Complexity of Computations with Matrices and Polynomials

Victor Pan

Department of Computer Science, Columbia University, New York, NY 10027

Computer Science Department

State University of New York at Albany

Albany, New York 12222

and

Mathematics and Computer Science Department

Lehman College, CUNY

Bronx, NY 10468

CUCS-031-90

Summary. We review 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.

0. Introduction.

Polynomial and matrix computations are highly important classical subjects. They have been thoroughly revised during the last decades due to the development of computer technology, whose latest notable progress was the advent of parallel computers.

The current polynomial and matrix algorithms perform far better than their classical predecessors, and one of our major objectives is to review the computational cost of such modern algorithms, together with other aspects of their design and analysis, including their various correlations to each other and to the algorithms for other algebraic and numerical computations.

In this review we will present some highlights of computations with univariate polynomials and matrices over the complex, real or rational fields; some further treatment can be found in [BP,a] (which is the most updated coverage) and, to some extent, in the earlier books [AHU], [BM] and (for polynomial computations) [Kn]. Many of the techniques and the algorithms that we review can be extended to computations over finite fields too (see [BP,a]).

Reviewing matrix computations, we will avoid overlaps with the excellent presentations in such books as [Wilk], [GL] and [GeLi] by focusing on distinct but still important issues, in particular, on **the complexity of general matrix computations**, on some very recent algorithms, on the complexity of computations with **dense structured matrices** and on their correlations to **polynomial computations**.

It is convenient and customary to characterize matrix and polynomial computations by their arithmetic complexity, and we will do this in Part II of our review, but beyond this, in Part I, we will review some techniques of computation with variable or fixed precision.

In the latter case (of the fixed precision) we recall the techniques of binary segmentation in section 1. In sections 2 and 3, we cover the very recent compact multigrid algorithm, which enables us to solve the constant coefficients linear PDEs discretized over grids and which only uses very low precision computations [with O(1) binary bits]. In section 4, we recall some powerful and universal techniques from computer algebra that bound the precision of rational computations.

In Part II, we review the complexity of matrix and polynomial computations under the RAM and PRAM arithmetic models, briefly recalled in section 5 (Part II). In section 6, we review the arithmetic complexity of polynomial computations and outline a recent algorithm that supports the record cost estimates for approximating to polynomial zeros. We review the complexity of computations with general matrices in section 7 and extend our study to computations with dense structured matrices in sections 8 and 9, indicating a way to their unification (section 9) and some correlations to polynomial computations. We review polynomial division and its correlation to matrix and power series computations in sections 10, 11 and 12.

Presenting the algorithms, we will indicate which of them may require a higher precision computation; in such a case their practical implementation for numerical computing may be a problem, of course, but they may be either implemented as computer algebra algorithms or modified for numerical computation with improved numerical stability (sometimes at the expense of some increase of their complexity).

Part I. Exploiting Fixed and Variable Precision Computations.

In this part we will show how some numerical and/or algebraic computations can be made more effective by means of either fixing their precision below the level of the 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; then we may decrease the number of arithmetic operations involved, by using binary segmentation.

Example 1.1. Interpolation via binary segmentation. Recall the interpolation problem of the recovery of an n-th degree polynomial p(x) from its values. Generally, we need the values at n+1 points, but let $p(x) = \sum_{i=0}^{n} p_i x^i$ 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 $p_0, \ldots, p_n$ can immediately be recovered as the appropriate segments of the binary value $p(2^h) = \sum_{i=0}^{n} p_i 2^{hi}$, 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 $p_0, \ldots, p_n$ are integers lying strictly between -2^g and 2^g ([Kn], p. 191). Choose an integer $h \geq g+1$ and compute $p(2^h) + \sum_{i=0}^{n} 2^{hi+h-1} = \sum_{i=0}^{n} (p_i + 2^{h-1}) 2^{hi}$; 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 $p_0, \ldots, p_n$ are *Gaussian integers*, that is, complex numbers with integer real and imaginary parts.

It is worth emphasizing that the 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 only need to have the longer output value fitted 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, **u** and **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 intervals between $-U$ and U and $-V$ and V, respectively. Then the coefficients of the polynomial $w(x)=u(x)v(x)$ lie between $-W$ and W where $W = \min\{m,n\}UV$, so that Example 1.1 suggests that we may recover the convolution of **u** and **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 $h \geq 1+\log W$.

Example 1.3, computing the inner and outer products of two vectors (section 40 of [P84]). Under the assumptions of Example 1.2, we may compute the inner product $\mathbf{u}^T\mathbf{v} = \sum_{j=0}^{n} u_j\, v_j$ (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,

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

and in this case h is an integer, $h \geq 1 + \log W$, $W = n\,U\,V$, whereas

$$q(2^h) = \Big(\sum_{i=0}^{m-1} 2^{inh}\, u_i\Big)\Big(\sum_{j=0}^{n-1} 2^{jh}\, v_j\Big) = \sum_{i,j} u_i v_j\, 2^{(in+j)h},$$

and in this case h is an integer, $h \geq 1 + \log W$, $W = U\,V$.

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 **u** and **v** have dimension 8 and have components 0, -1 and 1 and assume the computer precision of 64 binary bits. Then the straightforward evaluation of the convolution, inner product and outer product of **u** and **v** requires 64, 15 and 64 ops, respectively. With binary segmentation we only need single integer multiplication for each problem if we may use a sufficiently high precision. Even if the computer precision were 32 binary bits, the single multiplication of $u(2^h)$ by $2^{(n-1)h}v(2^{-h})$, for h=4, with chopping of the product to 32 bits and subsequent segmentation of the last 32 bits would suffice for computing the inner product. For the convolution and outer product, we would have needed two multiplications modulo 2^{32} for each, that is, of $u(2^h)$ by $v(2^h)$ and of $2^{mh}u(2^{-h})$ by $2^{nh}v(2^{-h})$ for convolution where h=4 and of $u(2^{nh/2})$ 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 $q(x) = (v(x) - p(x)) / x^{n/2}$ for the outer product where h=1.

2. Lower Precision Computation. Compact Multigrid.

Some computers, such as CM-1 (Connection Machine) and MASPAR, are capable of taking substantial advantages of lower precision computations, and next we will follow [PR89a] and will show how to solve a large class of partial differential equations (PDEs), (reduced to linear algebraic systems of equations) by using lower precision computations. To formalize the results, we will assume 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, $1 \le \alpha \le 2$.

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, but we will follow [PR89a] and will arrive at a compact representation with $O(N)$ bits, which in the case of constant coefficient linear PDEs, we will compute by using a low precision and $O(N)$ bit-operations [as long as O(1) bit-operations suffice to multiply a pair of O(1)-bit integers]. Note that this means by ten times improvement already in the case where N = 1024, which is far below the sizes of linear systems handled in the practical PDE computations. Following [PR89a], 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

$L_0, L_1, \ldots, L_k$, where each point of L_j lies at the distance $h_j = 2^{-j}$ from its 2d nearest neighbors, so that there are exactly $|L_j| = N_j = 2^{dj}$ points in L_j for j=0,1,...,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 0 or 1 from each other. (On the actual discretization grids for PDEs, all the boundary points are distinct, so that the j-th 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 $D_j(\mathbf{x}) = u(\mathbf{x}) - u_j(\mathbf{x})$ for $\mathbf{x} \in L_j$ denotes the discretization error function on L_j for j=1,...,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 $\tilde{c} \geq 0$,

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

Thus, we need O(j) binary digits (bits) to represent the value $u_j(\mathbf{x})$ approximated to the level of truncation, which 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, 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,...,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 $\mathbf{x}$, so that $\hat{u}_{j-1}(\mathbf{x}) = u_{j-1}(\mathbf{x})$ 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 $\mathbf{x} \in L_j - L_{j-1}$. Then

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

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

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

for all $\mathbf{x} \in L_j$, j=1,...,k, and for some fixed $c \geq 0$ and $a \geq 1$, and thus the prolongation of $u_{j-1}(x)$ 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 $2^{c-a\,k}$, so as to decrease the storage space required. The straightforward fixed point binary representation of these values of $u_k(\mathbf{x})$ generally requires $N\,\lceil a\,k-c\,\rceil$ binary bits.

As an alternative, let us store $u_k(\mathbf{x})$ on L_k in the compressed form by recursively approximating within $2^{c-a\,j-a}$ to the fixed point binary values $e_j(\mathbf{x})$ for $\mathbf{x} \in L_j$, $j=1,\ldots,k$. The storage space of $2^d\,\lceil a-c\,\rceil + a\,(N_2+N_3+\ldots+N_k) < 2^d\,\lceil a-c\,\rceil + 2a\,N = O(N)$ binary bits suffices for this compressed information, which 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 $j=1,\ldots,k$. Start with $u_0(\mathbf{x}) = 0$ for $\mathbf{x} \in L_0$ and recursively, for $j=1,\ldots,k$, 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 then

b) $u_j(\mathbf{x})$ on L_j, by applying the equations (2.3).

Perform both stages a) and b) with the precision $2^{c-a\,j-a}$, so that the stage b) amounts to appending a 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 $\mathbf{x} \in L_j$, and the stage a) amounts to scanning the values of $u_{j-1}(\mathbf{x})$ on L_{j-1} and to the summation of few b-bit binary numbers [where, say, $b = O(a)$] defined by the b *least significant* binary bits in the representation of $u_{j-1}(\mathbf{x})$ for appropriate $\mathbf{x}$ from L_{j-1}. 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 [such as Gauss-Seidel, SSOR or a multigrid algorithm of the next section] for linear systems with matrices D_j 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 the 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 $j=1,\ldots,k$:

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

where

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

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 the 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 evaluate the vectors $\mathbf{e}_j$ recursively for $j=1,\ldots,k$ by applying the so called V-*cycle multigrid scheme* and arriving at the desired Compact Multigrid algorithm, in which we will exploit the compressed representation of the solution. Initially, we will let $u_0(\mathbf{x}) = 0$ for $\mathbf{x} \in L_0$, and 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 the equations (2.3)and (3.2)],

c) $e_j(\mathbf{x})$ [by solving the linear system (3.1) "to the level of truncation"],

d) $u_j(\mathbf{x})$ in the compressed form [by using the equations (2.3), as in section 6].

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

The linear convergence assumption means that, for all j, the errors of the approximations to $u_j(\mathbf{x})$ 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 the k such systems in terms of the number of arithmetic operations involved is less than $1/(1-2^{-d})$ times the solution time for the single system (2.1) for j = k, which gives us the 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 the 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 the systems (3.1), are sought with the lower precision of a 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 a 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) ops *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 strong pseudo regularity], 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.

The above results can be immediately extended to the case of more general sequences of the sets $S_0, S_1, \ldots, S_k$ of the discretization of the PDEs, provided that each set S_j consists of $c_j s^j$ points where $0 < c < c_j < c^*$, $s > 1$, c, c^* and s are constants (this includes the grids with step sizes that may vary depending on the direction of the steps), and that the weak smoothness and linear convergence assumptions are respectively extended to the case of the sets S_j. We also need to assume a constant degree bound 2d for all the discretization points, that is, each of them is supposed to have at most 2d neighbors: this will imply that each equation of the associated linear algebraic system has at most 2d+1 nonzero coefficients. If we do not bound the coefficients, we may loose the bound of O(1) bit-operations per point but may still decrease by the factor of log N the precision of the conventional multigrid methods (by using our compact scheme), and this may be a great advantage for ill-conditioned or not-so-well-conditioned PDEs.

Finally, the presented approach can be further extended to some nonlinear PDEs, as long as our assumptions [such as (2.4)] hold.

4. Some Techniques From Computer Algebra: Infinite Precision = Low Precision

Let us now shift to computer algebra, that is, to computing with no rounding-off errors, which in particular, may handle the ill-conditioned problems. Even in this case, we may decrease the precision of computations, due to using modular arithmetic, complemented by some other techniques of symbolic and algebraic computing, and next we will briefly recall some of them.

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), and then we recursively lift the resulting solution to its value modulo p^k for recursively growing k (which mean 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: two positive integers H and p, an $n \times n$ matrix A and an n-dimensional vector **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, $i=1,\ldots,H-1$. 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 $i=1,\ldots,H-1$ involves O(n) ops 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, that is, these two multiplications are performed with lower precision, which may motivate, for instance, application of 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 $A\mathbf{x}_1 = \mathbf{b} \bmod p$, $A\mathbf{y}_i = \mathbf{w}_i\ /p^i \bmod p^2$.

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 ([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 $p_1, \ldots, p_k$, by using a lower precision of $O(\log p_j)$ binary bits for $j=1, \ldots, k$, and then obtain the solution modulo $p_1 \ldots p_k$ by means of the Chinese remainder algorithm ([AHU], [BM], [Kn], [BP,a]).

If the solution is an integer 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 that $\sqrt{M/2}$, we may immediately recover the solution from its value modulo M. The integer case is trivial; to recover p and q from (p/q) mod M, we may apply the algorithm of [Wang], which invokes the continued fraction approximation algorithm and outputs the solution value p/q where p and q are integers, $\max\{|p|, |q|\} < \sqrt{M/2}$. [The continued fraction approximation algorithm ([HW79]) 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.

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 more than to $\lfloor \log_2 M \rfloor$.

The recovery of the rationals from their values modulo an integer 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 of constants F 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 N-th roots of 1 in order to perform discrete Fourier transform at N points and/or interpolation to a degree N- 1 polynomial (see section 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 the Fermat small theorem, so that the transition to E increases the computational complexity by the factor of O(log log N).

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 section 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, as was the case in section 4 for the computer algebra computations and as is the case for the numerically stable numerical algorithms. Note that in section 3, the precision of the output, of the order of log N, 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, so that 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 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, one may immediately sum n numbers by using $\lceil \log_2 n \rceil$ parallel addition steps and n processors, but by slowing down the first $\lceil \log_2 \lceil \log_2 n \rceil \rceil$ steps, we may arrive at the bound of $O_A(\log n, n/\log n)$. 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 $O_A(t, p)$ where $t = O(\log N)$, $\log p = O(\log N)$, N being the dimension of the input to the computational problem (the input size), say, the number of the coefficients of the input polynomial.

Similar definitions, $O_B(T)$ and $O_B(t, p)$, as well as 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 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 section 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.

We will next survey the known estimates for the arithmetic complexity of polynomial computations. In some cases, we will show a slight decrease of the parallel complexity of the computations by computing any precision approximation (APA) to the output rather than computing it exactly.

Table 6.1. Arithmetic Complexity of Polynomial Computations

(c denotes constants not less than 1, upper bounds are within constant factors, except for evaluation at a single point).

Problem	Sequential Arithmetic Time: Lower Bounds	Sequential Arithmetic Time: Upper Bounds	Parallel Arithmetic Complexity: Time	Parallel Arithmetic Complexity: Processors
evaluation at a point (no preconditioning)	n(–), n(*, ,)	n(–), n(*, ,)	log n	n/ log n
evaluation at a point (with preconditioning)	n(–), $\frac{n+1}{2}$ (*, ,)	n+2(–), $\frac{n+2}{2}$ (*, ,)	log n	n/ log n
evaluation and interpolation at n-th 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	$\log^2$n $(\log n)^2 \log^* n$	m(APAoutput) 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 $\log^* k$	k(APA output) k/ $\log^* k$
gcd computation (degrees m and n, N=m+n)	cN	N $\log^2$N	N log N $\log^2$N	log N N^2 (integer coefficients)
computing a zero of a polynomial with error < 2^{-b}	n+log b	n log b log n	$\log^2$n log(bn)	n log b/ log n log(bn)
computing all the zeros with error < 2^{-b}	n+log b	n^2log b log n	n log n log(bn) $\log^3$(nb)	n log b/ log(bn) $(bn)^{O(1)}$
composition of two polynomials of degree n	cn^2	$n^2 \log^2 n$	$\log^2$n $(\log n)^2 \log^* n$	n(APA output) $n^2/ \log^* n$
composition and reversion modulo z^n	cn	n log $n^{3/2}$n	$\log^2$n $(\log n)^2 \log^* n$	$(n^3/ \log n)^{1/2}$ (APA) $(n^2/ \log n)^{1/2}/ \log^* n$
decomposition of a polynomial of degree N	cN	N $\log^2$N	$(\log N)^2$ log log N $\log^2$N	N/ log log N N (APA)

Table 6.1 shows a dramatic improvement of several classical polynomial computations, which typically involve of the order of N^2 arithmetic operations for the input polynomials of degree N. This improvement is due in particular to the famous fast Fourier transform (FFT) algorithm, which only costs $1.5\,N \log_2 N$ 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 N-th roots of 1. The algorithm recursively exploits the identity $p(x) = s(y) + x\,t(y)$ where $y = x^2$, which turns into (N/2)-th roots 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 application of FFT:

We may evaluate the polynomial product u(x) v(x) (vector convolution), that is, compute $w_i = \sum_{j=0}^{i} u_j v_{i-j}$, $i=0,1,\ldots,D$, $D = \deg(u(x)\,v(x))$, by first computing u(x) and v(x) at the N-th roots of 1, for $N = 2^n > D = \deg\,(u(x)\,v(x))$, for the cost $O_A(\log N, N)$ of two FFTs at 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 problem of round-off errors in FFT 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:

Fact 6.1 ([GS]). *Let* $Q = W$ *or* $Q = W^H$, K *be a power of 2, and the d-digit floating point computation of* Q**b** *be performed by means of FFT at* K *points. Then*

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

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, which can be found in [AHU], [BM], [Kn], [BP,a], and to some extent, in sections 8 and 10 below.

Here are just two simple examples:

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

Solution. Rewrite

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

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: $w_i = \sum_{j=0}^{i} u_j v_{i-j}$, $i=m,m+1,\ldots,n$. This reduces DFT at K points to the evaluation of w_i for $i=m,m+1,..,n$, which is a part of the convolution problem of computing $w_i = \sum_{j=0}^{i} u_j v_{i-j}$, $i=0,1,\ldots,n$.

Remark 6.1. An alternate approach of [W79] decreases the above estimates for both numbers of additions and multiplications for FFT at 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* $p_0, p_1, \ldots, p_n$ *of a polynomial* $p(x) = \sum_{i=0}^{n} p_i x^i$, *compute the coefficients* $q_0(\Delta)$, $q_1(\Delta), \ldots, q_n(\Delta)$ *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* $g=1,2,..,n$ *at* $x=\Delta$).

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

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

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]:

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

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

In sections 10, 11 and 12, we will revisit some current best algorithms for polynomial division, which we will restate also in the forms of matrix and power series computations. Divisions turns 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. Only 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 computing the gcd to computing the least-squares solutions to at most $\lceil \log_2 n\rceil$ linear systems of equations with dense structured matrices and thus arrive at NC algorithms (see [BGH], [G], [P90] and Remark 9.1 in section 9 below).

Multipoint evaluation and interpolation for polynomials will give us yet another occasion (in addition to the study of computing the gcd and division) for showing some simple correlations among polynomial and structured matrix computations (see section 8). Polynomial computations are also closely related to manipulation with formal power series (which turn into polynomials being reduced modulo a power) and with integers (for a binary integer can be considered the value of a polynomial with 0 and 1 coefficients at the point x = 2). We refer the reader to [AHU], [Kn] and [BP,a] on these correlations.

In the remainder of this section, we will survey some most effective methods for approximating to polynomial zeros and to linear factors of a polynomial in the complex domain; in particular, we will outline an algorithm based on [P87a] (see also [BP,a] and [Sc]), and supporting the record estimates of Table 6.1 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 n-th degree monic polynomial,

$$p(x) = \sum_{i=0}^{n} p_i x^i = \prod_{j=1}^{n} (x - x_j),\ p_n = 1.$$

It is well-known that this problem 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.

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 only ensures 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 $\max_j |z_j| \le \max_{i<n} |p_i|$), and 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% 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 3 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 twice 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, each time followed [until we output the desired approximations to the zeros of p(x)] 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 to the average value M of the k zeros of p(x) in S and then to 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 $x\,p'(x)/p(x)$ 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), which 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 $g < 1$ 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 (the numerator). We may decrease g by moving the center of C closer to the zero of p(x), and 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). This process, being recursively repeated, very rapidly converges to the square s and thus enables us to deduce the complexity estimates that are very close to ones of Table 6.1.

To arrive at the estimates of 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 i-th powers of such zeros for $i=1,2,\ldots,k$ are also approximated by means of contour integration, and having these power sums, we may 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 the deg $f(x)$ = deg $g(x)$ in all steps.

This approach leads us to the bounds of Table 6.1 for approximating to all the zeros of $p(x)$ and (after some modification) to a single zero of $p(x)$.

The recent ingenious algorithm of [Neff] (which followed the earlier work of [BFKT]) was a breakthrough, since it 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.)

All the above algorithms can be applied to approximate to the matrix eigenvalues as to the zeros of its characteristic polynomial, if we agree to compute its coefficients first. The latter step, however, can be avoided in the important case of a symmetric tridiagonal matrix, whose eigenvalues we may directly approximate by using near optimum time and number of processors in a divide-and-conquer algorithm ([BP90], see also [Pan89], [BP,a]).

In some applications (in particular, to the evaluation of the eigenvalues), we may seek the zeros of a polynomial $p(x)$ given by a subroutine for the evaluation of $p(x)$ [and of $p'(x)$] at any fixed point x. Then the above approach does not work, and Newton's iteration $x(k+1) = x(k) - p'(x(k))/p(x(k))$ is frequently the method of choice, in particular, where the initial approximation $x(0)$ is closer to one of the zeros of $p(x)$ than to the other zeros (see [S85], [R] on the convergence estimates of Newton's method). Newton's iteration can also be used within the presented construction of [P87c] as a (generally only slightly inferior) substitution for the contour integration.

7. Arithmetic Complexity of Matrix Computations.

We will continue reviewing some polynomial computations in sections 9-12, but our next topic is computations with general matrices (in this section) and with structured matrices (in sections 8 and 9), and 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 $N \cdot N$ 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-vector multiplication and matrix inversion as the blocks of many algorithms; moreover, the straightforward $N \cdot N$ matrix multiplication, 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 algorithms that involve $O(N^b)$ arithmetic operations, where for the currently implemented algorithms, $b = \log_2 7 = 2.807...$ (Strassen) and where $b < 2.78$ in another group of algorithms, also ready for practical implementation ([LPS]).

Weak (but sufficient in most of 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 $O(N^b)$ matrix multiplication algorithms allow their parallel implementation on the PRAMs for the cost $O_A(\log N, N^b)$ ([PR], [P87b]), and theoretically the exponent b can be decreased to $b = 2.375...$ ([CW]), although immense overhead constants are hidden in this "O" notation for $b < 2.775$.

The bounds $O_A(N^b)$ and $O_A(\log N,\ N^b)$, $b < 2.376$, as N fi ¥ , are fundamental for estimating the asymptotic complexity of numerous algebraic and combinatorial computations ultimately reduced to matrix multiplications. In particular, the asymptotic complexity estimate $O_A(N^b)$, $b < 2.376$, 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 $O_A(N^b)$ has also been extended to numerous combinatorial and graph computations ([P84], [P87b], [Datta], [KUW], [GP], [GPa], [MVV], [PR,a], [BP,a]).

Some of these 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:

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

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

where $S = E - DB^{-1}C$ 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,

$\|A\|_2 \ge \max\{\|B\|_2, \|S\|_2\}$,
$\|A^{-1}\|_2 \ge \max\{\|B^{-1}\|_2, \|S^{-1}\|_2\}$,
so that 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, so that (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, which recursively leads to the bound $O_A(N^\beta)$ for $N \times N$ matrix inversion ([St69]) based on an effective and numerically stable algorithm.

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 by using the matrix equation:

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

where $\det(\lambda I - A) = \lambda^N + \sum_{i=N-r}^{N-1} c_i \lambda^i$, $c_{N-r} \ne 0$, provided [with no loss of generality since $A^+ = A(A^H A)^+$ or since $\begin{bmatrix} O & A \\ A^H & O \end{bmatrix}^+ = \begin{bmatrix} O & (A^H)^+ \\ A^+ & O \end{bmatrix}$] that A is a Hermitian matrix, that is, $A^H = A$. Here and hereafter A^H denotes the conjugate transpose of A. Note that rank $A = \mathrm{trace}(A^+A)$, and $\mathbf{x} = A^+\mathbf{b}$, over the complex, real and rational fields, is a least-squares solution to a linear system $A\mathbf{x} = \mathbf{b}$, which implies further extensions of the complexity bound $O_A(N^b)$.

As we mentioned, the practical or potentially practical range for the exponent b 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 the computations involving the coefficients of the characteristic polynomial, as in (7.3) above. Numerically stable 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:

$$X_0 = A^H / \|A^H A\|_1,\ X_{k+1} = X_K(2I - AX_k),\ k=0,1,...,K-1, \tag{7.4}$$

so that $I - AX_{k+1} = (I - AX_k)^2$ ([Be]). An improved modification of (7.4) numerically converges to A^+ in $\log_2 \mathrm{cond}_2 A$ steps (7.4), $\mathrm{cond}_2 A = \|A\|_2 \|A^+\|_2$, or if A is h.p.d., in about (1/2) $\log_2 \mathrm{cond}_2 A$ steps ([PS88]), and thus, can be recommended for the generalized inversion of well-conditioned matrices A, particularly, on parallel computers, on which matrix multiplication can be performed fast. It is interesting that $X_k = \sum_{i=0}^{s(k)} (I - X_o A)^i X_o$, $s(k) = 2^k - 1$ under (7.4), whereas $X_k = p(A^H A)A^H$, $1 - zp(z)$ is a scaled Chebyshev polynomial of degree 2^k in the improved scheme of [PS88].

The algorithms of [PS88] also compute the matrices A(e) and $(A(e))^+$, where the unknown matrix A(e) is defined by zeroing all the singular values of A that are less than e, and again, A(e) and $(A(e))^+$ are computed in [PS88] without computing the SVD of A.

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 and/or structured. We refer the reader to [GL], [LRT] and [PR88] (containing further bibliography) on the sparse case, and we will 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 $O_A(N^b)$ time 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 $n \cdot n$ band Toeplitz matrices with bandwidth $k = o(n)$, the complexity of computations may decrease 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, we write $(T)_{i,k} = t_{i-k}$ 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 $(B)_{i,k} = \frac{1}{s_i - t_k}$ for generalized Hilbert matrices B. Each such an $N \cdot N$ matrix is completely defined by one or two vectors of dimension N. Any $N \cdot N$ Hankel matrix can be turned into a Toeplitz matrix by means of interchanging its columns (or rows) s and $N + 1 - s$, for $s = 1, 2, \ldots, N$.

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 $\mathbf{b} = [b_0, \ldots, b_{N-1}]^T$ is the vector of the values of the polynomial $\sum_{k=0}^{N-1} b_k v^k$ for $v = v_i$, $i = 0, \ldots, N-1$).

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 section 4), but lead to severe problems of numerical stability in the presence of round-off errors, in which case alternate numerically stable algorithms can be applied for the cost of $O_A(N^2)$ ([GL]). Moreover, some other alternate 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 e and is bounded by O(N) for a fixed constant e).

Let us comment on computing B**b** in [Rok85]. Rewrite B**b** as $R(x_l) = \sum_{k=0}^{K-1} \frac{a_k}{x_l - w_k}$, $l=0,\ldots,L-1$, denote $\alpha = \max_k |a_k / w_k|$, $J = \lceil \log \frac{\alpha K}{(1-q)\varepsilon} / \log(\frac{1}{q}) \rceil$ where $|x_l / w_k| < q < 1$ for all k and l. Then $(3K + 2L - 1)\,J$ 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

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

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 latter power series converges for sufficiently small $|x|$. Approximate to 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:

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

Now, it remains to compute the coefficients $A_0, \ldots, A_{J-1}$ of (8.2) by using $J(3K-1)$ ops and then compute $R(x_l)$ of (8.1) for $l=0,1,\ldots,L-1$ by using 2JL ops. 5

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 $\frac{1}{x - w_k} = \frac{1}{x} \sum_{j=0}^{\infty} \left(\frac{w_k}{x}\right)^j$. [AGR], [CGR] and [ODR] contain 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 recall computing A**b** and A^{-1}**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* **c** *denote the first row of a circulant matrix* C. *Let* W *denote the* $(n+1) \times (n+1)$ *Fourier matrix,* $(W)_{i,j} = \omega^{ij} / \sqrt{n+1}$, $(W^H)_{i,j} = \omega^{-ij} / \sqrt{n+1}$, $i,j = 0,1,...,n$, $\omega^{n+1} = 1$, $\omega^s \neq 1$ *for* $0 < s \le n$. *Then* $W^H W = I$, $W^H C W = D$, $D = \mathrm{diag}(d_0,..,d_n)$ *where* $d_i = (\mathbf{d})_i$, $\mathbf{d} = \sqrt{n+1}\, W^H \mathbf{c}$.

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

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

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

The first $N = n+1$ equations of (8.3) form a linear triangular Toeplitz system T**v** = **w**. If $w_0 = 1$, $w_i = 0$ for $i > 1$, then **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 $O_A(\log N \log^* N,\ N/\log^* N)$ or (in the APA case), $O_A(N \log N)$ (see Table 6.1 and sections 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 a 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 [Ch], [Strang,a] and [ChS] 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 $T^{+}\mathbf{b}$ for the cost $O_A(\log^2 N, N^2)$ (provided that T is filled with integers or rationals) by relying on Newton's iteration (7.4) and on (7.3) (see [P89b], [P90]); 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.

Let hereafter $D(\mathbf{v}) = \mathrm{diag}(v_1, \ldots, v_N)$; 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 $\mathbf{v} = [v_1, \ldots, v_N]^T$; 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 $J\,\mathbf{v} = [v_N, \ldots, v_1]^T$, $Z\,\mathbf{B} = [0, v_1, \ldots, v_{N-1}]^T$ for a vector $\mathbf{v} = [v_1, \ldots, v_N]^T$, and we will denote $\mathbf{v}^{-1} = [v_1^{-1}, \ldots, v_N^{-1}]^T$.

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

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

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

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

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

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

Observe that rank $F_{ZZ^T}(A) = \mathrm{rank}\, r_{Z^TZ}(A) \le 2$ 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 $N \cdot N$ Vandermonde matrix V(bold v), defined by a vector $\mathbf{v}$ (here, $\mathbf{e} = [1, 1, \ldots, 1]^T$, $\mathbf{u}^T(k)$ is the unit coordinate vector with the k-th component equal to 1), 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, $(A)_{ij} = \frac{1}{s_i - t_j}$. 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 $F_{Z^TZ}(A)$], $F_{\mathbf{v},Z}(A)$ [or $F_{\mathbf{v},Z^T}(A)$] 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.

Representing the $N \cdot N$ 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 $N \cdot r$ 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 i-th columns of the matrices G and H, respectively. Then

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

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

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

Suppose that we deal with $N \cdot N$ 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, and 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 nonToeplitz matrices. For instance, rank $F_{ZZ^T}(A)$ is at most 2 for the inverse of a Toeplitz matrix, is at most 4 if A is a product of two Toeplitz matrices, and is at most $m + n$ if A is an $m \cdot n$ block matrix with Toeplitz blocks.

Given a scalar a 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 $F(A + aB)$; 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,

$$F(AB) = AB - ZABZ^T =$$
$$0.5\,(A - ZAZ^T)(B + ZBZ^T) +$$
$$0.5\,(A + ZAZ^T)(B - ZBZ^T) +$$
$$Z\,A\,(I - Z^TZ)\,BZ^T,$$

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

All this leads us to the extension of the effective algorithms of [BA], [Mus] and [AG,b] to solving Toeplitz-like linear systems $A\ \mathbf{x} = \mathbf{b}$ for the cost $O_A(N\, r^h \log^2 N)$ where 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 the matrices $F_{ZZ^T}(A^{-1})$ and $F_{ZZ^T}(S^{-1})$ have the same rank.

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.) Then we may recursively compute generators of length r_k £ $2r_{k-1} + 5$ £ $2^k*7 - 5$ for $F(X_k)$, $k=0,1,2,...,$ for the cost $O_A(\log N,\ N \mathop{S}_{h=1}^{k} r_h^2) = O_A(\log N,\ 2^k N)$.

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 k-th steps become costly for larger k. To keep their 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 $F(A^{-1}) = 2$, $\|\tilde{X}_k - \tilde{X}_k\|_2 = O(\|X_k - A^{-1}\|)$, and then we restart the iteration (7.4) with $\tilde{X}_k$ replacing 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]) or on computing the SVD of $F(X_k)$ and zeroing some smallest (positive) singular values, ([P88b]). Both approaches decrease the complexity of performing (7.4).

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 $N \cdot N$ 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 which leave open the problem of devising practical NC algorithms for all Toeplitz-like linear systems.

For the general Toeplitz-like (and, possible, ill-conditioned) linear system $A\mathbf{x} = \mathbf{b}$, we may compute [for the cost $O_A(\log^2 N, N^2)$ and by involving (7.3), and the coefficients of the characteristic polynomial of A] their 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 $N \cdot N$ matrices $A_i = I - h z^i A$, $i=0,1,...,2N-1$, where $|h|$ is a small scalar, and z 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 $\mathrm{trace}(A^j)$, $j=1,2,...,N$, and the coefficients of the characteristic polynomial of A, $\det(l I - A) = \sum_{i=0}^{N} c_i l^i$, as well as the vectors $A^j\mathbf{b}$ for $j=1,2,...,N$. Finally, compute $A^+\mathbf{b}$ by applying (7.3).

This approach and the resulting overall complexity estimate $O_A(\log^2 N, N^2)$ for computing $A^+\mathbf{b}$ apply (at least theoretically) to the case of any $N \cdot N$ Toeplitz-like matrix A filled with integers or rationals and given with a generator of length O(1) for $F_{ZZ^T}(A)$. [P90a] contains a distinct approach to the same computations, over any field F of constants, for the cost $O_A(\log^2 n, n^3 \log\log n)$, which decreases back to $O_A(\log^2 n, n^2)$ if F has characteristic equal to 0 or exceeding n and if F supports FFT or to $O_A(\log^2 n, n^2 \log\log n)$ if the characteristic of F equals 0 or exceeds n but if F does not support FFT.

Remark 9.1 on the gcd. In particular, computing the greatest common divisor (gcd) of a pair of polynomials of degrees at most N can be reduced to finding least-squares solutions for $\lceil \log_2 N \rceil$ linear systems with such matrices ([BGH], [G]), and thus, we arrive at the parallel complexity estimate $O_A(\log^3 N, N^2)$ for computing the gcd over the rationals. With a further elaboration ([BG] or [P90a]), we may extend the bounds of [P90a] to computing the polynomial gcd over a more general field F (for the cost of O_A ($\log^2 N$, $N^3 \log\log N$) over any field and O_A ($\log^2 N$, $N^2 \log\log N$) if the characteristic of F is zero).

We recall that the Hankel matrices can be turned into Toeplitz matrices by row-or-column permutations, and we will next explore some further correlations among various structured matrices.

Observe that $V^T V$ is a Hankel matrix, $(V^T V)_{h,k} = \sum_i v_i^{h+k}$, if V is a Vandermonde matrix, $(V)_{ik} = v_i^k$; this gives us $(V^T)^{-1} = V(V^T V)^{-1}$ 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 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, the generators of lengths r_2 for $F_Z(B)$ and r for F(AB) can be immediately computed given a generator of length r_1 for $F_1(A)$, $r \le r_1 + r_2 + 1$, and $r_1 \le 2$ for HT, $r_1 = 1$ for V and H. Here, the operators F_1, F_2 and F are associated with the respective classes HT, V and H (see the details in [P89a] 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, and a similar reduction we have for computing the determinants, since $A^{-1} = B(AB)^{-1}$, $\det A = \det(AB)/\det B$ for nonsingular matrices A and B. Such a reduction enables us, for instance, to reduce computing a least-squares solution to 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 to solve the original problem for the cost $O_A(N \log^2 N) = O_A(N \log^2 N, 1)$ or $O_A(\log^2 N, N^2)$, $N = m+n$.

10. Five Versions of the Polynomial Division Problem.

Let us 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*

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

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 K-th roots of 1, ω^i, $i = 0, 1, \ldots, K-1$, $K = m-n+1$, 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, $\frac{s(x)}{t(x)} = q(x) + \frac{r(x)}{t(x)}$ and $\frac{r(x)}{q(x)} \to 0$ as $x \to \infty$ since $\deg q(x) > \deg r(x)$ (see [PLS]).

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

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

It is easy to observe the equivalence of Versions 10.1 and 10.2 to each other. Furthermore, it is sufficient to solve in $q_{m-n},\ldots,q_0$ the first $K=m-n+1$ linear equations of (10.2); then (10.1) immediately defines $r_0,\ldots,r_{n-1}$. 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 Problems 10.1 and 10.2 are reduced to solving such a system.

Hereafter, we let

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

$$V(z) = T^{-1}(z) \bmod z^K = \sum_{i=0}^{K-1} v_i z^{K-1-i}$$

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 $\mathbf{q} = [q_{m-n},\ldots,q_0]^T$ can be computed as the product $T^{-1}\mathbf{s}$, $\mathbf{s} = [s_m,\ldots,s_n]^T$, or as the leading coefficients of the polynomial product $V(z) \sum_{i=n}^{m} s_i z^{i-n}$. 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 $K \geq n$). 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* $d(x) = s(x)/t(x)$ *where* s(x) *and* t(x) *are formal power series of the form*

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

and h *is a fixed positive integer.*

In particular, s(x) and t(x) are polynomials if $s_g = t_g = 0$ for all the 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 $S(z) = z^m s(1/z) = \sum_{i=0}^{\infty} s_{m-i}z^i$ 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, multiplying s(x) by x^{2h-m} and t(x) by x^{h-n} and truncating 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), we may reduce version 10.5 of the problem to the equivalent versions 10.1, 10.2, 10.3 and 10.4, 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 d(x). Thus, any solution of the problem in its versions 10.1, 10.2, 10.3 and 10.4 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 generalizes (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 $q_{m-n}, q_{m-n-1}, \ldots, q_{m-n-h+1}$ 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 the polynomial $\sum_{i=0}^{K-1} w^i(z)$. Later on 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) by truncating the power series computed by Newton's method applied to the equation $f(w,z) = 1 - (T(z)w)^{-1} = 0$, to be solved for the power series $w = w(z)$. In this case, the Newton iteration takes form of the recurrence

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

where $w_j(z)$, $j=0,1,...$, 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,

$$T_h(z) = T(z) \bmod z^h,$$

$$V_h(z) = V(z) \bmod z^h = T^{-1}(z) \bmod z^h. \tag{11.3}$$

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)]. Multiplying the identity (11.4) by $V_{ch}(z)$ and substituting $V_{ch}(z)T_{ch}(z) = 1 \bmod z^{ch}$, we arrive at (11.2).

Due to the relation (11.2), we may compute $V_{ch}(z)$ given $V_h(z)$ 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 $K = 2^{b-1}$ for an integer b, we define $f_i = \lceil b(1-1/2^{i-1})\rceil$ and recursively apply (11.2) for $c = c_i$, $h = h_i$, $h_i = 2^{f_i}$, $c_i = h_{i+1}/h_i = 2^{f_{i+1}-f_i}$, $i=1,\ldots,\lceil\log b\rceil+1$, so that we compute $V_{h_{i+1}}(z)$ [given $V_{h_i}(z)$] for the cost $O_A(f_{i+1}, 2^{2f_{i+1}-f_i}) = O_A(\log K, K)$.

Recursively repeating this computation for $i=1,2,\ldots,\lceil\log b\rceil+1$, $b-1 = \log K$, 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 $a = \lfloor\log(K/\log K)\rfloor$ and compute $T_{2^a}(z)$ by applying the above scheme, for the cost $O_A(\log K \log\log K, K/\log K)$.

Stage 2. Compute $T_K(z)$ given $T_{2^a}(z)$ 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 [BP90], [BP90a], shown at the end of this section.

Table 11.1 (Arithmetic Cost of Polynomial Division Algorithms)

Algorithm	Arithmetic operations	Parallel steps	Processors
Classical (Problem 1.3.1)	O(K min{K,n})	O(K)	min{K,n}
Sieveking-Kung (Problem 1.3.4)	O(K log K)	O($\log^2$K)	K/ log K
Bini- Schönhage [B],[Sc]	O(K log K)	O(log K)	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 Ø$^{-h}$ $\log^{(h)}$Kø)	O(h log K)	(K/ h) Ø$^{-h}$ $\log^{(h)}$Kø
	O(K log K)	O(log K $\log^*$K)	K/ $\log^*$K

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 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] (page 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,...,h, according to the following formulae:

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

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:

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

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 $u^{(i)}(z) = -T(z)\,w^{(i)}(z)$ [all other coefficients of $1 + u^{(i)}(z)$ are zeros] is equivalent to the evaluation of the matrix-by-vector product $\mathbf{g}_i = -W_i \mathbf{v}_i$, and 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:

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

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 $\mathbf{s} = [s_m, ..., s_n]^T$ and $\mathbf{q} = [q_{k-1}, ..., q_0]^T$) 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 D-th roots of 1:

a) $T_{i-1}(z) = T(z) \bmod z^{k2^i}$, concurrently for $i = 1, \ldots, d$, 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 $i = 1, \ldots, d$.

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 $i = 1, \ldots, d$, 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], [BP90a] we will improve this bound to

$$c(K,h) = O_A(h \log K,\ (K/h)(1 + 2^{-h} \log^{(h)} K)) \text{ for any fixed } h,\ h = 1, \ldots, \log^* K. \tag{11.6}$$

Fix h and recursively apply the same algorithm, replacing K by K_j and k by k_j, where $k_0 = 1$, $k_{j+1} = K_j$, $j = 0, 1, \ldots$. Denote $K(h) = K2^{-h}$, $s_j = K(h)/k_j$, and set $K_j = \lceil K(h)/\log^2 s_j \rceil$, $j = 0, 1, \ldots, J-1$, 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 $K_j = K(h)/s_j$, and we note that $s_0 = K(h)$, $s_1 = \log^2 K(h)$, $s_2 = \log^2 \log^2 K(h), \ldots,$ so that $s_J = O(\log^{(h)} K(h))$ for some $J \le 2h$.

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, $O_A(h \log K(h),\ \frac{K(h)}{h} \log^{(h)} K(h))$. 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), and we observe that (11.6) turns into $O_A(\log K \log^* K, \frac{K}{\log^* K})$ for $h = \log^* K$, since $\log^{(h)} K \le 1$ 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 arbitrarily high precision. The algorithm exploits, in particular, the representation of a lower triangular Toeplitz matrix $T = [t_{n+j-i}]$, $t_s = 0$ unless $0 \le s \le n$, as $T = \sum_{i=0}^{m-n} t_{n-i} Z^i$ where Z is the displacement matrix defined in section 9. This representation also enables us to obtain the following a priori bound on the coefficients of q(x) in terms of the coefficients of s(x) and t(x):

$$\sum_{i=0}^{K-1} |q_i| \le (1 + t/t_n)^{K-1} \sum_{i=n}^{m} |s_i| / t_n \qquad (12.1)$$

where $t = \max_{0 \le i \le m-n} |t_{n-i}|$, $t_g = 0$ 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 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 [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 these cases (of the division and computing the gcd) the binary segmentation generally involves operations with long integers, so that the techniques of section 4 would be required in order to implement the resulting algorithms.

Remark 12.1. The approximation algorithm of [B] has been originally obtained for the triangular Toeplitz matrix inversion, but has also two equivalent versions: in the forms 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 the equation (10.1).

References

[AHU] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1976.

[ASU] A.V. Aho, K. Steiglitz, and J.D. Ullman, ‘‘Evaluating Polynomials at Fixed Set of Points,’’ *SIAM J. on Computing*, vol. 4, pp. 533-539, 1975.

[AR] B. Alpert and V. Rokhlin, ‘‘A Fast Algorithm for the Evaluation of Lagrange Expansions,’’ Research Report YALEU/DCS/RR-671, Yale University, Dept. of Computer Science, 1989.

[AGR] J. Ambrosiano, L. Greengard, and V. Rokhlin, ‘‘The Fast Multipole Method for Gridless Particle Simulation,’’ Research Report RR-565, *Yale Univ., Dept. of Computer Science*, 1987.

[AG,b] G.S. Ammar and W.G. Gragg, ‘‘Superfast Solution of Real Positive Definite Toeplitz Systems,’’ *SIAM J. on Matrix Analysis and Applications*, vol. 9,1, pp. 61-76, 1988.

[BD81] R. Bank and T. Dupont, ‘‘An Optimal Order Process for Solving Finite Element Equations,’’ *Mathematics of Computation*, vol. 36, pp. 35-51, 1981.

[Be] A. Ben-Israel, ‘‘A Note on Iterative Method for Generalized Inversion of Matrices,’’ *Math. Computation*, vol. 20, pp. 439-440, 1966.

[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. on Computing*, vol. 17,6, pp. 1081-92, 1989. (Short version in *Proc. 18-th Ann. ACM Symp. on Theory of Computing*, pp. 340-349, 1986).

[B] D. Bini, ‘‘Parallel Solution of Certain Toeplitz Linear Systems,’’ *SIAM J. Comp.*, vol. 13,2, pp. 268-276, 1984. Also T.R.B82-04, *I.E.I. of C.N.R.*, Pisa, Italy (April 1982).

[BG] D. Bini and L. Gemignani, ‘‘On the Euclidean Scheme for Polynomials Having Interlaced Real Zeros,’’ Proc. *2nd ACM Symp. on Parallel Algorithms and Architecture*, pp. 254-258, 1990.

[BL80] D. Bini and G. Lotti, Stability of Fast Algorithms for Matrix Multiplication, *Numerische Math.*, vol. 36, pp. 63-72, 1980.

[BP] D. Bini and V. Pan, ''Polynomial Division and Its Computational Complexity,'' *Journal of Complexity*, vol. 2, pp. 179-203, 1986.

[BP88] D. Bini and V. Pan, ''Efficient Algorithms for the Evaluation of the Eigenvalues of (Block) Banded Toeplitz Matrices,'' *Math. of Computations*, vol. 50,182, pp. 431-448, 1988.

[BP90] D. Bini and V. Pan, ''Computing Matrix Eigenvalues and Polynomial Zeros Where the Output is Real,'' Tech. Rep. CUCS 025-90, *Computer Science Dept., Columbia University*, NY, 1990.

[BP90a] D. Bini and V. Pan, ''Improved Parallel Polynomial Division,'' Tech. Report CUCS 026-90, *Computer Science Dept., Columbia Univ.*, 1990.

[BPc] D. Bini and V. Pan, ''On the Evaluation of the Eigenvalues of a Banded Toeplitz Block Matrix,'' Tech. Rep. CUCS-024-90, *Columbia University, Computer Science Dept.*, N.Y., 1990.

[BP,a] D. Bini and V. Pan, *Numerical and Algebraic Computations with Matrices and Polynomials*, Birkhaüser, Boston, 1991.

[BA] R.R. Bitmead and B.D.O. Anderson, ''Asymptotically Fast Solution of Toeplitz and Related Systems of Linear Equations,'' *Linear Algebra and Its Applics.*, vol. 34, pp. 103-116, 1980.

[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,'' *Information and Control*, vol. 52,3, pp. 241-256, 1982.

[Brent70] R.P. Brent, ''Error Analysis of Algorithms for Matrix Multiplication and Triangular Decompositions Using Winograd's Identity,'' *Numerische Math.*, vol. 16, pp. 145-

156, 1970.

[Bre] R.P. Brent, ''The Parallel Evaluation of General Arithmetic Expressions,'' *J. ACM*, vol. 21,2, pp. 201-206, 1974.

[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. of Algorithms*, vol. 1, pp. 259-295, 1980.

[Bun] J.R. Bunch, ''Stability of Methods for Solving Toeplitz Systems of Equations,'' *SIAM J. on Scientific and Statistical Computing*, vol. 6,2, pp. 349-364, 1985.

[CGR] J. Carier, L. Greengard, and V. Rokhlin, ''A Fast Adaptive Multipole Algorithm for Particle Simulation,'' Research Report RR-496, *Yale Univ., Dept. of Computer Science*, 1987.

[ChS] R.H. Chan and G. Strang, ''Toeplitz Equations by Conjugate Gradients with Circulant Preconditioner,'' *SIAM J. Sci. Stat. Comput.*, vol. 10, pp. 104-119, 1989.

[Ch] T.F. Chan, ''An Optimal Circulant Preconditioner for Toeplitz Systems,'' *SIAM J. Sci. Stat. Comput.*, vol. 9, pp. 766-771, 1988.

[CKL-A] J. Chun, T. Kailath, and H. Lev-Ari, ''Fast Parallel Algorithm for QR-factorization of Structured Matrices,'' *SIAM J. on Scientific and Statistical Computing*, vol. 8,6, pp. 899-913, 1987.

[CW] D. Coppersmith and S. Winograd, ''Matrix Multiplication via Arithmetic Progressions,'' *J. of Symbolic Computations* , vol. 9,3, 1990 (short version in *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pp. 1-6, 1987).

[CB] G. Cybenko and M. Berry, ''Hyperbolic Householder Algorithms for Factoring Structured Matrices,'' Tech. Report, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL , 1989, to appear in *SIAM J. on Matrix Analysis.*.

[Datta] K. Datta, ‘‘Parallel Complexity and Computations of Cholesky’s Decomposition and QR Factorization,’’ *International J. Computer Math.*, vol. 18, pp. 67-82, 1985.

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

[dH] F.R. deHoog, ‘‘On the Solution of Toeplitz Systems,’’ *Linear Algebra and Its Applics.*, vol. 88/89, pp. 123-138, 1987.

[Eberly] W. Eberly, ‘‘Very Fast Parallel Polynomial Arithmetic,’’ *SIAM J. on Computing*, (to appear).

[FP] M.J. Fischer and M.S. Paterson, ‘‘String Matching and Other Products,’’ *SIAM-AMS Proc.*, vol. 7, pp. 113-125, 1974.

[FMc] P.O. Frederickson and O.A. McBryan, ‘‘Parallel Superconvergent Multigrid,’’ in *Multigrid Methods: Theory, Applications and Supercomputing*, ed. S. McCormick, vol. 100, pp. 195-210, M. Decker, Inc., Lecture Notes in Pure and Applied Math, 1988.

[FMc87a] P.O. Frederickson and O.M. McBryan, ‘‘Superconvergent Multigrid Methods,’’ Preprint, *Cornell Theory Center*, May 1987.

[GP] Z. Galil and V. Pan, ‘‘Improving Processor Bounds for Algebraic and Combinatorial Problems in RNC,’’ *Proc. 26-th Ann. IEEE Symp. on Foundation of Computer Sci.,*, pp. 490-495, Portland, Oregon, 1985.

[GPa] Z. Galil and V. Pan, ‘‘Improved Processor Bounds for Combinatorial Problems in RNC,’’ *Combinatorica*, vol. 8,2, pp. 189-200, 1988.

[Gast60] N. Gastinel, ‘‘Inversion d’une Matrice Generalisant la Matrice de Hilbert,’’ *Chiffres*, vol. 3, pp. 149-152, 1960.

[G] J. von zur Gathen, ‘‘Parallel Algorithms for Algebraic Problems,’’ *SIAM J. on Comp.*, vol. 13,4, pp. 802-824, 1984.

[GS] W. Gentleman and G. Sande, ‘‘Fast Fourier Transform for Fun and Profit,’’ *Proc. Fall Joint Comput. Conf.*, vol. 29, pp. 563-578, 1966.

[GeLi] J.A. George and J.W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, New Jersey, 1981.

[Geor] R.E. Georgiev, ‘‘Inversion of Triangular Toeplitz Matrices by Using the Fast Fourier Transform,’’ *J. New Gener. Comput. Syst.*, vol. 2,3, pp. 247-256, 1989.

[Ger87] A. Gerasoulis, ‘‘A Fast Algorithm for the Multiplication of Generalized Hilbert Matrices with Vectors,’’ *Math. of Computations*, vol. 50, 181, pp. 179-188, 1987.

[GKKL] I. Gohberg, T. Kailath, I. Koltracht, and P. Lancaster, ‘‘Linear Complexity Parallel Algorithms for Linear Systems of Equations with Recursive Structure,’’ *Linear Algebra and Its Applications*, vol. 88/89, pp. 271-315, 1987.

[GL] G.H. Golub and C.F. van Loan, *Matrix Computations*, Johns Hopkins Univ. Press, Baltimore, Maryland, 1989.

[Ha77] W. Hackbusch, ‘‘On the Convergence of Multi-grid Iteration Applied to Finite Element Equations ,’’ Report 77-8, Universität zu Köln, 1977.

[Hac80] W. Hackbusch, ‘‘Convergence of Multi-grid Iterations Applied to Difference Equations,’’ *Mathematics of Computation*, vol. 34, pp. 425-440, 1980.

[Hac85] W. Hackbusch, *Multi-Grid Methods and Applications*, Springer, Berlin, 1985.

[HT82] W. Hackbusch and U. Trottenberg (eds.), ‘‘*Multigrid Methods*,’’ Springer’s Lecture Notes in Math., vol. 960, 1982.

[HW79] G.H. Hardy and E.M. Wright, *An Introduction to the Theory of Numbers*, Oxford Univ. Press, Oxford, 1979.

[He] P. Henrici, *Applied and Computational Complex Analysis*, Wiley, N.Y., 1974.

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

[KKM] T. Kailath, S.-Y. Kung, and M. Morf, ‘‘Displacement Ranks of Matrices and Linear Equations,’’ *J. Math. Anal. Appl.*, vol. 68,2, pp. 395-407, 1979.

[KVM] T. Kailath, A. Viera, and M. Morf, ‘‘Inverses of Toeplitz Operators, Innovations, and Orthogonal Polynomials,’’ *SIAM Review*, vol. 20,1, pp. 106-119, 1978.

[KR] R. Karp and V. Ramachandran, ‘‘A Survey of Parallel Algorithms for Shared Memory Machines,’’ *Handbook of Theoretical Computer Science*, North-Holland, Amsterdam, 1991.

[KUW] R.M. Karp, E. Upfal, and A. Wigderson, ‘‘Constructing a Perfect Matching Is in Random NC,’’ *Proc. 17-th Ann. ACM Symp. on Theory of Computing*, pp. 22-32, 1985.

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

[LPS] J. Laderman, V. Pan, and X-H Sha, ‘‘On Practical Acceleration of Matrix Multiplication,’’ Techical Report, TR 90-14, *Computer Science Dept., SUNYA*, Albany, NY, 1990.

[Laf] J.C. Lafon, ‘‘Base Tensorielle des matrices de Hankel (ou de Toeplitz), Applications,’’ *Numerische Math.*, vol. 23, pp. 249-361, 1975.

[LRT] R.J. Lipton, D. Rose, and R.E. Tarjan, ‘‘Generalized Nested Dissection,’’ *SIAM J. on Numerical Analysis*, vol. 16,2, pp. 346-358, 1979.

[McC87] S. McCormick, editor, *Multigrid Methods*, SIAM, Philadelphia, 1987.

[McT83] S. McCormick and U. Trottenberg (eds.), ‘‘Multigrid Methods,’’ in *Appl. Math. Comp.*, vol. 13, pp. 213-474, 1983.

[Mc86] S. McCormick (ed.), ‘‘Proceeding of the 2nd Copper Mountain Multigrid Conference,’’ *Appl. Math. Comp.*, vol. 19, (special issue), pp. 1-372, 1986.

[MC] R.T. Moenck and J.H. Carter, ‘‘Approximate Algorithms to Derive Exact Solutions to Systems of Linear Equation,’’ *Proc. EUROSAM, Lecture Notes in Computer Science*, vol. 72, pp. 63-73, Springer, 1979.

[MVV] K. Mulmuley, U. Vazirani, and V. Vazirani, ‘‘Matching Is As Easy As Matrix Inversion,’’ *Combinatorica*, vol. 7,1, pp. 105-114, 1987.

[Mus] B.R. Musicus, ‘‘Levinson and Fast Choleski Algorithms for Toeplitz and Almost Toeplitz Matrices,’’ Internal Report, *Lab. of Electronics, M.I.T.*, 1981.

[Neff] C.A. Neff, ‘‘Polynomial Rootfinding is in NC,’’ *Proceedings 31st Ann. IEEE Symp., FOCS*, 1990.

[ODR] S.T. O’Donnell and V. Rokhlin, ‘‘A Fast Algorithm for Numerical Evaluation of Conformal Mappings,’’ *SIAM J. on Scientific and Statistical Computing*, vol. 10, 3, pp. 475-487, 1989.

[P84] V. Pan, *How to Multiply Matrices Faster,* Lecture Notes in Computer Science, 179, Springer Verlag, 1984.

[P87b] V. Pan, ‘‘Complexity of Parallel Matrix Computations,’’ *Theoretical Computer Science,* vol. 54, pp. 65-85, 1987.

[P87a] V. Pan, ‘‘Sequential and Parallel Complexity of Approximate Evaluation of Polynomial Zeros,’’ *Computers and Mathematics (with Applications)*, vol. 14,8, pp. 591-622, 1987.

[P87c] V. Pan, ‘‘Sequential and Parallel Complexity of Approximate Evaluation of Polynomial Zeros,’’ *Computers and Mathematics (with Applications)*, vol. 14,8, pp. 591-622, 1987.

[P88b] V. Pan, ‘‘New Methods for Computations with Toeplitz-like Matrices,’’ *Technical Report 88-28, Computer Science Dept., SUNY Albany*, 1988.

[Pan89] V. Pan, ‘‘A New Algorithm for the Symmetric Eigenvalue Problem,’’ Techn. Report TR 89-3, Computer Science Dept., SUNYA, 1989.

[P89b] V. Pan, ‘‘Parallel Inversion of Toeplitz and Block Toeplitz Matrices,’’ *Operator Theory: Advances and Applications*, vol. 40, pp. 359-389, Birkhauser, Basel, 1989.

[P89a] V. Pan, ‘‘On Some Computations with Dense Structured Matrices,’’ *Proc. ACM-SIGSAM Intern. Symp. on Symbolic and Alg. Comp.*, pp. 34-42, 1989 and *Math. of Comp.*, vol. 55, 191, pp. 179-190, 1990.

[P90] V. Pan, ''Parallel Least-Squares Solution of General and Toeplitz-like Linear Systems,'' *Proc. 2nd Ann. ACM Symp. on Parallel Algorithms and Architecture*, pp. 244-253, 1990.

[P90a] V. Pan, ''Parametrization of Newton's Iteration for Computations with Structured Matrices and Applications,'' Tech. Report CUCS-032-90, *Columbia University, Computer Science Dept.*, 1990.

[PLS] V. Pan, E. Landowne, and A. Sadikou, ''Approximate Polynomial Division with a Remainder by Means of Evaluation and Interpolation,'' Tech. Report CUCS-030-90, *Columbia University, Computer Science Dept.*, NY, 1990.

[PR] V. Pan and J. Reif, ''Efficient Parallel Solution of Linear Systems,'' *Proc. 17-th Ann. ACM Symp. on Theory of Computing*, pp. 143-152, Providence, R.I., 1985.

[PR88] V. Pan and J. Reif, ''Fast and Efficient Parallel Solution of Sparse Linear Systems,'' Technical Report 88-18, Computer Science Dept., SUNYA, 1988.

[PR,a] V. Pan and J. Reif, ''Fast and Efficient Solution of Path Algebra Problems,'' *J. Computer and System Sciences*, vol. 38, pp. 494-510, 1989.

[PR89a] V. Pan and J. Reif, ''On the Bit-Complexity of Discrete Solution of PDEs: Compact Multigrid,'' *Computers and Mathematics (with Applications)*, (to appear in 1990).

[PS88] V. Pan and R. Schreiber, ''An Improved Newton Iteration for the Generalized Inverse of a Matrix, with Applications,'' Technical Report 88-35, Computer Science Dept., SUNYA, 1988 (to appear in *SIAM J. Sci. Stat. Comp.*).

[R] J. Renegar, ''On the Worst-Case Arithmetic Complexity of Approximating Zeros of Polynomials,'' *J. of Complexity*, vol. 3,2, pp. 90-113, 1987.

[Rok85] F. Rokhlin, ''Rapid Solution of Integral Equations of Classical Potential Theory,'' *J. Comput. Physics*, vol. 60, pp. 187-207, 1985.

[Rok] V. Rokhlin, ''A Fast Algorithm for the Discrete Laplace Transformation,'' *J. of Complexity*, vol. 4, pp. 12-32, 1988.

[Sc82] A. Schönhage, ''Asymptotically Fast Algorithms for the Numerical Multiplication and Division of Polynomials with Complex Coefficients,'' *Proc. EUROCAM*, Marseille, 1982.

[Sc] A. Schönhage, ''The Fundamental Theorem of Algebra in Terms of Computational Complexity,'' manuscript, *Dept. of Math., University of Tübingen*, Tübingen, West Germany, 1982.

[S85] S. Smale, ''On the Efficiency of the Algorithms of Analysis,'' *Bull. Amer. Math. Soc.*, vol. 13,2, pp. 87-121, 1985.

[Strang,a] G. Strang, ''A Proposal for Toeplitz Matrix Calculations,'' *Stud. Appl. Math.*, vol. 74, pp. 171-176, 1986.

[St69] V. Strassen, ''Gaussian Elimination is Not Optimal,'' *Numerische Math.*, vol. 13, pp. 354-356, 1969.

[UP83] S. Ursic and C. Patarra, ''Exact Solution of Systems of Linear Equations with Iterative Methods,'' *SIAM J. on Algebraic and Discrete Methods*, vol. 4, pp. 111-115, 1983.

[Wang] P. Wang, ''A p-adic Algorithm for Univariate Partial Fractions,'' *Proc. 1981 ACM Symp. on Symbolic and Algebraic Comp.*, pp. 212-217, 1981.

[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.*, vol. 32, pp. 83-117, 1979.