SIAM J. COMPUT.
Vol. 13, No. 4, November 1984
© 1984 Society for Industrial and Applied Mathematics
010

PARALLEL ALGORITHMS FOR ALGEBRAIC PROBLEMS*

JOACHIM VON ZUR GATHEN†

Abstract. Fast parallel algorithms are presented for the following problems in symbolic manipulation of univariate polynomials: computing all entries of the extended Euclidean scheme of two polynomials over an arbitrary field, gcd and lcm of many polynomials, factoring polynomials over finite fields, and the squarefree decomposition of polynomials over fields of characteristic zero and over finite fields.

For the following estimates, assume that the input polynomials have degree at most n, and the finite field has p^d elements. The Euclidean algorithm is deterministic and runs in parallel time $O(\log^2 n)$. All the other algorithms are probabilistic (Las Vegas) in the general case, but when applicable to $\mathbf{Q}$ or $\mathbf{R}$, they can be implemented deterministically over these fields. The algorithms for gcd and lcm use parallel time $O(\log^2 n)$. The factoring algorithm runs in parallel time $O(\log^2 n \log^2 (d+1) \log p)$. The algorithm for squarefree decomposition runs in parallel time $O(\log^2 n)$ for characteristic zero, and in parallel time $O(\log^2 n + (d-1)\log p)$ for finite fields. All Las Vegas algorithms have failure probability less than 2^{-n}. For all algorithms, the number of processors is polynomial in n.

Key words. parallel processing, algebraic computing, symbolic manipulation, Euclidean algorithm, factorization of polynomials, squarefree decomposition

1. Introduction. In Borodin–von zur Gathen–Hopcroft [1982] the following program is laid out: obtain a "theory package for parallel algebraic computations," i.e. fast parallel computations for the widely used problems of symbolic manipulation in an algebraic context. In that paper, two basic problems were considered: solving systems of linear equations and computing the gcd of polynomials, both over arbitrary ground fields.

The present paper continues this program, and fast parallel solutions to the following algebraic problems are given: computing all entries of the extended Euclidean scheme of two polynomials over an arbitrary field, computing the gcd and lcm of many polynomials over an arbitrary field, factoring polynomials over finite fields, and the squarefree decomposition of polynomials over fields of characteristic zero and over finite fields.

As our model of parallel computation, we can take an algebraic PRAM (with instructions +, −, *, /, constants) or the parallel algebraic computation (directed acyclic) graphs, PACDAG for short, which we describe informally below. We will describe the algorithms of this paper in "high-level language," and not give actual implementations on a PACDAG. A more formal description would follow the lines of the discussion in Strassen [1983] of (one-processor) algebraic computation trees and the collections that they compute.

A PACDAG has two kinds of processors and (shared) variables, "arithmetic" and "boolean" ones. At each node of the (rooted) directed acyclic computation graph, each arithmetic processor can either perform an arithmetic operation (+, −, *, /) on two arithmetic variables, or access an arithmetic input variable, or fetch a constant from the ground field. Each boolean processor can either compute the negation or conjunction of (one resp. two) boolean variables, or it can take an arithmetic variable x and set a boolean variable to "true" if $x \neq 0$, and to "false" otherwise. (One can in fact simulate these boolean computations in the ground field if a conditional division instruction of the form "if $x \neq 0$ then $y = 1/x$" is allowed.) No write-conflicts are

* Received by the editors February 8, 1983, and in revised form August 12, 1984. An extended abstract of this paper appeared in Proc. 15th ACM Symposium on Theory of Computing, Boston, 1983, pp. 17–23.

† Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4.

JOACHIM VON ZUR GATHEN (1984). Parallel algorithms for algebraic problems. *SIAM Journal on Computing* **13**(4), 802–824. URL http://dx.doi.org/10.1137/0213050. Extended abstract in *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, Boston MA (1983).
This document is provided as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, notwithstanding that these works are posted here electronically. It is understood that all persons copying any of these documents will adhere to the terms and constraints invoked by each copyright holder, and in particular use them only for noncommercial purposes. These works may not be posted elsewhere without the explicit written permission of the copyright holder. (Last update 2016/05/18-14:18.)

allowed. At each node of the graph, each of the branches of the graph emanating from that node is labeled with a boolean variable; if on a given input the computation reaches that node and exactly one of these boolean variables is true, then the corresponding branch is chosen; otherwise the computation is not defined at that input.

At each leaf, the output is given by a sequence of arithmetic variables. Thus a PACDAG computes a collection in the sense of Strassen [1983].

Any PACDAG can obviously be described by a string over a finite alphabet, provided the field constants used can be described in this way. A family $(P_n)_{n\in\mathbf{N}}$ of PACDAGs is uniform if the description of P_n can be generated by a deterministic Turing machine in space $O(\log n)$, given n in unary as input. (See Ruzzo [1981], Cook [1983].) All algorithms of this paper are uniform.

We are concerned with two cost measures of a PACDAG: its parallel time (= depth of the graph = length of longest path from root to any leaf) and its size (= number of processors). All the algorithms of this paper work in parallel time $\log^{O(1)} n$ (usually $O(\log^2 n)$) and use $n^{O(1)}$ processors, where n is the input size. One additional feature is needed for some algorithms: a random generator that produces elements of a finite subset of the ground field at random, i.e. new "random variables" from which these random elements can be read.

Among the problems for which we do not have a fast parallel algorithm are the gcd of two integers and the factorization of a polynomial with rational coefficients. We give a reduction from the first problem to a problem related to the second, namely computing short vectors in integer lattices.

All our results belong to the "asymptotic approach" to parallel algorithms, where one is interested in obtaining the fastest possible parallel algorithms, allowing an almost arbitrary (say, polynomially bounded) number of processors. This contrasts with the approach of getting a speed-up from sequential to parallel time close to the number of processors: $T_{\text{seq}}/T_{\text{par}}$ is approximately the number of processors.

One interesting phenomenon occurs: all the problems eventually reduce to solving systems of linear equations. Hence the paramount role of the latter problem. On the theoretical level, this is expressed to a certain extent by Valiant's [1980] universality of the determinant (disallowing branching and division).

The algorithms for linear algebra in Borodin–von zur Gathen–Hopcroft [1982] used a polynomial number of processors, but this number was impractically large. Soon after, Berkowitz [1984] presented an algorithm for computing the determinant with parallel time $O(\log^2 n)$ and $O(n^{3.5})$ processors.

2. The extended Euclidean scheme for polynomials. We start with an easy result about division with remainder of polynomials.

LEMMA 2.1. *One can compute the quotient and remainder of two polynomials of degree at most n in parallel time* $O(\log^2 n)$.

Proof. Let $f,\ g \in F[x]$ be given, where F is an arbitrary field, and $k = \deg f - \deg g + 1 \leqq n$. Their quotient $q \in F[x]$ is uniquely determined by the condition $\deg(f - qg) < \deg g$, which can be expressed by a nonsingular system of k linear equations in the k coefficients of q. This system can be solved in parallel time $O(\log^2 k)$, as in Borodin–von zur Gathen–Hopcroft [1982]. Computation of the remainder then takes $O(\log n)$ parallel steps. □

Actually, quotient and remainder can be computed in parallel time $O(\log n)$. This was shown by Reif [1983] under the assumption that F supports a fast Fourier transform, and by Eberly [1984] in general.

Let $f, g \in F[x]$ with $0 \leq \deg g = m \leq n = \deg f$, where F is an arbitrary field. We set $a_0 = f$, $a_1 = g$ and consider the extended Euclidean scheme for (f, g):

$$\begin{aligned} a_0 &= q_1 a_1 + a_2, & s_2 a_0 + t_2 a_1 &= a_2, \\ &\vdots & &\vdots \\ a_{l-2} &= q_{l-1} a_{l-1} + a_l, & s_{l-1} a_0 + t_{l-1} a_1 &= a_{l-1}, \\ a_{l-1} &= q_l a_l, & s_l a_0 + t_l a_1 &= a_l. \end{aligned}$$

where the following conditions are satisfied for $2 \leq k \leq l$: $a_k, q_1, q_k, s_k, t_k \in F[x]$, $\deg a_k < \deg a_{k-1}$, $s_0 = 1$, $t_0 = 0$, $s_1 = 0$, $t_1 = 1$, $s_k = s_{k-2} - q_{k-1} s_{k-1}$ and $t_k = t_{k-2} - q_{k-1} t_{k-1}$. Thus the q's are the quotients and the a's the remainders of Euclid's algorithm, $\gcd(f, g)$ is the unique monic scalar multiple of a_l by convention, all gcd's of polynomials in $F[x]$ are monic) and the s's and t's are the "cumulants", "convergents" or "continuants". (The terminology is rather unsatisfactory. The last term, proposed by a certain Muir, provoked an amusing controversy (see Muir [1878]).) Sequential algorithms for computing these polynomials are important and well-studied; see Knuth [1981, 4.6.1], for an overview. One interesting feature is that $q_1, \cdots, q_l$ can be computed faster than can $a_1, \cdots, a_l$ (see Strassen [1983]).

The above conditions imply that a_k has "small" degree and is a linear combination of f and g with coefficients s_k and t_k. Obviously one can multiply a_k, s_k, t_k by a polynomial of small degree and conserve these properties. The following lemma shows that this is the only way of obtaining these properties, and thus gives a characterization of a_k, s_k, t_k. Our proof follows Kronecker [1881]; see Knuth [1981, Exercise 4.6.1–26], for a different approach.

LEMMA 2.2. *Let $f, g, a_k, s_k, t_k \in F[x]$ be as above, and $a, s, t \in F[x]$ with $a, t \neq 0$. Then the following two conditions are equivalent:*

(i) *$sf + tg = a$, and $\deg a + \deg t < n$.*

(ii) *There exist $k \in \{1, \cdots, l\}$ and $b \in F[x]$ such that*

$$a = ba_k, \quad s = bs_k, \quad t = bt_k,$$

$$\deg a_k \leq \deg a < (\deg a_{k-1} + \deg a_k)/2 < \deg a_{k-1}.$$

Furthermore, if the conditions are satisfied, then k and b in (ii) *are uniquely determined.*

Proof. By induction on k one proves that $\deg t_k = \sum_{1 \leq i < k} \deg q_i$ and $\deg t_k + \deg a_{k-1} = n$, and then (i) obviously follows from (ii).

For the other implication, we define $k \in \{1, \cdots, l\}$ by

$$\deg a_k \leq \deg a < \deg a_{k-1}.$$

This determines k uniquely, since

$$\deg a_l < \deg a_{l-1} < \cdots < \deg a_0,$$

$$\deg a_l = \deg \gcd(f, g) \leq \deg a < n = \deg a_0.$$

Eliminating g from

$$s_k f + t_k g = a_k, \qquad sf + tg = a,$$

we get

$$(s_k t - s t_k) f = t a_k - t_k a,$$

$$\deg(t a_k - t_k a) \leq \max\{n - 1 - \deg a + \deg a_k, n - \deg a_{k-1} + \deg a\} < n.$$

Since $\deg f = n$, we conclude that $s_k t - s t_k = 0$. By induction on k one easily sees that

$$s_k t_{k-1} - t_k s_{k-1} = (-1)^k$$

for $1 \leqq k \leqq l$, hence in particular $\gcd(s_k, t_k) = 1$. This together with $s_k t = s t_k \neq 0$ implies that there exists $b \in F[x]$ such that

$$t = b t_k.$$

Then also

$$s = b s_k, \quad a = b a_k, \quad 0 \leqq \deg b,$$

$$\begin{aligned} n > \deg a + \deg t &= 2 \deg b + \deg a_k + \deg t_k \\ &= 2 \deg b + \deg a_k + n - \deg a_{k-1}, \end{aligned}$$

$$\begin{aligned} 2 \deg a &= 2 \deg b + 2 \deg a_k \\ &< \deg a_{k-1} - \deg a_k + 2 \deg a_k = \deg a_{k-1} + \deg a_k. \end{aligned}$$

□

Remark. If $a = 0$ then the statement of Lemma 2.2 becomes valid by introducing the following (natural) notation: $a_{l+1} = 0$, $s_{l+1} = s_{l-1} - q_l s_l$, $t_{l+1} = t_{l-1} - q_l t_l$. We have to allow $k = l+1$ in (ii), and interpret arithmetic expressions involving $\deg a = -\infty$ "correctly."

The theory of subresultants (Collins [1967], Brown–Traub [1971]) provided an important development for sequential algorithms computing gcd's of polynomials (or, more generally, for the entries of the extended Euclidean scheme). Euclid's algorithm for the gcd can suffer from exponential intermediate expression swell if the coefficients of the polynomials are integers or polynomials themselves (Brown [1971]). The subresultant algorithms avoid this difficulty—which makes the algorithm decidedly impractical—by translating the problem into systems of linear equations. For parallel algorithms, this strategy of employing linear equations was successfully exploited for the gcd in Borodin–von zur Gathen–Hopcroft [1982], and here we use it to compute all entries of the extended Euclidean scheme.

We first give a self-contained exposition of the relevant results about subresultants. For this simplified version with Lemma 2.2 as the cornerstone we only have to introduce the "principal subresultants" as follows.

We write $f = f_n x^n + \cdots + f_0$, $g = g_m x^m + \cdots + g_0$ with $0 \leqq m \leqq n$ and $f_n g_m \neq 0$. For $0 \leqq i \leqq m$ we consider the $(n+m-2i) \times (n+m-2i)$-submatrix P_i of the Sylvester matrix of (f, g) which consists of the first $m-i$ columns of f_j's and the first $n-i$ columns of g_j's:

$$P_i = \begin{bmatrix} f_n & & & g_m & & \\ f_{n-1} & f_n & & g_{m-1} & g_m & \\ \vdots & & \ddots & \vdots & & \ddots \\ f_{n-m+i+1} & \cdots & f_n & g_{m-n+i+1} & \cdots & g_m \\ \vdots & & \vdots & \vdots & & \vdots \\ f_{2i-m+1} & \cdots & f_i & g_{2i-n+1} & \cdots & g_i \end{bmatrix}.$$

Thus P_0 is the Sylvester matrix of (f, g), and $\det(P_0)$ their resultant. One might call P_i a "principal subresultant" since its rows are the highest among the matrices for subresultants of the same size. We denote by c_k the leading coefficient of a_k.

THEOREM 2.3. *Using the above notation, we have for all* i, $1 \leqq i \leqq m$:
(i) $\exists k \in \{1, \cdots, l\} \deg a_k = i \Leftrightarrow \det(P_i) \neq 0$.
(ii) *If* $\deg a_k = i$ *and* $y_0, \cdots, y_{m-i-1}, z_0, \cdots, z_{n-i-1} \in F$ *are such that*

$$P_i \cdot \begin{bmatrix} y_{m-i-1} \\ \vdots \\ y_0 \\ z_{n-i-1} \\ \vdots \\ z_0 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

then

$$s_k = c_k(y_{m-i-1}x^{m-i-1} + \cdots + y_0),$$
$$t_k = c_k(z_{n-i-1}x^{n-i-1} + \cdots + z_0).$$

Proof. Let $e_i = (0, \cdots, 0, 1) \in F^{m+n-2i}$, and for any $y_0, \cdots, y_{m-i-1}$, $z_0, \cdots, z_{n-i-1} \in F$ let

$$s = y_{m-i-1}x^{m-i-1} + \cdots + y_0,$$
$$t = z_{n-i-1}x^{n-i-1} + \cdots + z_0,$$

where $1 \leqq i \leqq m$. By abuse of notation, we write $P_i(s, t) = e_i$ for the system of linear equations as in (ii). Obviously for any i, $1 \leqq i \leqq m$, we have

$\det(P_i) \neq 0 \Leftrightarrow P_i(s, t) = e_i$ has exactly one solution
$\Leftrightarrow$ there exist unique $s, t \in F[x]$ such that $\deg s < m - i$, $\deg t < n - i$ and $sf + tg$ is a monic of degree i.

Using Lemma 2.2, we will show that the latter condition is satisfied iff there exists $k \in \{1, \cdots, l\}$ such that $\deg a_k = i$. Note that for any i with $\deg a_l \leqq i \leqq m$ there exists $k \in \{1, \cdots, l\}$ such that $\deg a_k \leqq i < \deg a_{k-1}$. We distinguish four cases.

Case 1. $\exists k \in \{1, \cdots, l\}$ such that $\deg a_k = i$. Setting $s = c_k^{-1}s_k$, $t = c_k^{-1}t_k$ we see that $sf + tg$ is monic of degree i. It follows from Lemma 2.2 that s, t are uniquely determined. This proves (i) in this case, and also the statement (ii).

Case 2. $\exists k \in \{1, \cdots, l\}$ such that $\deg a_k < i < (\deg a_k + \deg a_{k-1})/2$. For any $b \in F[x]$ of degree $i - \deg a_k$ and with leading coefficient c_k^{-1}, set

$$s(b) = bs_k, \quad t(b) = bt_k, \quad a(b) = ba_k.$$

Then $\deg(s(b)) < m - i$, $\deg(t(b)) < n - i$ and $a(b) = s(b)f + t(b)g$ is monic of degree i. Since there exists more than one such b, (i) is proven in this case.

Case 3. $\exists k \in \{1, \cdots, l\}$ such that $(\deg a_k + \deg a_{k-1})/2 \leqq i < \deg a_{k-1}$. Assume that $\deg s < m - i$, $\deg t < n - i$ and $a = sf + tg$ has degree i. From Lemma 2.2, we get $i < (\deg a_k + \deg a_{k-1})/2$. Hence no such s, t exist, and (i) is proven in this case.

Case 4. $i < \deg a_l$. No s, t satisfy the condition, since no nonzero polynomial of degree less than $\deg a_l = \deg(\gcd(f, g))$ is a linear combination of f and g. (This can be interpreted as Case 3 with $k = l + 1$.) □

We can now state the parallel algorithm that computes all entries of the extended Euclidean scheme of two polynomials.

ALGORITHM EUCLID.
Input: The coefficients of $f, g \in F[x]$ with $\deg g = m \leqq n = \deg f$.
Output: The coefficients of polynomials A_k, Q_k, S_k, T_k for $1 \leqq k \leqq l$, which are the entries of the extended Euclidean scheme of (f, g).

1. For all i, $0 \leqq i \leqq m$, compute $p_i = \det(P_i)$, where P_i is the ith principal subresultant as above.
2. Set

$$\{n_1, \cdots, n_l\} = \{i: 0 \leqq i \leqq m \text{ and } p_i \neq 0\}$$

with $m = n_1 > n_2 > \cdots > n_l$. ($n_k$ will be $\deg a_k$.)

3. For all k, $1 \leqq k \leqq l$, and $i = n_k$ compute $y_0, \cdots, y_{m-i-1}, z_0, \cdots, z_{n-i-1} \in F$ such that

$$P_i \cdot \begin{bmatrix} y_{m-i-1} \\ \vdots \\ y_0 \\ z_{n-i-1} \\ \vdots \\ z_0 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

(This is a nonsingular system of linear equations and has a unique solution.) Set

$$u_k = y_{m-i-1}x^{m-i-1} + \cdots + y_0,$$

$$v_k = z_{n-i-1}x^{n-i-1} + \cdots + z_0,$$

$$w_k = u_k f + v_k g.$$

(u_k, v_k, w_k will be scalar multiples of s_k, t_k, a_k. In steps 4 and 5 we compute these scalar factors.)

4. For all k, $2 \leqq k \leqq l$, compute d_k, $r_k \in F[x]$ such that

$$w_{k-2} = r_{k-1}w_{k-1} + d_k,$$

$$\deg d_k < \deg w_{k-1}.$$

(Dividing w_{k-2} by w_{k-1} with remainder. Use $w_0 = f$.)

5. For all k, $1 \leqq k \leqq l$, compute the following.

$$\delta_k = \text{leading coefficient of } d_k,$$

$$e_k = \begin{cases} \delta_k\delta_{k-2}\cdots\delta_2 & \text{if } k \text{ is even,} \\ \delta_k\delta_{k-2}\cdots\delta_1 & \text{if } k \text{ is odd,} \end{cases}$$

$$A_k = e_k w_k,$$

$$Q_k = (A_{k-1} - A_{k+1})/A_k,$$

$$S_k = e_k u_k,$$

$$T_k = e_k v_k.$$

(with $\delta_1 =$ leading coefficient of g, and $A_{l+1} = 0$.)

THEOREM 2.4. *Over any field, algorithm* EUCLID *computes the entries of the extended Euclidean scheme of* (f, g). *If* $\deg g \leqq \deg f \leqq n$, *then it can be performed in parallel time* $O(\log^2 n)$.

Proof. By Borodin–von zur Gathen–Hopcroft [1982] steps 1 and 3 can be performed in parallel time $O(\log^2 n)$. Using Lemma 2.1 for step 4 and the division in step 5, the timing estimate is clear. The proof of correctness below shows that the division in step 5 is exact.

It follows from Theorem 2.3(i) that

$$\#\{i: 0 \leqq i \leqq m \text{ and } p_i \neq 0\} = l = \text{length of Euclidean scheme},$$

$$(n_1, \cdots, n_l) = (\deg a_1, \cdots, \deg a_l),$$

and from Theorem 2.3(ii) that

$$s_k = c_k u_k, \quad t_k = c_k v_k, \quad a_k = c_k w_k,$$

for $1 \leqq k \leqq l$, where c_k is the leading coefficient of a_k. From

$$(c_{k-2} r_{k-1}) w_{k-1} + c_{k-2} d_k = c_{k-2} w_{k-2} = a_{k-2}$$

$$= q_{k-1} a_{k-1} + a_k = (c_{k-1} q_{k-1}) w_{k-1} + c_k w_k,$$

$$\deg (c_{k-2} d_k) < \deg w_{k-1},$$

$$\deg (c_k w_k) < \deg w_{k-1},$$

and the uniqueness of division with remainder we conclude that

$$c_{k-2} d_k = c_k w_k.$$

Since w_k is monic, we obtain

$$c_{k-2} \delta_k = c_k.$$

By induction on k it follows that $c_k = e_k$ and $a_k = A_k$. We also conclude that $q_k = Q_k$, $s_k = S_k$, $t_k = T_k$. □

Remark 2.5. Let us describe in some more detail how the branching in step 2 can be implemented on a PACDAG. Given the sequence $p_0, \cdots, p_m$, we want to store in variables q_{kj} $(0 \leqq k \leqq l,\ j \in \{1, \cdots, n\}^2)$ the matrices $\tilde{P}_i$, where $\tilde{P}_i$ is P_i extended by 1's on the diagonal to an $n \times n$-matrix, such that $Q_k = \tilde{P}_{n_k}$ for $1 \leqq k \leqq l$.

Using a binary splitting (starting with all values $0 \leqq u = v \leqq m$ and ending with $u = 0$, $v = m$), we compute larger and larger intervals $[u, \cdots, v]$ such that if $s = \#\{i: u \leqq i \leqq v \text{ and } p_i \neq 0\} - 1$, then $Q_u, \cdots, Q_{u+s}$ are the $\tilde{P}_i$ with $p_i \neq 0$ in the original order, and $Q_{u+s+1}, \cdots, Q_v = 0$. Let q_k be p_i for the i corresponding to k.

Given two such intervals $[u, \cdots, v]$ and $[v+1, \cdots, w]$, we first branch in depth $O(\log n)$ according to the value of s, $u \leqq s \leqq v$, such that $q_s \neq 0$ and $q_{s+1} = 0$ (or ($s = u - 1$ and $q_u = 0$) or ($s = v$ and $q_v \neq 0$)), and similarly for t, $v < t \leqq w$. Then set

$$Q_k = \begin{cases} Q_k & \text{if } u \leqq k \leqq s, \\ Q_{k+v-s} & \text{if } s < k \leqq s+t-v, \\ 0 & \text{if } s+t-v < k \leqq w. \end{cases}$$

Remark 2.6. As mentioned above, the entries of the Euclidean scheme of integer polynomials may be very large. However, the u_k, v_k, w_k computed in the algorithm will have reasonably small coefficients (with binary length polynomial in the input length, by Edmonds [1967]). Hence, as long as one wants a_k, q_k, s_k, t_k only up to a scalar multiple, steps 1, 2, 3 provide a solution with small coefficients (using the scalar multiple $(w_{k-1} - \delta_{k+1} w_{k+1}) / w_k$ of q_k). This is quite satisfactory for all practical purposes. Only if we want to compute the entries of the extended Euclidean scheme exactly, then we have to perform steps 4 and 5 and may be faced with very large integers. A similar observation applies to multivariate polynomials

Remark 2.7. Algorithm EUCLID can be considered as an "NC^1-reduction" (i.e. one using only operations of parallel time $O(\log n)$) from the problem of computing

the extended Euclidean scheme to that of computing the determinant of matrices. In fact, all the algorithms of this paper can be considered as (probabilistic) NC^1-reductions to the determinant problem. (We have to assume that the field is fixed; see § 4 for a discussion of finite fields described in the input.) In later sections, algorithms will make use of the rank of matrices. The rank problem itself is NC^1-reducible to the determinant computation—probabilistically in general, and deterministically over any real field, in particular over $\mathbf{Q}$ or $\mathbf{R}$. Thus over $\mathbf{Q}$, all reductions are deterministic, but over an arbitrary ground field they are probabilistic.

Remark 2.8. If F is $\mathbf{Q}$ or a finite field, then we can consider the input as represented by bit strings, and ask for Boolean circuits, say, that compute the functions considered here in small depth and polynomial size. A basic result is that the determinant function is in Boolean NC^2 (Csanky [1976], Borodin–Cook–Pippenger [1983]). It follows from Remark 2.7 that all the computational problems considered in this paper are in NC^2 (again, provided that the ground field is fixed).

3. Gcd and lcm of many polynomials. In § 5 below, we want to compute the squarefree decomposition of polynomials. It turns out that we first have to present an ancillary result, which may be of independent interest: how to compute the gcd of many polynomials in parallel. Thanks go to Steve Cook for pointing out the algorithm below. We also need the least common multiple lcm of many polynomials, and give an algorithm for this problem.

Let F be an arbitrary field, $f_1, \cdots, f_n \in F[x]$ have degree at most n, and $g = \gcd(f_1, \cdots, f_n)$. We want to compute g from $f_1, \cdots, f_n$. It is easy to see that there exist $s_1, \cdots, s_n \in F[x]$ such that $\sum s_i f_i = g$. We claim that in addition one can have $\deg s_i < n$. To prove this claim, reorder the polynomials such that $\deg f_1 \geqq \deg f_i$ for all i. For each $i \geqq 2$, divide s_i by f_1 with remainder: $s_i = q_i f_1 + \bar{s}_i$, and $\deg \bar{s}_i < \deg f_1 \leqq n$. Set $\bar{s}_1 = s_1 + \sum_{i \geqq 2} q_i f_i$. Then $\sum_{1 \leqq i \leqq n} \bar{s}_i f_i = g$, and $\bar{s}_1 f_1 = g - \sum_{i \geqq 2} \bar{s}_i f_i$ has degree less than $n + \deg f_1$. Hence $\deg \bar{s}_i < n$ for all i, and the claim is proven.

Now let

$$d = \min \{\deg f : \exists s_1, \cdots, s_n \in F[x] \deg s_i < n \text{ for all } i \text{ and } f = \sum s_i f_i \neq 0\}.$$

Then $d = \deg g$, by the claim above. We can now set up systems A_k of linear equations for $0 \leqq k \leqq n$, where A_k expresses "$\sum s_i f_i$ is monic of degree k." We write

$$f_i = \sum_{0 \leqq j \leqq n} f_{ij} x^j, \qquad s_i = \sum_{0 \leqq j < n} s_{ij} x^j,$$

and A_k consists of the linear equations

$$\sum_{\substack{1 \leqq i \leqq n \\ 0 \leqq j \leqq l}} s_{ij} f_{i,l-j} = \begin{cases} 0 & \text{for } k < l < 2n, \\ 1 & \text{for } l = k, \end{cases}$$

in the indeterminates s_{ij}. (Write zero whenever a subscript is out of range.) Thus A_k has $2n - k$ equations in at most n^2 variables. From the above we know that A_k has a solution iff $k \geqq d$. In particular, A_d has a solution, and from a solution of A_d we can easily compute g.

ALGORITHM GCD OF MANY POLYNOMIALS.

Input: a number $n \in \mathbf{N}$, and $f_1, \cdots, f_n \in F[x]$ with $\deg f_i \leqq n$ for all i.

Output: either $g = \gcd(f_1, \cdots, f_n)$ or "failure."

1. For all k, $0 \leqq k \leqq n$, determine whether A_k has a solution, and if it has, compute a solution $(s_{ij}(k))$ of A_k. (Using a Las Vegas algorithm in general, but a deterministic algorithm if F is real.)

2. Set $d = \min\{k: A_k \text{ has a solution}\}$.
3. Return $g = \sum_{\substack{1 \le i \le n, \\ 0 \le j < n}} s_{ij}(d) x^j f_i$.

THEOREM 3.1. *Algorithm* GCD OF MANY POLYNOMIALS *either returns the* gcd *g of the input polynomials, or it reports "failure." The latter happens with probability less than* 2^{-n}. *If F is real, then the algorithm can be performed deterministically. The parallel time for the algorithm is* $O(\log^2 n)$.

Proof. From the discussion above it is clear that the algorithm correctly computes the (monic) gcd of the input polynomials. Steps 2 and 3 can be performed (deterministically) in parallel time $O(\log n)$. In step 1 we apply the Las Vegas algorithm of Borodin–von zur Gathen–Hopcroft [1982, Thm. 5(ii)]. This algorithm works in parallel time $O(\log^2 n)$. The coefficient matrix of A_k is considered to be of size $n^2 \times n^2$, and the failure probability for each k is at most 2^{-n^2}. Thus the total failure probability in step 1 is not greater than $(n+1)2^{-n^2} < 2^{-n}$.

If F is real, then we can use the deterministic version of the parallel algorithm for solving singular systems of linear equations. □

We have used n as a separate input rather than read it off the input polynomials in order to ensure failure probability less than 2^{-n} in applications where we do not know the exact degree of the input polynomials. We remark that by computing gcd's of pairs of polynomials along a binary tree, and using the algorithm from Borodin–von zur Gathen–Hopcroft [1982, Thm. 2], we get a deterministic algorithm for $\gcd(f_1, \cdots, f_n)$ over an arbitrary field running in parallel time $O(\log^3 n)$.

For two polynomials $f_1, f_2 \in F[x]$ the relation $\gcd(f_1, f_2) \cdot \operatorname{lcm}(f_1, f_2) = f_1 f_2$ holds, and both the gcd and lcm can be computed in parallel time $O(\log^2 n)$. For more than two input polynomials, one does not have such a simple relationship between gcd and lcm.

ALGORITHM LCM OF MANY POLYNOMIALS.
Input: A number $n \in \mathbf{N}$, and polynomials $f_1, \cdots, f_n \in F[x]$ of degree at most n.
Output: Either the monic least common multiple $g = \operatorname{lcm}(f_1, \cdots, f_n)$, or "failure."

1. Set $d_i = \deg f_i$, $m = \max_{1 \le i \le n} d_i$, $s = \sum_{1 \le i \le n} d_i$, and replace each f_i by its monic multiple $f_i/(\text{leading coefficient of } f_i)$.
2. For all k, $m \le k \le s$, do the following. Let B_k be the system of (inhomogeneous) linear equations that expresses

$$u_1 f_1 - u_2 f_2 = u_2 f_2 - u_3 f_3 = \cdots = u_{n-1} f_{n-1} - u_n f_n = 0.$$

Here each $u_i = \sum_{0 \le j \le k - d_i} u_{ij} x^j$ is a monic polynomial of degree $k - d_i$, and hence B_k consists of $(n-1)k$ linear equations in the $\sum (k - d_i) = nk - s$ indeterminate coefficients u_{ij} $(1 \le i \le n, 0 \le j < k - d_i)$ of the u_i's. (Note that $(n-1)k \ge nk - s$.) Determine whether B_k has a solution, and compute a solution $u_{ij}(k)$, if it exists. (This can be done deterministically over a real field, and probabilistically in general.)

3. Set $d = \min\{k: B_k \text{ has a solution}\}$.
4. Set $u = \sum_{0 \le j < d - d_1} u_{1j}(d) x^j + x^{d - d_1}$, and return $g = uf_1$.

THEOREM 3.2. *Algorithm* LCM OF MANY POLYNOMIALS *either returns the* lcm *of the input polynomials, or it reports "failure." The latter happens with probability less than* 2^{-n}. *If F is real, then the algorithm can be performed deterministically. The algorithm runs in parallel time* $O(\log^2 n)$.

We remark that over an arbitrary field the lcm can be computed deterministically in parallel time $O(\log^3 n)$ along a binary tree.

4. Factoring polynomials over finite fields. The parallel complexity of the factorization of polynomials over finite fields was the original motivation for the work begun in Borodin–von zur Gathen–Hopcroft [1982]. We present here the Cantor–Zassenhaus [1981] probabilistic algorithm (some of whose ingredients go back to Berlekamp [1970]; see also Knuth [1981, 4.6.2]) with the appropriate modifications for parallel execution. So let F be a finite field with q elements, and $f \in F[x]$ monic with degree $n \geqq 2$. We want to factor f.

In order to get a better timing estimate in case q is not prime, we let $G \subseteq F$ be another field with p elements, and $g \in G[t]$ irreducible of degree d such that $F = G[t]/(g)$ and $q = p^d$. (For any field F, one can of course choose $G = F$ and $p = q$.) F is a vector space over G with basis $1, t, \cdots, t^{d-1}$, and $R = F[x]/(f)$ is a vector space over F with basis $1, x, x^2, \cdots, x^{n-1}$, and a dn-dimensional vector space over G with basis $\{t^i x^j : 0 \leqq i < d, 0 \leqq j < n\}$.

ALGORITHM FACTORIZATION OVER A FINITE FIELD.

Input: A polynomial $f \in F[x]$ of degree n.

Output: Either the complete factorization of f, or "failure."

1. *Frobenius matrix.* Replace f by its (unique) monic scalar multiple. Compute the matrix Q of the linear mapping $R \to R$ with $u \mapsto u^p$. (This is called the Frobenius mapping if p is prime.)
2. *Nullspace.* Compute the dimension r of the nullspace K of $Q - I$, and $g_1, \cdots, g_r \in F[x]$ of degree less than n such that $g_1 \bmod f, \cdots, g_r \bmod f$ form a basis for K. If $r = 1$, set $S = \{f\}$ and go to step 5. (r is the number of distinct irreducible monic factors of f.)
3. *Random nullspace elements.* Let $m = \lceil 5 \log_2 r \rceil$, choose $v_{ij} \in G$ for $1 \leqq i \leqq m$, $1 \leqq j \leqq r$ independently at random, and let $h_i = \sum_{1 \leqq j \leqq r} v_{ij} g_j \in F[x]$ for $1 \leqq i \leqq m$.
4. *Primary factorization.* For $1 \leqq i \leqq m$ compute $c_i = \gcd(f, h_i^{(p-1)/2} - 1) \in F[x]$. If p is even, say $p = 2^k$, use $c_i = \gcd(f, \sum_{0 \leqq j < k} h_i^{2^j})$. Compute the common refinement of these partial factorizations as follows. Let $M = \{0, 1\} \times \{1, \cdots, m\}$. For all $I \subseteq M$ compute

$$s_I = \gcd\left(\{c_i : (0, i) \in I\} \cup \left\{\frac{f}{c_i} : (1, i) \in I\right\}\right).$$

Then compute the following set T of "minimal I's:"

$$T = \{I \subseteq M : s_I \neq 1 \text{ and } \forall J \subseteq M \; I \subseteq J \Rightarrow s_J = 1 \text{ or } s_J = s_I\}.$$

(Note that $I \subseteq J \Rightarrow s_J \mid s_I$.) T can be computed by comparing each pair (s_I, s_J) with $I \subseteq J$ and marking s_I as "irrelevant" if ($s_I = 1$ or ($s_J \neq 1$ and $s_J \neq s_I$)). Then fan in to keep only those I for which s_I has never been marked "irrelevant." Set $S = \{s_I : I \in T\}$. (This eliminates duplicate occurences of $s_I = s_J$ with $I \neq J$. We expect S to contain all "primary" factors of f, i.e. all g^e where g is an irreducible factor of f and e its multiplicity in f.)

5. *Complete factorization.* If $\# S \neq r$, then return "failure". Otherwise, for each $a \in S$ do the following. Set $b = a$. While $b' = db/dx = 0$, replace $b = \sum_{0 \leqq k} b_k x^k$ by its p_0th root $\sum_{0 \leqq k} b_{kp_0}^{q/p_0} x^k$, where $p_0 = \operatorname{char} F$ is a prime number. If $b' \neq 0$, compute $g = b/\gcd(b, b')$. Now g is an irreducible factor of f, and $e = \deg a / \deg g$ its multiplicity.
6. Return the set of all (g, e) computed above as the complete factorizaton of f.

THEOREM 4.1. *Algorithm* FACTORIZATION OVER A FINITE FIELD, *applied to a polynomial of degree n over a finite field with q elements, either returns the complete factorization of the polynomial or reports "failure." The probability of the second case is at most* $\frac{1}{2}$. *The parallel time is* $O(\log^2 n \log q)$ *operations in F. If f is squarefree,* $G \subseteq F$ *a subfield with p elements and* $q = p^d$, *then the cost is* $O(\log^2 n \log^2 (d+1) \log p)$ *operations in G. The number of processors is polynomial in n* log *q.*

Proof. We start by proving the first estimate for the parallel time of the algorithm. So we count operations in F, and consider the matrix Q as an $n \times n$-matrix over F. Step 1 takes $O(\log^2 n \log p)$ and step 2 $O(\log^2 n)$ operations using Lemma 2.1 and the fast parallel algorithm for nullspace from Borodin–von zur Gathen–Hopcroft [1982]. The nullspace algorithm is probabilistic and has a probability of failure at most $2^{-n} \leqq \frac{1}{4}$.

The cost for step 3 is $O(\log r)$. (Note that $r \leqq n$.) In step 4, each c_i can be computed with $O(\log^2 n \log p)$ operations. For each s_I, we apply GCD OF MANY POLYNOMIALS eight times in parallel, with the same number n and using parallel time $O(\log^2 n)$. Unless all these applications fail, we take any of the answers. (They will all agree.) Note that *card* $2^M \leqq 2^{2m} \leqq 2^{12 \log_2 r} = r^{12} \leqq n^{12}$. Thus T and S can be computed in parallel time $O(\log n)$. Finally, each g in step 5 can be computed in time $O(\log_{p_0} n \log q + \log^2 n \log p)$. Thus the total parallel time is $O(\log^2 n \log q)$.

For the second estimate of the parallel time, simply note that one operation in F can be simulated by operations in G in parallel time $O(\log^2 (d+1))$. (We write $d+1$ rather than d in order not to get 0 for $d=1$.) Since now all elements of R are represented by coefficients from G, we consider Q as a $dn \times dn$-matrix over G. The computation of g in step 5 is unnecessary, since f is assumed to be squarefree. We thus get the estimate $O(\log^2 n \log^2 (d+1) \log p)$.

For the proof of correctness, let us recapitulate the overall strategy of the algorithm. In steps 1, 2, 3 we compute $h_1, \cdots, h_m \in F[x]$ such that

$$f \mid h_i^p - h_i = (h_i^{(p-1)/2} - 1)(h_i^{(p+1)/2} + h_i)$$

(if p is odd). Thus each h_i provides a partial factorization $f = c_i \cdot f/c_i$. In step 4 we compute the common refinement of these partial factorizations, and then step 5 accounts for the possible presence of multiple factors. Unless all distinct irreducible factors have been correctly separated in step 4, the procedure will report "failure."

We will show below that the number r from step 2 is the number of distinct irreducible monic factors of f. Then it is clear that the algorithm either reports "failure" at some stage or correctly computes the complete factorization of f. All that remains to do is to estimate the probabilities of failure p_2, p_4, p_5 in steps 2, 4 and 5. We have $p_2 \leqq 2^{-n} \leqq \frac{1}{4}$, and $p_4 \leqq n^{12} 2^{-8n} \leqq \frac{1}{8}$.

In order to estimate p_5, let $f = f_1^{e_1} \cdots f_s^{e_s}$ be the complete factorization of f, where $f_1, \cdots, f_s$ are pairwise distinct irreducible monic polynomials and $e_1, \cdots, e_s \geqq 1$. Let

$$R' = F[x]/(f_1^{e_1}) \times \cdots \times F[x]/(f_s^{e_s}),$$

$$K' = \{(u_1, \cdots, u_s) \in R' : u_1, \cdots, u_s \in G\}.$$

K' is the linear space over G of "locally constant polynomials," i.e. those that are constant ($\in G$) modulo each $f_i^{e_i}$. We denote by $\alpha : R \to R'$ the isomorphism of the Chinese remainder theorem, and by $\bar{h}$ the image of $h \in F[x]$ in R. We first prove that $\alpha(K) = K'$. Let $h \in F[x]$ with $\bar{h} \in K = \{g \in R : g^p - g = 0\}$, and $\alpha(\bar{h}) = (u_1, \cdots, u_s)$. Since $\bar{h}^p - \bar{h} = 0$ in R, we have

$$f \mid h^p - h = \prod_{w \in G} (h - w).$$

These factors are pairwise relatively prime, and it follows that

$$\forall i \leqq s \; \exists w_i \in G \quad f_i^{e_i} \mid h - w_i.$$

Then $u_i = w_i \in G$. Hence $\alpha(\bar{h}) \in K'$. The reverse inclusion is clear, and $\alpha(K) = K'$ follows. In particular, $r = s$, and f is a power of an irreducible polynomial iff $r = 1$. For the remainder of the proof we can assume that $r \geqq 2$.

We first consider the case where p is odd. For $1 \leqq i < j \leqq r$ and $h \in F[x]$ with $\bar{h} \in K$ we say that h separates f_i and f_j iff exactly one of $f_i^{e_i}$ and $f_j^{e_j}$ divides $h^{(p-1)/2} - 1$. Writing $\alpha(\bar{h}) = (u_1, \cdots, u_r)$, this is equivalent to the condition that exactly one of u_i and u_j satisfies $u^{(p-1)/2} - 1 = 0$. The number of pairs $(u_i, u_j) \in G^2$ satisfying this condition is

$$2 \cdot \frac{p-1}{2} \cdot \frac{p+1}{2} = \frac{1}{2}(p^2 - 1),$$

and thus the probability that a randomly chosen $\bar{h} \in K$ does not separate f_i and f_j is $\frac{1}{2}(1 + p^{-2}) \leqq \frac{5}{9}$. The probability that f_i and f_j get separated by none of the randomly chosen $h_1, \cdots, h_m$ is $\leqq (\frac{5}{9})^m$, and the probability that some pair of factors does not get separated is

$$\leqq \binom{r}{2}\left(\frac{5}{9}\right)^m \leqq \frac{r^2}{2}\left(\frac{5}{9}\right)^{5 \log r} \leqq \frac{r^4}{8} \cdot r^{5 \log (5/9)} = \frac{1}{8} r^{4+5 \log (5/9)} \leqq \frac{1}{8}.$$

Thus the total probability of failure is $\leqq p_2 + p_4 + p_5 \leqq \frac{1}{2}$, and Theorem 4.1 is proven for odd characteristic.

If $p = 2^k$ is even, the argument applies with the following modifications: h separates f_i and f_j iff exactly one of $f_i^{e_i}$ and $f_j^{e_j}$ divides $\sum_{0 \leqq j < k} h_i^{2^j}$ iff exactly one of u_i and u_j satisfies $\sum_{0 \leqq j < k} u^{2^j} = 0$. The latter polynomial has exactly 2^{k-1} zeros in G, and the number of pairs (u_i, u_j) that satisfy the above condition is 2^{2k-1}. The probability that $\bar{h}$ does not separate f_i and f_j is $\frac{1}{2}$. As above, it follows that the probability of failure is at most $\frac{1}{2}$. □

Of course, we can execute n instances (say) of the algorithm in parallel and obtain failure probability at most 2^{-n}, with the timing estimate of Theorem 4.1 remaining true. In von zur Gathen–Kaltofen [1983], a fast parallel algorithm for factoring a multivariate polynomial $f \in F[x_1, \cdots, x_r]$ is presented, where F is finite and f is given in a dense encoding.

In our model it would be natural to consider F as fixed, and then we have an $O(\log^2 n)$ algorithm. However, for the factoring problem it is important to consider the field F not as fixed but as somehow described in the input. We might think of the input as consisting of a prime number p, an irreducible polynomial $g \in \mathbf{Z}_p[t]$ of degree d such that $q = p^d$ and $F = \mathbf{Z}_p[t]/(g)$, and $f \in F[x]$, all represented by numbers in binary notation. Then we want an algorithm whose parallel time is polynomial in log (input size), i.e. polynomial in log log q and log n. Unfortunately, the version with $G = F$ of our algorithm falls far short from this goal, with its running time depending on log q rather than log log q. A corresponding unpleasant phenomenon occurs with the sequential deterministic algorithms using time $O(q)$ rather than $O(\log q)$, and the sequential probabilistic methods avoiding this do not translate into a fast parallel algorithm.

The stumbling block is the computation of powers of polynomials modulo f in steps 1 and 4. A related problem — which occurs in step 5 — is the modular computation of powers of numbers: given integers a, n, p compute a^n mod p. The "repeated

squaring" method for computing $a^n \bmod p$ does not make efficient use of parallelism. In particular, even with arbitrarily many processors it is not clear how to obtain parallel running time polynomial in $\log \log n$.

Open question 4.1. Is the above problem of computing powers in finite fields in NC?

For nonprime fields with $\# F = q = p^d$ the second version above goes a step in the right direction, essentially replacing $\log q$ by $\log^2 d \log p$. As an application, consider the algebraic BCH codes, where one wants to factor polynomials over large finite fields of characteristic 2; the parallel time of our algorithm is indeed polynomial in log (input size) for this problem. In particular, if we want to factor polynomials of degree n over $GF(2^n)$, then our Las Vegas algorithm provides an $O(\log^4 n)$ solution.

It seems to be a fundamental question what one can say about factors of polynomials over finite fields with the parallel time depending polynomially on log log (field size). Obviously a composite polynomial has a certificate (namely, a factorization) that can be verified in parallel time $O(\log n \log \log q)$. Also an irreducible polynomial f has such a certificate: for each $i < n = \deg f$, give $g_i \in F[x]$ with $x^{ip} = g_i \bmod f$ and $\deg g_i < n$. The first condition can be certified by a sequence of intermediate results in a computation for $x^{ip} \bmod f$. From the coefficients of the g_i's one can verify that the rank of {fix points of the Frobenius mapping} is $n-1$, thus proving irreducibility of f. (This is a special case of the general observation that problems in the Boolean class NP seem to be computable by $\log n$-depth polynomial-size nondeterministic circuits.)

Open question 4.2. Are the irreducible polynomials and the factoring problem, both over finite fields, in NC?

In the sequential analogue of this situation, step 1 and a test "$r = 1$?" in step 2 of the above algorithm provide a polynomial-time deterministic irreducibility test for polynomials over finite fields, going back to Berlekamp [1967]. The parallel time for this algorithm is $O(\log^2 nd \log p)$.

The recent breakthrough by Lenstra–Lenstra–Lovász [1982] provides a polynomial-time factorization procedure for univariate integer polynomials; it has been extended by Kaltofen [1982] to multivariate integer polynomials. A basic subroutine is to compute short vectors in integer lattices. From the point of view of this paper, it remains to adapt these algorithms to the parallel setting. In § 7 we will see that they give rise to reductions to the problem of finding short vectors; however, we do not have a good parallel algorithm for the latter problem.

5. Squarefree decomposition. Let F be an arbitrary field and $f \in F[x]$. f is called squarefree if there does not exist an $h \in F[x] \setminus F$ such that h^2 divides f. Let c be the leading coefficient of f, and $g = (g_1, \cdots, g_t)$ be a sequence of monic squarefree polynomials from $F[x]$ with $g_t \neq 1$. We call g the monotone squarefree decomposition of f if $f = c g_1 g_2 \cdots g_t$ and g_{i+1} divides g_i for $1 \leqq i < t$. This decomposition is unique, and g_1 is called the "squarefree part" of f. We call g the distinct power decomposition of f if $f = c g_1 g_2^2 \cdots g_t^t$ and $\gcd(g_i, g_j) = 1$ for $1 \leqq i < j \leqq t$. This decomposition is unique, too. The next two lemmas show that these two decompositions are closely related, and how to compute the decomposition of a product from the decompositions of the factors.

LEMMA 5.1. *If $(g_1, \cdots, g_t)$ is the monotone squarefree decomposition and $(h_1, \cdots, h_s)$ the distinct power decomposition of f, then $s = t$, $h_i = g_i / g_{i+1}$ (with $g_{t+1} = 1$) and $g_i = h_i h_{i+1} \cdots h_t$ for $1 \leqq i \leqq t$. In particular, one decomposition can be computed from the other in parallel time $O(\log^2 n)$.* □

LEMMA 5.2. *Let* $(f_1, \cdots, f_r)$ *and* $(g_1, \cdots, g_s)$ *be the monotone squarefree decompositions of* $f, g \in F[x]$. *For* $j, k \geqq 0$, $1 \leqq i \leqq r+s$ *set*

$$u_{jk} = \gcd(\operatorname{lcm}(f_j, g_j), \gcd(f_k, g_k)),$$

$$h_i = \operatorname{lcm}(\{u_{jk}: 0 \leqq j, k \text{ and } i \leqq j+k\}),$$

$$t = \max\{i: h_i \neq 1\}$$

(*with* $f_j = 1$ *for* $j > r$, $g_k = 1$ *for* $k > s$, *and* $f_0 = g_0 = fg$). *Then* $(h_1, \cdots, h_t)$ *is the monotone squarefree decomposition of* fg.

Proof. Each u_{jk} with $(j, k) \neq (0, 0)$ is squarefree, hence also each h_i. Also $h_{i+1} \mid h_i$. Let $a \in F[x]$ be irreducible, p, q the multiplicities of a in f, g, respectively. Thus, e.g. $a \mid f_j$ iff $j \leqq p$. It is sufficient to show that $a \mid h_{p+q}$ and $a \nmid h_{p+q+1}$. We can assume that $p \geqq q$, and then $a \mid \operatorname{lcm}(f_p, g_p)$ and $a \mid \gcd(f_q, g_q)$. Hence $a \mid u_{pq}$ and $a \mid h_{p+q}$. If $k > q$, then $a \nmid \gcd(f_k, g_k)$. Hence if $j + k > p + q$, then $a \nmid u_{jk}$. It follows that $a \nmid h_{p+q+1}$. □

Algorithms for computing the squarefree decomposition are of interest since they may yield an (incomplete) factorization with little effort, and since most factorization algorithms over $\mathbf{Q}$ and finite fields require squarefree polynomials as inputs (not, however, the first version of the algorithm of the previous section). Recall that one loop of the standard sequential squarefree factoring algorithms computes the squarefree part $f/\gcd(f, f')$ of f and passes $\gcd(f, f')$ to the next loop (see Knuth [1981, 4.6.2]). For a parallel algorithm over a field of characteristic zero, we use the following approach.

ALGORITHM SQUAREFREE.

Input: A number $n \in \mathbf{N}$, and a polynomial $f \in F[x]$ of degree at most n, where F is a field of characteristic zero.

Output: either the monotone squarefree decomposition $g = (g_1, \cdots, g_t)$ of f, or "failure."

1. For all i, j, $1 \leqq i < j \leqq n$, input $\prod_{i \leqq k \leqq j} k \in F$.
2. For all i, $0 \leqq i \leqq n$, compute

$$u_i = \frac{d^i f}{dx^i},$$

$$v_i = \gcd(u_0, \cdots, u_i),$$

$$g_i = v_{i-1}/v_i \quad (\text{for } i \geqq 1).$$

3. Set $t = \max\{i: 1 \leqq i \leqq n \text{ and } g_i \neq 1\}$.
4. Return $g = (g_1, \cdots, g_t)$.

THEOREM 5.3. *Algorithm* SQUAREFREE *either computes the monotone squarefree decomposition of the input polynomial, or reports "failure." The probability of the second case is less than* 2^{-n}. *The algorithm runs in parallel time* $O(\log^2 n)$. *If the field is real, then the algorithm can be performed deterministically.*

Proof. We first prove the bounds on the running time and failure probability, and then correctness. In step 2, we apply for each $i \geqq 1$ algorithm GCD OF MANY POLYNOMIALS twice to compute v_i, using the same number n as degree bound. The parallel time is $O(\log^2 n)$. SQUAREFREE reports "failure," if for some i both applications fail. (If neither fails, they will return the same value for v_i.) The total failure probability is at most $n2^{-2n} < 2^{-n}$.

For the proof of correctness, we can assume that f is monic, and let $f = f_1^{e_1} \cdots f_r^{e_r}$ be the factorization of f, where $f_1, \cdots, f_r$ are pairwise distinct irreducible monic

polynomials, and $e_1, \cdots, e_r \geqq 1$. Fix some $k \geqq r$, and let

$$a_i = (f_k')^i \prod_{e_k - i < j \leqq e_k} l \prod_{\substack{1 \leqq f \leqq r \\ f \neq k}} f_f^{e_f}.$$

Then $f_k \nmid a_i$, and it is easy to see by induction on i that

$$f_k^{e_k - i - 1} \mid u_i - f_k^{e_k - i} a_i$$

for $0 \leqq i \leqq e_k$. It follows that the multiplicity of f_k in u_i and v_i is exactly $e_k - i$ for $0 \leqq i \leqq e_k$, and in g_i exactly 1 for $1 \leqq i \leqq e_k$ and 0 for $i > e_k$. Hence $g = (g_1, \cdots, g_r)$ is the monotone squarefree decomposition of f. □

Again, for an arbitrary field of characteristic zero, the algorithm can be performed deterministically in parallel time $O(\log^3 n)$.

6. Squarefree decomposition over finite fields. Let F be a field with q elements, $p = \text{char } F$, $q = p^d$, and $f \in F[x]$ of degree at most n. We want to compute the squarefree decomposition of f. If our goal are $\log^2$ (input size)-algorithms and $q > p$ are both large, then a hard case is an innocent-looking polynomial like $f = x^p - a$. We only know how to compute $b = a^{q/p}$ (so that $f = (x-b)^p$) in parallel time $O(\log(q/p))$ which, just as in § 4, is in general not polynomial in log (input size) (see Open Question 4.1). For the estimates below, we let T_F be an upper bound on the parallel time to compute $a^p, a^{p^2}, \cdots, a^q$ for $a \in F$. Thus $T_F = O(\log(q/p))$. The algorithms of this section can be considered as reductions to this powering problem and systems of linear equations.

We first present an adaptation, called SQUAREFREE VIA DERIVATIVES, of the algorithm of § 5, which works for polynomials over a finite field. The only complication that requires some extra care is that dv_i/dx may be zero without v_i being constant. We will then have to compute $h = (v_i)^{1/p}$ at parallel cost $O(\log(q/p))$ and restart the algorithm with input h. Since this situation may occur several times consecutively, we only get $O((\log^2 n + \log(q/p)) \log_p n)$ as estimate for the parallel time. We then describe a second algorithm, SQUAREFREE VIA TAYLOR COEFFICIENTS, that avoids this factor $\log_p n$.

We thus have four parallel Las Vegas algorithms A_1, A_2, A_3, A_4 to compute the squarefree factorization. A_1 is the factoring algorithm of § 4 with $G = F$. A_2 is the second version of that factoring algorithm, with $G = \mathbf{Z}_p$, and step 5 of A_1 executed on the primary factors. A_3 and A_4 are SQUAREFREE VIA DERIVATIVES and SQUAREFREE VIA TAYLOR COEFFICIENTS, repectively. The parallel computing times are as follows:

$$T(A_1) = O(\log^2 n \log q),$$

$$T(A_2) = O(\log^2 n \log^2 (d+1) \log p + T_F),$$

$$T(A_3) = O((\log^2 n + T_F) \log_p n),$$

$$T(A_4) = O(\log^2 n + T_F).$$

Thus, A_4 has the best uniform timing estimate among these algorithms, comparing particularly well with A_3 if p is small, and with A_1 and A_2 if $F = \mathbf{Z}_p$ for large p. (Of course, A_1 and A_2 provide much more information than A_3 or A_4 do.)

A somewhat surprising example is provided by polynomials of the form $x^2 - a \in F[x]$, where $\# F = 2^n$; then T_F seems to be $\Omega(n)$, and all four algorithms use parallel time $\Omega(n)$. Therefore we have the counterintuitive consequence that "squarefree decomposition" can be harder in parallel than "factoring squarefree polynomials"

over finite fields, while in characteristic zero, the first problem is always easy and the second one not even known to be in NC.

ALGORITHM SQUAREFREE VIA DERIVATIVES.

Input: A number $n \in \mathbf{N}$, and $f \in F[x]$ of degree at most n, where F is a finite field of characteristic p.

Output: Either the monotone squarefree decomposition g of f, or "failure."

1, 2, 3. Execute steps 1, 2, 3 of algorithm SQUAREFREE.

4. If $v_t' \neq 0$, then return $g = (g_1, \cdots, g_t)$. If $v_t' = 0$, then compute $h \in F[x]$ such that $v_t = h^p$. (Note that such an h exists.)

5. Call SQUAREFREE VIA DERIVATIVES recursively with input $(\lceil n/p \rceil, h)$ to obtain the monotone squarefree decomposition $(h_1, \cdots, h_s)$ of h.

6. Use Lemma 5.2 to merge the two monotone squarefree decompositions $(g_1, \cdots, g_t)$ of f/h^p and $(h_1, \cdots, h_1, \cdots, h_s, \cdots, h_s)$ of h^p (with each h_i written p times), and return the monotone squarefree decomposition of f.

We say that a polynomial $f \in F[x]$ is p-power-free if there does not exist an $h \in F[x] \backslash F$ such that h^p divides f. Thus 2-power-free is the same as squarefree.

THEOREM 6.1. *Let F be a finite field with q elements and* char $F = p$, *and $f \in F[x]$ of degree at most n. Algorithm* SQUAREFREE VIA DERIVATIVES *either returns the monotone squarefree decomposition of f, or reports "failure." The latter occurs with probability less than 2^{-n}. The algorithm runs in parallel time $O((\log^2 n + \log(q/p)) \log_p n)$. If f is p-power-free, then it runs in parallel time $O(\log^2 n)$. The number of processors is polynomial in n.*

Proof. We leave the proof to the reader. Observe that steps 2, 5 and 6 have to be executed several times in parallel in order to get the estimate on the failure probability. □

To highlight the difference between the two algorithms presented in this section, let us first trace SQUAREFREE VIA DERIVATIVES on input $x^8 + x^7 + x^4 + x^3 \in \mathbf{Z}_2[x]$.

Step 2. $u_0 = x^8 + x^7 + x^4 + x^3$,
$u_1 = x^6 + x^2$,
$u_i = 0$ for $2 \leqq i \leqq 8$,
$v_0 = x^8 + x^7 + x^4 + x^3$,
$v_i = x^6 + x^2$ for $1 \leqq i \leqq 8$,
$g_1 = x^2 + x$,
$g_i = 1$ for $1 \leqq i \leqq 8$.

Step 3. $t = 1$.

Step 4. $h = x^3 + x$.

Step 5. $(h_1, h_2) = (x^2 + x, x + 1)$ by recursive application.

Step 6. Merge $(x^2 + x)$ with $(x^2 + x, x^2 + x, x + 1, x + 1)$ to get the output $(x^2 + x, x^2 + x, x^2 + x, x + 1, x + 1)$.

We now want to discuss a different approach to squarefree decomposition. In SQUAREFREE VIA DERIVATIVES, we compute pth roots from time to time, and apply the procedure recursively. Algorithm SQUAREFREE VIA TAYLOR COEFFICIENTS first computes the p-power-parts $w_0, \cdots, w_m$ of f, where $f = w_0 w_1^p w_2^{p^2} \cdots w_m^{p^m}$ and each w_i is p-power-free. This approach avoids the factor $\log_p n$ in the parallel time.

For $i, n \geqq 0$ define $(x^n)^{[i]} = \binom{n}{i} x^{n-i}$. Extending this by linearity, we get a mapping ${}^{[i]}: F[x] \to F[x]$. Thus $d^i f/dx^i = i! f^{[i]}$, $f^{[0]} = f$, and e.g. $(x^8+x^7+x^4+x^3)^{[2]} = x^5+x \in \mathbf{Z}_2[x]$. Clearly the $f^{[i]}$'s are more appropriate for computations than the derivatives, since for any $f \in \mathbf{Z}_2[x]$, $d^2 f/dx^2$ and all further derivatives are zero and thus carry no information at all. The $f^{[i]}$'s are the "universal Taylor coefficients" of f, since from them we get the Taylor expansion of f at an arbitrary point a as follows:

$$f = \sum_{0 \leqq i \leqq \deg f} f^{[i]}(a)(x-a)^i.$$

ALGORITHM SQUAREFREE VIA TAYLOR COEFFICIENTS.

Input: A number $n \in \mathbf{N}$, and a polynomial $f \in F[x]$ of degree at most n, where F is a finite field of characteristic p.

Output: Either the monotone squarefree decomposition $(g_1, \cdots, g_t)$ of f, or "failure".

1. For all i, j, $0 \leqq j \leqq i \leqq n$, input $\binom{i}{j} \in F$.
2. For all i, $0 \leqq i \leqq n$, compute $f^{[i]}$.
3. Set $l = \lfloor \log_p n \rfloor$. For all k, $0 \leqq k \leqq l$, compute

$$u_k = \gcd(f^{[0]}, \cdots, f^{[p^k-1]}),$$

$$v_k = \frac{u_k}{u_{k+1}},$$

(with $u_{l+1} = 1$. v_k will be the "p^k-power-part of f" in the following sense: v_k will divide f and be equal to $w_k^{p^k}$ for some w_k, and for no irreducible factor v of v_k will $v^{p^{k+1}}$ divide f.)

$$w_k = v_k^{1/p^k},$$

$$y_k := (y_{k1}, \cdots, y_{k,t_k}),$$

the monotone squarefree decomposition of w_k. (Using steps 1, 2, 3 of algorithm SQUAREFREE. Note that w_k is "p-power-free" and the complication mentioned at the beginning of this section does not turn up.)

4. For all i, $1 \leqq i \leqq n$, do the following. Consider the p-adic representation $i = i_0 + i_1 p + \cdots + i_l p^l$ of i, where $0 \leqq i_k < p$. Compute

$$z_i = \gcd(y_{0,i_0}, y_{1,i_1}, \cdots, y_{l,i_l}),$$

(with $y_{kj} = 1$ if $j > t_k$, and $y_{k,0} = f$ for all k.)

$$g_i = \operatorname{lcm}(z_i, z_{i+1}, \cdots, z_n).$$

5. Set $t = \max\{i : g_i \neq 1\}$ and return $g = (g_1, \cdots, g_t)$.

Before we analyse the algorithm, let us look at the example considered above. With $f = x^8+x^7+x^4+x^3 \in \mathbf{Z}_2[x]$ as input, SQUAREFREE VIA TAYLOR COEFFICIENTS produces the following:

Step 2. $f^{[0]} = x^8+x^7+x^4+x^3$, $f^{[5]} = x^2$,
$f^{[1]} = x^6+x^2$, $f^{[6]} = x$,
$f^{[2]} = x^5+x$, $f^{[7]} = 1$,
$f^{[3]} = x^4+1$, $f^{[8]} = 1$.
$f^{[4]} = x^3+1$,

Step 3. Set $l=3$, and

$$\begin{aligned} u_0 &= x^8+x^7+x^4+x^3, & v_0 &= x^2+x, \\ u_1 &= x^6+x^2, & v_1 &= x^2, \\ u_2 &= x^4+1, & v_2 &= x^4+1, \\ u_3 &= 1, & v_3 &= 1, \\ w_0 &= x^2+x, & y_0 &= (x^2+x), \\ w_1 &= x, & y_1 &= (x), \\ w_2 &= x+1, & y_2 &= (x+1), \\ w_3 &= 1, & y_3 &= (1). \end{aligned}$$

Step 4.

$$\begin{aligned} z_1 &= x^2+x, & & \\ z_2 &= z_3 = x, & g_1 &= g_2 = g_3 = x^2+x, \\ z_4 &= z_5 = x+1, & g_4 &= g_5 = x+1, \\ z_6 &= z_7 = z_8 = 1, & g_6 &= g_7 = g_8 = 1. \end{aligned}$$

Step 5. Return $g=(x^2+x, x^2+x, x^2+x, x+1, x+1)$.

THEOREM 6.2. *Let F be a finite field with q elements, $p=\operatorname{char} F$, and $f \in F[x]$ of degree n. With this input, algorithm* SQUAREFREE VIA TAYLOR COEFFICIENTS *either returns the squarefree decomposition of f or it reports "failure". The probability of the second case is less than 2^{-n}. The algorithm runs in parallel time $O(\log^2 n + \log_2 (q/p))$. If f is p-power-free, then it runs in parallel time $O(\log^2 n)$. The number of processors is polynomial in n.*

Proof. We first describe the execution of the algorithm in some more detail, simultaneously proving the time bound. Then we establish the bound on the failure probability, and finally correctness.

Steps 1 and 2 can be performed in parallel time $O(\log n)$. In step 3, we apply algorithm GCD OF MANY POLYNOMIALS three times for each k to compute u_k. Below we prove that v_k is a p^kth power, and hence can be written as $v_k = \sum_{0 \le i} v_{ki} x^{ip^k}$. Then $w_k = \sum_{0 \le i} v_{ki}^{q^m/p^k} x^i$ can be computed in parallel time $O(\log (q/p))$, where $q^{m-1} < p^k \le q^m$, using $O(n)$ processors. For y_k, we use steps 1, 2, 3 of SQUAREFREE, applying it three times for each k. The algorithm runs in parallel time $O(\log^2 n)$. In step 4, for each i we apply GCD OF MANY POLYNOMIALS three times, and LCM OF MANY POLYNOMIALS three times for each g_i. The overall parallel time is $O(\log^2 n + \log (q/p))$, as claimed. If f is p-power free, then $v_1 = \cdots = v_l = 1$, and no p^kth roots with $k \ge 1$ have to be computed.

Failure may occur only in the computation of u_k, y_k, z_i, and g_i. In every single such computation, the failure probability is $\le 2^{-3n}$. The total failure probability is at most $(2l+2n)2^{-3n} \le 4n2^{-3n} < 2^{-n}$.

For the proof of correctness, we assume that f is monic and let $f = h_0 h_1^p h_2^{p^2} \cdots h_l^{p^l}$ be the "p-power-decomposition" of f, where each $h_i \in F[x]$ is monic and p-power-free. This decomposition exists and is unique. Fix some k, $0 \le k \le l$, and write $r = h_0 \cdots h_{k-1}^{p^{k-1}}$, $s = h_k^{p^k} \cdots h_l^{p^l}$. Then $f = rs$, r is p^k-power-free, and s is a p^kth power. We now show that $u_k = s$. For $0 \le i < p^k$, we have

$$f^{[i]} = (rs)^{[i]} = \sum_{0 \le j \le i} r^{[j]} s^{[i-j]} = r^{[i]} s^{[0]} = r^{[i]} s,$$

using Lemma 6.3(i) and (ii). Hence $u_k = s \cdot \gcd (r^{[0]}, \cdots, r^{[p^k-1]})$. Lemma 6.3(v) implies that $\gcd (r^{[0]}, \cdots, r^{[p^k-1]}) = 1$, and hence $u_k = s$. It follows that for all k we have $v_k = h_k^{p^k}$ and $w_k = h_k$.

Now each $y_{k,j}$ (with $j \geqq 1$) is squarefree, hence also every z_i and g_i is squarefree. Also g_{i+1} divides g_i. In order to show that g is the monotone squarefree decomposition of f, we only have to prove that $f = g_1 \cdots g_r$. So let $a \in F[x]$ be an irreducible monic factor of f, and m its multiplicity in f (so that $a^m | f$ and $a^{m+1} \nmid f$). We have to show that a divides g_i iff $i \leqq m$. Using the p-adic representation $m = m_0 + m_1 p + \cdots + m_l p^l$ of m, where $0 \leqq m_k < p$, it follows from the above that a divides h_k iff $m_k \neq 0$. More precisely, a divides $h_k = w_k$ exactly with multiplicity m_k, and hence

$$a \mid y_{k,j} \Leftrightarrow j \leqq m_k.$$

It follows that a divides z_m and hence g_m. Then a also divides each of $g_1, \cdots, g_m$, and it is sufficient to prove that a does not divide g_{m+1}. But for any $i = i_0 + i_1 p + \cdots + i_l p^l > m$ some p-adic coefficient i_k is greater than m_k, hence a does not divide y_{k,i_k}, and also not z_i or g_i. It follows that the multiplicity of a in $g_1 \cdots g_i$ is exactly m, and hence $f = g_1 \cdots g_r$. □

We remark that in our model the binomial coefficients computed in step 1 are considered as constants in F, and hence are given for free. However, they can also be computed in F, just using the constants 0 and 1 and field operations, either by computing $(x+1)^i$ or by Lucas' [1877] formula (see also Fine [1947]):

$$\binom{i}{j} \equiv \binom{i_k}{j_k} \cdots \binom{i_0}{j_0} \bmod p,$$

if $i = i_0 + i_1 p + \cdots + i_l p^l$ with $0 \leqq i_k < p$ is given in p-adic representation, and similarly for j. Note that the definition via factorials may fail to give a computation in positive characteristic.

In the following lemma we collect the facts concerning $f^{[i]}$ that were used in the preceding proof. It may be interesting to compare with the properties of the ith derivative $f^{(i)}$. Statements (ii) and (iii) are also true in that case, (iv) and (v) are false in general, and (i) has to be replaced by the Leibniz rule

$$(fg)^{(i)} = \sum \binom{i}{j} f^{(j)} g^{(i-j)}.$$

LEMMA 6.3. *Let F be a field of characteristic $p > 0$, $f, g \in F[x]$, and $i \geqq 0$. Then*

(i) $(fg)^{[i]} = \sum_{0 \leqq j \leqq i} f^{[j]} g^{[i-j]}$.
(ii) $(f^{p^i})^{[j]} = 0$ *for* $0 < j < p^i$.
(iii) $f \mid (f^i)^{[j]}$ *for* $0 \leqq j < i$.
(iv) *If f is irreducible and $f' \neq 0$, then* $\gcd(f, (f^i)^{[i]}) = 1$.
(v) *If F is finite and f is p^i-power-free, then* $\gcd(f^{[0]}, f^{[1]}, \cdots, f^{[p^i-1]}) = 1$.

Proof. (i) The case $f = x^m$, $g = x^n$ follows from comparing coefficients of x^{n+m-i} in

$$\sum_{0 \leqq l \leqq m+n} (fg)^{[l]} = \sum_{0 \leqq l \leqq m+n} \binom{n+m}{l} x^{n+m-l} = (1+x)^{n+m}$$

$$= (1+x)^n (1+x)^m = \sum_{j,k} \binom{n}{j} x^{n-j} \binom{m}{k} x^{m-k} = \sum_{j,k} f^{[j]} g^{[k]}.$$

Since both sides are bilinear, (i) also follows for arbitrary f, g.

(ii) For $f = ax^n$ and $0 < j < p^i$ with $a \in F$, (ii) follows from $\binom{np^i}{j} = 0$. For general f, (ii) now follows from the additivity of the left-hand side.

(iii) We use induction on i, the case $i = 0$ or $i = 1$ being trivial. For $i > 0$ we write

$$(f^i)^{[j]} = (f^{i-1} f)^{[j]} = \sum_{0 \leqq l \leqq j} (f^{i-1})^{(l)} f^{[j-l]},$$

using (i). By the induction hypothesis, f divides each summand with $l < i-1$. The only summand that is possibly left is $(f^{i-1})^{[i-1]}f^{[0]} = (f^{i-1})^{[i-1]}f$, which is also divisible by f.

(iv) We use induction on i and write

$$h = (f^i)^{[i]} = (f^{i-1}f)^{[i]} = \sum_{0 \leqq j \leqq i} (f^{i-1})^{[j]}f^{[i-j]}.$$

Then f divides all summands except the one for $j = i-1$, and hence

$$\gcd(f, h) = \gcd(f, (f^{i-1})^{[i-1]}f^{[1]}) = \gcd(f, f') = 1.$$

(v) Assume that there exists an irreducible $g \in F[x]$ such that g divides $f^{[j]}$ for $0 \leqq j < p^i$. We show by induction on j for $0 \leqq j < p^i$ that $g^{j+1} | f$. This yields the desired contradiction for $j = p^i - 1$. The claim is clear for $j = 0$. For $j \geqq 1$, we can write $f = g^j h$ with some $h \in F[x]$ by induction hypothesis. Then g divides $f^{[j]} = \sum_{0 \leqq l \leqq j} (g^j)^{[l]}h^{[j-l]}$ and also by (iii) each summand with $0 \leqq l < j$. Hence g divides $(g^j)^{[j]}h^{[0]}$, and from (iv) we conclude that $g | h$ and $g^{j+1} | f$. □

We remark that in (v) it is sufficient to assume F perfect, and in (iv) that F is perfect and f squarefree. Without some such assumption, the conclusions may not hold: If $a \in F$ is not a pth power, then for $f = x^{p+1} - ax$ and $i = 1$ we have $\gcd(f^{[0]}, f^{[1]}, \cdots, f^{[p^i-1]}) = x^p - a \neq 1$.

Algorithm SQUAREFREE VIA TAYLOR COEFFICIENTS will work over any perfect field of characteristic $p > 0$, provided that we have an effective procedure for extracting pth roots. The question of squarefreeness over arbitrary fields is undecidable (von zur Gathen [1984]).

7. Some reductions. The previous sections have left open parallel versions for a number of factorization problems that have good sequential solutions. The two most important ones—concerning boolean circuits—are the gcd of two integers (see Open Question 1 in Borodin–von zur Gathen–Hopcroft [1982]) and:

Open question 7.1. Can univariate and multivariate integer polynomials be factored fast in parallel?

The univariate factorization problem would probably be attacked along the lines of Lenstra–Lenstra–Lovász [1982] via computing short vectors in $\mathbf{Z}$-modules. For multivariate polynomials, one might use Kaltofen's [1982], [1983a] reductions. We now give the parallel versions of these reductions, and also reduce the integer gcd's to a special case of the short vector problem.

In this section, we are concerned with the circuit (=parallel boolean) complexity (rather than algebraic complexity over an arbitrary field) of functions; see Cook [1983] for an excellent overview of this theory. Our notion of reduction comes from Cook's paper: A boolean function f is NC-reducible to another function g if there exists a U_{E^*}-uniform family $(\alpha_n)_{n \in \mathbf{N}}$ of boolean circuits—of depth $O(\log^k n)$ for some $k \in \mathbf{N}$—where α_n computes f on inputs of length n and is allowed to have oracle nodes for g. An oracle node for g has input edges $x_1, \cdots, x_r$ and output edges $y_1, \cdots, y_s$ whose values satisfy $g(x_1, \cdots, x_r) = (y_1, \cdots, y_s)$. Such an oracle node contributes $\lceil \log r \rceil$ to the depth of the circuit.

The functions INTEGER GCD, UNIVARIATE FACTORIZATION OVER $\mathbf{Q}$, and MULTIVARIATE FACTORIZATION OVER $\mathbf{Q}$ compute the (nonnegative) gcd of two integers, the complete factorization of a polynomial in $\mathbf{Q}[x]$, and of a polynomial in $\mathbf{Q}[x_1, \cdots, x_n]$, respectively. For the last problem, the input size is given by the length of a dense encoding of the polynomial. For the sparse encoding, a probabilistic sequential polynomial-time algorithm is known (von zur Gathen [1983a]), but a fast parallel version of that reduction to the bivariate case is open. A function

is called SHORT VECTORS if it takes as input vectors $a_1, \cdots, a_n \in \mathbf{Z}^n$, linearly independent over $\mathbf{Q}$, and returns a vector x in the $\mathbf{Z}$-module ("lattice") $M = \sum a_i \mathbf{Z} \subseteq \mathbf{Z}^n$ such that

$$\forall y \in M \backslash \{0\} \quad |x| \leqq 2^{(n-1)/2} |y|.$$

Thus x is a shortest vector in M, up to the factor $2^{(n-1)/2}$ arising in the work of Lenstra–Lenstra–Lovász, with respect to the L_2-norm $|y| = (\sum y_i^2)^{1/2}$. A function is called SHORT VECTORS IN DIMENSION TWO if it takes $a_1, a_2 \in \mathbf{Z}^2$ as inputs and produces an x as above; instead of the factor $2^{(2-1)/2}$ we allow an arbitrary constant c.

THEOREM 7.1. 1. UNIVARIATE FACTORIZATION OVER $\mathbf{Q}$ *is* NC-*reducible to* SHORT VECTORS.

2. MULTIVARIATE FACTORIZATION OVER $\mathbf{Q}$ *is* NC-*reducible to* UNIVARIATE FACTORIZATION OVER $\mathbf{Q}$.

3. INTEGER GCD *is* NC-*reducible to* SHORT VECTORS IN DIMENSION TWO.

Proof. 1. and 2. follow from the reductions of Lenstra–Lenstra–Lovász [1982] and Kaltofen [1983a], using the results of the previous sections. Note that one has to use a quadratic Hensel iteration instead of the more common linear one. One might either lift each factor separately and discard duplicate ones at the end, or one might lift all irreducible factors mod p simultaneously to factors mod p^k (p, k as usual). (See e.g. von zur Gathen [1984] for formulas for this lifting.)

For the reduction in 3., let c be the constant of the algorithm SHORT VECTORS IN DIMENSION TWO. We can assume $c \in \mathbf{N}$. In order to compute the gcd of a, $b \in \mathbf{Z}$, we can assume that a is positive, and associate to a, b the two vectors $u = (a(ac+1), 0)$, $v = (b(ac+1), 1) \in \mathbf{Z}^2$. If $x = (x_1, x_2)$ is a short vector in the $\mathbf{Z}$-module $M = u\mathbf{Z} + v\mathbf{Z} \subseteq \mathbf{Z}^2$, i.e.

$$\forall y \in M \backslash \{0\} \quad |x| \leqq c|y|,$$

then we compute $g = \gcd(a, b)$ as follows.

1. Set $S = \{i \in \mathbf{Z}: 1 \leqq i \leqq c,\ x_2/i \in \mathbf{Z},\ x_2 b/ai \in \mathbf{Z}\}$.
2. Set $m = \max S$. (We will show that $S \neq \emptyset$.)
3. Return $|am/x_2|$ as $\gcd(a, b)$.

We now make two claims:

(i) $\exists k \in \mathbf{Z}$, $0 < |k| \leqq c$ and $x = (0, ka/g)$. (This k is unique.)

(ii) $k = m$.

Then $g = am/x_2$, and the correctness of the algorithm follows. To prove claim (i), let

$$z = -\frac{b}{g} u + \frac{a}{g} v = \left(0, \frac{a}{g}\right) \in M \backslash \{0\}.$$

It is sufficient to show that for $w = (w_1, w_2) = su + tv \in M$ we have

$$|w| \leqq c|z| \Leftrightarrow \exists k \in \mathbf{Z}, 0 \leqq |k| \leqq c \text{ and } w = kz.$$

(In particular, $\pm z$ are the two shortest nonzero vectors in M, and any element of M in the circle around 0 of radius $c|z|$ is an integral multiple of z.) "$\Leftarrow$" is clear. For "$\Rightarrow$", assume $|w| \leqq c|z|$. Then

$$|(sa + tb)(ac+1)| = |w_1| \leqq |w| \leqq c|z| = c\frac{a}{g},$$

and $ac + 1 > ca/g$, hence $sa + tb = 0$. The only solutions s, t of this equation are of the form

$$s = k - \frac{b}{g}, \qquad t = k\frac{a}{g},$$

for some $k \in \mathbf{Z}$. Then $w = su + tv = (0, ka/g)$. From

$$|k|\frac{a}{g} = |w_2| = |w| \leqq c|z| = c\frac{a}{g}$$

we get $|k| \leqq c$.

In order to prove claim (ii), we use from claim (i) that there exists $k \in \mathbf{Z}$ such that $0 < |k| \leqq c$ and $x = (0, ka/g)$. It is sufficient to show that $S = \{i \in \mathbf{Z}: i \mid k\}$. The inclusion "$\supseteq$" is clear. For "$\subseteq$", let $i \in S$ and $p \in \mathbf{N}$ prime, and e_a, e_b, e_g, e_i, e_k be the multiplicities of p in a, b, g, i, k resp. Then $e_g = \min\{e_a, e_b\}$. If $e_g = e_a$, then from $ka/ig = x_2/i \in \mathbf{Z}$ we get $e_k + e_a \geqq e_i + e_g$ and $e_k \geqq e_i$. If $e_g = e_b$, then from $kb/ig = x_2 b/ai \in \mathbf{Z}$ we get $e_k + e_b \geqq e_i + e_g$ and $e_k \geqq e_i$. In either case, the multiplicity of p in k is not less than its multiplicity in i. It follows that i divides k. □

We remark that in the reduction for 3. only two special cases of integer division were used: for $a, b \in \mathbf{Z}$, test whether a divides b, and if it does, compute the quotient. Reif [1983] has shown that the quotient and remainder of two n-bit integers can be computed in $O(\log n \log^2 \log n)$ parallel bit operations, and Beame–Cook–Hoover [1984] have improved the parallel time to $O(\log n)$ with a slightly weaker uniformity property.

Open question 7.2. Can short vectors in $\mathbf{Z}$-modules be computed fast in parallel?

8. Conclusion. We have shown that a number of algebraic problems with polynomial-time sequential solutions have polynomial-log-time parallel solutions. The basic routines are those of linear algebra; all other problems get reduced to these.

The most important open questions are the factorization of integer polynomials, and the analogous sequential vs. parallel behaviour for integer problems, e.g. computing the gcd of two n-bit integers with $\log^{O(1)} n$ bit operations in parallel. We have reduced this problem to a subroutine that is likely to be employed in factoring integer polynomials.

In von zur Gathen [1983b] we show in a general framework that problems like Padé approximation, partial fraction decomposition (with factored denominators), and various interpolation problems also have a fast parallel solution. Ongoing work at Toronto has resulted in a fast parallel factorization procedure for multivariate polynomials over finite fields (von zur Gathen–Kaltofen [1983]) and an irreducibility test for multivariate polynomials over $\mathbf{C}$ (Kaltofen [1983b]).

REFERENCES

P. W. Beame, S. A. Cook and H. J. Hoover, *Log depth circuits for division and related problems*, Proc. 25th IEEE Symposium on Foundations of Computer Science, Singer Island, FL, 1984.

S. J. Berkowitz, *On computing the determinant in small parallel time using a small number of processors*, Inform. Process. Lett. 18 (1984), pp. 147–150.

E. R. Berlekamp, *Factoring polynomials over finite fields*, Bell System Tech. J., 46 (1967), pp. 1853–1859.

———, *Factoring polynomials over large finite fields*, Math. Comp., 24 (1970), pp. 713–735.

A. Borodin, S. Cook and N. Pippenger, *Parallel computation for well-endowed rings and space bounded probabilistic machines*, Tech. Rep. 162/83, Dept. Computer Science, Univ. Toronto, Inform. and Control, to appear.

A. Borodin, J. von zur Gathen and J. Hopcroft, *Fast parallel matrix and gcd computations*, Inform. and Control, 52 (1982), pp. 241–256.

W. S. Brown, *On Euclid's algorithm and the computation of polynomial Greatest Common Divisors*, J. Assoc. Comput. Mach., 18 (1971), pp. 478–504.

W. S. Brown and J. F. Traub, *On Euclid's algorithm and the theory of subresultants*, J. Assoc. Comput. Mach., 18 (1971), pp. 505–514.

D. G. CANTOR AND H. ZASSENHAUS, *On algorithms for factoring polynomials over finite fields*, Math Comp., 36 (1981), pp. 587–592.

G. E. COLLINS, *Subresultants and reduced polynomial remainder sequences*, J. Assoc. Comput. Mach., 14 (1967), pp. 128–142.

S. A. COOK, *The classification of problems which have fast parallel algorithms*, Proc. International Conference on Foundations of Computation Theory, Borgholm, 1983, pp. 78–93.

L. CSANKY, *Fast parallel matrix inversion algorithms*, this Journal, 5 (1976), pp. 618–623.

W. EBERLY, *Very fast parallel matrix and polynomial arithmetic*, Proc. 25th IEEE Symposium on Foundations of Computer Science, Singer Island, FL, 1984.

J. EDMONDS, *Systems of distinct representatives and linear algebra*, J. of Res. Nat. Bureau of Standards, 71B (1967), pp. 241–245.

N. J. FINE, *Binomial coefficients modulo a prime*, Amer. Math. Monthly, 54 (1947), pp. 589–592.

J. VON ZUR GATHEN, *Hensel and Newton methods in valuation rings*, Tech. Report 155 (1981), Dept. Computer Science, Univ. Toronto, Math. Comp. to appear.

——— [83a], *Factoring sparse multivariate polynomials*, Proc. 24th IEEE Symposium on Foundations of Computer Science, Tucson, AZ, 1983, pp. 133–137.

——— [83b], *Representations of rational functions*, Proc. 24th IEEE Symposium on Foundations of Computer Science, Tucson, AZ, pp. 172–179; this Journal, to appear.

J. VON ZUR GATHEN AND E. KALTOFEN, *Polynomial-time factorization of multivariate polynomials over finite fields*, Proc. 10th ICALP, Barcelona, 1983, pp. 250–263.

E. KALTOFEN, *A polynomial-time reduction from bivariate to univariate integral polynomial factorization*, Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science, Chicago, 1982, pp. 57–64.

——— [83a], *Polynomial-time reduction from multivariate to bivariate and univariate integer polynomial factorization*, this Journal, to appear.

——— [83b], *Fast parallel absolute irreducibility testing*, manuscript, November 1983.

D. E. KNUTH, *The Art of Computer Programming*, Vol. 2, 2nd ed., Addison–Wesley, Reading MA, 1981.

E. KRONECKER, *Zur Theorie der Elimination einer Variabeln aus zwei algebraischen Gleichungen*, Monatsberichte der Akademie der Wissenschaften, Berlin, 1881, pp. 535–600.

A. K. LENSTRA, H. W. LENSTRA AND L. LOVÁSZ, *Factoring polynomials with rational coefficients*, Math. Ann., 261 (1982), pp. 515–534.

E. LUCAS, *Sur les congruences des nombres eulériens et des coefficients différentiels des fonctions trigonométriques, suivant un module premier*, Bull. Soc. Math. France, 6 (1877/78), pp. 49–54.

J. MUIR, Letter from Mr. Muir to Professor Sylvester on the word continuant, Amer. J. Math., 1 (1878), p. 344.

J. REIF, *Logarithmic depth circuits for algebraic functions*, Proc. 24th Annual IEEE Symposium Foundations of Computer Science, Tucson, 1983, pp. 138–145.

W. L. RUZZO, *On uniform circuit complexity*, J. Comput. System Sci., 22 (1981), pp. 365–383.

V. STRASSEN, *The computational complexity of continued fractions*, this Journal, 12 (1983), pp. 1–27.

L. G. VALIANT, *Reducibility by algebraic projections*, in: Logic and Algorithmic, Zürich 1980, Monographie No. 30, Enseignement Mathématique, pp. 365–380.