language $L \subseteq \{0,1\}^*$ is said to be decided by a family of circuits $\{C_n\}$, where C_i takes i input variables, iff for all i holds: $C_i(x) = 1$ iff $x \in L$.

Deciding languages with circuits

Definition

A language $L \subseteq \{0,1\}^*$ is said to be decided by a family of circuits $\{C_n\}$, where C_i takes i input variables, iff for all i holds: $C_i(x) = 1$ iff $x \in L$.

Definition

Let $d, s : \mathbb{N} \to \mathbb{N}$ be functions. We say that a family $\{C_n\}$ has depth d and size s if for all n

- $depth(C_n) \leq d(n)$
- $size(C_n) \leq s(n)$

Examples

Example (Parity)

Parity $= \{x \in \{0,1\}^* \mid x$ has an odd number of 1s$\}$

Examples

Example (Parity)

Parity $= \{x \in \{0,1\}^* \mid x$ has an odd number of 1s$\}$

- circuits are binary trees of *xor* gates
- each xor-gate has depth 3

$\Rightarrow$ logarithmic depth

Examples

Example (Parity)

Parity $= \{x \in \{0,1\}^* \mid x$ has an odd number of 1s$\}$

- circuits are binary trees of *xor* gates
- each xor-gate has depth 3

$\Rightarrow$ logarithmic depth

Example (UHalt)

UHalt $= \{1^n \mid$
n's binary expansion encodes a pair $\langle M, x\rangle$ such that M halts on $x\}$

Examples

Example (Parity)

$\text{Parity} = \{x \in \{0,1\}^* \mid x$ has an odd number of 1s$\}$

- circuits are binary trees of *xor* gates
- each xor-gate has depth 3

$\Rightarrow$ logarithmic depth

Example (UHalt)

$\text{UHalt} = \{1^n \mid$
n's binary expansion encodes a pair $\langle M, x\rangle$ such that M halts on $x\}$

- circuit family of linear size decides UHalt even though it is undecidable
- for each n with $1^n \in \text{UHalt}$ is a tree of and-gates
- otherwise, constant 0 circuit

On Uniformity

Problem on previous slide: the description of the circuit family is not computable.

On Uniformity

Problem on previous slide: the description of the circuit family is not computable.

Solution: uniformity

On Uniformity

Problem on previous slide: the description of the circuit family is not computable.

Solution: uniformity

Definition (logspace uniform)

A family of polynomially-sized circuits, $\{C_n\}$ is logspace-uniform if there exists a logspace TM M such that for every n, $M(1^n) = desc(C_n)$, where $desc(C_n)$ is the description of C_n.

On Uniformity

Problem on previous slide: the description of the circuit family is not computable.

Solution: uniformity

Definition (logspace uniform)

A family of polynomially-sized circuits, $\{C_n\}$ is logspace-uniform if there exists a logspace TM M such that for every n, $M(1^n) = desc(C_n)$, where $desc(C_n)$ is the description of C_n.

Remarks

- a description could be a list of gates along with type and predecessors
- the circuit family for Parity is logspace-uniform

Signpost

Just seen:

- circuit definition
- families of circuits decide languages
- there exist families of polynomial size deciding undecidable languages

$\Rightarrow$ require logspace-uniformity

Next:

- circuits vs PRAMs

Circuits vs PRAMs

For efficient parallel computations only:

parallel time on PRAM	$\sim$	circuit depth
number of processors	$\sim$	circuit size

circuits $\rightarrow$ PRAM

- suppose L decided by family $\{C_n\}$ of polynomial size N and depth $\mathcal{O}(\log^d n)$
- a PRAM with N processors decides L:
- compute a description of C_n
- each circuit node $\rightarrow$ one processor
- each processor computes its output and sends it to all other processors that need it (might require logarithmic overhead for non-CR models)
- parallel time $\sim$ circuit depth
- circuit size $\sim$ number of processors

Circuits vs PRAMs

For efficient parallel computations only:

parallel time on PRAM	$\sim$	circuit depth
number of processors	$\sim$	circuit size

PRAM $\rightarrow$ circuits

- circuit with $N \cdot D$ nodes in D layers
- the i-th node in the t-th layer performs computation of processor i at time t

NC and AC

Obviously, variations of PRAMs and circuits are robust wrt. polynomial size/number of processors and polylogarithmic depth/parallel run time motivating the following definition.

Definition (NC and AC)

Let $k \geq 0$. $L \in \mathbf{AC^k}$ iff L is decided by a logspace-uniform family of circuits with polynomial size and depth $\mathcal{O}(\log^k n)$. If the family of circuits is of bounded fan-in, then $L \in \mathbf{NC^k}$.

- $\mathbf{NC} = \bigcup_{k \geq 0} \mathbf{NC^k}$
- $\mathbf{AC} = \bigcup_{k \geq 0} \mathbf{AC^k}$

NC and AC

Obviously, variations of PRAMs and circuits are robust wrt. polynomial size/number of processors and polylogarithmic depth/parallel run time motivating the following definition.

Definition (NC and AC)

Let $k \geq 0$. $L \in \mathbf{AC^k}$ iff L is decided by a logspace-uniform family of circuits with polynomial size and depth $\mathcal{O}(\log^k n)$. If the family of circuits is of bounded fan-in, then $L \in \mathbf{NC^k}$.

- $\mathbf{NC} = \bigcup_{k \geq 0} \mathbf{NC^k}$
- $\mathbf{AC} = \bigcup_{k \geq 0} \mathbf{AC^k}$

- **NC** is the class of problems with efficient parallel solutions
- **AC** circuits cannot be build easily in hardware
- it is an open problem whether $\mathbf{P} = \mathbf{NC}$, that is, whether all problems in **P** are efficiently parallelizable (conjecture: no)
- Parity $\in \mathbf{NC^1}$ (but not in $\mathbf{AC^0}$)

Summary

- three variations of a PRAM
- uniform and non-uniform circuit families can decide languages
- efficiently parallelizable: **NC**
- circuits and PRAM are equivalent wrt **NC** problems

Up next: small depth circuits (**AC** and **NC**)

- their relation to well-known (space) complexity classes
- some lower bounds