Complexity of Computations with Matrices and Polynomials

Victor Pan

SIAM Review, Vol. 34, No. 2 (Jun., 1992), 225-262.

Stable URL:
http://links.jstor.org/sici?sici=0036-1445%28199206%2934%3A2%3C225%3ACOCWMA%3E2.0.CO%3B2-L

SIAM Review is currently published by Society for Industrial and Applied Mathematics.

SIAM REVIEW
Vol. 34, No. 2, pp. 225–262, June 1992

002

COMPLEXITY OF COMPUTATIONS WITH MATRICES AND POLYNOMIALS*

VICTOR PAN†

Abstract. This paper reviews the complexity of polynomial and matrix computations, as well as their various correlations to each other and some major techniques for the design of algebraic and numerical algorithms.

Key words. algorithms, numerical linear algebra, matrix computations, algebraic computing, polynomials, computational complexity

AMS(MOS) subject classifications. 65F05, 65F10, 65G10, 65H05, 68Q25, 68Q40, 12Y05

1. Introduction.

Polynomial and matrix computations are highly important classical subjects that changed their face during the last decades due to the development of computer technology, whose latest notable accomplishment was the development of parallel computers.

The known and still appearing polynomial and matrix algorithms perform far better than their classical predecessors and need to be classified according to their performance in order to guide users, hardware and software developers, and algorithm designers. Some limited initial guidance is given by computational cost estimates, which, in particular, measure the computational time, space, and for parallel algorithms, the number of processors used.

Such an analysis for polynomial and matrix computations is a major subject of computational complexity theory, which influences many other areas. In particular, many important combinatorial computations "boil down to linear algebra" [Lo].

Complexity estimates for matrix and polynomial computations are one of the main topics of this review, but we also try to demonstrate some major techniques of the design of the effective algorithms that support such record estimates and to show various correlations among different problems of polynomial and matrix computations and between the techniques of their solution.

Modern improved algorithms for such fundamental computations as matrix multiplication and, of course, discrete Fourier transform (DFT) have been successfully implemented and have become popular codes. On the other hand, there is a large class of algorithms whose efficacy has been recognized in scientific literature but not in computing practice. For instance, numerous important computations with polynomials, such as multiplication, division, composition, decomposition, and evaluation of their coefficients after the shift of the variable can already be greatly accelerated by means of their reduction to DFTs for problems of moderate sizes. This reduction has not been used, however, even in very recent software for symbolic computation, such as DERIVE, MAPLE, and MATHEMATICA (see [Dur], [CGGW], [Wol]).

In another example, application of the recent techniques, reviewed in §§1–3 but not yet known to users, enables us to perform some important matrix computations with a very low precision of few binary digits and to translate this into algorithms that consist of relatively few single precision arithmetic operations. Such advantages should eventually bring the methods to users.

*Received by the editors January 1, 1991; accepted for publication (in revised form) July 31, 1991.

†Computer Science Department, State University of New York, Albany, New York 12222 and Mathematics and Computer Science Department, Lehman College, City University of New York, Bronx, New York 10468. The research of this author was supported by National Science Foundation grants CCR-8805782 and 9020690 and by PSC CUNY awards 661340, 668541, and 669290.

Furthermore, in §4, we recall some techniques of modular (residue) arithmetic. For an integer or rational input, these techniques enable us to perform computations simultaneously with lower precision and with no errors. The techniques seem to be highly appropriate for inclusion into the packages of subroutines for numerical linear algebra.

It would suffice just to implement them in simple microcodes, which would be easy and inexpensive to use, unlike the presently available software for symbolic computations. Such software is now the only source for practical access to modular (residue) computations, but contains so many other sophisticated techniques and algorithms that it cannot be used for large-scale computing due to the very large amount of resources of time and memory space required.

Some potential benefits of using modular arithmetic in matrix computations can be seen, in particular, in §7, where we recall some effective algorithms for parallel evaluation of the solution and of a least-squares solution to a linear system of equations, of matrix inverse, rank and determinant. (Note the fundamental role of these computations. Recall, for instance, that the linear least-squares computation is at the heart of the famous interior point algorithms for linear programming [K], [PR,a], [RS], [P90c].) The parallel algorithms of §7 are highly attractive since they reduce the solution to relatively few matrix multiplications and since the latter operation can be very effectively performed on multiprocessors. For large-scale computations, however, the algorithms of §7 generally involve large values, such as the determinant of the input matrix, which lead to numerical stability problems. Since these computations are rational, however, the latter defect can be easily avoided by using modular arithmetic (see §§4 and 7).

Besides reviewing the algorithms and techniques, we also tried to show some correlations between the computations with matrices and with polynomials—to the benefit of both (see §§8 and 11, Remark 9.1 and also the end of §7, on approximating to matrix eigenvalues and polynomial zeros).

We hope that our review will ignite the interest of researchers and designers of computational software and hardware in recent effective algorithms, even though we can only show some highlights of recent work in the area of matrix and polynomial computations. For more detailed information, we refer the reader to [BP,a], and also to the earlier books [AHU], [BM], and, for polynomial computations, [Kn]. To avoid overlap with the excellent expositions of matrix computations in [Wilk], [GL], we focus on recent nonorthodox approaches in this area and on the highly important computations with dense structured matrices. We organize the material as follows.

We survey the techniques for bounding the precision of computation in §§1–4, which form Part I of our survey. Then, in Part II, we will only count the arithmetic operations and the arithmetic steps and processors involved, thus adopting the RAM and PRAM arithmetic models of computing. (We will comment on these models in §5, Part II.)

We will pay particular tribute to the algorithms that support the current record asymptotic complexity estimates for matrix and polynomial computations, using the "O" notation. To balance the omission of the overhead constants, hidden in this notation, we will add comments through our survey, and in Table 0.1 we will specify simple parallel complexity estimates (including the constants) for some fundamental computations (for matrix product we rely on the straightforward algorithm).

Regarding specific topics in Part II, in §6, we will review the arithmetic complexity of polynomial computations and outline a recent algorithm that supports the record cost estimates for approximating to polynomial zeros. We will review the complexity of computations with general matrices in §7 and will extend our study to computations with dense structured matrices in §§8 and 9, indicating a way to their unification (§9) and

TABLE 0.1

	Parallel computation for	Parallel time	Processors	Source
1.	FFT on n points (n is a power of 2)	$3(1+\log_2 n)$	$2n$	[Pease]
2.	Summation of n numbers	$2\lceil\log_2 n\rceil$	$\lceil n/\log_2 n\rceil$	[Quinn]
3.	Inner product	$2\lceil\log_2 n\rceil+1$	$\lceil n/\log_2 n\rceil$	(reduce to summation)
4.	$n\times n$ matrix times a vector	$2\lceil\log_2 n\rceil+1$	$n\lceil n/\log_2 n\rceil$	(n inner products)
5.	Product of $n\times n$ matrices	$2\lceil\log_2 n\rceil+1$	$n^2\lceil n/\log_2 n\rceil$	(n^2 inner products)
6.	Product of $n\times n$ Toeplitz matrix by a vector (n is a power of 2)	$1+9(2+\log_2 n)$	$4n$	(reduce to 3FFTs at $2n$ points, [AHU])

some correlations to polynomial computations. We will review polynomial division and its correlation to matrix and power series computations in §§10–12.

Part I. Exploiting fixed and variable precision computations. In this part we will show how some numerical or algebraic computations can be made more effective by means of either fixing their precision below the level of machine precision or by varying such a precision.

1. Binary segmentation. Suppose that the machine precision is substantially higher than the precision of the computations that we need to perform. Can we take advantage of this? One way would be to use or to build a computer capable of performing the lower precision arithmetic faster. For instance, Maspar, if complemented with appropriate microcodes, has such a capacity (compare also Connection Machines CM-1 and CM-2). The techniques of binary segmentation that we are going to describe enables us to take such an advantage on a wide class of existent computers, those capable of concatenating binary strings, partitioning them into segments, and turning such strings into integers or vice versa.

With these simple operations, we may dramatically decrease the number of single precision arithmetic operations involved.

Example 1.1. *Interpolation via binary segmentation.* Recall the interpolation problem of the recovery of an nth degree polynomial $p(x)$ from its values. Generally, we need the values at $n+1$ points, but let $p(x)=\sum_{i=0}^{n}p_ix^i$ be a polynomial with nonnegative integer coefficients, which are all strictly less than 2^g for a positive g. Let $h\geq g$ be an integer. Then the binary values $p_0,\cdots,p_n$ can immediately be recovered as the appropriate segments of the binary value $p(2^h)=\sum_{i=0}^{n}p_i2^{hi}$; that is, in this case, we only need a single interpolation point $x=2^h$, rather than $n+1$ points.

Extension. The simple algorithm of Example 1.1 is immediately extended to the case where $p_0,\cdots,p_n$ are integers lying strictly between -2^g and 2^g ([Kn, p. 191]). Choose an integer $h\geq g+1$ and compute $p(2^h)+\sum_{i=0}^{n}2^{hi+h-1}=\sum_{i=0}^{n}(p_i+2^{h-1})2^{hi}<2^{h(n+1)}$; then, recover the nonnegative integers $p_i+2^{h-1}<2^h$ and, finally, p_i for all i. Surely, the method also works where the coefficients $p_0,\cdots,p_n$ are *Gaussian integers*, that is, complex numbers with integer real and imaginary parts.

It is worth emphasizing that binary segmentation relies on correlation between integers and polynomials with integer coefficients. Such a correlation has been successfully exploited in order to extend available algorithms for polynomial computations to computations with integers and vice versa; this includes multiplication, division, computing the greatest common divisors (gcds) and the least common multiples (lcms), and the Chinese remainder computations [AHU], [BM].

Example 1.1 suggests that we may replace some operations with polynomials in x (in particular, their multiplication, but see [P84], [BP], [BP,a], and [Eberly] on some other operations) by similar operations with the values of such polynomials at $x = 2^h$, for an appropriate positive integer h, so that several shorter output values (the coefficients) can be read from a single longer value. Similar techniques can be applied to compute the inner and the outer vector products. We need only to have the longer output value fitted to the computer precision; otherwise, we shall partition the original problem into subproblems of smaller sizes.

Example 1.2. *Convolution of vectors via binary segmentation* [FP]. Let m and n be two natural numbers, U and V be two positive constants, $\mathbf{u}$ and $\mathbf{v}$ be the coefficient vectors of $u(x)$ and $v(x)$, two polynomials of degrees $m-1$ and $n-1$, respectively, whose coefficients are integers in the closed intervals bounded by $-U$ and U and $-V$ and V, respectively. Then the coefficients of the polynomial $w(x) = u(x)v(x)$ lie in the closed interval bounded by $-W$ and W, where $W = \min\{m,n\}UV$, so that Example 1.1 suggests that we may recover the convolution of $\mathbf{u}$ and $\mathbf{v}$, that is, the coefficients of the polynomial $u(x)v(x)$, if we compute the binary value $u(2^h)v(2^h)$ for an integer $h \geq 1 + \log(W+1)$. We need at most $(m+n-1)h$ binary digits in this multiplication since $|u(2^h)v(2^h)| < 2^{(m+n-1)h}$. (Here and hereafter logarithms are to base 2.)

Example 1.3. *Computing the inner and outer products of two vectors* (§40 of [P84]). Under the assumptions of Example 1.2, we may compute the inner product $\mathbf{u}^T\mathbf{v} = \sum_{j=0}^{n-1} u_j v_j$ (if $m = n$) and the outer product $[w_{ij}] = \mathbf{u}\mathbf{v}^T = [u_i v_j]$ of two vectors $\mathbf{u} = [u_i]$ and $\mathbf{v} = [v_j]$ by means of binary segmentation of the two integers $p(2^h)$ and $q(2^h)$. Here,

$$p(2^h) = \left(\sum_{i=0}^{n-1} 2^{ih} u_i\right)\left(\sum_{j=0}^{n-1} 2^{(n-1-j)h} v_j\right) = \sum_{k=0}^{2n-2} z_k 2^{kh}, \quad z_{n-1} = \mathbf{u}^T\mathbf{v},$$

$$|p(2^h)| < 2^{(2n-1)h},$$

and in this case h is an integer, $h \geq 1 + \log(W+1)$, $W = nUV$, so that the precision of $(2n-1)h$ binary digits suffices in this case, whereas

$$q(2^h) = \left(\sum_{i=0}^{m-1} 2^{inh} u_i\right)\left(\sum_{j=0}^{n-1} 2^{jh} v_j\right) = \sum_{i,j} u_i v_j 2^{(in+j)h}, \qquad |q(2^h)| < 2^{mnh},$$

and in this case h is an integer, $h \geq 1 + \log(W+1)$, $W = UV$, so that the precision of mnh binary digits suffices.

Thus, each of the inner and outer products is computed by means of a single multiplication of integers and of binary segmentation of the product.

For demonstration, let the vectors $\mathbf{u}$ and $\mathbf{v}$ have dimension 7 and have components 0, -1, and 1, so that $UV = U = V = 1$. Then in the straightforward evaluation of the convolution, the inner product and the outer product of $\mathbf{u}$ and $\mathbf{v}$ requires 85, 13, and 49 ops, respectively. With binary segmentation we only need a single integer multiplication

for each problem if we may use a sufficiently high precision. In particular, if the computer precision were 60 binary bits, the single multiplication of $u(2^h)$ by $2^{(n-1)h}v(2^{-h})$, for $h = 4$, $n = 7$ and $(2n-1)h = 42$, with subsequent segmentation of the coefficient z_6, would suffice for computing the inner product. For the convolution, the same input implies that $m+n-1 = 13$, $h = 4$, $(m+n-1)h = 52$, and a single multiplication with the precision of 52 binary digits would suffice. For the outer product, we have $m = n = 7$, $h = 2$, $mnh = 98$, and a single multiplication with 98-bit precision suffices. In all cases, an alternative way is to use more multiplications but a lower precision. For instance, for an even n, it suffices to multiply $u(2^{nh/2})$ by $p(2^h)$ and by $q(2^h)$, where $p(x) = v(x) \bmod x^{n/2} = \sum_{j=0}^{n/2-1} v_j x^i$ and $q(x) = (v(x) - p(x))/x^{n/2}$, with the precision of $mnh/2$ binary bits in order to compute the outer product. We refer to [P92] on compact storage of vectors and matrices based on binary segmentation.

2. Lower precision computation. Compact multigrid. The previous section motivated us to try to select or to devise the algorithms that require a very low precision of computations, rather than just the single precision. It may be surprising that this is possible in some fundamental matrix computations. Consider, for instance, Wilkinson's iterative improvement of the solution of linear system of equations $A\mathbf{x} = \mathbf{b}$. Wilkinson's algorithm recursively computes the residual and error vectors,

$$\mathbf{r}_{p+1} = \mathbf{r}_p - A\mathbf{e}_p,$$
$$\mathbf{e}_{p+1} = \tilde{A}^{-1}\mathbf{r}_p,$$

$p = 0, 1, \cdots$, where $\|\tilde{A} - A\|$ is supposed to be small enough. Only few binary digits suffice to represent each component of the vectors $\mathbf{e}_p$, but longer binary values are generally needed in order to represent the entries of A and $\tilde{A}$ and the components of $\mathbf{r}_p$. If A is filled with bounded integers, however, only few digits of these longer binary values need to be used, and the modified algorithm can indeed be performed with a very low precision. The idea is to modify Wilkinson's scheme so as to avoid any influence on $\mathbf{e}_p$ from both "least significant" and "most significant" digits that appear in the floating point representation of the latter entries and components. Then such digits are discarded. We refer to [P91a] for details, and next we will elaborate on another surprising example.

We will follow [PR89a], [PR,b] and will show that lower precision computations suffice in order to solve a large class of partial differential equations (PDEs) reduced to linear algebraic systems of equations. To formalize the results, we will recall that multiplying a pair of (k-bit) integers modulo 2^k takes a storage space of the order of k bits and time of the order of k^α bit-operations, where α depends on the algorithm, $1 \le \alpha \le 2$, so that, in particular, $O(1)$ bits of storage space and $O(1)$ bit-operations suffice to multiply a pair of $O(1)$-bit integers.

We will first show how to decrease the storage space. The straightforward representation of the solution within the discretization (truncation) error bound requires $O(N \log N)$ binary bits. We will follow [PR89a], [PR,b] and will arrive at a compact representation with $O(N)$ bits. In the case of constant coefficient linear PDEs, we will compute the solution by using a low precision and $O(N)$ bit-operations. Note that $\log_2 N = 10$ already in the case where $N = 1024$, which is far below the size of linear systems handled in practical PDE computations. Following [PR89a], [PR,b], we will call this algorithm Compact Multigrid.

Let us next specify these results starting with some auxiliary facts and definitions. We will study linear PDEs on the unit d-dimensional cube, discretized over a family of

d-dimensional lattices $L_0, L_1, \cdots, L_k$, where each point of L_j lies at distance $h_j = 2^{-j}$ from all its nearest neighbors. There are exactly $|L_j| = N_j = 2^{dj}$ points in L_j for $j = 0, 1, \cdots, k$, and the overall number of points equals $N_k = N = 2^{dk}$, where $k = (\log N)/d$, provided that we identify the boundary points whose coordinates only differ by zero or 1 from each other. (On the actual discretization grids for PDEs, all the boundary points are distinct, so that the jth grid contains slightly more than N_j points.)

Let $u(\mathbf{x})$, a function in the d-dimensional vector of variables $\mathbf{x}$, represent the solution to the PDE, and let $u_j(\mathbf{x})$, for a fixed $\mathbf{x} \in L_j$, denote the respective component of the N_j-dimensional vector $\mathbf{u}_j$ that represents the solution to the linear system,

$$D_j \mathbf{u}_j = \mathbf{b}_j, \tag{2.1}$$

of the difference equations generated by a discretization of the PDE over the lattice L_j, so that $\Delta_j(\mathbf{x}) = u(\mathbf{x}) - u_j(\mathbf{x})$ for $\mathbf{x} \in L_j$ denotes the discretization error function on L_j for $j = 1, \cdots, k$.

Under the routine assumption that $\|D_j\|_2 \|D_j^{-1}\|_2 = O(h_j^a)$ for a constant a, which we will call the *weak smoothness assumption*, we may deduce that for some fixed $\tilde{b} \geq 1$ and $\tilde{c} \geq 0$,

$$|\Delta_j(\mathbf{x})| < 2^{\tilde{c} - \tilde{b}j} \quad \text{for all } \mathbf{x} \in L_j. \tag{2.2}$$

Thus, we need $O(j)$ binary digit (bits) to represent the value $u_j(\mathbf{x})$ approximated to the level of truncation. This means the order of $k = \log_2 N$ bits per point for $j = k$.

To compress this representation to $O(1)$ bits per point, we will follow the routine of the multigrid approach. We will let $u_0(\mathbf{x}) = 0$ for $\mathbf{x} \in L_0$ and will let $\hat{u}_{j-1}(\mathbf{x})$ for $j = 1, 2, \cdots, k$ denote the prolongation of $u_{j-1}(\mathbf{x})$ from L_{j-1} to L_j, obtained by means of the interpolation (by averaging) of the values of $u_{j-1}(\mathbf{x})$ at an appropriate array of points of L_{j-1} lying near $\mathbf{x}$. Then $\hat{u}_{j-1}(\mathbf{x}) = u_{j-1}(\mathbf{x})$ if $\mathbf{x} \in L_{j-1}$, and $\hat{u}_{j-1}(\mathbf{x})$ is the average of $u_{j-1}(\mathbf{y})$ over all $\mathbf{y}$ such that $\mathbf{y} \in L_j$ and, say, $|\mathbf{y} - \mathbf{x}| = h_j$ if $\mathbf{x} \in L_j - L_{j-1}$. Furthermore,

$$u_j(\mathbf{x}) = \hat{u}_{j-1}(\mathbf{x}) + e_j(\mathbf{x}), \quad \mathbf{x} \in L_j, \quad j = 1, \cdots, k, \tag{2.3}$$

where $e_j(\mathbf{x})$ denotes the interpolation error on L_j. It is easy to deduce from (2.2) that

$$|e_j(\mathbf{x})| \leq 2^{c - \alpha j} \tag{2.4}$$

for all $\mathbf{x} \in L_j$, $j = 1, \cdots, k$, and for some fixed $c \geq 0$ and $\alpha \geq 1$. Therefore, the prolongation of $u_{j-1}(\mathbf{x})$ from L_{j-1} to L_j only requires $O(N_j)$ bit-operations.

Compression of the output data. Now assume the weak smoothness and compress approximations to all the N values of $u_k(\mathbf{x})$ on L_k within absolute errors of at most $2^{c - \alpha k}$, so as to decrease the storage space required. The straightforward fixed point binary representation of these values of $u_k(\mathbf{x})$ generally requires $N\lceil \alpha k - c \rceil$ binary bits.

As an alternative, let us store $u_k(\mathbf{x})$ on L_k in the compressed form by recursively approximating within $2^{c - \alpha j - \alpha}$ to the fixed point binary values $e_j(\mathbf{x})$ for $\mathbf{x} \in L_j$, $j = 1, \cdots, k$. The storage space of $2^d\lceil \alpha - c \rceil + \alpha(N_2 + N_3 + \ldots + N_k) < 2^d\lceil \alpha - c \rceil + 2\alpha N = O(N)$ binary bits suffices for this compressed information. This means saving roughly the factor of $k = \log N$ binary bits against the straightforward representation.

Recovery of the solution values from the compressed data. Let us recover $u_k(\mathbf{x})$ on L_k from the compressed information given by $e_j(\mathbf{x})$ on L_j for $j = 1, \cdots, k$. Start with $u_0(\mathbf{x}) = 0$ for $\mathbf{x} \in L_0$, and recursively, for $j = 1, \cdots, k$, compute the values

(a) $\hat{u}_{j-1}(\mathbf{x})$ on L_j, by prolongation of $u_{j-1}(\mathbf{x})$ from L_{j-1} to L_j, and
(b) then $u_j(\mathbf{x})$ on L_j, by applying equations (2.3).

Perform both stages (a) and (b) with precision $2^{c-\alpha j-\alpha}$. Stage (b) amounts to appending α binary bits of $e_j(\mathbf{x})$ to the available string of binary bits in the fixed point binary representation of $\hat{u}_{j-1}(\mathbf{x})$ for each $\mathbf{x} \in L_j$. Stage (a) amounts to scanning the values of $u_{j-1}(\mathbf{x})$ on L_{j-1} and to the summation of few β-bit binary numbers defined by the β *least significant* binary bits in the representation of $u_{j-1}(\mathbf{x})$ for appropriate $\mathbf{x}$ from L_{j-1} (where, say, $\beta = O(\alpha)$). Since $\sum_j N_j = O(N)$, the computational complexity estimates for stages (a) and (b) stay within the desired bound $O(N \log N)$.

3. Computing the compressed solution by compact multigrid. In this section, in addition to our previous assumptions, we will assume that

(1) A fixed iterative algorithm for linear systems with matrices D_j (such as Gauss–Seidel, SSOR, or a multigrid algorithm of this section) uses $O(1)$ multiplications of submatrices of D_j by vectors for every j in order to decrease, by the factor independent of j and N, the norm of the error of the approximation to the solution $u_j(\mathbf{x})$ of system (2.1) (*linear convergence assumption*);

(2) The entries of the matrices D_j for all j, as well as the components of $\mathbf{b}_j$, are integers having magnitudes $O(1)$ or turn into such integers after the truncation and scaling of the system (2.1); this assumption of *coefficient bounding* holds for the constant coefficient PDEs.

Under these assumptions, we will compute the compressed solution to the system (2.1) for all j by using $O(1)$ bit-operations per point of a grid. The time complexity of computing the compressed data structure is dominated by the time required to obtain the solution vectors $\mathbf{e}_j$ for the linear systems of equations over L_j, for $j = 1, \cdots, k$:

$$D_j \mathbf{e}_j = \mathbf{r}_j. \tag{3.1}$$

Here

$$\mathbf{r}_j = \mathbf{b}_j - D_j \hat{\mathbf{u}}_{j-1}. \tag{3.2}$$

The matrices D_j and the vectors $\mathbf{b}_j$ are from the linear systems (2.1), and the vectors $\mathbf{e}_j$ and $\hat{\mathbf{u}}_{j-1}$ have components $e_j(\mathbf{x})$ and $\hat{u}_{j-1}(\mathbf{x})$ corresponding to the points $\mathbf{x} \in L_j$ and defined by (2.1) and (2.3).

We will follow the routine of the multigrid methods (compare [McC87], [FMc]) and will recursively evaluate the vectors $\mathbf{e}_j$ for $j = 1, \cdots, k$ by applying the so-called *V-cycle multigrid scheme*. We will arrive at the desired Compact Multigrid algorithm, in which we will exploit the compressed representation of the solution. Initially, we will let $u_0(x) = 0$ for $\mathbf{x} \in L_0$. At stage j, we will successively compute for all $\mathbf{x} \in L_j$:

(a) $\hat{u}_{j-1}(\mathbf{x})$ (by prolongation of $u_{j-1}(\mathbf{x})$ from L_{j-1} to L_j),
(b) $r_j(\mathbf{x})$ (by using (2.3)),
(c) $e_j(\mathbf{x})$ (by solving linear system (3.1) "to the level of truncation"),
(d) $u_j(\mathbf{x})$ in the compressed form (by using (2.3)).

We may then restrict $u_{j+1}(\mathbf{x})$ to $u_j(\mathbf{x})$ for $j = k-1, k-2, \cdots, 1$. (This stage contains no smoothing iterations, unlike some customary variants of the multigrid scheme.) Then we recursively repeat such a loop, customarily called V-cycle.

The linear covergence assumption means that, for all j, the errors of the approximations for $u_j(\mathbf{x})$ decrease by a constant factor independent of j and N when stages (a)–(d) are repeated once, even if only $O(1)$ iteration steps are used at stage (c) for solving linear systems (2.1) for every j. Such convergence results have been proven for the

customary multigrid algorithms applied to a wide class of PDEs (see [BD81], [Ha77], [HT82], [Hac80], [Hac85], [Mc86], [McT83], [FMc87a]).

Let us estimate the time complexity of these computations, dominated by the time needed for solving the linear systems (3.1).

The size $|L_j| = 2^{dj}$ of the linear system (3.1) increases by 2^d times as j grows by 1. Even if we assume that the solution time for the system (3.1) is linear in $|L_j|$, the overall solution time for all k such systems in terms of the number of arithmetic operations involved is still less than $1/(1-2^{-d})$ times the solution time for the single system (2.1) for $j = k$, which gives us uncompressed output values $u_k(\mathbf{x})$ for $\mathbf{x} \in L_k$. The bit-operation count is even more favorable to the solution of systems (3.1) for all j, as opposed to the single system (2.1) for $j = k$, because the output values $e_j(\mathbf{x})$, satisfying systems (3.1), are sought with the lower precision of α binary bits.

Furthermore, we solve the linear systems (3.1) by iterative methods, where each step is essentially reduced to a constant number, say, one or two, multiplications of a matrix D_j or its submatrices by vectors. Due to the linear convergence assumption that we made, a constant number of iterations suffices at each step j in order to compute the α desired binary bits of $e_j(\mathbf{x})$.

The computational cost of multiplication of D_j by a vector is $O(N_j)$ arithmetic operations for a sparse and structured discretization matrix D_j (having $O(1)$ nonzero entries in each row). The next two propositions summarize our estimates.

PROPOSITION 3.1. *$O(N)$ operations suffice to compute the vectors $\mathbf{e}_j$ for all j, that is, to compute the smooth compressed solution to a constant coefficient linear* PDE *discretized over the lattice L_k, under the weak smoothness and linear convergence assumptions.*

Furthermore, we only need $O(1)$ binary bits in order to represent $e_j(\mathbf{x})$ for every $\mathbf{x} \in L_j$ and every j. Since D_j has only $O(1)$ nonzero entries per row and since these entries are integers having magnitudes $O(1)$ (due to assumption (2) of coefficient bounding), it suffices to use $O(1)$ bits to represent $r_j(\mathbf{x})$. (These $O(1)$ bits may not occupy all the positions of the nonzero bits of the associated component of $\mathbf{b}_j$, since $|r_j(\mathbf{x})|$ may be much less than $\|\mathbf{b}_j\|_\infty$.) Thus, we will perform all the arithmetic operations with $O(1)$-bit operands and will arrive at Proposition 3.2.

PROPOSITION 3.2. *$O(N)$ bit-operations and $O(N)$ storage space under the Boolean model of computation suffice in order to compute (by using the Compact Multigrid algorithm) the compressed solution to a constant coefficient linear* PDE *that satisfies the weak smoothness and linear convergence assumptions.*

Extensions of the results. In [PR91] the acceleration of the solutions has been extended to all the piecewise-linear PDEs with constant coefficients (which enabled us to relax the linear convergence assumption). Furthermore, the compact storage scheme has been extended to the solutions of both linear *and* nonlinear PDEs (under the routine smoothness assumptions) and, moreover, to the representation on a grid of any smooth function. Finally, the method works over the discretization sets that are more general than the grids with equally spaced points. Also, this enables us to compactly store some nonsmooth functions.

4. Techniques of modular (residue) arithmetic: infinite precision = low precision. Let us now shift to some techniques of computer algebra, specifically, to computing with no rounding-off errors, which, in particular, may handle the ill-conditioned problems. Even in the ill-conditioned case, we may decrease the precision of computations, due to using modular (residue) arithmetic, complemented by some other simple techniques, which are much simpler and much less expansive to use than the various other techniques of algebraic and symbolic computation.

We will first recall p-adic lifting, which invokes the names of Newton and Hensel. The idea is to start with computing the solution modulo a prime p (which means the precision of the order of $\log p$ binary bits). Then we recursively lift the resulting solution to its value modulo p^k for recursively growing k (which means the order of $k \log p$ binary bits in the precision).

Typically, the computation modulo p involves more arithmetic operations than the lifting stages. Here is an example from [MC].

ALGORITHM 4.1. *Hensel's lifting process for linear systems.*
Input: three positive integers H, n, and p, an $n \times n$ matrix A and an n-dimensional vector $\mathbf{b}$, both filled with integers such that $\det A \neq 0 \bmod p$.
Output: $\mathbf{x}_H = A^{-1}\mathbf{b} \bmod p^H$.
Stage 0 (initialization). Compute $S(0) = A^{-1} \bmod p$,
$\mathbf{x}_1 = S(0)\mathbf{b} \bmod p, \mathbf{v}_1 = A\mathbf{x}_1 \bmod p^2$.
Stage i, $i = 1, \cdots, H-1$. Compute the vectors
$\mathbf{w}_i = \mathbf{b} - \mathbf{v}_i \bmod p^{i+1}$,
$\mathbf{y}_i = (S(0)\mathbf{w}_i/p^i) \bmod p^2$,
$\mathbf{v}_{i+1} = \mathbf{v}_i + p^i A\mathbf{y}_i \bmod p^{i+2}$,
$\mathbf{x}_{i+1} = \mathbf{x}_i + p^i\mathbf{y}_i \bmod p^{i+1}$.
Stage H. Recover the vector $A^{-1}\mathbf{b}$ from $\mathbf{x}_H = A^{-1}\mathbf{b} \bmod p^H$ ([Wang]). Output $A^{-1}\mathbf{b}$ (or in some applications skip Stage H and output $\mathbf{x}_H$).

Stage i for $i = 1, \cdots, H-1$ involves $O(n)$ operations in order to add vectors and to multiply them by constants, and also involves two multiplications of matrices A and $S(0)$ by vectors reduced modulo p^2. Performing these two multiplications with a lower precision may motivate, for instance, application of the binary segmentation at this point. We may avoid computing the matrix $S(0)$ and find the vectors $\mathbf{x}_1$ and $\mathbf{y}_i$ for all i by solving the linear equations $A\mathbf{x}_1 = \mathbf{b} \bmod p$, $A\mathbf{y}_i = \mathbf{w}_i/p^i \bmod p^2$.

Here is another effective algorithm from [MC] with p-adic lifting (for matrix inversion).

ALGORITHM 4.2. *p-adic lifting for matrix inversion.*
Input: Three positive integers h, n and p, and an $n \times n$ matrix A, such that $\det A \neq 0 \bmod p$.
Output: $A^{-1} \bmod p^H$ for $H = 2^h$.
Stage 0 (initialization). Compute $S(0) = A^{-1} \bmod p$.
Stage i, $i = 1, \cdots, h$. Compute the matrices $E(i) = I - AS(i-1) \bmod p^{2^i}$, $S(i) = S(i-1)(I + E(i))$.

To its advantage, Algorithm 4.2 only involves a few simple matrix operations (of the level 3BLAS), that is, a single inversion modulo p, $2h$ multiplications, and $2h$ additions/subtractions with the identity matrix I. The initialization stages of both Algorithms 4.1 and 4.2 could be potentially harder for parallel implementation. Actually, however, these stages are greatly simplified, due to the reduction modulo p, which means the precision of computation of $\lceil \log_2 p \rceil$ bits (compare §7 below).

Note that $(\det A) \bmod p$ may vanish even where $\det A$ does not, but this occurs with a low probability for a random choice of p in an appropriate interval [P87b], [BP,a].

In an alternate way to p-adic lifting for problems having an integer or rational output,we may compute the solution modulo several primes or pairwise relatively prime integers $p_1, \cdots, p_k$, by using a lower precision of $O(\log p_j)$ binary bits for $j = 1, \cdots, k$.

Then we may obtain the solution modulo $M \le p_1 \cdots p_k$ by means of the Chinese remainder algorithm [AHU], [BM], [Kn], [BP,a].

If the solution is an integer i between $-M/2$ and $M/2$ (as is the case, say, for polynomial multiplication, for an easily estimated M) or if it is the ratio p/q, where $|p|$ and $|q|$ are integers less than $\sqrt{M/2}$, we may immediately recover the solution from its value modulo M. The integer case is trivial since either $i = i \bmod M$ or $i = (i \bmod M) - M$. Even if i lies between $-M$ and M, we may recover i if we know $i \bmod M$ and $i \bmod N$ for any N not dividing M. To recover p and q from $(p/q) \bmod M$, we may apply the algorithm of [Wang], which invokes the continued fraction approximation algorithm [HW79] and outputs the solution value p/q, where p and q are integers, $\max\{|p|, |q|\} < \sqrt{M/2}$. The continued fraction approximation algorithm is actually the extended Euclidean algorithm [AHU], [Kn], [BP,a] applied to two positive integers a and b. It only requires $O(\mu(s) \log s)$ bit-operations [AHU], where $s = \log \max\{a, b\}$, $\mu(s) = O(s \log s \log\log s)$ is the number of bit-operations required in order to multiply a pair of integers modulo 2^s.

In the algorithm of [Wang], we need upper bounds on $|p|$ and $|q|$ in order to choose M. Typically, such an output value p/q is a component of the solution vector to a system of linear equations; then we may bound $|p|$ and $|q|$ by relying on Cramer's rule and on the inequality $|\det A| \le \|A\|^N$ for an $N \times N$ matrix A. By Cramer's rule, $s = (\det A)p/q$ is an integer. Therefore, we may immediately recover s, $\det A$, and then p/q from $s \bmod M$ and $(\det A) \bmod M$ if $M > 2\max\{|s|, |\det A|\}$. If we are given $s \bmod p_i$ and $(\det A) \bmod p_i$ for $i = 1, \cdots, k$ and appropriate $p_1, \cdots, p_k$, we may alternatively apply the Chinese remainder algorithm.

The modular arithmetic can be useful even in the fixed precision computations, where we may compute modulo M and then recover the integer or rational output. This way we ensure that the precision of the intermediate computations does not grow to more than $\lceil \log_2 M \rceil$.

The recovery of the rationals from their values modulo an integer by means of Wang's algorithm has an interesting analogy with their recovery from their binary approximation.

FACT 4.1 ([UP83]). *Let $z = p/q$; D, p, q, and s be integers, $|q| < D$, $\epsilon < 1/(2D^2)$, $x = a/2^s$ approximate to z within ϵ. Then p and q can be recovered (given a, s, ϵ, and D) by means of the continued fraction approximation algorithm applied to a and 2^s.*

Of course, simple rounding-off, instead of the continued fraction approximation algorithm, suffices if z is an integer, $q = 1$, and $\epsilon < 1/2$.

If we try to implement computer algebra algorithms over a finite field or ring of integers modulo a prime p or a prime power p^k (and this section should show why we may prefer to do this), there may arise the problem of supplying the Nth roots of 1 in order to perform discrete Fourier transform at N points and interpolation to a degree $N - 1$ polynomial (see §6 below). A simple solution to this problem is to perform the computation in an extension of F to a field E of integers modulo p^K for a larger K, which we may, however, bound by using special techniques, so that the transition to E increases the computational complexity by the factor of $O(\log\log N)$ (compare [CK]).

Part II. Arithmetic complexity of polynomial and matrix computations.

5. RAM and PRAM models. Hereafter we will adopt the customary and convenient measure for the complexity of algebraic and numerical computations in terms of the number of arithmetic operations involved under the *Random Access Machine* (RAM) model [AHU]. The algorithms of §1 (using binary segmentation) suggest that we should

then appropriately bound the precision of computations. Indeed, otherwise, a single multiplication suffices to compute a product of vectors or polynomials.

On the other hand, we avoid such a discrepancy and may estimate the complexity under the arithmetic RAM model of computing, as long as the computations are performed with the precision being at most of the order of the output precision. This was shown to be the case in §4 for the computer algebra computations, and this is the case for the numerically stable numerical algorithms. Note that in §3, the precision of the output was of the order of $\log N$ and even *exceeded* the precision of computations $O(1)$.

For parallel computations, we will assume Parallel RAM (PRAM) models, where in each step each nonidle processor performs one arithmetic operation [KR]. Under these models, the complexity of processor communication and synchronization is not included, but we will apply PRAM models to the computations that only require few or moderately many processors. In this case the cost of their communication and synchronization is dominated by the arithmetic cost.

We will assume Brent's scheduling principle, according to which a slowdown of parallel computations by $O(s)$ times suffices in order to decrease the number of processors from p to p/s provided that s and p/s are positive integers [Bre]. We will write $O_A(T)$ and $O_A(t,p)$ to denote the sequential and parallel arithmetic cost of an algorithm, where T, t, and p denote the numbers of arithmetic operations (sequential time), of arithmetic parallel steps (parallel time), and processors used, respectively, all defined within constant factors.

Then by *Brent's principle*, the bound $O_A(t,p)$ implies $O_A(st, p/s)$ as long as p/s and s are positive integers. In particular, $O_A(t,p)$ implies $O_A(tp,1)$ if we let $s = p$, and tp is called the *total work* of a parallel algorithm [KR], whereas the ratio $T/(tp)$ measures its *processor efficiency*, provided that T denotes the record sequential time bound for the same computational problem.

Brent's principle has a wide area of applications. For instance, we may immediately sum n numbers by using $\lceil \log_2 n \rceil$ parallel addition steps and n processors, but we may arrive at the bound of $O_A(\log n, n/\log n)$ by slowing down the first $\lceil \log_2 \lceil \log_2 n \rceil \rceil$ steps. A similar application of Brent's principle enables us to extend the bound of Proposition 3.2 to $O_A(\log N, N/\log N)$.

An arithmetic algorithm *is in* NC if its parallel cost is $O_A(t,p)$ for $t = O((\log N)^k)$, $\log p = O(\log N)$, k being a fixed constant and N being the length (size, dimension) of the input to the computational problem, say, the number of the coefficients of the input polynomial.

Similar definitions, $O_B(T)$, $O_B(t,p)$, the total work, NC, processor efficiency, and so on, can be applied to estimating sequential and parallel Boolean complexity of computations with Boolean operations (bit-operations) playing the role of arithmetic operations. We may immediately extend upper bounds on the arithmetic complexity of computations to the Boolean complexity bounds if we know the precision of computations. Indeed, we may simply apply the known estimates for the Boolean cost of arithmetic operations if we are given their precision (see [AHU], [Kn]). We will not comment more on this issue in our review, but will mention that the $O(1)$ precision algorithm of §3 immediately implies the bounds $O_B(N)$ and $O_B(\log N, N/\log N)$ on the Boolean complexity of the solution of PDEs, thus extending Proposition 3.2.

6. Arithmetic complexity of polynomial computations. Next we will survey the known asymptotic estimates for the arithmetic complexity of polynomial computations.

Table 6.1 shows a dramatic improvement of the asymptotic complexity of several classical polynomial computations, which typically involve arithmetic operations of the order of N^2 for the input polynomials of degree N. This improvement is due, in particular, to the famous fast Fourier transform (FFT) algorithm, which only costs $1.5N \log_2 N$ arithmetic operations or, in the parallel implementation, $O_A(\log N, N)$, and which may solve each of the two converse problems of multipoint evaluation and interpolation for a polynomial $p(x)$ of degree less than $N = 2^k$ with an integer k and for the nodes being the Nth roots of 1. The algorithm recursively exploits the identity $p(x) = s(y) + xt(y)$, where $y = x^2$, so that y is an $(N/2)$th root of 1, for $x^N = 1$, and where $s(y)$ and $t(y)$ are polynomials of degrees less than $N/2$. Thus, the original problem is recursively reduced to two problems of half size. Such a divide-and-conquer strategy generally leads to numerous effective algorithms. For example, the reader may apply such a strategy in order to shift from Newton's representation to the coefficient-wise representation of a polynomial of degree N and vice versa for the cost $O_A(\log N, N)$. Here is another (very famous) example of an application of FFT.

We may evaluate the polynomial product $u(x)v(x)$ (vector convolution), that is, compute $w_i = \sum_{j=0}^{i} u_j v_{i-j}$, $i = 0, 1, \cdots, D$, $D = \deg(u(x)v(x))$, by first computing $u(x)$ and $v(x)$ at the Nth roots of 1, for $N = 2^n > D = \deg(u(x)v(x))$, and for the cost $O_A(\log N, N)$ of two FFTs on N points, then multiplying the N pairs of the computed values, and finally recovering $u(x)v(x)$ by means of interpolation (inverse FFT) for the cost $O_A(\log N, N)$ (versus the classical algorithm using N^2 operations).

The constant factors omitted in the estimates of Table 6.1 are usually not too large; they grow from 1.5 for FFT and depend on the complexity of the reduction to FFT (see [BP,a]). Many of the algorithms based on FFT and supporting these bounds, as well as many algorithms of this kind for other similar polynomial computations, already outperform the classical algorithms for problems of moderate sizes and should be included in the symbolic computation software.

Some of the modern fast algorithms for polynomial computations (in particular, FFT itself) are widely used in the floating point numerical computations with roundoff. This, of course, raises the problem of roundoff errors. For FFT and related computations, this problem can be avoided by means of performing FFT over the ring of integers modulo a natural m, chosen so as to ensure the existence of the desired roots of 1 [AHU], [BM]. On the other hand, we have the following result on the numerical stability of FFT performed in the d-digit floating point arithmetic.

FACT 6.1 ([GS]). *Let* $\omega = \exp(2\pi\sqrt{-1}/K)$, $(\Omega)_{i,j} = \omega^{ij}/\sqrt{K}$, $Q = \Omega$, *or* $Q = \Omega^H$, K *be a power of* 2, *the* d*-digit floating point computation of* $Q\mathbf{b}$ *be performed by means of* FFT *on* K *points, and* $\mathrm{fl}(Q\mathbf{b})$ *denote the output vector of this computation. Then*

$$\mathrm{fl}(Q\mathbf{b}) = Q\mathbf{b} + \mathbf{f}(\mathbf{b}), \|\mathbf{f}(\mathbf{b})\|_2 \le 2^{-d}\phi(K)\|\mathbf{b}\|_2, \phi(K) = 8.5K\sqrt{K}\log K.$$

The power of FFT and of fast polynomial multiplication is extended to many other computations with polynomials, matrices, partial fractions, power series, and integers by means of various interesting, and sometimes intricate, techniques that can be found in [AHU], [BM], [Kn], [BP,a], and to some extent, in §§8 and 10 below. Here are just two simple examples.

Example 6.1. *Discrete Fourier transform* (DFT) *on any number of points* ([Kn, pp. 300 and 588]). Given the values $p_0, p_1, \cdots, p_{K-1}$, compute $r_h = \sum_{j=0}^{K-1} p_j \omega^{hj}$ for $h = 0, 1, \cdots, K-1$, where ω is a primitive Kth root of 1.

TABLE 6.1
Arithmetic complexity of polynomial computations.

(*c denotes constants not less than one, upper bounds are within constant factors, except for evaluation at a single point;* $\log^* n = \min\{k : \log_2^{(k)} n < 1\}$, $\log_2^{(0)} n = n$, $\log_2^{(k+1)} n = \log\log^{(k)} n$, $k = 0, 1, \cdots$)

	Sequential Arithmetic Time		Parallel Arithmetic Complexity	
Problem	Lower Bounds	Upper Bounds	Time	Processors
evaluation at a point (no preconditioning)	$n(\pm)$, $n(*, \div)$	$n(\pm)$, $n(*, \div)$	$\log n$	$n/\log n$
evaluation at a point (with preconditioning)	$n(\pm)$, $\frac{n+1}{2}(*, \div)$	$n+2(\pm)$, $\frac{n+2}{2}(*, \div)$	$\log n$	$n/\log n$
evaluation and interpolation at nth roots of 1 (FFT)	cn	$n \log n$	$\log n$	n
evaluation at m points. Interpolation at $m = n$ points	$m \log n$	$m \log^2 n$	$(\log n)^2 \log^* n$	$m/\log^* n$
multiplication (degrees m and n, $N = m + n$)	cN	$N \log N$	$\log N$	N
division (degrees m and n, $k = m - n$)	ck	$k \log k$	$\log k \log^* k$	$k/\log^* k$
gcd computation	cN	$N \log^2 N$	$N \log N$	$\log N$
(degrees m and n, $N = m + n$)			$\log^2 N$	$N^2/\log N$
computing a zero of a polynomial with error $< 2^{-b}$	$n + \log b$	$n \log b \log n$	$\log^2 n \log(bn)$	$\frac{n \log b}{\log n \log(bn)}$
computing all the zeros with error $< 2^{-b}$	$n + \log b$	$n^2 \log b \log n$	$n \log n \log(bn)$	$n \log b/\log(bn)$
			$\log^3(nb)$	$(bn)^{O(1)}$
composition of two polynomials of degree n	cn^2	$n^2 \log^2 n$	$(\log n)^2 \log^* n$	$n^2/\log^* n$
composition and reversion modulo z^n	cn	$n \log^{3/2} n$	$(\log n)^2 \log^* n$	$\frac{(n^2/\log n)^{1/2}}{\log^* n}$
decomposition of a polynomial of degree N	cN	$N \log^2 N$	$(\log N)^2 \log^* N$	$N/\log^* N$

Solution. Rewrite

$$r_h = \omega^{-h^2/2} \sum_{j=0}^{K-1} \omega^{(j+h)^2/2} \omega^{-j^2/2} p_j \quad \text{for } h = 0, 1, \cdots, K-1.$$

Substitute $m = K-1$, $n = 2K-2$, $w_{n-h} = r_h\omega^{h^2/2}$, $u_j = \omega^{-j^2/2}p_j$ for $j \le m$, $u_j = 0$ for $j > m$, $v_s = \omega^{(n-s)^2/2}$, and rewrite the expressions for r_h as follows: $w_i = \sum_{j=0}^{i} u_j v_{i-j}$, $i = m, m+1, \cdots, n$. This reduces DFT on K points to the evaluation of w_i for $i = m, m+1, \cdots, n$, which is a part of the convolution problem of computing $w_i = \sum_{j=0}^{i} u_j v_{i-j}$, $i = 0, 1, \cdots, n$.

Remark 6.1. An alternative approach [W79] decreases the above estimates for both numbers of additions and multiplications for FFT on K points for any K, simultaneously yielding the bounds of $O(K)$ multiplications and $O(K \log K)$ additions. The best available lower bounds are of the order of K.

Example 6.2. *Shift of the variable* [ASU]. Given a complex value Δ and the coefficients $p_0, p_1, \cdots, p_n$ of a polynomial $p(x) = \sum_{i=0}^{n} p_i x^i$, compute the coefficients $q_0(\Delta)$, $q_1(\Delta), \cdots, q_n(\Delta)$ of the polynomial

$$q(y) = p(y+\Delta) = \sum_{h=0}^{n} p_h (y+\Delta)^h = \sum_{g=0}^{n} q_g(\Delta) y^g$$

(or, equivalently, compute the values of the polynomial $p(x)$ and of all its derivatives $p^{(g)}(\Delta) = q_g(\Delta)g!$ of the order $g = 1, 2, \cdots, n$ at $x = \Delta$).

Solution. Expand the powers $(y+\Delta)^h$ above and obtain that

$$q_g(\Delta) = \sum_{h=g}^{n} p_h \Delta^{h-g} \frac{h!}{g!(h-g)!}, \qquad g = 0, 1, \cdots, n.$$

Substitute $w_{n-g} = g!q_g(\Delta)$, $u_{n-h} = h!p_h$, $v_s = \Delta^s/s!$, $j = n-h$, and $i = n-g$, and arrive at the following expressions (which are a part of a convolution problem):

$$w_i = \sum_{j=0}^{i} u_j v_{i-j}, \qquad i = 0, 1, \cdots, n.$$

Thus, the solution involves at most $4.5K \log K + K$ ops for $2n < K \le 4n$, not counting $4n-4$ multiplications and divisions by Δ and $s!$ for $s = 2, 3, \cdots, n$.

In §§10–12, we will revisit some of the best current algorithms for polynomial division, which we will restate also in the forms of matrix and power series computations. Divisions turn out to be reducible to multiplications and have almost the same computational cost (see Table 6.1). Polynomial division applies to several other computations. In particular, the Euclidean algorithm computes the gcd of two polynomials, $u(x)$ and $v(x)$, by means of recursive reduction of one of them modulo another by divisions. Relatively few leading coefficients of such polynomials are involved in each division, and this leads to the cost bound $O_A(N \log^2 N)$ for the gcd. The parallel time of this algorithm is, however, linear in N, but we may reduce the evaluation of the gcd to computing the solution to a linear system of equations with a dense structured matrix and thus arrive at NC algorithms (see [BGH], [Gat84], [P90], [P90a], and Remark 9.1 in §9 below).

In addition to the study of computing the gcd and division, multipoint evaluation and interpolation for polynomials will give us yet another occasion for showing some simple correlations among computations with polynomials and structured matrices (see §8). Polynomial computations are also closely related to manipulation with formal power series and with integers. Indeed, the power series turn into polynomials being reduced modulo a power, and a binary integer can be considered the value of a polynomial with

zero and 1 coefficients at the point $x = 2$. We refer the reader to [AHU], [Kn], and [BP,a] for these correlations.

In the remainder of this section, we will survey some effective methods for the important classical problems of approximating polynomial zeros and linear factors of a polynomial in the complex domain. These problems are closely related to each other and to approximating the eigenvalues and the singular values of a matrix (see the next section). Factorization over the rationals (see [LLL]) and over various other fields or Euclidean domains is a major subject of computer algebra [Ka], [Ka92].

Several algorithms for approximating polynomial zeros have been successfully used in practice. This experience shows the particular efficacy of the algorithm of [Dur], [Ker], and of its extensions. To display these algorithms, denote that $p(x) = \sum_{i=0}^{n} p_i x^i = \prod_{j=1}^{n}(x - x_j)$, $p_n = 1$, $r(x) = p'(x)/p(x)$, $x = [x_i, \cdots, x_n]^T$. Furthermore, let $x_j(k)$ denote the approximation to x_j computed at the kth iteration step. Then in [Dur] and [Ker],

$$x_j(k+1) = x_j(k) - p(x_j(k))/\prod_{i \neq j}(x_j(k) - x_i(k)), \qquad j = 1, \cdots, n, \tag{6.1}$$

$k = 0, 1, \cdots$. This is Newton's iteration for the Viète system of n equations in $x_1, \cdots, x_n$, obtained by equating to p_i the coefficients of x^i for $i = 0, \cdots, n-1$ in the expansion of $\prod_{j=1}^{n}(x - x_j)$ as a polynomial in x.

The iteration (6.1) has local quadratic convergence if it starts close enough to $x_1, \cdots, x_n$ and if $x_1, \cdots, x_n$ are simple zeros of $p(x)$. The order of local convergence increases to 3 and more in the following modifications of (6.1):

$$x_j(k+1) = x_j(k) - \left(r(x_j(k)) + \sum_{i \neq j}(x_i(k) - x_j(k))^{-1} \right), \quad j = 1, \cdots, n \quad \text{([Abe])},$$

$$x_j(k+1) = x_j(k) - p(x_j(k))/\prod_{i \neq j}(x_j(k) - x_i(k) - r(x_j(k))),$$

$$j = 1, \cdots, n \quad \text{([Wer])},$$

$$x_j(k+1) = x_j(k) - \left((r(x_j(k)))^{-1} - \sum_{i \neq j}(x_j(k) - x_i(k) + r(x_i(k)))^{-1} \right)^{-1},$$

$$j = 1, \cdots, n \quad \text{([Wer])}.$$

All these simple formulae can be easily implemented, and in practice, these algorithms usually converge in few steps if they start, say, with n equally spaced points of the circle $\{x : |nx - p_{n-1}| = an\}$, $a = 2\sum_{i=0}^{n} |p_i|$.

The failure of such algorithms remains at least plausible, however, and we still need an algorithm that for any input would rapidly converge right from the start. Furthermore, for the complexity theorem, we need an algorithm that supports the record complexity estimates of Table 6.1. Next, we will outline such an algorithm, based on [P87a] (see also [BP,a] and [Sc]), for the approximation within the relative error of 2^{-b} (that is, within an error bound $2^{-b} \max_{i<n} |p_i|$) to all the complex zeros x_j of an nth degree monic polynomial, $p(x) = \sum_{i=0}^{n} p_i x^i = \prod_{j=1}^{n}(x - x_j)$, $p_n = 1$. We know of no attempts to implement this algorithm so far.

It is well known that the problem of approximating polynomial zeros is ill-conditioned; specifically, the worst case precision of this computation must be at least of the

order of nb, and we will not require any higher precision of computing. The resulting complexity estimate, shown in Table 6.1, is expressed in *terms* of *both* n and b.

Unlike many other algorithms, this algorithm is insensitive to clustering the zeros of $p(x)$; it *always converges exponentially fast and right from the start.*

The algorithm exploits Weyl's geometric construction for search and exclusion [He], which itself ensures global linear convergence. Let us first specify this construction and then describe its further improvement.

We first define a square on the complex plane that contains all the zeros of $p(x)$ (we know that $\max_j |z_j| \le \max_{i<n} |p_i|$). Then we recursively partition it into four congruent subsquares. Each of them is either discarded if a test shows that it contains no zeros of $p(x)$ or is called *suspect* and is further recursively partitioned into four subsquares. The test is by Turan's algorithm that, for the cost of $O(1)$ FFTs, approximates within, say, 2 percent error to the minimum absolute value of a zero of $p(x)$. Since we may shift the variable x to any complex point for the cost of three FFTs (see above), we may also compute the desired minimum distance from a center of a square to a zero of $p(x)$. At most $4n$ suspect squares need to be examined in each Weyl's recursive step, since every zero of $p(x)$ may make at most four squares suspect in each such a step.

The side of a suspect square decreases by factor of 2 in each recursive step, and thus we arrive at the desired approximations to all the zeros of $p(x)$ within the relative error 2^{-b} for the cost $O_A(nb \log n, n)$. These bounds are quite effective for smaller b, but we may modify the approach and greatly decrease the total work and the parallel time if b is large, that is, we may decrease them by the factor of $b/\log b$, from the order of b to the order of $\log b$ (as in Table 6.1). For this, we need to ensure an accelerated convergence to an unknown smallest subsquare s of every given larger square S such that both squares S and s contain exactly the same k zeros of $p(x)$ (in particular, s is just a point if $k = 1$).

This turns out to be the critical step for improving the Weyl construction, in the cases where S is much larger than s (such cases always arise where b is large). Except for these cases, the construction soon defines new distinct connected components formed as unions of suspect squares (such a new component is defined in $O(\log n)$ steps of Weyl's construction if the ratio of the diameters of S and s is $n^{O(1)}$), and at any step there can be at most n such components, of course. Thus we will recursively perform the accelerated contraction of S towards s (until we output the desired approximations to the zeros of $p(x)$). Each contraction is followed by a series of $O(\log n)$ Weyl's steps, which ensure partition of the union of the suspect squares into more and more components. Since each time the number of components grows but never exceeds n, we will isolate all the zeros of $p(x)$ in at most $n - 1$ partition stages.

It remains to ensure the accelerated contraction of S towards s. For this purpose, we first approximate the average value M of the k zeros of $p(x)$ in S and then the maximum distance d from M to such a zero. Both approximations to M and to d can be computed by means of numerical integration of $xp'(x)/p(x)$ and of $(x - M)^2 p'(x)/p(x)$, respectively. For the contour of integration we choose a circle, C, containing exactly these k zeros of $p(x)$. This enables us to reduce the integration to FFTs. The errors of the integration are proportional to g^{Q-1} or g^{Q-2}, where Q is the number of points used for each FFT and where $g < 1$ is the ratio of the radius of the circle C (being the denominator) and of the distance from its center to a zero of $p(x)$ nearest to C (being the numerator). We may decrease g by moving the center of C closer to the zero of $p(x)$. This is easy to ensure when the current approximations to M and d give us a smaller region that includes the yet unknown smallest square s containing the k zeros of $p(x)$. We recursively repeat such a process, which very rapidly converges to the square s. This

enables us to deduce the complexity estimates that are very close to the ones in Table 6.1.

To arrive at the estimates in Table 6.1 themselves, we just need to replace the contraction of the square S towards s by splitting $p(x)$ into the product of two polynomials $f(x)$ and $g(x)$, $f(x)$ sharing with $p(x)$ exactly the k zeros of $p(x)$ in S. The sums of the ith powers of such zeros for $i = 1, 2, \cdots, k$ are approximated by means of contour integration. Having these power sums, we immediately compute approximations to the coefficients of $f(x)$ by using Newton's identities [He], [H]. The resulting approximations to $f(x)$ and $g(x)$ are then rapidly improved by means of Newton's iteration (see [BP,a], [Sc] for some error and complexity estimates of this process). Note that the recursive splitting is least effective when $k = 1$ in all its steps (in which case we need $n-1$ splitting steps), whereas we only need $O(\log n)$ steps if $\deg f(x) = \deg g(x)$ in all steps.

This approach leads us to the record complexity bounds of Table 6.1 for approximating all the zeros of $p(x)$ and (after some modification) a single zero of $p(x)$. For approximating a single zero of $p(x)$, there are also other powerful and practical techniques, in particular, Newton's iteration

$$x(k+1) = x(k) - p'(x(k))/p(x(k)) \tag{6.2}$$

(compare [S85], [R] on its convergence), which can also be incorporated in the above construction of [P87a] as a substitution for the contour integration.

All the known algorithms, including ones of [Sc], [P87a], and Newton's iteration, were not in NC, but the recent ingenious algorithm of [N] (which followed the earlier work of [BFKT]) gave us a long awaited NC (although so far processor inefficient) solution to the problem of approximation to all the zeros of $p(x)$. (NC algorithms for this problem must run in time polylogarithmic in *both* n and b.)

Approximating the zeros of $p(x)$ is sometimes called approximate factorization of $p(x)$ over the complex field, in which case approximation error is defined as, say, $e_n = \sum_{i=0}^{n} |p_i - p_i^*|$, where $\sum_{i=0}^{n} p_i^* x^i = \prod_{j=1}^{n}(x - x_j^*)$, $x_i^*, \cdots, x_n^*$ denote the computed approximations to the zeros of $p(x)$. In this case the problem is better conditioned, and its computational complexity decreases by roughly n times if we require that $e_n < 2^{-n} \sum_i |p_i|$, rather than $\max_j |x_j - x_j^*| < 2^{-n} \sum_i |p_i|$. Many details on this topic can be found in the paper [Sc], which is still incomplete, and in [BP,a]. [P91b] presents a randomized algorithm for this problem.

7. Arithmetic complexity of matrix computations. We will continue reviewing some polynomial computations in §§9–12, but our next topic is computations with general matrices (in this section) and with structured matrices (in §§8 and 9). We will observe some correlation between the computations with polynomials and structured matrices.

Matrix-by-vector and matrix-by-matrix multiplications play the same fundamental role for matrix computations as FFT and convolution play for polynomial computations, but the current improvement of the classical multiplication algorithms for matrices and vectors is not as significant as for polynomials. Furthermore, the straightforward algorithm for $N \times N$ matrix-by-vector multiplication supports the *optimum* cost bounds of $2N^2 - N$ arithmetic operations and $O_A(\log N, N^2/\log N)$ (compare Example 1.3, however).

It is increasingly popular to use matrix-by-matrix multiplication and matrix inversion as the blocks of many algorithms, since these operations have been very effectively implemented on (loosely coupled) multiprocessors. Indeed, suppose that $p = k^3$ processors can be used for multiplication of a pair of $k \times k$ block matrices with $h \times h$ blocks. This can be reduced to $p = k^3$ multiplications of $h \times h$ blocks, one per each processor.

Performing matrix multiplication in the straightforward way, each processor uses $2h^2$ data fetches, h^3 multiplications, and $h^3 - h^2$ additions, that is, roughly, h operations per fetch. This means relatively few data movements for larger h, which implies the efficacy and explains the popularity of the multiprocessor implementation of block matrix multiplication, as well as the closely related operation of matrix inversion. Moreover, the straightforward $N \times N$ matrix multiplication, performed for the cost of $2N^3 - N^2$ arithmetic operations and$O_A(\log N, N^3/\log N)$, is now getting replaced by faster methods that involve $O(N^\beta)$ arithmetic operations, where for the currently implemented algorithms, $\beta = \log_2 7 = 2.807\ldots$ (Strassen) and where $\beta < 2.78$ in another group of algorithms, also ready for practical implementation [LPS].

Weak (but sufficient in most applications) numerical stability of these and of many other fast matrix multiplication algorithms has been formally proven [Brent70], [BL80] and has been shown in numerous experiments to be actually quite strong. All the fast $O(N^\beta)$ matrix multiplication algorithms allow their parallel implementation on the PRAMs for the cost $O_A(\log N, N^\beta)$ [PR,85], [P87b], and theoretically the exponent β can be decreased to $\beta = 2.375\ldots$ [CW], although immense overhead constants are hidden in this "O" notation for $\beta < 2.775$.

The bounds $O_A(N^\beta)$ and $O_A(\log N, N^\beta)$, $\beta < 2.376$, as $N \to \infty$, are fundamental for estimating the asymptotic complexity of numerous algebraic and combinatorial computations ultimately reduced to matrix multiplications. In particular, the asymptotic complexity estimate $O_A(N^\beta)$, $\beta < 2.376$, has been extended to computing the determinant and all the coefficients of the characteristic polynomial of a matrix, its triangular and orthogonal factorization, its reduction to the Hessenberg form, its rank, its inverse, and its Moore–Penrose generalized inverse (which implies computing the solution and a least-squares solution to a linear system of equations). The same asymptotic bound $O_A(N^\beta)$ has also been extended to numerous combinatorial and graph computations [P84], [P87b], [Datta], [KUW], [GP85], [GPa], [MVV], [PR], [BP,a].

The extensions to matrix factorization and to computing the inverse and the determinant of a matrix rely on 2×2 block Gauss–Jordan elimination that leads to the following factorization, used already more than half a century ago and now implemented, for instance, on the Connection Machine CM-2:

$$A = \begin{bmatrix} B & C \\ D & E \end{bmatrix} = \begin{bmatrix} I & O \\ DB^{-1} & I \end{bmatrix} \begin{bmatrix} B & O \\ O & S \end{bmatrix} \begin{bmatrix} I & B^{-1}C \\ O & I \end{bmatrix}, \tag{7.1}$$

$$A^{-1} = \begin{bmatrix} I & -B^{-1}C \\ O & I \end{bmatrix} \begin{bmatrix} B^{-1} & O \\ O & S^{-1} \end{bmatrix} \begin{bmatrix} I & O \\ -DB^{-1} & I \end{bmatrix}. \tag{7.2}$$

Here $S = E - DB^{-1}C$ is the Schur complement of B. If A is a Hermitian positive definite (an h.p.d.) matrix $A^H = A$, then so are B and S, and furthermore,

$$\|A\|_2 \geq \max\{\|B\|_2, \|S\|_2\},$$
$$\|A^{-1}\|_2 \geq \max\{\|B^{-1}\|_2, \|S^{-1}\|_2\}.$$

Therefore, the factorization (7.1), (7.2) can be recursively applied to the h.p.d. matrices B and S in a numerically stable process. Likewise, numerical stability of the factorization (7.1), (7.2) can be proven if A is a diagonally dominant matrix and/or positive definite but not a Hermitian matrix.

We may always assume the blocks of 2×2 block matrices balanced in size. Then (7.1) reduces the inversion of an h.p.d. (and/or diagonally dominant) matrix A to the inversion of two half-size h.p.d. (and/or diagonally dominant) matrices B and S and to six multiplications of half-size matrices. This recursively leads us to the bound $O_A(N^\beta)$ for $N \times N$ matrix inversion [St69], based on an effective and numerically stable algorithm. Note that $N \times N$ matrix multiplication is not easier than $3N \times 3N$ matrix inversion since

$$\begin{bmatrix} I & A & O \\ O & I & B \\ O & O & I \end{bmatrix}^{-1} = \begin{bmatrix} I & -A & AB \\ O & I & -B \\ O & O & I \end{bmatrix}$$

[BM], and both problems have the same asymptotic complexity as the evaluation of the determinant of an $N \times N$ matrix [L], [BS].

The Cayley–Hamilton theorem and the Moore-Penrose conditions [GL] can be exploited in order to compute [for the cost $O_A(N^\beta)$] the Moore–Penrose generalized inverse A^+ of A. Indeed,

$$c_{N-r}A^+ = \sum_{i=N-r+1}^{N-1} \left(\frac{c_{N-r+1}}{c_{N-r}} c_i - c_{i+1} \right) A^{i-N+r} + \frac{c_{N-r+1}}{c_{N-r}} A^r, \tag{7.3}$$

where $\det(\lambda I - A) = \lambda^N + \sum_{i=N-r}^{N-1} c_i \lambda^i$, and where $c_{N-r} \neq 0$, provided that A is a Hermitian matrix, that is, $A^H = A$. Here and hereafter A^H denotes the conjugate transpose of A. The assumption that $A^H = A$ is no loss of generality since $A^+ = (A^H A)^+ A^H$ and since $\begin{bmatrix} O & A \\ A^H & O \end{bmatrix}^+ = \begin{bmatrix} O & (A^H)^+ \\ A^+ & O \end{bmatrix}$. Note that over the complex, real, and rational fields, $r = \text{rank } A = \text{trace } (A^+ A)$, and $\mathbf{x} = A^+ \mathbf{b}$ is a least-squares solution to a linear system $A\mathbf{x} = \mathbf{b}$. Therefore, computing such a solution and rank A has the complexity bounded by $O_A(N^\beta)$.

As we mentioned, the practical or potentially practical range for the exponent β is presently from 2.775 to 2.808 (not from 2.376), but these computations may be considered for practical use only where the (weak) numerical stability of matrix multiplication can be extended. This seems to be the case for several matrix inversion and factorization algorithms but not for computations involving the coefficients of the characteristic polynomial, as in (7.3) above. Numerically stable practical evaluation of A^+ may rely on computing the SVD of A or, alternatively, on Newton's iteration, whose each step essentially amounts to two matrix multiplications:

$$X_0 = \frac{A^H}{\|A^H A\|_1}, \quad X_{k+1} = X_k(2I - AX_k), \quad k = 0, 1, \cdots, K-1, \tag{7.4}$$

so that $I - AX_{k+1} = (I - AX_k)^2$ [Be]. An improved modification of (7.4) numerically converges to A^+ in $\log_2 \text{cond}_2 A$ steps (7.4), $\text{cond}_2 A = \|A\|_2 \|A^+\|_2$, or if A is h.p.d., in about $\frac{1}{2} \log_2 \text{cond}_2 A$ steps [PS]. Thus, such a modification seems effective and can be recommended for the generalized inversion of well-conditioned matrices A, particularly, on parallel computers, on which matrix multiplication can be very effectively performed. It is interesting that $X_k = \sum_{i=0}^{s(k)} (I - X_0 A)^i X_0$, $s(k) = 2^k - 1$ under (7.4), whereas

$X_k = p(A^HA)A^H$, $1 - zp(z)$ is a scaled Chebyshev polynomial of degree 2^k in the improved scheme of [PS]. The algorithms of [PS] have been implemented and tested by R. Schreiber at RIACS, Moffet Field, California.

The algorithms of [PS] also compute the matrices $A(\epsilon)$ and $(A(\epsilon))^+$, where the unknown matrix $A(\epsilon)$ is defined by zeroing all the singular values of A that are less than ϵ. Then again, $A(\epsilon)$ and $(A(\epsilon))^+$ are computed in [PS] without computing the SVD of A.

If $\text{cond}_2 A$ is large, the algorithm of [PS] may converge slowly. In this case, an effective alternative is given in [IMH], or we may rely on the use of Algorithm 4.2 (for solving linear systems, we may similarly apply Algorithm 4.1), provided that the input and therefore the output values are rational and that we agree to use rational arithmetic and p-adic lifting.

Particularly effective is the parallel implementation of the latter algorithm. Indeed, inverting A modulo p is simple unless p is very large, since this only required $\lceil \log_2 p \rceil$-bit precision, and all other stages are easy to perform concurrently, since they are reduced to matrix multiplications. In fact, this approach supports the record estimates for the parallel Boolean complexity of randomized matrix inversion [P87b], [BP,a]. With a random choice of a prime p of the order of, say, N^2, the singularity of the matrix A mod p can be made an unlikely event if A is nonsingular.

The algorithm can be practically used as long as the modular arithmetic is available. Since we may scale the input matrix A, we may assume its entries are integers. Then the matrix $\text{adj } A = A^{-1} \det A$ is filled with integers and can be recovered from (adj A) mod p^H if $\frac{1}{2}p^H$ exceeds the maximum absolute value of an entry of adj A. (An alternative way relies on computing (adj A) mod $p_i^{d(i)}$ for several primes p_i and applying the Chinese remainder algorithm (compare §4).)

To compute (adj A) mod p^H, for an $N \times N$ matrix A, we may apply the randomized algorithm of [P90a] or [KP]. This algorithm also computes A^{-1} and supports the record (and nearly optimum) parallel complexity bounds, $O_A(\log^2 N, N^\beta)$, assuming the cost bounds $O_A(\log N, N^\beta)$ for $N \times N$ matrix multiplication.

The algorithm has been implemented and tested at Rensselaer Polytechnic Institute, Troy, New York. It first computes the coefficients of $m_A(\lambda)$, the minimum polynomial of A. $m_A(\lambda)$ either coincides with the characteristic polynomial $c_A(\lambda) = \det(\lambda I - A)$ or can be used similarly to it in order to compute A^{-1} (or A^+) and $\det A$. (In particular, (7.3) can be extended, respectively.) This evaluation of A^{-1} and $\det A$ can be performed modulo an integer p and then lifted to modulo p^H by using Algorithms 4.2 or 4.1 ($\det A$ is computed from an auxiliary linear system of equations).

To compute the coefficients of $m_A(\lambda)$, we first choose two random column vectors $\mathbf{u}$ and $\mathbf{v}$, then compute $A^K\mathbf{v}$ for $k = 0, 1, \cdots, 2N-1$, (for the cost of $O(\log N)$ multiplications of $M \times N$ by $N \times N$ matrices where $M \leq N$), then compute $\mathbf{u}^T A^k \mathbf{v}$, $k = 0, 1, \cdots, 2N-1$, and finally, compute the coefficients of $m_A(\lambda)$ by solving, for the cost $O_A(\log^2 N, N^2/\log N)$, an auxiliary Toeplitz linear system of equations (see the next section).

The resulting algorithm can be extended to several other matrix computations, such as computing matrix rank, a least-squares solution to a linear system of equations, the PLU- and the QR-factorizations of a matrix (see [BP,a]). The same applies to the algorithm of [IMH].

To conclude this section, we will comment on approximating the eigenvalues of a matrix A, which are the zeros of the characteristic polynomial, $c_A(\lambda) = \det(\lambda I - A)$. The methods that use the coefficients of the characteristic polynomial of A are practically avoided because of the numerical stability problems. This, however, does not apply to

the iterative algorithms (6.1), (6.2), and the ones of [Abe] and [Wer]. Indeed, all we need in order to extend these algorithms to approximating the eigenvalues is to compute $c_A(\lambda)$ and $c_A(\lambda)/c'_A(\lambda) = [\text{trace}((\lambda I - A)^{-1}]^{-1}$ for the values λ that approximate the eigenvalues of A.

The algorithm of [P87a] can also be extended to approximating the eigenvalues of A, as long as we can effectively estimate the distance from a fixed complex point to the nearest eigenvalues of A. This step seems to be difficult for the general matrix A but is not hard in the important case of a Hermitian or real symmetric A, and the algorithm of [P87a] has been extended, respectively (in [PD]). The symmetric matrix eigenvalue problem also covers the evaluation of the SVD of a matrix.

The solution customarily starts with the reduction of the input matrix A to the tridiagonal form. The record (and nearly optimum) parallel and sequential complexity estimates for approximating the eigenvalues of a symmetric tridiagonal matrix have been recently obtained in [BP91] by using a new version of the divide-and-conquer algorithm. The new algorithm (in a further modified form) has been tested by Bini at the University of Pisa, Italy, in a series of numerical experiments [BP,b]. The algorithm seems to be particularly attractive for parallel implementation.

8. Some basic computations with structured matrices. The arithmetic cost of matrix computations and the storage space involved dramatically decrease if the matrices are sparse or structured. We refer the reader to [GeLi], [LRT], and [PR88] (containing further bibliography) on the sparse case. We will next revisit the dense structured case, where we typically need $O(N)$ words of storage space and from $O_A(N \log N)$ to $O_A(N \log^2 N)$ time (versus the much larger space bounds of $O(N^2)$ and from $O_A(N^2)$ to $O_A(N^\beta)$ time, for $\beta > 2$, in the general case). In particular, the cited improved bounds apply to computing $A\mathbf{b}$ and $A^{-1}\mathbf{b}$, where $\mathbf{b}$ is a vector and A is a matrix with the Toeplitz, Hankel, Vandermonde, or generalized Hilbert structure (see below). Such matrices and such computations are ubiquitous in sciences and engineering, so that the practical impact of the cited improvement is very high. In the important special case of $n \times n$ band Toeplitz matrices with bandwidth $k = o(n)$, the complexity of computations decreases even more, for instance, to $O_A(k \log k \log n)$ for computing the value of a characteristic polynomial of such a matrix at a given point [BPc], [BP88].

Technically, the computations with dense structured matrices are closely related to polynomial computations and greatly exploit FFT.

Let us recall some definitions and results. Let $(W)_{i,k}$ denote the (i,k) entry of a matrix W. Then for all (i,k) entries, $(T)_{i,k} = t_{i-k}$ for Toeplitz matrices T, $(C)_{i,k} = c_{i-k \bmod N}$ (for a fixed N) for circulant matrices C (which form an important subclass of Toeplitz matrices), $(H)_{i,k} = h_{i+k}$ for Hankel matrices H, $(V)_{i,k} = v_i^k$ for Vandermonde matrices V and $(B)_{i,k} = 1/(s_i - t_k)$ for generalized Hilbert matrices B. Each such an $N \times N$ matrix is completely defined by one or two vectors of dimension N. Any $N \times N$ Hankel matrix can be turned into a Toeplitz matrix by means of interchanging its columns (or rows) s and $N + 1 - s$, for $s = 1, 2, \cdots, N$.

The cost bound $O_A(N \log^2 N)$ for computing $A\mathbf{b}$ and $A^{-1}\mathbf{b}$ for a Vandermonde matrix $A = V$ and for a generalized Hilbert matrix $A = B$ can be deduced by means of the reduction of the problem to polynomial evaluation and interpolation (recall Table 6.1 and see [Gast60] and [Ger87] or [BP,a] in the generalized Hilbert case, $A = B$, whereas in the Vandermonde case, observe that $V\mathbf{b}$ for $\mathbf{b} = [b_0, \cdots, b_{N-1}]^T$ is the vector of the values of the polynomial $\sum_{k=0}^{N-1} b_k v^k$ for $v = v_i$, $i = 0, \cdots, N-1$).

The algorithms supporting the bound $O_A(N \log^2 N)$ for the polynomial evaluation and interpolation with N nodes [AHU], [BM], [BP,a] (and consequently for computing

$A\mathbf{b}$ and $A^{-1}\mathbf{b}$ with $A = V$ and $A = B$) can be safely used if they are performed in rational arithmetic (see §4), but lead to severe problems of numerical stability in the presence of roundoff errors. In the latter case alternate numerically stable algorithms can be applied for the cost of $O_A(N^2)$ [GL]. Moreover, some other alternative techniques of [Rok85], [Rok] enable us to compute approximations to $V\mathbf{b}$, $V^{-1}\mathbf{b}$, and $B\mathbf{b}$ for a real Vandermonde matrix V and a generalized Hilbert matrix B. The complexity then depends on the output error bound ϵ and is bounded by $O(N)$ for a fixed constant ϵ.

Let us comment on computing $B\mathbf{b}$ in [Rok85]. Rewrite $B\mathbf{b}$ as

$$R(x_l) = \sum_{k=0}^{K-1} \frac{a_k}{x_l - w_k}, \qquad l = 0, \cdots, L-1;$$

denote $\alpha = \max_k |a_k/w_k|$, $J = \lceil \log(\alpha K/(1-q)\epsilon)/\log(1/q) \rceil$, where $|x_l/w_k| < q < 1$ for all k and l. Then $(2K+2L)J - 2L$ arithmetic operations suffice for approximating to $R(x_l)$ within ϵ for all l [Rok85]. The algorithm relies on simple but effective techniques of summation reordering. Here are some details:

Substitute the expressions

$$\frac{1}{x - w_k} = -\frac{1}{w_k} \sum_{j=0}^{\infty} \left(\frac{x}{w_k}\right)^j, \qquad k = 0, 1, \cdots, K-1,$$

and represent the sum of partial fractions $R(x)$ as follows:

$$R(x) = \sum_{j=0}^{\infty} A_j x^j \tag{8.1}$$

where

$$A_j = -\sum_{k=0}^{K-1} a_k / w_k{}^{j+1}. \tag{8.2}$$

The power series (8.1) converges for sufficiently small $|x|$. Approximate $R(x)$ by the J-term finite power series and estimate the errors in terms of $|x_l/w_k|$ and α. Under our assumptions about the values J, $|x_l/w_k|$, and $|a_k/w_k|$, we have

$$E_J(x_l) = \left| R(x_l) - \sum_{j=0}^{J-1} A_j x_l^j \right| \le \alpha K q^J/(1-q) \le \epsilon.$$

Now, it remains to compute first the coefficients $A_0, \cdots, A_{J-1}$ of (8.2) by using $2JK$ operations and then to approximate the values $R(x_l)$ of (8.1) for $l = 0, 1, \cdots, L-1$ by using $2JL - 2L$ ops.

If $|w_k/x_l| < q < 1$ for a constant q and for all k and l, we will arrive at a similar algorithm by substituting $1/(x - w_k) = (1/x)\sum_{j=0}^{\infty}(w_k/x)^j$. [AGR], [CGR], and [ODR] show some important applications of both algorithms (and, moreover, the algorithms are extended in order to include approximation to logarithmic functions); [AR] presents some further extensions to the low precision multiplication of a matrix W with a certain structure by a vector.

Next, let us compute $A\mathbf{b}$ and $A^{-1}\mathbf{b}$ for the cost of $O_A(\log N, N)$, where $A = C$ is a circulant matrix. This is reduced to three FFTs.

THEOREM 8.1 ([Da74]). *Let* $\mathbf{c}$ *denote the first column of an* $N \times N$ *circulant matrix* C. *Let* Ω *denote the* $N \times N$ *Fourier matrix,* $(\Omega)_{i,j} = \omega^{ij}/\sqrt{N}$, $(\Omega^H)_{i,j} = \omega^{-ij}/\sqrt{N}$, $i, j = 0, 1, \cdots, N-1$, $\omega^N = 1$, $\omega^s \neq 1$ *for* $0 < s < N$. *Then* $\Omega^H\Omega = I$, $\Omega^H C\Omega = D$, $D = diag(d_0, \cdots, d_n)$, *where* $d_i = (\mathbf{d})_i$, $\mathbf{d} = \Omega^H\mathbf{c}/\sqrt{N}$.

Since we may embed any Toeplitz matrix $A = T$ into a circulant matrix, the result for $C\mathbf{b}$ is extended to computing $T\mathbf{b}$. An alternative way (slightly inferior but still supporting the bound $O_A(\log N, N)$ for computing $T\mathbf{b}$) is by embedding a general $N \times N$ Toeplitz matrix into a more special $(3N-2) \times N$ Toeplitz matrix U of the vector equation

$$\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ \vdots \\ \vdots \\ w_{m+n} \end{bmatrix} = \begin{bmatrix} u_0 & \ddots & 0 \\ u_1 & & \\ \vdots & \ddots & u_0 \\ u_m & \ddots & u_1 \\ & \ddots & \vdots \\ 0 & & u_m \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_n \end{bmatrix}, \tag{8.3}$$

where $N = n+1 = m-n+1$, $m = 2N-2$. Then $T\mathbf{v}$ is a part of $U\mathbf{v}$, and $U\mathbf{v}$ is the convolution of the vectors $\mathbf{u} = [u_0, \cdots, u_m]$ and $\mathbf{v}$.

The first $N = n+1$ equations of (8.3) form a linear triangular Toeplitz system $T\mathbf{v} = \mathbf{w}$. If $w_0 = 1$, $w_i = 0$ for $i > 1$, then $\mathbf{v}$ equals the first column of T^{-1}, which is the coefficient vector of the reciprocal polynomial $v(x) = \sum_{i=0}^{n} v_i x^i$ such that $v(x)w(x) = 1 \bmod x^N$, $w(x) = \sum_{i=0}^{n} w_i x^i$, $N = n+1$. Computing $v(x)$ essentially amounts to polynomial division and costs $O_A(\log N \log^* N, N/\log^* N)$ (see Table 6.1 and §§10–12).

For a general Toeplitz matrix T, there are several algorithms that compute $T^{-1}\mathbf{b}$ for the cost $O_A(N \log^2 N)$ [BGY], [BA], [Mus], and [dH], [AG,b]. The algorithm of [AG,b] supports this bound with a small overhead constant but still meets tough competition from some $O_A(N^2)$ algorithms unless N grows to several hundreds [CB]. We refer the reader to[Bun] on the numerical stability study and to [BdB], [Ch], [Strang], [ChS], and [PS92] on iterative solution of some more special Toeplitz linear systems.

All the cited algorithms for $T^{-1}\mathbf{b}$ (where T is the general Toeplitz matrix) require at best linear parallel time bound N. It is possible to compute $T^{-1}\mathbf{b}$ (for a nonsingular T), and, more generally, to compute $T^{+}\mathbf{b}$, the least-squares solution to $T\mathbf{x} = \mathbf{b}$, for the cost $O_A(\log^2 N, N^2/\log N)$ by relying on Newton's iteration (7.4) and on (7.3) (see [P89b], [P90], [P90a]). Newton's iteration (but without involving (7.3)) may also be used as a very effective means of refining approximations to T^{-1} and $T^{-1}\mathbf{b}$ (see the next section).

9. Structured matrices and the shift and displacement operators. The structured matrices of the previous section are naturally associated with some linear operators of displacement and scaling. This powerful approach, due to [KKM], [KVM], leads us to a unified treatment of computations with various structured matrices.

Hereafter, let $D(\mathbf{v}) = \text{diag}(v_1, \cdots, v_N)$. Let $L(\mathbf{v})$ and $V(\mathbf{v})$ denote the lower triangular Toeplitz matrix with the first column $\mathbf{v}$ and the square Vandermonde matrix $[v_i^k]$ with the second column $\mathbf{v}$, respectively, for $\mathbf{v} = [v_1, \cdots, v_N]^T$. Let J and Z denote the reflection matrix and the lower displacement matrix, respectively, both filled with zeros except for the antidiagonal of J and the first subdiagonal of Z, filled with ones, so that

$J\mathbf{v} = [v_N, \cdots, v_1]^T$, $Z\mathbf{v} = [0, v_1, \cdots, v_{N-1}]^T$ for a vector $\mathbf{v} = [v_1, \cdots, v_N]^T$, and we will denote $\mathbf{v}^{-1} = [v_1^{-1}, \cdots, v_N^{-1}]^T$.

Following [KKM], [KVM], [GKKL], consider the operators

$$F_{ZZ^T}(A) = A - ZAZ^T, \tag{9.1}$$

$$F_{Z^TZ}(A) = A - Z^TAZ, \tag{9.2}$$

$$F_{\mathbf{v},Z}(A) = D(\mathbf{v})A - AZ, \tag{9.3}$$

$$F_{Z,\mathbf{v}}(A) = ZA - AD(\mathbf{v}), \tag{9.4}$$

$$F_{\mathbf{s},\mathbf{t}}(A) = D(\mathbf{s})A - AD(\mathbf{t}). \tag{9.5}$$

Let $\mathbf{e} = [1, 1, \cdots, 1]^T$, $\mathbf{u}^T(k)$ be the unit coordinate vector with the kth component equal to 1. Observe that rank $F_{ZZ^T}(A) =$ rank $F_{Z^TZ}(A) \leq 2$ if A is a Toeplitz matrix; $F_{\mathbf{v},Z}(A) = D^{N-1}(\mathbf{v})\mathbf{e}\mathbf{u}^T(N-1)$ and thus has rank at most 1 if A is the $N \times N$ Vandermonde matrix $V(\mathbf{v})$, defined by a vector $\mathbf{v}$; and $F_{\mathbf{s},\mathbf{t}}(A) = \mathbf{e}\mathbf{e}^T$ and thus also has rank 1 if A is a generalized Hilbert matrix, $(A)_{ij} = 1/(s_i - t_j)$. These observations suggest measuring Toeplitz-like, Vandermonde-like, and Hilbert-like structures of any matrix A by the ranks of the matrices $F_{ZZ^T}(A)$ (or $F_{Z^TZ}(A)$), $V_{\mathbf{v},Z}(A)$, and $F_{\mathbf{s},\mathbf{t}}(A)$, respectively. The smaller this rank, the closer the structure of a given matrix to the structures of Toeplitz, Vandermonde, and generalized Hilbert matrices, respectively. In [BP,a] some other operators replace ones of (9.1) and (9.2), and substantial advantages of such a replacement have been shown.

Representing the $N \times N$ matrix $F(A)$ of rank r as GH^T, where F is an operator of (9.1)–(9.5), G and H are a pair of $N \times r$ matrices (called a *generator of length* r for $F(A)$ or an *F-generator of length* r for A), we may immediately recover the matrix A. In particular, let $\mathbf{g}_i$ and $\mathbf{h}_i$ denote the ith columns of the matrices G and H, respectively. Then

$$A = \sum_{i=1}^{r} L(\mathbf{g}_i)L^T(\mathbf{h}_i) \quad \text{if } F = F_{ZZ^T} \qquad \text{([KKM])}, \tag{9.6}$$

$$A = \sum_{i=1}^{r} L^T(J\mathbf{g}_i)L(J\mathbf{h}_i) \quad \text{if } F = F_{Z^TZ} \qquad \text{([KKM])}, \tag{9.7}$$

$$A = \sum_{i=1}^{r} L(\mathbf{g}_i)D^{-1}(\mathbf{v})V^T(\mathbf{v}^{-1})D(\mathbf{h}_i) \quad \text{if } F = F_{Z,\mathbf{v}} \qquad \text{([GKKL])}.$$

Suppose that we deal with $N \times N$ Toeplitz-like matrices given with generators of smaller length r for their F_{ZZ^T}-images. Then we may operate with the generators rather than with the matrices, dealing with $2rN$ entries of G and H, rather than with N^2 entries of A. Due to (9.6), (9.7), we may multiply A by a vector for the cost $O_A(\log N, rN)$, rather than $O_A(\log N, N^2/\log N)$. This is effective for smaller r, and r is small for many important non-Toeplitz matrices. For instance, rank $F_{ZZ^T}(A)$ is at most 2 for the inverse of a Toeplitz matrix, at most 4 if A is a product of two Toeplitz matrices, and at most $m+n$ if A is an $m \times n$ block matrix with Toeplitz blocks.

Given a scalar α and two generators of length r_1 for $F(A)$ and length r_2 for $F(B)$, we may immediately define a generator of length $r_1 + r_2$ for $F(A + \alpha B)$; furthermore, if F is an operator of (9.1) or (9.2), we may compute (for the cost $O_A(\log n, n(r_1 + r_2)^2)$) a generator of length at most $r_1 + r_2 + 1$ for $F(AB)$ [CKL-A]. Indeed,

$$\begin{aligned} F(AB) &= AB - ZABZ^T \\ &= 0.5(A - ZAZ^T)(B + ZBZ^T) + 0.5(A + ZAZ^T)(B - ZBZ^T) \\ &\quad + ZA(I - Z^TZ)BZ^T, \end{aligned}$$

and $I - Z^TZ = \mathbf{u}(n)\mathbf{u}^T(n)$, $\mathbf{u}(n) = [0, \cdots, 0, 1]^T$, which gives us the desired generator for $F(AB)$.

This leads to the extension of the effective algorithms of [BA], [Mus], and [AG,b] to the solution of Toeplitz-like linear systems $A\mathbf{x} = \mathbf{b}$ for the cost $O_A(Nr^h \log^2 N)$. Here h takes the values 2 or 3 (depending on the algorithm), provided that we are given a generator of length r for $F_{ZZ^T}(A)$.

For instance, such an extension is immediate for the algorithm of [BA], based on the recursive factorization (7.1), (7.2) and on the observation that the inverse matrix S^{-1} of the Schur complement S of (7.1) is a submatrix of A^{-1} and that consequently rank $F_{ZZ^T}(A^{-1}) \geq$ rank $F_{ZZ^T}(S^{-1})$.

Let us revisit Newton's iteration (7.4), $X_{k+1} = X_k(2I - AX_k)$, where A and X_0 are Toeplitz matrices given with length 2 generators for $F(A)$ and $F(X_0)$, $F = F_{ZZ^T}$. (This can be extended to the Toeplitz-like case.) We may recursively compute generators of length $r_k \leq 2r_{k-1} + 5 \leq 2^{k*}7 - 5$ for $F(X_k)$, $k = 0, 1, 2, \cdots$, for the cost $O_A(\log N, N\sum_{h=1}^{k} r_h^2) = O_A(\log N, 4^k N)$.

Thus, the first few steps (7.4) still involve only Toeplitz-like matrices and have lower complexity, but the displacement rank of X_k grows as k grows, so that the kth step becomes costly for larger k. To keep its cost lower for larger k, for which $\|X_k - A^{-1}\|$ is small, we may compute a close approximation $\tilde{X}_k$ to X_k such that rank $F(\tilde{X}_k) =$ rank $F(A^{-1}) = 2$, $\|X_k - \tilde{X}_k\|_2 = O(\|X_k - A^{-1}\|_2)$. Then we restart the iteration (7.4) with $\tilde{X}_k$ replacing X_k. Due to the latter properties of $\tilde{X}_k$, the iteration (7.4) is now simplified since $\tilde{X}_k$ is structured, and its convergence is preserved since $\tilde{X}_k$ is close to X_k. The transition from X_k to $\tilde{X}_k$ may rely on the Gohberg–Semencul formula for the inverse of Toeplitz matrices [P89b], on its extension (itself of independent interest) in [BP,a], or on computing the SVD of $F(X_k)$ and zeroing some smallest (positive) singular values [P88b], [P93].

It remains to find a good initial approximation to A^{-1}, which, for instance, for many Toeplitz matrices A, encountered in signal processing applications, are given by readily invertible band Toeplitz matrices. In some other important applications to signal processing, the matrix A is slowly updated, and the user ought to update A^{-1} in real time, too. Then the currently available value of the inverse A^{-1} can be used as an initial approximation to the inverse of the updated matrix A. In such cases, where a good initial approximation to A^{-1} is readily available, the above approach to the numerical solution of a linear system $A\mathbf{x} = \mathbf{b}$ has parallel cost $O_A(\log^2 N, N)$, is strongly stable numerically, and thus seems to be highly effective.

In the general case, we may arrive at the desired initial approximations by using homotopy techniques, and this leads us to NC algorithms with N processors for solving any well-conditioned Toeplitz-like linear system whose $N \times N$ coefficient matrix A is given with a fixed constant length generator $F_{ZZ^T}(A)$ (see [P88b], [P89b]). These algorithms are numerically stable, too, but may involve large overhead constants in their

cost estimates. This leaves open the problem of devising practical NC algorithms for all Toeplitz-like linear systems.

For the general Toeplitz-like (and, possibly, ill-conditioned) linear system $A\mathbf{x} = \mathbf{b}$, we may compute its least-squares solution as follows. First compute (for the cost $O_A(\log^2 N, N^2)$) the generators of $F(A_i^{-1})$ for the auxiliary $N \times N$ matrices $A_i = I - h\zeta^i A$, $i = 0, 1, \cdots, 2N-1$, where $|h|$ is the small scalar, and ζ is a primitive $(2N)$th root of 1. The desired initial approximation to A_i^{-1} is given by the identity matrix I. Then (for the cost $O_A(\log N, N^2)$) recover trace(A^j), $j = 1, 2, \cdots, N$, and the coefficients of the characteristic polynomial of A, $\det(\lambda I - A) = \sum_{i=0}^{N} c_i \lambda^i$, as well as the vectors $A^j\mathbf{b}$ for $j = 1, 2, \cdots, N$ (see [P90, Appendix A], on the transition from the traces to the coefficients). Finally, compute $A^+\mathbf{b}$ by applying (7.3). The overall cost of this solution is $O_A(\log^2 N, N^2)$. The solution involves the coefficients of the characteristic polynomial of A and does not allow us to use modular arithmetic. This deficiency is repaired in [P90a]. The inversion of $2N$ auxiliary matrices A_i is replaced in [P90a] by the inversion modulo λ^{N+1} of the single matrix polynomial $B = I - \lambda A$. The coefficients of the matrix polynomial $B^{-1} \bmod \lambda^{N+1} = I + \lambda A + (\lambda A)^2 + \cdots$ gives us trace (A^j) and $A^j\mathbf{b}$ for $j = 1, \cdots, N$, and we compute $B^{-1} \bmod \lambda^{N+1}$ in $\lceil \log_2(N+1) \rceil$ Newton's iteration steps,

$$X_0 = I, \quad X_{j+1} = X_j(2I - BX_j), \quad j = 0, 1, \cdots, k-1. \tag{9.8}$$

This follows since $X_0 = B^{-1} \bmod \lambda$, $I - BX_{j+1} = (I - BX_j)^2 \bmod \lambda^{2^{j+1}}$, so that $X_k = B^{-1} \bmod \lambda^{2^k}$. Moreover, since $X_k^{-1} = B \bmod \lambda^{2^k}$ is a Toeplitz-like matrix, we may simplify the steps (9.8) and arrive at trace $(B^{-1}) \bmod \lambda^{N+1}$ and at the vector $B^{-1}\mathbf{b} \bmod \lambda^{N+1}$ (for any given vector $\mathbf{b}$) for the parallel cost $O_A(\log^2 N, N^2/\log N)$ (see the details in [P90a] or [BP,a]).

To decrease the precision of this computation, we may perform it modulo a prime $p > N$, output $A^{-1} \bmod p$ and then shift to Algorithm 4.2. We immediately simplify every iteration step of this algorithm since the initial Toeplitz-like structure of $A^{-1} \bmod p$ is preserved.

Remark 9.1. (*Computing* gcd *and Padé approximation.*) The (m, n) Padé approximation to an analytic function $a(x) = \sum_{i=0}^{\infty} a_i x^i$ is given by the pair $(u(x), v(x))$ of two polynomials $u(x)$ and $v(x)$, $\deg u(x) \le m$, $\deg v(x) \le n$, such that $v(x)a(x) - u(x) = 0 \bmod x^N$, $N = m + n + 1$. The evaluation of $u(x)$ and $v(x)$ can be reduced for the cost $O(\log^2 N, N^2/\log N)$ to the solution of a Toeplitz linear system of size $n \times n$ ([G], [BGY]). The evaluation of the greatest common divisor (gcd) of two polynomials $u(x)$ and $v(x)$, $v(0) \neq 0$, $\deg u(x) = m$, $\deg v(x) = n$, can be reduced to computing first the polynomial $a(x) = u(x)/v(x) \bmod x^N$ and then its $(m-g, n-g)$ Padé approximants $u_g(x), v_g(x)$ for the maximum g for which $a(x) = u_g(x)/v_g(x)$. This enables us to extend the complexity bound $O_A(\log^2 N, N^2/\log N)$ to the evaluation of the gcd of two polynomials of degrees at most N (see the details in [P90a], [BP,a], and see [BG,a], [BP,a] for an alternative algorithm having the same parallel complexity).

Finally, any algorithm for the evaluation of the inverses or determinants of Toeplitz-like matrices can be easily extended to perform similar computations in the case of structured matrices of Hankel, Hilbert, and Vandermonde types (see [Pan89]).

Indeed, the Hankel matrices can be turned into Toeplitz matrices simply by row-or-column permutations. To see some further correlations among various structured matrices, observe that V^TV is a Hankel matrix, $(V^TV)_{h,k} = \sum_i v_i^{h+k}$, if V is a Vandermonde matrix, $(V)_{i,k} = v_i^k$; this gives us $(V^T)^{-1} = V(V^TV)^{-1}$ by means of inverting a Hankel matrix. Similarly, $V^T(\mathbf{v}^{-1})V(\mathbf{v})$ is a Toeplitz matrix. These are just simple examples of

more general correlations represented in Table 9.1. According to these correlations, for any given matrix A of the class HT (that is, of the Toeplitz-like or Hankel-like matrices), V (of Vandermonde-like matrices) or H (of Hilbert-like matrices), we may immediately define a Toeplitz, Hankel, Vandermonde, or generalized Hilbert matrix B such that AB belongs to one of the classes HT, V, or H, shown in the corresponding line of Table 9.1.

TABLE 9.1

A	B	AB	F-rank r of AB
HT	V	V	$r \le r_1 + r_2 + 1$
V	V	H	$r \le r_1 + r_2 + 1$
V	V	HT	$r \le r_1 + r_2$
H	V	V	$r \le r_1 + r_2$
HT	HT	HT	$r \le r_1 + r_2 + 1$
H	H	H	$r \le r_1 + r_2$

Furthermore, given the generators of lengths r_1 for $F_1(A)$, r_2 for $F_2(A)$, a generator of length r for $F(AB)$ can be immediately computed, for $r \le r_1 + r_2 + 1$. Here the operators F_1, F_2, and F are associated with the respective classes HT, V, and H (see the details in [Pan89] and compare (9.1)–(9.5)). This fact can be proven similarly to the above bound on the length of the generator of $F(AB)$ for the displacement operators F, and it reduces the inversion problem for the classes of Hankel-like, Toeplitz-like, Vandermonde-like, and Hilbert-like matrices to each other. In particular, all such problems can be reduced to the inversion of Toeplitz-like matrices. We have a similar reduction for computing the determinants, since $A^{-1} = B(AB)^{-1}$, $\det A = \det(AB)/\det B$ for nonsingular matrices A and B. For instance, this way we reduce the evaluation of a least-squares solution to a linear system with a transposed Vandermonde or a generalized Hilbert matrix A of full rank and of size $m \times n$ to a Toeplitz-like linear system with the coefficient matrix $A^T A$, and thus solve the original problem for the cost $O_A(N \log^2 N) = O_A(N \log^2 N, 1)$ or $O_A(\log^2 N, N^2/\log N)$, $N = m + n$.

10. Five versions of the polynomial division problem. To demonstrate some typical techniques of the design and analysis of algorithms for polynomial computations and their correlation to computations with structured matrices, we will next recall the fundamental problem of polynomial division with a remainder. We will state it in five equivalent versions, which will demonstrate its correlation to matrix and power series computations. In the next sections we will review some solution algorithms for this problem:

Version 10.1. Given the coefficients of two polynomials $s(x) = \sum_{i=0}^{m} s_i x^i$, $t(x) = \sum_{i=0}^{n} t_i x^i$, where $s_m t_n \neq 0$, find the coefficients of the quotient $q(x) = \sum_{i=0}^{m-n} q_i x^i$ and of the remainder $r(x) = \sum_{i=0}^{n-1} r_i x^i$ of the division of $s(x)$ by $t(x)$ such that

$$s(x) = t(x)q(x) + r(x), \qquad \deg r(x) < n. \tag{10.1}$$

If $q(x)$ is available, then $r(x)$ can be immediately obtained for the price of multiplication of $t(x)$ by $q(x)$ and subtraction of the result from $s(x)$. If $r(x)$ is available, then we may arrive at $q(x)$ by means of the evaluation of $q(x) = (s(x) - r(x))/t(x)$ at all the Kth roots of 1, ω^i, $i = 0, 1, \cdots, K-1$, $K = m - n + 1$, and subsequent interpolation. This will give us $q(x)$ for the cost of the forward and inverse FFTs at K points and of K scalar divisions.

Remark 10.1. If we perform the evaluation and interpolation at the points $N\omega^i$ for a large positive N, rather than at the points ω^i, then the latter approach can be extended

to approximate to $q(x)$ even when $r(x)$ does not vanish. Indeed, $s(x)/t(x) = q(x) + (r(x)/t(x))$ and $r(x)/t(x) \to 0$ as $x \to \infty$ since $\deg t(x) > \deg r(x)$; this approach saves arithmetic operations but requires a higher precision computation [PLS].

Version 10.2. Compute $q_0, \cdots, q_{m-n}, r_0, \cdots, r_{n-1}$ such that

$$\begin{bmatrix} s_m \\ s_{m-1} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ s_n \\ s_{n-1} \\ \cdot \\ \cdot \\ \cdot \\ s_0 \end{bmatrix} = \begin{bmatrix} t_n & 0 & & \\ t_{n-1} & \cdot & & \\ \cdot & \cdot\cdot & & \\ \cdot & \cdot\cdot\cdot & & \\ \cdot & \cdot\cdot\cdot\cdot & & \\ \cdot & \cdot\cdot\cdot\cdot\cdot & & \\ \cdot & \cdot\cdot\cdot\cdot\cdot & t_n & \\ t_0 & \cdot\cdot\cdot\cdot\cdot & t_{n-1} & \\ & \cdot\cdot\cdot\cdot\cdot & \cdot & \\ & \cdot\cdot\cdot & \cdot & \\ & \cdot & \cdot & \\ 0 & & t_0 & \end{bmatrix} \begin{bmatrix} q_{m-n} \\ q_{m-n-1} \\ \cdot \\ \cdot \\ \cdot \\ q_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ r_{n-1} \\ \cdot \\ \cdot \\ \cdot \\ r_0 \end{bmatrix}. \tag{10.2}$$

It is easy to observe the equivalence of Versions 10.1 and 10.2 to each other. Furthermore, it is sufficient to solve in $q_{m-n}, \cdots, q_0$ the first $K = m - n + 1$ linear equations of (10.2); then (10.1) immediately defines $r_0, \cdots, r_{n-1}$. Actually, we have already observed these equivalence relations when we analyzed (10.1). Indeed, the first K equations of (10.2) form a triangular Toeplitz system, so that the equivalent Versions 10.1 and 10.2 are reduced to solving such a system.

Hereafter, we let

$$S(z) = z^m s(1/z) = \sum_{i=0}^{m} s_{m-i} z^i, T(z) = z^n t(1/z) = \sum_{i=0}^{n} t_{n-i} z^i,$$
$$V(z) = T^{-1}(z) \bmod z^K = \sum_{i=0}^{K-1} v_i z^{K-1-i} \tag{10.3}$$

and arrive at two new versions of the problem.

Version 10.3. Invert a lower triangular Toeplitz matrix T formed by the first K rows of the matrix of (10.2).

Version 10.4. Compute $V(z) = T^{-1}(z) \bmod z^K$.

When the problem in its Versions 10.3 or 10.4 has been solved, the coefficient vector $\mathbf{q} = [q_{m-n}, \cdots, q_0]^T$ can be computed as the product $T^{-1}\mathbf{s}$, $\mathbf{s} = [s_m, \cdots, s_n]^T$, or as the leading coefficients of the polynomial product $V(z) \sum_{i=n}^{m} s_i z^{i-n}$, respectively. The latter product can be computed by applying FFTs.

If we only need to compute the coefficients of $q(x)$, we may truncate the polynomials $s(x)$ and $t(x)$ to the $K = m - n + 1$ leading terms, for this will not change the output (of course, we do not truncate $t(x)$ at all if $K > n$). Similarly, we may truncate each of $s(x)$ and $t(x)$ to the h leading terms if we use the following version of the problem.

Version 10.5. Compute the h leading coefficients of the formal power series $\delta(x) = s(x)/t(x)$, where $s(x)$ and $t(x)$ are formal power series in $1/x$ of the form

$$s(x) = \sum_{i=0}^{\infty} s_{m-i}x^{m-i}, \quad t(x) = \sum_{i=0}^{\infty} t_{n-i}x^{n-i}, \quad m \geq n,$$

and h is a fixed positive integer.

In particular, $s(x)$ and $t(x)$ are polynomials if $s_g = t_g = 0$ for all negative g. We may replace x by $1/z$, multiply $s(1/z)$ by z^m and $t(1/z)$ by z^n, and arrive at the equivalent problem of the division of the formal power series in z, that is, of $S(z) = z^m s(1/z) = \sum_{i=0}^{\infty} s_{m-i}z^i$ by $T(z) = z^n t(1/z) = \sum_{i=0}^{\infty} t_{n-i}z^i$ (compare (10.3)).

The output coefficients are, of course, invariant in the multiplication of $s(x)$ by the monomial x^a and of $t(x)$ by the monomial x^b for any a and b. In particular, multiply $s(x)$ by x^{2h-m} and $t(x)$ by x^{h-n}, truncate the two resulting power series to the first $h+1$ terms (which turns them into two polynomials $s^*(x)$ and $t^*(x)$ of degrees $m^* = 2h$ and $n^* = h$, respectively), and observe that this reduces Version 10.5 of the problem to the equivalent Versions 10.1–4, because the coefficients of the quotient of the division (with a remainder) of $s^*(x)$ by $t^*(x)$ equal the $h+1$ leading coefficients of $\delta(x)$. Thus, any solution of the problem in its Versions 10.1–4 can be immediately extended to the evaluation of the coefficients of the formal power series equal to the quotient of two given polynomials or of two given power series.

The same problem can also be equivalently represented by the infinite triangular Toeplitz system of linear equations, which generalized (10.2) in that the remainder-vector is removed from (10.2) and the two other vectors and the matrix are infinitely continued downward (the matrix is also infinitely continued rightward). For a given h, we may compute the values $q_{m-n}, q_{m-n-1}, \cdots, q_{m-n-h+1}$ that satisfy this infinite system truncated to its first h equations, which amount to Version 10.2 if the notation has been properly adjusted.

11. Algorithms for polynomial division and their arithmetic computational cost. Matrix versions of the algorithms. In this section we will first describe five effective algorithms for polynomial division, whose complexity is shown in Table 11.1. The algorithms exploit FFT, evaluation-interpolation techniques, Newton's iteration, and various identical representations of polynomials. Later in this section, we will reinterpret the latter identities as matrix identities and will arrive at the equivalent matrix versions of the same algorithms applied to the inversion of triangular Toeplitz matrices. Similarly, we will rewrite Newton's iteration for polynomial division as a divide-and-conquer inversion of a triangular Toeplitz matrix. Finally, we will briefly review some results on approximate polynomial division and on the application of binary segmentation to the division and to computing the gcd.

(i) Algorithm for "synthetic division" is the classical polynomial division algorithm (see [Kn]); it costs $O_A(nK)$ or $O_A(K, n)$.

(ii) The Sieveking–Kung algorithm evaluates, for the parallel computational cost of $O_A(\log^2 K, K/\log K)$, the coefficients of the polynomial $V(z)$ of (10.3). The algorithm truncates the power series $w = w(z)$ computed by Newton's method that is applied to the equation $f(w, z) = 1 - (T(z)w)^{-1} = 0$. In this case, the Newton iteration takes the form of the recurrence

$$(11.1) \quad w_{j+1}(z) = w_j(z) + w_j(z)(1 - T(z)w_j(z)) \bmod z^{2J}, \quad J = 2^j, \quad j = 0, 1, \cdots,$$

where $w_j(z)$, $j = 0, 1, \cdots$, is a sequence of polynomials in z, $w_0(z) = w_0 = 1/t_n$ (compare (7.4)). Note that the cost bound turns into $O_A(\log K \log(K/k), K/\log(K/k))$ if $w(z) \bmod z^k$ is known from the start.

(iii) Reif and Tate's algorithm relies on the following identity:

$$V_{ch}(z) = V_h(z) \sum_{j=0}^{c-1} (1 - T_{ch}(z)V_h(z))^j \bmod z^{ch}, \tag{11.2}$$

where c and h are natural numbers,

$$\begin{aligned} T_h(z) &= T(z) \bmod z^k, \\ V_h(z) &= V(z) \bmod z^h = T^{-1}(z) \bmod z^h. \end{aligned} \tag{11.3}$$

To prove (11.2), recall the identity $(1 - w)\sum_{j=0}^{c-1} w^j = 1 - w^c$, substitute $w = 1 - T_{ch}(z)V_h(z)$, and obtain that

$$T_{ch}(z)V_h(z) \sum_{j=0}^{c-1} (1 - T_{ch}(z)V_h(z))^j = 1 - (1 - T_{ch}(z)V_h(z))^{ch} = 1 \bmod z^{ch}, \tag{11.4}$$

where the last equality holds since $1 - T_{ch}(z)V_h(z) \bmod z^h = 1 - T_h(z)V_h(z) \bmod z^h = 0$ (due to (11.3)). Multiply the identity (11.4) by $V_{ch}(z)$, substitute $V_{ch}(z)T_{ch}(z) = 1 \bmod z^{ch}$, and arrive at (11.2).

Due to the relation (11.2), given $V_h(z)$ we may compute $V_{ch}(z)$ for the cost $O_A(\log(ch), c^2h)$. The polynomial on the right-hand side of (11.2) has degree at most $c^2h - 2c - h + 2 < c^2h$ (before the reduction $\bmod z^{ch}$), so that the evaluation-interpolation technique can be applied to compute the coefficients of $V_{ch}(z)$ by means of FFT at c^2h points.

The coefficients of the polynomial $V_K(z)$ can be computed by means of recursive application of (11.2). Assuming that $K = 2^{\beta-1}$ for an integer β, we define $f_i = \lceil \beta(1 - 1/2^{i-1}) \rceil$ and recursively apply (11.2) for $c = c_i$, $h = h_i$, $h_i = 2^{f_i}$, $c_i = h_{i+1}/h_i = 2^{f_{i+1} - f_i}$, $i = 1, \cdots, \lceil \log \beta \rceil + 1$. For a given $V_{h_i}(z)$, we compute $V_{h_{i+1}}(z)$ for the cost $O_A(f_{i+1}, 2^{2f_{i+1} - f_i}) = O_A(\log K, K)$ this way.

Recursively repeating this computation for $i = 1, 2, \cdots, \lceil \log \beta \rceil + 1$, $\beta - 1 = \log K$, we arrive at the coefficients of $V_K(z)$ for the cost $O_A(\log K \log\log K, K)$.

It is possible to reduce the number of processors to $O(K/\log\log K)$ by splitting the algorithm into two stages as follows.

Stage 1. Set $\alpha = \lfloor \log(K/\log K) \rfloor$ and compute $T_{2^\alpha}(z)$ by applying the above scheme, for the cost $O_A(\log K \log\log K, K/\log K)$.

Stage 2. Given $T_{2^\alpha}(z)$, compute $T_K(z)$ for the cost $O_A(\log K \log\log K, K/\log\log K)$ by applying the Sieveking–Kung algorithm.

The arithmetic computational cost of these three methods is summarized in Table 11.1, together with the cost of the algorithms of [B], [Sc82], [Geor] and [BP91], [BP90a], shown at the end of this section.

Next, we will show the equivalent matrix versions of the three algorithms described above. We will do this by presenting the basis matrix identities for the matrix versions of the polynomial division algorithms, which correspond to the basis polynomial identities for the polynomial division versions of the same algorithms. We will leave this to the reader to verify that the resulting algorithms for the triangular Toeplitz matrix inversion

TABLE 11.1
Arithmetic cost of polynomial division algorithms.

Algorithm	Arithmetic operations	Parallel steps	Processors
Classical	$O(K \min\{K,n\})$	$O(K)$	$\min\{K,n\}$
Sieveking–Kung	$O(K \log K)$	$O(\log^2 K)$	$K/\log K$
Reif and Tate [RT]	$O(K \log K)$	$O(\log K \log\log K)$	$K/\log\log K$
Georgiev [Geor]	$O(K \log^3 K)$	$O(\log K)$	$K \log^2 K$
Bini-Pan [BP90] [BP90a]	$O(K \log K \lceil 2^{-h} \log^{(h)} K \rceil)$	$O(h \log K)$	$(K/h)\lceil 2^{-h} \log^{(h)} K \rceil$
	$O(K \log K)$	$O(\log K \log^* K)$	$K/\log^* K$

evaluate the same intermediate and output values as their counterparts for computing the reciprocal of a polynomial modulo x^K.

(i) It is immediately verified that the classical polynomial division algorithm (for computing $q(x)$) can be rewritten as the back substitution stage algorithm of Gaussian elimination for the subsystem of the K first linear equations of the system (10.2).

(ii) The Sieveking–Kung algorithm is equivalent to the divide-and-conquer matrix inversion algorithm due to [BM, p. 146] and [Laf] and applied to the evaluation of the triangular Toeplitz matrix T^{-1}. Namely, let T_h be the $2^h \times 2^h$ triangular Toeplitz matrix (where $h = \lceil \log K \rceil$), whose first column coincides with the first column of T on the first K entries and is filled with zeros elsewhere. Then T^{-1} is the $K \times K$ leading submatrix of T_h^{-1}, and T_h^{-1} is computed by recursively inverting all its leading submatrices of sizes $2^i \times 2^i$ for $i = 0, 1, \cdots, h$, according to the following formulae:

$$T_{i+1} = \begin{bmatrix} T_i & O \\ W_i & T_i \end{bmatrix}, \qquad T_{i+1}^{-1} = \begin{bmatrix} T_i^{-1} & O \\ -T_i^{-1} W_i T_i^{-1} & T_i^{-1} \end{bmatrix}.$$

Here, T_i and T_i^{-1} are $2^i \times 2^i$ leading triangular Toeplitz submatrices of T_h and T_h^{-1}, respectively, so T_{i+1}^{-1} is defined by its first column:

$$\mathbf{v}_{i+1} = [\mathbf{v}_i^T, (-T_i^{-1} W_i \mathbf{v}_i)^T]^T, \tag{11.5}$$

where $\mathbf{v}_i$ denotes the first column of T_i^{-1}.

Comparing (11.1) with (11.5), we note that the evaluation of the leading 2^i coefficients of the polynomial product $u^{(i)}(z) = -T(z)w^{(i)}(z)$ is equivalent to the evaluation of the matrix-by-vector product $\mathbf{g}_i = -W_i \mathbf{v}_i$, that all other coefficients of $1 + u^{(i)}(z)$ are zeros and that the evaluation of the coefficients of the polynomial product $w^{(i)}(z)(1 + u^{(i)}(z))$ is equivalent to the evaluation of the matrix-by-vector product $T_i^{-1}\mathbf{g}_i$.

(iii) The matrix version of Reif and Tate's algorithm relies on the following matrix identity:

$$T_{ch}^{-1} = \begin{bmatrix} T_h & & O \\ & T_h & \\ & \vdots & \\ O & & T_h \end{bmatrix}^{-1} \sum_{j=0}^{c-1} \left(I - T_{ch} \begin{bmatrix} T_h & & O \\ & T_h & \\ & \vdots & \\ O & & T_h \end{bmatrix}^{-1} \right)^j,$$

where T_i is the $i \times i$ leading principal submatrix of the triangular Toeplitz matrix T.

Remark 11.1. If the integers m and n are such that $n < K$, then the matrix T is a band matrix with bandwidth $n+1$. In this case the system $T\mathbf{s} = \mathbf{q}$ (where $\mathbf{s} = [s_m, \cdots, s_n]^T$ and $\mathbf{q} = [q_{K-1}, \cdots, q_0]^T$) can be solved in a different way, that is, by considering T as a bidiagonal block matrix and applying block back substitution. This way the sequential cost is reduced to $O_A(K \log n)$ if $n < K$, that is, to $O_A(K \log \min\{n, K\})$ in the general case.

In the remainder of this section, we will recall another (very recent) approach to polynomial division. Let for a fixed natural $k \le K$, $d = \lceil \log(K/k) \rceil$, $D = 2K\lceil \log(K/k) + 1 \rceil - 2\lfloor K/k \rfloor + 2$. Now, suppose that the coefficients of $w_0(z) = T^{-1}(z) \bmod z^k$ are known and point-wise evaluate the following polynomials at all the Dth roots of 1.

(a) $T_{i-1}(z) = T(z) \bmod z^{k2^i}$, concurrently for $i = 1, \cdots, d$, by means of d applications of DFT;

(b) $w_i(z) = w_{i-1}(z)(2 - T_{i-1}(z)w_{i-1}(z))$, recursively for $i = 1, \cdots, d$.

Finally, apply inverse DFT to compute and output the first K coefficients of $w_d(z)$, which $w_d(z)$ shares with $T^{-1}(z)$. Indeed, observe that

$$w_i(z) = T^{-1}(z) \bmod z^{k2^i},$$
$$\deg w_i(z) \le (i+1)k2^i - 2^{i+1} + 1,$$

for $i = 1, \cdots, d$, which for $i = d$ turns into the relations $\deg w_d(z) < D$, $w_d(z) = T^{-1}(z) \bmod z^K$, and the correctness of the above algorithm follows. Its computational cost bound is given by $O_A(\log K, K \log^2(K/k))$; for $k = 1$, this is $O_A(\log K, K \log^2 K)$, the estimate of [Geor].

Following [BP90a], we will improve this bound to

(11.6)

$$c(K,h) = O_A(h \log K, (K/h)(1 + 2^{-h} \log^{(h)} K)) \quad \text{for any fixed } h, h = 1, \cdots \log^* K.$$

Fix h and recursively apply the same algorithm, replacing K by K_j and k by k_j, where $k_0 = 1$, $k_{j+1} = K_j$, $j = 0, 1, \cdots$. Denote $K(h) = K2^{-h}$, $s_j = K(h)/k_j$, and set $K_j = \lceil K(h)/\log^2 s_j \rceil$, $j = 0, 1, \cdots, J-1$, where we will specify J later on. Then the application of the above algorithm for each j costs $O_A(\log K(h), K(h))$, and the overall cost is given by $O_A(J \log K(h), K(h))$ (for all j).

In j applications, we arrive at $T^{-1}(z) \bmod z^{K_j}$, where $K_j = K(h)/s_{j+1}$, and we note that $s_0 = K(h)$, $s_1 = \log^2 K(h)$, $s_2 = \log^2 \log^2 K(h), \cdots, s_J = O(\log^{(h)} K(h))$ for some $J \le 2h$. (Assume for convenience that $K(h)/\log^2 s_j$ are integers for all j.)

One more application of the same algorithm, this time, for k replaced by K_J and for K replaced by $K(h)$, gives us $T^{-1}(z) \bmod K(h)$ for the additional cost $O_A(\log K(h), K(h)(\log^{(h+1)} K(h))^2)$, which can be replaced by the weaker bound, $O_A(h \log K(h), (K(h)/h) \log^{(h)} K(h))$. At this stage, the overall cost can be bounded by $O_A(h \log K(h),$

$K(h)(1 + (1/h)\log^{(h)} K(h)))$, or even by $O_A(h \log K, K2^{-h}(1 + \log^{(h)} K))$, which is consistent with (11.6).

Finally, we compute the remaining (if any) coefficients of $T^{-1}(z) \bmod z^K$ by applying the Sieveking–Kung's algorithm for the additional cost $O_A(h \log K, K/h)$, which is still consistent with (11.6). Observe that (11.6) turns into $O_A(\log K \log^* K, K/\log^* K)$ for $h = \log^* K$, since $\log^{(h)} K \leq 1$ for $h = \log^* K$.

12. Approximate polynomial division. Binary segmentation for polynomial division and polynomial GCD. The cited algorithms reduce the parallel cost of polynomial division almost to $O_A(\log N, N)$, the cost of FFT and polynomial multiplication, but the algorithm of [B] reached the latter cost bound for approximate evaluation of $q(x)$ and $r(x)$ of (10.1) with an arbitrary high precision, although this algorithm may generally lead to numerical stability problems. The algorithm exploits, in particular, the representation of a lower triangular Toeplitz matrix $T = [t_{n+j-i}]$, $t_s = 0$ unless $0 \leq s \leq n$, as $T = \sum_{i=0}^{n} t_{n-i} Z^i$, where Z is the displacement matrix defined in §9. This representation also enables us to obtain the following very useful a priori bound on the coefficients of $q(x)$ in terms of the coefficients of $s(x)$ and $t(x)$:

$$\sum_{i=0}^{K-1} |q_i| \leq \left(\frac{1+t}{t_n}\right)^{K-1} \sum_{i=n}^{m} \frac{|s_i|}{t_n}, \tag{12.1}$$

where $t = \max_{0 \leq i \leq m-n} |t_{n-i}|$, $t_g = 0$ if $g < 0$ (see [BP]).

It is easy to observe that binary segmentation reduces the division with a remainder of two polynomials with integer coefficients, $s(x)$ by $t(x)$, to the division of their values at $x = 2^h$, for a sufficiently large integer h. Due to the bound (12.1), we may choose an appropriate h, for which we arrive (in [BP]) at the record parallel asymptotic estimates for the Boolean complexity of polynomial division. Similarly, binary segmentation has led us in [BP] (see also [Sc85], [BP,a]) to the record sequential Boolean complexity estimates for the sequential evaluation of the gcd of two polynomials with integer coefficients. In this case, we also have a priori upper bounds on the magnitude of the output values.

In both cases of the division and computing the gcd, the binary segmentation generally involves operations with long integers, which complicates the practical implementation of the resulting algorithms.

Remark 12.1. The approximation algorithm of [B] has been originally obtained for the triangular Toeplitz matrix inversion, but also has two equivalent versions: in the form of the division of formal power series [Sc82] and of the division of polynomials with a remainder (see Remark 10.1).

Finally, it is interesting to point out that the estimate (12.1) has been deduced in [BP] by means of the study of the inversion of a triangular Toeplitz matrix, whereas both of the above applications (by using Remark 10.1 and binary segmentation) rely on the transition to polynomial division, in particular, on (10.1).

Acknowledgments. The author thanks the editor, Professor Paul Davis, and the referees for valuable comments and suggestions, and Joan Bentley for her superb assistance in typing the manuscript.

REFERENCES

[Abe] O. ABERTH, *Iteration methods for finding all zeros of a polynomial simultaneously*, Math. Comp., 27 (1973), pp. 339-344.

[AHU] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1976.

[ASU] A. V. AHO, K. STEIGLITZ, AND J. D. ULLMAN, *Evaluating polynomials at fixed set of points*, SIAM J. Comput., 4 (1975), pp. 533–539.

[AR] B. ALPERT AND V. ROKHLIN, *A fast algorithm for the evaluation of Lagrange expansions*, SIAM J. Sci. Statist. Comput., 12 (1990), pp. 158–179.

[AGR] J. AMBROSIANO, L. GREENGARD, AND V. ROKHLIN, *The fast multipole method for gridless particle simulation*, Comput. Phys. Comm., 48 (1988), pp. 115–125.

[Ag,b] G. S. AMMAR AND W. G. GRAGG, *Superfast solution of real positive definite Toeplitz systems*, SIAM J. Matrix Anal., 9 (1988), pp. 61–76.

[BD81] R. BANK AND T. DUPONT, *An optimal order process for solving finite element equations*, Math. Comp., 36 (1981), pp. 35–51.

[BS] W. BAUR AND V. STRASSEN, *On the complexity of partial derivatives*, Theoret. Comput. Sci., 22 (1983), pp. 317–330.

[Be] A. BEN-ISRAEL, *A note on iterative method for generalized inversion of matrices*, Math. Comput., 20 (1966), pp. 439–440.

[BFKT] M. BEN-OR, E. FEIG, D. KOZEN, AND P. TIWARI, *A fast parallel algorithm for determining all roots of a polynomial with real roots*, SIAM J. Comput., 17 (1989), pp. 1081–92. Short version in Proc. 18th Ann. Symp. on Theory of Comput. (1986), pp. 340–349.

[B] D. BINI, *Parallel solution of certain Toeplitz linear systems*, SIAM J. Comput., 13 (1984), pp. 268–276; T.R.B82-04, I.E.I. of C.N.R., Pisa, Italy (1982).

[BdB] D. BINI AND F. DI BENEDETTO, *A new preconditioner for the parallel solution of positive definite Toeplitz systems*, Proc. 2nd Ann. ACM Sympos. on Parallel Algorithms and Architecture (SPAA 90), (1990), pp. 220–223; SIAM J. Sci. Statist. Comput., to appear.

[BG,a] D. BINI AND L. GEMIGNANI, *The fast parallel computation of the polynomial remainder sequence via Bezout and Hankel matrices*, 1990, manuscript.

[BL80] D. BINI AND G. LOTTI, *Stability of fast algorithms for matrix multiplication*, Numer. Math., 36 (1980), pp. 63–72.

[BP] D. BINI AND V. PAN, *Polynomial division and its computational complexity*, J. Complexity, 2 (1986), pp. 179–203.

[BP88] ———, *Efficient algorithms for the evaluation of the eigenvalues of (block) banded Toeplitz matrices*, Math. Comp., 50 (1988), pp. 431–448.

[BP,90] ———, *Parallel polynomial computations by recursive processes*, Proc. ACM SIGSAM Internat. Sympos. on Symb. and Algebraic Comp., (1990), p. 299.

[BP,a] ———, *Numerical and Algebraic Computations with Matrices and Polynomials*, Birkhauser, Boston, MA, 1992.

[BP91] ———, *Parallel Complexity of Tridiagonal Symmetric Eigenvalue Problem*, Proc. 2nd Ann. ACM-SIAM Sympos. on Discr. Algor., (1991), pp. 384–393.

[BP,b] ———, *Practical improvement of the divide-and-conquer eigenvalue algorithms*, Computing (1992), to appear.

[BP90a] ———, *Improved parallel polynomial division*, Tech. Report CUCS 026-90, Computer Science Dept., Columbia University, 1990; SIAM J. Comput., to appear.

[BPc] ———, *On the evaluation of the eigenvalues of a banded Toeplitz block matrix*, Tech. Rep. CUCS-024-90, Computer Science Dept., Columbia University, New York, 1990; J. Complexity, 7(1991), pp. 408–424.

[BA] R. R. BITMEAD AND B. D. O. ANDERSON, *Asymptotically fast solution of Toeplitz and related systems of linear equations*, Linear Algebra Appl., 34 (1980), pp. 103–116.

[BM] A. BORODIN AND I. MUNRO, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, 1975.

[BGH] A. BORODIN, J. VON ZUR GATHEN, AND J. HOPCROFT, *Fast Parallel Matrix and GCD Computation*, Inform. and Control, 52 (1982), pp. 241–256.

[Brent70] R. P. BRENT, *Error analysis of algorithms for matrix multiplication and triangular decompositions using Winograd's identity*, Numer. Math., 16 (1970), pp. 145–156.

[Bre] ———, *The parallel evaluation of general arithmetic expressions*, J. ACM, 21 (1974), pp. 201–206.

[BGY] R. P. BRENT, F. G. GUSTAVSON, AND D. Y. Y. YUN, *Fast solution of Toeplitz systems of equations and computation of Padé approximations*, J. Algorithms, 1 (1980), pp. 259–295.

[Bun] J. R. BUNCH, *Stability of methods for solving Toeplitz systems of equations*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 349–364.

[CK] D. G. CANTOR AND E. KALTOFEN, *Fast multiplication of polynomials with coefficients from an arbitrary ring*, Tech. Report 87-35, Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY, 1987.

[CGR] J. CARIER, L. GREENGARD, AND V. ROKHLIN, *A fast adaptive multipole algorithm for particle simulation*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 669–686.

[ChS] R. H. CHAN AND G. STRANG, *Toeplitz equations by conjugate gradients with circulant preconditioner*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 104–119.

[Ch] T. F. CHAN, *An optimal circulant preconditioner for Toeplitz systems*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 766–771.

[CGGW] B. W. CHAR, K. O. GEDDES, G. H. GONNET, AND S. M. WATT, *First leaves: a tutorial introduction to MAPLE*, in MAPLE User's Guide, WATCOM Publications, Waterloo, Ontario, 1985.

[CKL-A] J. CHUN, T. KAILATH, AND H. LEV-ARI, *Fast parallel algorithm for QR-factorization of structured matrices*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 899–913.

[CW] D. COPPERSMITH AND S. WINOGRAD, *Matrix multiplication via arithmetic progressions*, Proc. 19th Ann. ACM Sympos. on Theory of Comput., (1987), pp. 1–6; J. Symbolic Comput., 9 (1990), pp. 251–280.

[CB] G. CYBENKO AND M. BERRY, *Hyperbolic Householder algorithms for factoring structured matrices*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 499–520.

[Datta] K. DATTA, *Parallel Complexity and Computations of Cholesky's Decomposition and QR Factorization*, Internat. J. Comput. Math., 18 (1985), pp. 67–82.

[Da74] P. DAVIS, *Circulant Matrices*, Wiley, New York, 1974.

[dH] F. R. DEHOOG, *On the solution of Toeplitz systems*, Linear Algebra Appl., 88 (1987), pp. 123–138.

[Dur] E. DURAND, *Solutions numériques des équations algébriques, Tome* I: *Equations du type* $F(x) = 0$: *Racines d'un polynôme*, Masson, Paris, 1960.

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

[FP] M. J. FISCHER AND M. S. PATERSON, *String matching and other products*, SIAM-AMS Proc., 7 (1974), pp. 113–125.

[FMc] P. O. FREDERICKSON AND O. A. MCBRYAN, *Parallel superconvergent multigrid*, in Multigrid Methods: Theory, Applications and Supercomputing, Lecture Notes in Pure and Appl. Math., 100, S. McCormick, ed., M. Dekker, New York, pp. 195–210.

[FMc87a] P. O. FREDERICKSON AND O. A. MCBRYAN, *Superconvergent multigrid methods*, Cornell Theory Center, Ithaca, NY, 1987, preprint.

[GP85] Z. GALIL AND V. PAN, *Improving processor bounds for algebraic and combinatorial problems in RNC*, Proc. 26th Ann. IEEE Sympos. on Foundation of Computer Sci., Portland, OR, 1985, pp. 490–495.

[GPa] ———, *Improved processor bounds for combinatorial problems in RNC*, Combinatorica, 8 (1988), pp. 189–200.

[Gast60] N. GASTINEL, *Inversion d'une matrice generalisant la matrice de Hilbert*, Chiffres, 3 (1960), pp. 149–152.

[Gat84] J. VON ZUR GATHEN, *Parallel algorithms for algebraic problems*, SIAM J. Comput., 13 (1984), pp. 802–824.

[GS] W. GENTLEMAN AND G. SANDE, *Fast Fourier Transform for Fun and Profit*, Proc. Fall Joint Comput. Conf., 29 (1966), pp. 563–578.

[GeLi] J. A. GEORGE AND J. W. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Geor] R. E. GEORGIEV, *Inversion of triangular Toeplitz matrices by using the fast Fourier transform*, J. New Gener. Comput. Syst., 2 (1989), pp. 247–256.

[Ger87] A. GERASOULIS, *A fast algorithm for the multiplication of generalized Hilbert matrices with vectors*, Math. Comput., 50 (1987), pp. 179–188.

[GKKL] I. GOHBERG, T. KAILATH, I. KOLTRACHT, AND P. LANCASTER, *Linear complexity parallel algorithms for linear systems of equations with recursive structure*, Linear Algebra Appl., 88 (1987), pp. 271–315.

[GL] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1989.

[G] W. B. GRAGG, *The Padé table and its relation to certain algorithms of numerical analysis*, SIAM Rev., 14 (1972), pp. 1–62.

[Ha77] W. HACKBUSCH, *On the convergence of multi-grid iteration applied to finite element equations*, Report 77-8, Universität zu Köln, Köln, Germany, 1977.

[Hac80] ———, *Convergence of Multi-grid Iterations Applied to Difference Equations*, Math. Comp., 34 (1980), pp. 425–440.

[Hac85] ———, *Multi-Grid Methods and Applications*, Springer-Verlag, Berlin, 1985.

[HT82] W. HACKBUSCH AND U. TROTTENBERG, EDS., *Multigrid Methods, Lecture Notes in Math.*, 960, Springer-Verlag, Berlin, 1982.

[HW79] G. H. HARDY AND E. M. WRIGHT, *An Introduction to the Theory of Numbers*, Oxford University Press, Oxford, 1979.

[He] P. HENRICI, *Applied and Computational Complex Analysis*, Wiley, New York, 1974.

[H] A. S. HOUSEHOLDER, *The Numerical Treatment of a Single Nonlinear Equation*, McGraw-Hill, New York, 1970.

[IMH] O. H. IBARRA, S. MORAN, AND R. HUI, *A generalization of the fast* LUP *matrix decomposition algorithm and applications*, J. Algorithms, 3 (1982), pp. 45–56.

[KKM] T. KAILATH, S.-Y. KUNG, AND M. MORF, *Displacement ranks of matrices and linear equations*, J. Math. Anal. Appl., 68 (1979), pp. 395–407.

[KVM] T. KAILATH, A. VIERA, AND M. MORF, *Inverses of Toeplitz operators, innovations, and orthogonal polynomials*, SIAM Rev., 20 (1978), pp. 106–119.

[Ka] E. KALTOFEN, *Factorization of polynomials*, in Computer Algebra. Symbolic and Algebraic Computation, B. Buchberger, G. E. Collins, and R. Loos, eds., Springer-Verlag, Berlin, 1983, pp. 95–113.

[Ka92] E. KALTOFEN, *Polynomial Factorization* 1987–1991, Proc. Latin '92, Lecture Notes in Comput. Sci., to appear.

[KP] E. KALTOFEN AND V. PAN, *Processor efficient parallel solution of linear systems over an abstract field*, Proc. 3rd Ann. ACM Sympos. on Parallel Algorithms and Architecture (SPAA), 1991, pp. 180–191.

[K] N. K. KARMARKAR, *A new polynomial time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373–395.

[KR] R. KARP AND V. RAMACHANDRAN, *A survey of parallel algorithms for shared memory machines*, in Handbook of Theoretical Computer Science, North-Holland, Amsterdam, 1990, pp. 869–941.

[KUW] R. M. KARP, E. UPFAL, AND A. WIGDERSON, *Constructing a perfect matching is in random* NC, Proc. 17th Ann. ACM Sympos. on Theory of Computing, (1985), pp. 22–32.

[Ker] I. O. KERNER, *Ein gesamtschrittverfahren zur berechnung der nullstellen von polynomen*, Numer. Math., 8 (1966), pp. 290–294.

[Kn] D. E. KNUTH, *The Art of Computer Programming: Seminumerical Algorithms*, 2, Addison-Wesley, Reading, MA, 1981.

[LPS] J. LADERMAN, V. PAN, AND X.-H. SHA, *On practical acceleration of matrix multiplication*, Linear Algebra Appl., 162–164 (1992), pp. 557–588.

[Laf] J. C. LAFON, *Base Tensorielle des matrices de Handel (ou de Toeplitz), Applications*, Numer. Math., 23 (1975), pp. 249–361.

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

[L] S. LINNAINMAA, *Taylor expansion of the accumulated rounding error*, BIT, 16 (1976), pp. 146–160.

[LRT] R. J. LIPTON, D. ROSE, AND R. E. TARJAN, *Generalized nested-dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346–358.

[Lo] L. LOVAŚZ, *Connectivity algorithms using rubber-bands*, Proc. Sixth Conf. on Foundations of Software Technology and Theoretical Computer Science, New Delhi, India: Lecture Notes in Comput. Sci. 241, Springer-Verlag, Berlin, 1986, pp. 394–412.

[McC87] S. MCCORMICK, ed., *Multigrid Methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[McT83] S. MCCORMICK AND U. TROTTENBERG, EDS., *Multigrid methods*, Appl. Math. Comput., 13 (1983), pp. 213–474.

[Mc86] S. MCCORMICK, ED., *Proceeding of the 2nd Copper Mountain Multigrid Conference*, Appl. Math. Comput., 19 (1986), pp. 1–372.

[MC] R. T. MOENCK AND J. H. CARTER, *Approximate algorithms to derive exact solutions to systems of linear equation*, Proc. EUROSAM, Lecture Notes in Comput. Sci., 72 (1979), Springer-Verlag, Berlin, pp. 63–73.

[MVV] K. MULMULEY, U. VAZIRANI, AND V. VAZIRANI, *Matching is as easy as matrix inversion*, Combinatorica, 7 (1987), pp. 105–114.

[Mus] B. R. MUSICUS, *Levinson and fast Choleski algorithms for Toeplitz and almost Toeplitz matrices*, Internal Report, Lab. of Electronics, MIT, Cambridge, MA, 1981.

[N] C. A. NEFF, *Specified precision polynomial root isolation is in* NC, Proc. 31st Annual IEEE Symposium FOCS, (1990), pp. 152–162.

[ODR] S. T. O'DONNELL AND V. ROKHLIN, *A fast algorithm for numerical evaluation of conformal mappings*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 475–487.

[P84] V. PAN, *How to Multiply Matrices Faster*, Lecture Notes in Comput. Sci. 179, Springer-Verlag, New York, 1984.
[P87b] ———, *Complexity of parallel matrix computations*, Theoret. Comput. Sci., 54 (1987), pp. 65–85.
[P87a] ———, *Sequential and parallel complexity of approximate evaluation of polynomial zeros*, Comput. Math. Appl., 14 (1987), pp. 591–622.
[P88b] ———, *New methods for computations with Toeplitz-like matrices*, Tech. Report 88-28, Computer Science Dept., SUNY, Albany, NY, 1988.
[P89b] ———, *Parallel inversion of Toeplitz and block Toeplitz matrices*, in Operator Theory: Advances and Applications, Vol. 40, Birkhauser, Basel, 1989, pp. 359–389.
[P90c] ———, *The modified barrier function method for linear programming and its extensions*, Comput. Math. Appl., 20 (1990), pp. 1–14.
[Pan89] ———, *On computations with dense structured matrices*, Proc. ACM-SIGSAM Intern. Sympos. on Symb. Alg. Comp. (ISAAC 89), (1989), pp. 34–42; Math. Comput., 55 (1990), pp. 179–190.
[P90] ———, *Parallel least-squares solution of general and Toeplitz-like linear systems*, Proc. 2nd Ann. ACM Sympos. on Parallel Algorithms and Architecture, (1990), pp. 244–253.
[P91b] ———, *Randomized incomplete numerical factorization of a polynomial over the complex field*, Tech. Report 91-026, International Computer Science Institute, Berkeley, CA, 1991.
[P90a] ———, *Parameterization of Newton's iteration for computations with structured matrices and applications*, Tech. Report CUCS-032-90, Columbia University, Computer Science Dept., New York, 1990; Comput. Math. Appl., to appear.
[P92] ———, *On binary segmentation for matrix and vector operations*, Comp. Math. Appl. (1992), to appear.
[P91a] ———, *On decreasing the precision of matrix computations*, Tech. Report 91-11, Computer Science Dept., SUNY, Albany, NY, 1991.
[P93] ———, *Decreasing the displacement rank of a matrix*, SIAM J. Matrix Analysis, 14 (1993), to appear.
[PD] V. PAN AND J. DEMMEL, *A new algorithm for the symmetric eigenvalue problem*, preprint.
[PLS] V. PAN, E. LANDOWNE, AND A. SADIKOU, *Approximate polynomial division with a remainder by means of evaluation and interpolation*, Inform. Process. Lett., to appear.
[PR,85] V. PAN AND J. REIF, *Efficient parallel solution of linear systems*, Proc. 17th Ann. ACM Symp. on Theory of Computing, Providence, RI, 1985, pp. 143–152.
[PR,a] ———, *Fast and efficient algorithms for linear programming and for the linear least squares problem*, Comput. Math. Appl., 12A (1986), pp. 1217–1227.
[PR88] ———, *Fast and efficient parallel solution of sparse linear systems*, Tech. Report 88-18, Computer Science Dept., SUNY, Albany, NY, 1988.
[PR] ———, *Fast and efficient solution of path algebra problems*, J. Comput. System Sci., 38 (1989), pp. 494–510.
[PR91] ———, *Generalized compact multigrid and backward interval analysis*, Tech. Report 91-4, Computer Science Dept., SUNY, Albany, NY, 1991.
[PR89a] ———, *On the bit-complexity of discrete solution of* PDEs: *compact multigrid*, Comput. Math. Appl., 20 (1991), pp. 9–16.
[PR,b] ———, *Compact multigrid*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 119–127.
[PS] V. PAN AND R. SCHREIBER, *An improved Newton iteration for the generalized inverse of a matrix, with applications*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1109–1131.
[PS92] ———, *A fast preconditioned conjugate gradient Toeplitz solver*, Comput. Math. Appl., to appear.
[Pease] M. PEASE, *An adaptation of the fast Fourier transform for parallel processing*, J. of ACM, 15 (1968), pp. 252–264.
[Quinn] M. J. QUINN, *Designing efficient algorithms for parallel computers*, McGraw-Hill, New York, 1987.
[RT] J. H. REIF AND S. H. TATE, *Optimal size division circuits*, SIAM J. Comput., 19 (1990), pp. 612–624.
[R] J. RENEGAR, *On the worst-case arithmetic complexity of approximating zeros of polynomials*, J. Complexity, 3, (1987), pp. 90–113.
[RS] J. RENEGAR AND M. SHUB, *Simplified complexity analysis for Newton LP methods*, Tech. Report 807, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, 1988.
[Rok85] F. ROKHLIN, *Rapid solution of integral equations of classical potential theory*, J. Comput. Phys., 60 (1985), pp. 187–207.
[Rok] V. ROKHLIN, *A fast algorithm for the discrete Laplace transformation*, J. Complexity, 4 (1988), pp. 12–32.
[Sc82] A. SCHONHAGE, *Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients*, Proc. EUROCAM, Marseille, 1982.
[Sc] ———, *The fundamental theorem of algebra in terms of computational complexity*, Dept. of Math., University of Tübingen, Tübingen, West Germany, 1982, manuscript.
[Sc85] ———, *Quasi*-gcd *computations*, J. Complexity, 1 (1985), pp. 118–137.

[S85] S. SMALE, *On the efficiency of the algorithms of analysis*, Bull. Amer. Math. Soc., 13 (1985), pp. 87–121.
[Strang] G. STRANG, *A proposal for Toeplitz matrix calculations*, Stud. Appl. Math., 74 (1986), pp. 171–176.
[St69] V. STRASSEN, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.
[UP83] S. URSIC AND C. PATARRA, *Exact solution of systems of linear equations with iterative methods*, SIAM J. Algebraic Discrete Meth., 4 (1983), pp. 111–115.
[Wang] P. WANG, *A p-adic algorithm for univariate partial fractions*, Proc. 1981 ACM Sympos. on Symbolic and Algebraic Comp., (1981), pp. 212–217.
[Wer] W. WERNER, *On the simultaneous determination of polynomial roots*, Lecture Notes in Math. 953, Springer-Verlag, Berlin, 1982, pp. 188–202.
[Wilk] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.
[W79] S. WINOGRAD, *On the multiplicative complexity of the discrete Fourier transform*, Advances in Math., 32 (1979), pp. 83–117.
[Wol] S. WOLFRAM, MATHEMATICA—*A System for Doing Mathematics by Computer*, Addison-Wesley, Redwood City, CA, 1988.