13.1 Introduction

A natural model of algebraic computation (for polynomials over a field, say) is the arithmetic circuit, using the arithmetic operations $+,-,\cdot,/$. 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 the 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 $NC^k_F$ and $NC^k_R$ are introduced, analogous to the Boolean complexity classes $NC^k$ and $NC^k_B$.

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.

This material has been used in courses and seminars at University of Toronto (Canada), Université Laval (Québec, Canada), Université 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 $3x_1 - \sqrt{2}x_1$; $3$ and $\sqrt{2}$ are constants belonging to the ground domain $\mathbb{R}$, and $x_1$ and $x_2$ inputs.

![Figure 13.1](image_url)

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 \(\ast\) 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 \(+, -, \ast, /\). 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 \( \sqrt{2} x_1 - 3 x_2 \) 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 \( \alpha \) are of interest here. The depth \( D(\alpha) \) (parallel time) is the number of arithmetic operations on a longest path in the graph of \( \alpha \). The size \( S(\alpha) \) (sequential time) is the total number of arithmetic operations. (It equals the "number of processors" when processors are not re-used.) For the \( \alpha \) 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.

**Theorem 13.1**

Let \( \alpha \) be an arithmetic circuit. Then

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

**Proof**

(1) is trivial. To prove (2), for any gate \( v \) in \( \alpha \), 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 \( 2^{D(\alpha)} - 1 \geq 2^{D(v)} - 1 \geq S(\alpha) \).
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
\[
(2^{D(w_1)} - 1) + (2^{D(w_2)} - 1) + 1 \leq 2^{D(v)-1} + 2^{D(v)-1} - 1 = 2^{D(v)} - 1, \tag{13.1}
\]
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 \( \alpha \) 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 \( \alpha_n \) computing \( f_n \).

**EXERCISE 13.1**
Let \( \alpha \) 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 \( \alpha \) is at least \( \log_2 n \).

b) Prove that the size of \( \alpha \) 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 \( C = AB \in F^{n \times n} \) 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 \textit{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 \(g = \sum_{0 \leq j \leq n} b_j x^j\), their product \(h = \sum_{0 \leq k \leq 2n} c_k x^k = f \cdot g\) has coefficients

\[ c_k = \sum_{0 \leq i, j \leq n} a_i b_j. \]

The obvious circuit:

1. For all \(i, j\) \((0 \leq i, j \leq n)\) compute \(a_i \cdot b_j\).
2. For all \(k\) \((0 \leq k \leq 2n)\) 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 \(O(\log n)\).

\textit{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, \ldots, \(k = \log_2 n\) (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 \textit{congruent modulo} \(g\) if their difference is divisible by \(g\):

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

If \(f \in F[x]\) and \(n \in N\), then \(f\) is \textit{invertible modulo} \(x^{n+1}\) if there exists some \(g \in F[x]\) such that \(g f \equiv 1 \mod x^{n+1}\); 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

\[ fg = 1 - x^4 \equiv 1 \mod x^4, \]

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

The following algorithm generalizes this geometric series.

\textbf{Algorithm 13.1}

\textit{Polynomial inversion}

\textbf{Input:} \(f \in F[x]\) with \(f(0) \in F\) invertible.

\textbf{Output:} \(g \in F[x]\) with \(fg \equiv 1 \mod x^{n+1}\).

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

\textbf{Example 13.1}

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

\[ g = 1 + h + h^2 + h^3 \equiv 1 - 7x + 20x^2 - 17x^3 \mod x^4. \]

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 \(fg \equiv 1 \mod x^4\).
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 \( h^{n+1} \equiv 0 \mod x^{n+1} \). Thus

\[
fg = (1 - h)g(0) \cdot b \sum_{0 \leq i \leq n} h^i = 1 - h^{n+1} \equiv 1 \mod x^{n+1},
\]

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 \( O(n^3) \).

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. [9] 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 \( i \)th 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

\[
A = \begin{pmatrix} 2 & -3 & 0 \\ 1 & 2 & -1 \\ -2 & 1 & 3 \end{pmatrix} \in \mathbb{Q}^{3 \times 3}.
\]
Then
\[ \chi(A) = \det \begin{bmatrix} x - 2 & 3 & 0 \\ -1 & x - 2 & 1 \\ 2 & -1 & x - 3 \end{bmatrix} = -17 + 20x - 7x^2 + x^3, \]
and \( \det A = -17. \)

If \( F \) is a field, \( A = (a_{ij})_{1 \leq i, j \leq n} \in F^{n \times n} \) and \( 1 \leq r \leq n, \) we consider the lower right submatrix
\[ A_r = (a_{ij})_{r \leq i, j \leq n} \in F^{r' \times r'} \]
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
\[ \chi(A) = \det(xI_n - A) = \det(xI_n \cdot (I_n - x^{-1}A)) = \det(xI_n) \cdot \det(I_n - x^{-1}A) = x^n \det(I_n - x^{-1}A_1) = x^n d_1(x^{-1}). \]
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
\[ B^{(r)} = (b_{ij}^{(r)})_{r \leq i, j \leq n} = (I_r - xA_r)^{-1} \]
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 \( b_r^{(r)} = (b_{ij}^{(r)})_{r \leq i, j \leq n} \) the leftmost column, and by \( b_r = b_r^{(r)} \in F(x) \) the top left entry of \( B^{(r)} \). Then by definition of the inverse
\[ (I_r - xA_r) \cdot b_r^{(r)} = (1, 0, \ldots, 0)^t. \]

The determinant of the matrix obtained by substituting \((1, 0, \ldots, 0)^t\) for the leftmost column of \( I_r - xA_r \) equals
\[ \det(I_r - xA_r) = d_{r+1}. \]

Thus expressing the top entry \( b_r \) of \( b_r^{(r)} \) 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
\[ \prod_{1 \leq r \leq n} b_r = d_1^{-1}. \]

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
\[ (1 - xa) \cdot \sum_{0 \leq k \leq n} x^k a^k \equiv 1 \mod x^{n+1}. \tag{13.4} \]

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
\[ B^{(r)} = (I_r - xA_r)^{-1} \equiv \sum_{0 \leq k \leq n} x^k A_r^k \mod x^{n+1}. \]

If we define \( \bar{B}^{(r)} \) as the sum on the right hand side, and \( \bar{b}_r \) as its top left entry, then each \( \bar{b}_{ij}^{(r)} \in F[x] \) is a polynomial in \( x \) of degree at most \( n, \) the denominator \( d_r \) of \( b_r \) is invertible modulo \( x, \) and \( b_r \equiv \bar{b}_r \mod x^{n+1}. \) Thus we have
\[ \prod_{1 \leq r \leq n} \bar{b}_r \equiv d_1^{-1} \mod x^{n+1}. \]

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 $n \in \mathbb{N}$.

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

1. For all $k, r$ (1 ≤ $k, r$ ≤ $n$), compute $A_k^r$ and $\hat{b}_{r, r} = \left( \sum_{0 \leq k \leq n} x^k A_k^r \right)_{rr}$.
2. Compute $b \in F[x]$ with $\deg b \leq n$ and
$$\hat{b} \equiv \prod_{1 \leq r \leq n} \delta_r \mod x^{n+1}.$$

[Then $b \equiv d_1^{-1} \mod x^{n+1}$.]

3. Compute $c \in F[x]$ with $\deg c \leq n$ and $c \equiv b^{-1} \mod x^{n+1}$, 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:

<table>
<thead>
<tr>
<th>Step</th>
<th>Subroutine</th>
<th>Depth</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iterated matrix product</td>
<td>$O(\log^2 n)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>2</td>
<td>iterated polynomial product</td>
<td>$O(\log^2 n)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>3</td>
<td>polynomial inversion</td>
<td>$O(\log^2 n)$</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>4</td>
<td>reverse of $c$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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 $\mathbb{Q}$ 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.

$$A_1 = \begin{bmatrix} 2 & -3 & 0 \\ 1 & 2 & -1 \\ -2 & 1 & 3 \end{bmatrix}$$

$$A_2^2 = \begin{bmatrix} 2 & 1 & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix}, \quad A_3^2 = \begin{bmatrix} 2 & 1 & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix}, \quad A_4^2 = \begin{bmatrix} 2 & 1 & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix}$$

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

$$d_1 \cdot b_1 = \det(I_3 - x A_1) \cdot b_1 \equiv (1 - 7x + 20x^2 - 17x^3) \cdot (1 + 2x + x^2 - 16x^3) \cdot (-17 + 20x - 7x^2 + x^3) \equiv 0 \mod x^4.$$
The standard algorithm used in Section 13.3 shows \( M(n) \leq 2n^3 \). Clearly \( M(n) \geq n^2 \), since \( n^2 \) outputs have to be computed; the best lower bound known today is \( M(n) \geq 2n^2 - 1 \) (Brockett & Dobkin [9]). The best upper bound known today is by Coppersmith & Winograd [13]:

\[
2n^2 - 1 \leq M(n) = O(n^{2.376}).
\]

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 \( l = \lceil \log_2 n \rceil \leq 1 + \log_2 n \), so that \( n \leq 2^l \). In a first stage we compute

\[
A^0, A^1, A^2, \ldots, A^l,
\]

in depth \( O(\log^2 n) \) and size \( IM(n) \). 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 \( n - 2^l \) columns are new.) One sees inductively that the \( j \)th column of \( B_j \) is \( A^{j-1}b \) if \( 1 \leq j \leq 2^l \), and zero otherwise. (In \( B_l \), this is valid for \( 1 \leq j \leq n \).) The cost is the same as for the first stage.

In step 1 of the algorithm Characteristic polynomial, we do not really need all \( A_i^l \), but only all top left entries \( e_i^0 A_i^l e_i^0 \), where \( e_i^0 = (1, 0, \ldots, 0) \in F^r \). For any \( r \leq n \), all vectors \( A_i^s e_i^0 \) (\( 0 \leq k \leq n \)) can be computed in size at most \( 2M(n)(1 + \log_2 n) \) 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, \( \# F \geq 6n^2 \)).

**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

\[
A^{-1} = \begin{pmatrix} B^{-1} & 0 \\ -D^{-1}C B^{-1} & D^{-1} \end{pmatrix}.
\]

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

\[
x_1 = c_1,
\]

\[
x_2 = a_2 x_1 + c_2,
\]

\[
\vdots
\]

\[
x_n = a_n x_1 + a_{n-2} x_2 + \cdots + a_{n-1} x_{n-1} + c_n,
\]

where \( a_i, c_i \in F \), 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

\[
\chi = \det (xI - A) = x^n - s_1 x^{n-1} + s_2 x^{n-2} + \cdots + (-1)^n s_n \in F[x]
\]

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 \( \chi \).
Chapter 13. Parallel Linear Algebra

13.5 Polynomial and Matrix Problems

The goal of this section is a set of reductions between polynomial and matrix problems. A reduction $f \leq g$ 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 $\tau_d : F[x] \to F^{d \times d}$

\[
\begin{pmatrix}
0 & a_0 & a_1 & \cdots & a_{d-1} \\
& & & \vdots & \\
& & \ddots & & \\
& & & & a_1 \\
& & & & 0 \\
\end{pmatrix}
\]

Thus the image of $\tau_d$ 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**

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

**DETERMINANT** = $(\text{DETERMINANT}_n)_{n \in \mathbb{N}}$ with

\[\text{DETERMINANT}_n = \det((x_{ij})_{1 \leq i, j \leq n})\]

is the determinant problem.

We define further computational problems:

**POLYPROD**: product of two polynomials.

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

---

**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.
Similarly, we have

\[ \text{ITPOLYPROD}_n: \quad f_1 \cdots f_n \text{ for } f_1, \ldots, f_n \in F[x] \text{ of degree at most } n, \]

\[ \text{POLYINV}_n: \quad f^{-1} \mod x^n \text{ for } f \in F[x], f(0) \neq 0, \]

\[ \text{MATPROD}_n: \quad A \cdot B \text{ for } A, B \in F^{n \times n}, \]

\[ \text{ITMATPROD}_n: \quad A_1 \cdots A_n \text{ for } A_1, \ldots, A_n \in F^{n \times n}, \]

\[ \text{MATINV}_n: \quad A^{-1} \text{ for } A \in F^{n \times n} \text{ invertible}. \]

**THEOREM 13.4**

1. \( \text{POLYPROD} \leq_F \text{MATPROD} \)
2. \( \text{POLYINV} \leq_F \text{MATINV} \)
3. \( \text{POLYINV} \leq_F \text{ITPOLYPROD} \leq_F \text{ITMATPROD} \)

**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

\[ \tau_{2n+1}(fg) = \tau_{2n+1}(f) \cdot \tau_{2n+1}(g) = \text{MATPROD}_{2n+1}(\tau_{2n+1}(f), \tau_{2n+1}(g)). \]

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 \( \text{MATPROD} \), 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 \( \text{MATPROD}_{2n+1} \), with \( 2(2n + 1)^2 \) inputs and \( (2n + 1)^2 \) outputs. The input gates and zero are connected to the \( \text{MATPROD} \) gate according to \( \tau_{2n+1} \), and the functions required for \( \text{POLYPROD} \) are among those computed by the \( \text{MATPROD} \) 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:

\[ \text{DETERMINANT}_n: \quad \text{det} A \text{ for } A \in F^{n \times n}, \]

\[ \text{CHARPOLY}_n: \quad \chi(A) \text{ for } A \in F^{n \times n}, \]

\[ \text{MATPOWERS}_n: \quad A^2, A^3, \ldots, A^n \text{ for } A \in F^{n \times n}, \]

\[ \text{NONSINGE}(A) \text{ with } Ax = b \text{ for } A \in F^{n \times n} \text{ invertible} \]

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 NONSINGE 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. \( \text{DETERMINANT} \leq_F \text{CHARPOLY} \): the determinant is the constant term of the characteristic polynomial, up to the sign.
2. \( \text{CHARPOLY} \leq_F \text{ITMATPROD} \): The algorithm Characteristic polynomial of Section 13.4 has shown that

\[ \text{CHARPOLY} \leq_F \text{MATPOWERS} + \text{ITPOLYPROD} + \text{POLYINV}. \]

Together with Theorem 13.4 (3) and the trivial \( \text{MATPOWERS} \leq_F \text{ITMATPROD} \), the required reduction follows. This is by far the most challenging reduction in this proof.

3. \( \text{ITMATPROD} \leq_F \text{MATPOWERS} \): Given \( A_1, \ldots, A_n \in F^{n \times n} \), we may consider

\[
B = \begin{bmatrix}
I & A_1 & \cdots & 0 \\
0 & \ddots & \ddots & \vdots \\
& & A_n & I \\
& & 0 & I
\end{bmatrix} \in (F^{n \times n})^{(n+1) \times (n+1)}
\]

as an \((n^2 + n) \times (n^2 + n)\)-matrix. Then

\[
B^n = \begin{bmatrix}
I & * & \cdots & A_1 \cdots A_n \\
* & \ddots & \ddots & \vdots \\
\vdots & \ddots & * & I \\
0 & & I & I
\end{bmatrix}
\]

and \( \text{ITMATPROD}_n(A_1, \ldots, A_n) \) can be read off \( \text{MATPOWERS}_{n^2+n}(B) \).
4. MATPOWERS $\leq F$ MATINV: Given $A \in F^{n \times n}$, we consider

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

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

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

$$B^{-1} = (1 - \tau_{n+1}(Ax))^{-1} = \sum_{0 \leq k \leq n} \tau_{n+1}(Ax)^k$$

$$= \tau_{n+1}(\sum_{0 \leq k \leq n} A^k x^k)$$

Again, MATPOWERS$_n(A)$ can be read off 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 $Ax_i = e_i$, 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 $i$th column of $A^{-1}$.

6. NONSINGEQ $\leq F$ DETERMINANT: follows with Cramer's rule.

We define the "complexity class"

$$DET_F = \{ f : f \leq P \}$$

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 13.3 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 $F = Q$ 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

$$r + \dim_F \ker A = n. \quad (13.5)$$

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 $\mu_0$. 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 rank $A$ to that of rank$_{\text{alg}} A$. 


What is the relation between rank and algebraic rank? Let \( s = n - \text{rank} A \), so that \( s = \dim \ker A \), and suppose that \( u_1, \ldots, u_s \) 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

\[
A = \begin{bmatrix}
1 & & & \\
0 & \ast & & \\
\vdots & & \ddots & \\
0 & & & B
\end{bmatrix}.
\tag{13.6}
\]

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

\[
\text{rank} A = r = n - s \leq n - t = \text{rank}_{\text{alg}} A.
\]

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

**LEMMA 13.3**

If \( \text{rank} A = \text{rank} A^2 \), then \( \text{rank} A = \text{rank}_{\text{alg}} A \).

**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 \( \text{rank} A = \text{rank}_{\text{alg}} A \).

So suppose \( a = (a_{s+1}, \ldots, a_n) \in F^{n-s} \) with \( B a = 0 \), and let \( \bar{a} = (0, \ldots, 0, a_{s+1}, \ldots, a_n) \in F^n \). Then \( \bar{a} \in \ker A \), and hence \( A^2 \bar{a} = 0 \).

Thus \( \bar{a} \in \ker A \), and hence \( A \bar{a} = 0 \), hence \( B \bar{a} = 0 \), and indeed \( B \) is nonsingular.

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

\[
A' = \begin{bmatrix}
0 & A \\
A^t & 0
\end{bmatrix} \in F^{2n \times 2n}.
\]

Then \( \text{rank} A = \frac{1}{2} \text{rank} A' \), 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 \( F \),

\[
Y = \text{diag}(1, y, \ldots, y^{n-1}) = \begin{bmatrix}
1 & & & \\
& y & & \\
& & \ddots & \\
& & & y^{n-1}
\end{bmatrix} \in F(y)^{n \times n},
\]

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

**LEMMA 13.4**

\( \text{rank} B^2 = \text{rank} B \).

**PROOF**

It is sufficient to show that

\[
\text{rank} AYA = \text{rank} A,
\]

since \( Y \) is nonsingular and

\[
\text{rank} B^2 = \text{rank} AYA = \text{rank} AYA = \text{rank} A = \text{rank} YA = \text{rank} B.
\]

Since \( \text{rank} AYA \leq \text{rank} A \), we only have to show \( \text{rank} AYA \geq \text{rank} A \).

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

\[
s = \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq n} a_{ij}z^i w_j = w^t Y v = v^t Y^t v = u^t A^t Y A u = 0,
\]

where we have used that \( A \) is symmetric: \( A^t = A \). Suppose that \( v \neq 0 \).

Let \( m_i = \deg v_i \) (with \( \deg 0 = -\infty \)), \( m = \max \{m_i; 1 \leq i \leq n\} \), \( k = \max \{i; 1 \leq i \leq n, m_i = m\} \). Terms containing \( z^{m_k} \) only occur in summands of \( \sum w_i y^{i-1} \in F[y, z] \) 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 \( z^{m_k} y^{m-k-1} \) has nonzero coefficient in the above sum. Thus \( s \neq 0 \). 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: \( \text{rank} A \).

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

THEOREM 13.7
Over any field \( F \), \( \text{MATRANK} \leq \text{ITMATPROD} \).

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 \( \phi_{n^2}(D_i) \in (F^{n \times n})^{n^2} \cong F^{n^2 \times n^2} \), by Lemma 13.2. Overall, we have reduced \( \text{MATRANK} \) to \( \text{ITMATPROD} \).

Note that just saying "a product of \( n \times n \)-matrices, each entry a polynomial of degree less than \( n^2 \) would only yield depth \( O(\log^2 n) \).

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: \( \text{DET}_F \) or \( \text{RANK}_F \).

We define the "complexity class"

\[ \text{RANK}_F = \{ f : f \leq \text{MATRANK} \}, \]

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. \( \text{RANK}_F \subseteq \text{DET}_F \).
2. \( \text{MATRANK}, \text{MAXMINOR}, \text{BASIS}, \text{and SOLVABILITY} \) are complete for \( \text{RANK}_F \).

PROOF
(1) follows from \( \text{MATRANK} \leq \text{ITMATPROD} \in \text{DET}_F \).
(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 \( A_1, \ldots, A_r \) 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

\[ \sum_{1 \leq j \leq i} A_j x_j = A_i \]
of \( n \) linear equations in \( i - 1 \) indeterminates has a solution \((x_1, \ldots, x_{i-1}) \in F^{n-1}\). (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

\[
\text{rank} A = \text{rank} (A|b),
\]

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,
- decide whether \( A \in F^{n \times n} \) is singular,
   - given \( A \in F^{n \times n} \) and \( b \in F^n \), compute the bit
   - \( c = (\exists y \in F^n \ Ax = b) \), and if \( c = \text{true} \),
   - 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

\[
\forall x \in F^n \quad \left( Ax = b \iff \begin{bmatrix} A & b \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0 \text{ with } y = -1 \right)
\]

Determine a basis \( z_1, \ldots, z_k \in F^{n+1} \) of the nullspace of

\[
\begin{bmatrix} A & b \\ 0 & 0 \end{bmatrix} \in F^{(n+1) \times (n+1)}.
\]

Then return \( c = \text{true} \) if and only if \( z_{i,n+1} \neq 0 \) for some \( i \), \( 1 \leq i \leq k \). If \( c = \text{true} \), 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. **NonsingEQ \leq EQ**: 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 \( r = \text{rank} A \). For all \( i \), \( r < i \leq n \), solve the nonsingular system of linear equations

\[
M x_i = y_i,
\]

where \( y_i \in F^r \) consists of the top \( r \) entries of the \( i \)th column of \( A \):

For \( r < i \leq n \), let

\[
z_i = (x_i, 0, \ldots, 0, -1, 0, \ldots, 0) \in F^n.
\]

We now claim that \( z_{r+1}, \ldots, z_n \) form a basis of the nullspace \( \text{ker} A \) of \( A \). Let \( r < i \leq n \). We first show that \( z_i \in \text{ker} A \). For \( 1 \leq j \leq r \), the \( j \)th entry of \( A z_i \) 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 \( j \)th 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 \( (A z_i)_j = A_{j*} x_i = 0 \). Combining these, we have \( A z_i = 0 \).

On the other hand, \( \dim \ker A = n - r \) 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 \( \text{RANK}_F \neq \text{DET}_F \)?
2. Is INDEPENDENCE complete for \( \text{RANK}_F \)?
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 already 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 \( B = \{ T, F \} \), 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 \( “x \neq 0” \) 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 \( \alpha \) is by a nonzero field element, for any specific input supplied for the variables.

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

![Diagram of an arithmetic Boolean circuit for a linear equation](image-url)

**FIGURE 13.5**
An arithmetic Boolean circuit for a linear equation \( x_1 t = x_2 \).
EXAMPLE 13.4

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

\[
\text{MATRANK}_n(A) = \frac{T \cdots T F \cdots F}{n-r} \quad \text{for } A \in F^{m \times n} \text{ and } r = \text{rank} A.
\]

EXAMPLE 13.5

Let us see how the computational problem MAXMINOR = \((\text{MAXMINOR}_n)_{n \in \mathbb{N}}\) of determining a maximal nonsingular minor of matrices, discussed in Section 13.7, fits into this framework. Intuitively, for each \(n \in \mathbb{N}\) and matrix \(A \in F^{m \times n}\) one should find some sets \(I, J \subseteq \{1, \ldots, n\}\) such that the minor \((=\text{submatrix})\) 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, MAXMINOR\(_n\) is the set of all functions

\[
g : F^{m \times n} \rightarrow \mathbb{B}^n \times \mathbb{B}^n
\]

(with \(N = \{1, \ldots, n\}\)) where \(g(A)\) are the row and column indices of a maximal nonsingular minor of \(A\), for all \(A \in F^{m \times n}\). 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 \(i \in I \iff y_i = T\), and \(j \in J \iff z_j = T\). As a specific example, consider

\[
A = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 1 & -2 \\ 2 & 2 & 0 \end{pmatrix} \in \mathbb{Q}^{3 \times 3}.
\]

The last row of \(A\) is the sum of the first two rows. In our formalism, the fact that \(
\begin{bmatrix} 1 & 2 \\ 1 & -2 \end{bmatrix}
\)

is a maximal nonsingular minor of \(A\) is expressed as follows:

\((A, \text{TFTFB}) \in \text{MAXMINOR}_3\).

An arithmetic Boolean circuit solving MAXMINOR would compute, on any input \(A \in F^{m \times n}\), some output \(f_n(A) = (y_1, \ldots, y_n, z_1, \ldots, z_n) \in \mathbb{B}^{2n}\), and the requirement is that indeed \(f_n(A)\) 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 \( \alpha_n \). 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 \( \beta_n \), whose only inputs or constants are the inputs to \( \alpha_n \) and 1, and whose outputs are the constants used in \( \alpha_n \). (Often, 1 will be the only constant used.)

Furthermore, we have to use the degree \( \deg \gamma_n \) of a piecewise rational function \( \gamma_n \). If \( \gamma_n = (\mathbb{F}_n, h_n) \) with \( h_n \in \mathbb{F}[x_1, \ldots, x_n] \) 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.

\[
P_F = \{ g = (g_n)_{n \in \mathbb{N}} : \exists \text{ P-uniform } \alpha = (\alpha_n)_{n \in \mathbb{N}} \text{ computing } g \text{ with } S(\alpha_n) = n^{O(1)}, \text{ and } \deg \gamma_n = n^{O(1)} \},
\]

\[
NC_F^k = \{ g = (g_n)_{n \in \mathbb{N}} : \exists \text{ P-uniform } \alpha = (\alpha_n)_{n \in \mathbb{N}} \text{ computing } g \text{ with } S(\alpha_n) = n^{O(1)} \text{ and } D(\alpha_n) = O(\log^n n), \text{ and } \deg \gamma_n = n^{O(1)} \text{, for any } k \in \mathbb{N},
\]

\[
NC_F = \bigcup_{k \in \mathbb{N}} NC_F^k.
\]

Here, \( g \) stands for a computational problem, and \( \alpha \) 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 \( O(\log^n n) \) 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 \( NC(F\text{-uniform}) \subseteq NC_F \), and similarly for the other classes.

In Boolean circuit complexity, another model of importance is the unbounded fan-in Boolean circuit, where \( v \)- and \( \Lambda \)-gates may have any number of inputs. Restricting the depth to \( O(\log^n n) \), 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 \( AC^0 \subseteq NC^1 \) of Furst et al. [19] and Ajtai [1]; the parity function is in \( NC^1 \), and they show that it is not in \( AC^0 \).

If we allow unbounded fan-in \( \Lambda \)-gates, but only \( v \)-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 \( v \) and bounded \( \Lambda \).

We take the latter model as our template for the arithmetic case. A semi-unbounded fan-in arithmetic Boolean circuit \( \alpha \) is like an ordinary arithmetic Boolean circuit, except that the + and \( v \)-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 \( \alpha \), and \( S(\alpha) \) the sum of the sizes of all gates in \( \alpha \). These circuits lead to the following complexity classes.

\[
SAC_F^k = \{ g = (g_n)_{n \in \mathbb{N}} : \exists \alpha = (\alpha_n)_{n \in \mathbb{N}} \text{ of semi-unbounded fan-in computing } g \text{ with } S(\alpha_n) = n^{O(1)}, D(\alpha_n) = O(\log^k n), \text{ and } \deg \gamma_n = n^{O(1)} \}, \text{ for any } k \in \mathbb{N},
\]

\[
SAC_F = \bigcup_{k \in \mathbb{N}} SAC_F^k.
\]

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

\[
NC_F^0 \subseteq SAC_F^0 \subseteq NC_F^1 \subseteq \cdots \subseteq NC_F = SAC_F \subseteq P_F.
\]

Whenever one has such a hierarchy, one of the first (and usually most difficult) questions is: Does it collapse or not? I.e., is \( NC_F^k \subseteq SAC_F^k \subseteq NC_F^{k+1} \) for all \( k \)? Is \( NC_F \subseteq P_F \)? Throughout this chapter we have worked at the low end of this hierarchy, \( NC_F^0 \).

Only one difference is easy to see: \( NC_F^0 \neq SAC^0 \). We have \( SUM \in AC^0 \), and by Theorem 13.1 (3), \( SUM_0 = x_1 + \cdots + x_n \) requires depth at least \( \log_2 n \) on arithmetic Boolean circuits with fan-in two, and so is not in the trivial class \( AC^0 \).

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 \( \alpha_n \) of constant
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

\[ \text{RANK}_F \subseteq \text{DET}_F \subseteq \text{SAC}^1_F \subseteq \text{NC}^1_F. \]

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

The following problems are unsolved.

**OPEN QUESTION 13.2**
1. Is $\text{DET}_F = \text{SAC}^1_F$?
2. Is $\text{SAC}^1_F \neq \text{NC}^1_F$?

**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 argument 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: $D(a) \geq \min\{\log_2 b, \log_2 (q - b)\}$ if $1 \leq b < q$ and $a$ computes $b$th 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 & Soroussi [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 $\text{NC}^2$, 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) $\text{NC}$?

It is widely conjectured that $\text{NC} \neq P$. However, Valiant et al. [52] showed that for polynomials over a field $F$ we have "$\text{NC}^2 = P_F$": 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 & Soroussi [29]), for inversion in finite fields (von zur Gathen [26]), and for the Boolean exponentiation problem of computing $a^b \mod 2^n$, 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 $\text{NC}^2$, 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 $Q$ uses "short vectors in $Z$-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 [16], 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 $\alpha$ 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 $D(\alpha) \geq \log_2 \deg f$ (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 $(n^2 + r M(r)) \log^{O(1)} n$ processors, or in parallel time $O((\log n + \log^2 r)$ using $(n^2 + n M(r)) \log^{O(1)} n$ 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 $(n^2/r + r^2) \log^{O(1)} n$ 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 $\sigma$ 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


