18.405J/6.841J Advanced Complexity Theory February 13, 1997

Lecture 4

Lecturer: Daniel A. Spielman *Scribe: Victor Boyko*

1 Parallel computation

Today we will consider a simple model for parallel computation. The model is not of much use in practice, since it is extremely fine-grained. It is, however, convenient to consider theoretically.

We will use boolean circuits, with each gate executing in parallel. Each gate will have 2 inputs. The underlying directed graph must be acyclic. We will usually be concerned with two characteristics of a circuit:

Size total number of gates.

Depth the length of the longest path from an input to an output.

Intuitively, the size of the circuit corresponds to the amount of work performed, while the depth corresponds to running time. Note that we will not be concerned whether the circuit can be efficiently embedded in 3D space.

Definition 1. [1] *[**NC**, **FNC**] A language L is in $\mathbf{NC}^k$ if there exists a family of circuits $\mathcal{C} = \{c_n\}_{n=0}^{\infty}$, polynomial $p(n)$, and a function $d(n) = O(\log^k n)$, such that for each n c_n has*

- *exactly n inputs*
- *exactly one output*
- *size at most $p(n)$*
- *depth at most $d(n)$*

and accepts $L \cap \{0,1\}^n$.

$\mathbf{NC} = \bigcup_k \mathbf{NC}^k$

***FNC** is the class of functions computable by such families circuits (i.e., the difference is that circuits will have multiple outputs instead of just one).*

NC is is an abbreviation for "Nick's class," named after Nick Pippinger.

[1] Note that we will change this definition later in the lecture

1.1 Examples

Example 1. *Matrix multiplication (over GF(2), for simplicity) is in* ***FNC****. The circuits will have depth* $\log n$ *and size* $2n^3$*.*

The formula for elements of the resulting matrix is as follows:

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

We can compute each $c_{i,j}$ *in depth* $\log n$*: Compute* $a_{i,k}b_{k,j}$ *for each* k *in parallel. Then, to*

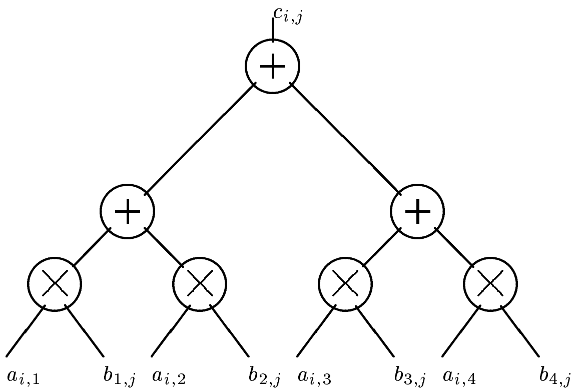

Figure 1: Matrix multiplication for $n = 4$

add these n *numbers we make a tree of depth* $\log n$ *(see fig. 1 for an example). We will use* $2n$ *nodes in each tree. Total is* $2n^3$ *gates.*

In practice, we usually don't have $2n^3$ *processors. We can adapt this scheme for coarse-grained parallelism by having each processor simulate several gates.*

Example 2. *Graph reachability is in* ***NC****.*

The problem of graph reachability is as follows: We are given a directed graph as input, which has two special nodes s *and* t*. Does this graph contain a path from* s *to* t*? The standard search algorithm does not parallelize well: Suppose the graph is one long path from* s *to* t *with* n *nodes. Then it will take* n *steps to get from* s *to* t*. We can, however, use matrix multiplication to solve this problem.*

Let's make a new matrix product, in which $c_{i,j} = \bigvee_k a_{i,k} \wedge b_{k,j}$*. Say* A *and* B *are adjacency matrices (which have self-loops for all nodes). Then* $c_{i,j}$ *is 1 if it is possible go from* i *to* j *by taking a step in* A *followed by a step in* B*. We want to compute the transitive closure of our graph* G*,* G^n*. We will compute it by repeated squaring. Thus we will have* $\log n$ *multiplications, each with* n^3 *size and* $\log n$ *depth.*

The resulting circuit has $n^3 \log n$ *size and* $\log^2 n$ *depth.*

It has been shown last semester that directed graph reachability is complete for **NL**. Therefore, we have

Corollary 1. ***NL*** $\subseteq$ ***NC****.*

Example 3. *Some problems are known to be in **NC**, but not in a practical fashion. For instance, determinant is in **NC** but the best **NC** algorithm for it has size n^8 and depth $\log^4 n$.*

1.2 Log-space mapping reduction

Definition 2 (log-space mapping reduction). *$A \leq_{\log} B$ if $\exists$ a log-space computable f s.t. $x \in A$ iff $f(x) \in B$.*

Note: in order to make sense of this definition, we need to have a read-only input tape and a write-only output tape.

It has been shown last semester that

Lemma 1. *If $B \in$ **LOGSPACE** and $A \leq_{\log} B$, then $A \in$ **LOGSPACE**.*

We'll re-define **NC** to be logspace-uniform: in addition to the properties in the definition above, require that there exist a space $O(\log^k n)$ machine M s.t. on input 1^n, M outputs c_n.

One consequence of logspace-uniformity is

Lemma 2. *$\mathbf{NC}^k \subseteq \mathbf{SPACE}(\log^k n)$*

Note that this might be false if we used polytime-uniformity (i.e., M was only required to be polytime).

Proof. The circuit can be generated in logspace by definition. We can then evaluate the circuit using DFS starting from the top. The whole circuit never needs to be stored, since whenever we need to find out children of a gate, we simulate the machine that generates the circuit until it gets to that point. This mode of evaluation could take up to $2^{\log^k n}$ time. The space needed is just the size of the stack, which is the depth of the circuit. $\square$

It is not known whether $\mathbf{NC}^k = \mathbf{SPACE}((\log n)^k)$. That would be unlikely, however, since $\mathbf{NC} \subseteq \mathbf{P}$ and it is unlikely that $\mathbf{SPACE}((\log n)^k) \subseteq \mathbf{P}$.

1.3 NC vs. P

The big question in the theory of parallel computation is whether $\mathbf{NC} = \mathbf{P}$. There exist problems that are **P**-complete under logspace reductions. (Note that all problems presently known to be **NP**-complete are complete under logspace reductions.)

1.3.1 Circuit value problem

The circuit value problem CVAL is, given a circuit C and inputs $x_1, \ldots, x_n$, output $C(x_1, \ldots, x_n)$.

Theorem 1. *CVAL is **P**-complete under logspace reductions.*

Proof. Obviously, the problem is in **P**. To show that CVAL is **P**-complete means for every TM M being able to construct a logspace computable function that takes as input $\bar{x}$, and outputs a circuit C and an input $\bar{y}$ s.t. $C(\bar{y})$ accepts iff M accepts x in time t.

Let $t(n)$ be a bound on the running time of M. We will make a table c, where $c_{i,t}$ is the contents of cell i at time t. $c_{i,t}$ only depends on $c_{i-1,t}$, $c_{i,t}$, and $c_{i+1,t}$. We will have for each i and t a circuit segment (of constant size) which computes $c_{i,t}$ from those three inputs. Our machine will be able to output the description of the whole circuit C by going through a loop over all the cells. $\bar{y}$ is $\bar{x}$ extended with blanks.

Thus, the circuit produced from a $t(n)$-time machine will have size $(t(n))^2$.[2] $\square$

[2]It is possible to get a circuit of size $t(n) \log t(n)$, but that is tricky.