13.1 Introduction

A natural model of algebraic computation (for polynomials over a field, say) is the *arithmetic circuit*, using the arithmetic operations $+, -, *, /$. Sections 13.2 and 13.3 give some easy parallel algorithms, and Section 13.4 deals with the first fundamental problem: how to solve systems of linear equations fast in parallel (and: how to compute the determinant or characteristic polynomial of a matrix). As usual in this volume, "fast parallel" means parallel time $(\log n)^{O(1)}$ (in fact, $O(\log^2 n)$ for our problems), and $n^{O(1)}$ processors, where n is the input size.

Section 13.5 introduces the important tool of *reductions* within our framework. This allows us to consider the "relative difficulty" of one problem with respect to another, without requiring precise knowledge of the "absolute difficulty" of these problems.

Section 13.6 gives the second fundamental algorithm: computing the rank of a matrix. With this machinery, the elementary problems of linear algebra can be classified into very few groups, with the same parallel complexity within each group (although we may not know what exactly this complexity is).

In Section 13.8, the model is extended to include *tests for zero* and *selection*; this is necessary in order to deal with problems like general (possibly singular) systems of linear equations, or the rank of matrices. Furthermore, the *parallel complexity classes* NC_F^k and SAC_F^k are introduced, analogous to the Boolean complexity classes NC^k and SAC^k.

The pervasion of the theoretical tool of reductions gives the following hint for the design of parallel computers: implement one of the problems in dedicated hard/software, and then use subroutine calls to this one problem to solve all others.

The last section mentions several other topics in parallel arithmetic complexity which cannot be discussed in detail here: the exponentiation problem in finite fields, which shows that for certain tasks, our arithmetic circuits are *not* the right model, the surprising fact that "$NC^2 = P$" for polynomial computations, optimal algorithms for division with remainder of polynomials, computing normal forms of matrices, general lower bounds in terms of the degree, and permutation group algorithms.

The theory of parallel algebraic computation as presented here is a younger cousin of the more classical sequential algebraic complexity theory, which has well-established models of computation and fundamental results.

Surveys of this sequential theory are in Strassen [51] and von zur Gathen [25]. The present chapter is largely based on von zur Gathen [23].

The main prerequisite for our subject is basic linear algebra, plus the material typically presented in a one-semester introductory algebra course. Only the last section requires more algebraic background.

The material has been used in courses and seminars at University of Toronto (Canada), Université Laval (Québec, Canada), Universität Zürich (Switzerland), Universität des Saarlandes (Saarbrücken, Germany), and Universidad Católica de Santiago (Chile).

13.2 Arithmetic Circuits

The simplest case of algebraic computation is provided by an *arithmetic circuit* (called a *straight-line program* in Strassen [49]) such as in Figure 13.1.

This arithmetic circuit computes the two polynomials $3x_1 + \sqrt{2}\,x_2$ and $3x_2 - \sqrt{2}\,x_1$; 3 and $\sqrt{2}$ are constants belonging to the ground domain $\mathbb{R}$, and x_1 and x_2 inputs.

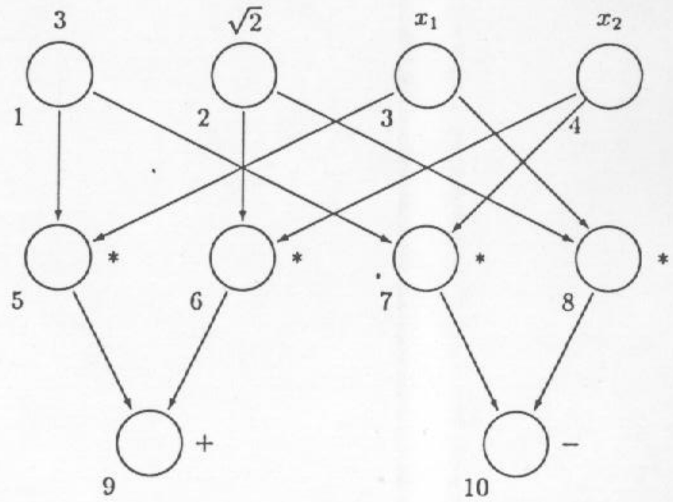


FIGURE 13.1
An arithmetic circuit over $\mathbb{R}$.

In general, we have a ground ring F, and an arithmetic circuit over F is a labelled directed acyclic graph. (Recall that a *ring* is equipped with two binary operations $+$ and $*$ and elements 0 and 1 having the usual properties; the ring $\mathbb{Z}$ of integers is an example.) The label of each gate (or node) has two components. The first component is either a constant from F, an input, or an arithmetic operation $+, -, *, /$. The second component is a numbering of the gates. This numbering is only needed to distinguish the two inputs of the non-commutative operations "$-$" and "/": the lower numbered input is the first operand. (In Section 13.8, we will have to introduce more operations, such as testing for zero.) This model of computation forms the theoretical basis for much of *computer algebra*, where all operations are exact, in contrast to numerical computations of limited precision.

In Figure 13.1, we have the two constants 3 and $\sqrt{2}$ from the ground field $F = \mathbb{R}$ of real numbers, and two inputs x_1 and x_2. Interchanging the two second components 7 and 8 of labels, the arithmetic circuit would compute $\sqrt{2}\,x_1 - 3x_2$ at gate 10. In further examples, we will usually leave out this second component of labels; in the figures, the "left" input is the first operand.

If F is a field and $x_1, \ldots, x_n$ are the inputs, then at each gate a rational function in $F(x_1, \ldots, x_n)$ is computed. (A *field* is a ring with the further property that each nonzero element has a multiplicative inverse; an example is the field $\mathbb{Q}$ of rational numbers.) A technical requirement is that no division by the rational function zero may occur. (In the general case, where F is a ring, but not necessarily a field, each division must be executable in the ring; this is the case, e.g., when the denominator has an inverse.) We say that an arithmetic circuit *computes* any of the rational functions computed at any of its gates.

In Figure 13.2, the graph to the left is not an arithmetic circuit, since division by $x - x = 0$ occurs. The arithmetic circuit to the right computes the rational function $1/(x^2 - x)$. Although a division by zero occurs for the special inputs 0 and 1 for x, it is still a legal arithmetic circuit—even in the extreme case that the ground field $F = \mathbb{Z}_2$ contains only 0 and 1.

Two measures of an arithmetic circuit α are of interest here. The *depth* $D(\alpha)$ (= parallel time) is the number of arithmetic operations on a longest path in the graph of α. The *size* $S(\alpha)$ (or sequential time) is the total number of arithmetic operations. (It equals the "number of processors" when processors are not re-used.) For the α of Figure 13.1, we have $D(\alpha) = 2$ and $S(\alpha) = 6$.

Since all operations are binary (i.e., with two inputs), increasing the depth by one can at most double the number of inputs, so that depth $\log_2 n$ is optimal.

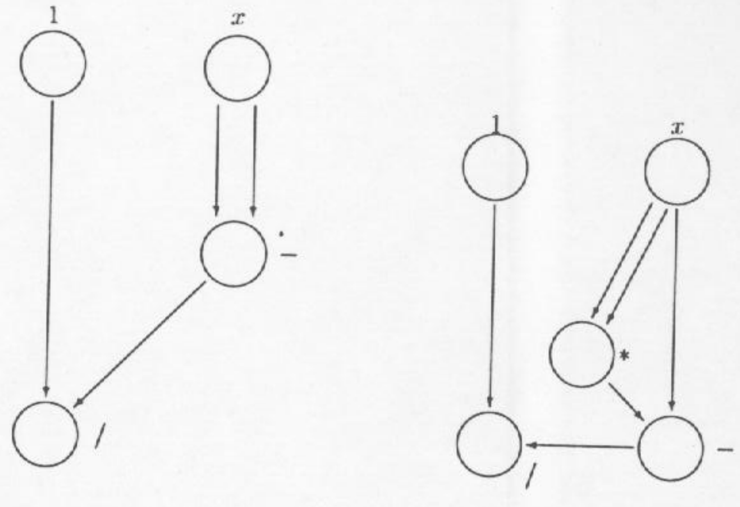


FIGURE 13.2
Two examples.

THEOREM 13.1
Let α be an arithmetic circuit. Then

1. $D(\alpha) \le S(\alpha)$.
2. *If there exists a single "output gate", with a directed path from every gate to it, then* $S(\alpha) \le 2^{D(\alpha)} - 1$.
3. *If there exists an "output gate" with a directed path from each of n inputs to it, then* $D(\alpha) \ge \log_2 n$.

PROOF

(1) is trivial. To prove (2), for any gate v in α, we consider the maximal length $D(v)$ of paths (where the length of a path is the number of arithmetic gates on it) leading from inputs or constants to v; thus $D(v)$ is the depth of v. We show by induction on $D(v)$ that v is connected to at most $2^{D(v)} - 1$ arithmetic gates. (We do not count the input or constant gates.) This is sufficient, since by assumption, there is some gate v connected to all $S(\alpha)$ gates, so that then $2^{D(\alpha)} - 1 \ge 2^{D(v)} - 1 \ge S(\alpha)$.

If $D(v) = 0$, then v is an input or constant gate. If $D(v) > 0$, let w_1 and w_2 be the two input gates to v. Then

$$D(v) = \max\{D(w_1), D(w_2)\} + 1,$$

and from the induction hypothesis it follows that the number of arithmetic gates connected to v is at most

$$(2^{D(w_1)}-1)+(2^{D(w_2)}-1)+1 \leq 2^{D(v)-1}+2^{D(v)-1}-1 = 2^{D(v)}-1, \quad (13.1)$$

where the $+1$ comes from the fact that v is connected to v.

The proof of (3) is similar (Exercise 13.1.a). ∎

Note that this is a purely graph-theoretic proof, independent of the types of gates we use, and thus valid for any directed acyclic graphs with fan-in two.

As an example, we consider the "iterated" sum $f = x_1 + \cdots + x_n$ of n indeterminates. A binary tree α of additions computes f with $D(\alpha) = \lceil \log_2 n \rceil$ and $S(\alpha) = n-1$. Figure 13.3 shows the case $n = 7$. The same size and depth works also for the product $x_1 \cdots x_n$.

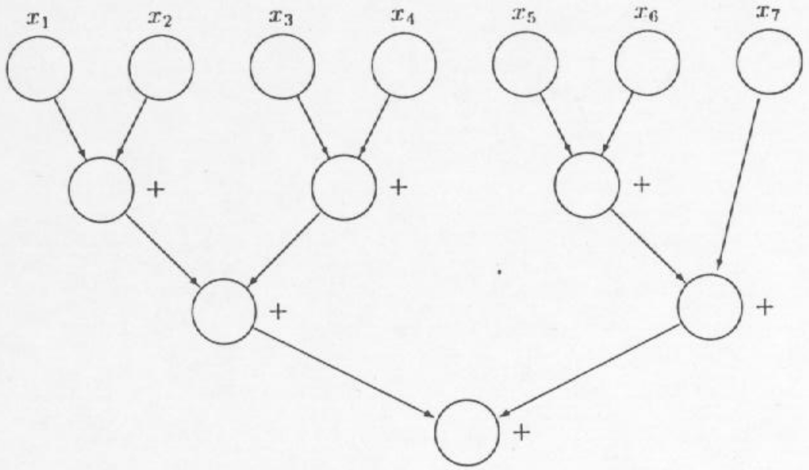


FIGURE 13.3
A binary addition tree, for $n = 7$.

Theorem 13.1 (3) says, in particular, that the binary trees for iterated sum or product cannot be improved in depth; the same is true for the size (see Exercise 13.1.b).

Only for rather simple problems like iterated sum are such optimal circuits known; Section 13.9 mentions other examples. In the next few sections we pursue a more modest but more realistic goal. For a variety of problems we will present algorithms which are not too far off from the trivial lower bounds of $\log_2 n$ and n for depth and size, resp. More precisely, their depth should be polynomial in $\log n$ (also called *polylogarithmic in* n, $O((\log n)^k)$ for some fixed k, or $(\log n)^{O(1)}$), and their size polynomial in n, i.e., $O(n^k)$ for some fixed k, or $n^{O(1)}$. Implicit in such asymptotic notions is the assumption that we are not dealing with a single computational problem, but with an infinite family $(f_n)_{n\in\mathbb{N}}$ of rational functions indexed by a parameter n, say $f_n \in F(x_1, \ldots, x_n)$, and a circuit family $(\alpha_n)_{n\in\mathbb{N}}$ with α_n computing f_n.

EXERCISE 13.1
Let α be a directed acyclic graph with indegree at most two, n "inputs" (= vertices with indegree 0), and a special "output" vertex (with outdegree zero) to which every vertex is connected.

a) Prove that the depth of α is at least $\log_2 n$.
b) Prove that the size of α is at least $n - 1$.

13.3 Multiplication

In this section, we consider multiplication of matrices and polynomials, and inversion of polynomials modulo a power of the indeterminate.

Given two square matrices $A, B \in F^{n\times n}$ over a ring F, their product $C = A \cdot B \in F^{n\times n}$ has entries

$$C_{ik} = \sum_{1\leq j\leq n} A_{ij}B_{jk}$$

for $1 \leq i, k \leq n$. ($F^{n\times n}$ is the ring of $n \times n$-matrices with entries from F.) An arithmetic circuit is obvious from the formula:

1. For all i, j, k $(1 \leq i, j, k \leq n)$ compute $A_{ij} \cdot B_{jk}$.
2. For all i, k $(1 \leq i, k \leq n)$ compute C_{ik} as the above sum.

The depth of this circuit is $1 + \lceil \log_2 n \rceil = O(\log n)$, and the size is $n^3 + n^2(n-1) = O(n^3)$. By Theorem 13.1 (3) the circuit is depth-optimal; is it also size-optimal? It is not obvious, but in fact the size of matrix multiplication circuits may be drastically improved. The first surprising improvement was by Strassen [48], to $O(n^{2.81})$, and the smallest size known today is $O(n^{2.376})$, with depth still $O(\log n)$ (Coppersmith & Winograd [13]).

An extension is the problem of *iterated matrix multiplication*, where we are given matrices $A_1, \ldots, A_n \in F^{n\times n}$, and want to compute their product $A_1 \cdots A_n$. We can form a binary tree, of depth $\lceil \log_2 n \rceil$ and size $n-1$, with each "gate" being a multiplication of two matrices. The resulting depth is $O(\log^2 n)$, and the size $O(n^4)$.

The ring $F[x]$ of polynomials in x over a ring F consists of formal expressions of the form

$$f = a_0 + a_1 x + \cdots + a_n x^n \in F[x],$$

with $n \in N$ and $a_0, \ldots, a_n \in F$. This f has degree at most n; if $a_n \neq 0$, then the degree $\deg f$ is equal to n. Given a second polynomial $g = \sum_{0\le j\le n} b_j x^j$, their product $h = \sum_{0\le k\le 2n} c_k x^k = f \cdot g$ has coefficients

$$c_k = \sum_{\substack{0\le i,j\le n \\ i+j=k}} a_i b_j.$$

The obvious circuit:

1. For all i, j $(0 \le i, j \le n)$ compute $a_i \cdot b_j$.
2. For all k $(0 \le k \le 2n)$ compute c_k as the above sum.

has depth $O(\log n)$ and size $O(n^2)$.

We note that addition of two matrices or polynomials can be done in depth 1, and iterated addition of n such items in depth $O(\log n)$.

Iterated polynomial product is the problem of computing $f_1 \cdots f_n$, where $f_1, \ldots, f_n \in F[x]$ have degree at most n. A binary tree of polynomial multiplications solves this problem in depth $O(\log^2 n)$. What is the resulting size? Let us assume for simplicity that n is a power of 2, and consider the levels 0 (inputs), 1, 2, ..., $k = \log_2 n$ (output) of the tree. At level i, a total of $n/2^i$ multiplications of polynomials of degree at most $2^{i-1}n$ are performed, in depth $O(\log n)$ and size at most

$$n/2^i \cdot O((2^{i-1}n)^2) = O(2^i n^3).$$

Summing these sizes over i, we get $O(n^4)$, and overall depth $O(\log^2 n)$. We will mention in Section 13.9 that this problem can even be solved in optimal depth $O(\log n)$.

Given three polynomials $f_1, f_2, g \in F[x]$, f_1 and f_2 are called *congruent modulo g* if their difference is divisible by g:

$$f_1 \equiv f_2 \bmod g \iff \exists h \in F[x] \;\; f_1 - f_2 = gh.$$

If $f \in F[x]$ and $n \in N$, then f is *invertible modulo x^{n+1}* if there exists some $g \in F[x]$ such that $fg \equiv 1 \bmod x^{n+1}$; such a g is called a (modular) *inverse* of f. If $f = a_0 + \cdots$ is invertible modulo x^{n+1}, then it is invertible mod x, so that $a_0 = f(0) \in F$ is invertible; if F is a field, this is equivalent to $a_0 \neq 0$. The algorithm below shows that also the converse is true.

As an example, let $n = 3$ and $f = 1 - x$. Then $g = 1 + x + x^2 + x^3$ satisfies

$$fg = 1 - x^4 \equiv 1 \bmod x^4,$$

and g is an inverse of f modulo x^4.

The following algorithm generalizes this *geometric series*.

ALGORITHM 13.1
Polynomial inversion
Input: $f \in F[x]$ with $f(0) \in F$ invertible.
Output: $g \in F[x]$ with $fg \equiv 1 \bmod x^{n+1}$.

1. Compute $b = f(0)^{-1} \in F$,
2. compute $h = (f(0) - f) \cdot b \in F[x]$,
3. for all i, $0 \le i \le n$, compute h^i,
4. return $g = b \cdot \sum_{0\le i\le n} h^i$.

EXAMPLE 13.1
Let $n = 3$ and $f = 1 + 7x + 29x^2 + 80x^3 \in \mathbb{Q}[x]$. Then $f(0) = 1$ and $h = -7x - 29x^2 - 80x^3$. The algorithm calculates

$$g = 1 + h + h^2 + h^3 \equiv 1 - 7x + 20x^2 - 17x^3 \bmod x^4.$$

Note that we have left out the terms of order 4 or higher, since they are irrelevant modulo x^4. The reader might check that indeed $fg \equiv 1 \bmod x^4$.

We first convince ourselves that algorithm **Polynomial inversion** works correctly. Note that x divides h (which we write as "$x \mid h$"), so that $x^{n+1} \mid h^{n+1}$, or $h^{n+1} \equiv 0 \bmod x^{n+1}$. Thus

$$fg = (1-h)f(0) \cdot b \sum_{0 \le i \le n} h^i = 1 - h^{n+1} \equiv 1 \bmod x^{n+1}, \qquad (13.2)$$

and g is indeed a modular inverse of f.

Making use of iterated polynomial product for the powers in step 2, the depth is $O(\log^2 n)$, and the size $O(n^5)$.

The algorithm displays the technique of "reduction" that we will use profusely in the next sections: solving one problem (here: modular inversion) by appealing to another one (here: iterated polynomial product). Although conceptually important and convenient, it has the disadvantage of blowing up the size (and sometimes the depth) more than necessary. In our case, we observe that—as in Example 13.1—we only need all polynomials modulo x^{n+1}, i.e., only the first $n+1$ coefficients. If we truncate all results modulo x^{n+1}, we only have to perform $n-1$ multiplications of polynomials of degree at most n, resulting in size $O(n^3)$. We have proved the following result.

THEOREM 13.2
Let F be a ring. Polynomials in $F[x]$ with constant term invertible in F can be inverted modulo x^{n+1} in depth $O(\log^2 n)$ and size $O(n^3)$.

13.4 The Determinant

The parallel algorithms discussed so far were straightforward. We now turn to a fundamental problem for which a good parallel solution is not obvious: the solution of systems of linear equations.

The problem is of central importance, and many sequential algorithms for it are well-studied. Suppose we want to solve

$$Ax = b,$$

where an $n \times n$-matrix $A \in F^{n \times n}$ over the ground field F and an n-vector $b \in F^n$ are given, and we are looking for a vector $x \in F^n$ satisfying the equation. Such a solution exists if and only if b is a linear combination of the n columns of A, and if the determinant $\det A$ is nonzero, there exists a unique solution x.

The classical algorithm of *Gaussian elimination* consists of n stages. In each stage, appropriate scalar multiples of a "pivot row" are subtracted from other rows to introduce zero entries in one column. The end result is an upper triangular system of linear equations with the same solutions as the original one. It can now easily be solved by "back-substitution". All the row operations of one stage can easily be performed in three parallel operations. However, the execution of the stages looks inherently sequential, and it is not clear how to obtain parallel time less than n, say.

We now discuss a very different algorithm, invented by the Leningrad mathematician Chistov [11]. Csanky [14] had presented the first parallel algorithm for the determinant using depth $O(\log^2 n)$ and size $n^{O(1)}$ (Exercise 13.4). It has the merit of being the first nontrivial parallel algorithm in linear algebra, within the framework of this chapter. Unfortunately, Csanky's algorithm only works over fields F of characteristic zero, i.e., if $\mathbb{Q} \subseteq F$, and this excludes the important case of finite fields. Next, Borodin et al. [8] gave an (admittedly awful) solution for the general case. Soon after that, Berkowitz [6]—then a student at University of Toronto—found an algorithm that competes with Chistov's in cost and clarity.

If $Ax = b$, $x = (x_1, \ldots, x_n) \in F^n$, and $\det A \neq 0$, then *Cramer's rule* says that $x_i = \det A^{[i]} / \det A$, where $A^{[i]} \in F^{n \times n}$ is obtained by substituting b for the ith column vector of A. Thus it is sufficient to compute determinants of matrices. Note that after performing Gaussian elimination, $\det A$ is the product of the diagonal entries of the resulting upper triangular matrix, and thus easy to compute.

We will actually solve.the seemingly harder problem of computing the characteristic polynomial

$$\chi(A) = \det(xI_n - A) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1} + x^n \in F[x]$$

of a matrix A, where F is a ring, $I_n \in F^{n \times n}$ the identity matrix (with ones on the diagonal, and zeroes elsewhere), and x an indeterminate. Then $\det A = (-1)^n c_0$ can be read off (and $-c_{n-1}$ is the sum of the diagonal entries of A).

EXAMPLE 13.2
Let us take

$$A = \begin{array}{|ccc|} 2 & -3 & 0 \\ 1 & 2 & -1 \\ -2 & 1 & 3 \end{array} \in \mathbb{Q}^{3 \times 3}.$$

Then

$$\chi(A) = \det \begin{vmatrix} x-2 & 3 & 0 \\ -1 & x-2 & 1 \\ 2 & -1 & x-3 \end{vmatrix} = -17 + 20x - 7x^2 + x^3,$$

and $\det A = -17$. ■

If F is a field, $A = (a_{ij})_{1\le i,j\le n} \in F^{n\times n}$ and $1 \le r \le n$, we consider the lower right submatrix

$$A_r = (a_{ij})_{r\le i,j\le n} \in F^{r'\times r'}$$

of A, where $r' = n - r + 1$ (see Figure 13.4). (Thus the rows and columns of A_r are indexed by $r, r+1, \ldots, n$.) If we let

$$d_r = \det(I_{r'} - xA_r) \in F[x],$$

then

$$\begin{aligned} \chi(A) &= \det(xI_n - A) = \det\left(xI_n \cdot (I_n - x^{-1}A)\right) \\ &= \det(xI_n)\cdot\det(I_n - x^{-1}A) = x^n \det(I_{1'} - x^{-1}A_1) = x^n d_1(x^{-1}). \end{aligned}$$

The polynomial $x^n d_1(x^{-1})$ is called the *reversal* (for degree n) of d_1, since its coefficient sequence is the reversed coefficient sequence of d_1.

The matrix $I_{r'} - xA_r \in F[x]^{r'\times r'}$ is invertible over $F(x)$, since its determinant d_r is a nonzero polynomial, with value 1 at $x = 0$. We denote by

$$B^{(r)} = (b_{ij}^{(r)})_{r\le i,j\le n} = (I_{r'} - xA_r)^{-1}$$

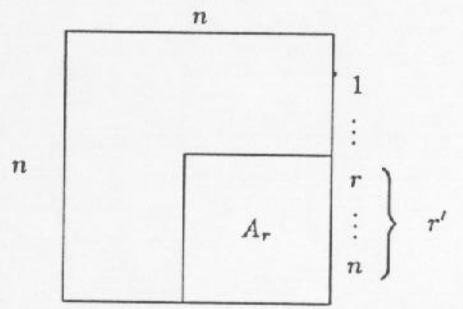


FIGURE 13.4
The lower right submatrix A_r.

its inverse. Thus each $b_{ij}^{(r)} \in F(x)$ is a rational function in x, and $b_{ij}^{(r)} \cdot d_r \in F[x]$. Let us further denote by $b_{r\bullet}^{(r)} = (b_{rj}^{(r)})_{r\le j\le n}$ the leftmost column, and by $b_r = b_{rr}^{(r)} \in F(x)$ the top left entry of $B^{(r)}$. Then by definition of the inverse

$$(I_{r'} - xA_r)\cdot b_{r\bullet}^{(r)} = (1, 0, \ldots, 0)^t.$$

The determinant of the matrix obtained by substituting $(1, 0, \ldots, 0)^t$ for the leftmost column of $I_{r'} - xA_r$ equals

$$\det(I_{(r+1)'} - xA_{r+1}) = d_{r+1}.$$

Thus expressing the top entry b_r of $b_{r\bullet}^{(r)}$ according to Cramer's rule, we find

$$b_r = d_{r+1}/d_r. \tag{13.3}$$

(With $d_{n+1} = 1$, we note that $d_r(0) = 1$ for all r, so that in particular all denominators are nonzero.) Multiplying all equations (13.3) together, we obtain

$$\prod_{1\le r\le n} b_r = d_1^{-1}.$$

This is an equation between rational functions in x (which again evaluate to 1 at $x = 0$).

A special case of the geometric series for the inverse (13.2) is

$$(1 - xa)\cdot \sum_{0\le k\le n} x^k a^k \equiv 1 \bmod x^{n+1}. \tag{13.4}$$

This equation holds when a is in any ring and x an indeterminate over this ring commuting with a. We apply (13.4) with $a = A_r$, and obtain

$$B^{(r)} = (I_{r'} - xA_r)^{-1} \equiv \sum_{0\le k\le n} x^k A_r^k \bmod x^{n+1}.$$

If we define $\tilde{B}^{(r)}$ as the sum on the right hand side, and $\tilde{b}_r$ as its top left entry, then each $\tilde{b}_{ij}^{(r)} \in F[x]$ is a polynomial in x of degree at most n, the denominator d_r of b_r is invertible modulo x, and $b_r \equiv \tilde{b}_r \bmod x^{n+1}$. Thus we have

$$\prod_{1\le r\le n} \tilde{b}_r \equiv d_1^{-1} \bmod x^{n+1}.$$

Putting things together, we have the following algorithm.

ALGORITHM 13.2

Characteristic polynomial

Input: A matrix $A \in F^{n\times n}$, where F is a commutative ring with 1, and $n \in N$.

Output: The coefficients $c_0, \ldots, c_n$ of $\chi(A) \in F[x]$.

1. For all k, r $(1 \le k, r \le n)$, compute A_r^k and $\tilde{b}_r = \left(\sum_{0\le k\le n} x^k A_r^k\right)_{rr}$.
2. Compute $b \in F[x]$ with $\deg b \le n$ and
$$b \equiv \prod_{1\le r\le n} \tilde{b}_r \bmod x^{n+1}.$$
[Then $b \equiv d_1^{-1} \bmod x^{n+1}$.]
3. Compute $c \in F[x]$ with $\deg c \le n$ and $c \equiv b^{-1} \bmod x^{n+1}$, using the algorithm **Polynomial inversion.** [Then $c = d_1$.]
4. Return the coefficients of the reverse $\chi(A) = x^n c(x^{-1})$ of c.

We have already seen that the algorithm works correctly over a field F. The cost follows from the estimates of the subroutines used:

Step	Subroutine	Depth	Size
1	iterated matrix product	$O(\log^2 n)$	$O(n^6)$
2	iterated polynomial product	$O(\log^2 n)$	$O(n^3)$
3	polynomial inversion	$O(\log^2 n)$	$O(n^3)$
4		0	0

For the estimates of steps 2 and 3, we use the truncating algorithm of Theorem 13.2. Step 4 is free, since only the coefficient sequence is reversed.

The algorithm actually works over an arbitrary commutative ring with 1. For this, we note that the algorithm has no divisions (the "division" in step 3 is by 1), and computes the characteristic polynomial of a matrix over $\mathbb{Z}$ with indeterminate entries. Therefore it computes $\chi(A)$ for any square matrix A over a commutative ring F.

EXAMPLE 13.3

We first trace the algorithm on the matrix of Example 13.1, and then check one of the equations used in deriving the algorithm.

$$A_1 = A = \boxed{\begin{matrix} 2 & -3 & 0 \\ 1 & 2 & -1 \\ -2 & 1 & 3 \end{matrix}},$$

1

$$A_1^2 = \boxed{\begin{matrix} 2 & -12 & 3 \\ 6 & 0 & -5 \\ -9 & 11 & 8 \end{matrix}},\quad A_1^3 = \boxed{\begin{matrix} -16 & -24 & 21 \\ 22 & -23 & -15 \\ -23 & 57 & 13 \end{matrix}},$$

$$A_2 = \boxed{\begin{matrix} 2 & -1 \\ 1 & 3 \end{matrix}},\quad A_2^2 = \boxed{\begin{matrix} 3 & -5 \\ 5 & 8 \end{matrix}},\quad A_2^3 = \boxed{\begin{matrix} 1 & -18 \\ 18 & 19 \end{matrix}},$$

$$A_3 = \boxed{3},\quad A_3^2 = \boxed{9},\quad A_3^3 = \boxed{27},$$

$$\begin{aligned} \tilde{b}_1 &= 1 + 2x + x^2 - 16x^3 \ (\equiv b_1 \bmod x^4), \\ \tilde{b}_2 &= 1 + 2x + 3x^2 + x^3 \ (\equiv b_2 \bmod x^4), \\ \tilde{b}_3 &= 1 + 3x + 9x^2 + 27x^3 \ (\equiv b_3 \bmod x^4), \\ b &= 1 + 7x + 29x^2 + 80x^3, \\ c &= 1 - 7x + 20x^2 - 17x^3 \ \ \text{(see Example 13.1)}, \\ \text{reverse of } c &= -17 + 20x - 7x^2 + x^3, \end{aligned}$$

which is indeed $\chi(A)$. Here is the special case $b_1 = d_2/d_1$ of equation (13.3):

$$\begin{aligned} d_1 \cdot b_1 &= \det(I_3 - xA_1) \cdot b_1 \equiv (1 - 7x + 20x^2 - 17x^3) \cdot \\ &\qquad (1 + 2x + x^2 - 16x^3) \\ &= 1 - 5x + 7x^2 = \det(I_2 - xA_2) = d_2 \bmod x^4. \end{aligned}$$

With a little care, we can improve the size of the circuit to $O(n^4)$, in fact, to even less. The following notation is convenient. We consider all families $\alpha = (\alpha_n)_{n\in N}$ of arithmetic circuits, where α_n computes the product of two $n \times n$-matrices over F; then α *computes matrix multiplication.* We define the (parallel) matrix multiplication size as the smallest size sufficient to multiply matrices with logarithmic depth:

$$M(n) = \min\{S(\alpha_n)\colon \ \alpha \text{ computes matrix multiplication and } D(\alpha_n) = O(\log n)\}.$$

The standard algorithm used in Section 13.3 shows $M(n) \le 2n^3$. Clearly $M(n) \ge n^2$, since n^2 outputs have to be computed; the best lower bound known today is $M(n) \ge 2n^2 - 1$ (Brockett & Dobkin [9]). The best upper bound known today is by Coppersmith & Winograd [13]:

$$2n^2 - 1 \le M(n) = O(n^{2.376}).$$

The reader misses nothing essential in the following if she thinks of $M(n) = 2n^3$ and the standard matrix multiplication algorithm. The following observation is from von zur Gathen & Eberly [28].

LEMMA 13.1
If $A \in F^{n\times n}$ and $b \in F^n$, then all vectors $A^0 b$, $A^1 b$, $A^2 b, \ldots, A^n b \in F^n$ can be computed in depth $O(\log^2 n)$ and size at most $2M(n)(1+\log_2 n)$.

PROOF
Let $l = \lceil \log_2 n \rceil < 1 + \log_2 n$, so that $n \le 2^l$. In a first stage we compute

$$A^{2^0}, A^{2^1}, A^{2^2}, \ldots, A^{2^l},$$

in depth $O(\log^2 n)$ and size $lM(n)$. In the second stage, we compute successively $B_0, B_1, B_2, \ldots, B_l \in F^{n\times n}$ as follows. B_0 has b as its first column, and zeroes elsewhere. B_i's first 2^{i-1} columns equal those of B_{i-1}, the next 2^{i-1} columns equal the first 2^{i-1} columns of $A^{2^{i-1}} \cdot B_{i-1}$, and the other columns are zero. (In B_l, only the next $n - 2^{l-1}$ columns are new.) One sees inductively that the jth column of B_i is $A^{j-1}b$ if $1 \le j \le 2^i$, and zero otherwise. (In B_l, this is valid for $1 \le j \le n$.) The cost is the same as for the first stage. ∎

In step 1 of the algorithm **Characteristic polynomial**, we do not really need all A_r^k, but only all top left entries $e_r A_r^k e_r^t$, where $e_r = (1, 0, \ldots, 0) \in F^{r'}$. For any $r \le n$, all vectors $A_r^k e_r^t$ $(0 \le k \le n)$ can be computed in size at most $2M(n)\log_2 n$ by the lemma. Thus we have the following result.

THEOREM 13.3
The characteristic polynomial of $n \times n$-matrices can be computed on an arithmetic circuit of depth $O(\log^2 n)$ and size at most $2nM(n)(1 + \log_2 n)$, or size $O(n^4 \log n)$.

A better size bound $O(n^{1/2}M(n))$ can be obtained in characteristic zero (Preparata & Sarwate [45], and a slight improvement in the general case is in Galil & Pan [20]. Kaltofen & Pan [37] show how to solve $Ax = b$ by a probabilistic circuit of depth $O(\log^2 n)$ and size $O(M(n)\log n)$ if A is nonsingular, the characteristic of the field F is zero or larger than n, and the field is sufficiently large (say, $\#F \ge 6n^2$).

EXERCISE 13.2 (Inversion of triangular matrices)
Let F be a field, and

$$A = \begin{pmatrix} B & 0 \\ C & D \end{pmatrix} \in F^{2\times 2}$$

a non-singular lower triangular matrix. Show that

$$A^{-1} = \begin{pmatrix} B^{-1} & 0 \\ -D^{-1}CB^{-1} & D^{-1} \end{pmatrix}.$$

Generalize this fact to arbitrary non-singular lower triangular matrices $A \in F^{n\times n}$ and use your results to construct a recursive parallel algorithm for computing the inverse of such matrices in $O(\log^2 n)$ time using $O(n^3)$ processors.

EXERCISE 13.3 (Linear recurrences)
Assume you are given a system of linear recurrences

$$\begin{aligned} x_1 &= c_1, \\ x_2 &= a_{21}x_1 + c_2, \\ &\vdots \\ x_n &= a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn-1}x_n + c_n, \end{aligned}$$

where $a_{ij}, c_i \in F$, and F is a field. Letting $A = (a_{ij}) \in F^{n\times n}$, $x = (x_i) \in F^n$ and $c = (c_i) \in F^n$, this can be rewritten

$$Ax + c = x.$$

Give an efficient parallel algorithm to solve this system of linear recurrences. [Hint: Use your solution to Exercise 13.2.]

EXERCISE 13.4 (Csanky's algorithm)
Let F be a field of characteristic zero (or of characteristic larger than n), $A \in F^{n\times n}$ and write

$$\chi = \det(Ix - A) = x^n - s_1x^{n-1} + s_2x^{n-2} - \cdots + (-1)^n s_n \in F[x]$$

for the characteristic polynomial of A. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of A (say in an algebraic closure of F); these are just the roots of χ.

a) Show that

$$s_n = \det(A) = \prod_{i=1}^{n} \lambda_i.$$

The trace $\mathrm{tr}(A) \in F$ of A is defined to be the sum of the diagonal entries of A:

$$\mathrm{tr}(A) = \sum_{1 \le i \le n} A_{ii}.$$

b) Show that

$$s_1 = \mathrm{tr}(A) = \sum_{1 \le i \le n} \lambda_i.$$

In other words, the trace of such a matrix is also the sum of its eigenvalues. Show also that $\mathrm{tr}(A^k) = \sum_{i=1}^{n} \lambda_i^k$ for all $k \in \mathbb{N}$.

c) (Newton identities) Prove that

$$s_k = \frac{1}{k}(s_{k-1} \cdot \mathrm{tr}(A) - s_{k-2} \cdot \mathrm{tr}(A^2) + \cdots + (-1)^{k-2} s_1 \cdot \mathrm{tr}(A^{k-1}) + (-1)^{k-1} \cdot \mathrm{tr}(A^k)).$$

[Hint: Use a) and the fact that

$$s_k = \sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} \lambda_{i1} \lambda_{i2} \cdots \lambda_{ik}.$$

In other words, s_k is the *kth elementary symmetric polynomial* in the λ_i's.]

d) Now apply c) and Exercise 13.3 above to give an efficient parallel algorithm for computing the coefficients s_i of the characteristic polynomial $\chi \in F[x]$ of $A \in F^{n \times n}$. .

EXERCISE 13.5 (Inversion of non-singular matrices)
The Cayley-Hamilton theorem states that any matrix A satisfies its characteristic polynomial: if $\chi = x^n - s_1 x^{n-1} + \cdots \pm s_{n-1} x \mp s_n x^0$ is the characteristic polynomial of $A \in F^{n \times n}$, then

$$\chi(A) = A^n - s_1 A^{n-1} + \cdots \pm s_{n-1} A \mp s_n I = 0.$$

Use this fact, together with Exercise 13.4 above, to show that if $A \in F^{n \times n}$ is non-singular, then the entries of A^{-1} can be computed from A in $O(\log^2 n)$ parallel arithmetic steps using $O(n^4)$ processors.

13.5 Polynomial and Matrix Problems

The goal of this section is a set of *reductions* between polynomial and matrix problems. A reduction $f \le g$ is an arithmetic circuit for f of depth $O(\log n)$ that uses g; precise definitions are in Section 13.8. The following construction is useful for our purpose. We let F be any ring, possibly non-commutative, x an indeterminate over F, $d \in \mathbb{N}$, and consider the mapping

$$\tau_d \colon F[x] \;\to\; F^{d \times d}$$

$$a_0 + a_1 x + \cdots \;\mapsto\; \boxed{\begin{matrix} a_0 & a_1 & \cdots & a_{d-1} \\ & & \ddots & \vdots \\ & & \ddots & a_1 \\ 0 & & & a_0 \end{matrix}}\,.$$

Thus the image of τ_d is the set of Toeplitz matrices A in upper triangular form: $A_{ij} = A_{i+k,j+k}$ for all appropriate values of i, j, k, and $A_{ij} = 0$ if $i > j$. The proof of the following lemma is left as Exercise 13.6.

LEMMA 13.2
τ_d is a ring homomorphism with kernel (x^d).

It is convenient to have a standard language to describe our computational problems, such as the following.

PROD $= (\mathrm{PROD}_n)_{n \in \mathbb{N}}$ with $\mathrm{PROD}_n = x_1 \cdots x_n$ is the product problem. DETERMINANT $= (\mathrm{DETERMINANT}_n)_{n \in \mathbb{N}}$ with

$$\mathrm{DETERMINANT}_n = \det\big((x_{ij})_{1 \le i,j \le n}\big)$$

is the determinant problem.

We define further computational problems:

POLYPROD: product of two polynomials.

This is shorthand for defining a family POLYPROD $= (\mathrm{POLYPROD}_n)_{n \in \mathbb{N}}$ of sequences of polynomials $\mathrm{POLYPROD}_n = (c_0, \ldots, c_{2n})$, where $c_k = \sum_{i+j=k} a_i b_j \in F[a_0, \ldots, a_n, b_0, \ldots, b_n]$, and $a_0, \ldots, b_n$ are indeterminates over F.

Similarly, we have

$\mathrm{ITPOLYPROD}_n$:	$f_1 \cdots f_n$ for $f_1, \ldots, f_n \in F[x]$ of degree at most n,
$\mathrm{POLYINV}_n$:	$f^{-1} \bmod x^n$ for $f \in F[x]$, $f(0) \neq 0$,
$\mathrm{MATPROD}_n$:	$A \cdot B$ for $A, B \in F^{n\times n}$,
$\mathrm{ITMATPROD}_n$:	$A_1 \cdots A_n$ for $A_1, \ldots, A_n \in F^{n\times n}$,
MATINV_n:	A^{-1} for $A \in F^{n\times n}$ invertible.

THEOREM 13.4

1. POLYPROD $\leq_F$ MATPROD,
2. POLYINV $\leq_F$ MATINV,
3. POLYINV $\leq_F$ ITPOLYPROD $\leq_F$ ITMATPROD.

PROOF

For (1), suppose we want to compute the product of $f, g \in F[x]$ with degree at most n. By Lemma 13.2 we have

$$\tau_{2n+1}(fg) = \tau_{2n+1}(f) \cdot \tau_{2n+1}(g) = \mathrm{MATPROD}_{2n+1}(\tau_{2n+1}(f), \tau_{2n+1}(g)).$$

Thus the reduction has three (trivial) steps: 1. produce the matrices $\tau_{2n+1}(f)$ and $\tau_{2n+1}(g)$, 2. form their product, by calling MATPROD, and 3. read off the required output. More formally, the reduction circuit has $2n+2$ input gates for the coefficients of f and g, the constant zero, and a single computation gate $\mathrm{MATPROD}_{2n+1}$, with $2(2n+1)^2$ inputs and $(2n+1)^2$ outputs. The input gates and zero are connected to the MATPROD gate according to τ_{2n+1}, and the functions required for POLYPROD are among those computed by the MATPROD gate (in fact, the first row). The depth is 1, and the size is $2(2n+1)^2 - 1$.

The reduction for (2), and the second one in (3) are similar. The first reduction in (3) is given by algorithm **Polynomial inversion**. ▪

We now define further problems:

$\mathrm{DETERMINANT}_n$:	$\det A$ for $A \in F^{n\times n}$,
$\mathrm{CHARPOLY}_n$:	$\chi(A)$ for $A \in F^{n\times n}$,
$\mathrm{MATPOWERS}_n$:	$A^2, A^3, \ldots, A^n$ for $A \in F^{n\times n}$,
$\mathrm{NONSINGEQ}_n$:	x with $Ax = b$ for $A \in F^{n\times n}$ invertible and $b \in F^n$.

We write "$f \leq_F g + h$" for a reduction computing f that makes oracle calls both to g and h, and observe that $f \leq_F g + h$ and $g \leq_F h$ imply that $f \leq_F h$. We say that f is *equivalent* to g ($f \equiv g$) if and only if $f \leq g$ and $g \leq f$.

THEOREM 13.5

DETERMINANT, CHARPOLY, ITMATPROD, MATPOWERS, MATINV, *and* NONSINGEQ *are equivalent.*

PROOF

We will exhibit a complete circle of six reductions. The claim then follows from the transitivity of $\leq_F$ (Theorem 13.10 (1)).

1. DETERMINANT $\leq_F$ CHARPOLY: the determinant is the constant term of the characteristic polynomial, up to the sign.
2. CHARPOLY $\leq_F$ ITMATPROD: The algorithm **Characteristic polynomial** of Section 13.4 has shown that
$$\begin{aligned}\text{CHARPOLY} \leq_F\ & \text{MATPOWERS} \\ & + \text{ITPOLYPROD} + \text{POLYINV}.\end{aligned}$$
Together with Theorem 13.4 (3) and the trivial MATPOWERS $\leq_F$ ITMATPROD, the required reduction follows. This is by far the most challenging reduction in this proof.
3. ITMATPROD $\leq_F$ MATPOWERS: Given $A_1, \ldots, A_n \in F^{n\times n}$, we may consider
$$B = \left[\begin{array}{cccc} I & A_1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & A_n \\ 0 & & & I \end{array}\right] \in (F^{n\times n})^{(n+1)\times(n+1)}$$
as an $(n^2+n) \times (n^2+n)$-matrix. Then
$$B^n = \left[\begin{array}{cccc} I & * & \cdots & A_1 \cdots A_n \\ & I & \ddots & * \\ & & \ddots & * \\ 0 & & & I \end{array}\right]$$
and $\mathrm{ITMATPROD}_n(A_1, \ldots, A_n)$ can be read off $\mathrm{MATPOWERS}_{n^2+n}(B)$.

4. MATPOWERS $\leq_F$ MATINV: Given $A \in F^{n\times n}$, we consider

$$B = \tau_{n+1}(1 - Ax) \in (F^{n\times n})^{(n+1)\times(n+1)}$$

as an $(n^2+n) \times (n^2+n)$-matrix. By Lemma 13.2,

$$\tau_{n+1}((Ax)^{n+1}) = 0,$$

$$\begin{aligned} B^{-1} &= (1 - \tau_{n+1}(Ax))^{-1} \\ &= \sum_{0\leq k\leq n} \tau_{n+1}(Ax)^k \\ &= \tau_{n+1}\Big(\sum_{0\leq k\leq n} A^k x^k\Big) \\ &= \boxed{\begin{matrix} I & A & \cdots & A^n \\ & I & \ddots & \vdots \\ & & \ddots & A \\ 0 & & & I \end{matrix}}. \end{aligned}$$

Again, $\text{MATPOWERS}_n(A)$ can be read off $\text{MATINV}_{n^2+n}(B)$.

5. MATINV $\leq_F$ NONSINGEQ: Given an invertible $A \in F^{n\times n}$, find $x_1, \ldots, x_n \in F^n$ satisfying $Ax_i = e_i^t$, where $e_i = (0, \ldots, 0, 1, 0, \ldots, 0) \in F^n$ has a 1 in position i, and zeroes elsewhere. Then x_i is the ith column of A^{-1}.
6. NONSINGEQ $\leq_F$ DETERMINANT: follows with Cramer's rule. ∎

We define the "complexity class"

$$DET_F = \{f : f \leq \text{DETERMINANT}\}$$

of problems reducible to the determinant. (This is not an honest complexity class, since it is not defined just by explicit constraints on computational resources like depth and size.) As usual, we call a problem $f \in DET_F$ *complete* if $g \leq f$ for all $g \in DET_F$. Theorem 13.5 can then be stated as follows.

THEOREM 13.6

Let F be a field. DETERMINANT, CHARPOLY, ITMATPROD, MATPOWERS, MATINV, *and* NONSINGEQ *are complete for DET_F.*

EXERCISE 13.6

Prove Lemma 13.2.

13.6 Rank of Matrices

In order to solve general (possibly singular) systems of linear equations, we start with a related problem: the *rank* of matrices, and present a fast parallel algorithm, due to Mulmuley [41], in this section. Before that result, Ibarra et al. [31] had found a very simple algorithm which works over a "real field" F such as $F = \mathbb{Q}$ or $F = \mathbb{R}$. The first shallow circuit for arbitrary F was in Borodin et al. [8]; it has the drawback of requiring random choices in the algorithm. All these methods use depth $O(\log^2 n)$ and size $n^{O(1)}$.

The rank $r = \text{rank}(A)$ of a matrix $A \in F^{n\times n}$ over the ground field F is the maximal size of nonsingular minors of A. If

$$\ker A = \{x \in F^n : Ax = 0\}$$

denotes the nullspace of A, then

$$r + \dim_F \ker A = n. \tag{13.5}$$

If $F \subseteq K$ are fields and $A \in F^{n\times n} \subseteq K^{n\times n}$, then A has the same rank whether considered as a matrix over F or K; in other words, the rank is invariant under field extensions.

Section 13.8 discusses in more detail the model in which the computations of this section are performed (see Example 13.4).

The rank $r = \text{rank} A$ is sometimes called the *geometric rank*. A related quantity is the *algebraic rank* $t = \text{rank}_{\text{alg}} A$ of A, defined by

$$t + \mu_0(A) = n,$$

where $\mu_0(A)$ is the multiplicity of 0 as a root of $\chi(A)$. Note the analogy with (13.5); the "geometric multiplicity" $\dim \ker A$ of 0 in A is replaced by the algebraic multiplicity μ_0. Since $\mu_0 \geq \dim \ker A$, we have $t \leq r$. Using the algorithm **Characteristic Polynomial**, we can compute $\chi(A)$ and t quickly. The idea now is to reduce the computation of $\text{rank} A$ to that of $\text{rank}_{\text{alg}} A$.

What is the relation between rank and algebraic rank? Let $s = n - \operatorname{rank} A$, so that $s = \dim \ker A$, and suppose that $u_1, \ldots, u_n$ is a basis of F^n, with $u_1, \ldots, u_s$ being a basis of $\ker A$. Such a basis always exists, and in this basis A has the form

$$A = \left[\begin{array}{c|c} 0 & * \\ \hline 0 & B \end{array}\right] \begin{array}{l} 1 \\ \vdots \\ s \\ s+1 \\ \vdots \\ n \end{array} \tag{13.6}$$

Clearly $\chi(A) = x^s \cdot \chi(B)$, and thus

$$\operatorname{rank} A = r = n - s \le n - t = \operatorname{rank}_{\mathrm{alg}} A.$$

We can calculate $\chi(A)$ and $\operatorname{rank}_{\mathrm{alg}} A$ fast in parallel (Section 13.4), and would like to use this to compute $\operatorname{rank} A$. Here is a sufficient criterion.

LEMMA 13.3
If $\operatorname{rank} A = \operatorname{rank} A^2$, *then* $\operatorname{rank} A = \operatorname{rank}_{\mathrm{alg}} A$.

PROOF
We clearly have $\ker A \subseteq \ker A^2$, so that the hypothesis implies that $\ker A = \ker A^2$. In (13.6), it is sufficient to have B nonsingular, since then $\chi(B)$ does not have 0 as a root, and t is the multiplicity of 0 as a root of $\chi(A)$, and hence $\operatorname{rank} A = \operatorname{rank}_{\mathrm{alg}} A$.

So suppose $a = (a_{s+1}, \ldots, a_n) \in F^{n-s}$ with $Ba = 0$, and let $\bar{a} = (0, \ldots, 0, a_{s+1}, \ldots, a_n) \in F^n$. Then $A\bar{a} \in \ker A$, and hence $A^2\bar{a} = 0$. Thus $\bar{a} \in \ker A^2 = \ker A$. The special form of $\bar{a}$ implies that $\bar{a} = 0$, hence $a = 0$, and indeed B is nonsingular. ∎

We now try to get into this favorable case by constructing from A a matrix B with $\operatorname{rank} B = \operatorname{rank} B^2$, and such that $\operatorname{rank} A$ is easy to compute from $\operatorname{rank} B$. We first replace $A \in F^{n\times n}$ by

$$A' = \boxed{\begin{array}{cc} 0 & A \\ A^t & 0 \end{array}} \in F^{2n\times 2n}.$$

Then $\operatorname{rank} A = \frac{1}{2}\operatorname{rank} A'$, and A' is symmetric. Writing A for A' now, we may assume that A is symmetric.

Let y be an indeterminate over F, $F(y)$ the field of rational functions in y over F,

$$Y = \operatorname{diag}(1, y, \ldots, y^{n-1}) = \begin{bmatrix} 1 & & & 0 \\ & y & & \\ & & \ddots & \\ 0 & & & y^{n-1} \end{bmatrix} \in F(y)^{n\times n},$$

and $B = YA$. Since Y is nonsingular, we have $\operatorname{rank} B = \operatorname{rank} A$. (We use the fact that $\operatorname{rank} A$ is invariant under the field extension $F \subseteq F(y)$.)

LEMMA 13.4
$\operatorname{rank} B^2 = \operatorname{rank} B$.

PROOF
It is sufficient to show that

$$\operatorname{rank} AYA = \operatorname{rank} A,$$

since Y is nonsingular and

$$\operatorname{rank} B^2 = \operatorname{rank} YAYA = \operatorname{rank} AYA = \operatorname{rank} A = \operatorname{rank} YA = \operatorname{rank} B.$$

Since $\operatorname{rank} AYA \le \operatorname{rank} A$, we only have to show $\operatorname{rank} AYA \ge \operatorname{rank} A$. We prove this by showing $\ker AYA \subseteq \ker A$.

So let $u \in F(y)^n$ with $AYAu = 0$. We want to show that $Au = 0$. After multiplying up the denominators (in $F[y]$) of the coordinates of u, we may assume that $u \in F[y]^n$. Set $v = Au \in F[y]^n$. Let z be a new indeterminate over F, $w = v(z) = Au(z) \in F[z]^n$, and

$$s = \sum_{1\le i\le n} w_i v_i y^{i-1} = w^t Y v = v(z)^t Y v = u(z)^t A^t Y A u = 0,$$

where we have used that A is symmetric: $A^t = A$. Suppose that $v \ne 0$. Let $m_i = \deg v_i$ (with $\deg 0 = -\infty$), $m = \max\{m_i \colon 1 \le i \le n\}$, $k = \max\{i \colon 1 \le i \le n, m_i = m\}$. Terms containing z^{m_k} only occur in summands of $\sum w_i v_i y^{i-1} \in F[y, z]$ with $\deg w_i = m = m_k$, and when $i < k$, then such a summand has degree less than $m_k + k - 1$ in y. Therefore $z^{m_k} y^{m_k} y^{k-1}$ has nonzero coefficient in the above sum. Thus $s \ne 0$. This contradiction shows that indeed $v = 0$, and thus $\ker A = \ker AYA$. ∎

ALGORITHM 13.3
Matrix rank
Input: A symmetric matrix $A \in F^{n\times n}$.
Output: rank$A$.

1. Compute $B = YA$, where Y is defined above, using an indeterminate y.
2. Compute $\chi(B) = \det(xI - B)$.
3. Return $r = \mathrm{rank}_{\mathrm{alg}} B$.

THEOREM 13.7
Over any field F, MATRANK $\leq$ ITMATPROD.

PROOF
The algorithm **Matrix rank** reduces the rank of $n \times n$-matrices to the computation of the characteristic polynomial of matrices in $F[y]^{n\times n}$, with each entry of degree less than n. Each coefficient of such a characteristic polynomial has degree less than n^2. Algorithm **Characteristic polynomial** in Section 13.4 reduces this in turn to the iterated product of $n \times n$-matrices over $F[y]$, again with degree in y less than n.

Thus suppose we want to compute $E = D_1 \cdots D_n$, with $D_1, \ldots, D_n \in F[y]^{n\times n}$. Write $D_i = \sum_{0\leq j<n} D_{ij}y^j$, with all $D_{ij} \in F^{n\times n}$. Then the entries of E have degree less than n^2 and can be read off the iterated product of all

$$\phi_{n^2}(D_i) \in (F^{n\times n})^{n^2\times n^2} \cong F^{n^3\times n^3},$$

by Lemma 13.2. Overall, we have reduced MATRANK to ITMATPROD. ∎

Note that just saying "a product of n $n \times n$-matrices, each entry a polynomial of degree less than n" would only yield depth $O(\log^3 n)$.

COROLLARY 13.1
Let F be a field. The rank of $n \times n$-matrices can be computed in depth $O(\log^2 n)$ and size $n^{O(1)}$.

13.7 Linear Algebra Classes

In this section, we show that most elementary problems from linear algebra are complete for one of two complexity classes: DET_F or $RANK_F$.

We define the "complexity class"

$$RANK_F = \{f : f \leq \text{MATRANK}\},$$

and the further problems:

BASIS:	compute an n-bit vector marking a maximal set of linearly independent columns of $A \in F^{n\times n}$ (i.e., a basis for the column space of A),
SOLVABILITY:	compute the bit $(\exists x \in F^n\ Ax = b)$,
MAXMINOR:	mark rows and columns of $A \in F^{n\times n}$ forming a maximal nonsingular submatrix.

These problems are in general not functions, but relations with several possible answers. This issue is discussed in greater detail in Section 13.8.

THEOREM 13.8
Let F be a field. Then

1. $RANK_F \subseteq DET_F$.
2. MATRANK, MAXMINOR, BASIS, *and* SOLVABILITY *are complete for $RANK_F$.*

PROOF
(1) follows from MATRANK $\leq$ ITMATPROD $\in DET_F$.

(2) We give a circle of four reductions:

1. MATRANK $\leq$ MAXMINOR: The rank equals the number of columns of a maximal nonsingular minor.
2. MAXMINOR $\leq$ BASIS: Given $A \in F^{n\times n}$, mark a basis $A_{i_1}, \ldots, A_{i_r}$ of columns of A for the column space of A. Append $n - r$ zero columns to these to get $B \in F^{n\times n}$. Mark a basis for the row space of B (obtained from a column basis for B^t). Then the marked columns and rows of A form a maximal nonsingular minor.
3. BASIS $\leq$ SOLVABILITY: Suppose we are given $A \in F^{n\times n}$, with columns $A_1, \ldots, A_n \in F^n$. For all i, $1 \leq i \leq n$, check whether the system

$$\sum_{1\leq j<i} A_j x_j = A_i$$

of n linear equations in $i-1$ indeterminates has a solution $(x_1,\ldots,x_{i-1}) \in F^{i-1}$. (To bring this into the required square format, append $n-i+1$ zero columns, and check for $x \in F^n$.) A basis is formed by those A_i for which no solution exists.

4. SOLVABILITY $\leq$ MATRANK: $Ax = b$ has a solution if and only if
$$\operatorname{rank} A = \operatorname{rank}(A|b),$$
where $A|b \in F^{n\times(n+1)}$ is A with column b appended. ∎

We define further problems:

INDEPENDENCE:	on input $x_1,\ldots,x_i \in F^n$, decide whether they are linearly independent,
SINGULAR:	decide whether $A \in F^{n\times n}$ is singular,
EQ:	given $A \in F^{n\times n}$ and $b \in F^n$, compute the bit $c = (\exists y \in F^n \ Ay = b)$, and if $c = $ true, compute $x \in F^n$ with $Ax = b$.
NULLSPACE:	compute a basis for the nullspace $\{b \in F^n : Ab = 0\}$ of $A \in F^{n\times n}$.

THEOREM 13.9

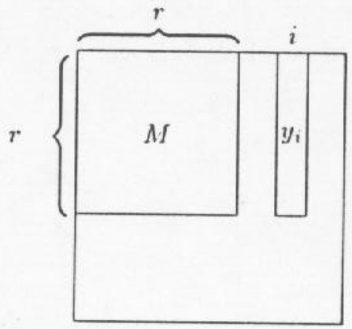
Let F be a field. EQ *and* NULLSPACE *are complete for DET_F.*

PROOF

We show
$$\text{EQ} \leq \text{NULLSPACE} \leq \text{NONSINGEQ} \leq \text{EQ}.$$

1. EQ $\leq$ NULLSPACE: Let $A \in F^{n\times n}$, $b \in F^n$. Then
$$\forall x \in F^n \quad \left(Ax = b \iff \boxed{\begin{array}{c|c} A & b \end{array}} \cdot \boxed{\begin{array}{c} x \\ y \end{array}} = 0 \text{ with } y = -1\right).$$
Determine a basis $z_1,\ldots,z_k \in F^{n+1}$ of the nullspace of
$$\boxed{\begin{array}{cc} A & b \\ 0\ldots0 & 0 \end{array}} \in F^{(n+1)\times(n+1)}.$$
Then return $c = $ true if and only if $z_{i,n+1} \neq 0$ for some i, $1 \leq i \leq k$. If $c = $ true, let i be the smallest index such that $z_{i,n+1} \neq 0$, and return also
$$x = \frac{-1}{z_{i,n+1}}(z_1,\ldots,z_n).$$

2. NULLSPACE $\leq$ NONSINGEQ: Given $A \in F^{n\times n}$, find a maximal nonsingular minor M of A, using MAXMINOR $\leq$ NONSINGEQ. For simplicity, we may assume that M is the upper left $r \times r$-submatrix, where $r = \operatorname{rank} A$. For all i, $r < i \leq n$, solve the nonsingular system of linear equations
$$Mx_i = y_i,$$
where $y_i \in F^r$ consists of the top r entries of the ith column of A:



For $r < i \leq n$, let
$$z_i = (x_i, 0, \ldots, 0, \underset{\substack{\uparrow \\ i}}{-1}, 0, \ldots, 0) \in F^n.$$

We now claim that $z_{r+1},\ldots,z_n$ form a basis of the nullspace $\ker A$ of A. Let $r < i \leq n$. We first show that $z_i \in \ker A$. For $1 \leq j \leq r$, the jth entry of Az_i is
$$\sum_{1\leq k\leq n} A_{jk} z_{ik} = \sum_{1\leq k \leq r} A_{jk} x_{ik} - A_{ji} = 0.$$

For $r < j \leq n$, the jth row A_{j*} of A is a linear combination of the first rows $A_{1*},\ldots,A_{r*}$, since M is a maximal nonsingular minor. Therefore again $(Az_i)_j = A_{j*}z_i = 0$. Combining these, we have $Az_i = 0$.

On the other hand, $\dim_F \ker A = n - r$ and $z_{r+1},\ldots,z_n \in \ker A$ are linearly independent, because z_i has a -1 in position i, and all the other z_k's have a zero there. Therefore these form a basis of $\ker A$.

3. NONSINGEQ $\leq$ EQ is trivial. ∎

The following problems are all unsolved.

OPEN QUESTION 13.1

1. *Is* $RANK_F \not\equiv DET_F$?
2. *Is* INDEPENDENCE *complete for* $RANK_F$?
3. *Is* INDEPENDENCE $\leq$ SINGULAR?

13.8 Arithmetic Boolean Circuits

In the previous sections, we have derived (exact) algorithms using parallel time $O(\log^2 n)$ for the basic problems of linear algebra. In this section, we describe the elements of a theory of parallel algebraic computation, and where the above results fit into that general framework.

We start with *arithmetic Boolean circuits*, a generalization of our arithmetic circuits necessary to deal with decision problems, which we have already used for the rank of matrices. Then we define some *parallel complexity classes*, in analogy with well-studied Boolean complexity classes, and finally formalize *reductions* between two problems. In Sections 13.5 and 13.7, many problems from linear algebra turned out to be equivalent either to the determinant or to the rank problem, so that any depth improvement for one of them would automatically improve the depth for all of them.

How can we solve a general system $Ax = b$ of linear equations, where A may be singular? Even for a single equation $ax = b$, with $a, b \in F$, all we can do with an arithmetic circuit is to return $x = b/a$. However, we would also like to output the information "no solution" if $a = 0$ and $b \neq 0$. So we now extend the model to allow such tests.

First recall that a *Boolean circuit* is a labelled directed acyclic graph, similar to an arithmetic circuit. The difference is that the values manipulated are not from an algebraic domain, but the two Boolean values T (for "true", or 1) and F (for "false", or 0). Accordingly, the operations are the Boolean $\neg$ (negation "not"), $\wedge$ (conjunction "and"), and $\vee$ (disjunction "or"). Boolean circuits are a model of the electronic circuits, the innards of digital computers.

We now define an *arithmetic Boolean circuit* (over a ring F) to be a labelled directed acyclic graph, where both arithmetic and Boolean labels are allowed. Thus we have arithmetic inputs and constants from F, and Boolean inputs and constants from the Boolean universe $\mathbb{B} = \{T, F\}$, and the seven operations $+$, $-$, $*$, $/$, $\neg$, $\wedge$, $\vee$. Each gate has a *type*—either arithmetic or

Boolean—and the appropriate number of inputs with the right type. As an example, a $\wedge$-gate has two inputs, both from a gate with type "Boolean".

Two further gates provide the interface between the arithmetic and the Boolean parts: test gates and selection gates. A *test gate* "$x \overset{?}{\neq} 0$" has an arithmetic input x and a Boolean output y:

$$y = \begin{cases} T & \text{if } x \neq 0, \\ F & \text{if } x = 0. \end{cases}$$

A *selection gate* has two arithmetic inputs x_1 and x_2, a Boolean input y, and an arithmetic output z:

$$z = \begin{cases} x_1 & \text{if } y = T, \\ x_2 & \text{if } y = F. \end{cases}$$

For simplicity, we assume in the sequel that the ground domain F is a field. Since we now have zero-tests, we insist that every division in α is by a nonzero field element, for any specific input supplied for the variables.

Figure 13.5 shows an arithmetic Boolean circuit α with two inputs x_1 and x_2 (at gates 2 and 3), the arithmetic constant 1 at gate 1, and one

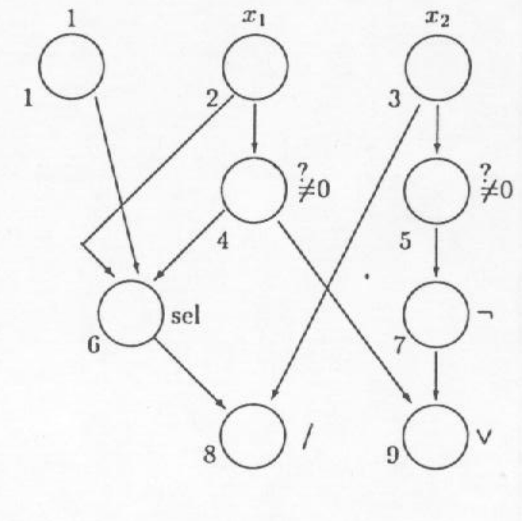


FIGURE 13.5
An arithmetic Boolean circuit for a linear equation $x_1 t = x_2$.

arithmetic plus one Boolean output, at gates 8 and 9. The value of gate 9 is T if $z = (x_1 \neq 0 \vee x_2 = 0)$ is true, and the value y of gate 8 is

$$y = \begin{cases} x_2/x_1 & \text{if } x_1 \neq 0, \\ x_2 & \text{if } x_1 = 0. \end{cases}$$

Thus α solves the linear equation $x_1 t = x_2$ in a satisfactory way: the output z is T if a solution exists, and F otherwise, and if $z = \mathrm{T}$, then y is indeed a solution.

The arithmetic Boolean circuit α splits the input space $S = F^2$ into three regions: $S_1 = F^2 \setminus (\{0\} \times F) = \{(x_1, x_2) \in F^2 : x_1 \neq 0\}$, $S_2 = (\{0\} \times F) \setminus \{(0,0)\}$, and $S_3 = \{(0,0)\}$. These form a *partition* of S: $S = S_1 \cup S_2 \cup S_3$, and $S_i \cap S_j = \emptyset$ if $i \neq j$. In each region, two functions are computed:

$$\begin{array}{lll} y = x_2/x_1 & \text{and } z = \mathrm{T} & \text{on } S_1, \\ y = x_2 & \text{and } z = \mathrm{F} & \text{on } S_2, \\ y = 0 & \text{and } z = \mathrm{T} & \text{on } S_3. \end{array}$$

We will call the collection f_α of all these

$$f_\alpha = \big((S_1, x_2/x_1, \mathrm{T}), (S_2, x_2, \mathrm{F}), (S_3, 0, \mathrm{T})\big)$$

a *piecewise rational function.* The problem of determining whether a linear equation $x_1 t = x_2$ has a solution t, and computing a solution if one exists, corresponds to "the" piecewise rational function

$$g = \big((S_1, x_2/x_1, \mathrm{T}), (S_2, \mathrm{F}), (S_3, w, \mathrm{T})\big),$$

where w is any element of F. Thus in fact g consists of not one, but many piecewise rational functions, one for each $w \in F$.

We say that α *computes* g since for any input $(x_1, x_2) \in F^2$, one of the piecewise rational functions making up g can be read off the gates of α. If $(x_1, x_2) \in S_1$, the outputs y and z contain the unique correct answer to g. If $(x_1, x_2) \in S_2$, z has the correct value, and y is irrelevant. If $(x_1, x_2) \in S_3$, the answer $(y, z) = (0, \mathrm{T})$ is among the correct ones for g (with $w = 0$); the value $y = 0$ was chosen arbitrarily.

In general, we call any of the objects computed by an arithmetic Boolean circuit α a piecewise rational function, and each *computational problem* is a family $g = (g_n)_{n \in \mathbb{N}}$ of collections g_n of such piecewise rational functions. A family $\alpha = (\alpha_n)_{n \in \mathbb{N}}$ of arithmetic Boolean circuits *computes* g if for each n, α_n computes one of the piecewise rational functions in g_n.

EXAMPLE 13.4
The problem of computing the rank of matrices would be formalized as follows: MATRANK = $(\mathrm{MATRANK}_n)_{n \in \mathbb{N}}$ with

$$\mathrm{MATRANK}_n(A) = \underbrace{\mathrm{T} \cdots \mathrm{T}}_{r}\underbrace{\mathrm{F} \cdots \mathrm{F}}_{n-r} \text{ for } A \in F^{n \times n} \text{ and } r = \operatorname{rank} A.$$

EXAMPLE 13.5
Let us see how the computational problem MAXMINOR = $(\mathrm{MAXMINOR}_n)_{n \in \mathbb{N}}$ of determining a maximal nonsingular minor of matrices, discussed in Section 13.7, fits into this framework. Intuitively, for each $n \in N$ and matrix $A \in F^{n \times n}$ one should find some sets $I, J \subseteq \{1, \ldots, n\}$ such that the minor (= submatrix) of A with row indices from I and column indices from J is a maximal nonsingular minor; i.e., it is a nonsingular square matrix, and A has no nonsingular minors of larger size. Formally, $\mathrm{MAXMINOR}_n$ is the set of all functions

$$g : F^{n \times n} \longrightarrow \mathbb{B}^N \times \mathbb{B}^N$$

(with $N = \{1, \ldots, n\}$) where $g(A)$ are the row and column indices of a maximal nonsingular minor of A, for all $A \in F^{n \times n}$. We have to agree on some coding of $g(A)$ over $\mathbb{B}$: one possibility is a string $(y_1, \ldots, y_n, z_1, \ldots, z_n) \in \mathbb{B}^{2n}$ with $i \in I \iff y_i = \mathrm{T}$, and $j \in J \iff z_j = \mathrm{T}$. As a specific example, consider

$$A = \begin{array}{|ccc|} \hline 1 & 1 & 2 \\ 1 & 1 & -2 \\ 2 & 2 & 0 \\ \hline \end{array} \in \mathbb{Q}^{3 \times 3}.$$

The last row of A is the sum of the first two rows. In our formalism, the fact that $\begin{array}{|cc|} \hline 1 & 2 \\ 1 & -2 \\ \hline \end{array}$ is a maximal nonsingular minor of A is expressed as follows:

$$(A, \mathrm{TTFFTT}) \in \mathrm{MAXMINOR}_3.$$

An arithmetic Boolean circuit solving MAXMINOR would compute, on any input $A \in F^{n \times n}$, some output $f_\alpha(A) = (\eta_1, \ldots, \eta_n, \zeta_1, \ldots, \zeta_n) \in \mathbb{B}^{2n}$, and the requirement is that indeed $f_\alpha(A)$ describe a maximal nonsingular minor of A. Natural algorithms, such as the ones to be discussed in Section 13.7 for this and similar problems, will compute a "natural" candidate among the $g(A)$'s, e.g., the "lexicographically first maximal nonsingular minor".

A powerful tool in the theory of computation is to collect problems of the "same" cost in *complexity classes*. Before we discuss these within our framework, we first need two rather technical notions; we do not give detailed definitions here. The first is *uniformity* of arithmetic circuits. Uniformity of Boolean circuits is discussed in Ruzzo [47] and Cook [12]. Among the various notions we choose P-uniform families of arithmetic Boolean circuits $\alpha = (\alpha_n)_{n\in\mathbb{N}}$ as our standard: there must exist a Turing machine which, on input n in unary, generates in polynomial time a description of the labels and connections of α_n. A further complication for us is that we have to define *uniformity of field constants*. For this, we require a polynomial-time Turing machine that, on input n, produces a polynomial-size arithmetic circuit β_n, whose only inputs or constants are the inputs to α_n and 1, and whose outputs are the constants used in α_n. (Often, 1 will be the only constant used.)

Furthermore, we have to use the *degree* $\deg g_n$ of a piecewise rational function g_n. If $g_n = (F^n, h_n)$ with $h_n \in F[x_1, \ldots, x_n]$ consists of a single polynomial, then its degree is simply the degree of h_n. In the general case, we require a deep generalization of this notion from algebraic geometry; for discussions, see Strassen [50] and Heintz [30].

Here are now some complexity classes of importance to parallel arithmetic computation.

$$
\begin{aligned}
P_F &= \{g = (g_n)_{n\in\mathbb{N}}\colon\ \exists\ P\text{-uniform } \alpha = (\alpha_n)_{n\in\mathbb{N}} \text{ computing } g \text{ with} \\
&\qquad S(\alpha_n) = n^{O(1)}, \text{and } \deg g_n = n^{O(1)}\}, \\
NC_F^k &= \{g = (g_n)_{n\in\mathbb{N}}\colon\ \exists\ P\text{-uniform } \alpha = (\alpha_n)_{n\in\mathbb{N}} \text{ computing } g \\
&\qquad \text{with } S(\alpha_n) = n^{O(1)} \text{and } D(\alpha_n) = O(\log^k n), \\
&\qquad \text{and } \deg g_n = n^{O(1)}\}, \text{ for any } k \in \mathbb{N}, \\
NC_F &= \textstyle\bigcup_{k\in\mathbb{N}} NC_F^k.
\end{aligned}
$$

Here, g stands for a computational problem, and α for a family of arithmetic Boolean circuits.

The arithmetic complexity classes defined above are analogues of the Boolean classes NC^k, defined as the set of Boolean functions computed by *log-space uniform* Boolean circuits of depth $O(\log^k n)$ and size $n^{O(1)}$, for input size n. Also, $NC = \bigcup_{k\in\mathbb{N}} NC^k$, and P is defined by polynomial size $n^{O(1)}$ only. Nick Pippenger [44] introduced these Boolean classes NC^k—an acronym for "Nick's class", coined at Toronto where he was then working. $NC^k(P\text{-uniform})$ and $NC(P\text{-uniform})$ are obtained by the more generous notion of P-uniformity, which we use for our arithmetic classes. Then $NC \subseteq NC(P\text{-uniform})$, and Cook [12] conjectures that inequality holds. For any field F, we have $NC(P\text{-uniform}) \subseteq NC_F$, and similarly for the other classes.

In Boolean circuit complexity, another model of importance is the *unbounded fan-in Boolean circuit*, where $\vee$- and $\wedge$-gates may have any number of inputs. Restricting the depth to $O(\log^k n)$, for some $k \in \mathbb{N}$, the size to $n^{O(1)}$, and requiring uniformity, one obtains the complexity class AC^k. This acronym stands for "alternating class" and comes from the fact that, for $k \geq 1$, it can also be characterized by *alternating Turing machines* using space $O(\log n)$ and alternation depth $O(\log^k n)$. We clearly have $NC^k \subseteq AC^k \subseteq NC^{k+1}$. One of the few separation theorems in Boolean complexity theory is the breakthrough result $AC^0 \subsetneq NC^1$ of Furst et al. [19] and Ajtai [1]; the parity function is in NC_F^1, and they show that it is not in AC_F^0.

If we allow unbounded fan-in $\wedge$-gates, but only $\vee$-gates with fan-in two, we obtain the classes SAC^k and SAC (for "semi-AC"). Borodin et al. [7] show that one obtains the same classes with unbounded fan-in $\vee$ and bounded $\wedge$.

We take the latter model as our template for the arithmetic case. A *semi-unbounded fan-in arithmetic Boolean circuit* α is like an ordinary arithmetic Boolean circuit, except that the $+$ and $\vee$-gates are allowed to have arbitrary fan-in. The depth of a gate with fan-in k is 1, and its size is $k - 1$. Then, as usual, $D(\alpha)$ is the depth of a deepest path in α, and $S(\alpha)$ the sum of the sizes of all gates in α. These circuits lead to the following complexity classes.

$$
\begin{aligned}
SAC_F^k &= \{g = (g_n)_{n\in\mathbb{N}}\colon\ \exists\alpha = (\alpha_n)_{n\in\mathbb{N}} \text{ of semi-unbounded fan-in} \\
&\qquad \text{computing } g \text{ with } S(\alpha_n) = n^{O(1)}, D(\alpha_n) = \\
&\qquad O(\log^k n), \text{ and } \deg g_n = n^{O(1)}\}, \text{ for any } k \in \mathbb{N}, \\
SAC_F &= \textstyle\bigcup_{k\in\mathbb{N}} SAC_F^k.
\end{aligned}
$$

The circuit families have to be P-uniform. We clearly have the following *hierarchy* of complexity classes:

$$
\begin{aligned}
&NC_F^0 \subseteq SAC_F^0 \subseteq NC_F^1 \subseteq \cdots \\
&\qquad\qquad NC_F^k \subseteq SAC_F^k \subseteq NC_F^{k+1} \subseteq \cdots NC_F = SAC_F \subseteq P_F.
\end{aligned}
$$

Whenever one has such a hierarchy, one of the first (and usually most difficult) questions is: Does it collapse or not? I.e., is $NC_F^k \subsetneq SAC^k \subsetneq NC_F^{k+1}$ for all k? Is $NC_F \subsetneq P_F$? Throughout this chapter we have worked at the low end of this hierarchy, within NC_F^2.

Only one difference is easy to see: $NC_F^0 \neq SAC^0$. We have $\mathrm{SUM} \in AC_F^0$, and by Theorem 13.1 (3), $\mathrm{SUM}_n = x_1 + \cdots + x_n$ requires depth at least $\log_2 n$ on arithmetic Boolean circuits with fan-in two, and so is not in the trivial class NC_F^0.

We define a *reduction* between two problems $f = (f_n)_{n\in\mathbb{N}}$ and $g = (g_n)_{n\in\mathbb{N}}$ to be a family $(\alpha_n)_{n\in\mathbb{N}}$ of arithmetic Boolean circuits α_n of constant depth and polynomial size $n^{O(1)}$, with α_n computing f_n. As gates we allow in α_n the usual arithmetic and Boolean gates, as above, plus gates computing some g_j. Furthermore, both the $+$-gates and the $\vee$-gates may have an arbitrary number of inputs. If such an "arbitrary fan-in"-gate or an "oracle gate" for g_j has k inputs, we define its size to be $\max\{j, k\} - 1$, and its depth to be 1. (This agrees with the previous definitions for the "binary" $+$, with only two inputs; the size comes from the lower bound of Exercise 13.1.) If such a reduction exists, we write $f \le_F g$: f is *reducible* to g. Furthermore, $f \equiv_F g$ means that $f \le_F g$ and $g \le_F f$; then f and g are *equivalent*. One ambiguity is that g_m—as a relation—may not have a single, but many possible answers. We stipulate that for any correct answer to any g_m a correct answer for f must result.

EXAMPLE 13.6
PROD $\le_F$ DETERMINANT. Consider the circuit α_n with input $x_1, \ldots, x_n$ and the constant 0, which produces an $n \times n$-matrix A with $x_1, \ldots, x_n$ on the diagonal and zeroes elsewhere, and then calls DETERMINANT$_n$ with input A. The output is PROD$_n$. The depth of this reduction is 1, and the size $n^2 - 1$.

Intuitively, "$f \le_F g$" means that f is not harder than g. If we find a good algorithm for g, then we automatically obtain a good algorithm for f. On the other hand, a lower bound on the complexity of f translates into one for the complexity of g.

THEOREM 13.10
Let F be an integral domain.

1. *Reducibility is a partial order.*
2. *Equivalence is an equivalence relation.*
3. *If $f \le_F g$ and g can be computed in depth $O(\log^k n)$, for some $k \ge 1$, and size $n^{O(1)}$, then f can be computed at the same (asymptotic) cost.*

The theorem is proven by taking a circuit family $\alpha = (\alpha_n)_{n \in \mathbb{N}}$ for g, plugging it into the oracle nodes calling g_j in the reduction, and thus obtaining a circuit family for f; we forego the details.

If C is one of our complexity classes (or any set of problems), we say that a problem f is *complete for* C if

1. $f \in C$,
2. $\forall g \in C \;\; g \le f$.

Thus a complete problem is hardest within its complexity class.

An important result is that the iterated product of 3×3-matrices (see Section 13.3) is complete for NC^1_F (Ben-Or & Cleve [4]).

The two main algorithmic results in this chapter, namely the computations for determinant and rank of matrices, can be summarized as follows.

THEOREM 13.11
Let F be a field. Then

$$RANK_F \subseteq DET_F \subseteq SAC^1_F \subseteq NC^2_F.$$

PROOF
The first inclusion is in Theorem 13.8 (1). For the second one, we check that MATPROD $\in SAC^0_F$ and ITMATPROD $\in SAC^1_F$. The claim then follows from Algorithm **Characteristic Polynomial**, using Theorem 13.4. ∎

The following problems are unsolved.

OPEN QUESTION 13.2

1. *Is $DET_F = SAC^1_F$?*
2. *Is $SAC^1_F \neq NC^2_F$?*

REMARK 13.1
Our arithmetic circuits are the arithmetic analogues of Boolean circuits, and one reason for choosing them for this chapter is their conceptual simplicity. Another highly popular model of parallel Boolean computation is the PRAM. Its analogue, the arithmetic PRAM, *has arithmetic values in its memory cells, and each processor can perform an arithmetic operation on two of those values in one time step. (Similarly, one defines* arithmetic Boolean PRAMs, *with the same instruction set as our arithmetic Boolean circuits.) There are various possibilities to regulate the read/write conflicts on Boolean PRAMs. Similarly, the arithmetic PRAMs come in several flavors; we do not discuss this here.*

For a comparison of the two models, it is easiest to use levelled arithmetic circuits, *where each gate has an integer associated to it, its* level, *and inputs to a gate come only from previous levels. The* width *of a levelled arithmetic circuit is the maximum number of arithmetic gates at each level. Then an arithmetic circuit can be simulated by an arithmetic PRAM, and, for a given input size n, an arithmetic PRAM by an arithmetic circuit, with circuit depth corresponding to PRAM time, circuit* *size corresponding to PRAM memory, and circuit width corresponding to the number of PRAM processors.*

The simulation question is not so clear, however, when we consider circuit families, complexity classes, and the various notions of circuit uniformity.

13.9 Further Results

In this section, we give pointers to various results in parallel algebraic complexity theory, without proofs.

Exponentiation turned out to be a very interesting problem for parallel computation: computing a^b in parallel, where $a \in F$ is in the ground domain, and $b \in \mathbb{N}$. This problem and related tasks are used in many algorithms, e.g., factoring integers, primality test, cryptographic protocols, and factoring polynomials over finite fields.

The standard sequential algorithm of "repeated squaring" has linear depth n when b is an n-bit integer. The problem looks unamenable to parallelization, and Kung [39] shows with a degree agrument that indeed over an infinite field no arithmetic circuits with less than the disappointing linear depth are possible. The argument seems to fail over finite fields, where one can use Fermat's Little Theorem ($a^q = a$ for all a in $\mathbb{F}_q$, the field with q elements) to compute the values of large powers at no cost. However, von zur Gathen [24] shows that this is the only obstacle: $D(\alpha) \geq \min\{\log_2 b, \log_2(q-b)\}$ if $1 \leq b < q$ and α computes bth powers in $\mathbb{F}_q$. It was a big surprise when Fich & Tompa [17] proved in a slightly different—yet perfectly reasonable—model that the problem does have a fast parallel solution in an important special case (large finite fields of small characteristic). This leads to the rather shocking observation that for this (and some other) problem arithmetic circuits are *not* the appropriate model of computation (von zur Gathen & Seroussi [29]). The use of "normal bases" in finite fields leads to a natural setting in which the parallel complexity of exponentiation can be determined exactly (von zur Gathen [27]).

Eberly [16] solves various problems (such as the determinant, characteristic polynomial, and solution of systems of linear equations) for *banded $n \times n$-matrices* of bandwidth b in depth $O(\log n \log b)$ and size $n^{O(1)}$; in particular, for constant bandwidth he has optimal depth $O(\log n)$. Kaltofen et al. [35, 36] prove that the *Hermite* and *Smith normal forms* of polynomial matrices can be computed in *probabilistic NC*. These normal forms contain much information about the (geometric) structure of the linear mapping associated with a matrix.

Many problems in *polynomial arithmetic* can be solved in NC_F^2, for a field F, such as the gcd (Borodin et al. [8]), more generally all entries of the Extended Euclidean Scheme of two polynomials, various interpolation problems (rational, Hermite), partial fraction decomposition (for a given factorization of the denominator), Chinese remainder algorithm, and Padé approximation (von zur Gathen [22]). One of the most important unresolved issues is the status of the Boolean analogue:

OPEN QUESTION 13.3
Is the gcd of integers in (Boolean) NC?

It is widely conjectured that $NC \neq P$. However, Valiant et al. [52] showed that for polynomials over a field F we have "$NC_F^2 = P_F$": polynomial families with polynomial degree and polynomial-size arithmetic circuits can be computed on arithmetic circuits of depth $O(\log^2 n)$. Miller et al. [40] give a different version of that result, and Kaltofen [34] extends it to rational functions; see Kaltofen's Chapter 16 in this book.

Reif [46] and Beame et al. [3] showed that (Boolean) problems like division with remainder and iterated product of n-bit integers can be solved in optimal (P-uniform) depth $O(\log n)$ on Boolean circuits. Eberly [16] shows similar results for polynomials over a field. This leads to optimal-depth solutions for the exponentiation problem in finite fields of small characteristic (von zur Gathen & Seroussi [29]), for inversion in finite fields (von zur Gathen [26]), and for the Boolean exponentiation problem of computing $a^b \bmod 2^n$, where $a, b \in \mathbb{N}$ are n-bit integers (von zur Gathen [24]).

A central problem in *computer algebra* is the *factorization of polynomials.* Over finite fields of *small characteristic*, the problem is in NC^2, but in general it is at least as hard as exponentiation (von zur Gathen [21]). Kaltofen [32] has an algorithm for absolute irreducibility. The polynomial-time sequential method over $\mathbb{Q}$ uses "short vectors in $\mathbb{Z}$-modules", which is conjectured to be P-complete; the (possibly "easy") integer gcd problem is reducible to a special case of this (von zur Gathen [21]).

A vast generalization of factoring polynomials is the question of determining the roots of a system of polynomial equations, or, more generally, of deciding first-order sentences in the theory of fields. Ben-Or et al. [4], Davenport & Heintz [15], and Fitchas et al. [18] contain parallel results for algebraically closed fields (like $\mathbb{C}$) and real closed fields (like $\mathbb{R}$).

Strassen [50] introduced the *degree* as an important tool in algebraic complexity theory (see Section 13.8). For an arithmetic circuit α computing a single polynomial f, one has $D(\alpha) \geq \log_2 \deg f$ (Kung [39]); this generalizes to a set of polynomials or rational functions, with the notion of degree from algebraic geometry. Unfortunately, the argument breaks down for our piecewise rational functions, and one can show that the best result here is $D(\alpha) \geq \log_2 \log_2 \deg f$ (see von zur Gathen [23]).

Throughout this chapter, we have implicitly assumed that an almost unbounded (namely, polynomial) number of processors is available. In practice today, however, one can count only on a limited number of processors, and *processor-efficient parallel algorithms* are important, which use small parallel time (say, $(\log n)^{O(1)}$) and not (many) more processors than the best sequential algorithm known; this question is briefly addressed at the end of Section 13.4. Kaltofen [34] has a result in this spirit on the gcd of polynomials, and Kaltofen & Pan [37] achieve this goal (up to logarithmic factors) for the solution of systems of linear equations. Pan & Reif [42, 43] started an interesting line of work for linear algebra; they work in a different "numerical" model where one has access to the individual bits of the real (or rational) inputs.

Let $M(n)$ be the number of PRAM processors required to multiply an $n \times n$ dense matrix in $O(\log n)$ time using $M(n)$ processors. For well-conditioned matrices, Pan and Reif use a Newton iteration to efficiently compute the matrix inverse within high accuracy in $O(\log^2 n)$ time using $M(n)$ PRAM processors. Kaltofen and Pan developed similarly efficient PRAM algorithms for matrix inverse, determinant, and rank over general fields.

Cheriyan and Reif [10] give output sensitive (where the complexity depends on the output) PRAM algorithms for various classes of algebraic problems. They present a randomized algorithm for computing the rank r of an $n \times n$ matrix which runs in parallel time $O(\log n + \log^3 r)$ using $(n^2 + M(r)) \log^{O(1)} n$ processors. They also present randomized algorithms for finding a maximum linearly independent subset of rows that run either in parallel time $O((\log n) \log^2 r)$ using $(n^2 + rM(r)) \log^{O(1)} n$ processors, or in parallel time $O(\log n + \log^2 r)$ using $(n^2 + nM(r)) \log^{O(1)} n$ processors. As an application, they give an output sensitive algorithm for computing greatest common divisors (GCD) of polynomials. Given two polynomials of degree n, the degree r of the polynomial GCD is computed in randomized parallel time $O(\log n + \log^3 r)$ using $(n^2/r + r^2) \log^{O(1)} n$ processors, and the GCD as well as the extended GCD are computed in randomized parallel time $O((\log n) \log^2 r)$ using $(n^2/r + r^3) \log^{O(1)} n$ processors.

A different set of problems concerns *permutation groups*. As an example, the *membership problem* is: given some permutations $\pi_1, \ldots, \pi_k, \sigma$ on n letters, in some standard representation, is σ in the subgroup generated by $\pi_1, \ldots, \pi_k$? A substantial line of research, culminating in Babai et al. [2], shows that the membership problem and related questions are in (Boolean) NC.

Bibliography

[1] M. Ajtai, Σ_1^1-Formulae on Finite Structures. Ann. of Pure and Applied Logic **24** (1983), 1–48.

[2] L. Babai, E.M. Luks, and Á. Seress, Permutation groups in NC. Proc. 19th Ann. ACM Symp. Theory Comput., New York NY, 1987, 409–420.

[3] P.W. Beame, S.A. Cook and H.J. Hoover, Log depth circuits for division and related problems. SIAM. J. Comput. **15** (1986), 994–1003.

[4] M. Ben-Or and R. Cleve, Computing Algebraic Formulas Using a Constant Number of Registers. Proc. 20th Ann. ACM Symp. Theory of Computing, Chicago IL, 1988, 254–257. SIAM. J. Comput., to appear.

[5] M. Ben-Or, D. Kozen, and J. Reif, The Complexity of elementary algebra and geometry. J. Computer System Sciences **32** (1986), 251–264.

[6] S.J. Berkowitz, On computing the determinant in small parallel time using a small number of processors. Information Processing Letters **18** (1984), 147–150.

[7] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa, Two applications of inductive counting for complementation problems. SIAM J. Comput. **18** (1989), 559–578.

[8] A. Borodin, J. von zur Gathen, and J. Hopcroft, Fast parallel matrix and GCD computations. Information and Control **52** (1982), 241–256.

[9] R.W. Brockett and D. Dobkin, On the number of multiplications required for matrix multiplication. SIAM J. Comput. **5** (1976), 624–628.

[10] J. Cheriyan and J. Reif, Parallel and Output Sensitive Algorithms for Combinatorial and Linear Algebra Problems, manuscript, 1992.

[11] A.L. Chistov, Fast parallel calculation of the rank of matrices over a field of arbitrary characteristic. Proc. Int. Conf. Foundations of Computation Theory, Springer Lecture Notes in Computer Science **199**, 1985, 63–69.

[12] S.A. Cook, A taxonomy of problems with fast parallel algorithms. Information and Control **64** (1985), 2–22.

[13] D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progressions. J. Symb. Comp. **9** (1990), 251–280.

[14] L. Csanky, Fast parallel matrix inversion algorithms. SIAM J. Comput. **5** (1976), 618–623.

[15] J. H. Davenport and J. Heintz, Real quantifier elimination is doubly exponential. J. Symb. Comp. **4** (1988).

[16] W. Eberly, Very fast parallel matrix and polynomial arithmetic. SIAM J. Comput. **18** (1989), 955–976.

[17] F. Fich and M. Tompa, The parallel complexity of exponentiating polynomials over finite fields. J. Assoc. Comput. Mach. **53** (1988), 651–667.

[18] N. Fitchas, A. Galligo, and J. Morgenstern, Algorithmes rapides en séquentiel et en parallèle pour l'élimination de quantificateurs en géometrie élémentaire, Séminaire Structures Algébriques Ordonnées, UER de Mathématiques : Université de Paris VII (1987).

[19] M. Furst, J. B. Saxe, and M. Sipser, Parity, Circuits, and the Polynomial Time Hierarchy. Math. Systems Theory **17** (1984), 13–28.

[20] Z. Galil and V. Pan, Parallel evaluation of the determinant and of the inverse of a matrix. Inform. Process. Letters **30** (1989), 41–45.

[21] J. von zur Gathen, Parallel algorithms for algebraic problems. SIAM J. Comput. **13** (1984), 802–824.

[22] J. von zur Gathen, Representations and parallel computations for rational functions. SIAM J. Comput **15** (1986a), 432–452.

[23] J. von zur Gathen, Parallel arithmetic computations: a survey. Proc. 12th Int. Symp. Math. Foundations of Computer Science, Bratislava, Springer Lecture Notes in Computer Science **233**, 1986b, 93–112.

[38] E. Kaltofen and V. Pan, Processor-efficient parallel solution of linear systems II. The positive characteristic and singular cases, Proc. 33rd Annual IEEE Symposium on F.O.C.S. (1992), 714–723.

[39] H.T. Kung, New algorithms and lower bounds for the parallel evaluation of certain rational expressions and recurrences. J. Assoc. Comput. Mach. **23** (1976), 252–261.

[40] G.L. Miller, V. Ramachandran, and E. Kaltofen, Efficient parallel evaluation of straight-line code and arithmetic circuits. SIAM J. Comput. **17** (1988).

[41] K. Mulmuley, A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. Combinatorica **7** (1987), 101–104.

[42] V. Pan and J. Reif, Efficient Parallel Solution of Linear Systems, Proceedings of the 17th Annual ACM Symposium on Theory of Computing, Providence, RI, May 1985, ACM-SIGACT, pp. 143–152.

[43] V. Pan and J. Reif, Fast and Efficient Parallel Solution of Dense Linear Systems. Computers and Mathematics with Applications, Vol. 17, no. 11, pages 1481–1491, 1989.

[44] N. Pippenger, On simultaneous resource bounds. Proc. IEEE 20th Annual Symp. Foundations of Computer Science, 1979, 307–311.

[45] F.P. Preparata and D.V. Sarwate, An improved processor bound in fast matrix inversion. Inform. Process. Letters **7** (1978), 148–150.

[46] J. Reif, Logarithmic depth circuits for algebraic functions. SIAM J. Comput. **15** (1986), 231–242.

[47] W.L. Ruzzo, On uniform circuit complexity. J. Computer System Sciences **22** (1981), 365–383.

[48] V. Strassen, Gaussian elimination is not optimal. Numer. Mathematik **13** (1969), 354–356.

[49] V. Strassen, Berechnung und Programm. I. Acta Inf. **1** (1972), 320–335.

[50] V. Strassen, Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. Numer. Math. **20** (1973), 238–251.

[51] V. Strassen, Algebraic complexity theory. In Handbook of Theoretical Computer Science, ed. by J. van Leewen, Volume A (1990), 633–672. Elsevier, Amsterdam.

[52] L. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, Fast parallel computation of polynomials using few processors. SIAM J. Comput. **12** (1983), 641–644.