Comp. & Maths. with Appls. Vol. 12A, No. 11, pp. 1105–1118, 1986
Printed in Great Britain.
0886-9553/86 $3.00 + 0.00

FAST PARALLEL ALGORITHMS FOR POLYNOMIAL DIVISION OVER AN ARBITRARY FIELD OF CONSTANTS

D. BINI
Department of Mathematics, University of Pisa, Pisa, Italy

V. PAN
Department of Computer Science, SUNY, Albany, NY 12222, U.S.A.

(*Received February 1986*)

Communicated by E. Y. Rodin

Abstract—It is shown that the division of an mth-degree polynomial by an nth-degree polynomial can be performed over an arbitrary field of constants F involving $O(\log(m-n+1)\log\log(m-n+1))$ parallel steps and a polynomial in m number of processors. If the field F has more than $2+2(m-n+1)\min\{n, m-n\}$ distinct elements, then $O(\log(m-n+1))$ parallel steps and the polynomial number of processors suffice. The values of the constants for these asymptotic bounds are also presented. The number of parallel steps can be reduced to $O(\log(m-n+1))$ over an arbitrary field using polynomial circuits for the operations in the extended field of constants. These methods can be extended to the design of parallel algorithms of the same asymptotic complexity for the inversion of circulant matrices, even where the field of constants does not support FFT. The latter result does not follow from the recent methods by the authors and Eberly, which otherwise lead to similar results by different means.

1. INTRODUCTION

In this paper we consider the following computational problem of polynomial division.

Problem 1.1

Given the coefficients of two polynomials

$$X_m(t) = \sum_{i=0}^{m} x_i t^i$$

and

$$Y_n(t) = \sum_{i=0}^{n} y_j t^j$$

of degree m and n, respectively, $x_m \neq 0$, $y_n \neq 0$; compute the coefficients of the unique pair of polynomials

$$Q_{m-n}(t) = \sum_{h=0}^{m-n} q_h t^h$$

and

$$R_{n-1}(t) = \sum_{k=0}^{m-1} r_k t^k$$

such that, identically in t,

$$X_m(t) = Q_{m-n}(t) Y_n(t) + R_{n-1}(t). \tag{1}$$

It is well-known that Problem 1.1 can be reduced to polynomial multiplication and then can be solved using Toom's evaluation–multiplication–interpolation scheme where the

evaluation and interpolation can be performed by fast Fourier transform (FFT) provided that the computation is over the field of complex constants. In that case $O(m \log m)$ arithmetical operations suffice for the solution of Problem 1.1, see Refs [1, 2]; see also Ref. [3] on the parallel complexity of Problem 1.1 over rational constants.

In this paper we will present some parallel algorithms for the same problem over an *arbitrary* field of constants. We will rely on the following (rather trivial) reduction of Problem 1.1.

Lemma 1.1

Problem 1.1 can be solved via the following operations: (i) solving a system of $m-n+1$ linear equations with $m-n+1$ unknowns whose coefficient matrix is a band upper-triangular Toeplitz matrix with the bandwidth

$$w = \min\{n, m-n\}; \tag{2}$$

(ii) multiplying a pair of polynomials in t of degree $m-n$ and n modulo t^n; and (iii) subtracting two polynomials of degree less than n one from the other.

Proof. The identity (1) is equivalent to the following system of linear equations in $q_0, q_1, \ldots, q_{m-n}; r_0, r_1, \ldots, r_{n-1}$:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} y_0 & 0 & 0 & & & \\ y_1 & y_0 & 0 & & & \\ y_2 & y_1 & y_0 & & & \\ \vdots & \vdots & \vdots & & & \\ y_n & y_{n-1} & y_{n-2} & & & \\ & y_n & y_{n-1} & & & y_0 \\ & & y_n & & & y_1 \\ & & & & & \\ & & & & & y_{n-2} \\ & & & & & y_{n-1} \\ & & & & & y_n \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ \vdots \\ q_{m-n} \end{bmatrix} + \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ \vdots \\ r_{n-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{3}$$

The last $m-n+1$ equations of the latter system form the following triangular Toeplitz system of linear equations in $q_0, q_1, \ldots, q_{m-n}$,

$$\begin{bmatrix} x_n \\ x_{n+1} \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} y_n & y_{n-1} & y_{n-2} & \cdots & y_{2n-m+1} & y_{2n-m} \\ & y_n & y_{n-1} & \cdots & & y_{2n-m+1} \\ & & \ddots & & & \vdots \\ & & & & & y_{n-2} \\ & & & & & y_{n-1} \\ & & & & & y_n \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ \vdots \\ q_{m-n} \end{bmatrix}. \tag{4}$$

Here $y_j = 0$ if $j < 0$.

Solving the latter system we obtain the coefficients of the polynomial $Q_{m-n}(t)$. Then the coefficients of $R_{n-1}(t)$ can be obtained from the first n equations of system (3) using the coefficients of $X_m(t)$, $Y_n(t)$ and $Q_{m-n}(t)$ that are already known. The latter evaluation amounts to stages (ii) and (iii) in the statement of Lemma 1.1.

Remark 1.1

Obviously the proof of Lemma 1.1 can be converted so that, performing polynomial division, we obtain a solution to a triangular Toeplitz system of linear equations presented in the form of system (4).

The evaluation of the coefficients of $R_{n-1}(t)$ (provided that those of $X_m(t)$, $Y_n(t)$ and $Q_{m-n}(t)$ are already known) can be performed by the straightforward algorithm in $2+\log(w+1)$ parallel steps using at most $(w+1)(2n-w)/2$ processors. Here and hereafter w is defined by equation (2) provided that we deal with Problem 1.1; all logarithms are to the base 2; and $\log n$ designates the value $\lceil \log n \rceil$, that is, the minimum integer that is not less than $\log n$.

Remark 1.2

The number of parallel steps of an algorithm is also called the *depth* of the associated arithmetic circuit; the size of that circuit is equal to the number of processors involved in the algorithm, cf. Ref. [4].

In Ref. [5] (and also in Ref. [6]) the band triangular Toeplitz system (4) has been solved in $7\log(m-n+1)+7$ parallel steps using $2.5(w+1)(m-n+1)$ processors over the field of complex numbers. Thus the total complexity of the solution of Problem 1.1 over the field of complex numbers is at most $9+7\log(m-n+1)+\log(w+1)$ parallel steps and at most $(w+1)\max\{2.5(m-n+1), 0.5(2n-w)\}$ processors.

This result was extended in Ref. [4] to the solution of Problem 1.1 in $O(\log n)$ parallel steps over rational constants where a different approach has been applied. The approach of Ref. [4] reduces polynomial division to polynomial multiplication via the representation of the inverse of a polynomial in the form of a power series. Two different extensions of that algorithm to the case of computing over an arbitrary field F of constants have been presented in Refs [7, 8].

In this paper we will obtain a similar extension based on the earlier algorithm of Ref. [5]. At first we will replace the special generator matrix of the approximating algebra of matrices of Ref. [5] by a more general class of generators in the form of companion matrices. This will immediately lead us to an algorithm for Problem 1.1 that works over an arbitrary field F of constants that contains at least $2(m-n+1)w+2$ distinct elements. This algorithm involves $O(\log(m-n+1))$ parallel steps and $O(w(m-n+1)^3+w(2n-w))$ processors. If the field F contains too few elements, then the standard approach is to extend F to a larger field of polynomials over F modulo a polynomial. We will apply that approach in two ways both leading to the algorithms for Problem 1.1 that work over an arbitrary field of constants. In one case the algorithms involve $O(\log(m-n+1)\,\mathrm{loglog}(m-n+1))$ parallel steps and $O((m-n+1)^4w^3\log^2(m-n+1)+w(2n-w))$ processors and in another case $O(\log(m-n+1)\,\mathrm{loglog}(m-n+1)\,\mathrm{logloglog}(m-n+1))$ steps and $O((m-n+1)^3w\log^7(m-n+1)[\mathrm{loglog}(m-n+1)]^2+w(2n-w))$ processors, see Corollary 6.6 and Theorem 6.7 in Section 6 and compare Corollaries 6.2 and 6.4 where we specify the constants hidden in the above estimates under the notation O. Of course, we have an alternative of counting all operations in the extension field of constants. This would preserve the bound $O(\log(m-n+1))$ on the number of parallel steps.

We will use the following order of presentation. In the next section we will recall the algorithm from Ref. [5] over the field of complex constants. In Section 3 we generalize the first part of that algorithm where we compute some auxiliary values. These values are used in Section 4 at the interpolation stage in order to obtain the solution of a triangular Toeplitz system of linear equations. In Section 5 we estimate how many nodes of interpolation are needed in the algorithm. In Section 6 we estimate the computational cost of the algorithm, first in the case where the given field of constants F contains sufficiently

many elements to be used as the nodes of interpolation and then in the case where we extend F to a larger field that has sufficiently many elements. We will also comment on some extensions of our method to the fast parallel solution of systems of linear equations whose matrices have certain structures, in particular, we consider the case where such matrices are circulant.

2. APPROXIMATE AND EXACT SOLUTIONS OVER THE FIELD OF COMPLEX NUMBERS (OUTLINE)

In this section we will recall the algorithm of Ref. [5]. The algorithm solves Problem 1.1 with any prescribed precision. If the exact solution is needed, then such a solution is obtained from the approximate solution by interpolation. This approach, due to Bini [9], was successfully applied to the design of efficient algorithms in the case of matrix multiplication, see Refs [10–12].

Specifically, consider the algebra τ_ϵ of $s \times s$ matrices generated by the following $s \times s$ matrix,

$$\mathbb{H}_\epsilon = (h_{ij}^{(\epsilon)}), \quad h_{ij}^{(\epsilon)} = \begin{cases} 1 & \text{if } j = i+1, \\ \epsilon^s & \text{if } i = s, j = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Then τ_0 is the algebra of all $s \times s$ upper-triangular Toeplitz matrices and every matrix of that algebra τ_0 can be approximated with any prescribed precision by a matrix from τ_ϵ provided that $|\epsilon|$ is sufficiently small. For illustration, here is the general 4×4 matrix of the class τ_ϵ:

$$\mathbb{A} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ \epsilon^s a_4 & a_1 & a_2 & a_3 \\ \epsilon^s a_3 & \epsilon^s a_4 & a_1 & a_2 \\ \epsilon^s a_2 & \epsilon^s a_3 & \epsilon^s a_4 & a_1 \end{bmatrix}.$$

If $\epsilon \neq 0$, then every invertible matrix $\mathbb{A}_\epsilon$ from the algebra τ_ϵ can be inverted using the following identity which does not hold for $\epsilon = 0$, see Ref. [5].

$$\mathbb{A}_\epsilon^{-1} = \mathbb{D}_\epsilon \Omega \mathbb{D}_\epsilon^{-1}(\mathbb{A}_\epsilon) \Omega^H \mathbb{D}_\epsilon^{-1}. \tag{6}$$

Here Ω and Ω^H are the matrices of forward and backward Fourier transform at s points, $\Omega\Omega^H = \mathbb{I}$; $\mathbb{D}_\epsilon = \operatorname{diag}(1, \epsilon, \epsilon^2, \ldots, \epsilon^{s-1})$; $\mathbb{D}_\epsilon(\mathbb{A}_\epsilon)$ is the diagonal matrix formed by the eigenvalues of $\mathbb{A}_\epsilon$.

The identity (6) enables us to compute $\mathbb{A}_\epsilon^{-1}$ quickly provided we compute over a field of constants that supports FFT at s points, e.g. over the field of complex numbers. We will cite the following result from Ref. [5].

Theorem 2.1

If $\epsilon \neq 0$, then $6 \log s + 3$ parallel steps with $2s$ processors [that means at most $6s(2 \log s + 1)$ arithmetical operations] suffice in order to invert an arbitrary invertible $s \times s$ matrix $\mathbb{A}_\epsilon$ from the algebra τ_ϵ performing over a field of constants that supports FFT at s points. Moreover, three additional steps with no increase in the number of processors suffice in order to evaluate $\mathbb{A}_\epsilon^{-1}\mathbf{u}$ for an arbitrary vector $\mathbf{u}$.

If we compute with real or complex constants, then Theorem 2.1 implies that the inverse $\mathbb{A}^{-1}$ of an arbitrary invertible $s \times s$ upper-triangular Toeplitz matrix $\mathbb{A}$ and the product $\mathbb{A}^{-1}\mathbf{u}$ for an arbitrary vector $\mathbf{u}$ can be evaluated with any prescribed precision using the cited numbers of steps, processors and arithmetical operations because, as we noted, the matrix $\mathbb{A}$ can be approximated by the matrices from the algebra τ_ϵ with any prescribed precision provided that $|\epsilon|$ is sufficiently small.

For any practical purpose, computing with arbitrarily small error by the cited algorithms is as good as computing exactly (this fact has been confirmed by the numerical experiments

and theoretical error analysis performed by Bruno Codenotti at the IEI, Pisa, Italy), so that these algorithms are efficient for polynomial division over real and complex fields of constants in both parallel and sequential settings. However, for theoretical purposes and for computing over finite fields, we need to obtain the exact values of $\mathbb{A}^{-1}$ and $\mathbb{A}^{-1}\mathbf{u}$.

We cannot directly apply the same algorithm to compute $\mathbb{A}^{-1}$ and $\mathbb{A}^{-1}\mathbf{u}$ because equation (6) does not hold for $\epsilon = 0$ but the transition from $\mathbb{A}_\epsilon^{-1}$ to $\mathbb{A}^{-1}$ can be performed using the interpolation techniques due to Bini [8]. Indeed, recall that $\mathbb{A}_\epsilon^{-1}$ is filled with some rational functions in ϵ; specifically,

$$\mathbb{A}_\epsilon^{-1} = (\text{adj } \mathbb{A}_\epsilon)/\det \mathbb{A}_\epsilon.$$

Observe that $\det \mathbb{A}_\epsilon$ and the entries of the matrix $\text{adj } \mathbb{A}_\epsilon$ are polynomials in ϵ of degree $s(s-1)$ at most. Suppose that at least $s^2 - s + 1$ distinct values ϵ_h, $h = 0, 1, \ldots, s(s-1)$, such that $\det \mathbb{A}_{\epsilon_h} \neq 0$ for all h can be chosen among the elements of the given field of constants F. Then we may easily compute $\mathbb{A}_\epsilon^{-1}$, $\det \mathbb{A}_\epsilon$ [which is equal to the product of the diagonal entries of the diagonal matrix $\mathbb{D}_\epsilon(\mathbb{A}_\epsilon)$, see equation (6)] and then $\text{adj}(\mathbb{A})_\epsilon = \mathbb{A}_\epsilon^{-1} \det \mathbb{A}_\epsilon$ at the points $\epsilon = \epsilon_h$ for $h = 0, 1, \ldots, s^2 - s$. Then we may interpolate to $\text{adj } \mathbb{A}_\epsilon$ at those $s^2 - s + 1$ points and compute all coefficients of $\text{adj } \mathbb{A}_\epsilon$ because the degree of $\text{adj } \mathbb{A}_\epsilon$ in ϵ is $s^2 - s$ at most. Of those coefficients we actually need to know only the ϵ-free terms of the entries of $\text{adj } \mathbb{A}_\epsilon$, which form the matrix $\text{adj } \mathbb{A}$ where $\mathbb{A} = \mathbb{A}_0$ is the given triangular Toeplitz matrix from the algebra τ_0. Since $\det \mathbb{A} = a_1^s$, where a_1 is the diagonal entry of $\mathbb{A}$, we can immediately evaluate $\mathbb{A}^{-1} = \text{adj } \mathbb{A}/\det \mathbb{A}$ and then $\mathbb{A}^{-1}\mathbf{u}$, see the details in Ref. [5] and in Section 4 below.

The total cost of such an algorithm for the evaluation of $\mathbb{A}^{-1}\mathbf{u}$ is $7 \log s + 7$ parallel steps and $2.5s(w+1)$ processors, see Ref. [5], provided that the field of constants (i) supports FFT at s points and (ii) contains more than $2s^2 - 2s$ distinct elements (recall that $\det \mathbb{A}_\epsilon$ is a polynomial in ϵ of degree $s^2 - s$ at most, so $\det \mathbb{A}_\epsilon$ may turn into 0 at most at $s^2 - s$ distinct values of ϵ).

This immediately implies that Problem 1.1 can be solved using $8 + \log w + 7 \log (m - n + 1)$ parallel steps and at most $(w+1)\max\{0.5(2n-w), 2.5(m-n+1)\}$ processors provided that the field of constants satisfies the above assumptions (i) and (ii).

In the following sections we will extend the latter approach to cases where we do not require that assumptions (i) and (ii) hold and we will show that such an extension implies only a minor deterioration of the upper bounds on the time complexity. Note that we should not simply operate in the extended field of F containing the sth roots of unity unless we agree to increase the number of parallel steps to $O(\log^2 m)$.

3. A GENERALIZED ALGORITHM FOR APPROXIMATION

In this section we will generalize the choice [equation (5)] of matrices $\mathbb{H}_\epsilon$.

Let $\lambda_1, \lambda_2, \ldots, \lambda_s$ be distinct elements of a given field of constants F. Consider the polynomial in λ,

$$P_s^{(c)}(\lambda) = \prod_{i=1}^{s} (\lambda - \lambda_i \epsilon) = \lambda^s + \sum_{j=0}^{s-1} c_j \epsilon^{s-j} \lambda^j, \tag{7}$$

whose coefficients equal $1, \epsilon c_{s-1}, \epsilon^2 c_{s-2}, \ldots, \epsilon^s c_0$, respectively, where $c_{s-1}, c_{s-2}, \ldots, c_0$ are symmetric functions in $\lambda_1, \lambda_2, \ldots, \lambda_s$. Define the matrix $\mathbb{H}_\epsilon$ as the $s \times s$ companion matrix whose eigenvalues are equal to $\lambda_1 \epsilon, \lambda_2 \epsilon, \ldots, \lambda_s \epsilon$, so that equation (7) defines the characteristic polynomial of $\mathbb{H}_\epsilon$:

$$\mathbb{H}_\epsilon = (h_{ij}^{(c)}), \quad h_{ij}^{(c)} = \begin{cases} 1 & \text{if } j = i+1, \\ -\epsilon^{s-j} c_j & \text{if } i = s, \quad j = 0, 1, \ldots, s-1, \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

This generalizes the approach of the previous section where we defined $\mathbb{H}_\epsilon$ by equation (8) in the special case, i.e. where $c_j = 0$ for $j \neq 0$; $c_0 = -1$, and $\lambda_1, \lambda_2, \ldots, \lambda_s$ were the sth

roots of unity, cf. equation (8) with equation (5). For example, if $s = 4$, then in the general case equation (8) defines the matrix

$$\mathbb{H}_\epsilon = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\epsilon^4 c_0 & -\epsilon^3 c_1 & -\epsilon^3 c_2 & -\epsilon c_3 \end{bmatrix};$$

and in the particular case [equation (5)]

$$\mathbb{H}_\epsilon = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \epsilon^4 & 0 & 0 & 0 \end{bmatrix}.$$

We will preserve the notation $\mathbb{H}_\epsilon$ for the matrix [equation (8)] and the notation τ_ϵ for the algebra of $s \times s$ matrices generated by that matrix. The matrix $\mathbb{H}_\epsilon$ and the algebra τ_ϵ may now depend on the parameters $\lambda_1, \lambda_2, \ldots, \lambda_s$. However, τ_0 remains the algebra of all $s \times s$ upper-triangular Toeplitz matrices, all of whose eigenvalues coincide with each other. Furthermore, the matrices of τ_0 can be approximated by the matrices of τ_ϵ with any prescribed precision if $|\epsilon|$ is sufficiently small.

Since all of $\lambda_1, \lambda_2, \ldots, \lambda_s$ are distinct, the matrix $\mathbb{H}_\epsilon$ has s distinct eigenvalues unless $\epsilon = 0$. This implies the following relations for all $\epsilon \neq 0$:

$$\left.\begin{aligned} &\mathbb{H}_\epsilon = \mathbb{Q}_\epsilon \tilde{\mathbb{D}}_\epsilon \mathbb{Q}_\epsilon^{-1}, \quad \tilde{\mathbb{D}}_\epsilon = \mathbb{Q}_\epsilon^{-1} \mathbb{H}_\epsilon \mathbb{Q}_\epsilon, \\ &\tilde{\mathbb{D}}_\epsilon = \epsilon \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_s) \\ &\mathbb{Q}_\epsilon = (q_{ij}^\epsilon) \quad \text{and} \quad q_{ij}^\epsilon = (\lambda_j \epsilon)^{i-1}, \quad i, j = 1, 2, \ldots, s. \end{aligned}\right\} \tag{9}$$

Here the column-vectors of $\mathbb{Q}_\epsilon$ are the eigenvectors of $\mathbb{H}_\epsilon$, cf. Ref. [13]. Relations (9) immediately imply that

$$\mathbb{H}_\epsilon^i = \mathbb{Q}_\epsilon \tilde{\mathbb{D}}_\epsilon^i \mathbb{Q}_\epsilon^{-1} \quad \text{for} \quad i = 0, 1, \ldots$$

The latter equations enable us to represent an arbitrary matrix $\mathbb{A}_\epsilon$ from the algebra τ_ϵ (generated by $\mathbb{H}_\epsilon$) as follows:

$$\mathbb{A}_\epsilon = \sum_{j=0}^{s-1} a_{j+1} \mathbb{H}_\epsilon^j = \mathbb{Q}_\epsilon \left(\sum_{j=0}^{s-1} a_{j+1} \tilde{\mathbb{D}}_\epsilon^j \right) \mathbb{Q}_\epsilon^{-1}. \tag{10}$$

Combining equations (9) and (10) gives

$$\left.\begin{aligned} &\mathbb{A}_\epsilon = \sum_{j=0}^{s-1} a_{j+1} \mathbb{H}_\epsilon^j = \mathbb{Q}_\epsilon \mathbb{D}_\epsilon(\mathbb{A}_\epsilon) \mathbb{Q}_\epsilon^{-1}, \\ &\mathbb{D}_\epsilon(\mathbb{A}_\epsilon) = \operatorname{diag}(d_1^{(\epsilon)}, d_2^{(\epsilon)}, \ldots, d_s^{(\epsilon)}), \quad \text{and} \\ &d_i^{(\epsilon)} = \sum_{j=0}^{s-1} a_{j+1} (\lambda_i \epsilon)^j, \quad i = 1, 2, \ldots, s. \end{aligned}\right\} \tag{11}$$

Next, recall the structure of the matrix $\mathbb{H}_\epsilon$ [see equation (8)] and note that the coefficients a_{j+1} in equation (10) for $j = 0, 1, \ldots, s-1$ are exactly the ϵ-free entries of the first row of $\mathbb{A}_\epsilon$, so that equation (10) enables us to reconstruct the matrix $\mathbb{A}_\epsilon$ from its first row vector,

$$\mathbf{a}^T = (a_j). \tag{12}$$

Now we will rewrite equations (11) using the product of the vector (12) and the Vandermonde matrix:

$$\mathbb{Q}_\epsilon = (q_{ij}^{(\epsilon)}), \quad q_{ij}^{(\epsilon)} = (\lambda_j \epsilon)^{i-1}, \quad i, j = 1, 2, \ldots, s, \quad \text{see equations (9).}$$

In this way we arrive at the following equations [cf. equations (6), (11) and (12)]:

$$\left.\begin{aligned} &\mathbb{D}_\epsilon(\mathbb{A}_\epsilon) = \operatorname{diag}(\mathbf{a}^T\mathbb{Q}_\epsilon), \\ &\mathbb{A}_\epsilon = \mathbb{Q}_\epsilon\mathbb{D}_\epsilon(\mathbb{A}_\epsilon)\mathbb{Q}_\epsilon^{-1} \quad \text{and} \quad \mathbb{A}_\epsilon^{-1} = \mathbb{Q}_\epsilon\mathbb{D}_\epsilon^{-1}(\mathbb{A}_\epsilon)\mathbb{Q}_\epsilon^{-1}. \end{aligned}\right\} \tag{13}$$

Moreover, from equations (9) we have

$$\left.\begin{aligned} &\mathbb{D}_\epsilon = \mathbb{D}_\epsilon\mathbb{Q}, \ \mathbb{D}_\epsilon = \operatorname{diag}(1, \epsilon, \epsilon^2, \ldots, \epsilon^{s-1}) \\ &\text{and} \\ &\mathbb{Q} = (q_{ij}) \quad \text{and} \quad q_{ij} = (\lambda_j)^{i-1}, \quad i,j = 1, 2, \ldots, s. \end{aligned}\right\} \tag{14}$$

Relations (13) and (14) suggest the following algorithm for parallel evaluation of $\mathbb{A}_\epsilon^{-1}\mathbf{u}$ where $\mathbb{A}_\epsilon$ is an arbitrary matrix from τ_ϵ and $\mathbf{u}$ is an arbitrary vector:

Algorithm 3.1

Stage	Parallel steps	Processors
(1) Compute $\mathbf{a}^T\mathbb{D}_\epsilon\mathbb{Q}$[i.e. $\mathbb{D}_\epsilon(\mathbb{A}_\epsilon)$] and $\mathbb{Q}^{-1}\mathbb{D}_\epsilon^{-1}\mathbf{u}$ (i.e. $\mathbb{Q}_\epsilon^{-1}\mathbf{u}$).	$2 + \log s$	$2s^2$
(2) Compute $\mathbb{D}_\epsilon^{-1}(\mathbb{A}_\epsilon)(\mathbb{Q}^{-1}\mathbb{D}_\epsilon^{-1}\mathbf{u})$.	1	s
(3) Compute $\mathbb{A}_\epsilon^{-1}\mathbf{u} = \mathbb{D}_\epsilon\mathbb{Q}(\mathbb{D}_\epsilon^{-1}(\mathbb{A}_\epsilon)\mathbb{Q}^{-1}\mathbb{D}_\epsilon\mathbf{u})$.	$2 + \log s$	s^2

The total computational cost of Algorithm 3.1 is therefore $2 \log s + 5$ parallel steps and $2s^2$ processors.

Remark 3.1

The above estimates do not include the cost of the evaluation of $\mathbb{Q}, \mathbb{Q}^{-1}$ and $\mathbb{Q}_\epsilon$, see equations (9) and (14); the matrices $\mathbb{Q}$ and $\mathbb{Q}^{-1}$ do not depend on the choice of $\mathbb{A}_\epsilon$ and can be precomputed once and for all. So we can think of $\mathbb{Q}$ and $\mathbb{Q}^{-1}$ as given constant matrices. The matrix $\mathbb{D}_\epsilon$ depends only on ϵ and can be considered as a given constant matrix; therefore the cost of the computation of $\epsilon^2, \epsilon^3, \ldots, \epsilon^{s-1}$ is not included in the above estimates. If we consider ϵ as an input variable (this situation occurs when, for example, we have to approximate $\mathbb{A}^{-1}$ using different precisions ϵ, $\epsilon/2$, $\epsilon/4$), we must introduce a new step in the above algorithm (say, step O), in which $\epsilon^2, \epsilon^2, \ldots, \epsilon^{s-1}$ are computed using $\log(s-1)$ steps and $\lceil (s-1)/2 \rfloor$ processors. In this case the total computational cost would be $3 \log s + 5$ steps and $2s^2$ processors. In the special case where $\mathbb{H}_\epsilon$ is defined by equation (5), the matrix $\mathbb{Q}$ turns into the matrix of the discrete Fourier transform and this enables us to eliminate the preprocessing phase and to reduce the number of processors at Stages 1 and 3.

Remark 3.2

Algorithm 3.1 can be easily extended to an algorithm for inverting the matrix $\mathbb{A}_\epsilon$. In this case the identity matrix substitutes for the vector $\mathbf{u}$, so that the number of processors at stages 2 and 3 increases s times. However, we actually need to know only the first row-vector of the matrix $\mathbb{A}_\epsilon^{-1}$ in order to approximate to $\mathbb{A}^{-1}$ because $\mathbb{A}^{-1}$ is an upper-triangular Toeplitz matrix. Then it is sufficient to substitute the vector $(1, 0, 0, \ldots, 0)^T$ for the vector $\mathbf{u}$, so that s^2 processors are sufficient at Stage 1 and the complexity of the computation at Stages 2 and 3 does not change; then the total number of parallel steps is unchanged and the number of processors is reduced to s^2.

4. THE INTERPOLATION ALGORITHM

Next we will reproduce the interpolation algorithm from Ref. [5]. The algorithm evaluates the vector $\mathbb{A}^{-1}\mathbf{u}$ provided that the vectors $\mathbb{A}^{-1}\mathbf{u}$ can be computed at $N+1$

distinct nonzero points $\epsilon = \epsilon_h$, $h = 0, 1, \ldots, N$, such that

(i) $\det \mathbb{A}_{\epsilon_h} \neq 0$ for all h,

and

(ii) $\det \mathbb{A}_\epsilon$ and all entries of the matrix $\operatorname{adj} \mathbb{A}_\epsilon = \mathbb{A}_\epsilon^{-1} \det \mathbb{A}_\epsilon$ are polynomials in ϵ of degree N at most.

If assumption (ii) holds, then $\det \mathbb{A}_\epsilon$ may turn into 0 at most at N distinct points ϵ, so it is sufficient to have a set of $2N + 1$ distinct nonzero values of ϵ.

Remark 4.1

Reproducing the interpolation algorithm, we will use the notation $\mathbf{v}^T = (v_h)$ for the first row vector of the matrix $\mathbb{V}^{-1}$ where $\mathbb{V} = (\epsilon_h^g)$ is the $(N + 1) \times (N + 1)$ Vandermonde matrix and g and h range from 0 to N. The values v_h do not depend on the choice of the input matrix $\mathbb{A}$, so that these values v_h can be precomputed once and for all. For this reason we will not include the cost of the precomputing of v_h in our estimates for the complexity of the interpolation algorithm, cf. Remark 3.1.

Algorithm 4.1

	Stage	Parallel steps	Processors
(1)	Compute $\mathbf{d}^{(\epsilon)} = \mathbb{Q}_\epsilon^T \mathbf{a}$, $\mathbf{d}^{(\epsilon)} = (d_i^{(\epsilon)})$ [i.e. $\mathbb{D}_\epsilon(\mathbb{A}_\epsilon)$] and $\mathbb{Q}_\epsilon^{-1}\mathbf{u}$ for $2N + 1$ distinct nonzero values of ϵ; choose $N + 1$ of them, $\epsilon_0, \epsilon_1, \ldots, \epsilon_N$, such that $d_i^{(\epsilon)} \neq 0$ if $\epsilon = \epsilon_h$, $h = 0, 1, \ldots, N$, $i = 1, 2, \ldots, s$.	$1 + \log s$	$2(2N + 1)s^2$
(2)	Compute $\mathbb{D}_\epsilon^{-1}(\mathbb{A}_\epsilon)(\mathbb{Q}_\epsilon^{-1}\mathbf{u})$ for $\epsilon = \epsilon_h$, $h = 0, 1, \ldots, N$.	1	$s(N + 1)$
(3)	Compute $\det \mathbb{A} = a_1^s$, $\det \mathbb{A}_\epsilon = \prod_{i=1}^{s} d_i^{(\epsilon)}$ and $\mathbb{A}_\epsilon^{-1}\mathbf{u} = \mathbb{Q}_\epsilon(\mathbb{D}_\epsilon^{-1}(\mathbb{A}_\epsilon)\mathbb{Q}_\epsilon^{-1}\mathbf{u})$ for $\epsilon = \epsilon_h$, $h = 0, 1, 2, \ldots, N$.	$1 + \log s$	$s^2(N + 1) + (N + 2)\lceil s/2 \rceil$
(4)	Compute $c_\epsilon = \det \mathbb{A}_\epsilon \det \mathbb{A}$ for $\epsilon = \epsilon_h$, $h = 0, 1, 2, \ldots, N$.	1	$N + 1$
(5)	Compute $b_h = c_{\epsilon_h} v_h$, $h = 0, 1, 2, \ldots, N$, where the values v_h are defined in Remark 4.1.	1	$N + 1$
(6)	Compute $\mathbb{A}^{-1}\mathbf{u} = \sum_{h=0}^{N} b_h(\mathbb{A}_{\epsilon_h}^{-1}\mathbf{u})$.	$1 + \log(N + 1)$	$N + 1$

Lemma 4.1

The total computational cost of Algorithm 4.1 for solving a triangular Toeplitz system of s linear equations is at most $2 \log s + \log(N + 1) + 6$ parallel steps and $2(N + 1)s^2$ processors provided that N is the maximum degree in ϵ of all entries of $\operatorname{adj} \mathbb{A}_\epsilon$ and of $\det \mathbb{A}_\epsilon$ over all matrices $\mathbb{A}_\epsilon$ of the algebra τ_ϵ and that the field of constants F contains at least $2N + 2$ distinct elements.

Remark 4.2

The preprocessing phase in Algorithm 4.1 consists of computing $\mathbb{Q}$, $\mathbb{Q}^{-1}$, $\mathbb{D}_\epsilon$, $\mathbb{D}_\epsilon\mathbb{Q}$ and $\mathbb{Q}^{-1}\mathbb{D}_\epsilon^{-1}$ for the $2N + 1$ different values of ϵ used at Stage 1. The preprocessing phase can be reduced to the computation of $\mathbb{Q}$, $\mathbb{Q}^{-1}$ and $\mathbb{D}_\epsilon$ for $2N + 1$ different values of ϵ, which means a total of $2s^2 + s(2N + 1)$ precomputed constants, by increasing the computational cost of Algorithm 4.1 by two more steps at Stages 1 and 3.

Remark 4.3

Algorithm 4.1 can be applied in order to reconstruct the matrix $\mathbb{A}^{-1}$ from the values of the first row of $\mathbb{A}_\epsilon^{-1}$ computed for $\epsilon_0, \epsilon_1, \ldots, \epsilon_N$. For this purpose, it is sufficient to substitute the vector $(1, 0, 0, \ldots, 0)^T$ for the vector $\mathbf{u}$ in Algorithm 4.1. This will reduce the number of processors at Stage 1 to $(2N+1)s^2$.

Remark 4.4

In the special case where $\mathbb{H}_\epsilon$ is defined by equation (5), the number of processors at Stages 1 and 3 can be reduced, cf. Remark 3.1.

5. HOW MANY NODES OF INTERPOLATION ARE NEEDED?

In this section we will prove the following result.

Lemma 5.1

Let $\mathbb{A}_\epsilon$ be an $s \times s$ matrix of the algebra τ_ϵ generated by the matrix $\mathbb{H}_\epsilon$, defined by equation (8). Then the degrees in ϵ of $\det \mathbb{A}_\epsilon$ and of all entries of the matrix $\operatorname{adj} \mathbb{A}_\epsilon$ are $s^2 - s$ at most. (Here $\det \mathbb{A}_\epsilon$ and the entries of $\operatorname{adj} \mathbb{A}_\epsilon$ are considered as polynomials in ϵ.) Furthermore, $\det \mathbb{A}_\epsilon$ has degree $s^2 - s$ in ϵ if $\mathbb{A}_\epsilon = \mathbb{H}_\epsilon^{s-1}$, so that the bound $s^2 - s$ is sharp.

Proof. The latter statement of Lemma 5.1 immediately follows because $\det \mathbb{H}_\epsilon^{s-1} = (\det \mathbb{H}_\epsilon)^{s-1}$ and $\det \mathbb{H}_\epsilon = p_n^{(c)}(0)$ has degree s, see equations (7) and (8).

Let us prove that $s^2 - s$ is the upper bound on the degree. It is sufficient to consider the straightforward expansion of $\det \mathbb{A}_\epsilon$ as the sum of $s!$ products of the entries of $\mathbb{A}_\epsilon$ (such that each product has s factors) and to show that for all products their degrees in ϵ do not exceed $s^2 - s$. (Indeed the degrees of the similar products of $s-1$ factors in the similar expansion of the entries of $\operatorname{adj} \mathbb{A}_\epsilon$ cannot be greater than in the case of the expansion of the determinant.) Furthermore it is sufficient to consider the cases where $\mathbb{A}_\epsilon = \mathbb{H}_\epsilon^j$ for $j = 1, 2, \ldots, s-1$ due to equations (11). Since we are interested only in the upper bounds on the degrees of the products of the entries of $\mathbb{H}_\epsilon^j$, we may ignore the coefficients c_j in equation (7) and we will assume that $c_j = -1$ for all j.

Hereafter let all constant polynomials in ϵ (including identical 0) be considered polynomials of degree 0 and let $\mathbb{M}^{(j)} = (m_{ik}^{(j)})$ denote the matrix of the degrees of the entries of $\mathbb{H}_\epsilon^j$. Then Lemma 5.1 will immediately follow from relation (15) where $i, j+1$ and k range from 1 to s:

$$m_{ik}^{(j)} \leqslant j + i - k \quad \text{if} \quad i > s - j, \quad m_{ik}^{(j)} = 0 \quad \text{otherwise.} \tag{15}$$

It remains to verify this relation. Let us assume that relation (15) holds for all $j < J < s$ and let us prove it for $j = J$.

Represent $\mathbb{H}_\epsilon^J$ as the product $\mathbb{H}_\epsilon \mathbb{H}_\epsilon^{J-1} = (\mathbb{E} + \mathbb{L}_\epsilon)\mathbb{H}_\epsilon^{J-1}$. Here we represent the matrix $\mathbb{H}_\epsilon$ as the sum $\mathbb{E} + \mathbb{L}_\epsilon$ where the matrix

$$\mathbb{E} = \begin{bmatrix} 0 & 1 \ldots 0 \\ \vdots & \vdots \ddots \vdots \\ \vdots & \vdots \ldots 1 \\ 0 & 0 \ldots 0 \end{bmatrix}$$

has been obtained from $\mathbb{H}_\epsilon$ by replacing the last row of $\mathbb{H}_\epsilon$ by the null row vector. Then the premultiplication of $\mathbb{H}_\epsilon^{J-1}$ by $\mathbb{E}$ amounts to shifting the row vectors of $\mathbb{H}_\epsilon^{J-1}$ up one place. This does not violate relation (15) which holds for the matrix $\mathbb{H}_\epsilon^{J-1}$ by the inductive assumption. Indeed j increases by 1; i decrease by 1, so that both sides of inequality (15) remain invariant.

The premultiplication of $\mathbb{H}_\epsilon^{j-1}$ by

$$\mathbb{L}_\epsilon = \begin{bmatrix} 0 & 0 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & 0 \\ \epsilon^s & \epsilon^{s-1} & \cdot & \epsilon \end{bmatrix}$$

amounts to multiplication of the last row of $\mathbb{H}_\epsilon^{j-1}$ by ϵ and to filling all other rows with zeros. This also preserves inequality (15) since both the degree of the entries of the last row and the value of j increase by 1. Summarizing, we have verified relation (15) for the entries of the matrix $\mathbb{H}_\epsilon^j$ provided that it holds for the entries of $\mathbb{H}_\epsilon^{j-1}$. This proves relation (15) for all j, since these relations are trivially satisfied for $j = 0$ and $j = 1$. Thus the proof of Lemma 5.1 has been completed.

Lemma 5.1 implies that we may choose the value $s^2 - s$ for N in our estimates of the previous section.

Remark 5.1

As follows from equation (11), for every nonsingular band upper-triangular Toeplitz matrix $\mathbb{A}$ with the bandwidth w, the matrix $\mathbb{A}_\epsilon$ can be represented as the linear combination of $\mathbb{I}, \mathbb{H}_\epsilon, \mathbb{H}_\epsilon^2, \ldots, \mathbb{H}_\epsilon^w$. Then relation (15) implies that the upper bound on N can be reduced to ws. Note that we may always choose $N = ws$, which will give $N = s(s-1) = s^2 - s$ for $w = s - 1$, i.e. for nonband triangular matrices.

6. FINAL ESTIMATES

Initially, we will substitute $N = sw$ into Lemma 4.1 and obtain the following result.

Theorem 6.1

Algorithm 4.1 involves at most $2\log s + \log(ws + 1) + 6$ parallel steps and at most $2(2ws + 1)s^2$ processors in order to solve a triangular Toeplitz system of s linear equations over an arbitrary field of constants F that contains at least $2ws + 2$ distinct elements, i.e. such that

$$|F| > 2N + 1, \tag{16}$$

where $|F|$ denotes the number of elements of F, $N = ws$, and $w \leqslant s - 1$ is the bandwidth of the matrix of the system.

Corollary 6.1

Problem 1.1 can be solved involving at most $2\log(m - n + 1) + \log(w + 1) + \log(w(m - n + 1) + 1) + 8$ parallel steps and at most $\max\{2(2w(m - n + 1) + 1)(m - n + 1)^2, (w + 1)(2n - w)/2\}$ processors over every field F of constants of at least cardinality $2(m - n + 1)w + 2$, where w is defined by equation (2).

The assumption that inequality (16) holds may not be satisfied for fields F of small cardinality. However, we may always extend field F to the field $F_{p(t)}[t]$ of polynomials in t over F modulo a monic irreducible polynomial $p(t)$ of any prescribed degree d, see Ref. [14]. We will choose

$$d = \deg p(t) = 1 + \lfloor \log(2N + 2)/\log|F| \rfloor, \tag{17}$$

where $N = sw$, w is the bandwidth of the triangular Toeplitz matrix of our computational problem, $w = s - 1$ if the matrix is not band. Equation (17) assures that

$$|F_{p(t)}[t]| > 2N + 1,$$

so that our algorithms work over $F_{p(t)}[t]$. The estimates for the complexity of the algorithms, see Theorem 6.1 and Corollary 6.1, can be applied if the operations, steps and number of processors are counted for the extension field $F_{p(t)}[t]$. This amounts to using the

Table 1

	Operation	degree of the operands	*Upper bound on the:* number of parallel steps in in the field F	number of processors in the field F
(1)	Addition/subtraction multiplication by a constant.	$d-1$	1	d
(2)	Multiplication.	$d-1$	$1+\log d$	d^2
(3)	Division by a monic polynomial (Problem 1.1 for $m=2d-2$, $n=d$).	$2d-2$ and d	$2+\log(d-1)$	$(d-1)^2\|F\|^{d-2}$
(4)	Inversion of a polynomial modulo a monic polynomial.	$d-1$ and d	$4+\log(d-1)$	$(2d^2-1)\|F\|^{2d-3}$

computational model, where special circuits perform arithmetical operations in $F_{p(t)}[t]$. We have to increase the values of our estimates if we compute in the original field F of constants because every operation in $F_{p(t)}[t]$ amounts to an operation with polynomials over F modulo $p(t)$ and costs more than one operation in F. Table 1 shows the cost of all operations with polynomials over F that we need in order to perform arithmetical operations in $F_{p(t)}[t]$. (The cost is presented in terms of the number of operations in F.)

Comments on Table 1

The estimates of Table 1 have been derived using straightforward algorithms. In particular, for the division of a polynomial $X_m(t)$ of degree $m < 2d-1$ by a monic polynomial $Y_n(t)$ of degree $n=d$ (row 3 of Table 1) we considered the algorithm that computed the coefficients of all of the $|F|^{d-2}$ possible products of $Y_d(t)$ by polynomials $Q_{m-d}(t)$ of degree $m-d$ over F such that the leading coefficients of $X_m(t)$ and of $Q_{m-d}(t)$ coincide. Then the algorithm (i) checked if the $m-d+1$ leading coefficients of $Q_{m-d}(t)Y_d(t)$ coincided with those of $X_m(t)$ and (ii) computed (in one parallel step with d processors at most) the coefficients of $X_m(t)-Q_{m-d}(t)Y_d(t)$ when the desired coincidence took place. For each $Q_{m-d}(t)$, the multiplication $Q_{m-d}(t)Y_d(t)$ cost $1+\log(d-1)$ steps and at most $(d-1)^2$ processors, i.e. $(d-1)^2|F|^{d-2}$ processors for all polynomials $Q_{m-d}(t)$. [Note that we did not need to compute the leading coefficients of the product $Q_{m-d}(t)Y_d(t)$.] Similarly, we inverted a polynomial modulo a monic polynomial (row 4) of Table 1. We applied the straightforward algorithm for the evaluation of the polynomials $G_{d-1}(t)X_{d-1}(t)-H_{d-2}(t)Y_d(t)$ for all of the $|F|^{2d-3}$ possible pairs of polynomials $G_{d-1}(t)$ of degree $d-1$ at most and $H_{d-2}(t)$ of degree $d-2$ at most over F. [Here the polynomial $X_{d-1}(t)$ of degree $d-1$ at most and the monic polynomial $Y_d(t)$ of degree d were assumed given as well as the leading coefficients of $G_{d-1}(t)$ and $H_{d-2}(t)$.] Every such evaluation cost at most $3+\log(d-1)$ parallel steps and at most $2d^2-1$ processors, i.e. $(2d^2-1)|F|^{2d-3}$ processors for all pairs $G_{d-1}(t)$, $H_{d-2}(t)$. One more step with $2d-1$ processors was needed in order to check for which choice of $G_{d-1}(t)$, $H_{d-2}(t)$ the computed polynomial $G_{d-1}(t)X_{d-1}(t)+H_{d-2}(t)Y_d(t)$ turned into the unit constant of F. Then $G_{d-1}(t)$ was selected as the desired inverse of $X_{d-1}(t)$ modulo $Y_d(t)$.

In the following we will comment on the ways of reducing the number of processors required for the operations of rows 3 and 4.

The arithmetical operations in the field $F_{p(t)}[t]$ are easily reduced to the operations of Table 1 with polynomials over F; namely, the cost of addition/subtraction in $F_{p(t)}[t]$ is shown in row 1 of Table 1; the cost of multiplication in $F_{p(t)}[t]$ is equal to the sum of the costs shown in rows 2 and 3 of Table 1; the cost of division is equal to the sum of the costs shown in rows 2–4 of Table 1.

Now we will examine Algorithms 3.1 and 4.1 (assuming that we work in $F_{p(t)}[t]$) and use Table 1 in order to estimate the computational cost of the operations which involve ε_h (which are now considered elements of $F_{p(t)}[t]$ while the values independent of ϵ still

remain in F). Then we may rewrite our estimates of Section 4 as follows [here d is defined by equation (17), $N = sw$, see Remark 5.1]:

Algorithm 4.1

	Stage	Parallel steps	Processors
(1)	Compute $\mathbf{d}^{(\epsilon)} = \mathbb{Q}_\epsilon^T \mathbf{a}$, $\mathbb{Q}_\epsilon^{-1}\mathbf{u}$ for $2N+1$ distinct nonzero values of ϵ.	$1 + \log s$	$2(2N+1)s^2 d$
(2)	Compute $\mathbb{D}_\epsilon^{-1}(\mathbb{A}_\epsilon)(\mathbb{Q}_\epsilon^{-1}\mathbf{u})$ for $\epsilon = \epsilon_h$, $h = 0, 1, \ldots, N$.	$7 + \log d + 2\log(d-1)$	$(2d^2-1)\lvert F\rvert^{2d-3}(N+1)s$
(3)	Compute $\det \mathbb{A} = a_1^s$, $\det \mathbb{A}_\epsilon = \prod_{i=1}^{s} d_i^{(\epsilon)}$ and $\mathbb{A}_\epsilon^{-1}\mathbf{u} = \mathbb{Q}_\epsilon(\mathbb{D}_\epsilon^{-1}(\mathbb{A}_\epsilon)\mathbb{Q}_\epsilon^{-1}\mathbf{u})$ for $\epsilon = \epsilon_h$, $h = 0, 1, \ldots, N$.	$[3 + \log d + \log(d-1)]\log s$	$(d-1)^2\lvert F\rvert^{d-2}(N+1)(s^2 + \lceil s/2\rceil) + \lceil s/2\rceil$
(4)	Compute $\mathbb{C}_\epsilon = \det \mathbb{A}_\epsilon / \det \mathbb{A}$ for $\epsilon = \epsilon_h$, $h = 0, 1, 2, \ldots, N$.	1	$(N+1)d$
(5)	Compute $b_h = \mathbb{C}_{\epsilon_h} v_h$, $h = 0, 1, 2, \ldots, N$.	$3 + \log d + \log(d-1)$	$(d-1)^2\lvert F\rvert^{d-2}(N+1)$
(6)	Compute $\mathbb{A}^{-1}\mathbf{u} = \sum_{h=0}^{N} b_h(\mathbb{A}_{\epsilon_h}^{-1}\mathbf{u})$.	$3 + \log d + \log(d-1) + \log(N+1)$	$(d-1)^2\lvert F\rvert^{d-2}(N+1)$

Next, taking into account that $|F|^{d-2} < 2(N+1)$, $|F|^{2d-3} < 4(N+1)^2$, see equation (17), we will extend Theorem 6.1 and Corollary 6.1; then we will comment on how the algorithms can be further improved.

Theorem 6.2

Algorithm 4.1 involves at most $14 + [4 + \log(d-1) + \log d]\log s + \log(N+1) + 3\log d + 4\log(d-1)$ parallel steps and at most $4(2d^2-1)(N+1)^3 s$ processors in order to solve an arbitrary triangular Toeplitz system of s linear equations over an arbitrary field of constants F where $N = s(s-1)$ and d is defined by equation (17). If, in addition, the matrix of the system of equations is band with bandwidth w, then it is possible to choose $N = ws$.

Corollary 6.2

Problem 1.1 can be solved over an arbitrary field of constants F involving at most $16 + [4 + \log(d-1) + \log d]\log(m-n+1) + \log(w+1) + \log[(m-n+1)w+1] + 3\log d + 4\log(d-1)$ parallel steps and at most $\max\{4(2d^2-1)[(m-n+1)w+1]^3(m-n+1), (w+1)(2n-w)/2\}$ processors, where w is defined by equation (2) and $d = 1 + \log[2(m-n+1)w+2]/\log|F|$, cf. equation (17).

Let us simplify the estimates of Theorems 6.1 and 6.2 and Corollaries 6.1 and 6.2 using O-notation and assuming that $s, m, n \to \infty$.

Theorem 6.3

The solution of a triangular Toeplitz system of s linear equations can be evaluated over a field F of constants involving $O(\log s)$ parallel steps and $O(s^4)$ processors if $|F| > 2s^2 - 2s + 1$, and $O(\log s \log\log s)$ parallel steps and $O(s^7 \log^2 s)$ processors over an arbitrary field. If in addition the system of equations is band with bandwidth w then the number of processors can be decreased to $O(s^3 w)$, provided that the cardinality of the field F exceeds $2sw + 1$, and to $O(s^4 w^3 \log^2 s)$ otherwise.

Corollary 6.3

Problem 1.1 can be solved over an arbitrary field of constants F involving

$O(\log(m-n+1)\log\log(m-n+1))$ parallel steps and $O((m-n+1)^4 w^3 \log^2(m-n+1) + (2n-w)w$ processors, where $w = \min\{m-n, n\}$. If the cardinality of F is at least $2(m-n+1)w+2$, then the problem can be solved involving $O(\log(m-n+1))$ parallel steps and $O(w(m-n)^3 + w(2n-w))$ processors.

The above estimates for the numbers of processors can be substantially improved if we use more efficient algorithms than the exhaustive search applied in rows 3 and 4 of Table 1. In particular, the division by the monic polynomial in row 3 can be performed by the algorithm of Corollaries 6.1, 6.2 and 6.3, so that $O(\log d \log\log d)$ parallel steps and $O(d^7 \log^2 d)$ processors always suffice while $O(\log d)$ parallel steps and $O(d^4)$ processors suffice if $|F| < 2(d-1)d+2$; recall that $d = O(\log s)$, see equation (17). The inversion of a polynomial modulo a monic polynomial (see row 4 of Table 1) can be reduced to computing the greatest common divisor of two polynomials. The latter problem can be solved using $O(\log^2 d)$ parallel steps and $d^{O(1)}$ processors, provided that the degrees of the two given polynomials are not greater than d, see Ref. [15]. Note that we need to invert polynomials in ϵ only at Stage 2 of Algorithm 3.1, so that the number of parallel steps involved in the other operations of Algorithms 3.1 and 4.1 will clearly dominate over the total cost of polynomial inversion since $d = O(\log s)$.

Summarizing, we obtain the following result.

Theorem 6.4

A band triangular Toeplitz system of s linear equations with the bandwidth w can be solved over an arbitrary field of constants involving $O(\log s \log\log s \log\log\log s)$ parallel steps and $O(ws^3 \log^7 s(\log\log s)^2)$ processors. Problem 1.1 can be solved over an arbitrary field of constants involving $O(\log(m-n+1)\log\log(m-n+1)\log\log\log(m-n+1))$ parallel steps and $O(w(m-n+1)^3 \log^7(m-n+1)[\log\log(m-n+1)]^2 + w(2n-w))$ processors where $w = \min\{m-n, n\}$.

Remark 6.1

At all stages of Algorithms 3.1 and 4.1, except for Stage 3 of Algorithm 4.1 [where in the computation of $\det \mathbb{A}_\epsilon$ it is required to multiply s polynomials in t modulo $p(t)$], the number of parallel steps can be kept below $O(\log s)$ (at the price of a moderate increase in the number of processors). Therefore if s polynomials modulo a polynomial of degree $O(\log s)$ can be multiplied in $O(\log s)$ parallel steps over an *arbitrary* field of constants F, then Problem 1.1 can be solved in $O(\log(m-n+1))$ parallel steps. (Both problems can be solved in that number of steps over the fields that support FFT, see Refs [5] and [4].)

Remark 6.2

Following the line of Section 4 of Ref. [1], we can easily extend our results to the solution of a system of linear equations whose matrices belong to one of the two following classes: (i) band Toeplitz matrices, (ii) an algebra generated by a given matrix $\mathbb{A}$ that can be transformed by a similarity transformation to the Jordan canonical form over the given field of constants F or over its small-degree extension field.

Remark 6.3

Our algorithms directly extend the approach of Bini [5, 6]. In Refs [7, 8] the same estimates [$O(\log s \log\log s)$ steps and a polynomial number of processors for the inversion of triangular Toeplitz matrices] were yielded by different methods. It is interesting that all the papers (both ours and Eberly's) rely on the inversion of a fixed Vandermonde matrix and on the interpolation techniques in the spirit of Refs [9, 5]. Eberly's paper [8] ends with stating the problem of fast parallel inversion of an $s \times s$ circulant matrix over the fields that do not support FFT. Let us demonstrate how the techniques of this paper enable

us to solve that problem over the field $F = GF(2)$ of integers modulo 2 provided that s is a power of 2. Indeed, in that case $\lambda^s + 1 = (\lambda + 1)^s$ over F. Let

$$p_s^{(\epsilon)}(\lambda) = \prod_{i=0}^{s-1} [\lambda + 1 + \epsilon p_i(\epsilon)] = \lambda^s + 1 + \epsilon \sum_{j=0}^{s-1} c_j^{(\epsilon)} \lambda^j,$$

where $p_i(\epsilon)$ for $i = 0, 1, \ldots, s-1$ are s distinct polynomials in ϵ over F of degree $\log s$, at most, cf. equation (7). Then $p_s^{(0)}(\lambda) = \lambda^s + 1$ over F. Let the companion matrix of $p_s^{(\epsilon)}(\lambda)$ be designated $\mathbb{H}_\epsilon$,

$$\mathbb{H}_\epsilon = (h_{ij}^{(\epsilon)}), \quad h_{ij}^{(\epsilon)} = \begin{cases} 1 \quad \text{if} \quad j = i + 1, \\ 1 + \epsilon c_0^{(\epsilon)} \quad \text{if} \quad i = s - 1, \quad j = 0, \\ \epsilon c_j^{(\epsilon)} \quad \text{if} \quad i = s - 1, \quad j = 1, 2, \ldots, s - 1, \\ 0 \text{ otherwise}, \end{cases}$$

cf. equation (8). Then the algebra of all $s \times s$ circulant matrices over F is generated by the matrix $\mathbb{H}_0$, so that we may apply the approach of this paper in order to invert an arbitrary $s \times s$ circulant matrix over F involving $O(\log s \log\log s)$ parallel steps and a polynomial number of processors. (Actually, we may apply all of our estimates of this section just increasing N by the factor $\log s$.)

Acknowledgements—This work was supported by the GNIM of the CNR and NSF Grants MCS 8203232 and DCR 8507573.

REFERENCES

1. A. Borodin and I. Munro, *The Computational Complexity of Some Algebraic and Numeric Problems*. Elsevier, New York (1975).
2. D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Vol. 2. Addison-Wesley, Reading, Mass. (1981).
3. D. Bini and V. Pan, A logarithmic Boolean time algorithm for polynomial division. Technical report TR 86-1, Comput. Sci. Dept, SUNY, Albany, N.Y. (1986). *Inform. Process. Lett.* (in press).
4. J. Reif, Logarithmic depth circuits for algebraic functions. *Proc. 24th A. Symp. FOCS*, Tucson, Ariz., pp. 138–145 (1983). Also *SIAM Jl Comput.* **15**(1), 231–242 (1986).
5. D. Bini, Parallel solution of certain Toeplitz linear systems. *SIAM Jl Comput.* **13**, 268–276 (1984).
6. D. Bini, Parallel solutions of certain Toeplitz linear systems. Technical Report B82-04, IEI del CNR, Pisa, Italy (1982).
7. D. Bini and V. Pan, Fast parallel polynomial division via reduction to triangular Toeplitz matrix inversion and to polynomial inversion modulo a power. *Inform. Process. Lett.* **21**, 79–81 (1985).
8. W. Eberly, Very fast parallel matrix and polynomial arithmetic. *Proc. 25th A. IEEE Symp. FOCS*, Singer Island, Fla, pp. 21–30 (1984).
9. D. Bini, Relations between exact and approximate bilinear algorithms, applications. *Calcolo* **17**, 87–97 (1980).
10. D. Coppersmith and S. Winograd, On the asymptotic complexity of matrix multiplication. *SIAM Jl Comput.* **11**, 472–492 (1982).
11. V. Pan, How can we speed up matrix multiplication? *SIAM Rev.* **26**, 393–415 (1984).
12. A. Schönhage, Partial and total matrix multiplications. *SIAM Jl Comput* **10**, 433–455 (1981).
13. J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*. Springer-Verlag, Berlin (1980).
14. E. Berlekamp, *Algebraic Coding Theory*. McGraw-Hill, New York (1968).
15. A. Borodin, J. von zur Gathen and J. Hopcroft, Fast parallel matrix and GCD computations. *Proc. 23rd A. Symp. FOCS*, Chicago, Ill., pp. 65–71 (1982). Also *Inform. Control* **53**(3), 241–256 (1982).