THE UNIVERSITY OF CALGARY

Efficient Exact Parallel Matrix Inversion

by

Wei Zhang

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA
JULY, 1995

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled, "Efficient Exact Parallel Matrix Inversion" submitted by Wei Zhang in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE.

Wayne Eberly

Supervisor, Wayne Eberly,

Department of Computer Science

Richard Cleve

Richard Cleve,

Department of Computer Science

Peter Lancaster

Peter Lancaster,

Department of Mathematics and Statistics

Date Sept 18, 1995

Abstract

Processor efficient parallel algorithms are those that use parallel time that is polylogarithmic in the input size and a number of operations which is asymptotically within a polylog factor of the best known sequential step count for solving the corresponding problems.

Recently Reif gave processor efficient parallel algorithms for general dense matrix computations provided that the input matrix has rational or integer entries. A similar method was exploited to achieve processor efficient algorithms for Toeplitz and Toeplitz-like matrix computations. These methods use different models and techniques from the previously known methods. This thesis provides an analysis and simplified version of Reif's algorithms and shows that the bit precision in these algorithms can be reduced significantly and be made optimal. The improvement of the bit precision gives evidence that Reif's algorithms can be made "practical".

Acknowledgements

When I was trying hard to understand John Reif's recent paper on Toeplitz and Toeplitz-like matrix computations, Wayne Eberly, my supervisor, suggested that I analyze the algorithms on a different model. This suggestion has led me to use my own way to prove the correctness of Reif's algorithms. The proof has become the main part of this thesis.

I would like to thank Wayne Eberly: for the challenging project and its area that lead to so many interesting ideas, for the patience and enthusiasm that helped me to finish the work, and for the thorough supervision that directly contributed to the final version of the thesis.

I would like to thank my examining committee, for their questions and comments, and for making my defense an enjoyable experience.

I would also like to thank fellow graduate students, Dave Wilson and Christino Tamon, for comments on my work, and for discussions of various research questions.

Finally I would like to thank my parents and all my friends back in China for encouraging me to continue my research interest and come to Canada.

Contents

Notation

$\mathbb{R}$	The real numbers
$\mathbb{Q}$	The rational numbers
$\mathbb{Z}$	The integers
$D^{m \times n}$	The set of $m \times n$ matrices over domain D
$\psi(A)$	The characteristic polynomial of the matrix A
$\phi(A)$	The displacement generator of the matrix A
$\alpha(A)$	The displacement rank of the matrix A
$\|A\|$	The norm of the matrix A
$\Delta(A)$	The Schur complement of the matrix A
$\det(A)$	The determinant of the matrix A
A^{-1}	The inverse of the matrix A
A^T	The transpose of the matrix A
$\text{rank}(A)$	The rank of the matrix A
$\text{adj}(A)$	The adjoint of the matrix A
$\text{tr}(A)$	The trace of the matrix A

CHAPTER 1
Introduction

Parallel matrix computation is a fertile research area in computer science. Problems in this area that will be considered in this thesis include computing the determinant, the inverse, the characteristic polynomial, and the rank of an input matrix, and solving a nonsingular linear system.

Processor efficient parallel algorithms are those that use parallel time that is polylogarithmic in the input size and a number of operations which is asymptotically within a polylog factor of the best known sequential step count for solving the corresponding problems. Processor efficient parallel algorithms for computing exact solutions of the above problems for arbitrary matrices have been found [KP91, KP92]. These algorithms require computation of the characteristic polynomial of a matrix generated from (but different from) the input matrix, and these computations are known to be numerically unstable in practice. More recently, Reif has given processor efficient parallel algorithms for solving the above problems based on matrix factorizations which are used extensively in sequential numerical computations and are essential in many applications [Rei94]. Reif also uses a similar approach to achieve processor efficient algorithms for Toeplitz and Toeplitz-like matrix computations, which are the first known processor efficient algorithms for exact Toeplitz and Toeplitz-like matrix computations [Rei95].

Reif's algorithms require the input matrix to have integer or rational entries. Thus the bit precision of the algorithms must be taken into consideration. It should be

"optimal" — the bit precision used by the algorithms should be asymptotically only as large as the bit precision required to represent the output. The algorithms use approximate Newton iterations. However, the analysis of approximate Newton iterations is not given in [Rei94, Rei95]. A complete analysis of approximate Newton iterations is desirable because these methods are being used to achieve fast algorithms more and more often. These techniques are used in general matrix computations as well as structured and sparse matrix computations. In this thesis, a complete analysis of approximate Newton iterations is given. The analysis shows that for a general matrix, only a "small" integer needs to be added to the diagonal entries to obtain a system that can be solved using these iterations. The analysis may also be applied to Toeplitz or Toeplitz-like matrix computations. This result significantly improves the bit precision of Reif's original algorithms and shows that it can be made optimal.

The results of the thesis will be introduced in the last section of this chapter. Before that, fundamental definitions and properties of matrix theory will be given and the computational model will be introduced.

1.1. Definitions

Basic definitions in matrix theory and asymptotic notations for complexity are introduced in this section.

1.1.1. Definitions in Matrix Theory. An $n \times m$ matrix is a matrix with n rows and m columns. The set of $n \times m$ matrices whose entries are chosen from a domain D is denoted as $D^{n\times m}$. For a matrix $A \in D^{n\times m}$, $A = [a_{ij}]$ denotes the matrix A with $a_{ij} \in D$ in the ith row and jth column of A, where $1 \leq i \leq n$ and $1 \leq j \leq m$.

Let $M_n = R^{n\times n}$, for some ring R. Let 0_n be the $n \times n$ matrix of 0's and let I_n be the $n \times n$ *identity matrix* with 1's on the main diagonal and 0's elsewhere. Let $+_n$ be matrix addition and $\times_n$ be matrix multiplication. Then it is easy to verify that $(M_n, +_n, \times_n, 0_n, I_n)$ is a ring, but it is not a commutative ring (unless $n = 1$ and R is commutative).

DEFINITION 1.1. *Let a matrix $A \in F^{n\times n}$ for some field F. The* determinant *of A, denoted* $\det(A)$, *is the sum over all $n!$ permutations σ of the integers 1 through n of the product*

$$(-1)^{k_\sigma} \prod_{i=1}^{n} a_{i\sigma(i)},$$

where k_σ is 0 if σ is even *(constructible from $(1, 2, \ldots, n)$ by an even number of interchanges) and k_σ is 1 if σ is* odd *(constructible by an odd number of interchanges).*

If $\det(A) \neq 0$, then A is *nonsingular* ; otherwise, A is *singular.*

DEFINITION 1.2. *An $m \times n$ matrix $A = [a_{ij}]$ is* upper triangular *if $a_{ij} = 0$ whenever $1 \leq j < i \leq m$. An $m \times n$ matrix A is* lower triangular *if $a_{ij} = 0$ whenever $1 \leq i < j \leq n$.*

DEFINITION 1.3. *A* submatrix *of a matrix A is a matrix obtained by deleting some rows and columns of A. A* principal submatrix *of an $n \times n$ matrix A is a square*

submatrix of A that consists of the first k rows of the first k columns of A, for $1 \leq k \leq n$.

DEFINITION 1.4. *The* transpose *of a matrix $A = [a_{ij}]$, denoted A^T, is the matrix formed by exchanging a_{ij} and a_{ji} for each i and j.*

Clearly $(AB)^T = B^T A^T$. Furthermore, $\det(A) = \det(A^T)$. A matrix A is *symmetric* if $A = A^T$.

DEFINITION 1.5. *Let a matrix $A \in F^{n \times n}$ for some field F. A pair of matrices $G, H \in F^{n \times d}$ is called* a generator of length d *of the matrix $A = GH^T$. The minimum length of a generator of A is called the* rank *of A, denoted* $\text{rank}(A)$.

DEFINITION 1.6. *Let a matrix $A \in F^{n \times n}$ for some field F. The* inverse *of A, denoted A^{-1}, is that $n \times n$ matrix, if it exists, such that $AA^{-1} = A^{-1}A = I_n$.*

It is also easy to verify that if A and B have inverses, then so does AB, and $(AB)^{-1} = B^{-1}A^{-1}$ in this case.

DEFINITION 1.7. *The* adjoint *matrix of A, denoted* $\text{adj}(A)$, *is the matrix whose entry in row i and column j is equal to $(-1)^{i+j}$ times the determinant of the submatrix obtained from A by removing row j and column i.*

For any square matrix A, $\text{adj}(A)$ exists and is unique. If A is nonsingular, then A^{-1} exists and is unique; furthermore, $A\text{adj}(A) = \text{adj}(A)A = \det(A)I$ and $A^{-1} = \text{adj}(A)/\det(A)$.

DEFINITION 1.8. *The* trace *of an $n \times n$ matrix A, denoted* $\text{tr}(A)$, *is the sum of the diagonal entries of A.*

DEFINITION 1.9. *The* characteristic polynomial *of an $n \times n$ matrix A is the polynomial $\psi(\lambda) = \det(\lambda I - A) = \sum_{i=0}^{n} c_i \lambda^i$, where $c_n = 1$.*

The characteristic polynomial of A includes the entries: $c_0 = (-1)^n \det A$, and $c_{n-1} = -\text{tr}(A)$.

DEFINITION 1.10. *An* eigenvalue *of a matrix A of* algebraic multiplicity *m is a zero of $\psi(\lambda)$ of multiplicity m.*

In the complex field, an $n \times n$ matrix has exactly n eigenvalues counted with their algebraic multiplicities.

DEFINITION 1.11. *The $n \times m$ matrix $K(a, v, m) = [v, Av, A^2v, \ldots, A^{m-1}v]$, defined for a matrix $A \in R^{n \times n}$ and a vector $v \in R^n$, for some ring R, is called an $n \times m$ Krylov matrix* [GVL90, BP94]. *Define $K(A, v) = K(A, v, n)$.*

DEFINITION 1.12. *A* Toeplitz *matrix is a matrix with the same entries along the diagonal and along each band that is parallel to the diagonal.*

Hence a square Toeplitz matrix has the form:

$$\begin{bmatrix} a_n & a_{n-1} & \cdots & a_1 \\ a_{n+1} & a_n & \cdots & a_2 \\ \vdots & \vdots & \ddots & \vdots \\ a_{2n-1} & a_{2n-2} & \cdots & a_n \end{bmatrix}.$$

DEFINITION 1.13. *A* Hankel matrix *is a matrix with the same entries along the antidiagonal and along each band that is parallel to the antidiagonal.*

Hence a square Hankel matrix has the form:

$$\begin{bmatrix} a_1 & \cdots & a_{n-1} & a_n \\ a_2 & \cdots & a_n & a_{n+1} \\ \vdots & \vdots & & \vdots \\ a_n & \cdots & a_{2n-2} & a_{2n-1} \end{bmatrix}.$$

If $T = [t_{ij}]$ is a Toeplitz matrix then $t_{ij} = t_{i+k,j+k}$, so that the matrix T is completely defined by its first row and its first column. Similarly, if $H = [h_{ij}]$ is a Hankel matrix then $h_{ij} = h_{i-k,j+k}$, so that the matrix H is completely defined by its first row and its last column.

The "lower-shift" matrix Z is the matrix

$$Z = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix},$$

with 1's on the band immediately below the diagonal and with 0's everywhere else.

As usual, $\mathbb{R}$ will denote the field of real numbers, $\mathbb{Q}$ will denote the field of rational numbers, and $\mathbb{Z}$ will denote the ring of integers.

DEFINITION 1.14. *A matrix $A \in \mathbb{R}^{n\times n}$ is* positive definite *if $x^T Ax > 0$ for all nonzero $x \in \mathbb{R}^n$.*

The proof of the following lemma can be found in [GVL90] (Theorem 4.2.1, Corollary 4.2.2, p. 140).

LEMMA 1.1. *Let $A \in \mathbb{R}^{n\times n}$ be positive definite. Let $Y \in \mathbb{R}^{n\times n}$ be nonsingular. Then*

(1) *A is nonsingular;*
(2) *$Y^T AY$ is positive definite;*
(3) *All the principal submatrices of A are positive definite;*
(4) *$Y^T Y$ is symmetric and positive definite.*

1.1.2. Matrix Norms. Norms serve the same purpose on vector spaces that the absolute value does on the real line: they furnish a measure of distance. The *p-norms* of a vector are functions $f : \mathbb{R}^n \to \mathbb{R}$, defined by

$$\|x\|_p = (|x_1|^p + \cdots + |x_n|^p)^{\frac{1}{p}}, \qquad p \geq 1.$$

For a matrix $A \in \mathbb{R}^{n\times n}$, define the p-norm of the matrix A as $\|A\|_p = \sup_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p}$. It can be shown that $\|A\|_1 = \max_j \sum_i |a_{ij}|$ and $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ [GVL90] (p. 56).

The following important fact can be found in [GVL90] (p. 57).

$$\|A\|_p / \sqrt{n} \leq \|A\|_2 \leq \|A\|_p \sqrt{n}, \tag{1.1}$$

for $p = 1, \infty$.

The p-matrix norms satisfy the three basic norm properties:

$$\begin{aligned} \|A\|_p \geq 0, &\qquad A \in \mathbb{R}^n \ (\|A\|_p = 0 \text{ iff } A = O); \\ \|A + B\|_p \leq \|A\|_p + \|B\|_p, &\qquad A, B \in \mathbb{R}^{n \times n}; \\ \|\alpha A\|_p = |\alpha| \, \|A\|_p, &\qquad \alpha \in \mathbb{R}, A \in \mathbb{R}^{n \times n}. \end{aligned}$$

It is easy to verify that the p-matrix norms also satisfy the *submultiplicative* property

$$\|AB\|_p \leq \|A\|_p \, \|B\|_p, \qquad A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times k}.$$

If $D = dI$, where d is a positive real number and I is the identity matrix, then clearly $\|D\|_p = d$ and $\|D^{-1}\|_p = 1/d$, for any $p = 1, 2, \infty$. If A is nonsingular, define the *p-condition number* of A, denoted by $\text{cond}_p(A)$, as $\text{cond}_p(A) = \|A\|_p \, \|A^{-1}\|_p$, for any p where $p = 1, 2, \infty$.

Matrices with small condition numbers are said to be *well-conditioned.* A matrix with nonzero diagonal entries is *row* (or, respectively, *column*) *diagonally dominant*, if $\|I - (\text{diag}(A))^{-1} A\|_\infty \leq c$ (or, respectively, if $\|I - A(\text{diag}(A))^{-1}\| \leq c$), for some constant c, $0 < c < 1$. A matrix A is *strongly diagonally dominant* if A is diagonally dominant for $c = 1 - 1/n_0^c$ for some constant $c_0 > 0$.

The determinant of a matrix A is bounded by the Hadamard inequality

$$|\det(A)| \leq \|A\|_p^n. \tag{1.2}$$

Assume $|a_{ij}| \leq 2^\beta$, for all i, j; then by Equation 1.1, $\|A\|_p \leq n 2^\beta$, for $p = 1$ or ∞, so that by Equation 1.2 $|\det(A)| \leq 2^\pi$, where $\pi = n(\beta + \log n)$.

1.1.3. Asymptotic Notation. To represent the efficiency of parallel algorithms the standard notation defined for the analysis of sequential algorithms is used. Con-

sider two functions $T, f : \mathbb{R} \to \mathbb{R}$ such that $T(n) \geq 0$ and $f(n) \geq 0$ for all n.

- $T(n) \in O(f(n))$ if there exist positive constants c and n_0 such that $T(n) \leq c(f(n))$, for all $n \geq n_0$.
- $T(n) \in \Omega(f(n))$ if there exist positive constants c and n_0 such that $T(n) \geq c(f(n))$, for all $n \geq n_0$.
- $T(n) \in \Theta(f(n))$ if $T(n) \in O(f(n))$ and $T(n) \in \Omega(f(n))$.
- $T(n) \in \tilde{O}(f(n))$ if $T(n) \in O(f(n)(\log n)^c)$, for some constant c.

1.1.4. Classes of Randomized Algorithms. Unlike a *deterministic algorithm*, which never returns different outputs when run twice on the same input, a *randomized algorithm* typically has access to a random number generator, and its output on one fixed input can be considered to be a random variable.

There are two commonly discussed classes of randomized algorithms: *Las Vegas* and *Monte Carlo*. A Las Vegas randomized algorithm will always either generate a correct output or report failure. A Monte Carlo algorithm will either return a correct output or an incorrect output — failure is not recognized. In both cases, for $\epsilon > 0$, it should be possible to guarantee that the probability of obtaining a correct answer exceeds $1-\epsilon$, by increasing the algorithm's cost by a factor that is polynomial (ideally, linear) in $\log(1/\epsilon)$.

1.2. Models of Computation

1.2.1. Sequential Model. A well-known sequential computational model is the *Random Access Machine* (RAM) model [AHU74]. The model assumes the presence of a central processing unit with a random-access memory attached to it, and some way to handle the input and the output operations.

The important measures on the RAM model of an algorithm are its time (and/or space) complexity, measured as functions of the size of input. To specify the time (and/or space) complexity exactly, it is necessary to specify the time of each operation and the size of each memory location. The *arithmetic* RAM model allows the operations on elements drawn from an arbitrary ring or field as primitive operations, which cost unit time. Each memory location has infinite size so that it can hold any element. The primitive operations are $+$, $-$, $\times$, $/$, mod, and a "zero test". This model is considered to be the standard sequential computational model.

The arithmetic RAM model does not reflect the dependence of the computational complexity of a problem on the precision of computing. In practice, the dependence is substantial. To measure time for computations over $\mathbb{Z}$ and $\mathbb{Q}$, one can measure time complexity based on the *logarithmic cost criterion.* The logarithmic cost criterion is based on the crude assumption that the cost of performing an operation is proportional to the length of the inputs. For example, to add two integers of magnitude $2^{O(n)}$, the cost is defined to be $O(n)$, whereas to add two integers of magnitude $2^{O(n^2)}$, the cost will be $O(n^2)$. Clearly under the logarithmic cost criterion, the bit precision of the elements involved in the computation will affect the time complexity substantially.

The computational model used by Reif [Rei94, Rei95] is an extended arithmetic RAM model with the following modifications.

(1) Real inputs are represented as floating point numbers.

(2) The model also allows one to randomly and uniformly select an integer from an interval $[M, N]$ for two given integers M, N, where $M < N$.

(3) The model includes a comparison ($\leq$) between two numbers as an operation.

(4) The model includes rounding a number to the nearest integer as an operation.

Unfortunately, Reif's model is not very clearly defined. It appears that the following is also required, although this is never explicitly stated:

(5) The model includes computation of an inverse modulo a prime. That is, given an integer x and a prime p that are relatively prime, one can find an integer y such that xy is congruent to one modulo p.

Section 4.4 of this thesis discusses the evidence for this additional requirement.

1.2.2. Parallel Model. The model for parallel computation to be used in this thesis is a natural extension of the above sequential model. In this model, many processors have access to a single shared memory unit. More precisely, the shared-memory model consists of a number of sequential processors. Each processor has its own local memory. The processors communicate by exchanging data through the shared memory unit. All processors operate synchronously under the control of a command clock. In one unit of time, each processor can read one global or local memory location, execute a single RAM operation, and write into one global or local memory location. This model is called the *parallel random access machine (PRAM) model.*

The parallel computation model in Reif's algorithms is an extended PRAM model because it is based on the above extended sequential RAM model.

The important measures on the PRAM model of the performance of an algorithm are its time and processor complexity. Suppose a PRAM algorithm runs in time $T(n)$ using $P(n)$ processors, for an instance of size n. The time-processor product $C(n) = T(n) \cdot P(n)$ represents the *cost* of the parallel algorithm. Since a single processor can simulate $P(n)$ processors in $O(P(n))$ time, for each of the $T(n)$ parallel steps, a parallel algorithm with cost $C(n)$ can be converted into a sequential algorithm that runs in $O(C(n))$ time.

Given $p \leq P(n)$ processors, p processors can simulate the $P(n)$ original processors in $\lceil P(n)/p \rceil < (P(n)/p) + 1$ substeps: in the first substep, the original processors numbered $1, 2, \ldots, p$ are simulated; in the second substep, processors numbered $p + 1, p + 2, \ldots, 2p$ are simulated; and so on. This entire simulation uses time at most $T(n) + \frac{T(n)P(n)}{p} \in O(T(n)P(n)/p)$ time.

Thus the following constraints on parallel algorithms are equivalent:

- the algorithm uses $O(P(n))$ processors and $T(n)$ time,
- the algorithm has cost $C(n) \in O(P(n)T(n))$ and runs in time $T(n)$,
- the algorithm uses time $O(T(n)P(n)/p)$ with $p \leq P(n)$ processors, for all $p \leq P(n)$.

Given a computational problem Q, let the sequential time complexity of Q be $T^*(n)$. A parallel algorithm to solve Q will be called *optimal* if the cost required by the algorithm is $\Theta(T^*(n))$.

There are several variations of the PRAM model based on the assumptions regarding the handling of the simultaneous access by several processors of the same location of global memory. The *exclusive read exclusive write (EREW)* PRAM does not allow any simultaneous access by multiple processors to a single memory location. The *concurrent read exclusive write (CREW)* PRAM allows simultaneous access for a read instruction only. Simultaneous access to a location for a read or a write is allowed in the *concurrent read concurrent write (CRCW)* PRAM. Though these three models differ in their computational power, it turns out that the difference will not affect the result in the thesis. Thus the thesis follows Reif's algorithms, assuming the PRAM model to be CRCW PRAM. Further information about the PRAM model can be in found, for example, in JáJá's text [JáJ92].

1.3. Problems and Preliminary Results

In this section, the problems in matrix computations discussed in this thesis are defined (also see [BP94]) and some preliminary results are given.

1.3.1. Problems in Matrix Computations.

PROBLEM 1.1. *Solving a nonsingular system of linear equations: Given a matrix* $A \in F^{n\times n}$ *and a vector* $b \in F^n$ *for some field* F, *compute the unique vector* $x = A^{-1}b$ *giving the solution to the linear system* $Ax = b$ *if the coefficient matrix* A *is nonsingular; otherwise, report that the coefficient matrix* A *is singular.*

PROBLEM 1.2. *Matrix inversion: Given a matrix* $A \in F^{n\times n}$ *for some field* F, *compute* A^{-1}, *the inverse of* A, *if* A *is nonsingular; otherwise, report that the matrix* A *is singular.*

PROBLEM 1.3. *Determinant: Given a matrix* $A \in R^{n\times n}$ *for some ring* R, *compute* $\det(A)$, *the determinant of the matrix* A.

PROBLEM 1.4. *Characteristic polynomial: Given a matrix* $A \in R^{n\times n}$ *for some ring* R, *evaluate the coefficients of the characteristic polynomial* $\psi(A) = \det(\lambda I - A) = \sum_{i=0}^{n} c_i\lambda^i$ *of the matrix* A.

The main focus in this thesis is parallel algorithms for solving the above problems efficiently. The following problems are given either because the algorithms in the thesis can be also applied to solve them efficiently or because an algorithm for solving the problem will be used as a subroutine.

PROBLEM 1.5. *Rank: Given a matrix* $A \in F^{m\times n}$, *for some integers* $m, n > 0$, *compute the rank of* A.

PROBLEM 1.6. *LU factorization: Given a matrix* $A \in F^{n\times n}$, *if possible, factor* $A = LU$, *where* L *is nonsingular and lower triangular, and* U *is nonsingular and upper triangular. Otherwise report that* A *has no LU factorization. (If* A *is symmetric and*

computations of square roots are allowed, then one can require that $U = L^T$. This problem is then known as Cholesky factorization).

PROBLEM 1.7. *QR factorization: Given a matrix $A \in \mathbb{R}^{n\times n}$, if possible, factor $A = QR$ where R is a nonsingular upper triangular matrix and Q is an orthogonal matrix ($Q^TQ = I$). If the QR factorization is not possible, then report that A has no QR factorization.*

PROBLEM 1.8. *Hessenberg reduction: Given a matrix $A \in \mathbb{R}^{n\times n}$, compute an orthogonal matrix Q and a matrix $H = [h_{ij}]$ such that $H = Q^TAQ$, $Q^TQ = I$ and such that $h_{ij} = 0$ if $i - j > 1$. If A is symmetric, then H is tridiagonal.*

PROBLEM 1.9. *Singular value decomposition (SVD): Given a matrix $A \in \mathbb{R}^{m\times n}$ compute a triple U, V, Σ such that*

$$A = U\Sigma V^T,$$

where $U \in \mathbb{R}^{m\times r}$, $V \in \mathbb{R}^{n\times r}$, $U^TU = I_r$, $V^TV = I_r$, $\Sigma = \mathrm{diag}(\sigma_1(A), \ldots, \sigma_r(A))$, and $\sigma_1(A) \geq \sigma_2(A) \geq \cdots \geq \sigma_r(A) > 0$, where $r = \mathrm{rank}(A)$. The values $\sigma_1(A), \ldots, \sigma_r(A)$ are called the singular values *of A.*

1.3.2. Complexity Bounds. The following results will be used throughout the thesis.

FACT 1.1. *Given an arbitrary square matrix $A \in R^{n\times n}$ and a vector $v \in R^n$ for some ring R, the product $u = Av$ can be computed in $O(\log n)$ time using $O(n^2/\log n)$ processors, which is optimal (see, for example,* [BM75]*).*

Given two square matrices $A, B \in R^{n\times n}$ for some ring R, their product can be computed in $O(\log n)$ time using $O(n^3)$ processors via the straightforward method. However, such a method is not optimal. The best known sequential bound for matrix multiplication is $O(n^\omega)$, for a real number ω such that $2 < \omega < 2.376$ [CW90], though

asymptotically slower algorithms are superior for computations one might expect to perform.

Many matrix computations have been shown to have the same sequential complexity as that of matrix multiplication. Computing the determinant, computing the inverse, solving a nonsingular linear system, and LU factorization are among these problems [AHU74]. Thus these problems can all be solved in time $O(M(n))$, where $M(n)$ is the number of the operations required to multiply two $n \times n$ matrices.

FACT 1.2. *Parallel matrix multiplication can be performed in* $O(\log n)$ *time using* $O(n^{\omega^*})$ *operations, for* $\omega < \omega^* < 2.376$ *(*[PR85]*, Appendix A).*

Hereafter let $P(n)$ denote a processor bound such that the bound $O(\log n)$ time and $O(P(n))$ processors holds for the parallel complexity of $n \times n$ matrix multiplication.

FACT 1.3. *Given a positive integer* m *and a square matrix* $A \in R^{n \times n}$ *for some ring* R*, the powers* $A^2, A^3, \ldots, A^m$ *can be computed in* $O(\log m \log n)$ *time using* $O(mP(n))$ *processors (see, for example,* [BP94]*).*

FACT 1.4. *Given a positive integer* m*, a square matrix* $A \in R^{n \times n}$*, and a vector* $v \in R^n$ *for some ring* R*, the Krylov matrix* $K(A, v, m)$ *can be computed in* $O(\log m \log n)$ *time using* $O(P(n))$ *processors (see* [BM75]*, p. 128).*

FACT 1.5. *The inverse of a triangular matrix* $A \in F^{n \times n}$ *for some field* F *can be computed in* $O(\log^2 n)$ *time using* $O(P(n))$ *processors (see* [BM75]*, p. 146).*

The inverse can be computed in $O(\log^2 n)$ time using $O(P(n))$ processors by a simple divide and conquer strategy. Assuming n is a power of 2, let A be an $n \times n$ lower triangular matrix partitioned as follows

$$A = \begin{bmatrix} B & 0 \\ C & D \end{bmatrix},$$

where B, C, D are of size $n/2 \times n/2$ and where B and D are also lower triangular. Then

$$A^{-1} = \begin{bmatrix} B^{-1} & 0 \\ -D^{-1}CB^{-1} & D^{-1} \end{bmatrix}.$$

Thus the following stages can be used to invert the matrix A efficiently.

(1) Invert B and D concurrently by the same strategy;

(2) Multiply $D^{-1}CB^{-1}$ and get A^{-1}.

This divide and conquer method will invert an $n \times n$ triangular matrix in $O(\log^2 n)$ time using $O(P(n))$ processors.

FACT 1.6. *The coefficients of the product of two polynomials of degree m and n respectively can be computed in $O(\log(n+m))$ time using $O((m+n))$ processors over a field supporting the Fast Fourier Transform (FFT) — that is, a field containing a kth primitive root of unity for $k \in \Theta(n+m)$ (see, for example,* [JáJ92]*).*

FACT 1.7. *Given two bivariate polynomials with degrees bounded by n for each variable, the coefficients of the product of the two polynomials can be computed in $O(\log n)$ time using $O(n^2)$ processors* [BP94].

FACT 1.8. *Given a positive integer m and a polynomial $T(z)$, $T(0) \neq 0$, the first K coefficients of the formal series $w(z)$ such that $w(z)T(z) = 1$ can be computed in $O(\log^2 K)$ time using $O(K)$ processors. When the coefficients of $w(z) \bmod w^{2^k}$ are given, the coefficients of $w(z) \bmod z^{2^{k+1}}$ can be computed in $O(k)$ time using (2^k) processors. (See, for example,* [BP94].*)*

The formal power series $w(z)$ of Fact 1.8 will be denoted $1/T(z)$.

An oracle PRAM is assumed to be a PRAM associated with a special shared memory for the (parallel) construction of oracle queries and for receiving the oracle's answers. It is assumed that each oracle query only costs unit time [Joh90] (p. 132).

DEFINITION 1.15. *An NC reduction from a problem A to a problem B is an oracle PRAM program that, given an oracle for B, solves A in polylog time using at most a polynomial number of processors. An efficient NC reduction from a problem A to a problem B is an NC reduction from A to B using $\hat{O}(Q(n))$ processors, where $Q(n)$ is a lower bound on the number of operations required to solve B sequentially, such that the sum of the sizes of all the values passed to the oracle is at most within a polylog factor of the size of the oracle PRAM's input.*

We write "$A \preceq B$" to denote the existence of an efficient NC reduction from A to B. The following can be found, for example, in [BP94].

FACT 1.9.

(1) *Solving a nonsingular linear system $\preceq$ Matrix inversion;*

(2) *Matrix inversion $\preceq$ Determinant;*

(3) *Determinant $\preceq$ Characteristic Polynomial.*

Csanky has given the first parallel algorithm for computing the determinant in $O(\log^2 n)$ time using a polynomial number of processors; his algorithm uses $O(nP(n))$ [Csa76] processors. Csanky's algorithm only works over fields of characteristic zero. Borodin, von zur Gathen and Hopcroft give an algorithm for this computation over arbitrary fields which uses far more processors than $O(nP(n))$ [BvzGH82]. Berkowitz [Ber84] and Chistov [Chi85] use different approaches to achieve parallel algorithms for computing the determinant in $O(\log^2 n)$ time using $O(nP(n))$ processors. For finding the rank, Mulmuley [Mul87] gives a fast parallel algorithm that works by computing the characteristic polynomial of a permuted matrix of the input with twice the size of the input matrix.

Kaltofen and Pan [KP91, KP92] give the first randomized processor efficient parallel algorithm for computing the determinant. They also give algorithms for inverting a nonsingular matrix, solving a nonsingular linear system, computing the rank, and computing a basis for the null space. All these algorithms use polylogarithmic time

and number of operations that is within a polylog factor of the amount needed for matrix multiplication.

Some related problems also have processor efficient parallel algorithms. Eberly [Ebe91] gives algorithms for computing a maximal linearly independent subset of a given set of vectors (see also [CR93, BP94]) and for computing a "PLU factorization" of a nonsingular matrix. Giesbrecht gives a processor efficient parallel algorithm for computing the Frobenius form and for computing the characteristic polynomial of a given matrix [Gie].

However, these fast parallel algorithms require computation of the characteristic polynomial of a matrix related to the input. These computations are known to be numerically unstable. For numerical matrix computations, Pan and Reif apply Newton iteration and its extensions to achieve processor efficient algorithms for computing approximate solutions for these problems [PR85, Pan87, PR89, PR93]. Newton iteration is generally quadratically convergent and numerically stable. However, these algorithms require that an initial approximation of the input matrix is given, or that the input matrix is *well-conditioned* so that an initial approximation of the input matrix can be computed. Their algorithms for matrix factorizations use $O(\log^3 n)$ parallel time. (When the input matrix is an integer matrix, the problems — solving a linear system, inverting a nonsingular matrix, and computing the determinant can be solved in $O(\log^2 n)$ time using $O(P(n))$ processors [Pan87], but the bit precision is greater than optimal.)

Reif gives a new parallel method for various exact factorizations of general dense matrices [Rei94]. His method can be further extended to block matrices, sparse separable matrices, and banded matrices. These methods reduce the previous known parallel time bounds for some matrix factorizations from $O(\log^3 n)$ to $O(\log^2 n)$, resolving an open question in [Pan87]. The exact factorizations that Reif's algorithms compute include recursive factorization sequences, LU factorizations, QR factorizations, and reduction into upper Hessenberg form. The method also provides algorithms for solv-

ing a linear system, inverting a nonsingular matrix, and computing the determinant. The algorithms assume the input matrices are square (of order n) with either integer entries of absolute value less than $2^{n^{O(1)}}$ or rational entries expressible as a ratio of integers of absolute value less than $2^{n^{O(1)}}$. Reif claims that the bit precision of the algorithm is "optimal" — that is, asymptotically only as large as the bit precision required to represent the output. However, it is not proved in Reif's paper and it appears that the precision used by Reif's algorithms is an $O(n)$ factor away from being optimal.

1.4. Thesis Results

This thesis provides a complete analysis of Reif's method for matrix factorizations of general dense matrices. The analysis reduces the bit precision significantly—achieving the asymptotically optimal bit precision claimed by Reif. Thus the results in this thesis provide evidence that Reif's algorithms can be made practical. The analysis is based mainly on the analysis of approximate Newton iteration, which may be of some independent interest. The thesis also shows some interesting properties of the recursive factorization sequence of a given matrix.

Reif has also given a new parallel method for Toeplitz and Toeplitz-like matrix computation and achieved the first processor efficient algorithms for exact Toeplitz and Toeplitz-like matrix computations [Rei95]. The thesis gives a brief introduction to that algorithm and shows that it is possible to improve the bit precision of this algorithm as well. However, certain steps remain difficult and unclear (as they exist in Reif's algorithms).

A brief survey of previous known parallel algorithms for general dense matrix computations is given in Chapter 2. Chapter 3 gives a complete analysis of approximate Newton iteration, which is entirely new. Reif's algorithm is analyzed in Chapter 4. Most of that material is also new. Toeplitz and Toeplitz-like matrix computations are briefly surveyed in Chapter 5. The idea to simplify Reif's algorithms for Toeplitz and Toeplitz-like matrix computations is also given in Chapter 5. Some part of that material is new. A conclusion of the thesis work is drawn and some further interesting problems are pointed out in Chapter 6.

CHAPTER 2
Parallel Matrix Computations

A brief survey on parallel algorithms for general dense matrix computations is given in this chapter. The survey includes the known deterministic approaches and randomized processor efficient approaches. These algorithms use exact arithmetic operations over a field as unit cost operations.

Some numerical algorithms for matrix computations are also introduced in this chapter. These algorithms have finite bit precision, so rounding errors will appear in the computation. These algorithms have better numerical stability than the exact algorithms given here.

2.1. Deterministic Parallel Matrix Computations

Csanky has given the first parallel algorithms for computing the determinant and inverse of a matrix, and solving a nonsingular system of linear equations in $O(\log^2 n)$ time, using $O(nP(n))$ processors [Csa76]. Csanky's algorithm only works over fields of characteristic zero or greater than n, where n is the order of the input matrix. The algorithm is based on a sequential method due to Leverrier for computing the characteristic polynomial. The algorithm has the following main steps:

(1) Compute $A^2, A^3, \ldots, A^{n-1}$;

(2) Compute the traces of the first n powers of A;

(3) Compute the coefficients $c_0, c_1, \ldots, c_{n-1}$ of the characteristic polynomial of A using the traces computed in the first step;

(4) Compute $\det(A) = (-1)^n c_0$ and $A^{-1} = -\sum_{i=1}^{n}(c_i/c_0)A^{i-1}$, if A is nonsingular.

The first step is known as the *matrix powers* problem, which can be solved in $O(\log^2 n)$ time using $O(nP(n))$ processors by Fact 1.3. The second step is straightforward. The fourth step is also easy since the powers A^i, for $1 \leq i \leq n-1$, have been computed. The third step is reduced to solving a triangular linear system, as shown below.

Suppose the characteristic polynomial is $\psi(\lambda) = \sum_{i=0}^{n} c_i \lambda^i$, where $c_n = 1$. The quantities $s_i = \mathrm{tr}(A^i)$, for $1 \leq i \leq n$, are available from Step (2). The coefficients c_i of the characteristic polynomial and the values s_k are related as shown by the following system of linear equations (see, for example, [Csa76]):

$$
\begin{pmatrix}
1 & 0 & 0 & \cdots & \cdots & & 0 \\
s_1 & 2 & 0 & \cdots & \cdots & & \vdots \\
s_2 & s_1 & 3 & \cdots & \cdots & & \vdots \\
s_3 & s_2 & s_1 & 4 & \cdots & & \\
\vdots & & \ddots & \ddots & \ddots & & \\
 & & & \ddots & \ddots & \ddots & \vdots \\
s_{n-1} & \cdots & & s_3 & s_2 & s_1 & n
\end{pmatrix}
\begin{pmatrix} c_{n-1} \\ c_{n-2} \\ c_{n-3} \\ \vdots \\ \vdots \\ c_1 \\ c_0 \end{pmatrix}
= -
\begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ \vdots \\ s_{n-1} \\ s_n \end{pmatrix}. \tag{2.1}
$$

Since the coefficient matrix is a lower triangular matrix, the system can be solved in $O(\log^2 n)$ time using $O(P(n))$ processors by Fact 1.5. Thus the cost of the whole algorithm is dominated by the first step, which requires $O(\log^2 n)$ time and $O(P(n))$ processors.

The above triangular linear system can be solved more efficiently than Fact 1.5 suggests — in $O(\log^2 n)$ time using $O(n/\log n)$ processors [Sch82, Pan90b]. The algorithm is shown below since it is required in the later sections of this thesis.

Let $g(z) = 1 + \sum_{i=1}^{n} c_{n-i} z^i = z^n \psi(\frac{1}{z})$, where $\psi(x)$ is the characteristic polynomial of A. Suppose $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the eigenvalues of A, so that $\psi(\lambda) = \sum_{i=0}^{n} c_i \lambda^i = \prod_{i=1}^{n} (\lambda - \lambda_i)$. Then $g(z) = z^n \psi(\frac{1}{z}) = z^n \prod_{i=1}^{n} (\frac{1}{z} - \lambda_i) = \prod_{i=1}^{n} (1 - z\lambda_i)$.

Now

$$g'(z) = \sum_{i=1}^{n} \left(\frac{g(z)}{1 - z\lambda_i} \cdot (-\lambda_i) \right).$$

If $(1 - z\lambda_i)^{-1}$ is replaced by the series $\sum_{k \geq 0} (z\lambda_i)^k$, where $1 \leq i \leq n$, then the following equation is obtained.

$$g'(z) = -g(z) \sum_{i=1}^{n} \left(\lambda_i \sum_{k \geq 0} (z\lambda_i)^k \right).$$

The above equation is rearranged as

$$\frac{g'(z)}{g(z)} = -\sum_{j=1}^{k} s_j z^{j-1} \bmod z^k, \qquad \text{for } k \geq 0, \tag{2.2}$$

using the fact that $s_j = \sum_i \lambda_i^j$.

Suppose the polynomial $g_r(z) = g(z) \bmod z^{r+1}$ has been computed. As shown below, it is possible to compute $g_{2r}(z) = g(z) \bmod z^{2r+1}$ in $O(\log r)$ time using $O(r)$ processors. The polynomials $g_1(z) = 1 + c_{n-1}(z)$ and $g_2(z) = 1 + c_{n-1} z + c_{n-2} z^2$ are readily available, because $c_{n-1} = -s_1$, and $2c_{n-2} = s_1^2 - s_2$. Let $g_{2r}(z)$ be expressed as

$$g_{2r}(z) = g_r(z)\,(1 + h_r(z)) \bmod z^{2r+1}, \tag{2.3}$$

where $h_r(z) = h_{r+1}z^{r+1} + \cdots + h_{2r}z^{2r}$ is an unknown polynomial. Then

$$g'_{2r}(z) = g'_r(z)(1 + h_r(z)) + g_r(z)h'_r(z) \bmod z^{2r}. \tag{2.4}$$

Now observe that

$$\frac{h'_r(z)}{1 + h_r(z)} = h'_r(z) \bmod z^{2r+1}, \tag{2.5}$$

because h_r is divisible by z^{r+1}. Also notice that

$$\frac{g'_{2r}(z)}{g_{2r}(z)} = \frac{g'_{2r}(z)}{g_r(z)\,(1 + h_r(z))} \bmod z^{2r+1}.$$

By Equation 2.4,

$$\frac{g'_{2r}(z)}{g_{2r}(z)} = \frac{g'_r(z)}{g_r(z)} + \frac{h'_r(z)}{1 + h_r(z)} \bmod z^{2r}. \tag{2.6}$$

Following Equations 2.2, 2.5, and 2.6,

$$\frac{g_r(z)}{g'_r(z)} + h'_r(z) = -\sum_{j=1}^{2r} s_j z^{j-1} \bmod z^{2r}. \tag{2.7}$$

Assume the coefficients of $g_r(z)$ are known, as well as the values s_j, for all $j \leq 2r+1$. Then $g_{2r}(z)$ can be computed as shown in step (2) of the following algorithm. This gives a recursive algorithm to compute $g(z)$ (since $g(z) = g_n(z) = g_{2^{\lceil \log n \rceil}}(z)$). The coefficients of $\psi(A)$ are immediately available from those of $g(z)$.

ALGORITHM 2.1. *Computing the coefficients of the characteristic polynomial.*

Input: The values of $s_k = \mathrm{tr}(A^k)$, $1 \leq k \leq n$, where A is an $n \times n$ matrix.

Output: The coefficients of the characteristic polynomial $\psi(\lambda) = \sum_{i=0}^{n} c_i \lambda^i$.

begin

step 1 Initialization:

$r := 1$;

$g_r(z) := 1 - s_1 z$;

step 2 While $r < n$ do (sequentially)

$u := 1/g_r(z) \bmod z^{2r}$;

(Comment: Recall that u is a polynomial including the leading terms of the Taylor series for $1/g_r(z)$.)

$v := g_r'(z)$;

$t := uv \bmod z^{2r}$;

$h_r' := -\sum_{j=1}^{2r+1} s_j z^{j-1} - t$;

$h_r := \int h_r'$;

(Comment: compute $h_r(z)$ by symbolic integration using the fact that $h_r(0) = 0$, and that the coefficients of h_r' are now available.)

$g_{2r}(z) = g_r(z)(1 + h_r(z)) \bmod z^{2r+1}$;

$r := 2r$;

end while

step 3 Output the coefficients of $g_n(z)$ with the order reversed.

end

LEMMA 2.1. *Given $s_j = \mathrm{tr}(A^j)$, for $1 \le j \le n-1$, where A is an $n \times n$ matrix over a field whose characteristic is zero or greater than n, the characteristic polynomial of the matrix A can be computed in $O(\log^2 n)$ time using $O(n/\log n)$ processors.*

PROOF. The algorithm uses the recursive method sketched above to compute the coefficients of the characteristic polynomial. Each recursive stage i requires computation of a polynomial reciprocal, polynomial multiplication, and polynomial addition with input polynomials of degree at most 2^i, $1 \le i \le \log n$. Polynomial multiplication and polynomial reciprocal computation can both be performed using an FFT in $O(\log 2^i)$ time and $O(2^i \log 2^i)$ operations. Thus the number of operations for Stage i is at most $c2^i \log 2^i$, for some constant $c > 0$ (independent of i), and the algorithm uses $O(\log^2 n)$ time and at most $2c \cdot 2^k \log 2^k \in O(n \log n)$ operations, where $k = \lceil \log n \rceil$. Thus the algorithm runs in $O(\log^2 n)$ time using $O(n/\log n)$ processors. □

The algorithm has a restriction which requires the ground field to be of characteristic zero or characteristic greater than n (see Equation 2.1). The number of processors can be reduced to $O(\sqrt{n}P(n))$ by using a similar approach [PS78, GP89]. These algorithms do not work for fields whose characteristic is positive and less than n. The best known deterministic parallel algorithms for computations over these fields require $O(nP(n))$ processors [Ber84, Chi85]. These last algorithms compute the characteristic polynomial of the input matrix without divisions. These results, together with Fact 1.9, imply the following theorem.

THEOREM 2.1. *There are deterministic parallel algorithms to evaluate the coefficients of the characteristic polynomial, to solve a linear system, and to compute the inverse and determinant of a matrix over any field in* $O(\log^2 n)$ *time using* $O(nP(n))$ *processors.*

2.2. Processor Efficient Algorithms

The field independent randomized sequential method for solving sparse linear systems given by Wiedemann reduces solving a linear system to finding the minimum generator of a linear recurrence [Wie86]. Kaltofen and Pan use the same approach but give a randomized parallel processor efficient reduction. Instead of trying to find the minimum generator of a linear recurrence, which so far has not been solved efficiently in parallel, Kaltofen and Pan use a reduction from computing the determinant to solving a nonsingular Toeplitz system based on the observation that when the degree of the minimum polynomial of the recurrence is known, the problems of finding the minimum generator of the recurrence and solving a nonsingular Toeplitz system of linear equations are equivalent.

Let V be a vector space over a field F. Let $\{a_i\}_{i=0}^{\infty}$ be an infinite sequence with elements $a_i \in V$. The sequence $\{a_i\}_{i=0}^{\infty}$ is *linearly generated* over F if there exist $c_0, c_1, \ldots, c_n \in F$, for some $n \geq 0$, such that

$$c_0 a_j + \cdots + c_n a_{j+n} = 0, \qquad \forall j \geq 0.$$

The polynomial $c_0 + c_1 x + \cdots + c_n x^n$ is called a *generating polynomial* for $\{a_i\}_{i=0}^{\infty}$. The set of all generating polynomials for any sequence, taken together with the zero polynomial, forms an ideal in $F[x]$. The unique polynomial generating that ideal, normalized to have leading coefficient 1, is called the *minimum polynomial* of the linearly generated sequence $\{a_i\}_{i=0}^{\infty}$. Every generating polynomial for $\{a_i\}_{i=0}^{\infty}$ is a multiple of the minimum polynomial for this sequence. Let $A \in F^{n \times n}$ be a square matrix over some field F. The sequence $\{A^i\}_{i=0}^{\infty}$ is linearly generated, and its minimum polynomial is the *minimum polynomial* of A, which is denoted by f^A. For any column vector $b \in F^n$, the sequence $\{A^i b\}_{i=0}^{\infty}$ is also linearly generated. The minimum polynomial of $\{A^i b\}_{i=0}^{\infty}$, denoted by $f^{A,b}$, is a divisor of f^A. Finally for any row vector $u \in F^{1 \times n}$, the sequence $\{uA^i b\}_{i=0}^{\infty}$ is linearly generated as well, and its minimum polynomial, denoted by $f_u^{A,b}$, is a divisor of $f^{A,b}$.

Let $\{a_i\}_{i=0}^{\infty}$ be linearly generated and let n be the degree of its minimum polynomial. For $m \geq 0$, construct the Toeplitz matrix

$$T_m = \begin{pmatrix} a_{m-1} & a_{m-2} & \cdots & a_1 & a_0 \\ a_m & a_{m-1} & \cdots & a_2 & a_1 \\ \vdots & a_m & \ddots & \vdots & a_2 \\ & \vdots & & & \vdots \\ a_{2m-3} & & & a_{m-1} & \\ a_{2m-2} & a_{2m-3} & \cdots & a_m & a_{m-1} \end{pmatrix} \in F^{m \times m}. \tag{2.8}$$

Then $\det(T_n) \neq 0$ and for all $m > n$, $\det(T_m) = 0$ (see Lemma 1 in [KP91]).

Furthermore, let S be a finite subset of F. Uniformly and randomly choose a row vector $u \in S^{1 \times n}$ and a column vector $b \in S^n$. If S is sufficiently large, then the probability that the minimum polynomial of $\{A^i b\}_{i=0}^{\infty}$ equals the minimum polynomial of A is high (Lemma 2 in [KP91]):

$$\Pr(f_u^{A,b} = f^A) \geq 1 - \frac{2\deg(f^A)}{|S|}.$$

Wiedemann shows that if the given matrix $\tilde{A}$ has the property that all its principal submatrices are nonsingular, and if

$$\hat{A} = \tilde{A}D, \qquad D = \mathrm{diag}(d_1, \ldots, d_n),$$

where d_i are uniformly and independently selected from S, then the probability that the characteristic polynomial of $\hat{A}$ equals the minimum polynomial of $\hat{A}$ is high:

$$\Pr(f^{\hat{A}}(\lambda) = \det(\lambda I - \hat{A})) \geq 1 - \frac{n(2n-2)}{|S|}.$$

For an arbitrary nonsingular matrix A, let

$$\tilde{A} = AH, \quad H = \begin{pmatrix} h_0 & h_1 & \cdots & h_{n-2} & h_{n-1} \\ h_1 & \cdots & \cdots & h_{n-1} & h_n \\ \vdots & & & & \vdots \\ h_{n-1} & h_n & \cdots & & h_{2n-2} \end{pmatrix},$$

where the elements of the Hankel matrix H are randomly and uniformly selected from the set S. Then the probability that all the principal submatrices of $\tilde{A}$ are nonsingular is also high:

$$\Pr(\text{All the principal submatrices of } \tilde{A} \text{ are nonsingular}) \geq 1 - \frac{n(n+1)}{2\,|S|}.$$

Kaltofen and Pan's algorithm is shown below. It reduces computing the determinant of an arbitrary matrix to solving a nonsingular Toeplitz system, to computing the determinant of a nonsingular Hankel matrix, and to matrix multiplication. Algorithms for solving a nonsingular Toeplitz system in parallel will be given in Chapter 5. These algorithms can also be extended to compute the determinant of a nonsingular Hankel matrix.

ALGORITHM 2.2. *Computing the Determinant*

Input: An $n \times n$ nonsingular matrix A

Output: $\det(A)$

begin Pick a random Hankel matrix H, a random diagonal matrix D, a random row vector u, and a random column vector b, all with entries in S.

step 1
(a) Compute $\tilde{A} = AHD$.
(b) Compute $\{\tilde{A}b, \tilde{A}^2b, \ldots, \tilde{A}^{2n-1}b\}$.
(c) Compute the sequence $\{a_0, \cdots, a_{2n-1}\}$, where $a_i = u\tilde{A}^ib$.

step 2 Solve the Toeplitz system

$$\begin{pmatrix} a_{n-1} & a_{n-2} & \cdots & a_1 & a_0 \\ a_n & a_{n-1} & \cdots & a_2 & a_1 \\ \vdots & a_n & \ddots & \vdots & a_2 \\ & \vdots & & & \vdots \\ a_{2n-3} & & & a_{n-1} & \\ a_{2n-2} & a_{2n-3} & \cdots & a_n & a_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ \\ \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} -a_n \\ -a_{n+1} \\ \vdots \\ \\ \\ -a_{2n-2} \\ -a_{2n-1} \end{pmatrix}.$$

If the system is singular, then report failure and halt. Otherwise, the coefficients of the minimum polynomial of the sequence $\{a_i\}_{i=0}^{\infty}$ have been computed. Since the minimum polynomial of the sequence is also the characteristic polynomial of $\tilde{A}$, the coefficients of the characteristic polynomial of $\tilde{A}$ are obtained such that $\psi(\tilde{A}) = \sum_{i=0}^{n} c_i \lambda^i$, where $c_i = x_i$ for $0 \leq i \leq n-1$.

step 3 Compute the determinant of A using the formula

$$\det(A) = \frac{f_u^{\tilde{A},b}(0)}{\det(H)\det(D)}.$$

end

The above result and the Fact 1.9 imply the following theorem [KP91].

THEOREM 2.2. *Given a nonsingular matrix $A \in F^{n\times n}$, for some field F of characteristic zero or greater than n, and a finite subset S of F, there exists a randomized algorithm (Las Vegas) for computing* $\det(A)$, *solving a linear system $Ax = b$, and evaluating A^{-1} in $O(\log^2 n)$ time using $O(P(n))$ processors. The algorithm outputs the correct answer with probability at least $1 - 3n^2/|S|$, and reports failure otherwise.*

For matrix computations over fields of small positive characteristic, Kaltofen and Pan also achieve randomized processor efficient parallel algorithms with both running time and number of processors increased by at most an $O(\log n)$ factor.

Kaltofen and Pan also gave processor efficient parallel algorithms for computing the rank and finding a basis for the nullspace of a matrix.

2.3. Rational and Integer Matrix Computations

The parallel algorithms for matrix inversion and related problems in the previous sections require computation of the characteristic polynomial or related forms (the minimum polynomial etc.), which are known to be highly unstable in practice [Wil65]. If the calculations are not performed using exact arithmetic, then their outputs may substantially differ from A^{-1}. The algorithms discussed in this section have better numerical stability.

Henceforth, $\|A\|$ will denote the p-norm $\|A\|_p$ of A, and $\mathrm{cond}(A)$ will denote the $\mathrm{cond}_p(A)$, for some $p \geq 1$. There are iterative methods for computing the inverse of a well-conditioned matrix [PR85, PR89, PR93]. These iterative methods are based on Newton iteration and its extensions. Given a matrix A, the Newton iteration method requires an initial approximate inverse B such that $\|I - AB\|$ is substantially less than 1. For a general matrix, there is no known technique to find such an initial approximate inverse; however, for a strictly diagonally dominant matrix A, an approximate inverse B of A can be a diagonal matrix consisting of the inverses of the diagonal entries of A. These iterative algorithms compute A^{-1} (up to error $2^{-n^{O(1)}}$) using $O(\log n)$ matrix multiplications (in the stages of Newton iterations) and hence in $O(\log^2 n)$ time using $O(P(n))$ processors.

The above results have been extended to the exact evaluation of the inverse and the determinant in $O(\log^2 n)$ time using $O(P(n))$ processors when the input matrix is an arbitrary integer matrix such that $\|A\| \leq 2^{\beta}$, where $\beta \leq n^c$, for some constant c [Pan85, Pan87]. The bit precision of these algorithms is not optimal.

Recent results by Reif [Rei94] show that a similar method can be applied to the exact evaluation of LU factors of a given matrix in $O(\log^2 n)$ time using $O(P(n))$ processors when the input matrix is an arbitrary integer matrix such that $\|A\| \leq 2^{\beta}$, where $\beta \leq n^c$, for some constant c. Thus the algorithms are available to compute the inverse, determinant, and various factorizations in $O(\log^2 n)$ time using $O(P(n))$ processors.

2.3.1. Inversion of Well-conditioned Matrices. Pan and Reif show that it takes $O(\log^2 n)$ time and $O(P(n))$ processors to compute the inverse (up to error $2^{-n^{O(1)}}$) of a well-conditioned matrix [PR85, PR89, PR93].

Given a matrix $A \in \mathbb{R}^{n\times n}$, a matrix B is called an *approximate inverse* of A if $\|I - AB\| = \epsilon < 1$. Let $B^{(0)} = B$ and $B^{(k+1)} = B^{(k)}(2I - AB^{(k)})$, for $k \geq 1$. This is exactly the Newton iteration applied to the equation $R(B) = I - AB = 0$. It can be shown that $B^{(k)}$ converges to A^{-1} quadratically, because

$$\begin{aligned}\left\|I - B^{(k)}A\right\| &= \left\|I - B^{(k-1)}\left(2I - AB^{(k-1)}\right)A\right\| \\ &= \left\|\left(I - B^{(k-1)}A\right)^2\right\| \\ &\leq \left\|I - B^{(k-1)}A\right\|^2.\end{aligned}$$

Therefore, if $\left\|I - B^{(0)}A\right\| \leq \epsilon$, then

$$\left\|I - B^{(k)}A\right\| \leq \epsilon^{2^k}.$$

It can also be shown that $\|A^{-1}\| \leq \|B\|/(1-\epsilon)$, because

$$\begin{aligned}\left\|A^{-1}\right\| &\leq \left\|A^{-1} - B\right\| + \|B\| \\ &\leq \left\|A^{-1}\right\| \|I - AB\| + \|B\| \\ &\leq \epsilon\left\|A^{-1}\right\| + \|B\|.\end{aligned}$$

Thus

$$\left\|A^{-1} - B^{(k)}\right\| \leq \epsilon^{2^k}\left\|A^{-1}\right\| \leq \epsilon^{2^k}\|B\|/(1-\epsilon).$$

Let $B = B^{(0)}$ satisfy the inequality

$$\|I - AB\| = \epsilon = 1 - 1/n^{O(1)}. \tag{2.9}$$

It can be shown that if $|\log\|B\|| \leq n^{O(1)}$, then $O(\log n)$ iterations, which take $O(\log^2 n)$ time and $O(P(n))$ processors, suffice in order to compute a matrix $\tilde{A}^{-1}$

such that

$$\left\|A^{-1} - \tilde{A}^{-1}\right\| \leq 2^{-n^c} \|B\|,$$

for any fixed constant c.

Given a nonsingular matrix A, let $B = tA^T$, where $t = 1/(\|A\|_1 \|A\|_\infty)$. It can be shown that $\|B\| \leq 1/\|A\|$ and $\|I - AB\| \leq 1 - 1/((\text{cond}(A))^2 n)$ [PR85] (Lemma 2.4). Thus when the matrix A is well-conditioned, say $\text{cond}(A) \leq n^{O(1)}$, an approximate inverse B can be computed in $O(\log n)$ time using $O(n^2/\log n)$ processors. Thus the algorithm only uses $(\log^2 n)$ time and $O(P(n))$ processors.

2.3.2. Exact Solution of Integer Matrix Computations. Pan shows that the problems of solving a linear system, evaluating the determinant, inverse, and computing the coefficients of the characteristic polynomial can be solved exactly by computing numerical approximations and rounding them to the nearest integers provided that the entries of the input matrix (and vector) are integers [Pan85, Pan87].

Given an integer matrix A, assume $\|A\| = 2^\beta$, where $\beta \leq n^c$, for some constant c, so $\log\log\|A\| = O(\log n)$.

Let q be an integer such that $q > 3n^2\|A\|$. Let $P = [p_{ij}]$ be the $n \times n$ cyclic permutation matrix such that

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

Let $V = PA + qP$. Let $v = (1, 0, \ldots, 0)^T$. It can be shown that the Krylov matrix $K(V, v) = [v, Vv, \ldots, V^{n-1}v]$ is strongly diagonally dominant. Thus $K(V, v)$ is nonsingular and the minimum polynomial of the sequence $\{V^i\}_{i=0}^\infty$ has degree at least n. Since the characteristic polynomial is a generating polynomial of the sequence $\{V^i\}_{i=0}^\infty$ and has degree n, the characteristic polynomial is the minimum polynomial of $\{V^i\}_{i=0}^\infty$. Let the characteristic polynomial of V be $\psi(V) = \det(\lambda I - V) = \sum_{i=0}^n c_i \lambda^i$ and let the vector $c(V) = [-c_0, -c_1, \ldots, -c_{n-1}]^T$ be the coefficient vector of of $\psi(V)$.

Then

$$K(V, v)c(V) = V^n v.$$

Since $K(V, v)$ is strongly diagonally dominant, it is well-conditioned. Apply the algorithm for computing the inverse (with high precision) of a well-conditioned matrix, then the exact values of $c(V) = c(A) \bmod q$ can be obtained by rounding the entries of $c(V)$ to the nearest integers.

It is necessary (and sufficient) that $q > 2(\|A\|)^n$ if $c(A)$ is to be recovered from the above residue, $c(A) \bmod q$. In this case $\det(A)$ is immediately available. Since $P^T V = A + qI$ is strongly diagonally dominant and hence well-conditioned, one can apply the algorithm for computing the inverse (with high precision) to compute $(P^T V)^{-1}$. The determinant of $P^T V$ is also available because $\det(P^T V) = \det(P^T)\det(V) = (-1)^{n-1} c_n(V)$. Thus $\mathrm{adj}(P^T V) = (P^T V)^{-1} \det(P^T V)$ can be computed. From $\mathrm{adj}(P^T V)$ it is easy to compute $\mathrm{adj}(A)$ via reduction modulo q. Finally $A^{-1} = \mathrm{adj}(A)/\det(A)$ can be computed.

Notice that the exact computation of $\det(A)$ by this method requires $q > 2\|A\|^n$, so the computation involves the use of $n^2 \log \|A\|$-bit integers.

2.3.3. Matrix Factorizations. Reif shows a method using recursive factorization to achieve exact solutions for integer matrices [Rei94]. The method can be extended to achieve new algorithms for various factorizations of a general matrix as well as some special classes of matrices. Reif's method and an improved version will be given in Chapter 4.

CHAPTER 3
Matrix Norms and Bounds

Some properties of matrix norms are given in this chapter. Approximate Newton iteration, a modified version of the standard Newton iteration, is introduced. Unlike the Newton iterative methods used to compute a numerical approximation of a matrix inverse, approximate Newton iteration does not require the exact input. It is shown that under certain conditions, approximate Newton iteration also produces a sequence of numerical approximations which converges quadratically.

3.1. Approximate Newton Iteration and Norm Properties

Some properties of matrix norms are given in this section. Let $A \in \mathbb{R}^{n\times n}$ be a nonsingular matrix. Let $d \in \mathbb{R}$ and $D = dI$, where I is the $n \times n$ identity matrix. Assume $d > 0$ hereafter. Let $\bar{A} = D + A$. For sufficiently large d, the matrix $\bar{A}$ is diagonally dominant, and D^{-1} is in some sense a "good approximation" for $\bar{A}^{-1}$. It will be shown in Chapter 4 that the inverse (and adjoint) of $\bar{A}$ can be of use in computing the inverse of A, as well. The lemmas given in this chapter will be applied to analyze the algorithms given in Chapter 4. The matrix norm $\|A\|$ will be used to denote $\|A\|_p$, where $p = 1, 2, \infty$.

3.1.1. Basic Norm Bounds. It is easy to verify the following facts using the properties of matrix norms.

- $\|A\| = \|-A\|$, for any matrix A.
- $\|D\| = d$ and $\|D^{-1}\| = 1/d$.
- $\left\|\bar{A} - D\right\| = \|A\|$ and $\left\|\bar{A}\right\| \leq \|A\| + d$.
- $\left\|I - D^{-1}\bar{A}\right\| = \|D^{-1}A\| = \|A\|/d$.
- If $d \geq \|A\|$, then $\left|\det(\bar{A})\right| \leq (2d)^n$.
- If $A = [a_{ij}]$ then $|a_{ij}| \leq \|A\|$, for $1 \leq i, j \leq n$.

LEMMA 3.1. *If* $d \geq 2\|A\|$, *then* $\left\|\bar{A}^{-1}\right\| \leq 2/d$.

PROOF. It is given that $d \geq 2\|A\|$. Thus $\|D^{-1}A\| \leq \frac{1}{2}$. According to a well known result for matrix norms ([GVL90], Theorem 2.3.4, p.59),

$$\left\|\bar{A}^{-1} - D^{-1}\right\| \leq \frac{\|A\|\,\|D^{-1}\|^2}{1 - \|D^{-1}A\|}.$$

A modification of the right side of the inequality shows the following.

$$\left\|\bar{A}^{-1} - D^{-1}\right\| \leq \frac{\|A\|}{d(d - \|A\|)} \leq \frac{1}{d}.$$

The lemma follows, since $\|D^{-1}\| = 1/d$. □

A careful observation shows that when d is sufficiently large, both $\left\|\bar{A}\right\|$ and $\left\|\bar{A}^{-1}\right\|$ are mainly affected by the choice of d. The norm $\left\|\bar{A}-D\right\|$ shows how close $\bar{A}$ is to a diagonal matrix. The norm $\left\|\bar{A}^{-1}-D^{-1}\right\|$ shows how close $\bar{A}^{-1}$ is to a diagonal matrix. The following lemma shows when A is close to a diagonal matrix D, A^{-1} is close to D^{-1}.

LEMMA 3.2. *If* $\left\|\bar{A}-D\right\| \leq d/k$ *for some* $k>1$, *then*

(1) $\left\|\bar{A}^{-1}-D^{-1}\right\| \leq 1/((k-1)d)$;

(2) $\left\|\bar{A}^{-1}\right\| \leq k/((k-1)d)$;

(3) *if* $k \geq 2$, *then* $\left\|\bar{A}^{-1}\right\| \leq 2/d$.

PROOF. Since $\left\|D^{-1}(\bar{A}-D)\right\| \leq 1/k < 1$, the result of [GVL90] (Theorem 2.3.4, p. 59) again implies that

$$\begin{aligned}\left\|\bar{A}^{-1}-D^{-1}\right\| &\leq \frac{\left\|\bar{A}-D\right\|\|D^{-1}\|^2}{1-\left\|D^{-1}(\bar{A}-D)\right\|} \\ &\leq \frac{1}{d(k-1)}.\end{aligned}$$

The lemma follows from the equality $\|D^{-1}\| = 1/d$. □

3.1.2. Determinant and Matrix Norms. The Hadamard inequality (Equation 1.2) shows the relation between matrix norms and the determinant.

LEMMA 3.3. *If* $\|A-B\| \leq \epsilon$, *where* $\epsilon \geq 0$ *and* $\|A\|, \|B\| \leq z$, $z>0$, *then* $|\det(A)-\det(B)| \leq (nz)^n\epsilon/z$.

PROOF. Let $A=[a_{ij}]$ and $B=[b_{ij}]$. Then $|a_{ij}-b_{ij}| \leq \|A-B\| \leq \epsilon$. By the definition of determinant, the following is easy to check.

$$|\det(A)-\det(B)| \leq \sum_{\sigma}\left|\prod_{i=1}^{n} a_{i\sigma(i)} - \prod_{i=1}^{n} b_{i\sigma(i)}\right|,$$

where the sum runs over all $n!$ permutations σ of the n items $\{1,\ldots,n\}$. Since $\prod_{i=1}^{n} a_{i\sigma(i)} - \prod_{i=1}^{n} b_{i\sigma(i)} = \sum_{i=1}^{n} \left(\prod_{j=1}^{i-1} b_{i\sigma(i)}\right) \left(a_{i\sigma(i)} - b_{i\sigma(i)}\right) \left(\prod_{j=i+1}^{n} a_{i\sigma(i)}\right)$,

$$\left|\prod_{i=1}^{n} a_{i\sigma(i)} - \prod_{i=1}^{n} b_{i\sigma(i)}\right| \leq \sum_{i=1}^{n} z^{n-1}\epsilon = nz^{n-1}\epsilon.$$

Thus

$$|\det(A) - \det(B)| \leq n!\left(nz^{n-1}\epsilon\right) \leq (nz)^n \epsilon/z.$$

□

3.1.3. Exact Newton Iteration. Let $B^{(0)}, B^{(1)}, \ldots$ be a sequence of matrices that are approximations of the matrix $\bar{A}^{-1}$ generated by Newton iterations starting with some initial estimate $B^{(0)}$, where $B^{(k)} = B^{(k-1)}(2I - \bar{A}B^{(k-1)})$. The sequence converges quadratically. This follows since

$$\begin{aligned} I - B^{(k)}\bar{A} &= I - B^{(k-1)}(2I - \bar{A}B^{(k-1)})\bar{A} \\ &= (I - B^{(k-1)}\bar{A})^2. \end{aligned}$$

Thus

$$\left\|I - B^{(k)}\bar{A}\right\| \leq \left\|I - B^{(k-1)}\bar{A}\right\|^2. \tag{3.1}$$

If $\left\|I - B^{(0)}\bar{A}\right\| = \epsilon < 1$, then it follows that

$$\left\|I - B^{(k)}\bar{A}\right\| \leq \epsilon^{2^k}.$$

Since $\bar{A}$ is given exactly, this Newton iteration is called *exact Newton iteration.*

3.1.4. Approximate Newton Iteration. Suppose the matrix $\bar{A}$ such that

$$\left\|\bar{A} - D\right\| \leq d\epsilon/2 \tag{3.2}$$

is not given, but that a sequence of approximations of $\bar{A}$ is given instead. Suppose the sequence $\tilde{A}^{(0)} = D, \tilde{A}^{(1)}, \ldots, \tilde{A}^{(k)}$ satisfies the inequalities

$$\left\|\tilde{A}^{(k)} - \bar{A}\right\| \leq \frac{d}{2}\epsilon^{2^k}, \tag{3.3}$$

and

$$\left\|\tilde{A}^{(k)} - \tilde{A}^{(k+1)}\right\| \leq \frac{d}{2}\epsilon^{2^k}, \tag{3.4}$$

where $0 \leq \epsilon \leq \frac{1}{2}$. A sequence $\tilde{B}^{(0)}, \tilde{B}^{(1)}, \tilde{B}^{(2)}, \ldots, \tilde{B}^{(k)}$ of approximations of $\bar{A}^{-1}$ is computed as follows: $\tilde{B}^{(0)} = D^{-1}$ and $\tilde{B}^{(k)} = \tilde{B}^{(k-1)}(2I - \tilde{A}^{(k)}\tilde{B}^{(k-1)})$. Since a sequence of approximations of $\bar{A}$ is used instead of $\bar{A}$ itself, this is called *approximate Newton iteration.*

LEMMA 3.4. *If* $0 \leq \epsilon \leq \frac{1}{2}$, *then* $\left\|I - \tilde{B}^{(k)}\tilde{A}^{(k+1)}\right\| \leq \frac{1}{2}(2\epsilon)^{2^k}$, *and* $\left\|\tilde{B}^{(k)}\right\| \leq 2/d$.

PROOF. It is given that $\tilde{B}^{(0)} = D^{-1}$ and $\left\|\tilde{A}^{(1)} - \bar{A}\right\| \leq d\epsilon^2/2$. Thus

$$\begin{aligned}
\left\|I - \tilde{B}^{(0)}\tilde{A}^{(1)}\right\| &\leq \left\|I - \tilde{B}^{(0)}\bar{A}\right\| + \left\|\tilde{B}^{(0)}\right\|\left\|\bar{A} - \tilde{A}^{(1)}\right\| \\
&\leq \frac{\left\|\bar{A} - D\right\|}{d} + \frac{1}{d}\left\|\bar{A} - \tilde{A}^{(1)}\right\| \\
&\leq \frac{\epsilon}{2} + \frac{\epsilon^2}{2} \leq \epsilon,
\end{aligned}$$

since $0 \leq \epsilon \leq 1$. Also $\left\|\tilde{B}^{(0)}\right\| = 1/d \leq 2/d$.

Suppose the claim in the lemma holds for $k-1$, so that

$$\left\|I - \tilde{B}^{(k-1)}\tilde{A}^{(k)}\right\| \leq \frac{1}{2}(2\epsilon)^{2^{k-1}},$$

where $k \geq 1$. Since $0 \leq \epsilon \leq \frac{1}{2}$, $\left\|I - \tilde{B}^{(k-1)}\tilde{A}^{(k)}\right\| \leq \frac{1}{2}$. It follows from the conditions on $\bar{A}$ and $\{\tilde{A}^{(k)}\}$ that

$$\left\|\tilde{A}^{(k)} - D\right\| \leq \left\|\tilde{A}^{(k)} - \bar{A}\right\| + \left\|\bar{A} - D\right\| \leq \frac{d}{2}\epsilon^{2^k} + \frac{d\epsilon}{2} \leq \frac{3d}{8},$$

for $k \geq 1$. It follows by Part (2) of Lemma 3.2 that $\left\|(\tilde{A}^{(k)})^{-1}\right\| \leq 8/(5d)$.

Since $\hat{B}^{(k)} = \hat{B}^{(k-1)}(2I - \hat{A}^{(k)}\hat{B}^{(k-1)})$,

$$\begin{aligned}
\left\|\hat{B}^{(k)}\right\| &\leq \left\|\hat{B}^{(k-1)}(2I - \hat{A}^{(k)}\hat{B}^{(k-1)})\hat{A}^{(k)}\right\| \left\|(\hat{A}^{(k)})^{-1}\right\| \\
&\leq \left\|-(I - \hat{B}^{(k-1)}\hat{A}^{(k)})^2 + I\right\| \left\|(\hat{A}^{(k)})^{-1}\right\| \\
&\leq \left(1 + \left\|I - \hat{B}^{(k-1)}\hat{A}^{(k)}\right\|^2\right) \left\|(\hat{A}^{(k)})^{-1}\right\| \\
&\leq \left(1 + \left(\frac{1}{2}\right)^2\right) \frac{8}{5d} = \frac{2}{d}.
\end{aligned}$$

With the hypothesis that the claim holds for $k-1$,

$$\begin{aligned}
\left\|I - \hat{B}^{(k)}\hat{A}^{(k+1)}\right\| &\leq \left\|I - \hat{B}^{(k)}\hat{A}^{(k)}\right\| + \left\|\hat{B}^{(k)}(\hat{A}^{(k)} - \hat{A}^{(k+1)})\right\| \\
&\leq \left\|I - \hat{B}^{(k-1)}\hat{A}^{(k)}\right\|^2 + \frac{2}{d}\left\|\hat{A}^{(k)} - \hat{A}^{(k+1)}\right\| \\
&\leq \left(\frac{1}{2}(2\epsilon)^{2^{k-1}}\right)^2 + \frac{2}{d}\frac{d}{2}\epsilon^{2^k} \\
&\leq \left(2^{2^k-2} + 1\right)\epsilon^{2^k} \leq 2^{2^k-1}\epsilon^{2^k},
\end{aligned}$$

since $2^{2^k-2} \geq 1$ when $k \geq 1$. Thus the lemma is proved by induction on k. □

LEMMA 3.5. *Suppose again that* $0 \leq \epsilon \leq \frac{1}{2}$. *Let* $\bar{A}$ *and* $\hat{A}^{(k)}$ *satisfy Equations* 3.2–3.4. *Let* $\tilde{B}^{(k)}$ *be computed as above. Then* $\left\|I - \tilde{B}^{(k)}\bar{A}\right\| \leq (2\epsilon)^{2^k}$, *where* $0 \leq \epsilon \leq \frac{1}{2}$.

PROOF. From Lemma 3.4, it immediately follows that

$$\begin{aligned}
\left\|I - \tilde{B}^{(k)}\bar{A}\right\| &\leq \left\|I - \tilde{B}^{(k)}\tilde{A}^{(k+1)}\right\| + \left\|\tilde{B}^{(k)}(\bar{A} - \tilde{A}^{(k+1)})\right\| \\
&\leq \frac{1}{2}(2\epsilon)^{2^k} + \frac{2}{d}\frac{d}{2}\epsilon^{2^k} \\
&\leq \left(2^{2^k-1} + 1\right)\epsilon^{2^k} \leq (2\epsilon)^{2^k}.
\end{aligned}$$

□

This lemma shows that if $\bar{A}$ satisfies Equation 3.2, a sequence $\{\tilde{A}^{(k)}\}$ of approximations to $\bar{A}$ satisfying Equations 3.3–3.4 is given instead of $\bar{A}$, and if $\epsilon < \frac{1}{2}$, then it is possible to compute a sequence $\{\tilde{B}^{(k)}\}$ of approximations to $\bar{A}^{-1}$ such that the sequence $\{\tilde{B}^{(k)}\}$ converges quadratically.

3.2. Schur Complement and Norm Bounds

Let $\bar{A}$ be partitioned into the following form:

$$\bar{A} = \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \\ \bar{A}_{21} & \bar{A}_{22} \end{bmatrix},$$

and suppose $\bar{A}_{11}$ is nonsingular. Define the *Schur complement* of $\bar{A}$, denoted by $\Delta(\bar{A})$, to be $\Delta(\bar{A}) = \bar{A}_{22} - \bar{A}_{21}\bar{A}_{11}^{-1}\bar{A}_{12}$.

According to the definitions of matrix norms, the following lemma is easy to verify (see [GVL90], p. 60, Exercise P2.3.2).

LEMMA 3.6. *Let B be any submatrix of A; then $\|B\| \leq \|A\|$.*

Suppose $\left\|\bar{A} - D\right\| \leq \delta$. Then, by Lemma 3.6,

- $\left\|\bar{A}_{11} - D_1\right\| \leq \left\|\bar{A} - D\right\| \leq \delta$, and $\left\|\bar{A}_{22} - D_2\right\| \leq \left\|\bar{A} - D\right\| \leq \delta$, where $D_1 = dI_1$, $D_2 = dI_2$, and I_1, I_2 are the identity matrices with the same shapes as $\bar{A}_{11}$ and $\bar{A}_{22}$ respectively;
- $\left\|\bar{A}_{12}\right\| \leq \left\|\bar{A} - D\right\| \leq \delta$ and $\left\|\bar{A}_{21}\right\| \leq \left\|\bar{A} - D\right\| \leq \delta$.

LEMMA 3.7. *If $\left\|\bar{A} - D\right\| \leq d\epsilon/2$, where $0 \leq \epsilon \leq 1$, then $\left\|\Delta(\bar{A}) - D_2\right\| \leq d\epsilon$.*

PROOF. The condition $\left\|\bar{A} - D\right\| \leq d\epsilon/2$ implies that $\left\|\bar{A}_{11} - D_1\right\| \leq d\epsilon/2$. By Part (3) of Lemma 3.2 with $\epsilon \leq \frac{1}{2}$, $\left\|\bar{A}_{11}^{-1}\right\| \leq 2/d$.

Recall that the Schur complement of $\bar{A}$ is $\Delta(\bar{A}) = \bar{A}_{22} - \bar{A}_{21}\bar{A}_{11}^{-1}\bar{A}_{12}$. Notice that $\left\|\bar{A}_{21}\right\| \leq \left\|\bar{A} - D\right\| \leq d\epsilon/2$, $\left\|\bar{A}_{12}\right\| \leq \left\|\bar{A} - D\right\| \leq d\epsilon/2$, and $\left\|\bar{A}_{22} - D_2\right\| \leq \left\|\bar{A} - D\right\| \leq d\epsilon/2$. It follows that

$$\begin{aligned} \left\|\Delta(\bar{A}) - D_2\right\| &\leq \left\|\bar{A}_{22} - D_2\right\| + \left\|\bar{A}_{21}\right\| \left\|\bar{A}_{11}^{-1}\right\| \left\|\bar{A}_{12}\right\| \\ &\leq \frac{d\epsilon}{2} + \frac{2}{d}\left(\frac{d\epsilon}{2}\right)^2 \\ &= \frac{d\epsilon}{2} + \frac{d\epsilon^2}{2} \leq \frac{d\epsilon}{2}(1+\epsilon) \leq d\epsilon. \end{aligned}$$

□

This lemma shows that if the original matrix $\bar{A}$ is relatively close to a diagonal matrix D with $\left\|\bar{A} - D\right\| \leq d\epsilon/2$, then the Schur complement of $\bar{A}$ is also relatively close to a diagonal matrix with the same diagonal entries d. In particular, $\left\|\Delta(\bar{A}) - D_2\right\| \leq (1+\epsilon)(d\epsilon)/2 \leq d\epsilon$, so the "distance" only doubles.

LEMMA 3.8. *Suppose $\bar{A}$ satisfies Equation 3.2, the sequence $\tilde{A}^{(0)}, \tilde{A}^{(1)}, \ldots, \tilde{A}^{(k)}$ satisfies Equation 3.3 and Equation 3.4. Suppose also that $\tilde{A}^{(k)} = \begin{bmatrix} \tilde{A}_{11}^{(k)} & \tilde{A}_{12}^{(k)} \\ \tilde{A}_{21}^{(k)} & \tilde{A}_{22}^{(k)} \end{bmatrix}$, for $k \geq 0$, and that $\tilde{B}^{(0)} = D_1^{-1}$, and $\tilde{B}^{(k)} = \tilde{B}^{(k-1)}(2I - \tilde{A}_{11}^{(k)}\tilde{B}^{(k-1)})$, for $k \geq 1$. Let $0 \leq \epsilon \leq \frac{1}{2}$. Then*

$$\left\|\hat{\Delta}^{(k)} - \Delta\right\| \leq \frac{d}{2}(2\epsilon)^{2^k},$$

where $\hat{\Delta}_0 = D_2$, $\Delta = \bar{A}_{22} - \bar{A}_{21}\bar{A}_{11}^{-1}\bar{A}_{12}$, and $\hat{\Delta}^{(k)} = \tilde{A}_{22}^{(k)} - \tilde{A}_{21}^{(k)}\hat{B}^{(k)}\tilde{A}_{12}^{(k)}$.

PROOF. Notice that $\left\|\bar{A}_{21}\right\| \leq \left\|\bar{A} - D\right\| \leq d\epsilon/2$, $\left\|\bar{A}_{12}\right\| \leq \left\|\bar{A} - D\right\| \leq d\epsilon/2$, and $\left\|\bar{A}_{22} - D_2\right\| \leq \left\|\bar{A} - D\right\| \leq d\epsilon/2$. Since $\tilde{\Delta}^{(0)} = D_2$, it is easy to verify that

$$\begin{aligned}
\left\|\tilde{\Delta}^{(0)} - \Delta\right\| &\leq \left\|\bar{A}_{22} - D_2\right\| + \left\|\bar{A}_{21}\right\|\left\|\bar{A}_{11}^{-1}\right\|\left\|\bar{A}_{12}\right\| \\
&\leq \frac{d\epsilon}{2} + \frac{2}{d}\left(\frac{d\epsilon}{2}\right)^2 \\
&\leq \frac{d\epsilon}{2} + \frac{d\epsilon^2}{2} \leq d\epsilon.
\end{aligned}$$

Also,

$$\begin{aligned}
\left\|\tilde{\Delta}^{(k)} - \Delta\right\| &= \left\|\tilde{A}_{22}^{(k)} - \tilde{A}_{21}^{(k)}\tilde{B}^{(k)}\tilde{A}_{21}^{(k)} - (\bar{A}_{22} - \bar{A}_{21}\bar{A}_{11}^{-1}\bar{A}_{12})\right\| \\
&= \| \left(\tilde{A}_{22}^{(k)} - \bar{A}_{22}\right) + \left(\tilde{A}_{21}^{(k)}(\tilde{B}^{(k)} - \bar{A}_{11}^{-1})\tilde{A}_{12}^{(k)}\right) \\
&\quad + \left(\tilde{A}_{21}^{(k)}\bar{A}_{11}^{-1}(\tilde{A}_{12}^{(k)} - \bar{A}_{12})\right) + \left((\tilde{A}_{21}^{(k)} - \bar{A}_{21})\bar{A}_{11}^{-1}\bar{A}_{12}\right) \| \\
&\leq \left\|\tilde{A}_{22}^{(k)} - \bar{A}_{22}\right\| + \left\|\tilde{A}_{21}^{(k)}(\tilde{B}^{(k)} - \bar{A}_{11}^{-1})\tilde{A}_{12}^{(k)}\right\| \\
&\quad + \left\|\tilde{A}_{21}^{(k)}\bar{A}_{11}^{-1}(\tilde{A}_{12}^{(k)} - \bar{A}_{12})\right\| + \left\|(\tilde{A}_{21}^{(k)} - \bar{A}_{21})\bar{A}_{11}^{-1}\bar{A}_{12}\right\|.
\end{aligned}$$

By Lemma 3.6,

$$\left\|\tilde{A}_{21}^{(k)} - \bar{A}_{21}\right\| \leq \left\|\tilde{A}^{(k)} - \bar{A}\right\| \leq \frac{d}{2}\epsilon^{2^k},$$

$$\left\|\tilde{A}_{12}^{(k)} - \bar{A}_{12}\right\| \leq \left\|\tilde{A}^{(k)} - \bar{A}\right\| \leq \frac{d}{2}\epsilon^{2^k},$$

and

$$\left\|\tilde{A}_{22}^{(k)} - \bar{A}_{22}\right\| \leq \left\|\tilde{A}^{(k)} - \bar{A}\right\| \leq \frac{d}{2}\epsilon^{2^k}.$$

It is given that $\left\|\bar{A} - D\right\| \leq \frac{d}{2}\epsilon$, so $\left\|\bar{A}_{21}\right\| \leq \frac{d\epsilon}{2}$ and $\left\|\bar{A}_{12}\right\| \leq \frac{d\epsilon}{2}$. Also $0 \leq \epsilon \leq \frac{1}{2}$. Thus,

$$\left\|\tilde{A}_{21}^{(k)}\right\| \leq \frac{d}{2}(\epsilon + \epsilon^{2^k}) \leq \frac{3d}{8},$$

and

$$\left\|\tilde{A}_{12}^{(k)}\right\| \leq \frac{d}{2}(\epsilon + \epsilon^{2^k}) \leq \frac{3d}{8},$$

when $k \geq 1$.

Lemma 3.6 also implies that $\left\|\bar{A}_{11} - D_1\right\| \leq \left\|\bar{A} - D\right\| \leq d\epsilon/2$. It follows that $\left\|\bar{A}_{11}^{-1}\right\| \leq 4/3d$ by Part 2 of Lemma 3.2 using the inequality $\epsilon \leq \frac{1}{2}$.

Since $\bar{A}$ and $\{\tilde{A}^{(k)}\}$ satisfy Equations 3.2–3.4, $\bar{A}_{11}$ and $\{\tilde{A}_{11}^{(k)}\}$ satisfy the corresponding conditions. By Lemma 3.5,

$$\left\|I - \tilde{B}^{(k)}\bar{A}_{11}\right\| \leq (2\epsilon)^{2^k}.$$

Thus,

$$\begin{aligned}
\left\|\hat{\Delta}^{(k)} - \Delta\right\| &\leq \left\|\hat{A}_{22}^{(k)} - \bar{A}_{22}\right\| + \left\|\hat{A}_{21}^{(k)}\right\| \left\|I - \tilde{B}^{(k)}\bar{A}_{11}\right\| \left\|\bar{A}_{11}^{-1}\right\| \left\|\hat{A}_{21}^{(k)}\right\| \\
&\quad + \left\|\tilde{A}_{21}^{(k)}\right\| \left\|\bar{A}_{11}^{-1}\right\| \left\|\hat{A}_{12}^{(k)} - \bar{A}_{12}\right\| + \left\|\hat{A}_{21}^{(k)} - \bar{A}_{21}\right\| \left\|\bar{A}_{11}^{-1}\right\| \left\|\bar{A}_{12}\right\| \\
&\leq \frac{d}{2}\epsilon^{2^k} + \frac{3d}{8}(2\epsilon)^{2^k}\frac{4}{3d}\frac{3d}{8} + \frac{3d}{8}\frac{4}{3d}\frac{d}{2}\epsilon^{2^k} + \frac{d}{2}\epsilon^{2^k}\frac{4}{3d}\frac{3d}{8} \\
&\leq \left(\frac{d}{2} + \frac{d}{4} + \frac{d}{4}\right)\epsilon^{2^k} + \frac{3d}{16}(2\epsilon)^{2^k} \\
&\leq \frac{d}{4}(2\epsilon)^{2^k} + \frac{3d}{16}(2\epsilon)^{2^k} \\
&\leq \frac{d}{2}(2\epsilon)^{2^k},
\end{aligned}$$

where $k \geq 1$, so that $d\epsilon^{2^k} \leq \frac{d}{4}(2\epsilon)^{2^k}$, and since $\frac{d}{4} + \frac{3d}{16} < \frac{d}{2}$. □

LEMMA 3.9. *Suppose $\bar{A}$ satisfies Equation* 3.2, *the sequence $\tilde{A}^{(0)}, \tilde{A}^{(1)}, \ldots, \hat{A}^{(k)}$ satisfies Equations* 3.3 *and Equation* 3.4, *$\tilde{B}^{(0)} = D_1^{-1}$, and $\tilde{B}^{(k)} = \hat{B}^{(k-1)}(2I - \tilde{A}_{11}^{(k)}\tilde{B}^{(k-1)})$ for $k \geq 1$. Let $0 \leq \epsilon \leq \frac{1}{2}$. Then*

$$\left\|\tilde{\Delta}^{(k)} - \tilde{\Delta}^{(k+1)}\right\| \leq \frac{d}{2}(2\epsilon)^{2^k},$$

where $\hat{\Delta}^{(0)} = D_2$, and $\tilde{\Delta}^{(k)} = \tilde{A}_{22}^{(k)} - \hat{A}_{21}^{(k)}\tilde{B}^{(k)}\tilde{A}_{12}^{(k)}$.

PROOF. The equality $\tilde{A}^{(0)} = D$ and Equation 3.4 imply that $\left\|\tilde{A}_{21}^{(1)}\right\| \leq \left\|\tilde{A}^{(1)} - D\right\| \leq d\epsilon^2/2$, $\left\|\tilde{A}_{12}^{(1)}\right\| \leq \left\|\tilde{A}^{(1)} - D\right\| \leq d\epsilon^2/2$, and $\left\|\tilde{A}_{22}^{(1)} - D_2\right\| \leq \left\|\tilde{A}^{(1)} - D\right\| \leq d\epsilon^2/2$. Also $\bar{A}_{11}$ and $\{\tilde{A}_{11}^{(k)}\}$ satisfy Equations 3.2–3.4. Thus Lemma 3.4 shows that $\left\|\tilde{B}^{(k)}\right\| \leq 2/d$ and $\left\|I - \tilde{B}^{(k)}\tilde{A}_{11}^{(k+1)}\right\| \leq 2^{2^k-1}\epsilon^{2^k}$. Since $\hat{\Delta}^{(0)} = D_2$, it is easy to verify that

$$\begin{aligned}
\left\|\hat{\Delta}^{(0)} - \tilde{\Delta}^{(1)}\right\| &\leq \left\|\tilde{A}_{22}^{(1)} - D_2\right\| + \left\|\tilde{A}_{21}^{(1)}\tilde{B}^{(1)}\tilde{A}_{12}^{(1)}\right\| \\
&\leq \frac{d\epsilon}{2} + \frac{d\epsilon^2}{2}\frac{2}{d}\frac{d\epsilon^2}{2} \\
&\leq \frac{d\epsilon}{2} + \frac{d\epsilon^4}{2} \leq d\epsilon.
\end{aligned}$$

When $k \geq 1$,

$$\begin{aligned}
\left\|\hat{\Delta}^{(k)} - \hat{\Delta}^{(k+1)}\right\| &= \left\|\hat{A}_{22}^{(k)} - \hat{A}_{21}^{(k)}\hat{B}^{(k)}\hat{A}_{21}^{(k)} - (\hat{A}_{22}^{(k+1)} - \hat{A}_{21}^{(k+1)}\hat{B}^{(k+1)}\hat{A}_{21}^{(k+1)})\right\| \\
&= \| \left(\hat{A}_{22}^{(k)} - \hat{A}_{22}^{(k+1)}\right) + \left(\hat{A}_{21}^{(k)}(\hat{B}^{(k)} - \hat{B}^{(k+1)})\hat{A}_{12}^{(k)}\right) \\
&\quad + \left(\hat{A}_{21}^{(k)}\hat{B}^{(k+1)}(\hat{A}_{12}^{(k)} - \hat{A}_{12}^{(k+1)})\right) + \left((\hat{A}_{21}^{(k)} - \hat{A}_{21}^{(k+1)})\hat{B}^{(k+1)}\hat{A}_{12}^{(k+1)}\right) \| \\
&\leq \left\|\hat{A}_{22}^{(k)} - \hat{A}_{22}^{(k+1)}\right\| + \left\|\hat{A}_{21}^{(k)}(\hat{B}^{(k)} - \hat{B}^{(k+1)})\hat{A}_{12}^{(k)}\right\| \\
&\quad + \left\|\hat{A}_{21}^{(k)}\hat{B}^{(k+1)}(\hat{A}_{12}^{(k)} - \hat{A}_{12}^{(k+1)})\right\| + \left\|(\hat{A}_{21}^{(k)} - \hat{A}_{21}^{(k+1)})\hat{B}^{(k+1)}\hat{A}_{12}^{(k+1)}\right\|.
\end{aligned}$$

By Lemma 3.6 and Equation 3.4,

$$\left\|\tilde{A}_{22}^{(k)} - \tilde{A}_{22}^{(k+1)}\right\| \leq \left\|\tilde{A}^{(k)} - \tilde{A}^{(k+1)}\right\| \leq \frac{d}{2}\epsilon^{2^k},$$

$$\left\|\tilde{A}_{12}^{(k)} - \tilde{A}_{12}^{(k+1)}\right\| \leq \left\|\tilde{A}^{(k)} - \tilde{A}^{(k+1)}\right\| \leq \frac{d}{2}\epsilon^{2^k},$$

and

$$\left\|\tilde{A}_{21}^{(k)} - \tilde{A}_{21}^{(k+1)}\right\| \leq \left\|\tilde{A}^{(k)} - \tilde{A}^{(k+1)}\right\| \leq \frac{d}{2}\epsilon^{2^k}.$$

It is given that $\left\|\bar{A} - D\right\| \leq \frac{d}{2}\epsilon$, and $0 \leq \epsilon \leq \frac{1}{2}$, so

$$\left\|\tilde{A}_{21}^{(k)}\right\| \leq \frac{d}{2}(\epsilon + \epsilon^{2^k}) \leq \frac{3d}{8},$$

and

$$\left\|\tilde{A}_{12}^{(k)}\right\| \leq \frac{d}{2}(\epsilon + \epsilon^{2^k}) \leq \frac{3d}{8},$$

when $k \geq 1$.

Since $\tilde{B}^{(k+1)} = \tilde{B}^{(k)}(2I - \tilde{A}_{11}^{(k+1)}\tilde{B}^{(k)})$,

$$\begin{aligned}
\left\|\tilde{B}^{(k+1)} - \tilde{B}^{(k)}\right\| &= \left\|\tilde{B}^{(k)}\left(I - \tilde{A}_{11}^{(k+1)}\tilde{B}^{(k)}\right)\right\| \\
&\leq \left\|\tilde{B}^{(k)}\right\|\left\|I - \tilde{A}_{11}^{(k+1)}\tilde{B}^{(k)}\right\|.
\end{aligned}$$

Lemma 3.4 shows that

$$\left\| I - \hat{A}_{11}^{(k+1)} \hat{B}^{(k)} \right\| \leq 2^{2^k - 1} \epsilon^{2^k},$$

and also that

$$\left\| \hat{B}^{(k)} \right\| \leq \frac{2}{d}.$$

All these facts and the inequality $k \geq 1$ imply that

$$\begin{aligned}
\left\| \hat{\Delta}^{(k)} - \hat{\Delta}^{(k+1)} \right\| &\leq \frac{d}{2}\epsilon^{2^k} + \frac{3d}{8}\frac{2}{d}2^{2^k-1}\epsilon^{2^k}\frac{3d}{8} + \frac{3d}{8}\frac{2}{d}\frac{d}{2}\epsilon^{2^k} + \frac{d}{2}\epsilon^{2^k}\frac{2}{d}\frac{3d}{8} \\
&\leq \left(\frac{d}{2} + \frac{3d}{8} + \frac{3d}{8}\right)\epsilon^{2^k} + \frac{9d}{32}2^{2^k-1}\epsilon^{2^k} \\
&\leq \frac{5d}{16}(2\epsilon)^{2^k} + \frac{9d}{64}(2\epsilon)^{2^k} \\
&\leq \frac{d}{2}(2\epsilon)^{2^k}.
\end{aligned}$$

□

CHAPTER 4

Efficient Parallel Factorization

Reif gives processor efficient parallel algorithms for general dense matrix factorizations. These algorithms reduce the parallel time of $O(\log^3 n)$ required by the previous known processor efficient algorithms to $O(\log^2 n)$. A complete analysis of Reif's parallel algorithms is given in this chapter. The analysis shows that the bit precision required for these algorithms can be significantly reduced. The analysis also shows that certain steps of Reif's algorithms can be simplified. Recursive factorization trees (RF trees) of matrices are introduced. Some important properties of factorization trees are shown.

4.1. Recursive Factorization Trees of Matrices

Assume n is a power of 2. If all principal submatrices of $A \in \mathbb{R}^{n\times n}$ are nonsingular, a full binary tree of depth $\log n$ can be associated with A as follows.

Every node in the binary tree is indexed with a binary string, and contains a matrix. The root is denoted as <>, the empty string, and contains the matrix A. A node at depth t, $0 \leq t \leq \log n$, contains an $n/2^t \times n/2^t$ matrix. The matrix in the node α at depth t is denoted as A_α, where α is a binary string of length t. Each node α at depth t, $0 \leq t < \log n$, has exactly two children, $\alpha 0$ and $\alpha 1$, which contain $A_{\alpha 0}$ and $A_{\alpha 1}$ respectively. The matrices $A_{\alpha 0}$ and $A_{\alpha 1}$ are constructed from A_α as follows. Let the matrix A_α at depth t, $0 \leq t < \log n$, be partitioned as

$$A_\alpha = \begin{bmatrix} A_{\alpha 0} & X_\alpha \\ Y_\alpha & Z_\alpha \end{bmatrix},$$

where $A_{\alpha 0}$, X_α, Y_α, and Z_α are $n/2^{t+1} \times n/2^{t+1}$ matrices. As stated above, the left child of the node α contains the matrix $A_{\alpha 0}$. The right child of α contains $\Delta(A_\alpha) = Z_\alpha - Y_\alpha A_{\alpha 0}^{-1} X_\alpha$, which is the Schur complement of A_α.

The above binary tree is called the *Recursive Factorization tree* (RF tree) of the matrix A. For $0 \leq t^* \leq t$, an RF tree of depth t^* is the RF tree defined only to the depth t^* [Rei95]. In this thesis, extended versions of the above RF tree are used. An "augmented" RF tree has the same structure except that each node α also includes the inverse A_α^{-1} of the matrix A_α found in that node of the regular "RF tree". An "extended" RF tree includes this extra matrix at each node α, as well as an integer m_α which will be defined later.

Not every $n \times n$ matrix has an RF tree. The following lemma is given by Reif [Rei94].

LEMMA 4.1. *Any symmetric positive definite matrix A has an RF tree and it is unique.*

PROOF. Let A be partitioned as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \tag{4.1}$$

where A_{11}, A_{12}, A_{21}, and A_{22} are $n/2 \times n/2$ matrices. Since A is symmetric and positive definite, A_{11} is invertible. If A_{11} is nonsingular then

$$A = \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & \Delta \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix},$$

where Δ is the Schur complement of the matrix A, $\Delta = A_{22} - A_{21}A_{11}^{-1}A_{12}$.

Since A is also symmetric, $A_{12} = A_{21}^T$ and $A_{11} = A_{11}^T$, so that

$$\begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} = \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix}^T,$$

and

$$\begin{bmatrix} I & -A_{11}^{-1}A_{21} \\ 0 & I \end{bmatrix} = \begin{bmatrix} I & 0 \\ -A_{12}A_{11}^{-1} & I \end{bmatrix}^T.$$

By Lemma 1.1, the matrix $\begin{bmatrix} A_{11} & 0 \\ 0 & \Delta \end{bmatrix}$ is also symmetric and positive definite, because

$$\begin{bmatrix} A_{11} & 0 \\ 0 & \Delta \end{bmatrix} = \begin{bmatrix} I & -A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix}^T \cdot A \cdot \begin{bmatrix} I & -A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix}.$$

Thus A_{11} and Δ are both symmetric and positive definite, and this factorization can continue. Thus existence of the RF tree follows by induction on n. The uniqueness follows from the deterministic construction of the RF Tree. □

LEMMA 4.2. *The inverse of A can be recursively computed from the RF tree of A,*

as

$$A^{-1} = \begin{bmatrix} I & -A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & \Delta^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix}.$$

LEMMA 4.3. *If A has an RF tree then the LU factorization of A can be recursively computed from the RF tree of A.*

PROOF.

$$A = \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & \Delta \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix},$$

and the RF tree of A includes RF trees for A_{11} and Δ. Suppose $A_{11} = L_1U_1$ and $\Delta = L_2U_2$; then $A = LU$, for

$$L = \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} L_1 & 0 \\ 0 & L_2 \end{bmatrix} \text{ and } U = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix}.$$

□

Since A_α is defined recursively, it is natural to check the relation between A_α and the original matrix A. This observation, (Lemma 4.4, which follows) leads to the important properties of the RF tree and clarifies one ambiguity in Reif's algorithms (see [Rei94, Rei95]).

Define $val(\alpha)$ to be the value obtained by treating α as a base-2 representation of an integer for any $\alpha \neq <>$, and define $val(<>) = 0$. For example, $val(0001) = 1$ and $val(1101) = 13$. It is clear that $val(\alpha) \leq 2^t - 1$ for any node α of depth t. Define $s(\alpha)$ to be a function of n and α, for some α at depth t, which shows the corresponding position of A_α in A: $s(\alpha) = \frac{n}{2^t} val(\alpha)$. For example, $s(01) = \frac{n}{4}$; $s(101) = \frac{5n}{8}$. It is easy to verify that $s(\alpha) \leq n - n/2^t$ because $val(\alpha) \leq 2^t - 1$

The following lemma shows that A_α is more directly related to A: in particular, A_α is a principal submatrix of A or a Schur complement for some principal submatrix

of A.

LEMMA 4.4. *Let α be a node of depth t in the RF tree of A. If $s(\alpha) = 0$, then A_α is the $n/2^t \times n/2^t$ principal submatrix of A. Otherwise, let $\hat{A}$ be the $(s(\alpha) + n/2^t) \times (s(\alpha) + n/2^t)$ principal submatrix of A, and let*

$$\hat{A} = \begin{bmatrix} A_{11} & X \\ Y & Z \end{bmatrix}, \tag{4.2}$$

where A_{11} is of size $s(\alpha) \times s(\alpha)$, X is of size $s(\alpha) \times n/2^t$, Y is of size $n/2^t \times s(\alpha)$, and Z is of size $n/2^t \times n/2^t$. Then $A_\alpha = Z - YA_{11}^{-1}X$.

PROOF. When $t = 1$, the claim obviously holds because A_0 is the $n/2 \times n/2$ principal submatrix of A and $s(0) = 0$, and A_1 is the Schur complement of A according to the definition of RF tree of A, and $s(1) = n/2$.

Suppose the claim holds for depth t and let α be a node of depth t. If $s(\alpha) = 0$, consider the nodes $\alpha 0$ and $\alpha 1$ at depth $t + 1$. Since $A_{\alpha 0}$ is the $n/2^{t+1} \times n/2^{t+1}$ principal submatrix of A_α, the claim obviously holds for $A_{\alpha 0}$; meanwhile, $A_{\alpha 1}$ is the Schur complement of A_α, so the claim holds for $\alpha 1$ as well.

If $s(\alpha) \neq 0$, then by the inductive hypothesis, $A_\alpha = \Delta = Z - YA_{11}^{-1}X$ for A_{11}, Y, X, and Z as given above.

Let $X = \begin{bmatrix} X_1 & X_2 \end{bmatrix}$ for $s(\alpha) \times 2^{t+1}$ matrices X_1 and X_2, let $Y = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}$ for $2^{t+1} \times s(\alpha)$ matrices Y_1 and Y_2, and let $Z = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix}$ for $2^{t+1} \times 2^{t+1}$ matrices Z_{11}, Z_{12}, Z_{21}, and Z_{22}. Then the Schur complement $A_\alpha = Z - YA_{11}^{-1}X$ has the form

$$A_\alpha = \begin{bmatrix} Z_{11} - Y_1A_{11}^{-1}X_1 & Z_{12} - Y_1A_{11}^{-1}X_2 \\ Z_{21} - Y_2A_{11}^{-1}X_1 & Z_{22} - Y_2A_{11}^{-1}X_2 \end{bmatrix}.$$

The matrix $A_{\alpha 0}$ is, by the definition of RF tree, the leading principal submatrix of A_α, so $A_{\alpha 0} = Z_{11} - Y_1A_{11}^{-1}X_1$, and since $s(\alpha) = s(\alpha 0)$, this is the Schur comple-

ment obtained using the $(s(\alpha)+n/2^{t+1}) \times (s(\alpha)+n/2^{t+1})$ principal submatrix of A, $\begin{bmatrix} A_{11} & X_1 \\ Y_1 & Z_{11} \end{bmatrix}$, as desired.

By definition of RF tree, and since

$$A_\alpha = \begin{bmatrix} Z_{11} - Y_1 A_{11}^{-1} X_1 & Z_{12} - Y_1 A_{11}^{-1} X_2 \\ Z_{21} - Y_2 A_{11}^{-1} X_1 & Z_{22} - Y_2 A_{11}^{-1} X_2 \end{bmatrix},$$

$$\begin{aligned} A_{\alpha 1} &= (Z_{22} - Y_2 A_{11}^{-1} X_2) - (Z_{21} - Y_2 A_{11}^{-1} X_1)(Z_{11} - Y_1 A_{11}^{-1} X_1)^{-1}(Z_{12} - Y_1 A_{11}^{-1} X_2) \\ &= (Z_{22} - Y_2 A_{11}^{-1} X_2) - (Z_{21} - Y_2 A_{11}^{-1} X_1) A_{\alpha 0}^{-1}(Z_{12} - Y_1 A_{11}^{-1} X_2). \end{aligned}$$

Now it is necessary and sufficient to show that this is the Schur complement,

$$\hat{Z} - \hat{Y} \tilde{A}_{11}^{-1} \hat{X},$$

for $\tilde{A}_{11} = \begin{bmatrix} A_{11} & X_1 \\ Y_1 & Z_{11} \end{bmatrix}$, $\tilde{X} = \begin{bmatrix} X_2 \\ Z_{12} \end{bmatrix}$, $\tilde{Y} = \begin{bmatrix} Y_2 & Z_{21} \end{bmatrix}$, and $\tilde{Z} = Z_{22}$.

Since A_{11} is nonsingular, and $A_{\alpha 0} = Z_{11} - Y_1 A_{11}^{-1} X_1$,

$$\tilde{A}_{11} = \begin{bmatrix} I & 0 \\ Y_1 A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & A_{\alpha 0} \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1} X_1 \\ 0 & I \end{bmatrix}$$

so

$$\tilde{A}_{11}^{-1} = \begin{bmatrix} I & -A_{11}^{-1} X_1 \\ 0 & I \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & A_{\alpha 0}^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -Y_1 A_{11}^{-1} & I \end{bmatrix}.$$

Thus

$$\begin{aligned} \tilde{Y} \tilde{A}_{11}^{-1} \tilde{X} &= \begin{bmatrix} Y_2 & Z_{21} \end{bmatrix} \begin{bmatrix} I & -A_{11}^{-1} X_1 \\ 0 & I \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & A_{\alpha 0}^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -Y_1 A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} X_2 \\ Z_{12} \end{bmatrix} \\ &= \begin{bmatrix} Y_2 & Z_{21} - Y_2 A_{11}^{-1} X_1 \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & A_{\alpha 0}^{-1} \end{bmatrix} \begin{bmatrix} X_2 \\ Z_{12} - Y_1 A_{11}^{-1} X_2 \end{bmatrix} \\ &= Y_2 A_{11}^{-1} X_2 + (Z_{21} - Y_2 A_{11}^{-1} X_1) A_{\alpha 0}^{-1}(Z_{12} - Y_1 A_{11}^{-1} X_2), \end{aligned}$$

and

$$\hat{Z} - \hat{Y}\tilde{A}_{11}^{-1}\hat{X} = (Z_{22} - Y_2 A_{11}^{-1} X_2) - (Z_{21} - Y_2 A_{11}^{-1} X_1) A_{\alpha 0}^{-1} (Z_{12} - Y_1 A_{11}^{-1} X_2) = A_{\alpha 1},$$

as required. The result follows by induction on t. □

If the given matrix A has integer entries, the determinant of A is an integer; furthermore, the adjoint matrix $\mathrm{adj}(A)$ is also a matrix with integer entries. However, the matrices in all the nodes of the RF tree of an integer matrix are not guaranteed to be integer matrices.

Recall that each node α in the extended RF tree has an associated integer value m_α. The integer m_α is chosen in such a way that $m_\alpha A_\alpha$ is guaranteed to be an integer matrix. Recursively define m_α for all the nodes in the RF tree as follows. Define $m_{<>}$ to be 1. Let the left child and right child of the node α contain $m_{\alpha 0} = m_\alpha$ and $m_{\alpha 1} = m_\alpha \det(A_{\alpha 0})$ respectively. The following lemma shows that m_α is also closely related to the matrix A.

LEMMA 4.5. *For a node α in the RF tree of a matrix A, if $s(\alpha) = 0$, then $m_\alpha = 1$; otherwise, $m_\alpha = \det(\hat{A})$, where $\hat{A}$ is the $s(\alpha) \times s(\alpha)$ principal submatrix of A.*

PROOF. For a node at depth 1, the claim holds because $m_0 = 1$ and $m_1 = \det(A_0)$ according to the definition of m_α.

Suppose the claim holds for depth t, so that $m_\alpha = \det(\tilde{A})$, where α is a node at depth t and $\tilde{A}$ is the $s(\alpha) \times s(\alpha)$ principal submatrix of A. Suppose $s(\alpha) > 0$, since the claim obviously holds for $\alpha 0$ and $\alpha 1$ if $s(\alpha) = 0$. Since $m_{\alpha 0} = m_\alpha$, and $s(\alpha 0) = s(\alpha)$, the claim holds for $\alpha 0$. Meanwhile $s(\alpha 1) = s(\alpha) + \frac{n}{2^{t+1}}$, and $m_{\alpha 1} = m_\alpha \det(A_{\alpha 0})$ by definition. Lemma 4.4 shows that $A_{\alpha 0}$ is the Schur complement of $\hat{A}$, where $\hat{A}$ is the $(s(\alpha 0)+n/2^{t+1}) \times (s(\alpha 0)+n/2^{t+1})$ principal submatrix of A and where $\hat{A}$ is partitioned as in Equation 4.2. Notice that $\det(\hat{A}) = \det(A_{11}) \cdot \det(\Delta(\hat{A})) = \det(A_{11}) \cdot \det(A_{\alpha 0})$. Clearly $A_{11} = \tilde{A}$, and by hypothesis $m_\alpha = \det(\tilde{A})$, so the claim holds for $A_{\alpha 1}$ as well. The lemma follows by induction on t. □

LEMMA 4.6. *In the RF tree of an integer matrix A, $m_\alpha A_\alpha$ and $m_\alpha \mathrm{adj}(A_\alpha)$ are integer matrices for every node α.*

PROOF. Lemma 4.4 shows that A_α is either a principal submatrix of A or a Schur complement of $\hat{A}$, where $\hat{A}$ is a principal submatrix of A with a certain size. If A_α is a principal submatrix of A, then $m_\alpha = 1$, and $m_\alpha A_\alpha$ is clearly an integer matrix. Suppose A_α is a Schur complement of a principal submatrix $\hat{A}$ of A. Clearly $\hat{A}$ is an integer matrix. Let $\hat{A}$ be partitioned as in Equation 4.2, so that $A_\alpha = Z - YA_{11}^{-1}X$. Lemma 4.5 shows that $m_\alpha = \det(A_{11})$. Thus $m_\alpha A_\alpha = \det(A_{11})(Z - YA_{11}^{-1}X) = \det(A_{11})Z - Y\mathrm{adj}(A_{11})X$ is also an integer matrix.

The same argument shows that $m_\alpha \mathrm{adj}(A_\alpha)$ is an integer matrix for any node α. Suppose α is a principal submatrix of A, then $m_\alpha = 1$ and $m_\alpha \mathrm{adj}(A_\alpha)$ is an integer matrix, since A_α is an integer matrix. Suppose A_α is a Schur compliment of a principal submatrix $\hat{A}$ of A. Let $\hat{A}^{-1}$ be partitioned according to the partitioning of $\hat{A}$. Then the bottom right corner of $\hat{A}^{-1}$ is A_α^{-1}. Since $\hat{A}$ is an integer matrix, $\mathrm{adj}(\hat{A}) = \det(\hat{A})\hat{A}^{-1}$ is also an integer matrix. Thus $\det(\hat{A})A_\alpha^{-1}$ is also an integer matrix, because A_α^{-1} is a submatrix of $\hat{A}^{-1}$. Now $m_\alpha \mathrm{adj}(A_\alpha) = \det(A_{11})\det(A_\alpha)A_\alpha^{-1}$, but $\det(A_{11})\det(A_\alpha) = \det(\hat{A})$, and hence $m_\alpha \mathrm{adj}(A_\alpha) = \det(\hat{A})A_\alpha^{-1}$ is an integer matrix as well. □

The following lemma shows a bound for the norm of $m_\alpha A_\alpha$.

LEMMA 4.7. *For any node α in the RF tree of A, $\|m_\alpha A_\alpha\| \leq n^2 \|A\|^n$.*

PROOF. Notice that A_α is either a principal submatrix of A or a Schur complement of a principal submatrix of A. If A_α is a principal submatrix of A, then $m_\alpha = 1$ and the claim is obvious. Suppose A_α is a Schur complement of a principal submatrix $\hat{A}$ of A, which is partitioned as in Equation 4.2. Then $A_\alpha = Z - YA_{11}^{-1}X$, where A_{11}, X, Y, and Z are submatrices of $\hat{A}$. It follows that $m_\alpha A_\alpha = \det(A_{11})Z - Y\mathrm{adj}(A_{11})X$ by Lemma 4.5. It is now easy to verify that $\|\mathrm{adj}(A_{11})\| \leq (n-1)^2 \|A\|^{n-2}$. Thus $\|m_\alpha A_\alpha\| \leq n^2 \|A\|^n$. □

4.2. RF Trees and Matrix Computations

In this section, it is shown that some matrix problems can be reduced to computing the RF tree. Then Reif's processor efficient algorithm for computing the RF tree of a symmetric positive definite matrix is outlined.

4.2.1. Reduction of Matrix Computations to the RF Tree. The following problems can be efficiently reduced to computation of the RF tree [Rei94]. The concept of efficient NC reduction is defined in Definition 1.15.

Given a nonsingular matrix $A \in \mathbb{R}^{n\times n}$, if A has an RF tree, then A^{-1} can be computed by the RF sequence (see Lemma 4.2). Otherwise, A^TA is symmetric and positive definite, so that if A is nonsingular, then A^TA has an RF tree (see Lemma 4.1). Then $(A^TA)^{-1} = A^{-1}(A^T)^{-1}$ can be computed from the RF tree of A^TA. Thus $A^{-1} = (A^TA)^{-1}A^T$ can be computed by an additional matrix multiplication.

LEMMA 4.8. *There is an efficient NC reduction from matrix inversion to computing the RF tree.*

Given a matrix $A \in \mathbb{R}^{n\times n}$, recall the LU factorization for A (see Definition 1.6).

LEMMA 4.9. *There is an efficient NC reduction from LU factorization to computing the RF tree.*

LEMMA 4.10. *There is an efficient parallel reduction from solving a nonsingular system of linear equations to computing the RF tree.*

Given a matrix $A \in \mathbb{R}^{n\times n}$, the QR factorization of A can be computed from the LU factorization of A^TA, assuming that square roots of positive real numbers can be computed efficiently. Let $A^TA = LU$ be a Cholesky factorization, so that $U = L^T$. Then $R = U = L^T$ and $Q = AU^{-1}$ gives a QR factorization of A. Since U is upper triangular, $O(\log^2 n)$ time and $O(P(n))$ processors are sufficient to compute this factorization from L and U.

LEMMA 4.11. *There is an efficient NC reduction from QR factorization to computing the RF tree.*

If the Krylov matrix $K(A,v) = (v, Av, A^2v, \ldots, A^{n-1}v)$ is nonsingular, with QR factorization $K(A,v) = QR$, then $H = Q^TAQ$ is in upper Hessenberg form (see, for example, [BP94]).

LEMMA 4.12. *There is an efficient randomized NC reduction from Hessenberg reduction to computing the RF tree when the characteristic polynomial of the input matrix equals the minimum polynomial.*

4.2.2. Computing the Exact RF Tree. Reif's processor efficient algorithm for computing the RF tree of a given symmetric positive definite matrix is sketched. A modified (and somewhat simplified) version of the algorithm appears in the later sections of this chapter.

The following definition and algorithm are directly taken from Reif's paper.

Let G be the directed acyclic digraph derived from the RF tree by simply (1) including all tree edges directed from the children to parents, and (2) adding directed edges from each node of form $\alpha 1$ to its sibling of form $\alpha 0$. The ordering of directed edges of G allows for the recursive computation of $m_{\alpha 0} = m_\alpha$ and $m_{\alpha 1} = m_\alpha \det(A_{\alpha 0})$ (since it will be the case that computing $\det(A_{\alpha 0})$ requires the computation of $m_{\alpha 0}$). The longest path in G has $1 + 2\log n$ nodes. The nodes of G can be partitioned into $1 + 2\log n$ blocks $\Pi_0, \Pi_1, \ldots, \Pi_{2\log n}$ so that the evaluation of m_α is executed in parallel for $\alpha \in \Pi_j$ in $1 + 2\log n$ sequential stages for $j = 0, \ldots, 2\log n$ (see [Rei94], pp. 15–16).

ALGORITHM 4.1. *Computing the RF tree of a Symmetric Positive Definite Matrix.*

Input: An $n \times n$ integer symmetric positive definite matrix A, with integer entries of magnitude at most 2^β, where $\beta \leq n^{O(1)}$, and where n is assumed to be a power of 2.

Output: RF tree of A

begin Let $\pi = n(\beta + \log n)$.

step 1 Depending on whether a deterministic or randomized algorithm is desired, perform (1) or (2).

(1) Let p be any prime from the interval $[(n\|A\|_\infty)^{nc_0}, (n\|A\|_\infty)^{nc_0'}]$ for some constants $2 < c_0 < c_0'$.

(2) Let p be a prime from the interval $[2(n\|A\|_\infty)^{c_0}/n, 2(n\|A\|_\infty)^{c_0}]$ for some constant $c_0 > 2$.

step 2 Let $\bar{A} = A + Ip(n\|A\|_\infty)^{c_1 n}$, for a sufficiently large positive constant c_1.

step 3 Apply approximate Newton iteration to compute in $O(\log^2 n)$ time using $O(P(n))$ processors, an approximate RF tree of $\bar{A}$ within accuracy $2^{-c\pi}$, for a sufficiently large positive constant c.

step 4 Construct an ordered partition $\Pi_0, \Pi_1, \ldots, \Pi_{2\log n}$ of the strings $\{\alpha \in \{0,1\}^t | t \leq \log n\}$.

for $i := 0$ to $2\log n$ do (sequentially)

for all $\alpha \in \Pi_i$ do (in parallel)

(1) Define integer multipliers $\bar{m}_\alpha$ for the matrices $\bar{A}_\alpha$.

(2) Compute $\det(\bar{A}_\alpha)$ to accuracy within $2^{-c\pi}$.

(3) Compute $\bar{m}_\alpha \det(\bar{A}_\alpha)$ exactly by rounding to the nearest integer the product of $\bar{m}_\alpha$ times this approximation to $\det(\bar{A}_\alpha)$.

(4) Represent $\det(\bar{A}_\alpha)$ exactly as the rational fraction $\bar{m}_\alpha \det(\bar{A}_\alpha)$ divided by $\bar{m}_\alpha$.

(5) if $\det(\bar{A}_\alpha) = 0 \bmod p$ then goto Step 1.

(6) Multiply the exact rational value of $\det(\bar{A}_\alpha)$ by the approximation of $(\bar{A}_\alpha)^{-1}$ to get an approximation of the adjoint

matrix $\mathrm{adj}(\bar{A}_\alpha)$ to accuracy within $2^{-c\pi}$.

(7) To compute $\bar{m}_\alpha \mathrm{adj}(\bar{A}_\alpha)$ exactly, round to the nearest integer the product of $\bar{m}_\alpha$ times this approximation of the adjoint matrix $\mathrm{adj}(\bar{A}_\alpha)$.

(8) Represent $\bar{A}_\alpha^{-1} = \frac{\mathrm{adj}(\bar{A}_\alpha)}{\det(\bar{A}_\alpha)}$ exactly as the rational fraction $\bar{m}_\alpha \mathrm{adj}(\bar{A}_\alpha)$ divided by $\bar{m}_\alpha \det(\bar{A}_\alpha)$.

end for

end for

step 5 Reduce (mod p) the exact RF tree of $\bar{A}$, yielding RF tree (mod p) of A.

step 6 Apply Newton-Hensel Lifting to compute the RF tree (mod p^{2^i}) of A for $i = 0, \ldots, k^* = \lceil \log_p(c\pi) \rceil$. If $\det(\bar{A}_\alpha) = 0 \bmod p^{2^i}$ for any i and α, then exit and report A has no RF tree.

step 7 Using the exact RF tree of A, compute $\det(A_\alpha)$ for each α.

step 8 for $i := 0$ to $2 \log n$ do (sequentially)

for all $\alpha \in \Pi_i$ do (in parallel)

(1) Compute integer multipliers m_α and $\det(A_\alpha)$.

(2) Using the exact RF tree of A, represent A_α exactly as the rational fraction of integer matrices: $m_\alpha \det(A_\alpha)$ divided by m_α.

(3) Compute A_α^{-1}.

(4) Let $m_\alpha \mathrm{adj}(\bar{A}_\alpha)$ be the integer matrix $m_\alpha \det(A_\alpha)(A_\alpha)^{-1}$.

(5) Represent $\bar{A}_\alpha^{-1} =$ exactly as the rational fraction of integer matrices: $\bar{m}_\alpha \mathrm{adj}(\bar{A}_\alpha)$ divided by $\bar{m}_\alpha \det(\bar{A}_\alpha)$.

end for

end for

end

In Reif's algorithm, $\bar{A} = A + Ip(n\|A\|_\infty)^{c_1 n}$ so the number of bits needed to represent the diagonal entries of $\bar{A}$ will be $O(n\beta)$. Since $\left|\det(\bar{A})\right|$ may be as great as the product of the diagonal entries, the number of bits needed to represent $\det(\bar{A})$ can be as large as $O(n\pi)$ (instead of $O(\pi)$) which is not optimal.

The matrices $\bar{A}_\alpha$ in the RF tree of $\bar{A}$ may have rational entries. For a rational number y/x, where x, y are relatively prime integers, define the "modular inverse" $a = (y/x) \bmod p$ to be an integer a, where $0 \leq a < p$ and $y = ax + bp$ for some integer b. Given a rational number y/x, and an integer p, there is no known efficient parallel algorithm to compute the value a. (An algorithm for integer GCD computation is required and there is no known fast parallel algorithm for integer GCD computation.) Notice that p is chosen from the interval $[2(n\|A\|_\infty)^{c_0}/n, 2(n\|A\|_\infty)^{c_0}]$. Given a rational number y/x, it will use at least $O(\log\|A\|)$ time to compute $a = (y/x) \bmod p$ by currently known methods. While this is not entirely clear, it seems that the RF tree $\pmod p$ of A mentioned in the above algorithm is required to have entries that are the modular inverses of the rational entries of $\bar{A}$. Thus, it is not clear that Step 5 can be performed quickly using a realistic model of computation.

In the later sections, a simplified version of Reif's algorithm is shown, which includes two parts. The first part shows that if d is chosen to be greater than $(2n\|A\|)^2$ and $\bar{A} = A + dI$, then the RF tree of $\bar{A}$ can be computed in $O(\log^2 n)$ time using $O(P(n))$ processors. The second part shows that if p is randomly and uniformly chosen from the set of prime numbers in the interval $[n^3\log^2\|A\|, 4n^3\log^2\|A\|]$ and $\bar{A} = A + dI$, where $d = p\hat{d}$ for some integer $\hat{d}$ such that $d \geq (2n\|A\|)^2$, then the RF tree of A can be recovered from the RF tree of $\bar{A}$ in $O(\log^2 n)$ time using $O(P(n))$ processors. The algorithm fails with a very small probability. In both parts, the steps in Reif's algorithm have been somewhat simplified. Thus the improved version of Reif's algorithm in this thesis has optimal bit precision and simplified procedures.

4.3. Computing Exact RF Tree via Approximation

Given a matrix A, let $\bar{A} = A + D$, where $D = dI$ and $d \geq (2n\|A\|)^2$. The exact RF tree of $\bar{A}$ can be computed in $O(\log^2 n)$ time using $O(P(n))$ processors via approximation.

The algorithm includes the following steps, which are described in more detail in Sections 4.3.1 and 4.3.2.

(1) Compute an approximate RF tree of $\bar{A}$ in $O(\log^2 n)$ time using $O(P(n))$ processors;

(2) Compute the exact RF tree from the approximate RF tree in $O(\log n)$ time using $O(n^2/\log n)$ processors.

The algorithm returns exact values for $\det(\bar{A})$, $\bar{A}^{-1}$, and the (extended) RF tree of $\bar{A}$ and replaces steps 2–4 of Reif's algorithm (Algorithm 4.1 in Section 4.2).

4.3.1. Computing the Approximate RF Tree. Let $\bar{A}_\alpha$ be the matrix on the node α in the RF tree of $\bar{A}$. Let I_α be the identity matrix of the same size as $\bar{A}_\alpha$. Let $D_\alpha = dI_\alpha$.

Consider again the approximate Newton iteration of Section 2.2.4, which estimates $\bar{A}_\alpha^{-1}$ where $\bar{A}_\alpha$ is nearly diagonal ($\left\|\bar{A}_\alpha - D_\alpha\right\| \leq d\epsilon/2$) without using the exact value for $\bar{A}_\alpha$: instead a sequence $\{\tilde{A}_\alpha^{(k)}\}$ is given, such that $\left\|\bar{A}_\alpha - \tilde{A}_\alpha^{(k)}\right\| \leq d/2\epsilon^{2^k}$, $\left\|\tilde{A}_\alpha^{(k+1)} - \tilde{A}_\alpha^{(k)}\right\| \leq d/2\epsilon^{2^k}$, and $\tilde{A}^{(0)} = D$, where $0 \leq \epsilon \leq \frac{1}{2}$. Given this sequence of approximations to $\bar{A}_\alpha$, approximations $\tilde{B}_\alpha^{(0)}, \tilde{B}_\alpha^{(1)}, \ldots$ to $\bar{A}_\alpha^{-1}$ are computed. The matrix $\tilde{B}_\alpha^{(0)}$ is assigned to be D_α^{-1} and $\tilde{B}_\alpha^{(k)} = \tilde{B}_\alpha^{(k-1)}(2I - \tilde{A}_\alpha^{(k)}\tilde{B}_\alpha^{(k-1)})$ for $k \geq 1$.

Given the matrix $\bar{A}$, $\bar{A}^{-1}$ can be approximated via exact Newton iterations. In order to compute the RF tree of $\bar{A}$, notice that $\bar{A}_0$ is the principal submatrix of $\bar{A}$; and $\bar{A}_0^{-1}$ is needed to compute $\bar{A}_1$. A simple divide and conquer strategy (in which an accurate approximation of $\bar{A}_0^{-1}$ is computed before the computation of $\bar{A}_1^{-1}$ begins) does not give a fast parallel algorithm for matrix factorization. However, the following algorithm computes a sequence of approximations of $\bar{A}_1$ efficiently by using

less accurate estimates for A_0^{-1} when these are sufficient, and starting the iteration for $\bar{A}_1^{-1}$ as soon as possible.

Define a function $v(\alpha)$ as follows: $v(<>) = 0$, $v(\alpha 0) = v(\alpha)$, and $v(\alpha 1) = 1 + v(\alpha)$, so that $v(\alpha)$ is the number of 1's in the binary string α.

ALGORITHM 4.2. *Computing an approximate RF tree using approximate Newton iteration.*

Input: An $n \times n$ matrix $\bar{A} = A + D$, where $D = dI$ and $d \geq (2n \|A\|)^2$, and a positive integer $t^* \leq \log n$ (n is assumed to be a power of 2).

Output: An approximate augmented RF tree of depth t^* of $\bar{A}$

begin

$h := \log n + 2$

step 1 Initialization:

for all nodes α in the extended RF tree do in parallel

$\tilde{A}_\alpha^{(0)} := D_\alpha$

$\tilde{B}_\alpha^{(0)} := D_\alpha^{-1}$

end for

for all nodes α with $v(\alpha) = 0$ do

for $i := 1$ to h do (sequentially)

Set $\tilde{A}_\alpha^{(i)}$ to be the $n/2^t \times n/2^t$ principal submatrix of $\bar{A}$, for α at depth t.

Set $\tilde{X}_\alpha^{(i)}$, $\tilde{Y}_\alpha^{(i)}$, and $\tilde{Z}_\alpha^{(i)}$ to the upper right, bottom left, and bottom right submatrices of order $n/2^{t+1}$ of $\tilde{A}_\alpha^{(i)}$.

end for

end for

step 2 for $i := 1$ to $(h + \log n + 1)$ do (sequentially)

for all nodes α in the RF tree such that $v(\alpha) < i$ and $i - v(\alpha) \leq h$ do in parallel

if $\alpha = \beta 1$ for some node β then

(Comment: note that $v(\beta 0) = v(\beta) = v(\alpha) - 1$, so $(i - v(\alpha)) = (i - 1 - v(\beta)) = (i - 1 - v(\beta 0))$ and $\tilde{A}_{\beta 0}^{(i-v(\alpha))}$, $\tilde{B}_{\beta 0}^{(i-v(\alpha))}$, $\tilde{X}_{\beta}^{(i-v(\alpha))}$, $\tilde{Y}_{\beta}^{(i-v(\alpha))}$, and $\tilde{Z}_{\beta}^{(i-v(\alpha))}$ have already been computed.)

$$\tilde{A}_{\alpha}^{(i-v(\alpha))} = \tilde{A}_{\beta 1}^{(i-v(\alpha))}$$
$$:= \tilde{Z}_{\beta}^{(i-v(\alpha))} - \tilde{Y}_{\beta}^{(i-v(\alpha))} \tilde{B}_{\beta 0}^{(i-v(\alpha))} \tilde{X}_{\beta}^{(i-v(\alpha))}$$

end if

if $\alpha = \gamma 10^j$ for some node $\gamma 1$ and some positive integer j then

(Comment: $v(\alpha) = v(\gamma 1)$, and $\tilde{A}_{\gamma 1}^{i-v(\alpha)}$ has just been computed.)

Let t be the depth of node α and let matrices $\tilde{A}_{\alpha 0}^{(i-v(\alpha))}$, $\tilde{X}_{\alpha}^{(i-v(\alpha))}$, $\tilde{Y}_{\alpha}^{(i-v(\alpha))}$, and $\tilde{Z}_{\alpha}^{(i-v(\alpha))}$ be the matrices of size $n/2^{t+1} \times n/2^{t+1}$ such that $\begin{bmatrix} \tilde{A}_{\alpha 0}^{(i-v(\alpha))} & \tilde{X}_{\alpha}^{(i-v(\alpha))} \\ \tilde{Y}_{\alpha}^{(i-v(\alpha))} & \tilde{Z}_{\alpha}^{(i-v(\alpha))} \end{bmatrix}$ $= \tilde{A}_{\alpha}^{(i-v(\alpha))}$ is the top left submatrix of size $n/2^t \times n/2^t$ of $\tilde{A}_{\gamma 1}^{(i-v(\alpha))}$.

end if

if $t < \log n$ then

$$\tilde{B}_{\alpha}^{(i-v(\alpha))} := \tilde{B}_{\alpha}^{(i-v(\alpha)-1)} \left(2I - \tilde{A}_{\alpha}^{(i-v(\alpha))} \tilde{B}_{\alpha}^{(i-v(\alpha)-1)}\right)$$

else

$$\tilde{B}_{\alpha}^{(i-v(\alpha))} := (\tilde{A}_{\alpha}^{(i-v(\alpha))})^{-1}$$

(Comment: $\tilde{A}_{\alpha}^{(i-v(\alpha))}$ is a 1×1 matrix, so the inverse can be directly computed.)

end if

end for

end for

end

The algorithm recursively computes an approximate augmented RF tree, working from the "top" node <> down, in the following way. Suppose a sequence of approximations of $\bar{A}_\alpha$ is computed. The sequence of approximations of $\bar{A}_{\alpha 0}$ immediately follows. This sequence of approximations of $\bar{A}_{\alpha 0}$ can be used to compute a sequence of approximations of $\bar{A}_{\alpha 0}^{-1}$ by means of approximate Newton iterations; and this sequence of approximations of $\bar{A}_{\alpha 0}^{-1}$ is used to compute a sequence of approximations of the Schur complement $\bar{A}_{\alpha 1}$. Thus, finally, an approximate augmented RF tree of depth $t^* \leq \log n$ of $\bar{A}$ can be computed.

LEMMA 4.13. *Let $\epsilon = 2\|A\|/d$ with $d \geq (2n\|A\|)^2$. The algorithm generates an approximate augmented RF tree such that the node α has the following properties, for all α at depth $t \leq \log n$ and for $0 \leq k \leq h = \log n + 2$.*

(1) $\begin{cases} \left\| I - \tilde{B}_\alpha^{(k)} \bar{A}_\alpha \right\| \leq (2^{t+1}\epsilon)^{2^k}, & t < \log n; \\ \left\| I - \tilde{B}_\alpha^{(k)} \bar{A}_\alpha \right\| \leq (2^t \epsilon)^{2^k}, & t = \log n. \end{cases}$

(2) $\left\| \bar{A}_\alpha - D_\alpha \right\| \leq \frac{d 2^t \epsilon}{2}$.

(3) $\left\| \bar{A}_\alpha - \tilde{A}_\alpha^{(k)} \right\| \leq \frac{d}{2}(2^t\epsilon)^{2^k}$.

(4) $\left\| \tilde{A}_\alpha^{(k)} - \tilde{A}_\alpha^{(k+1)} \right\| \leq \frac{d}{2}(2^t\epsilon)^{2^k}$.

PROOF. It is easy to verify that at depth $t = 0$, $\bar{A}_{<>} = \bar{A}$ and $D_{<>} = D = dI$. Thus $\left\| \bar{A}_{<>} - D_{<>} \right\| = \|A\| = \frac{d}{2}\epsilon$. Also according to the algorithm, since $v(<>) = 0$, $\tilde{A}_{<>}^{(k)} = \bar{A}$, and hence $\left\| \tilde{A}_{<>}^{(k)} - \bar{A}_{<>} \right\| = 0 \leq \frac{d}{2}\epsilon^{2^k}$; furthermore, $\left\| \tilde{A}_{<>}^{(k)} - \tilde{A}_{<>}^{(k+1)} \right\| = 0 \leq \frac{d}{2}\epsilon^{2^k}$. Thus $\left\| I - \tilde{B}_{<>}^{(k)} \bar{A}_{<>} \right\| \leq (2\epsilon)^{2^k}$ by Lemma 3.5.

Since the depth of the RF tree is less than $\log n$, $2^t \leq n$ for any depth t. Since ϵ is chosen to be $2\|A\|/d$ and $d \geq (2n\|A\|)^2$, so that $\epsilon \leq \frac{1}{2\|A\|n^2}$, $0 \leq 2^t\epsilon \leq \frac{1}{2}$. If, for a node α at depth $t < \log n$, the matrices $\bar{A}_\alpha$ and $\{\tilde{A}_\alpha^{(k)}\}$ satisfy properties (2), (3), and (4), then property (1) follows by Lemma 3.4. Following the steps in the

algorithm, Lemma 3.6 shows that $\left\|\bar{A}_{\alpha 0} - D_{\alpha 0}\right\| \leq \frac{d2^t\epsilon}{2}$, $\left\|\bar{A}_{\alpha 0} - \hat{A}_{\alpha 0}^{(k)}\right\| \leq \frac{d}{2}(2^t\epsilon)^{2^k}$, and $\left\|\hat{A}_{\alpha 0}^{(k)} - \tilde{A}_{\alpha 0}^{(k+1)}\right\| \leq \frac{d}{2}(2^t\epsilon)^{2^k}$, even though $\alpha 0$ has depth $t+1$.

Suppose now that $t+1 < \log n$. Then Lemma 3.7, Lemma 3.8, Lemma 3.9, and Lemma 3.5 show that $\left\|\bar{A}_{\alpha 1} - D_{\alpha 1}\right\| \leq \frac{d2^{t+1}\epsilon}{2}$, $\left\|\bar{A}_{\alpha 1} - \hat{A}_{\alpha 1}^{(k)}\right\| \leq \frac{d}{2}(2^{t+1}\epsilon)^{2^k}$, and $\left\|\hat{A}_{\alpha 1}^{(k)} - \tilde{A}_{\alpha 1}^{(k+1)}\right\| \leq \frac{d}{2}(2^{t+1}\epsilon)^{2^k}$.

When $t = \log n - 1$, since $\tilde{B}_{\alpha 0}^{(k)} = (\hat{A}_{\alpha 0}^{(k)})^{-1}$, $\left\|\bar{A}_{\alpha 0} - D_{\alpha 0}\right\| \leq \frac{d2^t\epsilon}{2}$ and $\left\|\bar{A}_{\alpha 0} - \tilde{A}_{\alpha 0}^{(k)}\right\|$ $\leq \frac{d}{2}(2^t\epsilon)^{2^k}$, it can be shown that $\left\|\tilde{B}_{\alpha 0}^{(k)}\right\| \leq 2/d$ by Lemma 3.1. Then $\left\|I - \tilde{B}_{\alpha 0}^{(k)}\bar{A}_{\alpha 0}\right\|$ $\leq \left\|\tilde{B}_{\alpha 0}^{(k)}\right\| \left\|\tilde{A}_{\alpha 0}^{(k)} - \bar{A}_{\alpha 0}\right\| \leq \frac{2}{d}\frac{d}{2}(2^t\epsilon)^{2^k} = (2^t\epsilon)^{2^k}$. Lemma 3.7 shows that $\left\|\bar{A}_{\alpha 1} - D_{\alpha 1}\right\| \leq$ $\frac{d2^{t+1}\epsilon}{2}$. Similar approaches as the proofs of Lemma 3.8 and Lemma 3.9 can show that $\left\|\bar{A}_{\alpha 1} - \hat{A}_{\alpha 1}^{(k)}\right\| \leq \frac{d}{2}(2^{t+1}\epsilon)^{2^k}$, and $\left\|\hat{A}_{\alpha 1}^{(k)} - \hat{A}_{\alpha 1}^{(k+1)}\right\| \leq \frac{d}{2}(2^{t+1}\epsilon)^{2^k}$. Since $\left\|\bar{A}_{\alpha 1} - D_{\alpha 1}\right\| \leq$ $\frac{d2^{t+1}\epsilon}{2}$ and $\left\|\bar{A}_{\alpha 1} - \tilde{A}_{\alpha 1}^{(k)}\right\| \leq \frac{d}{2}(2^{t+1}\epsilon)^{2^k}$, again it can be shown that $\left\|\tilde{B}_{\alpha 1}^{(k)}\right\| \leq 2/d$ by Lemma 3.1. Then since $\tilde{B}_{\alpha 1}^{(k)} = (\tilde{A}_{\alpha 1}^{(k)})^{-1}$, $\left\|I - \tilde{B}_{\alpha 1}^{(k)}\bar{A}_{\alpha 1}\right\| \leq \left\|\tilde{B}_{\alpha 1}^{(k)}\right\| \left\|\tilde{A}_{\alpha 1}^{(k)} - \bar{A}_{\alpha 1}\right\| \leq$ $\frac{2}{d}\frac{d}{2}(2^{t+1}\epsilon)^{2^k} = (2^{t+1}\epsilon)^{2^k}$. □

Let $\tilde{A}_\alpha$ denote $\tilde{A}_\alpha^{(h)}$ and $\tilde{B}_\alpha$ denote $\tilde{B}_\alpha^{(h)}$, where $h = \log n + 2$.

LEMMA 4.14. *Algorithm 4.2 generates an approximate augmented RF tree such that for a node α in the approximate augmented RF tree,*

(1) $\left\|I - \tilde{B}_\alpha\bar{A}_\alpha\right\| \leq (n\epsilon)^{2^h}$;
(2) $\left\|\bar{A}_\alpha - \tilde{A}_\alpha\right\| \leq \frac{d}{2}(n\epsilon)^{2^h}$.

PROOF. This follows from Lemma 4.13 with the fact $t \leq \log n$. □

LEMMA 4.15. *If the number of approximate Newton iterations h and the value d are chosen as described in Algorithm 4.2, then $\left\|I - \tilde{B}_\alpha\bar{A}_\alpha\right\|(2d)^n \leq (2(n\,\|A\|)^2)^{-n}$ and $\left\|\tilde{A}_\alpha - \bar{A}_\alpha\right\|(2d)^n \leq (2(n\,\|A\|)^2)^{-(n-1)}$.*

PROOF. Lemma 4.14 shows that

$$\begin{aligned}\left\|I - \tilde{B}_\alpha\bar{A}_\alpha\right\|(2d)^n &\leq (n\epsilon)^{2^h}(2d)^n \\ &= \left(\frac{2n\,\|A\|}{d}\right)^{2^h}(2d)^n.\end{aligned}$$

Since $h = \log n + 2$ and $d \geq (2n \|A\|)^2$, $2^h = 4n$ and

$$\begin{aligned} \left\|I - \tilde{B}_\alpha \bar{A}_\alpha\right\| (2d)^n &\leq (2n \|A\|)^{4n} d^{-3n} 2^n \\ &\leq (2n \|A\|)^{4n} (4n^2 \|A\|^2)^{-3n} 2^n \\ &= (2(n \|A\|)^2)^{-n}. \end{aligned}$$

The second claim is proved by the same argument. □

Define an approximate augmented RF tree of the matrix $\bar{A}$ to be a *good approximate augmented RF tree* if

(1) $\left\|A_\alpha - \hat{A}_\alpha\right\| (2d)^n < \frac{1}{2}$,
(2) $\left\|A_\alpha^{-1} - \hat{B}_\alpha\right\| (2d)^n < \frac{1}{2}$,

where $\hat{A}_\alpha$ and $\hat{B}_\alpha$ are, respectively, the matrix in the node α of the approximate RF tree, and the approximation of its inverse.

Since $\bar{m}_\alpha = \det(\hat{A})$, where $\hat{A}$ is a principal $s(\alpha) \times s(\alpha)$ submatrix of $\bar{A}$, and since $\left\|\hat{A}\right\| \leq \left\|\bar{A}\right\| \leq 2d$, $|\bar{m}_\alpha| \leq (2d)^n$; furthermore, for any node α at depth t in the RF tree of $\bar{A}$, $|\bar{m}_\alpha| \leq (2d)^{n-n/2^t}$ by Lemma 4.5 since $s(\alpha) \leq n - n/2^t$.

LEMMA 4.16. *The approximate augmented RF tree computed by Algorithm* 4.2 *has the following properties:*

(1) $\left\|\tilde{A}_\alpha - \bar{A}_\alpha\right\| |\bar{m}_\alpha| \leq \left\|\tilde{A}_\alpha - \bar{A}_\alpha\right\| (2d)^n \leq (2(n \|A\|)^2)^{-n}$;
(2) $\left\|\mathrm{adj}(\tilde{A}) - \mathrm{adj}(\bar{A})\right\| |\bar{m}_\alpha| \leq \left\|\tilde{B} - \bar{A}^{-1}\right\| (2d)^n \leq (2(n \|A\|)^2)^{-(n-1)}$.

PROOF. This is a direct consequence of Lemma 4.14 and Lemma 4.15. □

LEMMA 4.17. *Algorithm* 4.2 *computes a good approximate augmented RF tree in* $O(\log^2 n)$ *time using* $O(P(n))$ *processors.*

PROOF. Lemma 4.16 implies that the approximate RF tree is a good approximate RF tree. It is only necessary to show that the algorithm uses $O(\log^2 n)$ time and $O(P(n))$ processors.

Let $M(n) \leq P(n)\log n$ be the number of operations required for $n \times n$ matrix multiplication using time $O(\log n)$. Since the number of stages of approximate Newton iterations h is chosen to be $\log n + 2$, the algorithm only uses $h + \log n + 1 \in O(\log n)$ stages. Each stage only includes matrix multiplications and matrix additions, which cost $O(\log n)$ time and $O(M(n/2^t))$ operations for a node α at depth t. Since $M(n) \in \Omega(n^2)$, the total cost of a stage of approximate Newton iteration of the whole RF tree is $\sum_{i=0}^{\log n} 2^i M(n/2^i) \in O(M(n))$. Thus the algorithm can be implemented using $O(\log^2 n)$ time and $O(P(n))$ processors. □

REMARK 1. *Step 2 of Algorithm* 4.2 *actually uses a well-known pipelining technique in parallel computation. Since* $\tilde{A}_{\alpha 1}^{(i)}$ *only depends on* $\hat{B}_{\alpha 0}^{(i)}$, $\hat{A}_{\alpha 1}^{(i)}$ *is computed as soon as* $\tilde{B}_{\alpha 0}^{(i)}$ *is available—namely, during the stage immediately after* $\tilde{A}_{\alpha}^{(i)}$ *has been computed (*$\tilde{A}_{\alpha 0}^{(i)}$ *is the top left corner of* $\tilde{A}_{\alpha}^{(i)}$ *and is available as soon as* $\tilde{A}_{\alpha}^{(i)}$ *is).*

Thus the total time is reduced to $O(\log^2 n)$ instead of $O(\log^3 n)$, which would be used if no $\tilde{A}_{\alpha}^{(i)}$ at depth t were computed until after all the $\tilde{A}_{\beta}^{(i)}$'s at depth $t-1$.

4.3.2. Computing the Exact RF Tree. After a good approximate augmented RF tree of $\bar{A}$ has been computed by Algorithm 4.2, the exact extended RF tree of $\bar{A}$ can be computed by multiplying these matrices by the appropriate multipliers, $\bar{m}_\alpha$, and then by rounding the rational numbers that are entries of the resulting matrices to the nearest integers. Since a good approximate RF tree is so close to the RF tree of $\bar{A}$ (Lemma 4.16), this will correctly recover the (exact) extended RF tree of $\bar{A}$ from the good approximate augmented RF tree of $\bar{A}$.

ALGORITHM 4.3. *Computing the exact RF tree of* $\bar{A}$.

Input: A good approximate augmented RF tree of depth $t = \log n$ of $\bar{A}$

Output: Exact extended RF Tree of $\bar{A}$

begin

step 1 Compute the determinants of the matrices in $\{\tilde{A}_\alpha\}$

for all nodes α of depth t do in parallel

$det_\alpha := (\hat{A}_\alpha)_{11}$ ($\hat{A}_\alpha$ is an 1×1 matrix)

end for

for $s := t-1$ down to 0 do (sequentially)

for all nodes α of depth t do in parallel

$det_\alpha := det_{\alpha 0} \times det_{\alpha 1}$

end for

end for

step 2 Compute integer multipliers $\{\bar{m}_\alpha\}$ such that the matrices $\bar{m}_\alpha A_\alpha$ all have integer entries

$\bar{m}_{<>} := 1$

for $s := 0$ to $t-1$ do (sequentially)

for all nodes α of depth s do in parallel

$\bar{m}_{\alpha 0} := \bar{m}_\alpha$

$\tilde{m}_{\alpha 1} := \bar{m}_\alpha \times det_{\alpha 0}$

$\bar{m}_{\alpha 1} := \tilde{m}_{\alpha 1}$ rounded to the nearest integer

$\det(\bar{A}_{\alpha 0}) := \bar{m}_{\alpha 1}/\bar{m}_\alpha$

$t'_{\alpha 1} := \bar{m}_{\alpha 1} det_{\alpha 1}$

$t_{\alpha 1} := t'$ rounded to the nearest integer

$\det(\bar{A}_{\alpha 1}) := t_{\alpha 1}\ /\bar{m}_{\alpha 1}$

end for

end for

step 3 for $s := 0$ to t do (sequentially)

for all nodes α of depth s do in parallel

Set $\hat{A}_\alpha$ to be the integer matrix obtained by rounding each of the entries of $(\bar{m}_\alpha \tilde{A}_\alpha)$ to the nearest integer;

$\bar{A}_\alpha := \frac{1}{\bar{m}_\alpha}\hat{A}_\alpha$

end for

end for

step 4 for $s := 0$ to t do (sequentially)

for all nodes α of depth s do in parallel

Set $\mathrm{adj}\hat{A}_\alpha$ to be the integer matrix obtained by rounding each of the entries of $(\bar{m}_\alpha \det(\bar{A}_\alpha)\hat{B}_\alpha)$ to the nearest integer;

$\mathrm{adj}(\bar{A}_\alpha) := \frac{1}{\bar{m}_\alpha}\mathrm{adj}\hat{A}_\alpha$

end for

end for

end

The algorithm works in a bottom up manner in Step 1, moving from leaves to the root in Step 1 to compute the determinant $\det_\alpha$ of $\tilde{A}_\alpha$ for every node in the approximate RF tree. This step only involves multiplications of rational numbers and requires $O(\log n)$ time and $O(n/\log n)$ processors. Step 2 is similar. Instead the computation is top down, moving from the root to the leaves, and has the same cost. Step 3 requires multiplications of rational numbers by all entries of all the matrices in the RF tree. However, there are only $O(n^2)$ entries of all the matrices in the RF tree. Thus Step 3 uses $O(\log n)$ time and $O(n^2/\log n)$ processors. Step 4 is similar to Step 3.

LEMMA 4.18. *Algorithm* 4.3 *computes the (exact) extended RF tree of* $\bar{A}$ *in* $O(\log n)$ *time using* $O(n^2/\log n)$ *processors.*

PROOF. Define $E(t)$ to be the maximum error of $\det(\tilde{A}_\alpha)$ for any α at depth t:

$$E(t) = \max_{\alpha, depth(\alpha)=t} \left|\det(\bar{A}_\alpha) - det_\alpha\right|.$$

Since $\left\|\tilde{A}_\beta\right\| \leq 2d$, for any 1×1 matrix $\tilde{A}_\beta$, and from the algorithm, det_α is computed from the bottom up, it is clear that $|\det_\alpha| \leq (2d)^{\frac{d}{2^t}}$ for any node α at depth t. Also

because $\left\|\bar{A}_\alpha\right\| \leq 2d$, it is clear by the Hadamard inequality that $\left|\det(\bar{A}_\alpha)\right| \leq (2d)^{\frac{n}{2^t}}$. Thus

$$\begin{aligned} E(t) &= \max_\alpha \left|\det(\bar{A}_\alpha) - det_\alpha\right| \\ &= \max_\alpha \left|\det(\bar{A}_{\alpha 0})\det(\bar{A}_{\alpha 1}) - det_{\alpha 0} det_{\alpha 1}\right| \\ &\leq \max_\alpha \left(\left|(\det(\bar{A}_{\alpha 0}) - det_{\alpha 0})\det(\bar{A}_{\alpha 1})\right| + \left|(\det(\bar{A}_{\alpha 1}) - det_{\alpha 1}) det_{\alpha 0}\right|\right) \\ &\leq 2(2d)^{\frac{n}{2^{t+1}}} E(t+1). \end{aligned}$$

This recurrence shows that $E(0) \leq n(2d)^{n-1}E(\log n)$ and $E(t) \leq n(2d)^{n/2^t-1}E(\log n)$. The error $E(\log n)$ is the error in the estimate of the determinant of a 1×1 matrix, which is exactly $\left\|\bar{A}_\beta - \tilde{A}_\beta\right\|$, where β is a leaf node. Thus

$$E(\log n)(2d)^n = \max_{\beta \text{ at depth } \log n} \left\|\bar{A}_\beta - \tilde{A}_\beta\right\| (2d)^n < \frac{1}{2},$$

because the input is a good approximate augmented RF tree. Then

$$E(0) \leq n(2d)^{n-1}E(\log n) \leq \frac{n}{2d} E(\log n)(2d)^n < \frac{1}{2},$$

since $d \geq (2n\|A\|)^2 > n$, and

$$E(t)(2d)^{n-n/2^t} \leq n(2d)^{n-1}E(\log n) < \frac{1}{2}, \tag{4.3}$$

for any depth t. Again, rounding $det_{<>}$ to the nearest integer will yield the exact value of $\det(\bar{A})$, since $\det(\bar{A})$ is an integer.

For any α at depth t, suppose $\bar{m}_\alpha$ has been computed correctly—that is, suppose $\bar{m}_\alpha$ is the determinant of the leading principal submatrix of A of size $\sum_{i=1}^{s} \frac{n}{2^i}\alpha_i$, provided that $\alpha = \alpha_1\alpha_2\ldots\alpha_s$ for $0 \leq \log n$ and for $\alpha_i \in \{0,1\}$. Since $\bar{m}_{\alpha 0} = \bar{m}_\alpha$, $\bar{m}_{\alpha 0}$

is obviously correct. As well, $\hat{m}_{\alpha 1} = \bar{m}_\alpha \det_{\alpha 0}$, so

$$\begin{aligned} |\hat{m}_{\alpha 1} - \bar{m}_\alpha| &= \left|\bar{m}_\alpha(\det(\bar{A}_{\alpha 0}) - det_{\alpha 0})\right| \\ &\leq |\bar{m}_\alpha| E(t+1) \\ &\leq (2d)^{n-n/2^t} E(t+1) \\ &\leq (2d)^{n-n/2^{t+1}} E(t+1) < \frac{1}{2} \end{aligned}$$

by Equation 4.3. Therefore, $\bar{m}_{\alpha 1}$ can be obtained by rounding $\hat{m}_{\alpha 1}$ to the nearest integer as well.

When $\bar{m}_\alpha$ have been computed for all α in the RF tree of $\bar{A}$, the final steps of the algorithm will yield the exact extended RF tree of $\bar{A}$ by rounding each of the entries of the matrix $(\bar{m}_\alpha \tilde{A}_\alpha)$ and its adjoint $|\bar{m}_\alpha| \operatorname{adj}(\bar{A}_\alpha)$ to the nearest integer and then dividing the entry by $\bar{m}_\alpha$ to recover each of the entries of $\bar{A}_\alpha$ and its adjoint (see Lemma 4.6). The bounds for time and the number of processors are clear. □

LEMMA 4.19. *Given an $n \times n$ integer matrix A, where $\|A\| \leq 2^\beta$ and $\beta = n^c$, let $\bar{A} = A + dI$, where d is an integer and $d \geq (2n\|A\|)^2$. Then the extended RF tree of $\bar{A}$ can be exactly computed in $O(\log^2 n)$ time using $O(P(n))$ processors.*

PROOF. This is a direct consequence of Lemma 4.17 and Lemma 4.18. □

4.4. Recovering the RF tree of A

Given an $n \times n$ symmetric positive definite matrix A, assume again that $\|A\| \leq 2^{\beta}$, where $\beta \leq n^c$, for some constant c, and let $\bar{A} = A + dI$, where $d \geq (2n\|A\|)^2$. It has been shown in Section 4.3 that the exact extended RF tree of $\bar{A}$ can be computed in $O(\log^2 n)$ time using $O(P(n))$ processors.

If x is an integer such that $-y \leq x \leq y$, for some positive number y, then x can be determined from $x \bmod p$ if $p > 2y$. If d is chosen to be greater than $2n^2\|A\|^n$, then since it is known that $|m_\alpha| \leq \|A\|^n$, each integer multiplier m_α can be determined from its residue mod d. Also because $\|m_\alpha A_\alpha\| \leq n^2\|A\|^n$, all the entries of $m_\alpha A_\alpha$ can be easily recovered. Thus the RF tree of A could be recovered from the entries of the matrices in the RF tree of $\bar{A}$ modulo d for $d > 2n^2\|A\|^n$. As well, $\det(A) \leq \|A\|^n$, and $\det(A) \equiv \det(\bar{A}) \bmod d$. However, the use of such a large value for d would increase the bit complexity of the algorithm and affect the time complexity of the algorithm as well.

An alternative approach is to use a smaller value for d (as in Section 4.3) and then use a second step to "lift" the RF tree of A from its residue mod d. In order to do this, it is necessary to restrict the set of values for d that can be used. Let p be a prime number randomly and uniformly chosen from the set of prime numbers in the interval $[n^3\log^2\|A\|, 4n^3\log^2\|A\|]$. Let $\hat{d}$ be some number such that $p\hat{d} \geq (2n\|A\|)^2$. Let $\bar{A} = A + p\hat{d}I$. The exact RF tree of $\bar{A}$ can be computed (by Lemma 4.19) in $O(\log^2 n)$ time using $O(P(n))$ processors. In this section, it is shown that the exact RF tree of A can be recovered from the exact RF tree of $\bar{A}$ with high probability by Newton Hensel Lifting.

4.4.1. Reducing Rational Entries to Integer Entries. Given the (extended) RF tree of $\bar{A}$, the goal is to recover the (extended) RF tree of A.

For a rational number y/x, where x, y are relatively prime integers and where $\gcd(p, x) = 1$, define $a = (y/x) \bmod p$ to be an integer a such that $0 \leq a < p$

and $y = ax + bp$ for some integer p. Given y/x, if p and x are relatively prime, $a = (y/x) \bmod p$ can be computed as follows:

(1) Compute integers s and t such that $sx + tp = 1$;

(2) Compute $a = (ys) \bmod p$.

The cost of this computation is dominated by the the first step. However, if p is a prime number less than or equal to $4n^3 \log^2 \|A\|$, then it is relatively "small", because it is given that $\|A\| \leq 2^\beta$, where $\beta \leq n^c$, for some constant c. The first step can be performed by using the Extended Euclidean algorithm [AHU74] in $O(\log^2 n)$ time, because the binary representation of p has length $O(\log n)$. Now, given any $n \times n$ matrix $A = [a_{ij}]$ with rational entries whose denominators do not divide p, the matrix $A \bmod p$ can be defined to be the matrix with entries $a_{ij} \bmod p$ for $1 \leq i, j \leq n$, and the "RF tree (mod p)" of matrix A can be defined similarly.

Since the RF tree of $\bar{A}$ is available, $O(\log^2 n)$ time and $O(n^2)$ processors are sufficient to obtain the RF tree (mod p) of A.

4.4.2. Newton-Hensel Lifting. Fix a nonsingular $n \times n$ matrix A and a prime number p. Assume $A^{-1} \bmod p$ is given. Newton-Hensel Lifting is the following.

Algorithm: Newton-Hensel Lifting

Input: A positive number k, and $n \times n$ matrices A and $A^{-1} \bmod p$.

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

begin

step 1 $S^{(0)} := A^{-1} \bmod p$.

step 2 for $i := 1$ to k do

$$S^{(k)} := S^{(k-1)}(2I - AS^{(k-1)}) \bmod p^{2^k}$$

end for

end

The algorithm is similar to Newton iteration. Each iteration of Newton Hensel Lifting will produce a "closer" approximation of A^{-1} (where two integers a and b are now considered to be "close" if their difference is divisible by a higher power of p). The algorithm computes a sequence of approximations, which "converges" quadratically.

LEMMA 4.20. [MC79] $S^{(k)} = A^{-1} \bmod p^{2^k}$.

PROOF. Let E_k be an "error" matrix (with rational entries, whose denominators do not divide the prime number p) such that $AS^{(k)} = I - p^{2^k}E_k$. Note that $S^{(k+1)} = S^{(k)}(2I - AS^{(k)}) = S^{(k)}(I + p^{2^k}E_k)$. It follows that

$$\begin{aligned} AS^{(k+1)} &= AS^{(k)}(I + p^{2^k}E_k) \\ &= (I - p^{2^k}E_k)(I + p^{2^k}E_k) \\ &= I - p^{2^{k+1}}E_k^2 = I \bmod p^{2^{k+1}}. \end{aligned}$$

The lemma follows by induction. □

Apply Newton Hensel Lifting to the RF tree. Assume that the matrix A and its (extended) RF tree (mod p) of A are given. Recall that the (extended) RF tree has $\bar{A}_\alpha$ and $\bar{A}_\alpha^{-1}$ in the node α. The node α in the (extended) RF tree (mod p) of A includes A_α mod p and A_α^{-1} mod p. The following algorithm produces an RF tree (mod p^{2^k}) of A for any given k.

ALGORITHM 4.4. *Newton Hensel Lifting for an RF tree.*

Input: The matrix A, the augmented RF tree (mod p) of A, and an integer $k \geq 1$.

Output: Augmented RF tree (mod p^{2^k}) of A.

begin

step 1 Initialization:

for all nodes α with $v(\alpha) = 0$ do

$A_\alpha^{(0)} := A_\alpha \bmod p$

for $i := 0$ to h do (sequentially)

Set $A_\alpha^{(i)}$ to be the $n/2^t \times n/2^t$ principal submatrix of A, for α at depth t.

Set $X_\alpha^{(i)}$, $Y_\alpha^{(i)}$, and $Z_\alpha^{(i)}$ to be the upper right, bottom left, and bottom right submatrices of $A_\alpha^{(i)}$.

end for

end for

for all nodes α in the RF tree do in parallel

$B_\alpha^{(0)} := A_\alpha^{-1} \bmod p$

end for

step 2 for $i := 1$ to $k + \log n + 1$ do (sequentially)

for all nodes α in the RF tree such that $v(\alpha) < i$ and $i - v(\alpha) \leq h$ do in parallel

if $\alpha = \beta 1$ for some node β then

(Comment: note that $v(\beta 0) = v(\beta) = v(\alpha) - 1$, so $(i - v(\alpha)) = (i - 1 - v(\beta)) = (i - 1 - v(\beta 0))$ and $A_{\beta 0}^{(i-v(\alpha))}$, $B_{\beta 0}^{(i-v(\alpha))}$, $X_\beta^{(i-v(\alpha))}$, $Y_\beta^{(i-v(\alpha))}$, and $Z_\beta^{(i-v(\alpha))}$ have already been computed.)

$$A_\alpha^{(i-v(\alpha))} = A_{\beta 1}^{(i-v(\alpha))} := Z_\beta^{(i-v(\alpha))} - Y_\beta^{(i-v(\alpha))} B_{\beta 0}^{(i-v(\alpha))} X_\beta^{(i-v(\alpha))} \bmod p^{2^{(i-v(\alpha))}}$$

end if

if $\alpha = \gamma 10^j$ for some node $\gamma 1$ and some positive integer j then

(Comment: $v(\alpha) = v(\gamma 1)$, and $A_{\gamma 1}^{i-v(\alpha)}$ has just been computed.)

Let t be the depth of node α and let matrices $A_{\alpha 0}^{(i-v(\alpha))}$, $X_\alpha^{(i-v(\alpha))}$, $Y_\alpha^{(i-v(\alpha))}$, and $Z_\alpha^{(i-v(\alpha))}$ be the matrices of

size $n/2^{t+1} \times n/2^{t+1}$ such that $\begin{bmatrix} A_{\alpha 0}^{(i-v(\alpha))} & X_{\alpha}^{(i-v(\alpha))} \\ Y_{\alpha}^{(i-v(\alpha))} & Z_{\alpha}^{(i-v(\alpha))} \end{bmatrix}$
$= A_{\alpha}^{(i-v(\alpha))}$ is the top left submatrix of size $n/2^t \times n/2^t$
of $A_{\gamma 1}^{(i-v(\alpha))}$.

end if

$B_{\alpha}^{(i-v(\alpha))} := B_{\alpha}^{(i-v(\alpha)-1)} \left(2I - A_{\alpha}^{(i-v(\alpha))} B_{\alpha}^{(i-v(\alpha)-1)}\right)$
$\mod p^{2^{(i-v(\alpha))}}$

end for

end for

end

Since p is chosen to be at least $n^3 \log^2 \|A\|$, and $\|A\| \leq 2^{\beta}$, where $\beta \leq n^c$, for some constant c, $p^{2^k} > 2n^2 \|A\|^n$ for $k \geq 1 + \log n + \log\log \|A\|$; in particular, one can use $k \in O(\log n)$, where the hidden multiplicative constant is less than $c+2$. Now an RF (mod p^{2^k}) tree of A has been computed, where $p^{2^k} > 2n^2 \|A\|^n$.

LEMMA 4.21. *Given an $n \times n$ matrix A and its augmented RF tree* (mod p), *the RF tree* (mod p^{2^k}) *of A can be computed in $O((k + \log n) \log n)$ parallel time using $O(P(n))$ processors.*

PROOF. Let $M(n) \leq P(n) \log n$ be the number of operations required for $n \times n$ matrix multiplication using time $O(\log n)$. Since the number of Newton Hensel Lifting stages $k \in O(\log n)$, the algorithm uses $k + \log n + 1 \in O(\log n)$ stages. Each stage only includes matrix multiplications and matrix additions, which cost $O(\log n)$ time and $O(M(n/2^t))$ operations for a node α at depth t. Since $M(n) \in \Omega(n^2)$, the total number of operations used by a stage of Newton Hensel Lifting of the whole RF tree is at most $\sum_{i=0}^{\log n} 2^i M(n/2^i) \in O(M(n))$. Thus the algorithm uses $O(\log^2 n)$ time and $O(M(n) \log n)$ operations which implies the above time and processor bound. □

The following algorithm recovers the exact RF tree of A.

ALGORITHM 4.5. *Recovering the RF tree of A.*

Input: The RF tree (mod p^{2^k}) of A such that $p^{2^k} > 2n^2 \|A\|^n$

Output: $\det(A)$ and Exact RF Tree of A

begin

step 1 Compute the determinant of the matrices $A_\alpha \bmod p^{2^k}$ in the RF tree:

for all α at depth $\log n$ do in parallel

$(\det(A_\alpha \bmod p^{2^k})) := (A_\alpha)_{11} \bmod p^{2^k}$

(Comment: A_α is a 1×1 matrix)

end for

for $s := \log n - 1$ down to 0 do (sequentially)

for all nodes of depth s do in parallel

$(\det(A_\alpha \bmod p^{2^k})) := \det(A_{\alpha 0}) \det(A_{\alpha 1}) \bmod p^{2^k}$;

end for

end for

(Comment: $\det(A) \bmod p^{2^k}$ is computed at this stage.)

if $(\det(A_{<>}) \bmod p^{2^k}) \leq \frac{1}{2} p^{2^k}$ then

$\det(A_{<>}) := (\det(A_{<>}) \bmod p^{2^k})$;

else

$\det(A_{<>}) := p^{2^k} - (\det(A_{<>}) \bmod p^{2^k})$;

end if

(Comment: Since $|\det(A)| \leq \|A\|^n$, the exact value of $\det(A)$ can be computed.)

step 2 Compute integer multipliers m_α for all α in the RF tree:

$m_{<>} := 1$;

for $s := 0$ to $\log n - 1$ do (sequentially)

for all nodes at depth s do in parallel

$m_{\alpha 0} := m_\alpha$;

$m_{\alpha 1} := m_\alpha \ (\det(A_{\alpha 0}) \bmod p^{2^k})$;

(Comment: $m_{\alpha 1} \equiv m_{\alpha 1} \bmod p^{2^k}$ has been computed.)

if $m_{\alpha 1} > \frac{1}{2}p^{2^k}$ then $m_{\alpha 1} := p^{2^k} - m_{\alpha 1}$;

end if

end for

end for

(Comment: since $|m_\alpha| \leq \|A\|^n$, the exact values of m_α can be obtained.)

step 3 Compute A_α for all nodes α in the RF tree:

for $s := 0$ to $\log n$ do (sequentially)

for all α at depth s do in parallel

$A'_\alpha := (m_\alpha)(A_\alpha \bmod p^{2^k})$;

For all the entries a_{ij} in A'_α

if $a_{ij} > \frac{1}{2}p^{2^k}$ then $a_{ij} := p^{2^k} - a_{ij}$;

end if

$A_\alpha := A'_\alpha / m_\alpha$;

end for

end for

(**Comment:** Lemma 4.7 shows that $\|m_\alpha A_\alpha\| \leq n^2 \|A\|^n$, and all the entries in $m_\alpha A_\alpha$ have absolute value at most $n^2 \|A\|^n$, so the above stage computes $m_\alpha A_\alpha$ exactly. The RF tree of A is therefore recovered.)

end

The above algorithm only fails when p is a divisor of m_α for one or more α; however, the following lemma shows that this happens with a small probability.

LEMMA 4.22. *Let p be a prime randomly and uniformly chosen from the set of prime numbers in the interval $[n^3 \log^2 \|A\|, 4n^3 \log^2 \|A\|]$. If A is symmetric and*

positive definite, then $m_\alpha = 0 \bmod p$ *with probability* $O(\log n/(n \log \|A\|))$.

PROOF. Let $\pi(x)$ be the number of prime numbers less than x, then $\pi(x) = O\left(\frac{x}{\log x}\right)$ (see, for example, [Zip93]). Furthermore, a tighter bound for $\pi(x)$ is also known (also [Zip93]):

$$0.92\frac{x}{\log x} < \pi(x) < 1.105\frac{x}{\log x}.$$

Assuming $n \geq 2$, one can verify that there are at least $cn^3 \log^2 \|A\| / \log(n^3 \log^2 \|A\|)$ prime numbers in the interval $[n^3 \log^2 \|A\|, 4n^3 \log^2 \|A\|]$, for some constant $c \approx 0.73$.

On the other hand, there are at most $n-1$ distinct m_α's (not including $m_{<>} = 1$), since each m_α is the determinant of a principal submatrix of A. Also because A is symmetric and positive definite, $m_\alpha \neq 0$. Clearly the determinant of the $i \times i$ principal submatrix has absolute value at most $\|A\|^i$. Thus there are at most $\frac{n^2}{2} \log \|A\|$ prime numbers that divide m_α, for one or more α's. Since p is uniformly and randomly chosen from the set of the prime numbers in the given interval, the probability that p divides m_α, for one or more α's, is

$$\begin{aligned} \Pr(p|m_\alpha) &\leq \frac{\frac{n^2}{2} \log \|A\| \cdot \log(n^3 \log^2 \|A\|)}{cn^3 \log^2 \|A\|} \\ &\leq \frac{\hat{c}(\log n + \log\log \|A\|)}{n \log \|A\|}, \end{aligned}$$

for some constant $\hat{c} > 2$. Thus the lemma follows. □

The following theorem is a direct result of the previous two.

THEOREM 4.1. *Given a symmetric and positive definite* $n \times n$ *integer matrix* A *such that* $\|A\| \in 2^{n^{O(1)}}$, *there exists a randomized algorithm (Monte Carlo) that computes the RF tree of* A *in* $O(\log^2 n)$ *time using* $O(P(n))$ *processors. The algorithm fails with probability* $O(\log n/(n \log \|A\|))$.

Assume $\|A\| \leq 2^\beta$, for the input matrix A. When $\beta \geq \log n$, the number of bits needed to represent the inverse and each entry in the LU factorization is at least

$\Omega(n\beta)$ in the worst case, because the determinant for a triangular matrix with the entries of magnitude (2^{β}) has value $2^{\Omega(n\beta)}$. From the analysis of the algorithm, it is sufficient to choose $d \in O(n^2 \log \|A\|)$, which can be represented using $O(\beta + \log n)$ bits. It is also shown that only $O(\log n)$ Newton iteration stages are required in the computation. Observe that the algorithm never uses values that are represented using more that $O(n(\log n + \beta))$ bits and this large number of bits is only used at the end of the approximate Newton iteration stages and Newton Hensel Lifting stages. Thus the bit precision of the algorithm is $O(\pi)$, where $\pi = n(\beta + \log n)$, which is optimal, when $\beta \geq \log n$.

CHAPTER 5

Toeplitz and Toeplitz-like Matrix Computations

Fast sequential algorithms for Toeplitz and Toeplitz-like matrix computations require $O(n \log^2 n)$ steps [BGY80, BA80, Mor80, Kal94]. However, known fast parallel algorithms for general Toeplitz and Toeplitz-like matrix computations still require $\Omega(n^2 \log^2 n)$ operations using the parameterized Newton iteration [Pan90b, Pan92d, BP93]. Numerical algorithms compute the inverse of an $n \times n$ Toeplitz or Toeplitz-like matrix in $O(\log^2 n)$ time using $O(n \log^2 n)$ operations, but they require that a good initial approximation to a short displacement generator for the inverse of the input matrix is readily available or that the input matrix is well-conditioned [PR87, Pan92c, Pan92b].

Reif gives the first processor efficient parallel algorithm for the exact solution for Toeplitz and Toeplitz-like matrix computations [Rei95] with integer matrices as inputs. Modifications of the algorithm, similar to the changes made to Reif's algorithm for dense unstructured matrix computations, are considered, and it is conjectured that bit precision for this algorithm can also be improved.

In the beginning of this chapter, the concept of displacement rank and the definition of Toeplitz-like matrices are introduced. Then some known fast parallel algorithms for Toeplitz and Toeplitz-like matrix computations are briefly surveyed. Finally Reif's algorithm and ideas for its modification are discussed.

5.1. Displacement Ranks and Toeplitz-like Matrices

In this section, definitions of displacement rank and Toeplitz-like matrices are given. Some important properties of displacement rank are introduced.

5.1.1. Displacement Ranks. It is well-known that computing the product of a Toeplitz matrix and a vector is computationally equivalent to the computation of the product of two polynomials with degrees no more than twice the dimension of the Toeplitz matrix ([BP94], pp. 137–138). Polynomial multiplication with coefficients over a field supporting the Fast Fourier Transform (FFT) can be performed in $O(\log n)$ time using $O(n)$ processors; whereas, $O(\log n \log\log n)$ time is required when the coefficients are over an arbitrary ring assuming one can perform exact division by some small prime [CK87]. Since only an $O(\log\log n)$ factor is added in the general case, it is assumed that such a computation can be performed in $O(\log n)$ time using $O(n)$ processors in the later discussion. An attempt to generalize the properties of Toeplitz matrices leads to the concept of displacement rank [KKM79].

Notice that a Toeplitz matrix is uniquely defined by its first column and last column. Let $L(x)$ denote a lower-triangular Toeplitz matrix whose first column is x and let $U(y^T)$ denote an upper triangular Toeplitz matrix whose first row is y^T. It is easy to check that any Toeplitz matrix T can be represented as

$$T = L(v)I + IU(u^T), \tag{5.1}$$

where v is the first column of T and u is the last column of T with the last entry set to 0 and the order reversed.

DEFINITION 5.1. *The (+)-displacement rank of an $n \times n$ matrix A is the smallest integer $\alpha_+(A)$ such that*

$$A = \sum_{i=1}^{\alpha_+(A)} L_i U_i$$

for some lower triangular Toeplitz matrices $\{L_i\}$ and upper-triangular Toeplitz matrices $\{U_i\}$. The (−)-displacement rank of an $n \times n$ matrix A is the smallest integer

$\alpha_-(A)$ such that

$$A = \sum_{i=1}^{\alpha_-(A)} U_i L_i$$

for some upper-triangular Toeplitz matrices $\{U_i\}$ and lower triangular Toeplitz matrices $\{L_i\}$.

If T is a Toeplitz matrix then $\alpha_+(T) \leq 2$ and $\alpha_-(T) \leq 2$ by Equation 5.1. A general matrix A has displacement ranks no more than n. If the displacement ranks of a matrix A are in $O(n^\alpha)$, where $0 \leq \alpha < 1$, then there are two advantages for computations:

(1) The matrix can be represented by $O(n^{1+\alpha})$ elements instead of $O(n^2)$ elements;

(2) The product of the matrix and a vector can be computed in $O(\log n)$ time using $O(n^{1+\alpha})$ processors based on the fact that the product of a Toeplitz matrix and a vector can be computed in $O(\log n)$ time using $O(n)$ processors.

Recall that Z is the $n \times n$ matrix with 1's immediately below the diagonal and zeroes everywhere else. The following lemma suggests equivalent definitions for displacement rank—as the (usual) rank of "shifted" matrices related to A.

LEMMA 5.1. [KKM79] *Given column vectors $\{x_i, y_i, i = 1, \ldots, \alpha\}$, the functional equation (for an "unknown" matrix A)*

$$A - ZAZ^T = \sum_{i=1}^{\alpha} x_i y_i^T$$

has the unique solution

$$A = \sum_{i=1}^{\alpha} L(x_i) U(y_i^T),$$

where $L(x)$ denotes a lower-triangular Toeplitz matrix whose first column is x, and $U(y^T)$ denotes an upper-triangular Toeplitz matrix whose first row is y^T.

Similarly, the functional equation.

$$A - Z^T AZ = \sum_{i=1}^{\alpha} x_i y_i^T$$

has the unique solution

$$A = \sum_{i=1}^{\alpha} U(\tilde{x}_i^T) L(\tilde{y}_i),$$

where $\tilde{x}^T = [x_n, \ldots, x_1]$ *when* $x^T = [x_1, \ldots, x_n]$.

LEMMA 5.2. [KKM79] *The* $(\pm)$*-displacement ranks can be computed as*

$$\alpha_+(A) = \text{rank}(R - ZAZ^T), \qquad \alpha_-(A) = \text{rank}(A - Z^T AZ).$$

Let ϕ be a function $\phi : F^{m\times n} \to F^{m\times n}$. Call $\phi_+(A) = A - ZAZ^T$ and $\phi_-(A) = A - Z^T AZ$ $(\pm)$*-displacement functions.* Recall that a generator for a matrix A is the pair of matrices G, H such that $A = GH^T$. Define a generator of length d of $\phi_\pm(A)$ to be a $(\pm)$*-displacement generator of length* d of A. Since the minimum length of a generator of a matrix is the rank of the matrix, the minimum length of a $(\pm)$-displacement generator of matrix A is the $(\pm)$-displacement rank of A. When the above definitions appear without $\pm$, one can assume either $+$ or $-$ may apply to the definitions.

5.1.2. Important Properties.

LEMMA 5.3. *If* A, B *are* $n \times n$ *matrices then* $\text{rank}(I - AB) = \text{rank}(I - BA)$.

PROOF. It is sufficient to show that $I - AB$ and $I - BA$ have the same nullity.

Let $\{x_1, x_2, \ldots, x_r\}$ be a maximal linearly independent set of vectors satisfying

$$(I - AB)x_i = 0, \tag{5.2}$$

and let vectors $y_i = Bx_i$, for $1 \le i \le r$. Thus

$$(I - BA)y_i = y_i - B(ABx_i) = Bx_i - Bx_i = 0.$$

It can also be shown that all y_i's are independent: Suppose there are scalars α_i such that $\sum_{i=1}^r \alpha_i y_i = 0$, so $B\sum_{i=1}^r \alpha_i x_i = 0$. Equation 5.2 shows that $\sum_{i=1}^r \alpha_i x_i = AB\sum_{i=1}^r \alpha_i x_i = 0$, which shows that the independence of the set $\{x_1, x_2, \ldots, x_r\}$

implies the independence of $\{y_1, y_2, \ldots, y_r\}$. A similar argument with the roles of A and B interchanged completes a one-to-one correspondence between the null vectors of $I - AB$ and $I - BA$, so that $I - AB$ and $I - BA$ have the same nullity and hence have the same rank. □

LEMMA 5.4. [KKM79] *The* $(\pm)$*-displacement rank of a nonsingular matrix* A *is equal to the* $(\mp)$*-displacement rank of its inverse, i.e.,*

$$\alpha_+(A) = \alpha_-(A^{-1}) \quad \textit{and} \quad \alpha_-(A) = \alpha_+(A^{-1}).$$

PROOF. Since the rank of an arbitrary matrix is unaffected by multiplication by a nonsingular matrix,

$$\begin{aligned}
\alpha_-(A^{-1}) &= \operatorname{rank}(A^{-1} - Z^T A^{-1} Z) \\
&= \operatorname{rank}((A^{-1} - Z^T A^{-1} Z^T)A) \\
&= \operatorname{rank}(I - Z^T A^{-1} ZA) \\
&= \operatorname{rank}(I - ZAZ^T A^{-1}) \quad \text{(by Lemma 5.3)} \\
&= \operatorname{rank}((I - ZAZ^T A^{-1})A) \\
&= \operatorname{rank}(A - ZAZ^T) \\
&= \alpha_+(A).
\end{aligned}$$

A similar argument will establish that

$$\alpha_+(A^{-1}) = \alpha_-(A).$$

□

There is an explicit formula for converting a $\sum LU$ representation to a $\sum UL$ representation [KKM79, BA80]. Let $x, y \in R^n$, for some ring R. Then

$$L(x)U(y^T) = IL(\hat{y}) + U(\hat{x}^T)I - U((ZJx)^T)L(ZJy), \tag{5.3}$$

where $\hat{y}^T$ is the reversed last row of $L(x)U(y^T)$, and $\hat{x}$ is the reversed last column of $L(x)U(y^T)$ with the first entry set to 0. The dual formula is

$$U(y^T)L(x) = L(\check{x})I + IU(\check{y}^T) - L(ZJy)U((ZJx)^T), \tag{5.4}$$

for $x, y \in R^n$, where $\check{y}^T$ is the first row of $U(y^T)L(x)$, and $\check{x}$ is the first column of $U(y^T)L(x)$ with its first entry set to 0. This establishes the following.

LEMMA 5.5. $|\alpha_+(A) - \alpha_-(A)| \leq 2$.

5.1.3. Toeplitz-like Matrices. A matrix that has constant displacement ranks is called a *Toeplitz-like matrix*. Many efficient algorithms for Toeplitz matrix computations can be extended to Toeplitz-like matrix computations.

Block matrices with Toeplitz blocks, such as the Sylvester matrix corresponding to the resultant of two univariate polynomials, are Toeplitz-like; the product of two Toeplitz matrices is Toeplitz-like, because it can be shown to have (+)-displacement rank at most 4 by Equation 5.1 and Equation 5.3. The inverse of a Toeplitz matrix is also Toeplitz-like, since $\alpha_-(T) \leq 2$ and therefore $\alpha_+(T^{-1}) \leq 2$ by Lemma 5.4.

Let T be an $n \times n$ invertible Toeplitz-like matrix subdivided as

$$T = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix},$$

where T_{11}, T_{12}, T_{21}, and T_{22} are of size $k \times k$, $k \times (n-k)$, $(n-k) \times k$, and $(n-k) \times (n-k)$ respectively.

LEMMA 5.6. *Suppose T has (+)-displacement rank α and suppose T_{11} is nonsingular. Then $\alpha_+(T_{11})$, $\alpha_-(T_{11}^{-1})$, $\alpha_+(T_{22} - T_{21}T_{11}^{-1}T_{12})$, and $\alpha_-[(T_{22} - T_{21}T_{11}^{-1}T_{12})^{-1}]$ are at most α; $\alpha_+(T_{12})$ and $\alpha_+(T_{21})$ are at most $\alpha + 1$; and $\alpha_+(T_{22})$ is at most $\alpha + 2$.*

PROOF. It is easy to check that $T_{11} - ZT_{11}Z^T$ is the $k \times k$ principal submatrix of

$T - ZTZ^T$. So

$$\begin{aligned}\alpha_+(T_{11}) &= \text{rank}(T_{11} - ZT_{11}Z^T) \\ &\leq \text{rank}(T - ZTZ^T) \\ &= \alpha.\end{aligned}$$

By Lemma 5.4, $\alpha_-(T_{11}^{-1}) = \alpha_+(T_{11}) \leq \alpha$.

Because T is invertible, if T_{11} is invertible, then the inverse of T has the following form:

$$T^{-1} = \begin{bmatrix} T_{11}^{-1} + T_{11}^{-1}T_{12}\Delta^{-1}T_{21}T_{11}^{-1} & -T_{11}^{-1}T_{12}\Delta^{-1} \\ -\Delta^{-1}T_{21}T_{11}^{-1} & \Delta^{-1} \end{bmatrix},$$

where $\Delta = T_{22} - T_{21}T_{11}^{-1}T_{12}$. It is also easy to check that $\Delta^{-1} - Z^T\Delta^{-1}Z$ is the $(n-k)\times(n-k)$ bottom right submatrix of $T^{-1} - Z^TT^{-1}Z$. Thus $\alpha_-(\Delta) \leq \alpha_-(T^{-1})$. Again by Lemma 5.4, $\alpha_-[(T_{22} - T_{21}T_{11}^{-1}T_{12})^{-1}] \leq \alpha$ and $\alpha_+(T_{22} - T_{21}T_{11}^{-1}T_{12}) \leq \alpha$.

Compare $T_{12} - ZT_{12}Z^T$, $T_{21} - ZT_{21}Z^T$, and $T_{22} - ZT_{22}Z^T$ with the corresponding submatrices of $T - ZTZ^T$. They are only different by one column, one row, and one column plus one row respectively. Thus $\alpha_+(T_{12})$ and $\alpha_+(T_{21})$ have value at $\alpha + 1$; $\alpha_+(T_{22})$ has value at most $\alpha + 2$. □

The above lemma has a requirement that both T and T_{11} are nonsingular. A further observation shows that if T_{11} is nonsingular, then $\alpha_+(T_{22} - T_{21}T_{11}^{-1}T_{12}) \leq \alpha$ even when T is singular [Kal94].

LEMMA 5.7. *If T and T_{11} are the matrices defined as above, and T_{11} is nonsingular, then $\alpha_+(T_{22} - T_{21}T_{11}^{-1}T_{12}) \leq \alpha$.*

PROOF. Suppose T is represented as $\sum_{j=1}^{\alpha} L^{(j)}U^j$. Let $L_{11}^{(j)}$ be the $k \times k$ principal submatrix of $L^{(j)}$ and let $U_{11}^{(j)}$ be the $k\times k$ principal submatrix of $U^{(j)}$, for all $1 \leq j \leq \alpha$. Thus $T_{11} = \sum_{i=1}^{\alpha} L_{11}^{(j)}U_{11}^{(j)}$. Because T_{11} is nonsingular, there is at least one j such that $U_{11}^{(j)}[1,1] \neq 0$; otherwise, all the terms have the first column all 0, which implies

that T_{11} is singular. Without loss of generality suppose $U_{11}^{(1)}[1,1] \neq 0$, which implies that $U^{(1)}[1,1] \neq 0$.

Introduce the parameterized matrix

$$T(\lambda) = (L^{(1)} + \lambda I)U^{(1)} + \sum_{j=2}^{\alpha} L^{(j)}U^{(j)} = T + \lambda U^{(1)}$$

which is nonsingular, (since $U^{(1)}$ is, and λ is an indeterminate) and of displacement rank α. Partition $T(\lambda)$ according to the partitioning of T so that

$$T(\lambda) = \begin{bmatrix} T_{11} + \lambda U_{11}^{(1)} & T_{12} + \lambda U_{12}^{(1)} \\ T_{21} & T_{22} + \lambda U_{22}^{(1)} \end{bmatrix}.$$

Let

$$\Delta(\lambda) = T_{22} + \lambda U_{22}^{(1)} - T_{21}(T_{11} + \lambda U_{11}^{(1)})^{-1}(T_{12} + \lambda U_{12}^{(1)}).$$

Thus $\alpha_+(\Delta(\lambda)) \leq \alpha$ by Lemma 5.6.

The inverse of a matrix $I + \lambda B$ can be expressed as $I - \lambda B + \lambda^2 B^2 - \lambda^3 B^3 + \cdots$. Use this to expand $(T_{11} + \lambda U_{11}^{(1)})^{-1}$ as

$$\begin{aligned} (T_{11} + \lambda U_{11}^{(1)})^{-1} &= [T_{11}(I + \lambda T_{11}^{-1} U_{11}^{(1)})]^{-1} \\ &= (I + \lambda T_{11}^{-1} U_{11}^{(1)})^{-1} T_{11}^{-1} \\ &= T_{11}^{-1} - \lambda T_{11}^{-1} U_{11} T_{11}^{-1} + \lambda^2 (T_{11}^{-1} U_{11}^{(1)})^2 T_{11}^{-1} - \cdots, \end{aligned}$$

then

$$\Delta(\lambda) = \Delta + \lambda(U_{22}^{(1)} + T_{21}T_{11}^{-1}U_{11}^{(1)}T_{11}^{-1}T_{12} - T_{21}T_{11}^{-1}U_{12}^{(1)}) + \cdots.$$

Thus the displacement rank of Δ cannot be higher than the displacement rank of $\Delta(\lambda)$. Therefore, $\alpha_+(\Delta) \leq \alpha_+(\Delta(\lambda)) \leq \alpha$. □

The displacement rank of the product of two Toeplitz-like matrices is also bounded by a constant [Pan92a] (Proposition A.3).

LEMMA 5.8. *Let A, B be two $n \times n$ matrices with displacement ranks $\alpha_+(A)$ and*

$\alpha_+(B)$. *Then* $\alpha_+(AB) \leq \alpha_+(A) + \alpha_+(B) + 1$ *and a* $(+)$*-displacement generator of length* $\alpha + \beta + 1$ *can be computed in* $O(\log n)$ *time using* $O((\alpha + \beta)n)$ *processors. The dual claim holds for* $(-)$*-displacement rank and* $(-)$*-displacement generator.*

PROOF. First, observe that $I = Z^T Z + e_n e_n^T$, where I is the $n \times n$ identity matrix, Z is the lower shift matrix, and e_n is the nth unit vector. Therefore,

$$\begin{aligned} AB - ZABZ^T &= AB - ZAIBZ^T \\ &= AB - (ZAZ^T)(ZBZ^T) - ZAe_n e_n^T BZ^T \\ &= (A - ZAZ^T)B + ZAZ^T(B - ZBZ^T) - ab^T, \end{aligned}$$

where $a = ZAe_n \in F^n$ and $b = ZB^T e_n \in F^n$. The equation implies that $\text{rank}(AB - ZABZ^T) \leq \text{rank}(A - ZAZ^T) + \text{rank}(B - ZBZ^T) + 1$, since the rank of the sum of several matrices is less than or equal to the sum of the ranks of these matrices. The cost of the computation is obvious. □

5.1.4. Computing Generators of Minimum Length. A generic $n \times n$ matrix has displacement rank n. For $n \times n$ matrices, a displacement generator can usually be obtained efficiently, though the length of the displacement generator may be greater than the displacement rank. A displacement generator of length n of a generic $n \times n$ matrix can easily be computed by setting one of the pair of matrices in the displacement generator to be an identity or a shift matrix. Suppose the displacement generator X of length $\beta \geq \alpha(X)$ for a matrix $X \in F^{n \times n}$ is given, where $\alpha(X)$ is the displacement rank of X. Let $Y = \phi(X) = \hat{G} \cdot \hat{H}^T$, where $\hat{G}, \hat{H} \in F^{n \times \beta}$ is the displacement generator of X. The goal is to determine the displacement rank $\alpha = \alpha(X)$ and to compute a displacement generator of length α for X, $G, H \in F^{n \times \alpha}$, such that $Y = \phi(X) = GH^T$.

LEMMA 5.9. [KS91] *Given a matrix* $A \in F^{n \times n}$ *of rank* r, *let* U^T *and* L *be two unit lower triangular Toeplitz matrices, which have 1's on the diagonal and the other entries uniformly and independently selected from a subset* S *of the field* F. *Then*

the probability that the $i \times i$ principal submatrices of $\hat{A} = UAL$ are all nonsingular for $i = 1, \ldots, r$ is at least $1 - r(r+1)/|S|$.

If S is sufficiently large, then the probability will be high. Let $\hat{Y} = UYL$, where U and L are chosen according to Lemma 5.9. Since U and L are both nonsingular, $\text{rank}(Y) = \text{rank}(\hat{Y})$. All the $i \times i$ principal submatrices of $\hat{Y}$ will be nonsingular for $i = 1, \ldots, \alpha$ with high probability. If these principal submatrices are nonsingular, then every column to the right of the first α columns of $\hat{Y}$ is a linear combination of the first α columns. These linear combinations determine generators for $\hat{Y}$. Let G be the matrix consisting of the first α columns of $\hat{Y}$. Let $y = [y_1|y_2]$ be the first α rows of $\hat{Y}$, where y_1 is the $\alpha \times \alpha$ principal submatrix of $\hat{Y}$ and is nonsingular. Then

$$\hat{Y} = \tilde{G} \cdot \tilde{H}^T,$$

where $\hat{H} \in F^{n \times \alpha}$ and $\tilde{H}^T = [I|y_1^{-1}y_2]$ (so that $y = y_1\tilde{H}^T$). A displacement generator of minimum length for X is then obtained as $Y = (U^{-1}\tilde{G}) \cdot (\tilde{H}^T L^{-1})$. The following parallel algorithm is modified from the sequential algorithm of Kaltofen [Kal94].

ALGORITHM 5.1. *Computing a Generator of Minimum Length*

Input: $\hat{G}, \hat{H} \in F^{n \times \beta}$, where $\beta > \alpha(X)$ and where $Y = \phi(X) = \hat{G}\hat{H}^T$

Output: $G, H \in F^{n \times \alpha}$, where $\alpha = \alpha(X)$ and where $Y = \phi(X) = GH^T$

begin

Generate a unit upper triangular Toeplitz matrix U and a lower triangular Toeplitz matrix L by uniformly and independently choosing the entries from a sufficiently large subset $S \subset F$.

step 1 Compute $\tilde{Y} = UYL$:

$\bar{G} := U \cdot \hat{G}$

$\bar{H} := \hat{H}^T \cdot L$

step 2 Find α, the rank of $\hat{Y}$:

$i := 1$;

Compute entries of $\tilde{Y}_i$, the $i \times i$ principal submatrix of $\tilde{Y}$.

while $\tilde{Y}_i^{-1}$ exists do

$i := 2 \times i$;

Compute entries of $\tilde{Y}_i$, the $i \times i$ principal submatrix of $\tilde{Y}$.

end while

Now $i/2 \leq \alpha < i$. Find α using binary search.

step 3 Compute $\tilde{G}$, the first α columns of $\tilde{Y}$, and $\tilde{H}$, the first α rows of $\tilde{Y}$.

step 4 Compute $\tilde{Y}_\alpha^{-1} y_2$ to produce $\tilde{H}^T$, where y_2 is the right $n - \alpha$ columns of the first α rows of $\tilde{Y}$.

step 5 $G := U^{-1}\tilde{G}$.

$H := (\tilde{H}L^{-1})^T$.

end

LEMMA 5.10. *Given a displacement generator of length β of a matrix $X \in F^{n\times n}$, one can compute a displacement generator of length $\alpha = \alpha(X)$, the displacement rank of X, in $O(\log n + \log\alpha \log\beta + \log^3\alpha)$ time using $O(\alpha\beta n + \beta n + P(\alpha))$ processors. The algorithm is Monte Carlo and requires $2n - 2$ elements uniformly and independently chosen from a set $S \subseteq F$; it returns a correct result with probability at least $1 - \alpha(\alpha+1)/|S|$ (and a wrong answer otherwise).*

PROOF. Step 1 only requires multiplication of an $n \times n$ triangular Toeplitz matrix and n-dimensional vectors. There are only 2β multiplications involved. Thus Step 1 can be performed in $O(\log n)$ time using $O(\beta n)$ processors.

In Step 2, assume $\tilde{Y}_i$ has been computed. The additional entries in $\tilde{Y}_{2i}$ are in three $i \times i$ matrices with each entry equal to a sum of β products, which can be computed in $O(\log\beta)$ time using $O(i^2\beta)$ processors because it takes $O(\log\beta)$ time and $O(\beta)$ processors to compute the inner product of two β-dimensional vectors. If

$\tilde{Y}_{2i}$ is nonsingular, the inverse $\tilde{Y}_{2i}^{-1}$ can be computed in $O(\log^2 i)$ time using $O(P(i))$ processors. Since i never exceeds 2α, the total time is $O(\log^3 \alpha + \log \alpha \log \beta)$ and the total number of processors required is $O(P(\alpha) + \alpha^2\beta)$.

In Step 3 there are αn entries of $\tilde{Y}$ to be computed, and each entry is equal to a sum of β products. Thus this step can be performed in $O(\log \beta)$ using $O(\alpha\beta n)$ processors.

Step 4 is to compute the product of an $\alpha \times \alpha$ matrix and an $\alpha \times (n - \beta)$ matrix. It can be performed in $O(\log \alpha)$ time using $O(\alpha^2 n)$ processors.

Step 5 is similar to Step 1. □

It follows that when α and β are constant, a displacement generator of minimum length can be computed in $O(\log n)$ time using $O(n)$ processors.

5.2. Previous Parallel Algorithms

Fast sequential algorithms are known for solving a linear system with a Toeplitz or Toeplitz-like coefficient matrix that use $O(n \log^2 n)$ steps [BGY80, BA80, Mor80, Kal94]. However, known fast parallel algorithms for exact solutions of a general Toeplitz or Toeplitz-like matrix require $\Omega(\log^2 n)$ time and $\Omega(n^2/\log n)$ processors using a parameterized Newton iteration [Pan90b, Pan92d, BP93]. There are also numerical algorithms that compute the inverse of an $n \times n$ Toeplitz or Toeplitz-like matrix in $O(\log^2 n)$ time using $O(n)$ processors. However, these numerical algorithms require that a good initial approximation to a short displacement generator for the inverse of the input matrix is readily available or that the input matrix is well-conditioned [PR87, Pan92c, Pan92b].

5.2.1. Fast Parallel Algorithms. The algorithm of Bini and Pan [BP93] is similar to Csanky/Leverrier's algorithm for computing the characteristic polynomial of a general dense matrix. Given an $n \times n$ Toeplitz matrix T, the algorithm includes the following steps:

(1) Compute the traces of $T^2, T^3, \ldots, T^{n-1}$;

(2) Compute the coefficients $c_0, c_1, \ldots, c_{n-1}$ of the characteristic polynomial with the traces computed in the first step.

The second step can be performed in $O(\log^2 n)$ time using $O(n/\log n)$ processors by the algorithm for computing the characteristic polynomial given in Section 2.1. The following method shows that the first step can be computed in $O(\log^2 n)$ time using $O(n^2/\log n)$ processors.

Let λ be an indeterminate over the ground field F, define $X_0 = I$, $B = I - \lambda T$, and let $X_i = (2I - X_{i-1}B)X_{i-1}$, for $i \geq 1$. Then

$$X_i = I + \lambda T + \lambda^2 T^2 + \cdots + \lambda^{2^i-1} T^{2^i-1} \equiv (I - \lambda T)^{-1} \bmod \lambda^{2^i}.$$

Let $k = \lceil \log n \rceil$. Then $X_k = I + \lambda T + \lambda^2 T^2 + \cdots + \lambda^{\hat{n}-1} T^{\hat{n}-1}$, where $\hat{n} = 2^k \geq n > \hat{n}/2$. The trace of the matrix X_k is given by $\mathrm{tr}(X_k) = \sum_{i=0}^{\hat{n}-1} \lambda^i \mathrm{tr}(T^i)$. Therefore, the traces of all the ith powers of the matrix T for $1 \leq i \leq \hat{n} - 1$ can be deduced immediately from X_k.

The straightforward algorithm cannot be used to compute $X_i = (2I - X_{i-1}B)X_{i-1}$, because it requires matrix multiplications which cost at least $O(P(n))$ processors. A careful observation shows that step one may only require computation of the product of a Toeplitz matrix or the inverse of a Toeplitz matrix and a vector, because the inverse of a Toeplitz matrix is also uniquely determined by its first column and last column [Tre64, GS72].

LEMMA 5.11. *Let T^{-1} be the inverse of an $n \times n$ Toeplitz matrix T, suppose the top left entry of this matrix is nonzero; and let $u = [u_1, \ldots, u_n]^T$ and $v = [v_1, \ldots, v_n]^T$ be the two vectors representing the first and the last columns, respectively, of T^{-1}. Then*

$$u_1 T^{-1} = L(u)U(v^{(r)}) - L(v^{(s)})U(u^{(t)}), \tag{5.5}$$

where $v^{(r)} = [v_n, v_{n-1}, \ldots, v_1]^T$, $v^{(s)} = [0, v_1, \ldots, v_{n-1}]^T$, and $u^{(t)} = [0, u_n, \ldots, u_2]^T$.

This formula is known to be Gohberg-Semencul formula.

Since λ is an indeterminate and the top left entry of $(1 - \lambda T)^{-1}$ is congruent to one modulo λ, this entry is clearly nonzero. Since $X_i = (1 - \lambda)^{-1} \bmod \lambda^{2^i}$, it is clear that the top left entry of X_i is also nonzero, for $i \geq 0$. Thus, since $X_i = (I - \lambda T)^{-1} \bmod \lambda^{2^i}$ and $(I - \lambda T)$ is also a Toeplitz matrix, the above lemma implies that X_i is uniquely determined by its first column and its last column. Suppose the first column u_{i-1} and last column v_{i-1} of X_{i-1} are given. Then, the first column u_i of X_i can be computed by

$$u_i = (2I - X_{i-1}B)u_{i-1};$$

that is,

$$u_i = 2u_{i-1} - \frac{1}{u_{i-1}^{(1)}}[L(u_{i-1})U(v_{i-1}^{(r)}) - L(v_{i-1}^{(s)})U(u_{i-1}^{(t)})]Bu_{i-1}, \tag{5.6}$$

where $u_{i-1}^{(1)}$ is the first entry of the vector u_{i-1}. The last column of v_i of X_i can be computed by a similar method.

Notice that B is also a Toeplitz matrix. Thus the column u_i can be obtained from u_{i-1} and v_{i-1} by applying several matrix-vector products, where each matrix is a Toeplitz matrix and such a computation can be reduced to polynomial multiplication. Since the elements involved are polynomials in λ, such a matrix-vector product is further reduced to polynomial multiplication of two bivariate polynomials as follows:

(1) Compute Bu_{i-1}, where B is a Toeplitz matrix with polynomial entries in λ with degrees at most 1, and u_{i-1} is a vector with polynomial entries in λ with degrees no more than $2^{i-1}-1$. The computation can be reduced to the multiplication of two bivariate polynomials with degrees $2n-1$ and $n-1$ respectively in the first variable, and 1 and $2^{i-1}-1$ respectively in the second, so that it can be performed in $O(i+\log n)$ time using $O(n2^i)$ operations by 2-dimensional FFT or Kronecker substitution [BP94].

(2) Compute $L(u_{i-1})U(v_{i-1}^{(r)})(Bu_{i-1})$ and $L(v_{i-1}^{(s)})U(u_{i-1}^{(t)})(Bu_{i-1})$. Since the vector Bu_{i-1} has been computed, again the computations are only the multiplication of Toeplitz matrices by vectors. The Toeplitz matrices have polynomial entries in λ with degree at most $2^{i-1}-1$ and the vector Bu_{i-1} has polynomial entries with degree at most 2^{i-1}. Thus the computation can also be reduced to the multiplication of two bivariate polynomials with degrees $n-1$ and $2^{i-1}-1$ respectively in the first variable, and $n-1$ and 2^{i-1} (or 2^i-1) respectively in the second. Thus this step takes $O(\log n)$ time using $O(n2^i)$ operations.

(3) The reciprocal of $u_{i-1}^{(1)}$ modulo λ^{2^i} can also be computed efficiently [BM75]. In each iteration step of computing X_i, one actually only needs to compute one more stage of a Newton iteration for the reciprocal, which is equivalent

to the multiplication of two polynomials with degrees less than 2^i. Thus the computation only costs $O(i)$ time using $O(i2^i)$ operations.

The above shows that for $1 \leq i \leq k$, the ith stage of the algorithm can be performed in $O(\log n)$ time using $O(n2^i)$ operations. Thus the first columns of $X_0, X_1, \ldots, X_k$ can be computed in $O(\log^2 n)$ time using $O(n^2 \log n)$ operations. Clearly a similar procedure can be used to compute v_i.

The whole matrix X_k can be recovered from its first column and its last column by the Gohberg-Semencul formula, but the cost will be as high as $O(n^3 \log^2 n)$ because all the entries of X_k are polynomials of degree $n-1$ in λ. However, notice that it is only necessary to compute the trace of X_n. Thus after the first column and the last column of X_k have been computed, the trace of X_k can be recovered using the following formula (with the first column $u_k = (x_1, x_2, \ldots, x_n)^T$ of X_k and the last column $v_k = (y_1, y_2, \ldots, y_n)^T$ of X_k.)

$$\operatorname{tr}(X_k) = \operatorname{tr}(I - \lambda T)^{-1} \bmod \lambda^{2^k} = x_0^{-1} \sum_{j=1}^{n} \left(\sum_{i=1}^{j} x_i y_i - \sum_{i=1}^{j-1} x_{n-i} y_{n-i} \right) \bmod \lambda^{2^k}.$$

This computation is only vector convolution (equivalent to polynomial multiplication) with entries being polynomials of degree less than n. The reciprocal of x_0 has already been computed in the kth iteration step. Thus this step can be performed in $O(\log n)$ time using $O(n^2 \log n)$ operations. Since computation of the traces of $T, T^2, T^3, \ldots, T^{n-1}$ is the most expensive step of the above algorithm, the following lemma has been proved.

LEMMA 5.12. *The characteristic polynomial of an $n \times n$ Toeplitz matrix T over a field of characteristic zero or greater than n can be computed in $O(\log^2 n)$ time, using $O(n^2/\log n)$ processors.*

The algorithm works over any field of characteristic zero or greater than n. The restriction comes from the method for computing the characteristic polynomial from the traces.

The above method can clearly also be used to compute the determinant. The algorithm uses $O(\log^2 n)$ time and $O(n^2/\log n)$ processors.

Computing a Krylov Sequence. Given an $n \times n$ Toeplitz matrix T and a vector b, it is possible to compute the Krylov sequence $b, Tb, T^2b, \ldots, T^{n-1}b$ in $O(\log^2 n)$ time using $O(n^2/\log n)$ processors by a modified version of the above algorithm.

Recall the above algorithm. At the kth stage of Newton iterations, the first column u_k and the last column v_k of X_k have been computed. Since $X_k = I + \lambda T + \lambda^2 T^2 + \cdots + \lambda^{n-1}T^{n-1}$, $X_k b = (I + \lambda T + \lambda^2 T^2 + \cdots + \lambda^{n-1}T^{n-1})b$, so $X_k b = b + (Tb)\lambda + (T^2b)\lambda^2 + \cdots + (T^{n-1}b)\lambda^{n-1}$. Let $b(\lambda) = X_k b$. Then $b(\lambda)$ can be computed by the following formula, again with the first column of X_k equal to $u_k = (x_1, x_2, \ldots, x_n)^T$ and the last column of X_k equal to $v_k = (y_1, y_2, \ldots, y_n)^T$.

$$b(\lambda) = x_0^{-1}[L(u_k)U(v_k^{(r)}) - L(v_k^{(s)})U(u_k^{(t)})]Bb, \tag{5.7}$$

where x_0 is the first entry of u_k.

Compute $b(\lambda)$ using Equation 5.7 and performing multiplications from the right to the left, so that one repeatedly computes the product of Toeplitz matrices and vectors with polynomial entries of degree at most n. As argued above, this can be performed in $O(\log n)$ time using $O(n^2/\log n)$ processors.

The time and processor bounds are the same as those for computing the characteristic polynomial. The vector $b(\lambda)$ has entries that are polynomials in λ. The coefficients in λ^k of the entries form the vector $T^k b$ in the Krylov sequence.

Computing the Solution of a Linear System. Suppose T is nonsingular. Then the system $Tx = b$ has a unique solution. The coefficients of the characteristic polynomial and the the Krylov sequence can be computed in $O(\log^2 n)$ time using $O(n^2/\log n)$ processors. By the Cayley-Hamilton theorem, the solution of the system for the above system is

$$x = -\frac{1}{c_0}\sum_{i=0}^{n-1} c_{i+1}T^i b, \tag{5.8}$$

if the characteristic polynomial of T is $c_0+c_1x+\cdots+c_nx^n$. Thus computing x from the characteristic polynomial and Krylov sequence only requires multiplication of vectors by scalars, which can be performed in constant time using $O(n^2)$ processors, and vector additions (or subtractions) of n vectors, which can be performed in $O(\log n)$ time using $O(n^2/\log n)$ processors.

Thus if T is an $n \times n$ nonsingular Toeplitz matrix (again, over a field whose characteristic is zero or greater than n), the system of linear equations $Tx = b$ can be solved in $O(\log^2 n)$ time using $O(n^2/\log n)$ processors.

Toeplitz-like Matrix Computations. The above fast parallel algorithms for Toeplitz matrix computations can be extended to Toeplitz-like matrix computations [BP93].

5.2.2. Numerical Algorithms. Suppose that a nonsingular $n \times n$ matrix A is given with its displacement generator of minimum length (equal to the displacement rank) and the displacement rank of A is bounded by a small constant. It is possible to compute a displacement generator of A^{-1} in $O(\log^2 n)$ time using $O(n)$ processors when a good initial approximation with small displacement rank of A^{-1} is readily available [Pan92c, Pan92b].

Recall the numerical algorithms based on Newton iteration or its extensions for general dense matrix computations in Section 2.3. Given an $n \times n$ nonsingular matrix A, and a sufficiently close initial approximation B to A^{-1}, let $B^{(0)} = B$ and let $B^{(k)} = B^{(k-1)}(2I - AB^{(k-1)})$ for $k \geq 1$. Then $I - B^{(k)}A = (I - B^{(k-1)}A)^2$ for $k \geq 1$, so $\left\|I - B^{(k)}A\right\| \leq \left\|I - B_{(k-1)}A\right\|^2$. If the initial approximation B to A^{-1} is given such that $\|I - BA\| \leq \epsilon < 1 - 1/n^{O(1)}$, then $B^{(k)}$ is quadratically convergent to A^{-1} such that $\left\|I - B^{(k)}A\right\| \leq \epsilon^{2^k}$. If $\delta > 0$ and $\log\log(1/\delta) \in O(\log n)$, then $k \in O(\log n)$ stages of iterations will ensure $\left\|I - B^{(k)}A\right\| \leq \delta$. Each iteration requires $O(\log n)$ time and $O(P(n))$ processors, which is the cost for matrix multiplication and matrix addition (or subtraction).

The above iteration method can be modified to achieve solutions for Toeplitz and Toeplitz-like matrix computations. Assume that the (+)-displacement generator of a Toeplitz-like matrix A of length r is given, and that an initial approximation B to A^{-1} is also given, with its (−)-displacement generator of length at most r, such that $\|I - BA\| \leq \epsilon < \frac{1}{2}$. Apply exact Newton iteration to compute an approximation to A^{-1} of accuracy within δ. Suppose $B^{(0)} = B$ is given with its (−)-displacement generator of length r, where r is a constant. Clearly $B^{(1)} = B^{(0)}(2I - AB^{(0)})$ can be computed in $O(\log n)$ time using $O(n)$ processors. However, $B^{(1)}$ may have (−)-displacement rank as high as $3r+5$ by Lemma 5.8. If the Newton iteration method for general dense matrices is directly applied, then after $O(\log\log n)$ stages of iteration, to compute $B^{(k)}$ for $k \in O(\log\log n)$, then the new generated approximation of A^{-1} may have (−)-displacement rank $\Omega(\log n)$, which will affect the cost of each stage of iterations dramatically (it will require $\Omega(\log n)$ matrix multiplications and matrix additions (or subtractions)). Thus it is necessary to keep the displacement rank of these approximations small, in particular, to be at most r. Therefore, after each stage of Newton iteration computing $B^{(k)} = B^{(k-1)}(2I - AB^{(k-1)})$, a subroutine should be called to replace the current approximation of A^{-1} with one whose (−)-displacement rank is at most r.

Define τ_- to be a subroutine that takes a matrix B and a constant r as inputs, and generates a matrix B_r as an output that has (−)-displacement rank at most r such that $\|B - B_r\|_2$ is small. Abusing notation, write $\tau_-(B, r)$ to represent B_r, the output of the subroutine τ_- with (−)-displacement rank bounded by r. Similarly τ_+ is defined to be a subroutine to reduce the (+)-displacement rank. Thus the desired modified Newton iteration is the following.

$$\hat{B}^{(k)} = B^{(k-1)}(2I - AB^{(k-1)}), \qquad B^{(k)} = \tau_-(\hat{B}^{(k)}, r). \tag{5.9}$$

A suitable subroutine (which, however, is assumed when analyzed to use exact real arithmetic) is described by Pan [Pan90b, Pan92b]. This subroutine generates an

output by solving the Singular Value Decomposition (SVD) problem defined in Section 1.3 for auxiliary matrices of rank $O(1)$, which is essentially reduced to computing the zeros of polynomials of degrees $O(1)$ have only real zeros. The algorithm only uses $O(\log n)$ time and $O(n)$ processors. Pan also shows that if B is an approximation of X such that $\|X - B\|_2 = \Delta$, X has displacement rank r, and B has displacement rank R, then $\|B - B_r\|_2 \leq (1 + 2n(R - r))\Delta$. Thus each stage of extended Newton iterations uses $O(\log n)$ time and $O(n)$ processors (see Equation 5.9).

Assume that an $n \times n$ nonsingular matrix A has been given with its $(+)$-displacement generator of length at most $r = O(1)$ and that an initial approximation B to A^{-1} is also available with its $(-)$-generator of length at most r. Suppose one recursively computes $\hat{B}^{(1)}, B^{(1)}, \hat{B}^{(2)}, B^{(2)}, \ldots$ as given in Equation 5.9. Pan shows that if $\|I - AB\|_2 = \epsilon$ is small — in particular, if $(1+2n(2r+3))\text{cond}_2(A)\epsilon^{1-v} < 1$, for some fixed positive $v < 1$, then

$$\left\|B^{(k)} - A^{-1}\right\|_2 \leq \epsilon^{(1+v)^k} \left\|A^{-1}\right\|_2.$$

The matrices $\hat{B}^{(1)}, B^{(1)}, \hat{B}^{(2)}, \ldots, \hat{B}^{(k)}, B^{(k)}$ can be computed in $O(k \log n)$ time using $O(n)$ processors. Since $\epsilon < \frac{1}{2}$ and $\|A^{-1}\|_2 \leq 2^{n^{O(1)}}$, if $\log\log(1/\delta) \in O(\log n)$, then $\left\|B^{(k)} - A^{-1}\right\| \leq \delta$ for some $k \in O(\log n)$, and the algorithm can be used to compute such an approximation $B^{(k)}$ of A^{-1} in $O(\log^2 n)$ time using $O(n)$ processors.

Approximation by a Matrix of Bounded Displacement Rank. The algorithm τ_- solves the problem as follows. Let X be an unknown $n \times n$ matrix such that $\alpha_-(X) = r$, for a fixed r. In the above case, $X = A^{-1}$ and $\alpha_+(A) = r$. Let B be an approximation to X (in the above case $B = \hat{B}^{(i+1)}$) with $\alpha_-(B) \leq R$ and $R \geq r$ (in the above case $R = 3r+5$). Furthermore, the matrix B is given with its $(-)$-displacement generator $M, N \in \mathbb{R}^{n \times R}$ such that $W = \phi_-(B) = MN^T$. Let $\delta = \|X - B\|_2$. The algorithm is to compute $G, H \in \mathbb{R}^{n \times r}$, a $(-)$-displacement generator of length r of a matrix B_r (in the above case, $B_r = B^{(i+1)}$), such that $\phi_-(B_r) = GH^T$ and $\|X - B_r\|_2$ is small.

This problem is reduced to computing the SVD of A, by the following lemma ([GVL90], p.73):

LEMMA 5.13. *Let $A = U\Sigma V^T$ be the SVD of a matrix $A \in \mathbb{R}^{m\times n}$, where Σ is an $m \times n$ diagonal matrix with exactly r nonzeros on the diagonal, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$. Let $s < r$ be a positive integer, Σ_δ be the $m \times n$ diagonal matrix, $\Sigma_s = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_s, 0, \ldots, 0)$, $A_\delta = U\Sigma_s V^T$. Then $\|A - A_s\|_2 = \sigma_{s+1} \leq \|A - Y\|_2$ for all matrices Y of rank at most s.*

Pan shows that an algorithm to compute the desired matrix B_r [Pan90b, Pan92b].

(1) Compute the SVD of the matrix $W = \phi_-(B) = MN^T$. Let U_r and V_r denote the pair of $k \times r$ matrices formed by the first r columns of the matrices U and V, respectively, of the SVD and let $\Sigma_r = \mathrm{diag}(\sigma_1(W), \ldots, \sigma_r(W))$.

(2) Compute and output the desired $(-)$-generator $G = U_r\Sigma_r$, $H = V_r$ of length r for the matrix B_r.

The complexity of this algorithm is dominated by the complexity of the first step, of computing the SVD of $W = MN^T$, which can be performed in $O(\log n)$ time using $O(n/\log n)$ processors. Unfortunately, there is no rational algorithm for SVD computation, since this computation produces outputs that are not rational functions of the inputs. Pan assumes that exact real arithmetic is used in the analysis of his algorithm. Of course, we are interested here in finite precision computations. An error (and complexity) analysis for an implementation of this algorithm using finite precision arithmetic remains to be performed.

Pan shows that $\|X - B_r\|_2 \leq (1 + 2n(R - r))\,\|X - B\|_2$ [Pan90b, Pan92b]. The above bound can be improved.

LEMMA 5.14. $\|X - B_r\|_2 \leq (1 + 2n)\,\|X - B\|_2$.

PROOF. Since $\|\phi_-(X) - \phi_-(B)\|_2 \leq 2\|X - B\|_2$, and B_r is computed from the above algorithm using SVD, Lemma 5.13 shows that

$$\|\phi_-(B) - \phi_-(B_r)\|_2 \leq \|\phi_-(B) - \phi_-(X)\|_2 \leq 2\|X - B\|_2 .$$

It can be shown that $A = \sum_{i=0}^{n-1} (Z^T)^i \phi_-(A) Z^i$ ([Woo93] Lemma 2). Thus

$$\|B - B_r\|_2 \leq n\|\phi_-(B) - \phi_-(B_r)\|_2 ,$$

so $\|B - B_r\|_2 \leq 2n\|X - B\|_2$. Therefore,

$$\|X - B_r\|_2 \leq \|X - B\|_2 + \|B - B_r\| \leq (1 + 2n)\|X - B\|_2 .$$

□

5.3. Efficient Toeplitz and Toeplitz-like Matrix Computations

Reif has given a processor efficient parallel algorithm for Toeplitz and Toeplitz-like matrix computations using a method similar to the approach for general dense matrix computations [Rei95].

The structure of the algorithm for Toeplitz and Toeplitz-like matrix computations is identical to that of the algorithm for general dense matrix computations (given in Section 4.2). The differences are as follows.

- The input matrix A is required to have "constant" displacement rank r, and is given by a displacement generator of length r—that is, a pair of $n \times r$ matrices G, H such that $\phi_+(A) = GH^T$.
- The input matrix in each Newton iteration should have small displacement rank and be given by a displacement generator. In particular, the matrix should have displacement rank at most r and be given by its displacement generator of length at most r.
- The output matrix computed by each Newton iteration has small (but increased) displacement rank (at most $R \leq 3r + 5$) and is represented by a displacement generator of length R.

Unfortunately, Newton iteration and Newton Hensel Lifting produce displacement ranks much higher than r (likely $R = 3r + 5$). In order to keep the displacement ranks and the length of the displacement generators small, one additional change is required. Each Newton iteration and Hensel Lifting stage is followed by an additional approximation stage, in which the current approximation is replaced by a slightly less accurate estimate whose displacement rank is at most r.

For most of the algorithm it is sufficient to replace general dense matrix multiplication and matrix vector multiplication by Toeplitz-like matrix multiplication and Toeplitz-like matrix-vector multiplication in order to reduce the number of processors to $O(n)$. It is not so easy to recover an exact RF tree of $\bar{A}$ from an approximate RF

tree efficiently—more sophisticated techniques are required for this step.

Changes to Newton iteration are discussed in Section 5.3.1. An algorithm for recovering of an exact RF tree of $\bar{A}$ is given in Section 5.3.2. Hensel Lifting is discussed briefly in Section 5.3.3. Additional work that should be done to complete an analysis (and, perhaps, description) of this algorithm is summarized in Section 5.3.4.

5.3.1. Approximate Newton Iteration for Toeplitz-like Matrices. Recall that approximate Newton iteration replaces an approximation $B^{(k-1)}$ for the inverse of a matrix A by $B^{(k)} = B^{(k-1)}(2I - A^{(k)}B^{(k-1)})$, where $A^{(k)}$ is an approximation of A. If $A^{(k)}$ has $(+)$-displacement rank r (and hence $A^{(k)}$ has $(-)$-displacement rank $r+2$ by Lemma 5.5) and $B^{(k-1)}$ has $(-)$-displacement rank r, then, by Lemma 5.8, $A^{(k)}B^{(k-1)}$ has $(-)$-displacement rank at most $2r+3$, $2I - A^{(k)}B^{(k-1)}$ has $(-)$-displacement rank at most $2r+4$, and $B^{(k+1)}$ has $(-)$-displacement rank at most $R = 3r+5$.

In order to keep the displacement rank of all the matrices returned as output small, one has to find a matrix $\check{B}^{(k+1)}$ with $(-)$-displacement rank at most r such that $\left\|\check{B}^{(k+1)} - A^{-1}\right\|$ is not substantially larger than $\left\|B^{(k+1)} - A\right\|$. Then $\tilde{B}^{(k-1)}$ can be used to continue the iteration instead of $B^{(k+1)}$. One method to solve this problem has been used in the previous numerical algorithms for Toeplitz and Toeplitz-like matrix computations [Pan90b, Pan92b] (see also Section 5.2.2). However, Reif shows a different way to solve this problem.

In the approximate Newton iteration, let $\hat{B}^{(k)} = \tilde{B}(2I - \tilde{A}^{(k)}\tilde{B}^{(k-1)})$. Reif's algorithm computes $\tilde{B}^{(k)}$ from $\hat{B}^{(k)}$ by the following steps.

- Find the maximum number $r^{(k)} \leq r$ such that $\phi_{-}(\hat{B}^{(k)})$ has a nonsingular $r^{(k)} \times r^{(k)}$ principal submatrix $M^{(k)}$.
- Compute the $(-)$-displacement generator of length $r^{(k)}$ of $\tilde{B}^{(k)}$ using the first $r^{(k)}$ columns and the first $r^{(k)}$ rows of $\phi_{-}(\hat{B}^{(k)})$.

Reifs shows the following lemma [Rei95] (Proposition 7.4).

LEMMA 5.15. *Suppose* $\epsilon = \left\| I - \hat{B}^{(k)}\tilde{A}^{(k)} \right\|_\infty$; *then* $\left\| I - \tilde{B}^{(k)}\tilde{A}^{(k)} \right\|_\infty \leq s\epsilon$, *where* $s = (n\|A\|_\infty)^{r+1}$.

Reif appears to base a proof of this on the assumption that "if a matrix A has displacement rank r then the first r rows and columns of $\phi_+(A)$ are linearly independent". However, this assumption is not generally true (as can be seen by considering $A = \begin{bmatrix} I & 0 \\ 0 & \tilde{A} \end{bmatrix}$, where $\tilde{A}$ is an $n/2 \times n/2$ matrix with displacement rank $r-2$).

It has already been shown that the problem can also be solved by a numerical method—using an SVD computation as described in Section 5.2.2. That method is numerically stable and has a very small numerical error (see [Wil65, GVL90]). Hereafter, suppose that the subroutine τ_- (and τ_+) can be found which computes a matrix B_r efficiently from B using finite precision arithmetic such that $\|B_r - X\| \leq 2nc\|B - X\|$, for some constant $c > 1$. Also assume all the norms $\|\ \|$ are $\|\ \|_2$.

Suppose the matrix $\bar{A}$ of (+)-displacement rank r satisfies Equation 3.2, the sequence $\tilde{A}^{(0)}, \tilde{A}^{(1)}, \ldots, \tilde{A}^{(k)}$ satisfies Equation 3.3 and Equation 3.4, and all $\tilde{A}^{(i)}$ have (+)-displacement rank at most r for $i \geq 0$. Let $\tilde{B}^{(0)} = D^{-1}$, $\hat{B}^{(k)} = \tilde{B}^{(k-1)}(2I - \tilde{A}^{(k)}\tilde{B}^{(k-1)})$, $\check{B}^{(k)} = \hat{B}^{(k)}(2I - \tilde{A}^{(k)}\hat{B}^{(k)})$, and $\tilde{B}^{(k)} = \tau_-(\check{B}^{(k)}, r)$, for $k \geq 1$.

LEMMA 5.16. *If* $\epsilon^2 \leq 1/(5cn)$, *then* $\left\| I - \tilde{B}^{(k)}\tilde{A}^{(k+1)} \right\| \leq \frac{1}{2}(2\epsilon)^{2^k}$, *and* $\left\| I - \tilde{B}^{(k)}\tilde{A}^{(k)} \right\| \leq \left\| I - \tilde{B}^{(k-1)}\tilde{A}^{(k)} \right\|^2$.

PROOF. The inequality $\left\| I - \tilde{B}^{(0)}\tilde{A}^{(1)} \right\| \leq \epsilon$ has been shown in the proof of Lemma 3.4. Suppose the claim holds for $k-1$, so $\left\| I - \tilde{B}^{(k-1)}\tilde{A}^{(k)} \right\| \leq \frac{1}{2}(2\epsilon)^{2^{k-1}}$. Then

$$\begin{aligned} \left\| I - \hat{B}\tilde{A}^{(k)} \right\| &= \left\| I - \tilde{B}^{(k-1)}(2I - \tilde{A}^{(k)}\tilde{B}^{(k-1)})\tilde{A}^{(k)} \right\| \\ &\leq \left\| I - \tilde{B}^{k-1}\tilde{A}^{(k)} \right\|^2 \\ &\leq \left(\frac{1}{2}(2\epsilon)^{2^{k-1}} \right)^2 \leq \left(\frac{1}{2}(2\epsilon) \right)^2 = \epsilon^2. \end{aligned}$$

Also

$$\begin{aligned}\left\|I - \check{B}^{(k)}\tilde{A}^{(k)}\right\| &= \left\|I - \hat{B}^{(k)}(2I - \tilde{A}^{(k)}\hat{B}^{(k)})\tilde{A}^{(k)}\right\| \\ &\leq \left\|I - \hat{B}\tilde{A}^{(k)}\right\|^2\end{aligned}$$

Since $\hat{B}^{(k)} = \tau_-(\check{B}^{(k)}, r)$ and $\tilde{A}^{(k)}$ has $(+)$-displacement rank at most r,

$$\left\|\hat{B}^{(k)} - \check{B}^{(k)}\right\| \leq 2cn\left\|(\tilde{A}^{(k)})^{-1} - \check{B}^{(k)}\right\|.$$

Since $\tilde{A}^{(k)}$ satisfies Equation 3.3, it is easy to verify that

$$\mathrm{cond}(\tilde{A}^{(k)}) = \left\|\tilde{A}^{(k)}\right\|\left\|(\tilde{A}^{(k)})^{-1}\right\| \leq 2.$$

Thus

$$\begin{aligned}\left\|I - \hat{B}^{(k)}\tilde{A}^{(k)}\right\| &\leq \left\|I - \check{B}^{(k)}\tilde{A}^{(k)}\right\| + \left\|\check{B}^{(k)}\tilde{A}^{(k)} - \tilde{B}^{(k)}\tilde{A}^{(k)}\right\| \\ &\leq \left\|I - \check{B}^{(k)}\tilde{A}^{(k)}\right\| + \left\|\tilde{A}^{(k)}\right\|\left\|\hat{B}^{(k)} - \tilde{B}^{(k)}\right\| \\ &\leq \left\|I - \check{B}^{(k)}\tilde{A}^{(k)}\right\| + \left\|\tilde{A}^{(k)}\right\|\left(2cn\left\|(\tilde{A}^{(k)})^{-1} - \check{B}^{(k)}\right\|\right) \\ &\leq \left\|I - \check{B}^{(k)}\tilde{A}^{(k)}\right\| + 2cn\left\|\tilde{A}^{(k)}\right\|\left\|(\tilde{A}^{(k)})^{-1}\right\|\left\|I - \check{B}^{(k)}\tilde{A}^{(k)}\right\| \\ &\leq (1 + 2cn \cdot \mathrm{cond}(\tilde{A}^{(k)}))\left\|I - \check{B}^{(k)}\tilde{A}^{(k)}\right\| \\ &\leq (1 + 4cn)\left\|I - \hat{B}\tilde{A}^{(k)}\right\|^2 \\ &\leq (5cn\left\|I - \hat{B}\tilde{A}^{(k)}\right\|)\left\|I - \hat{B}\tilde{A}^{(k)}\right\| \\ &\leq (5cn\epsilon^2)\left\|I - \tilde{B}^{(k-1)}\tilde{A}^{(k)}\right\|^2 \leq \left\|I - \tilde{B}^{(k-1)}\tilde{A}^{(k)}\right\|^2.\end{aligned}$$

The inequality $\left\|I - \tilde{B}^{(k)}\tilde{A}^{(k+1)}\right\| \leq \frac{1}{2}(2\epsilon)^{2^k}$ can now be proved by an argument similar to the one used to prove Lemma 3.4. Thus the lemma follows by induction. □

Suppose again that $\bar{A}$ satisfies Equation 3.2, the sequence $\tilde{A}^{(0)}, \tilde{A}^{(1)}, \ldots, \tilde{A}^{(k)}$ satisfies Equation 3.3 and Equation 3.4, $\tilde{B}^{(0)} = D_1^{-1}$, and $\tilde{B}^{(k)}$ is computed as above, for $k \geq 1$. Let $\tilde{\Delta}^{(0)} = D_2$, $\hat{\Delta}^{(k)} = \tilde{A}_{22}^{(k)} - \tilde{A}_{21}^{(k)}\tilde{B}^{(k)}\tilde{A}_{12}^{(k)}$, and $\tilde{\Delta}^{(k)} = \tau_+(\hat{\Delta}^{(k+1)}, r)$, for $k \geq 1$.

LEMMA 5.17. *If* $\epsilon^2 \leq 1/(2+4cn)$, *then*

$$\left\|\tilde{\Delta}^{(k)} - \Delta\right\| \leq \frac{d}{2}(2\epsilon)^{2^k},$$

where $\Delta = \bar{A}_{22} - \bar{A}_{21}\bar{A}_{11}^{-1}\bar{A}_{12}$, *and*

$$\left\|\tilde{\Delta}^{(k)} - \hat{\Delta}^{(k+1)}\right\| \leq \frac{d}{2}(2\epsilon)^{2^k},$$

for $k \geq 0$.

PROOF. Lemma 3.8 shows that $\left\|\hat{\Delta}^{(k)} - \Delta\right\| \leq \frac{d}{2}(2\epsilon)^{2^k}$. Since Δ has displacement rank r and $\tilde{\Delta}^{(k)} = \tau_+(\hat{\Delta}^{(k+1)}, r)$,

$$\begin{aligned}
\left\|\tilde{\Delta}^{(k)} - \Delta\right\| &\leq (1+2cn)\left\|\hat{\Delta}^{(k+1)} - \Delta\right\| \\
&\leq (1+2cn)\frac{d}{2}(2\epsilon)^{2^{k+1}} \\
&\leq \left(2(1+2cn)\epsilon^2\right)\frac{d}{2}(2\epsilon)^{2^k} \\
&\leq \frac{d}{2}(2\epsilon)^{2^k}.
\end{aligned}$$

A similar method can be used to prove the bound on $\left\|\tilde{\Delta}^{(k)} - \tilde{\Delta}^{(k+1)}\right\|$. □

Thus the analysis in Section 4.2 can be applied, assuming that the subroutines τ_+ and τ_- have the properties described above.

5.3.2. Computing the Exact RF Tree of $\bar{A}$. Suppose a good approximate augmented RF tree of $\bar{A}$ has been computed, as defined in Section 4.3. All the matrices $\tilde{A}_\alpha$ and their inverses in the approximate augmented RF tree are given by their $(\pm)$-displacement generators. The goal is to compute the $(\pm)$-generators for the exact matrices from the generators for their approximations.

The algorithm for general dense matrices requires $O(n^2/\log n)$ processors to recover the exact entries of $\bar{A}_\alpha$. It only uses $O(\log n)$ time and $O(n/\log n)$ processors to compute $\bar{m}_\alpha$ and $\det(\bar{A}_\alpha)$ exactly, but it uses $\Omega(n^2/\log n)$ processors to recover the exact entries of $\bar{A}$. The following shows how to recover $(+)$-displacement generators of all the (exact) $\bar{A}_\alpha$ in $O(\log n)$ time using $O(n)$ processors.

Let $\tilde{A}$ be an approximation of A such that $\left\|\tilde{A}-A\right\| \leq \epsilon$, for a small ϵ, where A is an unknown integer matrix and has (+)-displacement rank r, and $\tilde{G}, \tilde{H}$, a (+)-displacement generator of length r of $\tilde{A}$, is given.

Let U^T and L be two unit lower triangular Toeplitz matrices, which have 1's on the diagonal with the other entries uniformly and independently selected from the set of integers $\{1,2,\ldots,n\}$, where n is the order of the matrices A and $\tilde{A}$. Lemma 5.9 shows that the probability that the $i \times i$ principal submatrices of $\tilde{B} = U\phi_+(\tilde{A})L$ are all nonsingular for $i = 1,\ldots,r$ is at least $1-r(r+1)/n$. Since the entries of U and L are chosen from $\{1,2,\ldots,n\}$, $\|U\|, \|L\| \leq n^2$. Let $\tilde{M}$ and M be the $r \times r$ principal submatrices of $\tilde{B}$ and of $B = U\phi_+(A)L$ respectively. Since $\left\|\phi_+(A) - \phi_+(\tilde{A}\right\| \leq 2\left\|A-\tilde{A}\right\| \leq 2\epsilon$, $\left\|B-\tilde{B}\right\| \leq \|U\|\left\|A-\tilde{A}\right\|\|L\| \leq 2n^4\epsilon$. Thus $\left\|M-\tilde{M}\right\| \leq \left\|B-\tilde{B}\right\| \leq 2n^4\epsilon$. When ϵ is small enough (say $\epsilon < 1/(4n^4)$), rounding each of the entries of $\tilde{M}$ to the nearest integer will yield the exact matrix M. Also the first r columns and the first r rows of B can be computed exactly from the first r columns and the first r rows of $\tilde{B}$ by rounding each entry to the nearest integer. If M is nonsingular, let $\hat{G}$ be the first r columns of B and $\hat{H}^H = [I|y_1^{-1}y_2]$, where $[y_1|y_2]$ is the matrix consisting of the first rows of B. Then B can be exactly represented by $\hat{G}\hat{H}^T$. Let $G = U^{-1}\hat{G}$ and $H^T = \hat{H}^T L^{-1}$; then $\phi_+(A) = GH^T$.

ALGORITHM 5.2. *Rounding-off for Toeplitz-like matrices.*

Input: $\phi_+(\tilde{A}) = \tilde{G}\tilde{H}^T$, where $\tilde{G}$ and $\tilde{H}$ are two $n \times r$ matrices, for a matrix $\tilde{A}$ that is an approximation to an integer matrix A such that $\alpha_+(A) = r$ and $\left\|A-\tilde{A}\right\| \leq \epsilon$, where $\epsilon \leq 1/(4n^4)$.)

Output: $\phi_+(A) = GH^T$

begin Generate two unit lower triangular Toeplitz matrices U^T and L which have 1's on the diagonal and with the other entries uniformly and independently selected from the set of integers $\{1,2,\ldots,n\}$.

step 1 Compute $U\tilde{G}$ and $\tilde{H}^T L$.

step 2 Compute $\tilde{M}$, the $r \times r$ principal submatrix of $\tilde{B} = U\phi_+(\tilde{A})L = (U\hat{G})(\hat{H}^T L)$. Set M to be the integer matrix obtained by rounding each entry of $\tilde{M}$ to the nearest integer.

step 3 If M is singular then report failure and stop; otherwise, compute M^{-1}.

step 4 Compute the first r rows and the first r columns of $\tilde{B}$ from $U\tilde{G}$ and $\hat{H}^T L$. Round each entry in these rows and columns to the nearest integer to yield $\hat{G}$, the first r columns, and $\tilde{y} = [M|y_2]$, the first r rows, of $B = U\phi_+(A)L$.

step 5 Compute $\hat{H}^T = [I|M^{-1}y_2]$ so that $B = \hat{G}\hat{H}^T$.

step 6 Compute $G = U^{-1}\hat{G}$ and $H^T = \hat{H}^T L^{-1}$.

end

LEMMA 5.18. *Given a displacement generator of length $r \in O(1)$ for $\tilde{A}$, an approximation of an integer matrix A, such that $\left\|\tilde{A} - A\right\| \leq \epsilon < 1/4n^2$, the above algorithm computes a displacement generator $G, H \in \mathbb{Q}^{n\times r}$ of A such that $\phi_+(A) = GH^T$, in $O(\log n)$ time using $O(n)$ processors. The algorithm returns a correct result with probability at least $1 - r(r+1)/n$.*

Using the above algorithm as a subroutine, a $(+)$-displacement generator of $\bar{A}_\alpha$ and a $(-)$-displacement generator of $\bar{A}_\alpha^{-1}$ can be exactly computed from the approximate RF tree for every node α in the RF tree of $\bar{A}$.

ALGORITHM 5.3. *Recovering the exact RF tree from an approximation of Toeplitz-like matrix*

Input: An approximate augmented RF tree of depth $t = t^*$

Output: The exact extended RF tree of depth $t = t^*$

begin

step 1 Compute the determinant of the matrices $\tilde{A}_\alpha$:

for all nodes α of depth t do in parallel

$\det_\alpha := (\tilde{A}_\alpha)_{11}$ ($\tilde{A}_\alpha$ is a 1×1 matrix)

end for

for $s := t-1$ down to 0 do (sequentially)

for all nodes α of depth s do in parallel

$\det_\alpha := \det_{\alpha 0} \times \det_{\alpha 1}$

end for

end for

step 2 Compute integer multipliers $\{\bar{m}_\alpha\}$ such that the matrices $\bar{m}_\alpha A_\alpha$ all have integer entries

$\bar{m}_{<>} := 1$

for $s := 0$ to $t-1$ do (sequentially)

for all nodes α of depth s do in parallel

$\bar{m}_{\alpha 0} := \bar{m}_\alpha$

$\tilde{m}_{\alpha 1} := \bar{m}_\alpha \times \det_{\alpha 0}$

$\bar{m}_{\alpha 1} := \tilde{m}_{\alpha 1}$ rounded to the nearest integer

end for

end for

step 3 for $s := 0$ to t do (sequentially)

for all nodes α of depth s do in parallel

Call Algorithm 5.2 to compute a (+)-displacement generator of $\bar{m}_\alpha \bar{A}_\alpha$

Compute the (+)-displacement generator of $\bar{A}_\alpha$ using the fact that $\phi_+(\bar{A}_\alpha) := \frac{1}{\bar{m}_\alpha} \phi_+(\bar{m}_\alpha \bar{A}_\alpha)$, so it suffices to divide each of the entries of one of the matrices in the displacement generator for $\bar{m}_\alpha \bar{A}_\alpha$ by $\bar{m}_\alpha$.

end for

end for

step 4 for $s := 0$ to t do (sequentially)

for all nodes α of depth s do in parallel

Call Algorithm 5.2 to compute a (+)-displacement generator of $\bar{m}_\alpha \mathrm{adj}(\hat{A}_\alpha)$

Compute the (−)-displacement generator of $\mathrm{adj}(\bar{A}_\alpha)$ using the fact that $\phi_-(\mathrm{adj}(\bar{A}_\alpha)) = \frac{1}{\bar{m}_\alpha}\phi_+(\bar{m}_\alpha \mathrm{adj}\hat{A}_\alpha)$, so it suffices to divide each entry of one of the matrices in the displacement generator for $\bar{m}_\alpha \mathrm{adj}(\bar{A}_\alpha)$ by $\bar{m}_\alpha$.

end for

end for

end

LEMMA 5.19. *The exact extended RF tree of $\bar{A}$ can be computed from the approximate extended RF tree of $\bar{A}$ in $O(\log^2 n)$ time using $O(n/\log n)$ processors.*

5.3.3. Newton Hensel Lifting for RF Trees of Bounded Displacement Rank. Recall the step to recover the extended RF tree of A from the extended RF tree of $\bar{A}$ in the algorithm for general dense matrices. The main technique used is Newton Hensel Lifting. Reif's algorithm for Toeplitz and Toeplitz-like matrix computations uses the same approach.

Given a displacement generator of a nonsingular matrix A and $S^{(0)}(A) = A^{-1} \bmod p^{2^k}$, let $S^{(k)}(A) = A^{-1} \bmod p^{2^k}$ and compute $S^{(k)}(A) = S^{(k-1)}(A)(2I - AS^{(k)}(A)) \bmod p^{2^k}$. All these matrices involved are given by their (±)-displacement generators, thus $S^{(k)}(A)$ can be computed from $S^{(k-1)}(A)$ in $O(\log n)$ time using $O(n)$ processors. Reif gives the following lemma ([Rei95], Proposition 6.1, p.24).

LEMMA 5.20. *For all $k \geq 0$, if A has (+)-displacement rank r, then $S^{(k)}(A) = A^{-1} \bmod p^{2^k}$ has (−)-displacement rank at most r in $\mathbb{Z}_{p^{2^k}}$.*

The assertion "$S^{(k)}(A) = A^{-1} \bmod p^{2^k}$ has $(-)$-displacement rank at most $r \in \mathbb{Z}_{p^{2^k}}$" (apparently) means that there exist integer matrices $G, H \in \mathbb{Z}^{n \times r}$ such that $\phi_-(S^{(k)}(A)) - GH^T = p^{2^k} C_1$, and also that $S^{(k)}(A) - A^{-1} = p^{2^k} C_2$, where C_1 and C_2 are $n \times n$ matrices with rational numbers as entries, such that none of the integer denominators of the entries of these matrices is divisible by p.

Suppose $S^{(k)}(A)$ has $(-)$-displacement rank at most r in $\mathbb{Z}_{p^{2^k}}$ for $k \geq 0$, $S^{(k+1)}(A)$ is computed from $S^{(k)}(A)$ in the usual way. In general $S^{(k+1)}(A)$ is then given by a $(-)$-generator of length $3r + 5$ using the method for Toeplitz-like matrix multiplication (see Lemma 5.8). Then an additional step is required to reduce the length of the displacement generator to minimum.

Reif solves this problem by assuming that the $r \times r$ principal submatrix of the displacement generator is nonsingular (see Corollary 2.1 in [Rei95]) and, again, this assumption is not true for arbitrary inputs. It does not appear that Reif's solution for this problem is correct.

5.3.4. Further Work. The following questions need to be solved to complete an analysis (and correction) of Reif's processor efficient algorithm for Toeplitz and Toeplitz-like matrix computations.

- In the stage of approximate Newton iteration, $\hat{B}^{(k)} = \tilde{B}^{(k-1)}(2I - \tilde{A}^{(k)}\tilde{B}^{(k-1)})$, how can one compute an approximation $\tilde{B}^{(k)}$ from $\hat{B}^{(k)}$ such that $\tilde{B}^{(k)}$ has small displacement rank and $\left\|\tilde{B}^{(k)} - \hat{B}^{(k)}\right\|$ is small? Possibly, all that is required here is a more complete analysis of Pan's solution for this problem—and estimation of the precision needed to find a sufficiently accurate estimate $\tilde{B}^{(k)}$ when finite precision arithmetic is used.
- How can one solve the corresponding problem for the later stage of the algorithm that uses Newton Hensel Lifting?

It appears that these problems can be solved and that it is not necessary to increase the parameters p and d in order to do so. Thus it is conjectured that the improvement

on Reif's algorithm for general dense matrix computations can also be achieved for this algorithm.

CHAPTER 6
Conclusion

This thesis has provided a complete analysis for Reif's recent processor efficient parallel algorithm for general dense matrix factorizations [Rei94]. As a by-product of the analysis, a modified (and somewhat simplified) version of Reif's algorithm has been achieved. A brief description of Reif's processor efficient parallel algorithm for Toeplitz and Toeplitz-like matrix computations [Rei95] has also been given in the thesis. Some ideas to improve this algorithm have been discussed. However, certain difficulties remain. A complete analysis of this algorithm is considered to be future work.

It was shown in Chapter 3 that approximate Newton iteration (which works with approximations of the input matrix A rather than A itself in order to approximate A^{-1}) could be used to generate a sequence of approximations which converges quadratically. Suppose a matrix $\bar{A}$ is close to a diagonal matrix $D = dI$ such that $\left\|\bar{A} - D\right\| \leq \frac{d\epsilon}{2}$ for some $\epsilon \leq \frac{1}{2}$. It is possible to compute approximations of $\bar{A}^{-1}$ with a sequence of approximations of $\bar{A}$ instead of $\bar{A}$. The results in Chapter 3 have shown that if the sequence of the approximations of $\bar{A}$ converges quadratically, then a sequence of approximations of $\bar{A}^{-1}$ can be computed so that the sequence converges quadratically as well; furthermore, a sequence of approximations to the Schur complement of $\bar{A}$ that converges quadratically can also be computed.

Reif's algorithm for general dense matrix computations was analyzed and its modified version was given in Chapter 4. Several problems in matrix computations can

be efficiently reduced to computing the RF tree of the input matrix. The RF tree used in this thesis differs slightly from the one in Reif's original paper. The definition was changed to simplify the algorithms that compute and use these trees. The analysis showed that for a symmetric positive definite matrix A, the RF tree of $\bar{A} = A + dI$ can be computed in $O(\log^2 n)$ time using $O(P(n))$ processors provided that $d = p\hat{d} \geq (2n \|A\|)^2$ and p is a prime number. Furthermore, it was shown that if p is uniformly and randomly chosen from the interval $[n^3 \log^2 \|A\|, 4n^3 \log^2 \|A\|]$, then the RF tree of A can be exactly recovered from the RF tree of $\bar{A}$ in $O(\log^2)$ time using $O(M(n) \log n)$ operations with high probability. Since the modified version of Reif's algorithm in this thesis uses relatively "small" numbers throughout the computation, the bit precision could be shown to be optimal.

The ideas to simplify Reif's algorithm for Toeplitz and Toeplitz-like matrix computations were given in Chapter 5. Reif's algorithm for Toeplitz and Toeplitz-like matrix computations was very similar to the one for general dense matrix computations. Thus it was conjectured that the improvement in the modified version of Reif's algorithm for general dense matrix computations could be also achieved in the case of Toeplitz and Toeplitz-like matrix computations; however, certain steps of Reif's algorithm remain unclear. The remaining question is how to reduce the displacement rank of a Toeplitz-like matrix and how to find a close approximation with small displacement rank in the stages of Newton iterations and Hensel Lifting.

Once the analysis of Reif's algorithm for Toeplitz and Toeplitz-like matrix computations is complete, one may extend the algorithm to the computations of other classes of matrices, such as Hankel-like, Hilbert-like, and Vandermonde-like matrices by applying techniques in [Pan90a] (see also [BP94]).

It is still interesting to look for processor efficient parallel algorithms for Toeplitz and Toeplitz-like matrix computations over an arbitrary field.

Bibliography

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, 1974.

[BA80] R. R. Bitmead and B. D. O. Anderson. Asymptotically fast solution of Toeplitz and related systems of linear equations. *Linear Algebra and Its Applications*, 34:103–116, 1980.

[Ber84] S. J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18:147–150, 1984.

[BGY80] R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun. Fast solution of Toeplitz systems of equations and computation of Padé approximants. *Journal of Algorithms*, 1:259–295, 1980.

[BM75] A. Borodin and I. Munro. *The Computational Complexity of Algebraic and Numerical Problems.* American Elsevier, New York, 1975.

[BP93] D. Bini and V. Pan. Improved parallel computations with Toeplitz-like and Hankel-like matrices. *Linear algebra and its applications*, 188/189:3–29, July 1993.

[BP94] D. Bini and V. Pan. *Polynomial and Matrix Computations*, volume 1. Birkhäuser, Boston, 1994.

[BvzGH82] A. Borodin, J. von zur Gathen, and J. Hopcroft. Fast parallel matrix and GCD computations. *Information and Control*, 52:241–256, 1982.

[Chi85] A. L. Chistov. Fast parallel calculation of the rank of matrices over a field of arbitrary characteristic. In *Lecture Notes in Computer Science*, volume 199, pages 63–69. Springer, 1985.

[CK87] D. G. Cantor and E. Kaltofen. Fast multiplication of polynomials with coefficients from an arbitrary ring. Technical Report 87-35, Dept. Computer Science, Rensselaer Polyt. Inst., Troy, New York, 1987.

[CR93] J. Cheriyan and J. Reif. Parallel and output sensitive algorithms for combinatorial and linear algebra problems. In *Proc. ACM Symposium on Parallel Algorithms and Architecture*, pages 50–56, New York, 1993. ACM Press.

[Csa76] L. Csanky. Fast parallel matrix inversion algorithms. *SIAM Journal on Computing*, 5:618–623, 1976.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal on Symbolic Computation*, 64:151–280, 1990.

[Ebe91] W. Eberly. On efficient parallel independent subsets and matrix factorizations. In *Proc. 3rd IEEE Conference on Parallel and Distributed Processing*, pages 204–211. IEEE Computer Society Press, 1991.

[Gie] M. Giesbrecht. Nearly optimal algorithms for canonical matrix forms. To appear in SIAM Journal on Computing.

[GP89] Z. Galil and V. Pan. Parallel evaluation of the determinant and of the inverse of a matrix. *Information Processing Letters*, 30:41–45, 1989.

[GS72] I. C. Gohberg and A. A. Semencul. On the inversion of finite Toeplitz matrices and their continuous analogs. *Mat. Issled.*, 2:201–233, 1972. In Russian.

[GVL90] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins Series in Mathematical Sciences. Johns Hopkins Press Ltd., London, 2nd edition, 1990.

[JáJ92] J. JáJá. *An Introduction to Parallel Algorithms.* Addison Wesley, 1992.

[Joh90] D. S. Johnson. A catalog of complexity classes. In *Handbook of Theoretical Computer Science*, chapter 2, pages 67–161. Elsevier Science Publishers B. V., 1990.

[Kal94] E. Kaltofen. Asymptotically fast solution of Teoplitz-like singular linear systems. In *Proc. International Symposium on Symbolic and Algebraic Computation*, pages 297–304, 1994.

[KKM79] T. Kailath, S.-Y. Kung, and M. Morf. Displacement ranks of matrices and linear equations. *Journal of Mathematical Analysis and Applications*, 68(2):395–407, 1979.

[KP91] E. Kaltofen and V. Pan. Processor efficient parallel solutions of linear systems over an abstract field. In *Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 180–191, 1991.

[KP92] E. Kaltofen and V. Pan. Processor-efficient parallel solutions of linear systems part II: The positive characteristic and singular cases. In *Proc.,33rd IEEE Symposium on Foundations of Computer Science*, pages 714–723, 1992.

[KS91] E. Kaltofen and B. D. Saunders. On Wiedemann's method of solving sparse linear systems. In *AAECC-5, Lecture Notes in Computer Science*, volume 539, pages 29–38. Springer, 1991.

[MC79] R. T. Moenck and J. H. Carter. Approximate algorithms to derive exact solutions to systems of linear equations. In *Proc. Eurosam*, volume 72 of *Lecture Notes in Computer Science*, pages 63–73, Berlin, 1979. Springer.

[Mor80] M. Morf. Doubling algorithms for Toeplitz and related equations. In *Proc. IEEE International Conference on ASSP*, pages 954–959. IEEE Computer Society Press, 1980.

[Mul87] K. Mulmuley. A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. *Combinatorica*, 7:101–104, 1987.

[Pan85] 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*, volume 206 of *Lecture Notes in Computer Science*, pages 504–521, New Delhi, India, 1985. Springer, Berlin.

[Pan87] V. Pan. Complexity of parallel matrix computations. *Theoretical Computer Science*, 54:65–85, 1987.

[Pan90a] V. Pan. On computations with dense structured matrices. *Math. Comp.*, 55(191):179–190, 1990.

[Pan90b] V. Pan. Parallel least-squares solution of general and Toeplitz-like linear systems. In *Proc. 2nd Annual ACM Symposium on Parallel Algorithms and Architecture*, pages 244–253, 1990.

[Pan92a] V. Pan. Complexity of computations with matrices and polynomials. *SIAM Review*, 34(2):225–262, June 1992.

[Pan92b] V. Pan. Concurrent iterative algorithm for Toeplitz-like linear systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(5):592–600, 1992.

[Pan92c] V. Pan. Parallel solution of Toeplitz-like linear systems. *Journal of Complexity*, 8:1–21, 1992.

[Pan92d] V. Pan. Parametrization of Newton's iteration for computations with structured matrices and applications. *Computers Math Applic*, 24(3):61–75, 1992.

[PR85] V. Pan and J. Reif. Efficient parallel solutions of linear systems. In *Proc. 17th Annual ACM Symposium on Theory of Computing*, pages 143–152, Providence, RI, 1985.

[PR87] V. Pan and J. Reif. Some polynomial and Toeplitz matrix computations. In *Proc. 28th Annual IEEE. Symposium Foundations of Computer Science*, pages 173–181, 1987.

[PR89] V. Pan and J. Reif. Fast and efficient parallel solution of dense linear systems. *Computers Math. Applic.*, 17(11):1481–1491, 1989.

[PR93] V. Pan and J. Reif. Fast and efficient parallel solution of sparse linear systems. *SIAM Journal on Computing*, 22(6):1227–1250, December 1993.

[PS78] F. P. Preparata and D. V. Sarwate. An improved processor bound in fast matrix inversion. *Information Processing Letters*, 7:148–150, 1978.

[Rei94] J. Reif. $O(\log^2 n)$ time efficient parallel factorization of dense, sparse separable, and banded matrices. In *Proc. 6th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '94)*, July 1994. Page references in this thesis are for the draft of the journal version of this paper (manuscript, 1995).

[Rei95] J. Reif. Work efficient parallel solution of Toeplitz and other structure linear systems. In *Proc. 27th Annual ACM Symposium on Theory of Computing*, 1995. Page references in this thesis are for the draft of journal version of this paper (submitted to SIAM Journal on Computing.

[Sch82] A. Schönhage. The fundamental theorem of algebra in terms of computation complexity. manuscript, 1982.

[Tre64] W. F. Trench. An algorithm for inversion of finite Toeplitz matrices. *Journal of SIAM*, 12(3):515–522, 1964.

[Wie86] D. H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, IT-32:54–62, 1986.

[Wil65] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, 1965.

[Woo93] D. H. Wood. Product rules for the displacement of near-Toeplitz matrices. *Linear Algebra and Its Applications*, 188/189:641–663, 1993.

[Zip93] R. Zippel. *Effective Polynomial Computation.* Kluwer Academic Publishers, MA, USA, 1993.