Theoretical Computer Science 54 (1987) 65–85
North-Holland

COMPLEXITY OF PARALLEL MATRIX COMPUTATIONS * **

Victor PAN
Computer Science Department, State University of New York at Albany, Albany, NY12222, U.S.A.

Abstract. We estimate parallel complexity of several matrix computations under both Boolean and arithmetic machine models using deterministic and probabilistic approaches. Those computations include the evaluation of the inverse, the determinant, and the characteristic polynomial of a matrix. Recently, processor efficiency of the previous parallel algorithms for numerical matrix inversion has been substantially improved in (Pan and Reif, 1987), reaching optimum estimates up to within a logarithmic factor; that work, however, applies neither to the evaluation of the determinant and the characteristic polynomial nor to exact matrix inversion nor to the numerical inversion of ill-conditioned matrices. We present four new approaches to the solution of those latter problems (having several applications to combinatorial computations) in order to extend the suboptimum time and processor bounds of (Pan and Reif, 1987) to the case of computing the inverse, determinant, and characteristic polynomial of an arbitrary integer input matrix. In addition, processor efficient algorithms using polylogarithmic parallel time are devised for some other matrix computations, such as triangular and QR-factorizations of a matrix and its reduction to Hessenberg form.

Key words. Parallel algorithms, computational complexity, matrix computations.

1. Introduction

Matrix computations are performed many times every day in scientific and engineering computational practice. Such computations include, in particular,

(i) solving a linear system of equations;
(ii) inversion of a given matrix;
(iii) evaluation of its determinant;
(iv) evaluation of the coefficients of its characteristic polynomial;
(v) its QR-factorization;
(vi) its LU-factorization;
(vii) its reduction to the upper Hessenberg form.

The solution of those problems is also theoretically important; it can be applied in several other algebraic and combinatorial computations, see [1, 3, 4, 10, 11, 16, 19, 20]. Here are two citations from the two recent articles surveying the known parallel

* Supported by NSF Grant DCR-8507573.

** The results of this paper have been announced in the Journal of Abstracts of the American Mathematical Society [22]; some of them have been presented at the Sixth Conference on Foundations of Software Technology and Theoretical Computer Science, New Delhi, India, 1985.

algorithms for algebraic and combinatorial problems. "One interesting phenomenon occurs: all the problems eventually reduce to solving systems of linear equations. Hence the paramount role of the latter problem," [30, p. 803]. "While the motivation of these algorithms may come from different branches of mathematics, they eventually boil down to linear algebra, most often to matrix inversion," [16, p. 394].

Our objective in this paper is estimating the complexity of parallel matrix computations. Regarding the importance of the subject, it is rather surprising that the available parallel algorithms for those computational problems can be substantially improved, in particular, in their processor efficiency. Let us next outline the previous results and our improvements, postponing the formal definitions.

Until 1985 the known methods for problems (i)-(iv) required $\log^2 n$ parallel arithmetic steps and $\sqrt{n}\, P_A(*, n)$ processors, see [7, 27]. Here and hereafter we consider the asymptotic estimates for the numbers of parallel steps and processors defined up to within constant factors; $P_A(*, n)$ denotes the minimum number of processors that support $n \times n$ matrix multiplication performed in $\log n$ parallel arithmetic steps, $P_A(*, n) = o(n^{2.376})$, see [6, 25]. Those methods of 1985 were not processor efficient; they involved $\sqrt{n}\, P_A(*, n)$ processors exceeding the record sequential time bound roughly by a factor of $\sqrt{n}$. Under the sequential model, problems (i)-(v) can be reduced to matrix multiplication; the converse is also true for problems (ii)-(iv), see [2, 13, 20], so $P_A(*, n) \log n$ is a lower bound on the sequential time for those problems and $P_A(*, n)/\log n$ is a lower bound on the number of processors supporting the time bound $O(\log^2 n)$. In [9] the processor bound $\sqrt{n}\, P_A(*, n)$ was slightly improved, but still has not approached to the known sequential time bound.

In [25] the processor bound has been decreased to the desired value $P_A(*, n)$ (preserving the arithmetic parallel time bound $\log^2 n$) in the case of numerical inversion (with a very high output precision) of a well-conditioned or strongly diagonally dominant input matrix (see formal definitions in Section 5). That result (giving *optimum estimates* within at most a logarithmic factor) applies to a large and practically important class of input instances to problems (i) and (ii), but not to the case of ill-conditioned matrices that are not strongly diagonally dominant. Furthermore, the method gives neither any good approximation to the determinant of a matrix nor the exact value of its inverse, even if that matrix is filled with integers and well-conditioned. The exact solution to problems (i)-(iii) for arbitrary integer matrices was required in particular in several combinatorial algorithms, see [4, 10, 16, 19].

In this paper we will present four new approaches that complement each other and lead to the desired progress. The resulting upper estimates for the arithmetic complexity of problems (i)-(vii) are presented in Table 1, where $T_A(n) = o(n^{2.86})$ is defined by the equations (8)-(10) in Section 3.

The outputs of the algorithms supporting the estimates of Table 1 are computed numerically with very high precision; in the case of problems (i)-(iv), they can be turned into exact solutions via rounding-off provided that the inputs are integers

Table 1.

Problems	Input matrix	Arithmetic steps	Processors	Section
(v)-(vii) ($m = n$)	arbitrary	$O(\log^3 n)$	$T_A(n)$	4
(vi), (vii)	well-conditioned and symmetric positive definite	$O(\log^3 n)$	$P_A(*, n)$	5
(iii), (v)	well-conditioned	$O(\log^3 n)$	$P_A(*, n)$	5
(i)-(iv)	integer	$O(\log^2 n)$	$P_A(*, n)$	6

(or rationals). Technically we apply various reductions of problems (i)-(vii) to each other and to computations with Krylov matrices (Sections 4, 6, 10); we employ the results of [25] and the *variable diagonal techniques* of [21] (Section 6), and (for problems (i)-(iv)) the rational interpolation table (compare [12]) and the Toeplitz matrix algorithm of [24] (Appendix). We bound the precision of computations using modular arithmetic. This enables us to extend the arithmetic estimates of the last row of Table 1 to upper estimates for the Boolean complexity of problems (i)-(iv). Specifically we prove that $O(\log^g d \log^h n)$ Boolean steps, and $b(d)p_A$ processors suffice where arithmetic algorithms involve $O(\log^h n)$ steps, p_A processors. Here d denotes the precision required in order to *represent* the output values, $b(d) = O(d \log d \log\log d)$ denotes the number of Boolean operations required in order to multiply two integers modulo 2^d, and $g \leqslant 2$, $h \leqslant 3$; compare (22) and (28) in Sections 7 and 8. We will also deduce a little more refined bounds on the parallel Boolean complexity of matrix multiplication (Section 7), which might be of some independent interest, and will use those bounds, as well as Schwartz's techniques of randomization and Newton-Hensel's lifting algorithm (Sections 8 and 9) in order to obtain even better upper estimates for the Boolean complexity of problems (i)-(iv), which we will summarize in Table 2 in Section 9.

We will present our results in the following order. In the next section we will formally state problems (i)-(vii) and extend their list by adding one more problem, (viii). In Section 3 we will recall the customary machine model of parallel computations, some definitions, and some known estimates to be used later on. In Sections 4-6 we will estimate the parallel arithmetic complexity of problems (i)-(viii). In Section 4 we will consider problems (v)-(viii) for a general input matrix A. In Section 5 we will work on problems (iii), (v)-(vii) for a well-conditioned input matrix A. In Section 6 we will study problems (i)-(iv) in case of integer matrix A. In Section 7 we will estimate the Boolean circuit complexity of parallel matrix multiplication. In Sections 8 and 9 we will estimate the Boolean circuit complexity of parallel deterministic and probabilistic solution of problems (i)-(iv); we will display those estimates in Table 2 in Section 9. In Section 10 we will state some open problems. Sections 7, 8, and 9 and the Appendix can be read independently of Sections 4, 5, and 6, except for the appliction of Definition 4.4 and Proposition

4.5 in Section 9 and the Appendix, and of Theorem 5.3 and Corollary 5.7 in Section 8.

2. Problems

In this section we will formally define problems (i)-(vii), cited in the Introduction, as well as another related problem. We will assume that A is a given $n \times n$ input matrix, except that in the case of problem (v), (QR-FACTORS), we will assume that A is an $m \times n$ matrix, $m \geq n$.

Hereafter u will denote a given vector, I and O will denote the identity and the null matrices of appropriate sizes; W^{T} denotes the transpose of W; all logarithms are to the base 2; for simplicity we let n be a power of 2, $n = 2^s$.

(i) LINEAR SYSTEM: Output: SINGULAR if A is singular, otherwise compute $A^{-1}u$.

(ii) INVERT: Output: SINGULAR if A is singular, otherwise compute A^{-1}.

(iii) DETERMINANT: Compute $\det A$ (or $|\det A|$).

(iv) CHARACTERISTIC POLYNOMIAL: Compute the coefficient vector $c = [c_n, \ldots, c_1]^{\mathrm{T}}$ of the characteristic polynomial of A,

$$\det(\lambda \mathrm{I} - A) = \lambda^n - c_1 \lambda^{n-1} - \cdots - c_n. \tag{1}$$

(v) QR-FACTORS: Compute an $m \times n$ (orthogonal) matrix Q for $m \geq n$ and a nonsingular upper triangular matrix R such that

$$A = QR \qquad Q^{\mathrm{T}}Q = \mathrm{I}. \tag{2}$$

Output: RANK DEFICIENT if A has no such QR-factors.

(vi) LU-FACTORS (CHOLESKY FACTORS where A is symmetric): Compute two nonsingular matrices; that is, L lower triangular and U upper triangular such that $A = LU$. If there exist no such matrices L and U, output: NO LU-FACTORS. ($U = L^{\mathrm{T}}$ if A is symmetric.)

(vii) HESSENBERG REDUCTION: Compute an orthogonal matrix Q and the matrix $H = [h_{ij}]$ having the upper Hessenberg form such that

$$Q^{\mathrm{T}}Q = \mathrm{I}, \qquad Q^{\mathrm{T}}AQ = H, \qquad h_{ij} = 0 \quad \text{if } i - j > 1.$$

(If A is symmetric, H is tridiagonal.) Hessenberg reduction is a customary means of facilitating the evaluation of the eigenvalues of a matrix, see [11, 26].

(viii) RECURSIVE FACTORS: Compute a sequence of matrices, $A = A_0, A_1, A_2, \ldots, A_s$, such that A_h has size $(n/2^h) \times (n/2^h)$,

$$A_h = \begin{bmatrix} W_h & X_h \\ Y_j & Z_h \end{bmatrix}, \qquad A_{h+1} = Z_h - Y_h W_h^{-1} X_h; \tag{3}$$

also compute W_h^{-1}, $W_h^{-1}X_h$, $Y_h W_h^{-1}$ for $h = 0, 1, \ldots, s-1$. Output: NO RECURSIVE FACTORS if there exists no desired sequence $A_0, \ldots, A_s$.

Equation (3) defines the following recursive factorization (for $h = 0, 1, \ldots, s-1$), used in [25] in order to solve sparse symmetric linear systems.

$$A_h = \begin{bmatrix} I & O \\ Y_h W_h^{-1} & I \end{bmatrix} \begin{bmatrix} W_h & O \\ O & A_{h+1} \end{bmatrix} \begin{bmatrix} I & W_h^{-1} X_h \\ O & I \end{bmatrix}, \tag{4}$$

$$A_h^{-1} = \begin{bmatrix} I & -W_h^{-1} X_h \\ O & I \end{bmatrix} \begin{bmatrix} W_h^{-1} & O \\ O & A_{h+1}^{-1} \end{bmatrix} \begin{bmatrix} I & O \\ -Y_h W_h^{-1} & I \end{bmatrix}. \tag{5}$$

3. Some definitions and auxiliary estimates

Hereafter we will assume the customary arithmetic (respectively Boolean) machine models of parallel computation, where in each step each processor performs at most one arithmetic (respectively Boolean) operation, [4, 8]. Let t_A, p_A (respectively t_B, p_B) denote a minimal pair of the numbers of arithmetic (respectively Boolean) parallel steps and processors that suffice for a solution of a given computational problem. Minimality means that both numbers of steps and processors cannot be simultaneously decreased more than by constant factors. (Recall that we define t_A, p_A, t_B and p_B up to within constant factors). In particular, it is known that $(t_A, p_A) \leq (\log n, P_A(*, n))$ for $n \times n$ matrix multiplication, where $P_A(*, n)$ has been defined in the introduction,

$$n^2/\log n \leq P_A(*, n) \leq n^{\omega}, \tag{6}$$

provided that $M(n) = O(n^{\omega^*})$ arithmetic operations suffice for $n \times n$ matrix multiplication and that $\omega > \omega^*$, see [25, Appendix A]. Ignoring the overhead, we will use the current asymptotic bound of [6],

$$M(n) = O(n^{\omega^*}), \qquad \omega^* < \omega < 2.38, \tag{7}$$

although $\omega = 3$ and $M(n) = 2n^3$ in the algorithms presently used in practical computations. (The reader may easily adjust all our subsequent results using those current practical bounds.)

We will also apply the following upper bound for problems (i)-(iii) from [9], $(t_A, p_A) = (\log^2 n, T_A(n))$,

$$T_A(n) = M(*, n^{1.25}n, n^{1.25}) + M(*, n^{0.5}, n^2, n^{0.5}) + \min_{q,r}(M(*, r+1, q, n^2) + M(*, n, nr, n)), \tag{8}$$

where the minimization is over all pairs q and r such that $qr \leq n+1 \leq (q+1)r$; here and hereafter $M(*, p, q, s)$ denotes the minimum number of arithmetic operations required for $p \times q$ by $q \times s$ matrix multiplication. It is shown in [9] that

$$T_A(n) = o(n^{\omega - 0.5 - \delta}) \quad \text{for a } \textit{positive } \delta = \delta(\omega), \tag{9}$$

$$T_A(n) = o(n^{2.86}). \tag{10}$$

Equation (10) relies on the current record upper bounds on the number of arithmetic operations required for square and rectangular matrix multiplication whose further asymptotic acceleration could automatically decrease the exponent 2.86.

4. Reduction of *QR*-FACTORS, *LU*-FACTORS, and RECURSIVE FACTORS to INVERT. Application to HESSENBERG REDUCTION

In this section we will obtain the following upper estimates in the cases of problems (v)-(viii) (with $m = n$ in the case of problem (v)).

$$(t_A, p_A) \leqslant (\log^3 n, T_A(n)) \tag{11}$$

where $T_A(n) = o(n^{2.86})$ is defined by (8)-(10).

Theorem 4.1. *For problems* (vi) (*LU*-FACTORS) *and* (viii) (RECURSIVE FACTORS), *the bound* (11) *holds.*

Proof. Denote $A = A_0$ and consider (3) for $h = 0$. If the matrix W_0 is singular, then A has no nonsingular LU-factors. (Indeed, let the matrices

$$A = LU, \qquad L = \begin{bmatrix} L_{11} & O \\ L_{12} & L_{22} \end{bmatrix}, \qquad U = \begin{bmatrix} U_{11} & U_{12} \\ O & U_{22} \end{bmatrix}$$

be nonsingular. Then $L_{11}U_{11} = W_0$. Since L and U are nonsingular, so are L_{11} and U_{11}. Therefore W_0 is also nonsingular.) If W_0 is nonsingular, compute W_0^{-1}, $W_0^{-1}X_0$, $Y_0W_0^{-1}$ and A_1, see (3), (4) for $h = 0$. Due to (4), this reduces the original problem (viii) or (vi) for A to one or two such problems for half-size matrices (that is, for A_1 in the case of computing the RECURSIVE FACTORS or for W_0 and A_1 in case of computing the *LU*-FACTORS). Recursive application of that argument for $h = 0, 1, \ldots, s-1$ yields Theorem 4.1. □

Theorem 4.2. *Let A be an $m \times n$ matrix $m \geqslant n$. Then for problem* (iv) (*QR*-FACTORS)

$$(t_A, p_A) \leqslant (\log m + \log^3 n, M(n, m, n)/(\log m + \log^3 n) + T_A(n))$$

where $T_A(n)$ is defined by (8)-(10) *and $M(n, m, n)$ denotes the minimum number of arithmetic operations required for $n \times m$ by $m \times n$ and $m \times n$ by $n \times n$ matrix multiplications. In particular* (11) *certainly holds if $m \leqslant n^{1.25}$.*

Theorem 4.2 follows from Theorem 4.1, because QR-factors of A can be computed via computing LU-factors of A^TA. Indeed $A^TA = R^TQ^TQR = R^TR$ for a triangular matrix R such that (2) holds, so the Cholesky factorization $A^TA = LU = LL^T$ defines QR-factors of A as follows: $R = U$, $Q = AR^{-1}$.

Remark 4.3. Even a nonsingular matrix A may have neither LU-factorization nor recursive factorization (4); this is the case for $A = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ for example. To avoid that problem, one may factor a matrix BA, or AC, or BAC where B and C are permutation matrices (compare [1]) or, say orthogonal matrices.

Next we will estimate the arithmetic complexity of the parallel Hessenberg reduction. We will need to define Krylov matrices (to be used also in Sections 6 and 9) and to recall two auxiliary facts.

Definition 4.4. The $n \times m$ matrix $K(A, v, m) = [v, Av, A^2v, \ldots, A^{m-1}v]$, defined for an $n \times n$ matrix A and for an n-dimensional vector v, is called an $n \times m$ *Krylov matrix* (compare [11, 26]). We will write $K(A, v) = K(A, v, n)$.

Proposition 4.5 (see Borodin and Munro [3, p. 128]; Keller-Gehrig [13]). *For given A and v, the Krylov matrix $K(A, v, m)$ can be computed using $2\lceil \log_2 m \rceil$ multiplications of matrices of sizes $n \times n$ by $n \times k$ for $k \leqslant m$ or $k = n$.*

HESSENBERG REDUCTION of A in many cases can be reduced to QR-factorization of $K(A, v)$, due to the following fact.

Fact 4.6. *If $K(A, v)$ is nonsingular and $K(A, v) = QR$, $Q^{\mathrm{T}}Q = \mathrm{I}$, and R is upper triangular, then $Q^{\mathrm{T}}AQ = H$ where H is in the upper Hessenberg form.*

Proof (compare [26, p. 253], [11, p. 224]). Let $Q = [q_1, q_2, \ldots, q_n]$. Recall that Q is orthogonal, R is upper triangular, and $Q = K(A, v)R^{-1}$. It follows that the vector q_i is orthogonal to any linear combination of the first $i-1$ columns of $K(A, v)R^{-1}$ and consequently to the first $i-2$ columns of $AK(A, v)R^{-1} = AQ$ for $i = 3, 4, \ldots, n$. Therefore, $q_i^{\mathrm{T}}Aq_j = 0$ if $i > j+1$. □

Corollary 4.7. *Let the Krylov matrix $K(A, v)$ be nonsingular for a given $n \times n$ matrix A and for a fixed vector v. Then, for problem* (vii) (HESSENBERG REDUCTION), (11) *holds.*

The assumption that $K(A, v)$ is nonsingular holds with probability close to 1 for a random choice of A and v (compare Section 9).

Remark 4.8. The above reduction of problem (vii) to computing the Krylov matrix and its QR-factors also implies that the sequential arithmetic complexity of problem (vii) (HESSENBERG REDUCTION) is $O(M(n)\log n) = o(n^{2.38})$ in the case where the Krylov matrix $K(A, v)$ (associated with an input matrix A and with a fixed vector v) is nonsingular.

Remark 4.9. Kozen [15] has independently arrived at polylog time solutions of problems (v) and (vi), but his solutions are not as processor efficient as ours announced in [22].

5. Computations for a well-conditioned and/or strongly diagonally dominant matrix

In this section we will extend the results of [25] on numerical inversion of well-conditioned matrices to computing $|\det A|$ and to the solution of problems (v)–(viii) for the price of a small increase (by a factor of $\log n$) of the parallel arithmetic time bound, so

$$(t_A, p_A) \leq (\log^2 n, P_A(*, n)) \tag{12}$$

for the numerical matrix inversion in [25], and

$$(t_A, p_A) \leq (\log^3 n, P_A(*, n)) \tag{13}$$

for the problems solved in this section. Here $P_A(*, n) = o(n^{2.38})$ is defined by (6) and (7).

Definition 5.1. Hereafter $\|W\|_h$ denotes the h-norm of a matrix W, $h = 1, 2, \infty$; $\|W\|_h / n \leq \max_{i,j} |w_{ij}| \leq \|W\|$ if $W = [w_{ij}]$, see [11]; $\operatorname{cond}_h W = \infty$ if W is singular, $\operatorname{cond}_h W = \|W\|_h \|W^{-1}\|_h$ otherwise; $\|W\| = \|W\|_2$, $\operatorname{cond} W = \operatorname{cond}_2 W$.

Definition 5.2 (see Pan and Reif [25]; compare also Golub and Van Loan [11]). We will call an $n \times n$ matrix W *well-conditioned* if

$$\operatorname{cond} W = n^{O(1)}; \tag{14}$$

we will call a matrix $W = [w_{ij}]$ *strongly diagonally dominant* (with respect to a constant C) if

$$\|I - D^{-1}(W) W\|_1 < 1 - n^{-C} \quad \text{or} \quad \|I - W D^{-1}(W)\|_\infty < 1 - n^{-C}, \tag{15}$$

where $D(W) = \operatorname{diag}(w_{11}, \ldots, w_{nn})$. Here and hereafter $n^{O(1)}$ denotes the values upper bounded by a polynomial in n.

Theorem 5.3 (Pan and Reif [25]). *Let A be a well-conditioned and/or strongly diagonally dominant matrix and g be a positive constant. Then the bounds* (12) *hold for the problem of computing a matrix $\tilde{A}^{-1}$ such that $\|A^{-1} - \tilde{A}^{-1}\| / \|A^{-1}\| \leq 2^{-n^g}$.*

Next we will extend this result to problems (vi) and (vii) for a symmetric positive definite matrix A and then to computing $|\det A|$ for an arbitrary matrix A. We will use the customary concept of a symmetric positive definite matrix but will simplify its customary definition as follows.

Definition 5.4. A matrix A is called symmetric positive definite if $A = V^T V$ for a nonsingular matrix V.

Next we will recall an auxiliary result from [25].

Proposition 5.5 (Pan and Reif [25]). *Let* (3) *hold and A_h be symmetric positive definite. Then $\|W_h\| \leq \|A_h\|$, $\|W_h^{-1}\| \leq \|A_h^{-1}\|$, $\|A_{h+1}\| \leq \|A_h\|$, $\|A_{h+1}^{-1}\| \leq \|A_h^{-1}\|$, and W_h, A_{h+1} are symmetric positive definite.*

If we need to compute the LU-factorization and/or the recursive factorization (3)–(5) for a well-conditioned and symmetric positive definite matrix A, we may proceed as in the proof of Theorem 4.1, applying the algorithm of [25] and Theorem 5.3 in order to invert the matrices A_h and W_h for $h = 0, 1, \ldots, s$. By the virtue of Proposition 5.5, all those matrices are well-conditioned and symmetric positive definite if $A = A_0$ is well-conditioned and symmmetric positive definite. Furthermore, the recursive process does not magnify the relative errors of the computed approximations to the inverses. Indeed, recall that $\|V\| * \|V^{-1}\| \geqslant 1$ for any nonsingular matrix V; consequently, $\|W_h\| \geqslant 1/\|W_h^{-1}\|$ and therefore, $\|W_h\| \geqslant 1/\|A_h^{-1}\|$: see Proposition 5.5; furthermore, we obtain along this line that

$$\min\{\|W_h\|, \|A_{h+1}\|\} \geqslant 1/\|A_h^{-1}\|, \qquad \min\{\|W_h^{-1}\|, \|A_{h+1}^{-1}\|\} \geqslant 1/\|A_h\|.$$

Thus the norms of W_h, of A_{h+1}, and of their inverses are nicely bounded from above and below in terms of the values $\|A\|$, $\|A^{-1}\|$, and their reciprocals, so the bound of Theorem 5.3 on the relative error norms (2^{-n^g} for any fixed constant g) can be extended throughout the computations (compare also [20, Part II] on the error analysis of matrix operations). Summarizing we arrive at the following theorem.

Theorem 5.6. *Let A be a well-conditioned and symmetric positive definite $n \times n$ matrix. Then the matrices A_{h+1}, W_h^{-1}, A_h^{-1} in the recursive factorization* (3)–(5) *for $h = 0, 1, \ldots, s-1$ (as well as the LU-factors of A) can be computed with relative error norms at most 2^{-n^g} (for any constant g) using $O(\log^3 n)$ parallel arithmetic steps and $P_A(*, n)$ processors, that is, for those computational problems bound* (13) *holds.*

Corollary 5.7. *Let A be a well-conditioned matrix. Then $|\det A|$ can be computed with relative error at most 2^{-n^g}, and QR-factors of A can be computed with relative error norms of at most 2^{-n^g} for any constant g using $O(\log^3 n)$ arithmetic parallel steps, $P_A(*, n)$ processors, that is, for those two computational problems bound* (13) *holds. The same complexity bound holds for the Hessenberg reduction of A, provided that the Krylov matrix $K(A, v)$ is well-conditioned for a fixed vector v.*

Proof. If A is well-conditioned, then so is $A^T A$. Besides, $A^T A$ is symmetric positive definite, so we may apply Theorem 5.6 in order to compute the LU-factors of $A^T A$. This will give the QR-factors of A and the value of the $\det(A^T A) = |\det A|^2$ with the required precision (see the proof of Theorem 4.2). Due to Fact 4.6 and Proposition 4.5, that algorithm can be extended to the Hessenberg reduction of A. □

Remark 5.8. If A is strongly diagonally dominant, then $AD^{-1}(A)$ and/or $D^{-1}(A)A$ are well-conditioned (see (14), (15)) so we may compute $|\det(AD^{-1}(A))| = |\det(D^{-1}(A)A)|$ and then easily recover $|\det A| = |\det(D^{-1}(A)A)| * |\det D(A)|$; we may proceed similarly in order to compute QR-factors of A. Using Fact 4.6, we may extend this argument to computing the Hessenberg reduction of A, provided that the matrix $K(A, v)$ is strongly diagonally dominant for a fixed vector v. In all

these cases we assure the relative error norms at most 2^{-n^s} using $O(\log^3 n)$ steps, $P_A(*, n)$ processors.

Remark 5.9. The matrix inversion algorithm of [25], and consequently the results of this section, can be extended to all ill-conditioned matrices, but then the parallel time may increase; an order of $O(\log^k n \log(n \log \mathrm{cond}(A)))$ parallel arithmetic steps will suffice, with $k = 1$ for computing $\tilde{A}^{-1}$ (compare Theorem 5.3 and [25]) and with $k = 2$ for the computations studied in Theorem 5.6 and Corollary 5.7.

6. Arithmetic cost of parallel inversion of integer matrices

In this section we will prove the following estimate.

Theorem 6.1. $(t_A, p_A) \leq (\log^2 n, P_A(*, n))$ *for problems* (i)-(iv) *for an integer input matrix* A *such that*

$$\log\log\|A\| = O(\log n). \tag{16}$$

We will start with the following auxiliary result, which may be of some interest in numerical linear algrebra.

Proposition 6.2. *Let*

$$q > 3n^2\|A\|. \tag{17}$$

Let $H = [h_{ij}]$ *denote the* $n \times n$ *matrix of cyclic permutation such that*

$$h_{ij} = \begin{cases} 1 & \text{if } i - j = 1 \bmod n, \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

Let

$$v = [1, 0, \ldots, 0], \qquad V = HA + qH. \tag{19}$$

Then the matrix $K(V, v) = [v, Vv, \ldots, V^{n-1}v]$ *is strongly diagonally dominant.*

Proof. Expand the expressions

$$V^s v = (HA + qH)^s v \tag{20}$$

for $s = 0, 1, \ldots, n-1$. The term $(qH)^s v$ is the vector filled with zeros except for its $(s+1)$st entry q^s, so this term brings the value q^s into the (s, s) position in $K(V, v)$. It remains to show that the sum of the absolute values of all the entries of all other terms in the expansion of (20) combined is less than $\frac{1}{2}q^s$ (provided that (17)-(19) hold). It suffices to give the similar estimate for the expansion of $W^3 v = (aE + qH)^3 v$, where $a \leq \|A\|$ denotes the maximum absolute value of an entry of A and where E

denotes the $n \times n$ matrix filled with ones, so $E^i = n^{i-1}E$ for all i. Specifically, it suffices to show that

$$\sigma_s < \tfrac{1}{2}q^s, \tag{21}$$

where σ_s denotes the sum of all the entries of $W^s v - (qH)^s v$.

$$\sigma_s = \sum_{i=1}^{s} q^{s-i}(an)^i s!/((s-i)!\,i!) = (q+an)^s - q^s.$$

Therefore, (21) is equivalent to the inequalities $(q+an)^s < \frac{3}{2}q^s$ or $(1+(an/q))^s < \frac{3}{2}$, which immediately follow from (17) for all $s < n$. (It is sufficient to consider the case where $s = n-1$.) □

Proof of Theorem 6.1. Now we will arrive at Theorem 6.1 as follows (see the details in [21]). Due to Proposition 6.2, we may apply the algorithm of [25] in order to invert $K(V, v)$ (with high precision). $O(\log n)$ iterations, that is, $O(\log^2 n)$ arithmetic steps, $P_A(*, n)$ processors, will suffice under (16)-(19). The coefficient vector $c(V)$ of the characteristic polynomial of the matrix V satisfies the linear system of equations, $K(V, v)c(V) = V^n v$. Knowing $K^{-1}(V, v)$ (with high precision) we may compute $c(V)$ with high precision and then recover the exact values of $c(V) = c(A) \bmod q$ via rounding-off the entries of $c(V)$ to the neasest integers. Then we may recover $c(A)$ if $q > 2(1+\|A\|)^n$. Similarly we may compute $(H^T V)^{-1} \det(H^T V) = \operatorname{adj}(H^T V)$. (We may compute $(H^T V)^{-1}$ using the algorithm of [25], because $H^T V = A + q1$ is strongly diagonally dominant, and because we have already computed $\det(H^T V) = \det H^T \det V = (-1)^{n-1} c_n(V)$.) Then we may recover adj A via reduction modulo q and finally compute $A^{-1} = \operatorname{adj} A / \det A$ and $A^{-1}u$. □

Remark 6.3. $\log_2|\det V|$ may exceed q^n, and we generally need to have $q > 2(1+\|A\|)^n$ above, so we may need to involve $n^2 \log_2\|A\|$-bit numbers in the above computations.

7. Boolean circuit complexity: some auxiliary estimates

Hereafter we will assume that the matrix A is filled with integers, we will apply the Boolean circuit machine model of [4, 8] (where in each step each processor may perform at most one Boolean operation), and we will estimate (t_B, p_B) from above for problems (i)-(iv) and for some auxiliary problems, defining both t_B and p_B up to within constant factors (compare Section 3). Such bounds on (t_B, p_B) are required in the study of the asymptotic complexity of combinatorial computations (see [4, 10, 16, 19]).

Let us recall some known estimates for the parallel Boolean complexity (t_B, p_B) of arithmetic operations (see [14, 28]).

Proposition 7.1. *For a natural* d, $(t_B, p_B) \leq (c, d)$ *for an addition of two integers modulo* 2^d; $(t_B, p_B) \leq (\log d, b(d))$ *for a multiplication of two integers modulo* 2^d; $(t_B, p_B) \leq (\log^2 d, b(d))$ *for computing the quotient and the remainder of the division of two integers lying between* 0 *and* 2^d. *Here (and hereafter)* $c = O(1)$ *is a positive constant, and*

$$b(d) = O(d \log d \log\log d) \tag{22}$$

denotes the minimum number of Boolean operations required in order to multiply two integers modulo 2^d.

We will also deduce and use the following estimates for the Boolean complexity of matrix multiplication (extending the estimates of [5, 25]).

Proposition 7.2. *For the evaluation of an* $n \times n$ *matrix product* UV, $(t_B, p_B) \leq (\log(d^* n), n^\omega b(d^*))$ *where* ω *is the exponent from* (6), (7), $\omega < 2.38$, $b(d)$ *is defined by* (22), *and*

$$d^* = O(\log(\|U\| * \|V\| \log n)) \tag{23}$$

Proof. Without loss of generality, we may assume that $n = s^h$ for a fixed but sufficiently large s and for $h \to \infty$, that the matrix products are evaluated via recursive bilinear algorithms, and that there exists a basis bilinear algorithm of rank $M \leq s^\omega$ for computing the product XY of a pair of $s \times s$ matrices $X = [x_{ij}]$ and $Y = [y_{jk}]$ (see [3, Section 2.5] and [20]). Such a bilinear algorithm computes at first $2M$ linear functions,

$$L_q = \sum_{i,j} f(i, j, q) x_{ij}, \qquad L_q^* = \sum_{j,k} f^*(j, k, q) y_{jk}, \quad q = 1, \ldots, M, \tag{24}$$

then the M products $L_q L_q^*$ and finally the s^2 output values

$$\sum_j x_{ij} y_{jk} = \sum_{q=1}^{M} f^{**}(k, i, q) L_q L_q^* \quad \text{for all } i, k. \tag{25}$$

The bilinear algorithm (24), (25) can be applied, where all the x_{ij}, y_{jk} and consequently all the L_q, L_q^* denote $p \times p$ matrices for a natural p, so we apply the basis bilinear algorithm recursively with successive substitutions of $s^h \times s^h$ matrices for x_{ij} and y_{jk} (for all i, j and k), where one needs to multiply a pair of $n \times n$ matrices with $n = s^{h+1}$. Then (24) and (25) immediately imply that

$$t_{A,h+1} \leq t_{A,h} + 2 \log_2 s + \log_2 M + 4,$$

$$p_{A,h+1} \leq \max\{2s^{2h+2}, Ms^{2h}, p_{A,h} M\},$$

where $t_{A,g}, p_{A,g}$ denote the arithmetic time and the number of processors used in the above recursive bilinear algorithm for $s^g \times s^g$ matrix multiplication. Since $M \leq s^\omega$, this immediately implies the bound

$$(t_A, p_A) \leq (\log n, n^\omega) \tag{26}$$

for $n \times n$ matrix multiplication (compare [25, Appendix A]).

Furthermore it is possible to assume, without loss of generality, that the values $f(i, j, q)$, $f^*(j, k, q)$, and $f^{**}(k, i, q)$ are rational for all i, j, k, q (see [20]). Then we may even assume them to be integers if we evaluate $QQ^*Q^{**}\sum_j x_{ij}y_{jk}$ (rather than $\sum_j x_{ij}y_{jk}$) for all i and k (where Q, Q^* and Q^{**} are common multiples of the denominators of all $f(i, j, q)$, of all $f^*(j, k, q)$, and of all $f^{**}(k, i, q)$ respectively). In that case recursive application of the basis bilinear algorithm will output the product of two given $s^h \times s^h$ matrices times the multiplier $(QQ^*Q^{**})^h$; the subsequent division of the output by that multiplier (one arithmetic step, n^2 processors) will give the desired matrix product.

All the integer constants f, f^*, f^{**} are bounded as $h \to \infty$, so each multiplication by such a constant can be replaced by O(1) additions/subtractions. Then all the arithmetic steps of the recursive algorithm turn into additions/subtractions except for the single bilinear multiplication step (which multiplies L_q by L_q^*) and for the single step of division by QQ^*Q^{**}. Now Proposition 7.2 immediately follows from Proposition 7.1, except that we need to show that d^*-bit precision of computations will suffice with d^* bounded in (23). That bound on d^* follows from (24), (25), because all the input values other than f, f^*, f^{**} are upper bounded by $\|U\|$ and $\|V\|$ and because our scaling may increase the magnitudes of the values involved in the algorithm (24), (25) at most by a factor of $|QQ^*Q^{**}|^h < C^h$ for a constant C, which means adding at most $\log_2|QQ^*Q^{**}|^h = O(h) = O(\log n)$ bits to the precision of computation. The complexity of the divisions by $(QQ^*Q^{**})^h$ is small enough, because QQ^*Q^{**} is a natural constant (we may even use the QQ^*Q^{**}-ary rather than the binary or decimal representation of integers in our computations). □

Remarks 7.3. It is interesting that the straightforward combination of the available bounds on t_A ($t_A \leqslant \log n$) and on the bit precision of computations d^* only implies the inferior bound $t_B \leqslant \log d^* \log n$.

8. Deterministic Boolean complexity bounds

In this and in the next sections we will extend the parallel arithmetic complexity estimates of our Section 5 and of [9] in order to arrive at some deterministic and probabilistic upper bounds on the Boolean complexity (t_B, p_B) of the exact solution of problems (i)-(iv) in the case where the input entries of the matrix A (and of the vector u) are integers. The resulting estimates are summarized in Table 2 in Section 9. In the present section we will deal with the deterministic case, where the estimates are inferior but easier to deduce. In that case we will easily obtain the desired extension using Proposition 7.2; the algorithms can be reduced to $O(\log n)$ or $O(\log^2 n)$ matrix multiplications, so it essentially remains to bound the precision of computations.

At first let us assume that A is well-conditioned and/or strongly diagonally dominant and apply the algorithms supporting Theorem 5.3 and Corollary 5.7 in

order to compute A^{-1}, $A^{-1}u$, $|\det A|$, and then $|\det A|A^{-1}$ (with small relative errors). We will need $\lceil \log_2 |\det A| \rceil$ binary bits in order to represent $\det A$, so we will compute with $d \geq n \log_2 \|A\|$ bits. Indeed Hadamard's inequality [17],

$$|\det A| \leq \|A\|^n, \tag{27}$$

valid for all matrices A, turns into equality for some matrices A. ($\|A\|$ denotes the 2-norm of A.)

In fact, it will suffice to compute with d bits where

$$d = O(n \log(1 + \|A\| + \|u\|)), \tag{28}$$

where we assume that $\|u\| = 0$ unless we deal with problem (i). Indeed, we only need approximations to $|\det A|$ and to $A^{-1}|\det A|$with *absolute* errors below 0.5; this will enable us to recover the exact integer values of $|\det A|$ and of the entries of the matrix $A^{-1}|\det A|$ via rounding-off their approximations to the nearest integers. (The pair $(A^{-1}|\det A|, |\det A|)$ defines A^{-1}.) The Hadamard inequality (27) implies the upper bounds $\|A\|^n$ and $\sqrt{n}\|A\|^{n-1}$ on the values of $|\det A|$ and $\|\text{adj } A\|$ respectively. (We denote $\text{adj } A = A^{-1} \det A$.) Therefore, it suffices to compute $|\det A|$ and $A^{-1}|\det A|$ with relative errors less than $0.5/\|A\|^n$ and $0.5/(\sqrt{n}\|A\|^{n-1})$ respectively, in order to keep the absolute errors below 0.5. The computation (ultimately based on matrix inversions via Newton's iterations of [25]) is stable and even self-correcting, so already d-bit precision of computations with $d = O(n \log(1 + \|A\|))$ supports the latter error bounds, which are assumed after $O(\log^2 n)$ iterations of the algorithms (see our Corollary 5.7 and [25, Appendix D]). The complexity of each iteration is dominated by the complexity of a pair of $n \times n$ matrix multiplications, so we apply Proposition 7.2 and arrive at the following result.

Theorem 8.1. *Let A be a well-conditioned and/or strongly diagonally dominant $n \times n$ matrix filled with integers. Then $(t_B, p_B) \leq (\log^2 n \log(dn), P_A(*, n)b(d))$ for the problems of the exact evaluation of A^{-1}, $A^{-1}u$, and $|\det A|$. Here the values $P_A(*, n)$, $b(d)$, and d are defined by* (6), (7), (22), *and* (28).

Theorem 8.1 does not cover the case of arbitrary integer input matrices. In that case the current best deterministic estimates for the Boolean circuit complexity are obtained via the algorithm of [9]. That algorithm is reduced to $O(\log n)$ matrix multiplications, and it suffices to compute with $\tilde{d}$-bit precision where

$$\tilde{d} = O(n \log(n\|A\|)). \tag{29}$$

Thus the desired estimates follow from Proposition 7.2. (Note that we may afford higher precision of computations in the stage of solving the system of Newton's identities in the algorithm of [9] because in that stage $(t_A, p_A) \leq (\log^2 n, n)$, (see [24]).)

Theorem 8.2. *For problems* (i)-(iv) *in case of an integer input matrix A,*

$$(t_B, p_B) \leq (\log n \log(\tilde{d}n), T_A(n)b(\tilde{d})), \tag{30}$$

where $T_A(n)$, $b(d)$, $\tilde{d}$ are defined by (8)-(10), (22) *and* (29).

Remark 8.3. The processor bound of Theorem 8.2 is inferior to one of Theorem 8.1 (because the algorithm of [9] is less processor efficient than the algorithms supporting Theorem 5.3 and Corollary 5.7), but it is superior to the deterministic bound on p_B, which could follow from the algorithm in Section 6 (compare Remark 6.3) (even though Theorem 6.1 gives superior and, in fact, suboptimal bounds on the arithmetic parallel complexity of problems (i)-(iv)).

9. Probabilistic Boolean complexity estimates for problems (i)–(iv).

In this section we will establish probabilistic upper bounds on the parallel Boolean complexity of problems (i)-(iv), which will substantially improve the worst case deterministic bounds of Theorem 8.2. We will start with an auxiliary algorithm. Let, for an integer p and a matrix A,

$$\det A \neq 0 \bmod p, \tag{31}$$

and let $S(i)$ denote $A^{-1} \bmod p^{2^i}$.

Newton–Hensel's algorithm (Moenck and Carter [18]). *Input*: $S(0) = A^{-1} \bmod p$. Compute

$$E(i) = \mathrm{I} - AS(i-1) \bmod p^{2^i}, \qquad S(i) = S(i-1)(\mathrm{I} + E(i)), \quad i = 1, \ldots, k.$$

Output: $S(k) = A^{-1} \bmod p^{2^k}$.

Now, let us assume that for a fixed prime $p = \|A\| + n^{O(1)}$, for a matrix A, and for a vector v, the matrices A^{-1} and $K^{-1}(A, v)$ have been computed modulo p (recall Definition 4.4 and Proposition 4.5). Then Newton-Hensel's algorithm will give us $A^{-1} \bmod p^{2^k}$ and $K^{-1}(A, v) \bmod p^{2^k}$; then we may compute $c(A) \bmod p^{2^k} = K^{-1}(A, v)A^n v \bmod p^{2^k}$. (By the virtue of the Cayley–Hamilton theorem, the vector $c = c(A)$ of (1) satisfies the matrix-vector equation $K(A, v)c = A^n v$ for any vector v.) The entries of the vector $c(A)$ and of the matrix $\operatorname{adj} A = A^{-1} \det A$ are integers, whose absolute values are less then $(1 + \|A\|)^n$. If $p^{2^k} > 2(1 + \|A\|)^n$, we may now recover $\operatorname{adj} A$ and $c(A)$, that is, we may solve problems (i)-(iv), because $\operatorname{adj} A$ and $c_n(A) = (-1)^n \det A$ also define $A^{-1} = \operatorname{adj} A / \det A$. We will compute the matrices $A^{-1} \bmod p$ and $K^{-1}(A, v) \bmod p$ using the algorithm of [9], which supports (30) in that computation (in that case we perform all arithmetic operations modulo p and choose $p > n$ to avoid divisions by zero modulo p, for the algorithm includes divisions by $2, 3, \ldots, n$). Thus, in spite of the relatively high *arithmetic* cost of that stage of the computation, its Boolean cost will be dominated by the cost of Newton-Hensel's algorithm (because the bit precision needed for computing A^{-1} and $K^{-1}(A, v)$ modulo p is only $\lceil \log_2 p \rceil = O(\log(\|A\| + n))$), so we have obtained a simple proof of the following result.

Theorem 9.1. *Let A be a matrix filled with integers, u and v be two vectors filled with integers, and p be a prime such that*

$$\det K(A, v) \neq 0 \bmod p. \tag{32}$$

Then for problems (iii) *and* (iv),

$$(t_B, p_B) \leq (\log^2(dn), P_A(*, n)b(d)), \tag{33}$$

where the values $P_A(*, n)$, $b(d)$, and d are defined by (6), (7), (22) *and* (28). *If, in addition,* (31) *holds, then* (33) *holds also for problems* (i) *and* (ii).

Remark 9.2. The arithmetic cost of the algorithm supporting Theorem 6.1 is much lower than the arithmetic cost of the algorithm supporting Theorem 9.1, but the opposite is true for the Boolean costs of those two algorithms. In fact, the algorithm of Section 6, appropriately combined with Newton-Hensel's algorithm, may be also used to derive (33) assuming (31) and (32) (see [21]), but this would require more work than the proof of Theorem 9.1.

Remark 9.3. If (under (32)) we solve problem (ii), we may immediately check if (31) holds. If $\det A \bmod p = 0$, then we may choose and test new primes p (sequentially or in parallel), until we assure (31). If $\det A \neq 0$, eq. (31) must hold for at least one of n distinct primes $p \geq \|A\|$, due to (27).

In the remainder of this section we will extend the latter observation in order to estimate the probability that (31), (32) hold for a random choice of p. We will use the following auxiliary result.

Proposition 9.4 (Schwartz [29]). *Let $f(n)$, $h(n)$, $k(n)$ be three functions in n such that $h(n)$ is integer valued,*

$$0 < (h(n))^{1/k(n)} < f(n)/n, \qquad k(n) \geq 1, \qquad \lim_{n\to\infty} f(n) = \infty. \tag{34}$$

Let $p = p(n)$ be a random prime in the interval

$$f(n)/n < p < f(n). \tag{35}$$

Then $h(n) \bmod p = 0$ with probability $O((k(n) \log f(n))/f(n))$ as $n \to \infty$.

Proof. The interval (35) contains at least $Cf(n)/\log f(n)$ distinct primes for a positive constant C. Less than $k(n)$ of them may divide $h(n)$, due to (34). □

We will apply Proposition 9.4 in the two cases where $f(n) = 2(n\|A\|)^r$, $r > 2$, and either $h(n) = |\det A|$ and $k(n) = n/r$, or $h(n) = |\det K(A, v)|$, $\|v\| \leq n\|A\|$, and $k(n) = n^2/r$. In both cases (34) holds, due to (27) and to the following simple inequality: $\|K(A, v)\| \leq n\|A\|^{n-1}\|v\|$. Those two applications of Proposition 9.4 imply the following corollary.

Corollary 9.5. *Let A be an $n \times n$ integer matrix, v be an integer vector, $\|v\| \leq n\|A\|$, and p be a random prime in the interval $2(n\|A\|)^r/n < p < 2(n\|A\|)^r$ where $r > 2$. If*

$$\det A \neq 0, \tag{36}$$

then (31) *holds with probability* $1 - O(\log(n\|A\|)/(n^{r-1}\|A\|^r))$ *as* $n \to \infty$. *If*

$$\det K(A, v) \neq 0, \tag{37}$$

then (32) *holds with probability* $1 - O(\log(n\|A\|)/(n^{r-2}\|A\|^r))$ *as* $n \to \infty$.

Corollary 9.5 implies that the above random choice of p satisfies the assumptions of Theorem 9.1 with probability converging to 1 as long as (36), (37) hold.

Let us extend our randomization trying to relax assumption (37). Let $m_A(\lambda)$ denote the *minimum polynomial* of A, that is, let $m_A(\lambda)$ be the minimum-degree monic polynomial such that $m_A(A) = 0$, and let us assume hereafter that $m_A(\lambda)$ has degree n or, equivalently, that

$$\det(\lambda I - A) = m_A(\lambda). \tag{38}$$

(In fact (38) holds with probability 1 for a random choice of an integer matrix A, compare [29], but we assume that A is *fixed* input matrix.)

In the algorithm supporting Theorem 8.1 and based on the algorithm of [9] and on Newton–Hensel's algorithm, we may choose the integer entries v_i of the vector v at random, say in the following interval.

$$-n^2 \leq v_i \leq n^2, \quad i = 0, \ldots, n-1. \tag{39}$$

(We will choose $r > 4$ in Corollary 9.5 because the latter choice of v_i only implies that $\|v\| \leq n^3$ rather than $\|v\| \leq n\|A\|$.)

Proposition 9.6. *Under the random choice of $v_0, \ldots, v_{n-1}$ in the interval* (39), *inequality* (37) *holds with probability converging to* 1 *as* $n \to \infty$, *provided that* (38) *holds.*

Proof. Equation (38) implies that the matrices $I, A, \ldots, A^{n-1}$ are linearly independent; then $\det K(A, v)$ is not equal to 0 identically in v. It remains to observe that $\det K(A, v)$ is a polynomial of degree at most n in v_i for each i and to apply [29, Corollary 1, p. 702]. □

For problems (i)-(iii), we only need to use (37) and (38) when we compute $\det A$. However, for that purpose it is sufficient if the following generalized version of (38) holds,

$$\det(\lambda I - BAC) = m_{BAC}(\lambda), \tag{40}$$

where matrices B and/or C have been chosen at random in a class of some special matrices (say let $C = B^T$ and B be triangular or diagonal or permutation matrices filled with integers). That class can be chosen to assure (40) with high probability

unless $A = O$ (in fact, it is sufficient to assure linear independence of the matrices I, $BAC, \ldots, (BAC)^{n-1}$). For instance, for a fixed integer matrix A, the choice of $C = B^T$ and the uniform random choice of the entries of integer triangular matrix B in the interval between $-N$ and N as $N \to \infty$, $N = n^{O(1)}$ will suffice (recall that a violation of (38) and/or (40) would imply some polynomial equations of degrees $\leq n$ in the entries of the matrices A and/or BAC respectively, and again recall [29, Corollary 1]). If (40) holds, then $\det K(BAC, v) \neq 0$ with high probability, see Proposition 9.6, so we may compute $\det(BAC)$ and then recover $\det A = \det(BAC)/(\det B \det C)$. Summarizing we arrive at the following results.

Theorem 9.7. *There exist randomized algorithms for problems* (i)-(iii) *with an integer input matrix A and an integer input vector u that yield* (33) *with probability converging to* 1 *as* $n \to \infty$. *The same is true for problem* (iv) *provided that* (38) *holds.*

Our estimates for the Boolean complexity of problems (i)-(iii) in terms of $P_A(*, n) = o(n^{2.38})$, $T_A(n) = o(n^{2.86})$, $d = O(n \log(1 + \|A\|))$, $\tilde{d} = O(n \log(n\|A\|))$, and $b(d) = O(d \log d \log\log d)$, (see (6)-(10), (22), (28) and (29)), are summarized in Table 2, where Newton's matrix means a well-conditioned and/or strongly diagonally dominant matrix (for such matrices Newton's iterations converge fast, [25]).

Table 2. Boolean complexity estimates.

Problem	Integer matrix A	(t_B, p_B)	Algorithm
(i), (ii), (iii)	Newton's	$(\log^2 n \log(dn), P_A(*, n)b(d))$	deterministic
(i), (ii), (iii), (iv)	arbitrary	$(\log n \log(\tilde{d}n), T_A(n)b(\tilde{d}))$	deterministic
(i), (ii), (iii), (iv)	arbitrary, for (iv) it satisfies (38)	$(\log^2(dn), P_A(*, n)b(d))$	probabilistic

10. Some open problems

(1) Is it possible to prove bounds (12), or at least the bounds $(t_A, p_A) \leq (\log^3 n, P_A(*, n))$, for problems (i)-(viii) for a general matrix A?

(2) Is it possible to extend the bounds on (t_B, p_B) in Section 9 to the case of problems (v)-(viii) and to obtain similar deterministic bounds for all problems (i)-(viii)?

Appendix A. Alternative probabilistic parallel matrix computations

In this section we will give an alternative way of deriving probabilistic bounds on the complexity of parallel solutions of problems (i)-(iv); the bounds are almost

as good as the ones of Section 9, and we hope that the techniques used are of some interest in their own right.

Algorithm A.1. *Inputs*: integer matrix A and (for problem (i)) integer vector u.

Stage 0 (*Initialization*). Choose a random integer vector v of norm $\leqslant n^{O(1)}$ and $2n+1$ distinct integers $\lambda_1, \lambda_{2n}, \ldots, \lambda_{2n}$ (say, let $\lambda_j = j$) and compute the coefficients of the $2n+1$ polynomials $\prod_{j=1}^{2n}(\lambda - \lambda_j)$, $\prod_{k\neq j}(\lambda - \lambda_k)$, $j = 1, \ldots, 2n$. Here and hereafter $\prod_{k\neq j}$ denotes the product in k ranging from 1 to $2n$ but not taking the value j.

Stage 1. Compute the Krylov matrix $K(A, v, 2n)$ with the column vectors $A^i v$, $i = 0, 1, \ldots, 2n$, see Definition 4.4.

Stage 2. Compute the $2n+1$ vectors

$$b = \prod_{j=1}^{2n}(A - \lambda_j \mathrm{I})v, \qquad x(j) = \prod_{k\neq j}(A - \lambda_k \mathrm{I})v, \quad j = 1, \ldots, 2n, \tag{A.1}$$

which satisfy the following $2n$ linear systems of equations: $(A - \lambda_j \mathrm{I})x(j) = b$, $j = 1, \ldots, 2n$.

Stage 3. Compute the coefficients of the polynomials $q(\lambda)$ (of degree $\leqslant n-1$) and $p(\lambda)$ (monic of degree $\leqslant n$) such that $x_1(j) = q(\lambda_j)/p(\lambda_j)$ for $j = 1, \ldots, 2n$, where $x_1(j)$ is the first entry of the vector $x(j)$. If $p(\lambda)$ has degree n, output

$$p(\lambda) = \det(\lambda \mathrm{I} - A); \tag{A.2}$$

otherwise redefine a random vector v and go to Stage 1.

Stage 4. Compute $c_n A^{-1} u = s(A)u$ where $s(\lambda) = (p(\lambda) + c_n)/\lambda$. If $c_n \neq 0$, obtain $A^{-1}u$ via divisions by c_n.

Theorem A.2. *For an integer matrix A and integer vector u Algorithm* A.1 *computes the solutions to problems* (i), (iii) *and* (iv) *probabilistically for the arithmetic cost of* $O_A(\log^2 n, P(*, n))$ *and for the Boolean cost of* $O_B(\log^2 n \log^2 d, P(*, n)b(d))$ *where* $P(*, n)$, d, *and* $b(d)$ *are defined by* (6), (7), (22), *and* (28).

Proof. The correctness of equation (A.2) follows from Cramer's formulae for the solutions of linear systems and from (A.1). The correctness of Stage 4 follows from the Cayley-Hamilton theorem, $p(A) = \mathrm{O}$, so $s(A) = A^{-1}$. The cost of Stage 1 is $O_A(\log^2 n, P(*, n))$, see Proposition 4.5. The cost of Stage 2 is $O_A(\log n, P(*, n))$ since the $2n$ vectors $x(j)$, $j = 1, \ldots, 2n$, are the column vectors of the matrix $K(A, v, 2n-1)M$ where M is the $(2n) \times (2n)$ matrix whose column j is the coefficient vector of the polynomial $\prod_{k\neq j}(\lambda - \lambda_k)$ of degree $2n-1$. The cost of Stage 4 is obviously $O_A(\log n, n^2/\log n)$; indeed, the matrix $K(A, v)$ has already been computed, so it remains to compute a linear combination of its columns. To estimate the cost of Stage 3, observe that this is the stage of computing rational interpolation at $2n$ points, where *the existence of the rational interpolation function* $q(\lambda)/p(\lambda)$ *is assured.* Thus it remains to compute the $(n-1, n)$ entry of the *rational interpolation*

table, which is a specific entry of the extended Euclidean scheme for computing the greatest common divisor of the polynomial $\prod_{j=0}^{2n}(\lambda-\lambda_j)$ and of the interpolation polynomial of degree $\leqslant 2n-1$ taking the values $q(\lambda_j)/p(\lambda_j)$ at the points λ_j, $j=j=1,\ldots,2n$ (see [12]). The evaluation of such an entry is reduced to the solution of a linear system of equations with the matrix of the form $[T_1 \mid T_2]$ where T_1 and T_2 are $(2K)\times K$ integer Toeplitz matrices with $K=O(n)$ (see [4, 30]). Applying the algorithm of [24] we arrive at the estimates $O_A(\log^2 n, n^2)$ for the cost of Stage 3. Summarizing, the cost of Algorithm A.1 is $O_A(\log^2 n, P_A(*, n))$. To bound the Boolean circuit complexity estimates, we compute in Stage 3 with $O(n\log(n\|A\|)$-bit precision (compare [24, Corollary 10.2]) and perform all other computations in Stages 1-4 modulo p^h (except for the divisions by c_n in Stage 4). We choose a prime p and integers $\lambda_1,\ldots,\lambda_n$ such that $p=n^{O(1)}$ and all the values λ_j mod p are distinct for $j=1,\ldots,2n$; say we let $\lambda_j=j$ for all j and choose a prime p lying between $2n$ and $4n$. We choose h large enough, so that $p^h>2(n\|A\|^n\|u\|)$. This will imply that the precision of computations (28) will suffice in order to recover the coefficients of $\det(\lambda I-A)$, as well as the entries of the vector $c_nA^{-1}u$, from their values modulo p^h and we arrive at the desired Boolean complexity estimates of Theorem A.1. □

Acknowledgment

Back in 1985, John Reif and then Zvi Galil pointed out to me the importance of parallel inversion of integer matrices; the two referees made several helpful comments and suggestions; particularly they pointed out a flaw in the author's original proof of Proposition 9.6, the omission of the motivation for estimating the Boolean complexity in the original draft of this paper, and a simple argument useful for estimating the exponent of $T_A(n)$. Sally Goodall assisted in typing the paper.

References

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

[2] W. Baur and V. Strassen, On the complexity of partial derivatives, *Theoret. Comput. Sci.* **22** (1983) 317-330.

[3] A. Borodin and I. Munro, *The Computational Compliexity of Algebraic and Numeric Problems* (Elsevier, Amsterdam, 1975).

[4] A. Borodin, J. von zur Gathen and J. Hopcroft, Fast parallel matrix and GCD computation, *Inform. and Control* **52**(3) (1982) 241-256.

[5] A.K. Chandra, Maximal parallelism in matrix multiplication, Report RC-6193, IBM T.J. Watson Research Center, Yorktown Heights, NY, 1976.

[6] D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progressions, in: *Proc. 19th Ann. ACM Symp. on Theory of Computing* (1987) 1-6.

[7] L. Csanky, Fast parallel matrix inversion algorithm, *SIAM J. Comput.* **5**(4) (1976) 618-623.

[8] W. Eberly, Very fast parallel matrix and polynomial arithmetic, in: *Proc. 25th Ann. IEEE Symp. FOCS* (1984) 21-30.

[9] Z. Galil and V. Pan, Improving the ecffiiency of parallel algorithms for the evaluation of the determinant and of the inverse of a matrix, Tech. Report TR 85-5, Dept. of Computer Science, SUNYA, Albany, NY, 1985 (revised 1987).

[10] Z. Galil and V. Pan, Improved processor bounds for combinatorial problems in RNC, *Combinatorica* to appear.

[11] G.H. Golub and C.F. van Loan, *Matrix Computations* (John Hopkins Univ. Press, Baltimore, MD, 1983).

[12] F.G. Gustavson and D.Y.Y. Yun, Fast algorithms for rational Hermite approximation and solution of Toeplitz systems, *IEEE Trans. Circuits and Systems* **CAS-26**(9) (1979) 750-755.

[13] W. Keller-Gehrig, Fast algorithms for the characteristic polynomial, *Theoret. Comput. Sci.* **36** (1985) 309-317.

[14] D.E. Knuth, *The Art of Computer Programming: Seminumercial Algorithms, Vol.* 2 (Addison-Wesley, Reading, MA, 1981).

[15] D. Kozen, Fast parallel orthogonalization, *SIGACT News* **18**(2) (1987) 47.

[16] L. Lovász, Connnectivity algorithms using rubber-bands, in: *Proc. Sixth Conf. on Foundations of Software Technology and Theoretical Computer Science*, New Delhi, India, Lecture Notes in Computer Science **241** (Springer, Berlin, 1986) 394-412.

[17] M. Marcus and H. Minc, *A Survey of Matrix Theory and Matrix Inequalities* (Allyn & Bacon, Newton, MA, 1964).

[18] R.T. Moenck and J.H. Carter, Approximate algorithms to derive exact solutions to systems of linear equation, in: *Proc. Eurosam*, Lecture Notes in Computer Science **72** (Springer, Berlin, 1979) 63-73.

[19] K. Mulmuley, U. Vazirani and V. Vazirani, Matching is as easy as matrix inversion, *Combinatorica* (1987) to appear.

[20] V. Pan, *How to Multiply Matrices Faster*, Lecture Notes in Computer Science **179** (Springer, Berlin, 1984).

[21] V. Pan, Fast and efficient parallel algorithms for exact inversion of integer matrices, in: *Proc. Fifth Conf. on Foundations of Software Technology and Theoretical Computer Science*, New Delhi, India, Lecture Notes in Computer Science **206** (Springer, Berlin, 1985) 504-521.

[22] V. Pan, Parallel complexity of polylogarithmic time matrix computations, *J. Abstracts Amer. Math. Soc.* **7**(4) (1986) 288.

[23] V. Pan, Algebraic complexity of computing polynomial zeros, *Comp. Math. Appl.* (1987) to appear.

[24] V. Pan and J. Reif, Displacement structure and the processor efficiency of fast parallel Toeplitz computations, Tech. Report TR 87-9, Dept. of Computer Science SUNYA, Albany, New York 1987 (short version in: *Proc. 28th Ann. IEEE Symp. FOCS* (1987)).

[25] V. Pan and J. Reif, Fast and efficient parallel solution of linear systems, Tech. Report TR 87-2, Dept. of Computer Science SUNYA, Albany, New York, 1987 (short version in: *Proc. 17th Ann. ACM Symp. on Theory of Computing* (1985) 143-152).

[26] B.N. Parlett, *The Symmetric Eigenvalue Problem* (Prentice-Hall, Englewood Cliffs, NJ, 1980).

[27] F.P. Preparata and D.V. Sarwate, An improved parallel processor bound in fast matrix inversion, *Inform. Process. Lett.* **7**(3) (1978) 148-149.

[28] J.E. Savage, *The Complexity of Computing* (Wiley, New York, 1976).

[29] J.T. Schwartz, Fast probabilistic algorithms for verification of polynomial identities, *J. ACM* **27**(4) (1980) 701-717.

[30] J. von zur Gathen, Parallel algorithms for algebraic problems, *SIAM J. Comput.* **13**(4) (1984) 802-824.