# Efficient Parallel Factorization and Solution of Structured and Unstructured Linear Systems

John H. Reif [1]

*Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129*

E-mail: reif@cs.duke.edu

---

This paper gives improved parallel methods for several exact factorizations of some classes of symmetric positive definite (SPD) matrices. Our factorizations also provide us similarly efficient algorithms for exact computation of the solution of the corresponding linear systems (which need not be SPD), and for finding rank and determinant magnitude. We assume the input matrices have entries that are rational numbers expressed as a ratio of integers with at most a polynomial number of bits $\beta$. We assume a PRAM model of parallel computation, with unit cost arithmetic operations, including division, over a finite field. We require only bit precision $O(n(\beta + \log n))$, which is the asymptotically optimal bit precision for $\beta \geq \log n$. Our algorithms are randomized, giving the outputs with high likelihood $\geq 1 - 1/n^{\Omega(1)}$. We compute $LU$ and $QR$ factorizations for dense matrices, and $LU$ factorizations of sparse matrices which are $s(n)$-separable, reducing the known parallel time bounds for these factorizations from $\Omega(\log^3 n)$ to $O(\log^2 n)$, without an increase in processors (matching the best known work bounds of known parallel algorithms with polylog time bounds). Using the same parallel algorithm specialized to structured matrices, we compute $LU$ factorizations for Toeplitz matrices and matrices of bounded displacement rank in time $O(\log^2 n)$ with $n \log\log n$ processors, reducing by a nearly linear factor the best previous processor bounds for polylog times (however, these prior works did not generally require unit cost division over a finite field). We use this result to solve in the same bounds: polynomial resultant; and Padé approximants of rational functions; and in a factor $O(\log n)$ more time: polynomial greatest common divisors (GCD) and extended GCD; again reducing the best processor bounds by a nearly linear factor.

---

**Keywords:** parallel algorithms, linear systems, LU factorization, dense matrices, sparse matrices, Newton iteration, structured matrices, Toeplitz matrices, displacement rank, polynomial greatest common divisors, GCD, resultant, Padé approximation.

**List of symbols:**

$O$ —'oh'

$0$ — 'zero'
$\ell$ — 'ell'
$1$ — 'one'
$\bmod$ — 'mod'
$\times$ — 'times'
$n$ — 'en'
$\mathcal{NC}$ — ' script NC
$\mathcal{RNC}$ — ' script RNC
$\alpha$ — 'alpha'
$\beta$ — 'beta'
$\delta$ — 'delta'
$\epsilon$ — 'epsilon'
$\phi$ — 'phi'
$\gamma$ — 'gamma'
$\kappa$ — 'kappa'
$K$ — 'upper case kay'
$k$ — 'lower case kay'
$\lambda$ — 'lambda'
$\mu$ — 'mu'
$\omega$ — 'lower case omega'
$\Omega$ — 'upper case Omega'
$\pi$ — 'pi'
$\rho$ — 'rho'
$\sigma$ — 'sigma'
$\Sigma$ — 'upper case Sigma'
$\tau$ — 'tau'
$\xi$ — 'xi'
$\zeta$ — 'zeta'

# 1. INTRODUCTION

## 1.1. Assumptions and Machine Model

For our model of computation, we assume the algebraic Parallel Random Access Machine (PRAM) where each arithmetic or logical operation such as addition, subtraction, multiplication, division, and comparison over the domain of rational numbers can be done in one step by a given processor. We also assume the floor function, which gives the largest integer $\leq$ a given rational number. Processors can execute such operations in parallel. Our time complexity bounds are based on arithmetic complexity, that is the number of these parallel steps. We also assume

[1]Supported in part by Grants NSF/DARPA CCR-9725021, CCR-96-33567, NSF IRI- 9619647, ARO contract DAAH-04-96-1-0448, and ONR contract N00014-99-1-0406. A preliminary version of a section of this document appeared as "$O(\log^2 n)$ Time Efficient Parallel Factorization of Dense, Sparse Separable, and Banded Matrices" in Proc. 6th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'94), NJ, July 1994, and also a section appeared as "Work Efficient Parallel Solution of Toeplitz Systems and Polynomial GCD" Proc. 27th Annual ACM Symposium of Theory of Computing (STOC'95), Las Vegas, Nevada, May,1995. A PDF version of this paper is at http://www.cs.duke.edu/~reif/paper/newton/Toeplitz.Newton.pdf

the PRAM has a sequential source of random numbers. We assume the $n \times m$ matrices input to our algorithms have *P-rational entries*: either integer entries of magnitude $\leq 2^{\beta}$, where $\beta \leq n^{O(1)}$, or rational entries expressible as ratio of integers of this magnitude.

### 1.2. Motivation

Many problems in engineering and science rely on the solution of linear systems. As the problem size of a linear system grows, the resulting linear systems can grow to enormous size, and can, in turn, require very large computational effort to solve. This motivated the search for algorithms which are work efficient. One of the success stories in the field of computer science algorithms and numerical analysis was the development of efficient sequential algorithms for rapid solution of linear systems. Previous researchers have exploited sparsity and/or the structure of the linear system to improve these computations. The use of parallel processing can potentially give a further increase in speed. However, there remains a considerable discrepancy between the theoretical methods proposed for parallel solution of linear systems and the methods that are actually used. Our goals are more modest; we wish to establish, theoretically, improved time and processor bounds, keeping in mind that this is at best just a first step toward the more practical goals just discussed.

### 1.3. Bounds on Basic Computations on Matrices and Polynomials

Basic computations on dense matrices, such as multiplication, inverse, etc. require a large amount of sequential time which is theoretically $n^{\omega^*}$ where $\omega^*$ drops significantly below 3 ([81, 25] give the best known bound 2.376) but with significant increase in constant factors, so that in practice this sequential time is close to $n^3$. Even with major breakthroughs, the sequential time must remain $\geq n^2$. This can be excessive for many applications. Let $M(n)$ be the minimum number of PRAM processors necessary to multiply two $n \times n$ matrices in $O(\log n)$ parallel steps. The best known bound [81, 25] on $M(n)$ is $n^{\omega}$, where $\omega = 2.376$. Parallel algorithms for dense matrices must also do at least this amount of work, and this results in either excessive processor requirements or slow time bounds.

Let $P(n)$ denote the number of arithmetic processors used to multiply two degree $n$ polynomials in $O(\log n)$ parallel time. (Since we are primarily concerned only with parallel complexity bounds, we will not define separate notation for the sequential time for this task. Instead, we observe that $O(P(n)\log n)$ upper bounds the number of arithmetic operations used to multiply two degree $n$ polynomials.) It is known that $P(n) = O(n)$ if the field supports an FFT of size $n$ but otherwise the best bound is $P(n) = O(n \log\log n)$ [21].

### 1.4. Known Techniques and Results

**Newton's iteration.** The sequential use of Newton's iteration for the approximate inverse of well conditioned or diagonally dominant (DD) matrices was developed by Ben-Israel [4], Ben-Israel and Cohen [5], and Hotelling [40, 41]. Later Pan and Reif [69, 71] considered parallel applications; they showed that the inverse of a well conditioned nonsingular $n \times n$ dense integer or rational matrix, and the solution of the corresponding linear system, can be approximately computed with high accuracy by Newton iterations in parallel time $O(\log^2 n)$ with $M(n)$ processors, without construction of the characteristic polynomial.

**Hensel Lifting and Variable Diagonal.** Subsequently Pan [62, 63] showed that the inverse of an arbitrary (not necessarily well conditioned) nonsingular $n \times n$ dense integer or rational matrix $A$ can be exactly computed in parallel time $O(\log^2 n)$ with $M(n)$ processors (similar results are also known for matrix inverse over arbitrary fields [47, 48, 42] using a reduction to the computation of the characteristic polynomial or a related form). Thus the linear system $Ax = b$ can be exactly solved within this complexity by computing $x = A^{-1}b$, where $A^{-1}$ is the matrix inverse.

The general technique of approximate solution via Newton's iteration, followed by Hensel lifting, has a long history in numerical and algebraic computation. Pan [62, 63] developed what he calls the *variable diagonal technique*, which modifies the input matrix (which initially may have arbitrary condition, so could be very badly conditioned), so that the resulting matrix is strongly diagonally dominant and has condition nearly 1. Explicit computation of matrix inverses of badly conditioned matrices is thus avoided, and instead inverses of diagonally dominant matrices are approximated.

Given a nonsingular integer matrix $A$, the variable diagonal technique constructs a matrix $\bar{A} = A + psI$ for a random prime $p$ and for a large integer $s$, so that $\bar{A}$ is strongly diagonally dominant and $\bar{A} = A \pmod p$. Since $\bar{A}$ is diagonally dominant, Newton's iteration can be effectively applied to approximate the inverse $\bar{A}^{-1}$, the determinant $det(\bar{A})$ and $adj \bar{A} = \bar{A}^{-1} det(\bar{A})$. Since $\bar{A}$ is an integer matrix, rounding-off turns the approximations into exact values of $det(\bar{A})$ and $adj \bar{A}$. Then applying the standard homomorphism from the rationals $\mathbf{Q}$ to the finite field $\mathbf{Z}_p$ to rational matrices $\bar{A}^{-1}$ and $A^{-1}$ gives $\bar{A}^{-1} \pmod p \equiv A^{-1} \pmod p$. This is extended by Moenck and Carter's Newton-Hensel lifting procedure, to $\bar{A}^{-1}$ modulo a high power of $p$. From this, $adj A, det(A), A^{-1}$ are recovered.

**Matrix Factorizations.** Numerical analysis practitioners often solve linear systems by computing a matrix factorization and solving the resulting triangular linear systems. For example, given a nonsingular $n \times n$ matrix $A$. If $A$ is symmetric positive definite (SPD), then $A$ can be factored $A = LU$ where $U = L^T$ (also known in this case as a Cholesky factorization), where $L$ is nonsingular lower triangular and $U$ is nonsingular upper triangular.

Previous results of Pan [63] gave parallel time $O(\log^3 n)$ with $M(n)$ processors for factoring dense SPD matrices, including $LU$ and $QR$ factorizations, and also computing their reduction into upper Hessenberg form. Also, Pan, Sibe, and Atinkpahoun [74] show the solution and determinant of $b$-banded matrices can be computed in parallel time $O(\log n \log b)$ with $M(b)n/b$ processors.

Many efficient parallel algorithms for exact computation of the determinant of $A$ (for recent examples, see [47, 48, 42]) also require the computation of the characteristic polynomial of $A$. In contrast, given the $LU$ decomposition as above, if $A = LU$, then $det(A) = det(L)det(U)$, and otherwise if $A^T A = LU$, then $det(A)^2 = det(A^T A) = det(L)det(U)$. The determinants of these triangular matrices are obtained by multiplying all the elements of their principal diagonals.

Furthermore, singular value decompositions and eigenvalue computations generally begin with $QR$ factorization which is computable from the $LU$ factorization.

Eigenvalue computations generally use a further reduction to upper Hessenberg form computed from the $QR$ factorization (see Golub and van Loan [35]). Thus matrix factorizations of various types are used extensively in numerical computations and are essential in many applications. For dense matrices, known efficient parallel algorithms for exact $LU$ and $QR$ factorization and reduction to upper Hessenberg form ([63]) cost $O(\log^3 n)$ time.

**Recursive Factorization of Matrices.** *Recursive factorization* of matrices is a divide and conquer technique used in many theoretically efficient sequential algorithms for matrix inverse. For example, the Strassen block matrix algorithm computes the inverse of an $n \times n$ matrix by partitioning it into four blocks (each of size $(n/2) \times (n/2)$), and so reduces the problem to computing matrix inverses and products on the block submatrices. A similar technique can be used for $LU$ factorization of SPD matrices (see [1]) by recursive factorization of the Schur complement submatrices induced by block Gaussian elimination, and Trench [83] also used this technique for the inversion of Toeplitz matrices. The requirement that the matrix be SPD presents no difficulties, as we can use the normal form reduction given in Subsection 2.2.

**Symmetric Matrices with Separable Graphs.** A family of graphs is $s(n)$-*separable* if, given a graph $G$ in the family of $n \geq O(1)$ nodes, we can delete a set of $s(n)$ nodes, separating $G$ into subgraphs in the family of size $\leq 2/3n$ nodes. Clearly $d$-dimensional grids or dissection graphs are $s(n) = O(n^{\frac{d-1}{d}})$ separable, and Lipton and Tarjan [54] showed planar graphs are $O(\sqrt{n})$-separable. A sparsity graph of a symmetric matrix has a vertex for every row (column) of the matrix and an edge wherever there is a non-zero entry of the adjacency matrix. Matrices with separable sparsity graphs arise naturally from VLSI circuit problems, structure problems, and discretization of 2- or 3- dimensional PDEs. For example, $d$-dimensional PDEs result in matrices whose sparsity graphs are $d$-dimensional grids or related dissection graphs which are $s(n) = O(n^{\frac{d-1}{d}})$ separable. Lipton, Rose, and Tarjan [53] developed sequential algorithms for recursive factorization of sparse matrices with separable sparsity graphs. Pan and Reif [69, 73] later developed parallel algorithms for recursive factorization (but not $LU$ or $QR$ factorizations) of nonsingular SPD matrices. For this, Pan and Reif [69, 73] gave bounds of parallel time $O(\log^3 n)$ with $n + M(s(n))$ processors. Gazit and Miller [31] gave bounds of parallel time $O(\log^2 n \log\log n)$ with $n + M(s(n))$ processors and Armon and Reif [3] decreased this parallel time to $O(\log^2 n)$ with $n + M(s(n))^{1+\epsilon}$ processors, for $\epsilon > 0$.

**Stream Contraction** is a technique developed by Pan and Reif [72] to decrease the time to solve combinatorial matrix problems over semirings; the stream contraction method decreased the parallel time by a logarithmic factor without a processor penalty.

## 1.5. Dense Structured Matrices

There is a large body of work on sequential algorithms which reduce the amount of work in the case where the dense matrices possess certain regular structures. Throughout this paper, we refer to such matrices as *structured*. Examples of structured matrices include: Toeplitz matrices and their generalizations, Vandermonde matrices and their generalizations, Hilbert matrices and their generalizations, Hankel matrices and their generalizations; the generalizations include block matrices with a constant number of submatrix blocks of a single above type (for example Hankel block matrices).

**Displacement Operators for Compact Generation and Compression of Structured Matrices.** Kailath and his collaborators ([45], [46]) generalized Toeplitz and Hankel matrices by defining various classes of matrices with bounded *displacement rank*. These matrices can be stored compactly by representing them by their *displacement generators*. As an additional benefit, the use of displacement generators gave fast sequential algorithms for the inversion and factorization of such matrices (see [14] [59] [2] [23] [44]). These so called Schur algorithms for matrices with constant displacement rank run efficiently on sequential machines. Their implementation on parallel machines has been extensively studied by Kailath and his many coworkers and PhD students. Unfortunately, it is not clear how to parallelize the Schur algorithm to get *provable small parallel time with small processor bounds*; the known [67, 68] polylog time algorithms have quadratic processor bounds.

A matrix $A = [a_{ij}]$ is *Toeplitz* if $a_{i,j} = a_{i+k,j+k}$ for each $k$ where the matrix elements are defined. We define (this is a slight simplification of standard definitions) an $n \times n$ matrix to have *displacement rank* $\delta$ if it can be written as the sum of $\delta$ terms, where each term is either (i) the product of a lower triangular Toeplitz matrix and an upper triangular Toeplitz matrix or (ii) the product of an upper triangular Toeplitz matrix and a lower triangular Toeplitz matrix.

Note that any matrix has displacement rank at most $n$, and a Toeplitz matrix and its inverse has displacement rank 2 (see [34]). Also, for this definition of displacement rank, the inverse of a matrix of displacement rank $\delta$ has displacement rank $\delta$. We will also consider block matrices, with a constant number of submatrix blocks of a single above type (for example block Toeplitz matrices), which have bounded displacement rank.

It happens (and we will discuss this in detail in later sections) that the above structured matrices, particularly generalized Toeplitz and their generalizations, appear naturally and are used in many applications. Thus, computations on structured matrices are interesting in their own right. Perhaps the most prevalent class of structured matrices are Toeplitz matrices and Toeplitz block matrices, which arise in many computations on polynomials. Examples of Toeplitz block matrices are Sylvester and their submatrices known as subresultant matrices (see Brown and Traub [19]), which arise in polynomial greatest common divisors (GCD), LCD and univariate resultant computations. It is well known (see [19] and [15]) that Toeplitz matrices and matrices of bounded displacement rank also arise naturally in many other algebraic computation and signal processing applications, such as: linear prediction (see Makhoul [55]), decoding error correcting codes and linear feedback shift-register synthesis (see Gustavson [37]), Padé Approximants least-squares estimation (see [43], [36]), and data compression applications.

**Previous Results for Structured Matrices.** There are known efficient sequential algorithms [14, 59, 18] for inverse, determinant, linear system solution, factorization, and finding the rank for the case of Toeplitz matrices and matrices of bounded displacement rank with sequential time cost $P(n)\log^2 n$ (see Subsection 1.3 for definition of $P(n)$), but there are no such results for efficient parallel algorithms. Previously, the best parallel algorithms [70, 64, 67, 68] for these problems required $\Omega(\log^2 n)$ time using $nP(n)/\log n$ processors, and all polylog ($O(\log^{O(1)} n)$) time parallel algorithms used at least $\Omega(n^2/\log^{O(1)} n)$ processors.

### 1.6. Our Results

This paper provides efficient parallel algorithms for factoring matrices in time $O(\log^2 n)$ with a near optimal number of processors. We show our algorithms are nearly optimal in terms of time, processors, and bit precision. Our parallel algorithms are randomized, giving the outputs within the stated bounds with high likelihood $\geq 1 - 1/n^{\Omega(1)}$, using constant number of random variables each ranging over a domain of size $(n\|A\|)^{O(1)}$. The only exceptions are the rank and Hessenberg computations, which require a further linear number of such random variables.

All our computations require bit precision $O(n(\beta + \log n))$, which is the asymptotically optimal bit precision for $\beta \geq \log n$ since the determinant, exact $LU$ factorization, and matrix inverse require bit precision at least $\Omega(n\beta)$.

Note that we assume a PRAM model of parallel computation, with unit cost arithmetic operations, including division, over a finite field, whereas prior works did not generally require unit cost division over a finite field. See the Conclusion Section 9 for a further discussion of the repercussions of this assumption. Since numerical analysis practitioners would not necessarily use algorithms which need the precision of the computation in the worst case (i.e., worst ill conditioned matrix) as it happens in our proposed algorithms, the most convincing motivation for our algorithms is their theoretical interest.

To solve the various matrix problems considered in this paper, we compute a recursive factorization (RF) of an SPD matrix, using the techniques of Newton's iteration and Newton-Hensel lifting, as well as the Variable Diagonal technique. We provide improvements to these techniques by application of a generalization of the stream contraction technique of Pan and Reif [72] to do a multilevel, pipelined Newton iteration, followed by multilevel, pipelined Newton-Hensel lifting. Our algorithms give an exact factorization, but nevertheless avoid computation of the characteristic polynomial or related forms.

Using reductions to the recursive factorization algorithm, we exactly compute, for SPD matrices, $LU$ and $QR$ factorizations, and in the generic case where the minimal polynomial is the characteristic polynomial, we also compute their reduction into upper Hessenberg form via a Las Vegas randomized algorithm. Using further reductions to the $LU$ factorization, for arbitrary integer or rational matrices (which need not be SPD), we exactly compute solution of the corresponding linear systems, the determinant magnitude, the inverse, and the rank.

**Note:** To simplify our presentation, we present *a gradual development and refinement of the RF algorithm.* We first describe our RF algorithm and provide the analysis in the case of dense unstructured matrices, and only later introduce the complexities of structured matrices; there we specialize the RF algorithm and extend its analysis to the important case of inputs matrices with bounded displacement rank.

*Our Parallel Algorithms for Dense and Sparse Matrices.* In the case of dense and sparse matrices, we reduce the known parallel time bounds for factorization of these matrices from $\Omega(\log^3 n)$ to $O(\log^2 n)$ without an increase in processors, matching the best known work bounds of known parallel algorithms with polylog time bounds. For dense matrices, we show that all of these factorizations and computations can be done in parallel time $O(\log^2 n)$ with $M(n)$ processors. For sparse matrices, (with $O(n)$ non-zero entries) which are $s(n)$-separable as defined above, we show $LU$ and $QR$ factorizations can be done in parallel time $O(\log^2 n)$ with $n + M(s(n))$ processors where $s(n)$ is of the form $n^\gamma$ for $0 < \gamma < 1$.

*Our Results for Structured Matrices.* Our results apply also to a large class of structured matrices, in particular Toeplitz matrices and matrices of bounded displacement rank. *This is where we get our most significant improvements and results over previous work.* We show that for this class of SPD structured matrices, we can compute an exact $LU$ factorization efficiently in parallel with polylog time bounds, dropping processor bounds from quadratic to linear.

We first describe our parallel algorithm for structured linear systems of bounded displacement rank which costs time $O(\log^3 n)$ using $P(n)$ processors (where $P(n)$ is defined in Subsection 1.3). This processor reduction from $n^2$ to $P(n)$ is the major result of this paper, and uses techniques specific to these structured matrices. Using the generalized stream contraction technique, we decrease our time bounds to $O(\log^2 n)$, using $P(n)$ processors. We also give parallel algorithms, with the same bounds, for finding the exact solution, determinant magnitude, inverse, and rank of these structured matrices.

We apply these results to develop efficient randomized parallel algorithms for the following problems in the same parallel time $O(\log^2 n)$ and $P(n)$ processor bounds: (i) polynomial resultant, and (ii) Padé approximants of rational functions [61, 36], and with a factor $O(\log n)$ more time, (iii) polynomial greatest common divisors (GCD) and extended GCD. With a factor of $O(\log n)$ time increase and the same processor bounds, we also solve: (v) the real root problem: finding all the roots of a polynomial with only real roots. We are the first to give parallel algorithms for these problems that cost polylog time with a linear number of processors. Our results drop by a nearly linear factor the best previous processor bounds for polylog time parallel algorithms for all these problems, and our results are within a constant factor of work compared to the best sequential work bounds of $O(P(n)\log^2 n)$. (Also we obtain similar parallel bounds for the symmetric tridiagonal eigenvalue problem: finding all the eigenvalues of a symmetric tridiagonal $n \times n$ matrix. However, [11] previously gave polylog time bounds for this problem with a linear number of deterministic processors.)

*Cautionary Remarks:* Due to the size of the constant factors in our complexity bounds, and our use the arithmetic model (which does not take into account the Boolean cost), we feel our results are of theoretical interest only.

### 1.7. Organization of the Paper

In Section 1, we have motivated the problems we solve and stated our results and previous results.

Section 2 gives some preliminary definitions. In this section, we define matrix notations, and problems, as well as introduce the Recursive Factorization (RF) Sequence and Tree of matrices. We also discuss a well known reduction involving normal forms which allows us to assume that the input matrix is symmetric positive definite (SPD).

To simplify the presentation of RF algorithms, we first describe in Section 3 our parallel algorithm to compute the RF Tree of a matrix and provide the (much simpler) analysis in the case of dense matrices. A statement of the RF algorithm is given Subsection 3.3. Details of the two components of the algorithm, namely Newton-Hensel lifting and Newton Iteration are separately dealt with in detail in Subsections 3.4 and 3.5. The analysis of the RF algorithm, with the proof that it costs parallel time $O(log^2 n)$ using $M(n)$ processors, is completed in Subsection 3.6.

In Section 4 we introduce the complexities of structured matrices and the important concept of *displacement rank*. We describe the specialization of our parallel algorithm for the RF computation required for matrices of bounded displacement rank, and modifications and extensions required to efficiently do Newton-Hensel Lifting and Newton Iteration in this case. There we also extend the analysis of the RF algorithm to this considerably more complicated case, which is our main result.

Section 5 specializes our RF computation to parallel nested dissection, giving efficient parallel factorizations of symmetric matrices with $s(n)$-separable graphs in $O(log^2 n)$ time and $n + M(s(n))$ processors.

Section 6 gives applications of the RF computation. We provide efficient reductions of several matrix problems (such as determinant, matrix inverse, linear system solution, LU, QR, and Hessenberg factorizations, singular value decomposition, and RANK) to RF computation, and applications to specialized classes of matrices, including sparse and structured matrices. We also give applications to polynomial computations, and to parallel algorithms for Sturm sequences and the real root problem.

Section 7 proves some condition bounds for random matrices. Section 8 gives a randomized algorithm for general problem of constructing displacement generators.

Section 9 concludes the paper with mention of open problems and acknowledgements.

## 2. PRELIMINARIES

This section contains some definitions, notations and previously known results that will be of use later in this paper.

### 2.1. Matrix Definitions

Throughout this paper all logarithms are base 2. All matrix products are inner products. Vectors and matrices are denoted by lower and upper case characters, respectively.

Generally, we assume input matrix $A$ is a square matrix of size $n \times n$, except where we compute $QR$-factors, where the input $A$ can be a rectangular matrix of size $m \times n$, where $m \geq n$. Many of our algorithms assume, without loss of generality, that $n$ is a power of 2. Note that, although all our results apply to rational matrices, we can always multiply a rational matrix by an appropriate integer to form an integer matrix. For simplicity, we assume without loss of generality throughout this paper the matrices *input* to our factorization algorithms have integer entries of $\leq 2^\beta$, where $\beta \leq n^{O(1)}$. However, our algorithms will in general generate and output rational matrices.

$I$ and $O$ denote identity and null matrices of the appropriate sizes. Superscript $T$ indicates transposition, so $A^T$ denotes the transpose of a matrix or of a vector $A$. $A$ is defined to be *symmetric* if $A^T = A$. Let $det(A)$ denote the *determinant* of $A$. $A$ is *singular* if $det(A) = 0$, else it is *nonsingular*. The *rank* of $A$ is the dimension of the linear space spanned by the columns of $A$. If $A$ is nonsingular, then the *inverse* $A^{-1}$ is defined and the *adjoint* is the matrix $\mathrm{adj}(A) = \det(A)A^{-1}$. If $A$ is an integer matrix, then so is the adjoint.

Let $A = [a_{ij}]$ denote the elements of $A$. The *principal diagonal* of $A$ are the elements $a_{11}, a_{22}, \ldots, a_{nn}$. Let $\mathrm{diag}(a_0, \ldots, a_{n-1})$ be the diagonal matrix with diagonal elements $a_0, \ldots, a_{n-1}$ and 0 at the off-diagonal elements. A submatrix of $A$ will be induced by a sequence of rows, say $i_1, i_2, \ldots, i_{n'}$ and columns, say $j_1, j_2, \ldots, j_{m'}$. A *principal submatrix* of $A$ is a square submatrix induced by selecting the same sequence of row indices, say $i_1 = j_1, i_2 = j_2, \ldots, i_{n'} = j_{n'}$ as column indices, and it is a *leading principal submatrix* of $A$ if these indices are consecutive starting at 1, say $i_1 = j_1 = 1, i_2 = j_2 = 2, \ldots, i_{n'} = j_{n'} = n'$. $A$ is *positive definite* if $x^T Ax > 0$ for all nonzero vectors $x$. If $A$ is positive definite, every principal submatrix is also positive definite. If $A$ is positive definite, then $A$ is always nonsingular. A matrix is called *SPD* if it is symmetric and positive definite. For any nonsingular matrix $A$, $A^T A$ is SPD.

$A$ has *orthonormal columns* if $A^T A = I$. If $A$ is also square then $A$ is an *orthogonal* matrix. $A$ is *lower (upper) triangular* if $a_{ij} = 0$ for $i < j$ ($j < i$, respectively). $A$ is *$b$-banded* if $a_{ij} = 0$ for $2|j - i| + 1 > b$, so the non-zeros occur within only a band of width $b$ around the diagonal. $A$ is *$b$-block diagonal* if $A$ has nonzero entries only at a sequence of disjoint $b \times b$ blocks on the diagonal.

**Matrix Norms and Required Bit Precision.** Let the elements of matrix $A = [a_{ij}]$. For $p = 1, 2, \infty$, let $\|A\|_p = \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}$ denote the $p$-norm of matrix $A$, so $\|A\|_1 = \max_j \sum_i |a_{ij}|$ and $\|A\|_\infty = \max_i \sum_j |a_{ij}|$. For each such $p = 1, \infty$, (see Golub and van Loan [35]) $\|A\|_p / n \leq \max_{i,j} |a_{i,j}| \leq \|A\|_2$. Also, for each such $p = 1, \infty$, (see Golub and van Loan [35], p. 57) $\|A\|_p / \sqrt{n} \leq \|A\|_2 \leq \|A\|_p \sqrt{n}$. We can bound $|det(A)| \leq n!(\max_{i,j} |a_{ij}|)^n \leq (n(\max_{i,j} |a_{ij}|))^n$. These norm bounds imply:

Proposition 2.1. *Assuming for each $i, j$ that $|a_{ij}| \leq 2^\beta$, then $\|A\|_2 \leq 2^\beta n^{1.5}$ and $|det(A)| \leq 2^\pi$, where $\pi = n(\beta + \log n)$.*

Note that this bound on the determinant is tight for $\beta \geq \log n$ since the determinant for the tridiagonal matrix with entries of magnitude $2^\beta$ has value $2^{\Omega(n\beta)}$ and so the exact matrix inverse and $LU$ factorization requires bit precision at least $\Omega(n\beta)$. Thus, although the input bit precision is only $O(\beta)$ per input entry, any algorithm for these problems will require computations involving bit precision $\Omega(n\beta)$ per output entry.

If $A$ is nonsingular, let $cond_p(A) = \|A\|_p \cdot \|A^{-1}\|_p$ for $p = 1, 2, \infty$, Note that for nonsingular $A$, $cond_2(A) = \|A\|_2 \cdot \|A^{-1}\|_2 \geq 1$, so $\|A^{-1}\|_2 \geq 1/\|A\|_2$. The above norm bounds imply that $cond_\infty(A)n \leq cond_2(A) \leq cond_\infty(A)n$ for each $p = 1, \infty$. $A$ is *well conditioned* if $cond_p(A) \leq n^{O(1)}$ for any $p \in \{1, 2, \infty\}$. Note that if $A$ is well conditioned, then so are $A^T A$ and $AA^T$. Note: throughout this paper, *we drop the subscript $p$ when we wish to indicate the 2-norm*, so $\|A\| = \|A\|_2$.

### 2.2. The Normal Form Reduction

We compute the inverse and solve linear systems for the class of rational matrices. The only restriction is that they must have (a) input representation of a polynomial number of bits and (b) be nonsingular. They do not have to be positive definite.

Our paper gives an efficient parallel algorithm for the recursive factorization of such matrices with the additional restriction that the input be SPD. The requirement that the matrix be *symmetric positive definite (SPD)* presents no difficulties. To solve the associated matrix inverse and linear systems problems, the input matrix is not SPD, we use a well known reduction involving normal forms. Even if a nonsingular $A$ is not SPD, *the normal form* $AA^T$ *always is SPD*, so $AA^T$ can be factored as $AA^T = LU$ where $U, L$ are triangular (this squares the condition number, which is not a problem since using the techniques of this paper we can make the condition number nearly 1). So we can apply the recursive factorization algorithm giving factorization $AA^T = LU$. We now show this factorization of $AA^T$ allows us to solve the associated linear system and to compute the inverse of $A$.

With this preprocessing, and given an $n$-vector $b$, the linear system $Ax = b$ can be solved using the same factorization $AA^T = LU$ in two back-solving stages by first solving for $n$-vector $y$ and then for $n$-vector $z$ in the triangular linear systems $Ly = b, Uz = y$ (for which there are well known highly efficient parallel algorithms; see JáJá [42]). Then we let $x = A^T z$.

Note that $(AA^T)^{-1} = U^{-1}L^{-1}$, so the inverse of $A$ is $A^{-1} = A^T(AA^T)^{-1} = A^T U^{-1} L^{-1}$.

### 2.3. Recursive Factorization (RF) Sequences of Matrices

Recursive Factorization (RF) SEQUENCE Problem: If possible, compute a sequence of matrices $A = A_0, A_1, \ldots, A_{\log n}$ where for $d = 0, 1, \ldots, \log n - 1$, $A_d$ is an $n/2^d \times n/2^d$ matrix which is partitioned as

$$A_d = \begin{bmatrix} W_d & X_d \\ Y_d & Z_d \end{bmatrix},$$

where $W_d, X_d, Y_d, Z_d$ are matrices each of size $n/2^{d+1} \times n/2^{d+1}$ and $A_{d+1} = Z_d - Y_d W_d^{-1} X_d$ is called the *Schur complement* of Gaussian elimination of the variables of block $W_d$.

If, for any $d$, no such factorization is possible, then $A$ has no RF Sequence. However, any SPD matrix has a unique RF sequence (see Golub and van Loan [35]).

PROPOSITION 2.2. *(follows from known properties of Schur complements and Pan and Reif [69, 73]) In the RF sequence, assuming $A_0$ is SPD, then for all $d, 0 \leq d \leq \log n - 1$ and $\alpha \in \{0,1\}^d$, $A_{d+1}$ and $W_d$ are SPD, $\|A_{d+1}\|, \|W_d\| \leq \|A_d\|$, $\|A_{d+1}^{-1}\|, \|W_d^{-1}\| \leq \|A_d^{-1}\|$, $1/\|A_d^{-1}\| \leq \min(\|A_{d+1}\|, \|W_d\|)$, $1/\|A_d\| \leq \min(\|A_{d+1}^{-1}\|, \|W_d\|^{-1})$, $cond(W_d) \leq cond(A)$.*

PROPOSITION 2.3. *The inverse of $A$ can be recursively computed from the RF sequence of $A$, as follows:*

$$A_d^{-1} = \begin{bmatrix} I & -W_d^{-1}X_d \\ O & I \end{bmatrix} \begin{bmatrix} W_d^{-1} & O \\ O & A_{d+1}^{-1} \end{bmatrix} \begin{bmatrix} I & O \\ -Y_d W_d^{-1} & I \end{bmatrix}.$$

The above formula expresses the inverse of the input matrix in terms of a constant number of products, sums and two inverses of four $n/2 \times n/2$ submatrices.

Note that the Schur complement of an integer matrix $A$ may not be an integer matrix. Therefore we define a multiplier $m_0 = 1$ and for $d > 0$, $m_d = \prod_{i=0}^{d-1} det(W_i)$.

LEMMA 2.1. *Assuming $A_0 = A$ is an integer matrix, then for each $d \geq 0$, $m_d A_d$ is an integer matrix.*

**Proof:** We have defined $A_d$ to be the matrix derived from $A$ by $d$ stages of successive block Gaussian elimination of the variables corresponding to $W_1, \ldots, W_{d-1}$. Note that $W_d$ is of size $n/2^{d+1} \times n/2^{d+1}$. The eliminated variables after stage $d$ are thus the first $e_d = n(1 - 1/2^d) = \sum_{i=0}^{d-1} n/2^{i+1}$ variables of the original linear system of $A$ corresponding to the upper left $e_d \times e_d$ submatrix $W$ of $A$. Consider the alternative partition

$$A = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix}.$$

By well known properties of block Gaussian elimination, we can get the same matrix $A_d$ by a block Gaussian elimination from $A$, where we eliminate all these $e_d$ variables in a single stage rather than $d$ stages. Thus $A_d$ is also equal to the Schur complement $Z - YW^{-1}X$ derived by Gaussian elimination of the variables of submatrix $W$ from $A$. Hence the recursive formulae for the RF sequence can be applied to give a partial LU factorization of $W$ into a product of block triangular matrices with the matrices $W_0, \ldots, W_{d-1}$ on the diagonals. Since the determinants of a triangular matrix are obtained by multiplying all the elements of their principal diagonals (see the proof of Lemma 6.2), we have that

$$det(W) = \prod_{i=0}^{d-1} det(W_i) = m_d.$$

Since $A$ is assumed to be an integer matrix, all its submatrices including $W, X, Y, Z$ are integer matrices and $det(W)W^{-1}$ is the integer adjoint matrix. Thus

$$m_d A_d = det(W)A_d = det(W)Z - Y(det(W)W^{-1})X$$

is an integer matrix. ∎

### 2.4. Recursive Factorization (RF) Trees of Matrices

The *RF TREE Problem* is defined as follows: given an $n \times n$ matrix $A$, where $n$ is a power of 2, compute, if possible, a full binary tree of depth $\log n$ whose nodes are matrices, and where all nodes at depth $d, 0 \leq d \leq \log n$ are $n/2^d \times n/2^d$ matrices with notation $A_\alpha$ where $\alpha \in \{0,1\}^d$ is a binary string of length $d$.

Let <> denote the empty string. If $d = 0$ then $\alpha$ is the empty string and $A_{<>} = A$ is the root of the tree. For $0 \le d < \log n$, each matrix $A_\alpha$ at depth $d$ has exactly two children in the tree, $A_{\alpha 1}$ and $A_{\alpha 0}$ at depth $d+1$ which will be defined by recursion. In particular, for $d = 0, 1, \ldots, \log n - 1$, $A_\alpha$ is an $n/2^d \times n/2^d$ matrix which is partitioned as

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

where $A_{\alpha 0}, X_\alpha, Y_\alpha, Z_\alpha$ are matrices each of size $n/2^{d+1} \times n/2^{d+1}$ and $A_{\alpha 1} = Z_\alpha - Y_\alpha A_{\alpha 0}^{-1} X_\alpha$ is the *Schur complement.* If, for any $d$, no such factorization is possible (this occurs when some $A_{\alpha 0}$ is singular), then $A$ has no RF tree.

**Important Note:** The RF tree of $A$ is very similar to the RF sequence $A = A_0, A_1, A_2, \ldots, A_{\log n}$ defined in Subsection 2.3. All nodes in the RF sequence are subscripted with an integer denoting the depth, whereas all nodes in the RF tree are labeled with a binary string whose length is the depth (thus in the RF tree, the depth is implicitly defined from the length of this binary string and does not have to be explicitly included in the subscript). In particular, $A_d = A_\alpha$ for each $d = 1, \ldots, \log n$, where $\alpha = 1^d$. The only difference is that the RF tree also recursively factors each of the $W_d = A_{\alpha 0}$ matrices appearing in the RF sequence. Any SPD matrix has an RF sequence and therefore an RF tree, and it is unique. Since we have recursively defined: $A_{\alpha 1} = Z_\alpha - Y_\alpha A_{\alpha 0}^{-1} X_\alpha$, this implies:

Proposition 2.4. *(follows from known properties of Schur complements and Proposition 2.2 Pan and Reif [69, 73]) In the RF, assuming $A_d$ is SPD, then for all $d, 0 \le d \le \log n - 1$ and $\alpha \in \{0,1\}^d$; $A_{\alpha 1}, A_{\alpha 0}$ are SPD, $\|A_{\alpha 1}\|, \|A_{\alpha 0}\| \le \|A_\alpha\|$; $\|A_{\alpha 1}^{-1}\|, \|A_{\alpha 0}^{-1}\| \le \|A_\alpha^{-1}\|$; $1/\|A_\alpha^{-1}\| \le \min(\|A_{\alpha 1}\|, \|A_{\alpha 0}\|)$; $1/\|A_\alpha\| \le \min(\|A_{\alpha 1}^{-1}\|, \|A_{\alpha 0}^{-1}\|)$; $cond(A_{\alpha 0}) \le cond(A)$.*

This RF tree gives the following useful recursive formulae holding for all $\alpha \in \{0,1\}^d$, where $0 \le d \le \log n - 1$.

Proposition 2.5. *The LU factorization of $A$ can be recursively computed from the RF tree of $A$, as follows:*

$$A_\alpha = \begin{bmatrix} I & O \\ Y_\alpha A_{\alpha 0}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{\alpha 0} & O \\ O & A_{\alpha 1} \end{bmatrix} \begin{bmatrix} I & A_{\alpha 0}^{-1} X_\alpha \\ O & I \end{bmatrix}.$$

Proposition 2.6. *The determinant of $A$ can be recursively computed from the RF tree of $A$, as follows: $det(A_\alpha) = det(A_{\alpha 0}) det(A_{\alpha 1})$.*

Proposition 2.7. *The inverse of $A$ can be recursively computed from the RF tree of $A$, as follows:*

$$A_\alpha^{-1} = \begin{bmatrix} I & -A_{\alpha 0}^{-1} X_\alpha \\ O & I \end{bmatrix} \begin{bmatrix} A_{\alpha 0}^{-1} & O \\ O & A_{\alpha 1}^{-1} \end{bmatrix} \begin{bmatrix} I & O \\ -Y_\alpha A_{\alpha 0}^{-1} & I \end{bmatrix}.$$

Note: throughout the rest of this paper, we will deal only with RF trees. Thus we will hereafter simply call an RF tree an *RF*, and we will call the RF tree problem simply the RF problem unless otherwise indicated.

## 3. OUR PARALLEL ALGORITHM FOR COMPUTING AN RF OF A MATRIX

In this section, we describe how to compute an RF for SPD matrices. We start with some definitions. Fix an SPD $n \times n$ matrix $A$, where $n$ is a power of 2, with integer entries of magnitude $\leq 2^{\beta}$, where $\beta \leq n^{O(1)}$.

### 3.1. Random Choice of Modulus

PROPOSITION 3.1. *(See Schwartz [80] and Zippel [84]) Let p be a prime number selected at random from the interval* $[2(n\|A\|_{\infty})^{c_0}/n, 2(n\|A\|_{\infty})^{c_0}]$*, for any* $c_0 > 2$*. If* $det(A) \neq 0$*, then* $0 \not\equiv det(A) \pmod p$ *with probability* $\geq 1 - \Omega(n \log(n\|A\|_{\infty})/(n\|A\|_{\infty})^{c_0})$ $\geq 1 - 1/n^{\Omega(1)}$.

### 3.2. Multipliers for the RF

We now define multipliers $m_{\alpha}$ for the matrices $A_{\alpha}$. Define $m_{<>} = 1$ for $<>=$ the empty string, and for any binary string $\alpha$, let $m_{\alpha 0} = m_{\alpha}$ and let $m_{\alpha 1} = m_{\alpha} det(A_{\alpha 0})$. These multipliers are defined similarly to the multipliers defined in Subsection 2.3, except that they are subscripted by binary strings rather than integers. For example, for $\alpha = 1^d$, $m_{\alpha}$ is identical to the multiplier $m_d$ defined in Subsection 2.3, since $A_{\alpha} = A_d$. Note that each $A_{\alpha 0}$ is a submatrix of $A_{\alpha}$, so the multiplier $m_{\alpha 0}$ for $A_{\alpha 0}$ is defined to be the same as the multiplier $m_{\alpha}$ for $A_{\alpha}$. We have defined $A_{\alpha}$ to be the upper left $n/2^{|\alpha|} \times n/2^{|\alpha|}$ submatrix of the matrix derived from $A$ by $k$ stages of successive block Gaussian elimination of the variables of the blocks $A_{\alpha_i 0}$, for $i = 1, \ldots, k$. Note that $A_{\alpha_i 0}$ is of size $n/2^{|\alpha_i|+1} \times n/2^{|\alpha_i|+1}$. The variables eliminated (by this Gaussian elimination) after stage $\alpha$ are thus the first $e_{\alpha} = \sum_{i=1}^{k} n/2^{|\alpha_i|+1}$ variables of the original linear system of $A$ corresponding to the upper left $e_{\alpha} \times e_{\alpha}$ submatrix $W$ of $A$.

We now show that $m_{\alpha} A_{\alpha}$ is an integer matrix if $A$ is an integer matrix. Using an inductive argument similar to one used in the proof of Lemma 2.1. To see this, let $k$ be the number of ones in $\alpha$. Let $\alpha_1 1, \ldots, \alpha_k 1$ be the prefixes of $\alpha$ ending with 1 in order of increasing length. Using a similar argument as Lemma 2.1 (but instead with $W$ being the upper left submatrix of $A$), we consider the partition

$$A = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix}.$$

Since $A$ is assumed to be an integer matrix, all its submatrices including $W, X, Y, Z$ are integer matrices. By well known properties of block Gaussian elimination, we get the same matrix as $A_{\alpha}$ by block Gaussian elimination from $A$, where we eliminate all these $e_{\alpha}$ variables in a single stage rather than $k$ stages. Hence $A_{\alpha}$ is also equal to the upper left $n/2^{|\alpha|} \times n/2^{|\alpha|}$ submatrix of the Schur complement $Z - YW^{-1}X$ derived by Gaussian elimination of the variables of submatrix $W$ from $A$. Note that $A_{\alpha}$ depends only on the elements of the upper left $(e_{\alpha} + n/2^{|\alpha|}) \times (e_{\alpha} + n/2^{|\alpha|})$

submatrix of $A$, and not on any other elements of $A$ outside on this submatrix. Thus the recursive formulae for the LU factorization of the RF factorization given in Proposition 2.5 can be applied to give a partial LU factorization of $W$ into a product of block triangular matrices with the matrices $A_{\alpha_1 0}, \ldots, A_{\alpha_k 0}$ on the diagonals. Since the determinant of a triangular matrix is obtained by multiplying all the elements of its principal diagonal, we have that $det(W) = \prod_{i=1}^{k} det(A_{\alpha_i 0}) = m_\alpha$. Since $det(W)W^{-1}$ is the adjoint matrix of integer matrix $W$, it is also an integer matrix. Thus $m_\alpha A_\alpha = det(W)A_\alpha = det(W)Z - Y(det(W)W^{-1})X$ is an integer matrix.

By Proposition 2.4, $\|A_{\alpha 1}\|, \|A_{\alpha 0}\| \leq \|A_\alpha\|$. Hence by Proposition 2.1, it follows that there is a constant $\hat{c}$ such that $|m_\alpha| \leq \prod_{i=1}^{d} 2^{\hat{c}(1/2^i)\pi} \leq 2^{(\sum_{i=1}^{d} 1/2^i)\hat{c}n(\beta+\log n)} \leq 2^{\hat{c}n(\beta+\log n)}$ where $\alpha$ is of length $d$ and $\pi = n(\beta + \log n)$. We have shown

LEMMA 3.1. *Assuming $A_{<>} = A$ is an integer matrix, and the entries of $A$ have magnitude $\leq 2^\beta$, then for each $d \geq 0$, and for each binary string $\alpha$ of length $d$, $m_\alpha A_\alpha$ is an integer matrix and $|m_\alpha| \leq 2^{\hat{c}\pi}$ where $\pi = n(\beta + \log n)$ and $\hat{c}$ is a constant.*

Now we will define an ordering to evaluate the multipliers $m_\alpha$ fast in parallel. Let $G$ be the directed acyclic digraph derived from the RF tree by simply (i) renaming each node $A_\alpha$ by its index $\alpha$, so the nodes of $G$ are strings $\{\alpha \in \{0,1\}^d | d \leq \log n\}$, (ii) including all tree edges reverse directed from the children to parents, and (iii) adding directed edges from each node of form $\alpha 1$ to its sibling of form $\alpha 0$. Note that the longest directed path in $G$ has $1 + 2\log n$ nodes. Hence the nodes of $G$ can be partitioned into $1 + 2$(depth of the RF tree) $= 1 + 2\log n$ blocks $\Pi_0, \ldots, \Pi_{2\log n}$ so that the evaluation of $m_\alpha$ is executed in parallel for all $\alpha \in \Pi_j$ in $1 + 2\log n$ sequential stages for $j = 0, \ldots, 2\log n$. In particular, we can let $\Pi_0 = \{<>\}$ contain just the empty string $<>$ corresponding to the root, and for each $d = 1, \ldots, \log n$ let $\Pi_{2d-1} = \{\alpha 0 | \alpha \in \{0,1\}^{d-1}\}$, and $\Pi_{2d} = \{\alpha 1 | \alpha \in \{0,1\}^{d-1}\}$. This ordering of the 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 $m_{\alpha 0}$).

### 3.3. RF Algorithm

We now describe our algorithm:

**Algorithm RF**

**INPUT:** an $n \times n$ integer nonsingular SPD matrix $A$, with integer entries of magnitude $\leq 2^\beta$, where $\beta \leq n^{O(1)}$, and w.l.o.g., we assume $n$ is a power of 2.

Let $\pi = n(\beta + \log n)$. All of our computations will use bit precision at most $O(\pi)$.

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

2. Let $\bar{A} = A + \kappa I$, where $\kappa = p(n\|A\|_\infty)^{c_1}$ is an integer and $c_1$ is a sufficiently large positive integer constant.

3. Apply the Newton Iteration of Subsection 3.5 using bit precision $O(\pi)$, to compute an approximate RF of $\bar{A}$ within accuracy $2^{-c\pi}$, for a sufficiently large positive constant $c$.

4. Using the RF tree for $\bar{A}$, within accuracy $2^{-c\pi}$, compute by Proposition 2.5 an approximate $LU$ factorization of the rational matrices $\bar{A}_\alpha$ in the RF of $\bar{A}$ and from these compute by Proposition 2.6 a rational approximation to $det(\bar{A}_\alpha)$ up to accuracy within $2^{-c\pi}$.

5. Construct the ordered partition $\Pi_0, \ldots, \Pi_{2\log n}$ of the strings $\{\alpha \in \{0,1\}^d | d \leq \log n\}$ defined above.

**For** each $j = 0, \ldots, 2\log n$ **do** (in sequence)

**For** each binary string $\alpha \in \Pi_j$ **do** in parallel

Recursively approximately compute, up to accuracy within $2^{-c\pi}$, multipliers $\bar{m}_\alpha$ for the matrices $\bar{A}_\alpha$ from the approximation to their determinants, as follows:

**If** $\alpha =<>$ **then** let $\bar{m}_{<>} = 1$.

**Else if** $|\alpha| > 0$ **then do**

(i) Let $\alpha = \alpha' b$, where $b$ is the last bit of $\alpha$.

(ii) **If** $b = 0$ **then do** let $\bar{m}_\alpha = \bar{m}_{\alpha'}$

**Else if** $b = 1$ **then do** let $\bar{m}_\alpha =$ the product of $\bar{m}_{\alpha'}$ and the approximation of $det(\bar{A}_{\alpha' 0})$.

6. **For** each $d = 0, \ldots, \log n$ and $\alpha \in \{0,1\}^d$, in parallel **do**:

To compute the integer matrix $\bar{m}_\alpha \bar{A}_\alpha$ exactly, round to the nearest integer the product of $\bar{m}_\alpha$ times this rational approximation to $\bar{A}_\alpha$. This gives the exact RF of $\bar{A}$.

Let $A \pmod{p}$ denote the matrix derived from a rational matrix $A$ by applying the standard homomorphism from the rationals $\mathbf{Q}$ to the finite field $\mathbf{Z}_p$.

7. Applying the homomorphism from $\mathbf{Q}$ to $\mathbf{Z}_p$, reduce $\pmod{p}$ the exact RF of $\bar{A}$, yielding the RF $\pmod{p}$ of $A$. (Note that we can apply this homomorphism from $\mathbf{Q}$ to $\mathbf{Z}_p$, since we have assumed a model of computation with unit cost division over a finite field.) Also, compute $det(A_\alpha) \pmod{p}$ by Proposition 2.6 and compute $(A_\alpha)^{-1} \pmod{p}$ by Proposition 2.7 in parallel for each $\alpha \in \{0,1\}^d$ for $d = \log n, \log n - 1, \ldots, 0$.

8. **If** $det(A_\alpha) \equiv 0 \pmod{p}$ for any $\alpha$ in the previous steps, **then** go to step 1 and choose another $p$.

9. Apply Newton-Hensel lifting of Subsection 3.4 to compute **for** $i = 0, \ldots, k^* = \lceil \log(c\pi / \log p) \rceil$ the RF $\pmod{p^{2^i}}$ of $A$. This gives us the RF $\pmod{p^{2^{k^*}}}$ of $A$.

10. **For** each $\alpha \in \{0,1\}^{\log n}$ in parallel **do**

Set $det(A_\alpha) \pmod{p^{2^{k^*}}}$ to be the scalar entry in $A_\alpha \pmod{p^{2^{k^*}}}$.

**Comment:** Here we consider each leaf of the RF tree. In this case, $A_\alpha \pmod{p^{2^{k^*}}}$ is a $1 \times 1$ matrix, whose scalar entry is its determinant $\pmod{p^{2^{k^*}}}$.

11. **For** each $d = \log n - 1, \log n - 2, \ldots, 0$ **do** (in sequence)

**for** all $\alpha \in \{0,1\}^d$ in parallel **do** compute $det(A_\alpha) \pmod{p^{2^{k^*}}}$ by the recursive formulae $det(A_\alpha) = det(A_{\alpha 0}) det(A_{\alpha 1}) \pmod{p^{2^{k^*}}}$ implied by Proposition 2.6.

12. **For** each $j = 0, \ldots, 2\log n$ **do** (in sequence)

**for** each binary string $\alpha \in \Pi_j$ **do** in parallel

Recursively compute the multipliers $m_\alpha \pmod{p^{2^{k^*}}}$ for the matrices $A_\alpha$ from their determinants $\pmod{p^{2^{k^*}}}$, as follows:

**If** $\alpha = <>$ **then** let $m_{<>} = 1$.

**Else if** $|\alpha| > 0$ **then do**

(i) Let $\alpha = \alpha' b$, where $b$ is the last bit of $\alpha$.

(ii) **If** $b = 0$ **then do** let $m_\alpha = m_{\alpha'} \pmod{p^{2^{k^*}}}$

**Else if** $b = 1$ **then do** let $m_\alpha = m_{\alpha'} det(A_{\alpha' 0}) \pmod{p^{2^{k^*}}}$.

13. **for** each $d = 0, \ldots, \log n$ and $\alpha \in \{0,1\}^d$, in parallel **do**:

Using the RF $\pmod{p^{2^{k^*}}}$ of $A$ computed above, represent $A_\alpha$ exactly as the rational fraction of integer matrices: $m_\alpha A_\alpha \pmod{p^{2^{k^*}}}$ divided by $m_\alpha \pmod{p^{2^{k^*}}}$.

14. **OUTPUT** The exact RF of $A$.

After we define Newton-Hensel lifting (Subsection 3.4) and Newton iterations (Subsection 3.5), we complete the proof and analysis of the RF algorithm in Subsection 3.6. There we show that if the input to our RF algorithm is assumed (as stated in the algorithm) to be a nonsingular matrix $A$, then with high likelihood $\geq 1 - 1/n^{\Omega(1)}$, the execution will not loop from step 8 back to step 1, so the stated time bounds thus hold with high likelihood $\geq 1 - 1/n^{\Omega(1)}$. The RF algorithm is thus always correct, but theoretically may loop forever, with probability 0. (Alternatively, we may alter the RF algorithm to not necessarily assume the input is nonsingular, and simply terminate after only one such loop from step 8. Then if the input matrix $A$ is nonsingular, this termination will occur with low likelihood $\leq 1/n^{\Omega(1)}$. So on termination, we can conclude, with high likelihood $\geq 1 - 1/n^{\Omega(1)}$, that the input matrix $A$ is nonsingular, noting that we may now be in error with low likelihood $\leq 1/n^{\Omega(1)}$.)

### 3.4. Newton-Hensel Lifting

**Lifting of a Matrix.** Fix a nonsingular $n \times n$ matrix $A$ and a prime $p$. In this section we assume that we have already computed $A^{-1} \pmod p$. Zassenhaus extended Hensel's lifting to exponentially increase the modulus. The resulting Newton-Hensel lifting is the following algorithm:

**INPUT:** a positive number $k$, and $n \times n$ matrices $A$ and $A^{-1} \pmod p$.

1. $S^{(0)} = A^{-1} \pmod p$
2. **For** $i = 1, \ldots, k$ **do** $S^{(i)} = S^{(i-1)}(2I - AS^{(i-1)}) \pmod{p)^{2^i}}$.

Moenck and Carter [57] show

PROPOSITION 3.2. $S^{(k)} = A^{-1} \pmod{p^{2^k}}$.

**Newton-Hensel Lifting of Modular Recursive Factorizations.** Recall that the RF of $A$ is a full binary tree of depth $\log n$. Each internal node is an $n/2^d \times$

$n/2^d$ matrix $A_\alpha$. The RF of $A_\alpha$ is defined in terms of the RF of two $n/2^{d+1} \times n/2^{d+1}$ matrices, namely $A_{\alpha 1}$ and $A_{\alpha 0}$, and furthermore $A_{\alpha 1}$ is defined in terms of submatrices of $A_\alpha$ and the inverse of $A_{\alpha 0}$.

Let an *RF* $\pmod p$ of an $n \times n$ matrix $A$ be an RF of matrix $A$ where each element is taken $\pmod p$. This will also be called a *modular RF*. In this section we assume that we are given an RF $\pmod p$ of matrix $A$ . We shall compute the RF $\pmod{p^{2^k}}$ of $A$.

Recall that $M(n) = n^\omega$ is the minimum number of PRAM processors necessary to multiply two $n \times n$ matrices in $O(\log n)$ parallel steps, where we assume $\omega > 2$.

Proposition 3.3. $\sum_{d=0}^{\log n} 2^d O(M(n/2^d)) \leq O(M(n))$.

Note that the obvious way to compute the RF $\pmod{p^{2^k}}$ of $A$ is by proceeding level by level through the RF in $\log n$ stages, each requiring $O(k \log n)$ time and $O(M(n)) \geq 2^d O(M(n/2^d))$ processors for the required $k$ matrix products for the nodes of depth $d$. This requires total parallel time $O(k \log^2 n)$ using $O(M(n))$ processors. However, we can do better with the following:

**Newton-Hensel Lifting Algorithm for an RF**

**INPUT:** RF $\pmod p$ of $A$ and number $k \geq 1$.

**For** each matrix $A_\alpha$ in the RF **do in parallel:**

1. INITIALIZATION:

(i) Let $A_\alpha^{(0)} \equiv A_\alpha \pmod p$

(ii) Compute $S_\alpha^{(0)} \equiv A_\alpha^{-1} \pmod p$ from the RF $\pmod p$ of $A$ by applying Proposition 2.7.

2. **For each** $i = 1, \ldots, k$ **do**

**Loop Invariant:** We have just computed
$A_\alpha^{(i-1)} \equiv A_\alpha \pmod{p^{2^{i-1}}}$ and $S_\alpha^{(i-1)} \equiv A_\alpha^{-1} \pmod{p^{2^{i-1}}}$.
Let $S_\alpha^{(i)} = S_\alpha^{(i-1)} \, (2I - A_\alpha^{(i-1)} S_\alpha^{(i-1)})$.
For $d < \log n$, let

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

where $A_{\alpha 0}^{(i)}, X_\alpha^{(i)}, Y_\alpha^{(i)}, Z_\alpha^{(i)}$ are matrices each of size $n/2^{d+1} \times n/2^{d+1}$ and let $A_{\alpha 1}^{(i)} = Z_\alpha^{(i)} - Y_\alpha^{(i)} S_{\alpha 0}^{(i)} X_\alpha^{(i)}$.

**OUTPUT:** RF $\pmod{p^{2^k}}$ of $A$ given by the $\{A_\alpha^{(k)}\}$.

This algorithm, as stated, requires parallel time $O(k \log^2 n)$ using $O(M(n))$ processors. However, we can use the technique of *stream contraction* (see Pan and Reif [72]) to decrease the time to $O((k + \log n) \log n)$ time without a processor penalty. The idea is to note that for each $d, 0 \leq d < \log n$, and each $\alpha \in \{0,1\}^d$ defining an internal node $A_\alpha$ at depth $d$, the following hold:

1. The matrices $A_{\alpha 0} \pmod{p^{2^{i-1}}}$ and $A_{\alpha 1} \pmod{p^{2^{i-1}}}$ are defined in terms of submatrices of $A_\alpha \pmod{p^{2^{i-1}}}$.

2. The computation $S_\alpha^{(i)} \equiv A_\alpha^{-1} \pmod{p^{2^i}}$ depends on the previous computations $S_{\alpha 1}^{(i-1)} \equiv A_{\alpha 1}^{-1} \pmod{p^{2^{i-1}}}$ and $S_{\alpha 0}^{(i-1)} \equiv A_{\alpha 0}^{-1} \pmod{p^{2^{i-1}}}$.

This implies that we can pipeline the computation, using stream contraction, as follows: As our basis step $t = 0$ we compute $S_\alpha^{(0)}$ for all $d = 0, \dots \log n$ and each $\alpha \in \{0,1\}^d$. We assume for our induction hypothesis that at time $t, 0 \le t < k + \log n$ we have computed each $S_\alpha^{(i)}$ for all $i, d, \alpha$ where $0 \le d \le \log n$, $0 \le i \le t - d$ and $\alpha \in \{0,1\}^d$. Then we apply the RF formula and compute one further product at each node of the recursion tree to compute by time $t+1$ each $S_\alpha^{(i+1)}$ for all $i, d, \alpha$ where $0 \le d \le \log n, i = t + 1 - d$ and $\alpha \in \{0,1\}^d$. Summarizing, we have (using a small constant factor slowdown to reduce the processor bounds from $O(M(n))$ to $M(n)$):

LEMMA 3.2. *Given an RF* $\pmod p$ *of an* $n \times n$ *matrix* $A$, *we can compute an RF* $\pmod{p^{2^k}}$ *of* $A$ *in parallel time* $O((k + \log n)\log n)$ *using* $M(n)$ *processors.*

### 3.5. Newton Iterations for Approximation of an RF of Diagonally Dominant Matrices

**$\epsilon$-Diagonally Dominant Matrices.** Let $A$ be an $n \times n$ symmetric matrix where $A = [a_{ij}]$. We define $A$ to be *$\epsilon$-DD* for some $\epsilon > 0$ if for all $i, 1 \le i \le n$, $\epsilon |a_{ii}| > \sum_{j=1, j \ne i}^n |a_{ij}|$. $A$ is *DD* if $A$ is $\epsilon$-DD for $0 < \epsilon < 1$. Let $D(A) = diag(a_{11}, a_{22}, \dots, a_{nn})$ denote the $n \times n$ diagonal matrix with the diagonal entries $a_{11}, a_{22}, \dots, a_{nn}$. Then $D(A)^{-1} = diag(1/a_{11}, 1/a_{22}, \dots, 1/a_{nn})$ is the $n \times n$ diagonal matrix with the diagonal entries $1/a_{11}, 1/a_{22}, \dots, 1/a_{nn}$. Since $\|I - D(A)^{-1}A\|_\infty \le \max_i \frac{\sum_{j=1, j\ne i}^n |a_{ij}|}{|a_{ii}|} < \epsilon$, it follows:

PROPOSITION 3.4. *If* $A$ *is $\epsilon$-DD then* $\|I - D(A)^{-1}A\|_\infty < \epsilon$.

We will use $B^{(0)} = D(A)^{-1}$ as the initial approximation to the inverse of $A$ in our Newton iterations.

Let $mindiag(A) = \min_i |a_{i,i}|$.

**Note:** Lower bounds on $mindiag(A)$ will later provide us with upper bounds on the norm of an approximate inverse of $A$, which will be used in the proof of our RF algorithm.

PROPOSITION 3.5. *If* $A = (a_{i,j})$ *is a symmetric $\epsilon$-DD matrix of size* $n \times n$, *then one stage of Gaussian elimination results in a matrix* $A'$ *that is symmetric,* $\epsilon/(1-\epsilon)$*-DD and with* $mindiag(A') \ge (1-\epsilon) mindiag(A)$.

**Proof:** Let the row and column indices of $A$ be in $\{1, \dots, n\}$. Let $A' = (a'_{i,j})$ be the symmetric matrix of size $n-1 \times n-1$ matrix derived from $A$ by one stage of Gaussian elimination, say by elimination of the first column and row of $A$. For notational ease, we allow the row and columns indices of $A'$ to be in $\{2, \dots, n\}$. For all $i, 2 \le i \le n$, Gaussian elimination gives $a'_{i,j} = a_{i,j} - a_{1,j}a_{i,1}/a_{1,1}$. Since $A$ is symmetric and $\epsilon$-DD, $|a'_{i,j}| \le |a_{i,j}| + |a_{1,j}||a_{i,1}|/|a_{1,1}| \le |a_{i,j}| + \epsilon |a_{i,1}|$. Also $|a_{i,i}| \le |a'_{i,i}| + |a_{1,i}|^2/|a_{1,1}| \le |a'_{i,i}| + \epsilon^2 |a_{i,i}|$, so $|a_{i,i}| \le |a'_{i,i}|/(1-\epsilon^2)$. This also implies that $|a'_{i,i}| \ge |a_{i,i}|(1-\epsilon^2)$ so $mindiag(A') \ge mindiag(A)(1-\epsilon^2) \ge mindiag(A)(1-\epsilon)$.

Since $A$ is $\epsilon$-DD we have

$$\sum_{j=2,j\neq i}^{n} |a'_{i,j}| = \sum_{j=2,j\neq i}^{n} |a_{i,j} - a_{1,j}a_{i,1}/a_{1,1}| \leq \epsilon(|a_{i,i}| + |a_{i,1}|) \leq \epsilon(|a_{i,i}| + \epsilon|a_{i,i}|)$$

$$= \epsilon(1+\epsilon)|a_{i,i}| \leq \epsilon(1+\epsilon)|a'_{i,i}|/(1-\epsilon^2) = \epsilon|a'_{i,i}|/(1-\epsilon).$$

It is also easy to verify that $A'$ is symmetric. ∎

It follows by induction that if $A$ is symmetric and $\epsilon$-DD, then $k$ Gaussian elimination stages results in a matrix that is symmetric, $\epsilon/(1-k\epsilon)$-DD, and where the *mindiag* of that resulting matrix is $\geq$ *mindiag*$(A)(1-k\epsilon)$. Since block Gaussian elimination can be done by at most $n$ repeated Gaussian elimination stages,

PROPOSITION 3.6. *if $A$ is $\epsilon$-DD, then all the matrices $A_\alpha$ in the RF of $A$ are at least $\epsilon/(1-n\epsilon)$-DD and have mindiag$(A_\alpha) \geq$ mindiag$(A)(1-n\epsilon)$.*

In the following, we will consider an $n \times n$, DD, symmetric $A = (a_{i,j})$. Define a quantity to be *very small* with respect to $A$ if it is of the form $(n\|A\|)^{-c'}$, for some constant $c' > 1$. Note that the sum of two very small quantities with respect to the same matrix is also very small. (Note that this definition allows $\epsilon = (n\|A\|)^{-c'}$ and $\epsilon' = (n\|A\|)^{-c''}$ to both be very small quantities, which may have distinct $c', c''$. Also note, this definition of *very small* simplifies our proofs of rapid convergence; we can alter this definition of very small so that $\epsilon$ is a larger quantity, but this decreases the overall bit complexity of our algorithm only by a constant factor.) Let a nonsingular $A$ be *strongly well conditioned* if $cond_2(A) \leq 1 + \epsilon$ for a very small $\epsilon$ (with respect to $A$), and let a nonsingular $A$ be *strongly DD* if $A$ is $\epsilon$-DD for a very small $\epsilon$. Let $A \approx \tilde{A}$ if $\|A - \tilde{A}\|$ is very small with respect to $A$.

PROPOSITION 3.7. *Suppose $A$ is symmetric and strongly DD. Then:*

1.*All the matrices $A_\alpha$ in the RF of $A$ are strongly DD and have mindiag$(A_\alpha) \geq$ mindiag$(A) - o(1)$.*

2.*$A^{-1} \approx D(A)^{-1}$ and is also strongly DD.*

3.*If $\|I - BA\|_\infty$ is very small (with respect to $A$), then $\|B\|_\infty \leq (1/(mindiag(A)) + o(1)$.*

**Proof:** The first statement is implied by Proposition 3.6, and the observation that if $A_\alpha$ is strongly DD then both $A_{\alpha 0}$ and $A_{\alpha 1}$ are also strongly DD. The second statement requires a detailed proof. Since $A$ is strongly DD, then $A$ is $\epsilon$-DD for a very small $\epsilon = (n\|A\|)^{-c'}$, for some constant $c' > 2$. Let $b$ be the $i$th column of $A^{-1}$. Let $b_*$ the maximum of any $|b_j|$, for $j \neq i$. For each $j \neq i$, since $(Ab)_j = 0$ and $A$ is symmetric and $\epsilon$-DD, it follows that $|b_j| \leq \epsilon b_* + |b_i||a_{i,j}|/|a_{j,j}| \leq \epsilon(b_* + |b_i|)$, since $|a_{i,j}|/|a_{j,j}| \leq \epsilon$. Thus $b_* = \max_{j\neq i}|b_j| \leq \epsilon(b_* + |b_i|)$, so $b_* \leq \epsilon|b_i|/(1-\epsilon)$. Also, since $(Ab)_i = 1$, it follows that $|b_i| \geq 1/|a_{i,i}| - \epsilon b_* \geq 1/|a_{i,i}| - \epsilon^2(|b_i|/(1-\epsilon))$, so $|b_i| \geq 1/(|a_{i,i}|(1-\epsilon^2/(1-\epsilon))) \geq (1-\epsilon)/(|a_{i,i}|(1-2\epsilon)) \approx 1/|a_{i,i}|$, since $\epsilon$ is very small (with respect to $A$). This implies that $b_* \leq \epsilon|b_i|/(1-\epsilon) \approx 0$, since $\epsilon$ and $\epsilon|b_i| \approx \epsilon/|a_{i,i}|$ are both very small (with respect to $A$). Thus we have shown that $A^{-1} \approx D(A)^{-1}$, so $A^{-1}$ is also strongly DD. The same proof implies that if $\|I - BA\|_\infty$ is very small (with respect to $A$), then $B \approx D(A)^{-1}$. So if $\|I - BA\|_\infty$ is very small (with respect to $A$), then since $\epsilon$ is very small (with respect to $A$), $\|B\|_\infty \approx \|D(A)^{-1}\|_\infty \leq 1/\min_i |a_{i,i}| \leq 1/mindiag(A)$, so $\|B\|_\infty \leq (1/mindiag(A)) + o(1)$, thus proving the third statement. ∎

$A$ is *a nearly identity multiple (NIM)* if $A \approx \kappa I$ for a scalar constant $\kappa$.

PROPOSITION 3.8. *Suppose $A$ is symmetric and NIM. Then:*

1.*$A$ is strongly DD and strongly well conditioned,*

2.*All the matrices $A_\alpha$ in the RF of $A$ are NIM and thus strongly DD and strongly well conditioned, and*

3.*$A^{-1} \approx$ is also NIM.*

**Proof:** $A$ is NIM so $A \approx \kappa I$, and by definition is strongly DD. Also $cond_2(A) = \|A\|_2 \|A^{-1}\|_2 \approx \|\kappa I\|_2 \|\kappa^{-1} I\|_2 = \kappa\kappa^{-1} = 1$, so $A$ is strongly well conditioned. The second statement is implied by Proposition 3.6, and the observation that if $A_\alpha$ is NIM then both $A_{\alpha 0}$ and $A_{\alpha 1}$ are also NIM. Also, since $A^{-1} \approx D(A)^{-1}$, we have that $A^{-1}$ is also NIM. ∎

**Approximate Inverse of DD Matrices.** Let $A$ be an $n \times n$ symmetric matrix. We will let $B^{(0)} = D(A)^{-1}$ be the initial approximation to the inverse of $A$ in our Newton iterations. The Newton iteration generates a sequence of matrices $B^{(1)}, B^{(2)}, \ldots$ where $B^{(k)} = B^{(k-1)}(2I - AB^{(k-1)})$ for $k > 0$ (see Ben-Israel [4], Ben-Israel and Cohen [5], and Hotelling [40, 41] for sequential Newton iteration and also see Pan and Reif [69, 71] for parallel applications). Since $I - B^{(k)}A = I - B^{(k-1)}(2I - AB^{(k-1)})A = (I - B^{(k-1)}A)^2$, it follows that:

PROPOSITION 3.9. *If $A$ is $\epsilon$-DD then $\|I - B^{(k)}A\|_\infty < \epsilon^{2^k}$.*

Thus $\|I - B^{(k)}A\|_\infty = \|(I - B^{(k-1)}A)^2\|_\infty < (\epsilon^{2^{k-1}})^2 = \epsilon^{2^k}$.

Let $k^+$ be the integer $\lceil log(\bar{c}\pi) \rceil$ where $\bar{c}$ is a positive constant. Note that since $\pi = n(\beta + \log n)$, it follows that $k^+ = \lceil log(\bar{c}\pi) \rceil \leq O(\log n)$, assuming that the bit precision of the entries of $A$ are $\beta \leq n^{O(1)}$.

So if $A$ is $\epsilon$-DD, for $\epsilon < 1/2$, then at most $k^+$ Newton iterations suffice to compute the approximate inverse $B^{(k^+)}$ with error $2^{-\bar{c}\pi}$.

**Newton Iterations within Bounded Bit Precision.** Fix a very small (with respect to $A$) $\epsilon = (n\|A\|)^{-c'} < 1/32$ for some positive constant $c' > 2$. We assume $A$ is NIM, and thus strongly DD, in particular is $\epsilon$-DD, for very small $\epsilon$.

For all $k \geq 0$ let $E_k(\epsilon) = (16\epsilon)^{2^k}/16$. By definition $E_0(\epsilon) = \epsilon$ and observe that $E_k(\epsilon)$ exactly satisfies the recurrence $E_k(\epsilon) = (4E_{k-1}(\epsilon))^2$ for all $k > 0$. Also since we assume $\epsilon < 1/32$, it follows $E_{k^+}(\epsilon) < (16\epsilon)^{2^{k^+}} \leq 2^{-2^{k^+}} \leq 2^{-\bar{c}\pi}$.

Next we consider the case of Newton iteration where the input matrix $A$ is not initially given with full accuracy, and instead we are provided on-line a sequence of approximates $\tilde{A}^{(0)}, \tilde{A}^{(1)}, \ldots$ where $\|A - \tilde{A}^{(k)}\|_\infty \leq E_k(\epsilon)$, and each $\tilde{A}^{(k)}$ has bit precision $\leq O(\pi)$. Let $\tilde{B}^{(0)} = D(\tilde{A}^{(0)})^{-1} = diag(1/\tilde{a}_{11}, 1/\tilde{a}_{22}, \ldots, 1/\tilde{a}_{nn})$ where $\tilde{A}^{(0)} = [\tilde{a}_{ij}]$. If $\tilde{A}^{(0)}$ is $\epsilon$-DD then by Proposition 3.4, $\|I - \tilde{B}^{(0)}\tilde{A}^{(0)}\|_\infty < \epsilon$.

We will generate a sequence of matrices $\tilde{B}^{(0)}, \hat{B}^{(1)}, \tilde{B}^{(1)}, \ldots$ where $\hat{B}^{(k)} = \tilde{B}^{(k-1)}(2I - \tilde{A}^{(k)}\tilde{B}^{(k-1)})$ is the exact Newton iterate, and $\tilde{B}^{(k)}$ is an approximation to $\hat{B}^{(k)}$ obtained by rounding within bit precision $O(\pi)$, so that $\|I - \tilde{B}^{(k)}\tilde{A}^{(k)}\|_\infty \leq \|I - \hat{B}^{(k)}\tilde{A}^{(k)}\|_\infty + E_k(\epsilon)$. Since $A$ is assumed to be NIM, then by assumption on the close approximation of $A$ by the $\tilde{A}^{(k)}$, we have:

Proposition 3.10. *Each $\tilde{A}^{(k)}$ is also NIM, and thus strongly DD and strongly well conditioned.*

Finally, we give an inductive proof of quadratic convergence of the entire iteration sequence:

Proposition 3.11. *If $\|A\|_\infty \geq 1$, $A$ and $\tilde{A}^{(0)}$ are $\epsilon$-DD, and also $mindiag(A) \geq 1 - o(1)$, then we have $\|I - \tilde{B}^{(k)}\tilde{A}^{(k)}\|_\infty < E_k(\epsilon)$.*

**Proof:** We use a proof by induction on $k$. Note that the basis of the induction holds by assumption, since $\epsilon = E_0(\epsilon)$. Now for $k > 1$, assume the induction hypothesis: $\|I - \tilde{B}^{(k-1)}\tilde{A}^{(k-1)}\|_\infty < E_{k-1}(\epsilon)$. By definition of the Newton Iterations,

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

By assumption on the approximations to the $\tilde{A}^{(k)}$,

$$\|\tilde{A}^{(k)} - \tilde{A}^{(k-1)}\|_\infty \leq \|A - \tilde{A}^{(k)}\|_\infty + \|A - \tilde{A}^{(k-1)}\|_\infty \leq E_k(\epsilon) + E_{k-1}(\epsilon).$$

By Proposition 3.10, $\tilde{A}^{(k-1)}$ is strongly DD. By the induction hypothesis, $\|I - \tilde{B}^{(k-1)}\tilde{A}^{(k-1)}\|_\infty < E_{k-1}(\epsilon)$. Since we have assumed $mindiag(A) \geq 1 - o(1)$, Proposition 3.7 implies $\|\tilde{B}^{(k-1)}\|_\infty \leq 2$ for any sufficiently large $n$. Also, $I - \tilde{B}^{(k-1)}\tilde{A}^{(k)} = (I - \tilde{B}^{(k-1)}\tilde{A}^{(k-1)}) + \tilde{B}^{(k-1)}(\tilde{A}^{(k-1)} - \tilde{A}^{(k)})$, so we have: $\|I - \tilde{B}^{(k-1)}\tilde{A}^{(k)}\|_\infty \leq \|I - \tilde{B}^{(k-1)}\tilde{A}^{(k-1)}\|_\infty + \|\tilde{B}^{(k-1)}\|_\infty \cdot \|\tilde{A}^{(k-1)} - \tilde{A}^{(k)}\|_\infty$
$\leq E_{k-1}(\epsilon) + 2(E_k(\epsilon) + E_{k-1}(\epsilon)) \leq 4E_{k-1}(\epsilon) - 2E_k(\epsilon)$ since $4E_k(\epsilon) \leq E_{k-1}(\epsilon)$.

Thus we have: $\|I - \tilde{B}^{(k)}\tilde{A}^{(k)}\|_\infty \leq \|I - \hat{B}^{(k)}\tilde{A}^{(k)}\|_\infty + E_k(\epsilon) \leq \|(I - \tilde{B}^{(k-1)}\tilde{A}^{(k)})^2\|_\infty + E_k(\epsilon)$
$\leq \|(I - \tilde{B}^{(k-1)}\tilde{A}^{(k)})\|_\infty^2 + E_k(\epsilon) \leq (4E_{k-1}(\epsilon) - 2E_k(\epsilon))^2 \leq E_k(\epsilon)$, by definition of $E_k(\epsilon)$. ∎

By definition of $\tilde{A}^{(k)}$ and $\tilde{B}^{(k)}$, these approximate Newton iterations require $\log(E_k(\epsilon)) \leq O(\pi)$ bit precision.

Lemma 3.3. *If $A$ is NIM and thus strongly DD, then even with the above approximations to $A$, $k^+ = \lceil log(\bar{c}\pi) \rceil \leq O(\log n)$ Newton iterations suffice to compute, using bit precision $\leq O(\pi)$, the approximate inverse $\tilde{B}^{(k)}$ with error $2^{-\bar{c}\pi}$, for a positive constant $\bar{c}$.*

*Note: we can also show that Lemma 3.3 holds even with more moderate conditions on $A$ (for example where $A$ is well conditioned rather that strongly well conditioned), but the Lemma will suffice for the RF algorithm.*

**Non-Pipelined Newton Iterations for RF.** Here we assume that we are given a matrix $A$ which is NIM and with $mindiag(A) \geq 1$. We shall compute an approximation to the RF of $A$ within error $2^{-\Omega(\pi)}$, using bit precision $\leq O(\pi)$. Recall again that the RF of $A$ is a full binary tree of depth $\log n$. Each internal node is an $n/2^d \times n/2^d$ matrix $A_\alpha$. The RF of $A_\alpha$ is defined in terms of the RF of two $n/2^{d+1} \times n/2^{d+1}$ matrices, namely $A_{\alpha 1}$ and $A_{\alpha 0}$, and furthermore $A_{\alpha 1}$ is defined in terms of submatrices of $A_\alpha$ and the inverse of $A_{\alpha 0}$. Note that the simplest and most obvious way we might compute the RF of $A$ within error $2^{-\Omega(\pi)}$ is by proceeding up the recursion tree in $\log n$ stages. We first sketch this simple algorithm (we will give full details of a more efficient algorithm below). At each stage $d = 0, 1, \ldots, \log n$, we compute for all nodes of depth $d$ an approximation $\tilde{A}_\alpha$ of $A_\alpha$ within error $2^{-\Omega(\pi)}$. Then we compute $B_\alpha^{(k)}$, which is a $k$-th Newton iteration matrix approximating the inverse of matrix $A_\alpha$ up to error $\epsilon^{2^k}$, where $\epsilon = (n\|A\|)^{-c'}$, for some sufficiently large positive constant $c'$. Recalling that $k^+ = \lceil log(\bar{c}\pi) \rceil \leq O(\log n)$, we use $B_\alpha^{(k^+)}$ to approximate $A_\alpha^{-1}$ within error $2^{-\Omega(\pi)}$. Since there are only $\log n$ levels in the RF, and since $A$ is NIM and thus strongly DD and strongly well conditioned, the norm bounds given in Proposition 2.2 ensure that the error of the overall approximate computation is at most $2^{-\Omega(\pi)}$. At each stage we need to compute approximations to two recursive inverses. Each such stage requires $O(\log^2 n)$ time and $O(M(n)) \geq 2^d O(M(n/2^d))$ processors for the required $k$ repeated $n/2^d \times n/2^d$ matrix products for each of the $2^d$ nodes of depth $d$. The $\log n$ stages requires total parallel time $O(k \log^2 n) = O(log^3 n)$ using $\sum_{d=0}^{\log n} 2^d O(M(n/2^d)) \leq O(M(n))$ processors. However, this simple algorithm can be sped up using pipelining.

**Pipelined Newton Iterations for RF.** Next, we will improve on this $O(log^3 n)$ algorithm for the approximate RF, decreasing the time to $O(log^2 n)$ without an increase in processor bounds. As in Lemma 3.3, we consider the case of Newton iteration where the recursively defined matrices $A_\alpha$ are not initially given with full accuracy, and instead we are provided a sequence of improving approximations $\tilde{A}_\alpha^{(0)}, \tilde{A}_\alpha^{(1)}, \ldots$ where $\|A - \tilde{A}^{(k)}\|_\infty \leq E_k(\epsilon)$. In the special case $d = 0$, the approximation is exact, $\tilde{A}_{<>} = A$, where $<>$ is the empty string. Given these approximants to $A$, we will generate a sequence of matrices $\tilde{B}_\alpha^{(0)}, \tilde{B}_\alpha^{(1)}, \ldots$ where $\tilde{B}_\alpha^{(k)} = \tilde{B}_\alpha^{(k-1)}(2I - \tilde{A}_\alpha^{(k)} \tilde{B}_\alpha^{(k-1)})$. Thus, $\tilde{B}_\alpha^{(k)}$ will approximate the inverse of $\tilde{A}_\alpha^{(k)}$. We will then use Lemma 3.3 to show quadratic convergence.

**Newton Iteration Algorithm for an RF**
**INPUT:** A $\epsilon$-DD symmetric matrix $A$ of size $n \times n$ with $mindiag(A) \geq 1$.
**For each** $d, \alpha$ for $0 \leq d \leq \log n$ and $\alpha \in \{0,1\}^d$, do in parallel:

1. INITIALIZATION: Let $\tilde{A}_{<>}^{(0)} = A$, where $<>$ is the empty string.
2. Let $k^+ = log(\bar{c}\pi)$ for a positive constant $\bar{c}$.
3. Let $\tilde{B}_\alpha^{(0)} = diag(1/\tilde{a}_{11}, 1/\tilde{a}_{22}, \ldots, 1/\tilde{a}_{nn})$, where $\tilde{A}_\alpha^{(0)} = [\tilde{a}_{ij}]$.
4. **For each** $i = 1, \ldots, k^+$ **do**

**Loop Invariant:** We have just computed $\tilde{A}_\alpha^{(i-1)}$ which approximates $A_\alpha$ with error $\|\tilde{A}_\alpha^{(i-1)} - A_\alpha\|_\infty < E_{i-1}(\epsilon)$ for $\epsilon = (n\|A\|)^{-c'}$ with $c' > 2$. Also, we have just computed $\tilde{B}_\alpha^{(i-1)}$ which approximates $A_\alpha^{-1}$ with error specified by $\|I - \tilde{B}_\alpha^{(i-1)} A_\alpha\|_\infty < E_{i-1}(\epsilon)$.

(i) Let $\tilde{B}_\alpha^{(i)} = \tilde{B}_\alpha^{(i-1)}\ (2I - \tilde{A}_\alpha^{(i)} \tilde{B}_\alpha^{(i-1)})$.

(ii) For $d < \log n$, let

$$\tilde{A}_\alpha^{(i)} = \begin{bmatrix} \tilde{A}_{\alpha 0}^{(i)} & \tilde{X}_\alpha^{(i)} \\ \tilde{Y}_\alpha^{(i)} & \tilde{Z}_\alpha^{(i)} \end{bmatrix},$$

where $\tilde{A}_{\alpha 0}^{(i)}, \tilde{X}_\alpha^{(i)}, \tilde{Y}_\alpha^{(i)}, \tilde{Z}_\alpha^{(i)}$ are matrices each of size $n/2^{d+1} \times n/2^{d+1}$ and $\tilde{A}_{\alpha 1}^{(i)} = \tilde{Z}_\alpha^{(i)} - \tilde{Y}_\alpha^{(i)} \tilde{B}_{\alpha 0}^{(i)} \tilde{X}_\alpha^{(i)}$.

**OUTPUT:** The approximate RF $\{\tilde{A}_\alpha^{(k^+)}\}$, which approximates the RF of $A$ within 2-norm error $2^{-\Omega(\pi)}$.

**Bounding Error Accumulation in Recursions.** We now show that in the initial approximation $\{\tilde{A}_\alpha^{(0)}\}$ to the RF of $A$, the errors do not accumulate much on the recursions. Assuming $A$ is NIM and thus strongly well conditioned, by Proposition 3.7, all the matrices $A_\alpha$ in the RF of $A$ are NIM and thus strongly well conditioned. Since our initial approximations to the inverses of matrices occurring in the RF are themselves NIM and thus strongly well conditioned, it follows by Proposition 3.7, that our initial approximations to its inverse give an approximation to the RF, with very small (with respect to $A$) relative error.

LEMMA 3.4. *Fix a very small (with respect to A) $\epsilon = (n\|A\|)^{-c'}$ for some large enough positive constant $c'$. In the Newton Iteration Algorithm for the RF, at each depth $d = 1, \ldots, \log n$, and for each $\alpha \in \{0,1\}^d$, on the first stage of Newton iteration, the initial approximation of the RF matrices at depth d have very small (with respect to A) $\infty$-norm error $\epsilon$. In particular, $\|A_\alpha^{(0)} - A_\alpha\|_\infty \leq \epsilon$ and $\|I - B_\alpha^{(0)} A_\alpha\|_\infty \leq \epsilon$.*

LEMMA 3.5. *For each $d = 1, \ldots, \log n$, and each $\alpha \in \{0,1\}^d$, the $\infty$-norm error for the i-th iteration is $\|A_\alpha^{(i)} - A_\alpha\|_\infty \leq E_i(\epsilon)$ and $\|I - B_\alpha^{(i)} A_\alpha\|_\infty \leq E_i(\epsilon)$.*

**Proof:** Lemma 3.5 is proved by induction. We use Lemma 3.4 as a basis

for the induction. The inductive step follows directly from Lemma 3.3 applied to $\{A_\alpha^{(i)}\}$ and $\{\tilde{B}_\alpha^{(i)}\}$, which provides the required bounds on the errors of the $i$-th approximate RF $\{A_\alpha^{(i)}\}$. ∎

Note again that these approximate Newton iterations require $O(\pi)$ bit precision.

**Applying Stream Contraction to the RF Newton Iterations.** As in Subsection 3.4, we use the technique of *stream contraction* to decrease the time to $O(\log^2 n)$ time without a processor penalty. To simplify notation, let us define $COMP_\alpha^i$ to be the computation of $\tilde{A}_\alpha^{(i)}$ and $\tilde{B}_\alpha^{(i)}$, where this approximate computation is within error specified by Lemma 3.5. Note that for each $d, 0 \leq d < \log n$, and each $\alpha \in \{0,1\}^d$ defining an internal node $\tilde{A}_\alpha$ at depth $d$, the computation $COMP_\alpha^i$ depends only on the computations $COMP_{\alpha 1}^{i-1}$ and $COMP_{\alpha 0}^{i-1}$. Furthermore, the computation $COMP_{\alpha 1}^{i-1}$ is defined in terms of submatrices of $\tilde{A}_\alpha^{(i-1)}$ and of $\tilde{A}_{\alpha 0}^{(i-1)}$. In particular, we need only to apply Newton iteration one more time to these approximated matrices. This implies that we can pipeline the computation of $COMP_\alpha^i$, using stream contraction, just as we did in Subsection 3.4 for $S_\alpha^{(i)}$, to reduce the parallel time from $\Omega(\log^3 n)$ to $O(\log^2 n)$ without a processor increase.

We pipeline the computation as follows: As our basis step $t = 0$ we have done the computation $COMP^0_\alpha$ for all $d = 0, \ldots \log n$ and each $\alpha \in \{0,1\}^d$. We assume for our induction hypothesis that at time $t, 0 \leq t < k + \log n$ we have done the computation $COMP^i_\alpha$ for all $i, d, \alpha$ where $0 \leq d \leq \log n$, $0 \leq i \leq t - d$, and $\alpha \in \{0,1\}^d$. Then we apply the approximation RF formula and perform one more product at each node of the recursion tree to do computation $COMP^{i+1}_\alpha$ by time $t+1$ for all $i, d, \alpha$ where $0 \leq d \leq \log n$ and $i = t + 1 - d$ and $\alpha \in \{0,1\}^d$.

Summarizing, using a small constant factor slowdown to reduce the processor bounds from $O(M(n))$ to $M(n)$, we have

LEMMA 3.6. *Given an $n \times n$ matrix $A$, which is NIM and with $mindiag(A) \geq 1$, we can compute, using bit precision $\leq O(\pi)$, an approximation to the RF of $A$ with error $2^{-\Omega(\pi)}$ in parallel time $O(\log^2 n)$ using $M(n)$ processors.*

### 3.6. Analysis of the RF algorithm

THEOREM 3.1. *Our algorithm for computing the exact RF with high likelihood $\geq 1 - 1/n^{\Omega(1)}$, takes parallel time $O(\log^2 n)$ using $M(n)$ randomized processors, and bit precision $O(\pi)$, for $\pi = O(n(\beta + \log n))$, using a constant number of random variables ranging over a domain of size $(n\|A\|)^{O(1)}$.*

**Proof:** Fix as input an $n \times n$ integer nonsingular SPD matrix $A$, with integer entries of magnitude $\leq 2^\beta$, where $\beta \leq n^{O(1)}$. We can assume w.l.o.g. that $n$ is a power of 2. Since Proposition 2.1 and Lemma 3.1 bound $|det(A)| \leq 2^\pi$ and $|m_\alpha| \leq 2^{\hat{c}\pi}$, all rational quantities in the RF of $A$ can be expressed as a quotient of two integers of magnitude $\leq 2^{O(\pi)}$, and thus in bit precision $O(\pi)$, Hence all rational quantities in the RF of $A$ are P-rational as defined in the introductory Subsection 1.1. We will show that all our computations will use bit precision at most $O(\pi)$.

In step 1, we choose a random prime $p$ from the interval $2(n\|A\|_\infty)^{c_0}/n < p < 2(n\|A\|_\infty)^{c_0}$, for some $c_0 > 2$. By Proposition 3.1, since $det(A) \neq 0$ then $0 \not\equiv det(A) \pmod p$ with high likelihood $\geq 1 - 1/n^{\Omega(1)}$.

In step 2, we construct $\bar{A} = A + \kappa I$, where $\kappa = p(n\|A\|_\infty)^{c_1}$ is an integer for a sufficiently large positive constant $c_1$. Since $A$ is an integer matrix, so is $\bar{A}$. For some positive constant $\bar{c} > 1$, Proposition 2.1 bounds $|det(\bar{A})| \leq 2^{\bar{c}\pi}$ so all rational quantities in the RF of $\bar{A}$ are P-rational; they can be expressed as a quotient of two integers of magnitude $\leq 2^{O(\pi)}$, and thus in bit precision $O(\pi)$. $\bar{A}$ is NIM and so strongly DD and strongly well conditioned. Let $\bar{A}^{-1} \pmod p$ and $A^{-1} \pmod p$ denote the matrices derived from rational matrices $\bar{A}^{-1}, A^{-1}$ by applying the standard homomorphism from the rationals $\mathbf{Q}$ to the finite field $\mathbf{Z}_p$ (we will use this notation below, as well). Since $\bar{A} - A = \kappa I$ is divisible by $p$, $\bar{A} \pmod p \equiv A \pmod p$, and thus $(\bar{A})^{-1} \pmod p \equiv A^{-1} \pmod p$.

In step 3, we apply the Newton Iteration of Subsection 3.5 using bit precision $O(\pi)$, to get an approximate RF of $\bar{A}$ within accuracy $2^{-c\pi}$, for a sufficiently large positive constant $c$. Clearly $mindiag(\bar{A}) \geq 1$, so by Lemma 3.6, step 3 costs parallel time $O(\log^2 n)$ using $M(n)$ processors. This approximate RF gives a rational matrix $\bar{A}_\alpha$, for each $d = 0, \ldots, \log n$ and $\alpha \in \{0,1\}^d$, a rational approximation of $(\bar{A}_\alpha)^{-1}$

and a rational approximation of the $LU$ factorization of $\bar{A}_\alpha$ to accuracy $2^{-c\pi}$ (using bit precision $O(\pi)$).

Applying Proposition 2.6, which states DETERMINANT has an efficient $O(\log n)$ time parallel reduction to RF using the recursive formulae, we can compute a rational approximation to $det(\bar{A}_\alpha)$ up to accuracy within $2^{-c\pi}$ (using bit precision $O(\pi)$) from the approximate $LU$ factorization of $\bar{A}_\alpha$. Thus step 4 is executed in time $O(\log n)$ using $n$ processors.

Recursive computation of the integer multipliers in step 5 and the multiplication and round off operations in step 6 clearly cost at most parallel time $O(\log^2 n)$ using $M(n)$ processors. The correctness of step 5 follows from the formulae derived in Subsection 3.2.

In step 6 we compute the integer matrix $\bar{m}_\alpha \bar{A}_\alpha$ exactly, rounding to the nearest integer the product of $\bar{m}_\alpha$ times the rational approximation to $\bar{A}_\alpha$. This requires bit precision $O(\pi)$. By Proposition 2.1 and Lemma 3.1, $|det(\bar{A}_\alpha)| \leq 2^{\hat{c}\bar{c}\pi}$, and $|\bar{m}_\alpha| \leq 2^{\hat{c}\bar{c}\pi}$. Each entry in the approximate $LU$ factorization can be in error by a factor of at most $2^{-c\pi}n^2$, so the error of approximation of each $A_\alpha$ is at most a factor $O(n)$ more, namely $O(2^{-c\pi}n^3)$. Since $\bar{m}_\alpha$ is computed from the product of at most $\log n$ determinants, the error of approximation of each $\bar{m}_\alpha$ is at most a factor $O(\log n)$ more than of the error of approximation of the determinants, namely $O(2^{-c\pi}n^3 \log n)$. Note that by Proposition 2.4, $\|\bar{A}_\alpha\|_\infty \leq n2^{\bar{c}\pi}$, so the total error in the approximation to $\bar{m}_\alpha \bar{A}_\alpha$ is at most $2^{2\hat{c}\bar{c}\pi}O(2^{-c\pi}n^3 \log n)\|\bar{A}_\alpha\|_\infty \leq$ $2^{2\hat{c}\bar{c}\pi}O(2^{(\bar{c}-c)\pi}n^4 \log n) \leq 2^{(2\hat{c}\bar{c}+\bar{c}-c)\pi+4\log n+\log\log n+O(1)} < 1/2$, with choice of a sufficiently large constant $c > 2\hat{c}\bar{c} + \bar{c} + O(1)$. We now have the exact RF of $\bar{A}$.

In step 7, as in previous efficient parallel algorithms [62], [63] for the exact inversion of integer matrices, we apply the standard homomorphism to make a conversion from rational numbers to integers $\pmod p$, and we have assumed an arithmetic model where finite field arithmetic has unit cost. Recall $\bar{A} \pmod p \equiv A \pmod p$, so the RF $\pmod p$ of $A$ is identical to the RF $\pmod p$ of $\bar{A}$. Hence for each $\alpha$, $\bar{A}_\alpha = A_\alpha \pmod p$, and we can apply Propositions 2.6 and 2.7 to compute $det(A_\alpha) \pmod p$ and $(A_\alpha)^{-1} \pmod p$ in parallel for each $\alpha \in \{0,1\}^d$ for $d = \log n, \log n - 1, \ldots, 0$. This requires bit precision $O(\pi)$.

In step 8, we go to step 1 and choose another $p$ if $det(A_\alpha) \equiv 0 \pmod p$ for any $d, \alpha$. Note since $\bar{A}$ is strongly DD, then by Proposition 3.6, each $\bar{A}_\alpha$ is also strongly DD and so nonsingular, so always $det(\bar{A}_\alpha) \neq 0$. By Proposition 3.1, since $det(\bar{A}_\alpha) \neq 0$, then $0 \not\equiv det(\bar{A}_\alpha) \pmod p$ with high likelihood $\geq 1 - 1/n^{\Omega(1)}$. Recall $\bar{A}_\alpha \pmod p = A_\alpha \pmod p$. So with low likelihood $\leq 1/n^{\Omega(1)}$, $det(\bar{A}_\alpha) \equiv 0 \pmod p$, and hence with the same low likelihood, $det(A_\alpha) \equiv 0 \pmod p$ and we must repeat our choice of $p$ in step 1.

The Newton-Hensel lifting of step 9 is described in detail in Subsection 3.4. Lemma 3.2 implies the cost to do this Newton-Hensel lifting by pipelining is time $O(\log^2 n)$ using $M(n)$ processors. Since $p^{2^{k^*}} \geq 2^{c\pi}$, the Newton-Hensel lifting requires bit precision at most $O(\pi)$. We have RF $\pmod{p^{2^{k^*}}}$ of $A$.

The recursive computation in step 10 of the determinants of the RF matrices $\pmod{p^{2^{k^*}}}$ is justified by the recursive formulae for determinants of the RF matrices given in Proposition 2.6. The recursive computation in steps 10 and 11 of the determinants of the RF matrices $\pmod{p^{2^{k^*}}}$ and of the integer multipliers in step 12, and of the exact RF in steps 13 costs at most parallel time $O(\log n)$ using $M(n)$

processors. The use of the RF $\pmod{p^{2^{k^*}}}$ of $A$ and multipliers $m_\alpha \pmod{p^{2^{k^*}}}$ in step 13 gives us the exact values of the RF of $A$ since $p^{2^{k^*}} \geq 2^{c\pi} > |\bar{m}_\alpha| \|A_\alpha\|_\infty$ for a choice of a large enough constant $c$.

Note that by Lemmas 3.6, 6.2, and 3.2, each major stage of the above RF algorithm takes at most time $O(\log^2 n)$ using $M(n)$ processors, using bit precision $O(\pi)$. Since there are only $O(1)$ such major stages, we have by constant slowdown that the total time is $O(\log^2 n)$ using $M(n)$ processors. ∎

If the input matrix $A$ is not SPD, then we apply the RF algorithm instead to the SPD $AA^T$, as described in Subsection 2.2 by use of the normal form reduction.

## 4. PARALLEL RF COMPUTATION FOR MATRICES OF BOUNDED DISPLACEMENT RANK

The techniques for computing an RF described in Section 3 can be applied to structured matrices with some refinements that exploit their structure. We first define Toeplitz matrices, as well as their finite generation and compact representation, and we give generalizations of these properties to matrices of bounded displacement rank. We next describe a known sequential algorithm for the RF of a matrix of bounded displacement rank. Finally, we present a specialization of our parallel algorithm for the RF of a matrix of bounded displacement rank, including modifications and extensions required to efficiently do Newton-Hensel Lifting and Newton Iteration in this case.

### 4.1. Toeplitz Matrices and their Computations.

Toeplitz matrices are quite simple to define, but bounded displacement matrices are not as simple. Therefore we first discuss Toeplitz matrices and their computations, then we extend the discussion to other more general classes of structured matrices. An $n \times n$ matrix $A$ with row and column indices beginning at 1 is said to be *Toeplitz* if $a_{i,j} = a_{(i-j+1),1}$ for $i \geq j$, and $a_{i,j} = a_{1,(j-i+1)}$ otherwise. From this definition, Toeplitz matrices have the structure:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} \\ a_{3,1} & a_{2,1} & a_{1,1} & \cdots & a_{1,n-2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ a_{n,1} & a_{n-1,1} & a_{n-2,1} & \cdots & a_{1,1} \end{bmatrix}.$$

Note that the column $u = (a_{1,1}, a_{2,1}, \ldots, a_{n,1})$ and row $v = (a_{1,1}, a_{1,2}, \ldots, a_{1,n})$, suffice to uniquely define the Toeplitz matrix $A$ (the matrix can also be uniquely be defined by the first and last rows or columns). Thus we can *compactly store* the Toeplitz matrix in space $2n-1$ using $2n-1$ indices of the matrix. The transpose $A^T$ of a Toeplitz matrix $A$ is Toeplitz, and every block of a Toeplitz matrix is Toeplitz.

Toeplitz matrices arise naturally in many polynomial computations. For example, for polynomials of degree $n-1$:

1. The product of such univariate polynomials can be computed by multiplying a banded Toeplitz matrix by a length $n$ vector. Also, the reciprocal and division of

univariate polynomials can be computed using the solution of triangular Toeplitz systems [17].

2. The GCD and resultant of two such univariate polynomials can be determined by the solution of a linear system defined by a *Sylvester* matrix consisting of two $n \times n$ Toeplitz submatrices [19, 16].

As mentioned above, we will compactly represent an $n \times n$ Toeplitz matrix by specifying the first row and column.

Recall that $P(n)$ denotes the number of arithmetic processors used to multiply two degree $n$ polynomials in $O(\log n)$ parallel time.

LEMMA 4.1. *(see [18, 14, 70, 64])*
*(a) The product of an $n \times n$ Toeplitz matrix with an $n$-vector can be computed in sequential time $O(P(n) \log n)$ and space $O(n)$, and also in parallel time $O(\log n)$ using $P(n)$ processors.*
*(b) The product of two $n \times n$ Toeplitz matrices can be computed (i.e., compactly represented as a sum of products of upper and lower compactly represented Toeplitz matrices) in sequential time $O(P(n) \log n)$ and space $O(n)$, and also in parallel time $O(\log n)$ using $P(n)$ processors.*

LEMMA 4.2. *[34] Let $T^{-1}$ be the inverse of an $n \times n$ Toeplitz matrix $T$, 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 last columns, respectively, of $T^{-1}$. Let $L(u)$ denote the lower triangular Toeplitz matrix whose first column is $u$, and let $U(v)$ be the upper triangular Toeplitz matrix whose first row is $v$. Then*

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

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

The pair of the vectors $u, v$ will be called the *generators* of $T^{-1}$. (All of the above results are known to extend to the bounded displacement matrices which will be defined below.)

**Generation and Compression of Structured Matrices by Matrix Operators.**

There is a large body of work (e.g. [45, 46]) concerned with the definition of structured matrices by the use of matrix operators as defined below. Let $\mathbf{R}_{n,m}$ denote the class of $n \times m$ matrices, which are vectors if $n = 1$ or $m = 1$. Let $\Phi : \mathbf{R}_{n,m} \to \mathbf{R}_{n,m}$. Consider a matrix $A \in \mathbf{R}_{n,m}$. A $\Phi$-*generator* of $A$ of length $\delta$ consists of two matrices $G \in \mathbf{R}_{n,\delta}$ and $H \in \mathbf{R}_{m,\delta}$ such that $\Phi(A) = GH^T$. The $\Phi$-*rank* of $A$ is defined to be the rank $r = r(\Phi(A))$ of the matrix $\Phi(A)$. Assuming $\Phi$ is 1-1, the $\Phi$-*generator* of $A$ will be stored in space $(n+m)\delta$ and provide unique information about $A$, whereas $A$ has $nm$ entries to store. As a trivial example of a matrix operator, consider the identity operator $I$. Then a generator of $A$ is an $I$-generator of $A$ and the $I$-rank of $A$ is just the rank of $A$.

**Displacement Operators for Generalizations of Toeplitz Matrices.** Let $Z$ be an $n \times n$ square matrix, zero everywhere, except for its first lower subdiagonal which is filled with ones. Note that premultiplications and postmultiplications by $Z$ and by $Z^T$ displace or shift the entries of an $n \times n$ matrix $A = [a_{ij}]$ as follows:

$$ZA = [a_{i-1,j}] \text{ giving a shift down}, Z^T A = [a_{i+1,j}] \text{ giving a shift up},$$

$$AZ = [a_{i,j+1}] \text{ giving a shift left}, AZ^T = [a_{i,j-1}] \text{ giving a shift right}$$

where $a_{i,j} = 0$ for $i$ or $j$ out of range.

[45][46] generate Toeplitz matrices and their generalizations using the following *displacement operators:*

1. Operator $\Delta_+(A) = A - ZAZ^T$ zeros all the entries of a Toeplitz matrix $A$, except for its first row and column.
2. Operator $\Delta_-(A) = A - Z^T AZ$ zeros all the entries of a Toeplitz matrix $A$, except for its last row and column.

For notational consistency with the rest of the paper, Let $\Delta$-rank$(A)$ denote rank$(\Delta(A))$ for each of the $\Delta$ operators.

**Finite Generation of Toeplitz Type Matrices.** We let $L(x)$ denote the lower triangular Toeplitz matrix with first column $x$. [43, 45, 46, 29] (for a summary of these results see [14, 66]) show the following relationships between the Toeplitz displacement operators and the finite generation of these structured matrices.

LEMMA 4.3. *For all matrices $A$, $G = [g_1, \dots, g_\delta]^T$ and $H = [h_1, \dots, h_\delta]^T$ where the $g_i$ and $h_i$ are column vectors, we have:*

1.$\Delta_+(A) = A - ZAZ^T = GH^T = \sum_{i=1}^{\delta} g_i {h_i}^T$ *iff* $A = \sum_{i=1}^{\delta} L(g_i) L^T(h_i)$, *and*
2.$\Delta_-(A) = A - Z^T AZ = GH^T = \sum_{i=1}^{\delta} {g_i^r}^T h_i^r$ *iff* $A = \sum_{i=1}^{\delta} L^T(g_i^r) L(h_i^r)$, *where superscript $r$ denotes the reverse of a vector.*

This Lemma implies that the $G, H$ matrices suffice to specify and generate the corresponding matrix classes.

## 4.2. Bounded Displacement Rank Matrices

Hereafter, we define for Toeplitz operators the *displacement rank* and *displacement generator* of a matrix to be its $\Delta$-rank and $\Delta$-generator, for $\Delta \in \{\Delta_+, \Delta_-\}$. Slightly simplifying the above definitions, we will define here (in the context of Toeplitz operators) an $n \times n$ matrix to have *displacement rank* $\delta$ if it can be written as the sum of $\delta$ terms, where either:

1. each term is the product of a lower triangular Toeplitz matrix and an upper triangular Toeplitz matrix, or
2. each term is the product of an upper triangular Toeplitz matrix and a lower triangular Toeplitz matrix.

This is sometimes known as the *compact or dyadic representation* of bounded displacement rank matrices, where these triangular Toeplitz matrices are themselves compactly represented by specifying the first or last rows. Note that any $n \times n$ matrix has displacement rank at most $n$. The results of [34] imply that for our above simplified definition of displacement rank, the inverse of matrix of displacement rank $\delta$ has displacement rank $\delta$. Summarizing the above known results (see also [34, 43, 45, 46, 29, 14, 18, 66] and the summary of [66], p. 8):

LEMMA 4.4.
*Let $\delta = \delta(\mathbf{C})$ be the displacement rank of a given class $\mathbf{C}$ of matrices.*

1. $\delta \leq 2$ *for Toeplitz matrices and their inverses,*

2. *the inverse of a matrix of displacement rank $\delta$ has displacement rank $\delta$,*

3. $\delta \leq 4$ *for the product of two Toeplitz matrices,*

4. $\delta \leq b + b'$ *for $n \times m$ matrices built by expanding $b \times b'$ block matrices whose elements are Toeplitz blocks,*

5. $\delta \leq 3$ *for Sylvester (resultant) and subresultant matrices (since these are block Toeplitz matrices with $b + b' = 3$).*

### 4.3. Constructing Displacement Generators

We now describe how to construct the displacement generators of a matrix with known minimal displacement rank $\delta$. It is known (see [18, 14]) that the displacement generator of rank $\delta$ is simply constructed from the first $\delta$ columns and first $\delta$ rows of a generic matrix with displacement rank $\delta$. It is also known (see [14]) that given a fixed operator $\Delta$ and a generic matrix $A$, a $\Delta$-generator of $A$ of minimum length can be efficiently computed from an appropriate size submatrix of $\Delta(A)$. We will now briefly describe this construction. Recall that Lemma 4.3 implies that the $G, H$ matrices suffice to specify elements of the corresponding matrix classes. In the following, let $[\delta]$ denote the sequence $(1, \ldots, \delta)$. Given a $n \times n$ matrix $S$, let $S([\delta])$ denote the $\delta \times n$ matrix consisting of the first $\delta$ rows of $S$, let $S(-, [\delta])$ denote the $n \times \delta$ matrix consisting of the first $\delta$ columns of $S$, and let $S([\delta], [\delta])$ denote the $\delta \times \delta$ leading principal submatrix of $S$. The following is well known in linear algebra:

PROPOSITION 4.1. *If $S$ is of rank $\delta$, and $S([\delta], [\delta])$ is nonsingular, then $S = S(-, [\delta]) S([\delta], [\delta])^{-1} S([\delta])$.*

Using this, [14] show in their Lemmas 2 and 6:

LEMMA 4.5. *Suppose $A$ has minimum displacement rank $\delta$, and $S = \Delta_+(A) = A - ZAZ^T$ has a nonsingular $\delta \times \delta$ leading principal submatrix $S([\delta], [\delta])$. Define $\Delta_+$-displacement generator $n$-vectors $\{g_i\}, \{h_i\}$, $i = 1, \ldots, \delta$, where $(g_1, \ldots, g_\delta) = S(-, [\delta])$ are the first $\delta$ columns of $S$, and ${h_i}^T$ is the $i$th row of the product $S([\delta], [\delta])^{-1} S([\delta])$, and $S([\delta])$ are the first $\delta$ rows of $S$. Then $S = \Delta_+(A) = S(-, [\delta]) S([\delta], [\delta])^{-1} S([\delta])$ $= \sum_{i=1}^{\delta} g_i {h_i}^T$ and $A = \sum_{i=1}^{\delta} L(g_i) L^T(h_i)$.*

(A similar lemma holds for $\Delta_-$, and in fact all the other displacement operations.) If $A$ has minimum displacement rank $\delta$, then $S = \Delta_+(A)$ always has rank $\delta$. (Note that in general, we may have only an upper bound on the displacement rank of the matrix $A$. In that case, we must apply a known rank test [47] to find the minimum displacement rank $\delta$ where $S = \Delta_+(A)$ has rank $\delta$.) Section 8 gives a randomized algorithm for general problem of constructing displacement generators, when the $\delta \times \delta$ leading principal submatrix is singular. But for our purposes the above construction will suffice, since we will need to construct displacement generators only for matrices with an $S$ that has a nonsingular $\delta \times \delta$ leading principal submatrix.

Since $S([\delta], [\delta])$ is of size $\delta \times \delta$, the rank and matrix inverse computation $S([\delta], [\delta])^{-1}$ can be done in sequential time $O(\delta^{\omega^*})$ for $\omega^* = 2.376$, and also in parallel time $O(\log^2 \delta)$ using $\delta^\omega$ processors by known dense matrix algorithms. The other inner products can be broken into $n/\delta$ inner products of (general) matrices of size $\delta \times \delta$ (each costing sequential time $O(\delta^{\omega^*})$ and space $O(\delta^2)$, or parallel time $O(\log \delta)$ using $\delta^\omega$ processors), followed by a sum of $n/\delta$ of these resulting matrices (costing sequential time $O(n\delta)$, or parallel time $O(\log n\delta)$ using $n\delta/\log n\delta$ processors); this costs sequential time $O(n\delta + (n/\delta)\delta^{\omega^*}) = O(\delta^{\omega^*-1}n)$ and space $O(n\delta)$, or parallel time $O(\log n\delta)$ using $\delta^{\omega-1}n/\log n\delta$ processors. Note that the total time bound is $O((\log^2 \delta) + (\log n\delta)) = O((\log \delta) \log n\delta)$ and the processor bound is $\delta^\omega + \delta^{\omega-1}n/\log n\delta = O(\delta^{\omega-1}n/\log n\delta)$. Summarizing the above, we have by constant slowdown:

LEMMA 4.6. *Given as input a matrix $A$ of size $n \times n$, GENERATOR$_\delta(A)$ can be efficiently computed in sequential time $O(\delta^{\omega^*-1}n)$ and space $O(\delta n)$, and also in parallel time $O((\log \delta) \log n\delta)$ using $\delta^{\omega-1}n/\log n\delta$ processors. GENERATOR$_\delta(A)$ yields displacement generator $n$-vectors $\{g_i\}, \{h_i\}$, $i = 1, \ldots, \delta$ where $\sum_{i=1}^{\delta} L(g_i)L^T(h_i)$ has displacement rank $\delta$.*

In general, our Newton iteration algorithm for bounded displacement rank will be given, as input to GENERATOR$_\delta$, an $n \times n$ matrix which is a close approximation to a matrix of displacement rank $\delta$ (but which itself may not be of displacement rank $\delta$). Then GENERATOR$_\delta$ with this input yields displacement generator $n$-vectors $\{g_i\}, \{h_i\}$, $i = 1, \ldots, \delta$. We will later show (see Proposition 4.7) that the resulting matrix $\sum_{i=1}^{\delta} L(g_i)L^T(h_i)$ of displacement rank $\delta$ approximates the matrix input to GENERATOR$_\delta$ with small error.

### 4.4. Computations on Bounded Displacement Rank Matrices

As a consequence of Lemma 4.6, we can assume throughout the rest of this paper that for a given matrix with displacement rank $\delta$, a displacement generator of length $\delta$ is also available.

The next lemma summarizes results on matrices with displacement rank $\delta$ that will be of use. The proof is by reduction of the computations to convolution and FFTs [1, 14, 18, 59, 70, 64].

LEMMA 4.7.

*(a) Given a displacement rank $\delta$ matrix $A$, any fixed column or row of $A$ can be computed in $O(\log(\delta n))$ time using $\delta P(n)$ processors.*

*(b) The product of an $n \times n$ displacement rank $\delta$ matrix and an $n \times n$ displacement rank $\delta'$ matrix is a displacement rank $\delta\delta' + O(1)$ matrix computable (in com-*

*pact form as a sum of products of upper and lower Toeplitz matrices) in sequential time $O(\delta\delta' P(n)\log n)$ and space $O(\delta\delta' n)$, and also in parallel time $O(\log n)$ using $\delta\delta' P(n)$ processors. Furthermore, if $\delta' \leq \delta$, the minimum displacement rank of the product is computable, by additional use of Lemma 4.6 (costing sequential time $O((\delta\delta')^{\omega^*-1} n + \delta\delta' P(n)\log n) \leq O(\delta\delta' P(n)\log n)$ or parallel time $O((\log\delta\delta')\log n\delta\delta')$ using $(\delta\delta')^{\omega-1} n/(\log n\delta\delta') + \delta\delta' P(n) \leq O(\delta\delta' P(n))$ processors) in no further asymptotic cost.*

*(c)The inverse of an $n\times n$ displacement rank $\delta$ lower triangular matrix is a displacement rank $\delta$ matrix computable (in compact form as a sum of $\delta$ products of upper and lower Toeplitz matrices) in sequential time $O(\delta^2 P(n)\log n)$ and space $O(\delta n)$, and also in parallel time $O(\log n)$ using $\delta^2 P(n)$ processors.*

*(d)The inverse of an $n\times n$ displacement rank $\delta$ matrix is a displacement rank $\delta$ matrix computable (by use of Lemma 4.6 and Lemma 4.8; also see [14, 18, 59] for the explicit compact formulas as a sum of $\delta$ products of upper and lower Toeplitz matrices) in sequential time $O(\delta^2 P(n)\log n)$.*

Recall the best previously known parallel algorithms [70, 64, 67, 68]) for the problems of inverse, determinant, linear system solution, $LU$ factorization, and finding rank of an $n\times n$ Toeplitz matrix required parallel time $\Omega(\log^2 n)$ using $O(nP(n))$ work. Also recall the best previously known parallel algorithms [70, 64, 67, 68]) for the inverse of an $n\times n$ displacement rank $\delta = O(1)$ matrix required parallel time $\Omega(\log^2 n)$ using $O(nP(n))$ processors, or polylog time using $\Omega(n^2/\log^{O(1)} n)$ processors. We will drop these processor bounds to linear, without significant time slowdown.

### 4.5. The Normal Form Reduction for Bounded Displacement Rank

This section computes the inverse and solves linear systems for the class of rational matrices of bounded displacement rank. We give an efficient parallel algorithm for the recursive factorization of such matrices with the additional restriction that the input be SPD. To solve the associated matrix inverse and linear systems problems, where the nonsingular input matrix $A$ is not SPD, we apply the recursive factorization algorithm to the SPD normal form $AA^T$ as described in Subsection 2.2, so $AA^T$ is factored as $AA^T = LU$ where $L, U$ are triangular and of bounded displacement rank. There are known highly efficient parallel algorithms (see JáJá [42], [66]) for triangular inverse $L^{-1}$ and $U^{-1}$ of displacement rank $\delta$. After we get the inverse $(AA^T)^{-1} = (LU)^{-1} = U^{-1}L^{-1}$, then we immediately get the inverse $A^{-1} = A^T(AA^T)^{-1}$ in one further product of a matrix of bounded displacement rank (via FFT). Also, given an $n$-vector $b$, the linear system $Ax = b$ can be solved, as in Subsection 2.2, by two back-solving stages for triangular linear systems over $L$ and $U$ of bounded displacement rank. By Lemma 4.7, these further computations (including the matrix product $AA^T$ and backsolving) cost $O(\log^2 n)$ time using $O(\delta^2 P(n))$ processors, since the operations are on triangular matrices of displacement rank $\delta$.

### 4.6. Sequential Computation of an RF of a Matrix of Bounded Displacement Rank

Let $A_\alpha$ be the matrices of the RF tree of matrix $A$. The sequential algorithm of Bitmead and Anderson[14] (also see Morf [59]) for Toeplitz matrix inverse, takes as input a symmetric diagonally dominant (DD) matrix $A$ and constructs an RF sequence of depth $\log n$. That RF sequence consists of RF matrices of form $A_{1^d}$, for $d = 0, \ldots, \log n$ (these are denoted $A_d$ in section 2.4). They show (using the results on displacement ranks of products and inverse summarized in Lemma 4.4) that the displacement rank of the induced submatrices at each recursion level is increased only by an additive factor of 2. This holds also for the RF tree of $A$.

PROPOSITION 4.2. *(Bitmead and Anderson[14]) Let $A$ be an $n \times n$ integer matrix of displacement rank $\delta$. For $0 \leq d < \log n$, for each matrix $A_\alpha$ at depth $d$ in the recursive factorization tree, the displacement rank of the children $A_{\alpha 1}$ and $A_{\alpha 0}$ is not more than that of $A_\alpha$. Hence the displacement rank of $A_\alpha$ is at most $\delta$.*

This Proposition allows them to bound the sequential work for the RF sequence for Toeplitz matrices, and their algorithm generalizes to compute the RF sequence of a matrix $A$ of displacement rank $\delta$. By the obvious recursion, their algorithm can be immediately extended to construct the RF tree.

LEMMA 4.8. *Given an $n \times n$ input matrix of displacement rank $\delta$, the sequential algorithm of Bitmead and Anderson[14] can be applied recursively to compute the complete RF tree in sequential time $T(n) \leq O(\delta^2 P(n) \log^2 n)$.*

An understanding of a sequential time analysis for computing the RF tree will aid us in the design and analysis of our parallel algorithms. Their recursive calls have depth at most $\log n$. Thus the displacement rank of the submatrices is at most $\delta$ at any level of recursion, and no more than the number of rows of the submatrix. Let $n_d = n/2^d$. The inverse of each submatrix is computed by Proposition 2.7 in a recursive (bottom-up) fashion. The matrix computations involved are only recursive inverse, multiplication and addition. Each procedure call for the RF tree results in two recursive calls: one call for a matrix inverse (which can be computed by a recursive for the RF tree) and also a further recursive call for the RF tree.

For the matrix at each node of the RF tree at depth $d$, we must compute the inverse of a matrix of size $n_d \times n_d$ with displacement rank $\leq \delta$, and construct a $\delta$-generator of minimum length. This can be done by the techniques given in Subsection 4.3; Lemma 4.6 implies the sequential time cost is $O(\delta^{\omega^*-1} n_d)$. By Lemma 4.7, the further matrix operations have sequential time cost $O(\delta^{\omega^*-1} n_d + \delta^2 P(n_d) \log n_d)$. Hence the total sequential cost per node, is $t_d = O(\delta^{\omega^*-1} n_d + \delta^2 P(n_d) \log n_d)$.

The total sequential time complexity $T(n)$ of the sequential algorithm of Bitmead and Anderson[14] (also see Morf [59]) for the RF tree is bounded by $T(n) = O(1)$ for $n = 1$ and by $T(n) \leq \sum_{d=0}^{\log n} 2^d t_d$ for $n > 1$. Since $2^d t_d \leq t_{\log n} = O(\delta^{\omega^*-1} n_d + \delta^2 P(n_d) \log n_d) \leq O(\delta^2 P(n) \log n)$, these bounds imply $T(n) \leq O(\delta^2 P(n) \log n) \log n = O(\delta^2 P(n) \log^2 n)$.

### 4.7. Our Parallel Algorithm for Computing an RF of a Matrix of Bounded Displacement Rank

An *RF Tree of depth* $d'$ of $A$ is an RF Tree defined only to depth $d'$. Here we apply the stages of the RF Algorithm of Subsection 3.3 which we modify (i) to be specialized to matrices of displacement rank $\delta$ and (ii) to only compute the RF tree to depth $d^* = \log n - \Omega(\log\log n)$ by parallel algorithms, and (iii) to extend the RF tree to depth $\log n$ by the parallel use of sequential algorithms in subtrees.

**Algorithm RF for Bounded Displacement Rank**

**INPUT:** An $n \times n$ integer nonsingular matrix $A$, of constant displacement rank $\delta$, and with integer entries of magnitude $\leq 2^\beta$, where $\beta \leq n^{O(1)}$. w.l.o.g., we assume $n$ is a power of 2.

**If** the input $A$ is not SPD, **then** we apply the RF algorithm instead to the SPD $AA^T$, as described above.

Apply all the same steps as the RF Algorithm of Subsection 3.3, except that we redefine steps 2, 3, 4, and 9 as follows:

2. Let $D$ be an $n \times n$ diagonal integer matrix, such that the value of the $i$th diagonal element is $\hat{\delta}!/j$ where $j \in \{1, 2, \dots, \hat{\delta}\}$ and $j = i \bmod \hat{\delta}$, where $\hat{\delta} = 2^{\lceil \log \delta \rceil}$ $\leq 2\delta - 1$. Let $\bar{A} = A + \kappa D$, where $\kappa = p(n\|A\|_\infty)^{c_1}$ is an integer and $c_1$ is a sufficiently large positive constant.

3. Let $d^* = \log n - \log\log n$. Apply to $\bar{A}$ the approximate Newton Iteration of bounded displacement rank of Subsection 4.9, using bit precision $O(\pi)$, to compute a rational approximate RF tree for $\bar{A}$ within accuracy $2^{-c\pi}$, for a sufficiently large positive constant $c$.

4. Using the RF tree of depth $d^*$ for $\bar{A}$, within accuracy $2^{-c\pi}$, compute an approximate $LU$ factorization of $\bar{A}_\alpha$ and from this compute a rational approximation to $det(\bar{A}_\alpha)$. Also, by parallel use of known sequential algorithms for RF sequences applied at all the subtrees rooted at depth $d^*$, extend the computation of the rational approximation of RF sequences to a rational approximation of LU factorizations and determinants of all matrices from depth $d^*$ to depth $\log n$ of this approximate RF tree.

Again we can apply the homomorphism from $\mathbf{Q}$ to $\mathbf{Z}_p$, since we have assumed a model of computation with unit cost division over a finite field.

5. Apply Newton-Hensel lifting of Subsection 4.8 to compute **for** $i = 0, \dots, k^* = \lceil \log(c\pi/\log p) \rceil$ the RF $\pmod{p^{2^i}}$ of $A$. By parallel use of known sequential algorithms for Newton-Hensel lifting applied at all the subtrees rooted at depth $d^*$, extend the computation of the RF $\pmod{p^{k^*}}$ of all matrices from depth $d^*$ to depth $\log n$.

**Matrices that are Nearly Diagonal with Bounded Diagonal Elements.** Recall that we defined a matrix to be NIM if it is $\approx$ the product of a scalar and an identity matrix. Now we generalize the definition of NIM to matrices that are nearly diagonal with bounded diagonal elements. For a scalar $\rho$, let $A$ be NIM($\rho$) if $A \approx D(A)$ where $D(A)$ is a diagonal matrix with $\rho \geq \max_i(D(A)_{i,i})\max_j((D(A)^{-1})_{j,j})$.

Proposition 4.3. *If $A$ is NIM($\rho$) then $A$ is strongly DD and $cond_\infty(A) \leq \rho(1 + o(1))$.*

**Proof:** Since $A$ is NIM($\rho$), $A \approx D(A)$, so $A$ is strongly DD. Furthermore, $cond_\infty(A) = \|A\|_\infty \cdot \|A^{-1}\|_\infty \leq \|D(A)\|_\infty \cdot \|D(A)^{-1}\|_\infty(1 + o(1))$ $\max_i(D(A)_{i,i})$ $\max_j((D(A)^{-1})_{j,j})(1 + o(1)) = \rho(1 + o(1))$. ∎

PROPOSITION 4.4. *Each $\bar{A}_\alpha$ of the RF of $\bar{A}$ is NIM($\hat{\delta}$), so each $\bar{A}_\alpha$ is strongly DD and well conditioned with $cond_\infty(\bar{A}_\alpha) \leq \hat{\delta}(1 + o(1)) = O(1)$. Also, the $\delta \times \delta$ leading principal submatrix of $\Delta_+(\bar{A}_\alpha^{-1}) = \bar{A}_\alpha^{-1} - Z\bar{A}_\alpha^{-1}Z^T$ is NIM, so is strongly well conditioned.*

**Proof:** By definition, $\bar{A} = A + \kappa D$, where $\kappa = p(n\|A\|_\infty)^{c_1}$. Also by definition, $D$ an $n \times n$ diagonal integer matrix, such that the value of the $i$th diagonal element is $\hat{\delta}!/j$ where $j \in \{1, 2, \ldots, \hat{\delta}\}$ and $j = i \bmod \hat{\delta}$. Note that $\max_i(D_{i,i})= \hat{\delta}!$ and $\max_j(D_{j,j}^{-1})= ((\hat{\delta} - 1)!)^{-1}$. These definitions immediately imply that $\bar{A} \approx \kappa D$ and that $\bar{A}$ is NIM($\rho$) where $\rho= \max_i(\kappa D_{i,i})\max_j(\kappa^{-1}D_{j,j}^{-1}) = \max_i(D_{i,i})\max_j(D_{j,j}^{-1})$ $= (\hat{\delta}!)((\hat{\delta} - 1)!)^{-1} = \hat{\delta}$. Since $\bar{A} \approx \kappa D$, and $D$ is diagonal, it follows that $\bar{A}^{-1} \approx \kappa^{-1}D^{-1}$, so $\bar{A}^{-1}$ is also NIM($\hat{\delta}$). Since $ZD^{-1}Z^T$ shifts each entry of $D^{-1}$ by one entry to the right and down, it follows that the first $\hat{\delta}$ diagonal elements of $\Delta_+(D^{-1}) = D^{-1} - ZD^{-1}Z^T$ are each $1/\hat{\delta}!$. Let $S = \Delta_+(\bar{A}^{-1}) = \bar{A}^{-1} - Z\bar{A}^{-1}Z^T$ and let $S([\delta], [\delta])$ be the $\delta \times \delta$ leading principal submatrix of $S$. Thus $S \approx (\kappa^{-1}D^{-1}) - Z(\kappa^{-1}D^{-1})Z^T \approx \kappa^{-1}(D^{-1} - ZD^{-1}Z^T)$ and so the diagonal elements of $S([\delta], [\delta])$ are each $\approx \kappa^{-1}/\hat{\delta}!$. Hence $S([\delta], [\delta]) \approx (\kappa\hat{\delta}!)^{-1}I$ and so $S([\delta], [\delta])$ is NIM.

Note that the diagonal elements of $D$ repeat every $\hat{\delta}$ elements, where $\hat{\delta}$ is a power of 2. Using this fact, an easy induction shows that each $\bar{A}_\alpha \approx \kappa D_\alpha$, where $D_\alpha$ is an $n/2^{|\alpha|} \times n/2^{|\alpha|}$ diagonal integer matrix defined so again the value of the $i$th diagonal element is $\hat{\delta}!/j$ where $j \in \{1, 2, \ldots, \hat{\delta}\}$ and $j = i \bmod \hat{\delta}$. Hence by the same proof, $\bar{A}_\alpha$ is again NIM($\hat{\delta}$), and again the $\delta \times \delta$ leading principal submatrix of $\Delta_+(\bar{A}_\alpha^{-1}) = \bar{A}_\alpha^{-1} - Z\bar{A}_\alpha^{-1}Z^T$ is NIM. ∎

After we first specialize and analyze the Newton-Hensel lifting (Subsection 4.8) and Newton iterations (Subsection 4.9) for bounded displacement rank matrices, we complete the proof and analysis of the RF algorithm for bounded displacement rank in Subsection 4.10.

### 4.8. Newton-Hensel Lifting for RF of Bounded Displacement Rank

**Displacement Rank Bounds for Newton-Hensel Lifting.** We provide a known bound (see [70, 64, 67, 68]) on displacement rank for Newton-Hensel lifting which will be useful to our new parallel algorithms. Let $A$ be an $n \times n$ integer matrix of displacement rank $\delta$. Then by lemma 4.4, $A^{-1}$ has displacement rank $\delta$.

Let us introduce a scalar indeterminate $\lambda$. Since the modular operation is scalar, and $A^{-1}$ has displacement rank $\delta$, it follows (see [42]):

PROPOSITION 4.5. *For each integer $j \geq 0$, if $A$ has displacement rank $\delta$, then*

$$(I - \lambda A)^{-1} = \sum_{i=0}^{j-1} (\lambda A)^i \pmod{\lambda^j}$$

*has displacement rank $\delta$.*

**Newton-Hensel Lifting for Matrices of Bounded Displacement Rank.** Recall that Moenck and Carter [57] show that $S^{(k)}(A) = A^{-1} \pmod{p^{2^k}}$. Letting $\lambda = p$, by Proposition 4.5, we have, LEMMA 4.9. *For all $k \geq 0$, if $A$ has displacement rank $\delta$, then $S^{(k)}(A) = A^{-1} \pmod{p^{2^k}}$ has displacement rank $\leq \delta$ in $Z_{p^{2^k}}$.*

**Newton-Hensel Lifting for RF of Bounded Displacement Rank.** Let an *RF* $\pmod p$ of an $n \times n$ matrix $A$ be an RF of matrix $A$ where each matrix in the factorization is taken $\pmod p$. This will also be called a *modular RF*. In this section we assume we are given an RF $\pmod p$ of matrix $A$. We shall compute an RF $\pmod{p^{2^k}}$ of $A$.

Fix again $d^* = \log n - \Omega(\log\log n)$. Given an RF $\pmod p$ of depth $d^*$ for $n \times n$ matrix $A$ of displacement rank $\delta$, we wish to compute the RF $\pmod{p^{2^k}}$ of depth $d^*$. In each stage of Newton-Hensel lifting, Lemma 4.9 implies that the matrices resulting from Hensel lifting do not increase in displacement rank. Proposition 4.2 implies that all nodes of the tree of depth $d$ are matrices of displacement rank $\delta$.

The obvious way to compute the RF $\pmod{p^{2^k}}$ of $A$ of depth $d^*$ is by proceeding level by level through the RF in $1 + d^*$ stages, for $d = 0, 1, \ldots, d^*$. This gives:

LEMMA 4.10. *Given an RF $\pmod p$ of depth $d^* = \log n - \Omega(\log\log n)$ for an $n \times n$ matrix $A$ of displacement rank $\delta$, we can compute an RF $\pmod{p^{2^k}}$ of $A$ in parallel time $O(k \log^2 n)$ using $\delta^2 P(n)$ processors.*

**Proof:** Let $n_d = n/2^d$. Let us consider the cost for each of the $k$ Newton-Hensel lifting iterations at each node at depth $d$ in the RF. This requires multiplication of matrices of size $n_d \times n_d$ with displacement rank $\delta$, and so by Lemma 4.7 costs $O((\log\delta)\log n_d\delta)$ time using $\delta^2 P(n_d)$ processors. We also require the parallel complexity stated in Lemma 4.6 to construct the displacement rank $\delta$ generators after each iteration. By Lemma 4.6, at each node of the RF tree at depth $d$ we can compute the $\delta$ displacement rank generator of the corresponding $n_d \times n_d$ matrix at that node, in $O(\log\delta)\log n_d\delta$ parallel time and $\delta^{\omega-1}n_d/(\log n_d\delta)$ processors. Thus the total cost per node at depth $d$ on each iteration is $T_d = O(\log\delta)\log n_d\delta$ parallel time and $P_d \leq \delta^{\omega-1}n_d/(\log n_d\delta) + \delta^2 P(n_d)$ processors. Note that $T_d \leq O(\log n)$ for all $d$. The total time for all $1 + \log n$ levels and all $k$ iterations at each of these levels is thus $O(k\log^2 n)$. Since there are $2^d$ nodes at depth $d$, the total processor bound is $\max_{0 \leq d \leq d^*} 2^d P_d \leq O(P_{d^*}) = O(\delta^{\omega-1}n/\log n\delta + \delta^2 P(n)) \leq O(\delta^2 P(n))$. Thus, with a constant further slowdown, the $1 + \log n$ levels require total parallel time $O(k\log^2 n)$ using $\delta^2 P(n)$ processors. ∎

We can further improve on Lemma 4.10 by use of the pipelining (stream contraction) techniques described in Subsection 3.4.

LEMMA 4.11. *Given an RF $\pmod p$ of an $n \times n$ matrix $A$ of displacement rank $\delta$, we can compute an RF $\pmod{p^{2^k}}$ of $A$ in parallel time $O(k \log n)$ using $\delta^2 P(n)$ processors.*

**Proof:** The pipelined algorithm for the RF $\pmod{p^{2^k}}$ of $A$ described in Subsection 3.4 for a dense $A$ requires $O(k)$ iterations of simultaneous Newton-Hensel lifting iterations at every node of the tree. We now analyze this pipelined algorithm for the RF $\pmod{p^{2^k}}$, as specialized to an input matrix $A$ of displacement rank $\delta$. By the proof of Lemma 4.10, the cost for each of the $k$ Newton-Hensel lifting iterations at each node at depth $d$ in the RF is: $T_d = O(\log n)$ parallel time using $\delta^{\omega-1} n_d / (\log n_d \delta) + \delta^2 P(n_d)$ processors, where again $n_d = n/2^d$. Thus the total time for $k$ Newton-Hensel lifting iterations is $O(k \log n)$. Since there are $2^d$ nodes at depth $d$, the total processor bound for all $1 + d^*$ levels of the RF tree is $\sum_{d=0}^{d^*} 2^d P_d \leq O(P_{d^*}) = O(\delta^{\omega-1} n / \log n\delta + \delta^2 P(n)) \leq O(\delta^2 P(n))$ processors. A further constant slowdown completes the Lemma. ∎

### 4.9. Approximate Newton Iterations with Bounded Displacement Rank

Fix a very small (with respect to $A$) $\epsilon = (n\|A\|)^{-c'} < 1/(32s)$ for some sufficiently large positive constant $c'$ and an $s$ where $1 \leq s \leq (n\|A\|)_\infty)^{O(\delta)}$ and $\delta = O(1)$. For all $k \geq 0$ let $E_k(\epsilon, s) = (16s\epsilon)^{2^k}/(16s)$. Note that by definition $E_0(\epsilon, s) = \epsilon$ and observe that $E_k(\epsilon, s)$ exactly satisfies the recurrence $E_k(\epsilon, s) = s(4E_{k-1}(\epsilon, s))^2$ for all $k > 0$. Also, since we assume $\epsilon < 1/(32s)$, it follows that

$$E_{k^+}(\epsilon, s) < (16s\epsilon)^{2^{k^+}} \leq 2^{-2^{k^+}} \leq 2^{-\bar{c}\pi},$$

for $k^+ = \lceil log(\bar{c}\pi) \rceil$ where $\bar{c}$ is a positive constant. Observe that $k^+ \leq O(\log n)$, assuming $\beta \leq n^{O(1)}$. We use GENERATOR, as described in Subsection 4.3, to reduce the displacement rank of the iterates. To take into consideration the errors due to application of GENERATOR, we will use the error bound $E_k(\epsilon, s)$ in place of $E_k(\epsilon, 1) = E_{k-1}(\epsilon)$ defined in Subsection 3.5.

Let $A$ be an $n \times n$ symmetric matrix. We consider the case of Newton iteration where

1. $A$ has constant displacement rank $\delta$,
2. $\|A\|_\infty \geq 1$,
3. $A$ is NIM($\hat{\delta}$), in particular $A$ is $\epsilon$-DD for a very small $\epsilon = (n\|A\|)^{-c'}$ where $c' = c''\delta$, for a large enough positive constant, say $c'' > 8$,
4. the $\delta \times \delta$ leading principal submatrix of $\Delta_+(A^{-1}) = A^{-1} - ZA^{-1}Z^T$ is NIM,
5. $A$ is not initially given with full accuracy, and instead we are provided a sequence of approximates $\tilde{A}^{(0)}, \tilde{A}^{(1)}, \ldots$ where $\|A - \tilde{A}^{(k)}\|_\infty \leq E_k(\epsilon, 1)$, and each $\tilde{A}^{(k)}$ is NIM($\hat{\delta}$) and hence strongly DD, and has bit precision $\leq O(\pi)$,
6. the $B^{(k)}$ are NIM($\hat{\delta}$) and hence strongly DD and are approximated within displacement rank $\delta$ by use of the $\Delta_+$-displacement generator GENERATOR$_{\delta^{(k)}}$ defined just before Lemma 4.6 in Subsection 4.3.

(In our applications, we apply the iterations to matrix $\bar{A}$, and the above assumptions 3 and 4 are justified by Proposition 4.4.)

Let $\tilde{B}^{(0)} = D(\tilde{A}^{(0)})^{-1} = diag(1/\tilde{a}_{11}, 1/\tilde{a}_{22}, \ldots, 1/\tilde{a}_{nn})$ where $\tilde{A}^{(0)} = [\tilde{a}_{ij}]$.

By Proposition 3.4, if $\tilde{A}^{(0)}$ is $\epsilon$-DD then $\|I - \tilde{B}^{(0)}\tilde{A}^{(0)}\|_\infty < \epsilon$.

We will generate a sequence of matrices $\tilde{B}^{(0)}, \hat{B}^{(0)}, \tilde{B}^{(1)}, \hat{B}^{(1)}, \ldots$ (which are clearly also NIM($\hat{\delta}$) and so strongly DD) where:

1. $\hat{B}^{(k)}$ is derived by applying a single exact Newton iteration, just as described in Subsection 3.5, for approximate inverse of $\tilde{A}^{(k)}$, with initial inverse approximation $\tilde{B}^{(k-1)}$. That is $\hat{B}^{(k)} = \tilde{B}^{(k-1)}(2I - \tilde{A}^{(k)}\tilde{B}^{(k-1)})$. Since $\tilde{B}^{(k-1)}$ is NIM($\hat{\delta}$), so is $\hat{B}^{(k)}$.

2. Apply $\text{GENERATOR}_{\delta^{(k)}}$ $(\hat{B}^{(k)})$, as defined in Subsection 4.3, where $\delta^{(k)} = \delta$ if $\Delta_+(\hat{B}^{(k)})$ has rank $\geq \delta$ and otherwise $\delta^{(k)}$ is the rank of $\Delta_+(\hat{B}^{(k)})$. Then define $\breve{B}^{(k)} = \sum_{i=1}^{\delta^{(k)}} L(g_i)L^T(h_i)$ to be the $n \times n$ matrix with displacement rank $\delta^{(k)} \leq \delta$, with generators $g_i, h_i, i = 1, \ldots, \delta^{(k)}$.

3. Let $\tilde{B}^{(k)}$ be the displacement rank $\delta^{(k)}$ matrix derived from $\breve{B}^{(k)}$ by rounding (i.e., each of the terms $L(g_i), L^T(h_i)$ in the $\delta^{(k)}$ term matrix sum are so rounded) to within bit precision $O(\pi)$, so that $\|I - \tilde{B}^{(k)}\tilde{A}^{(k)}\|_\infty \leq \|I - \breve{B}^{(k)}\tilde{A}^{(k)}\|_\infty + E_k(\epsilon, s)$. By the close approximation of $\breve{B}^{(k)}$ by NIM$(\hat{\delta})$ matrix $\tilde{B}^{(k)}$, is follows that $\breve{B}^{(k)}$ is also NIM$(\hat{\delta})$ and hence also strongly DD.

**Bounding Error Due to Application of GENERATOR.** In step 2 of the above Newton iteration algorithm for bounded displacement rank, we apply $\text{GENERATOR}_{\delta^{(k)}}$ to the $n \times n$ matrix $\hat{B}^{(k)}$ which is a close approximation to the matrix $A^{-1}$ of displacement rank $\delta$ (but which itself may not be of displacement rank $\delta$). Then $\text{GENERATOR}_{\delta^{(k)}}(\hat{B}^{(k)})$ yields displacement generator $n$-vectors $\{g_i\}, \{h_i\}$, $i = 1, \ldots, \delta$. We will now show that the resulting matrix $\tilde{B}^{(k)}$, which is of displacement rank $\delta$, approximates the matrix $\hat{B}^{(k)}$ with small error. Since $A$ is assumed to be NIM$(\hat{\delta})$, then by assumption on the close approximation of $A$ by the $\tilde{A}^{(k)}$, we have:

PROPOSITION 4.6. *Each $\tilde{A}^{(k)}$ is also NIM$(\hat{\delta})$ and so strongly DD.*

PROPOSITION 4.7. $\|I - \tilde{B}^{(k)}\tilde{A}^{(k)}\|_\infty \leq s\epsilon_k + E_k(\epsilon, s)$, *where* $s = (n\|A\|_\infty)^{O(\delta)}$ *and* $\epsilon_k = \|I - \hat{B}^{(k)}\tilde{A}^{(k)}\|_\infty$.

**Proof:** We have assumed that the $\delta \times \delta$ leading principal submatrix of $\Delta_+(A^{-1}) =$ $A^{-1} - ZA^{-1}Z^T$ is NIM. (In our applications, we apply the iterations to matrix $\bar{A}$, and this assumption is justified by Proposition 4.4.) Since each $\tilde{B}^{(k)}$ differs from $A^{-1}$ by at most $\epsilon$, it follows that the $\delta \times \delta$ leading principal submatrix of $\Delta_+(\tilde{B}^{(k)}) = \tilde{B}^{(k)} - Z\tilde{B}^{(k)}Z^T$ is also NIM, so is strongly DD and strongly well conditioned, with condition number $\leq 1 + 1/(n\|A\|)^{\Omega(1)}$. The construction of GENERATOR involves the inverse of this strongly well conditioned $\delta \times \delta$ leading principal submatrix of $\Delta_+(\tilde{B}^{(k)})$, which induces a multiplicative error factor of at most $(n\|A\|_\infty)^{O(\delta)}$. The construction of GENERATOR also involves products by the first $\delta$ rows and columns of $\Delta_+(\tilde{B}^{(k)})$. Since these products induce an additional multiplicative error factor of $n^{O(1)}\|A\|_\infty$, the error due to this approximation of $\hat{B}^{(k)}$ by $\breve{B}^{(k)}$ is $\|I - \breve{B}^{(k)}\tilde{A}^{(k)}\|_\infty \leq s\epsilon_k$. Hence the total error due to the approximation of $\hat{B}^{(k)}$ by $\tilde{B}^{(k)}$ is $\|I - \tilde{B}^{(k)}\tilde{A}^{(k)}\|_\infty \leq \|I - \breve{B}^{(k)}\tilde{A}^{(k)}\|_\infty + E_k(\epsilon, s)$ $\leq s\epsilon_k + E_k(\epsilon, s)$. ∎

Thus, we have that $\tilde{B}^{(k)}$ is a displacement rank $\delta^{(k)} \leq \delta$ approximation to $\hat{B}^{(k)}$ with error $\leq s\epsilon_k + E_k(\epsilon, s)$.

**Inductive Proof of Quadratic Convergence.** Next, we give an inductive proof of quadratic convergence of the entire iteration sequence, which is very similar to, but somewhat more involved than, Proposition 3.11. Note that at certain places we must use the error bound $E_k(\epsilon, s)$ in place of $E_k(\epsilon, 1) = E_{k-1}(\epsilon)$ to take into consideration the errors due to application of GENERATOR.

PROPOSITION 4.8. *If $\|A\|_\infty \geq 1$, and $A$ is NIM($\hat{\delta}$) and $\epsilon$-DD, then if $\|(I - \tilde{B}^{(0)}\tilde{A}^{(0)}\|_\infty < \epsilon$, we have $\|I - \tilde{B}^{(k)}\tilde{A}^{(k)}\|_\infty < E_k(\epsilon, s)$.*

**Proof:** Again, we use a proof by induction on $k$. The basis $k = 0$ holds by assumption, since $\epsilon = E_0(\epsilon, s)$. Suppose Proposition 4.8 holds for $k-1$, so $\|I - \tilde{B}^{(k-1)}\tilde{A}^{(k-1)}\|_\infty < E_{k-1}(\epsilon, s)$.

Since we use one step of exact Newton iteration per stage, by the quadratic convergence bounds of Subsection 3.5, we have (as in the proof of Proposition 3.11), $I - \hat{B}^{(k)}\tilde{A}^{(k)} = (I - \tilde{B}^{(k-1)}\tilde{A}^{(k)})^2$, and hence again $\epsilon_k \leq \|(I - \tilde{B}^{(k-1)}\tilde{A}^{(k)})\|_\infty^2$. Again, by assumption on the approximation of $A$ by the $\tilde{A}^{(k)}$, $\|\tilde{A}^{(k)} - \tilde{A}^{(k-1)}\|_\infty \leq \|A - \tilde{A}^{(k)}\|_\infty + \|A - \tilde{A}^{(k-1)}\|_\infty \leq E_k(\epsilon, 1) + E_{k-1}(\epsilon, 1)$.

By the induction hypothesis, $\|I - \tilde{B}^{(k-1)}\tilde{A}^{(k-1)}\|_\infty < E_{k-1}(\epsilon, s)$, and by Proposition 4.6, $\tilde{A}^{(k-1)}$ is strongly DD. So by Proposition 3.7, $\|\tilde{B}^{(k-1)}\|_\infty \leq 2$. Also,

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

so we have the bound: $\|I - \tilde{B}^{(k-1)}\tilde{A}^{(k)}\|_\infty$

$$\leq \|I - \tilde{B}^{(k-1)}\tilde{A}^{(k-1)}\|_\infty + \|\tilde{B}^{(k-1)}\|_\infty \cdot \|\tilde{A}^{(k-1)} - \tilde{A}^{(k)}\|_\infty$$

$$\leq E_{k-1}(\epsilon, s) + 2(E_k(\epsilon, 1) + E_{k-1}(\epsilon, 1)) \leq 4E_{k-1}(\epsilon, s) - 2E_k(\epsilon, s)$$

since $4E_k(\epsilon, s) \leq E_{k-1}(\epsilon, s)$. Also by the error bounds of Proposition 4.7, $\|I - \tilde{B}^{(k)}\tilde{A}^{(k)}\|_\infty \leq s\epsilon_k. = 1 - 1/n^{\Omega(1)}$. Thus, by the above bounds we get:

$$\|I - \tilde{B}^{(k)}\tilde{A}^{(k)}\|_\infty \leq s\epsilon_k + E_k(\epsilon, s) \leq s\|(I - \tilde{B}^{(k-1)}\tilde{A}^{(k)})\|_\infty^2 + E_k(\epsilon, s)$$

$$\leq s(4E_{k-1}(\epsilon, s) - 2E_k(\epsilon, s))^2 \leq E_k(\epsilon, s)$$

by definition of $E_k(\epsilon, s)$. ∎

Note that by definition of $\tilde{A}^{(k)}$ and $\tilde{B}^{(k)}$, these approximate Newton iterations require $O(\pi)$ bit precision.

Thus we have:

LEMMA 4.12. *If $A$ is NIM($\hat{\delta}$), then even with the above approximations to $A$ and the $\tilde{B}^{(k)}$, at most $k^+ = \lceil log(\bar{c}\pi)\rceil \leq O(\log n)$ stages suffice to compute, using bit precision $\leq O(\pi)$, the approximate inverse $\tilde{B}^{(k)}$ with error $2^{-\bar{c}\pi}$.*

**Note:** we can also show that Lemma 4.12 holds even for more moderate conditions on $A$, but the Lemma will suffice for our RF algorithm.

**Non-pipelined Newton Iterations for RFs of Bounded Displacement Rank.** Suppose we are given a NIM($\hat{\delta}$) $n \times n$ matrix $A$ of displacement rank $\delta$, with $mindiag(A) \geq 1$. We now provide a generalization of the non-pipelined Newton Iteration algorithm of Subsection 3.5 to the case of bounded displacement rank. Fix $d^* = \log n - \Omega(\log\log n)$. We will compute the RF of depth $d^*$ for $A$ within error $2^{-\Omega(\pi)}$, using bit precision $\leq O(\pi)$, by proceeding down the recursion tree in $\log n$ stages, without pipelining. For simplicity we will first describe a simple $O(\log^3 n)$ time, $\delta^2 P(n)$ processor algorithm for approximate RF without pipelining, specialized to the case of bounded displacement rank.

At each stage $d = 0, \ldots, d^*$, for all $\alpha$ of length $d$, we compute for all nodes of depth $d$ an approximation of $A_\alpha$ within error $2^{-\Omega(\pi)}$ and use $B^{(k)}(A_\alpha)$ to approximate $A_\alpha^{-1}$ within error $2^{-\Omega(\pi)}$ (using bit precision $\leq O(\pi)$). We use the algorithm given in the proof of Lemma 4.6 to construct displacement generators of the matrices at each stage. The stage at depth $d$ in the RF involves $O(\log n)$ iterations of multiplication of matrices of displacement rank $\delta$, so requires $O(\log^2 n)$ time for the required parallel matrix products for the $2^d$ nodes of depth $d$ as well as the parallel bounds stated in Lemma 4.6 for construction of the displacement generators of the matrices at each stage. The processor bounds are exactly the same as those described in the proof of Lemma 4.10 with $k = O(\log n)$, resulting in a total time bound of $O(\log^3 n)$ using $\delta^{\omega-1}n/(\log n\delta) + \delta^2 P(n) \leq O(\delta^2 P(n))$ processors. Thus, with an appropriate slowdown as described in the proof of Lemma 4.10, the $\log n$ stages require total parallel time $O(log^3 n)$ using $\delta^2 P(n)$ processors. Summarizing, we have shown that given a NIM($\hat{\delta}$) $n \times n$ matrix $A$ of displacement rank $\delta$, with $mindiag(A) \geq 1$, we can compute, using bit precision $\leq O(\pi)$, an approximation to the RF of depth $d^*$ for $A$ within error $2^{-\Omega(\pi)}$ in time $O(log^3 n)$ using $\delta^2 P(n)$ processors.

**Bounding Error Accumulation in Recursions of RFs of Bounded Displacement Rank.** We now show that error analysis and quadratic convergence of the approximate RF computation of Subsection 3.5 extends to the case where $A$ and the approximate inverse matrices of the RF are specialized to be of constant displacement rank. We prove a generalization of Lemma 3.5:

LEMMA 4.13. *For each $d = 1, \ldots, \log n$, and each $\alpha \in \{0,1\}^d$, the $\infty$-norm error for the i-th iteration is $\|A_\alpha^{(i)} - A_\alpha\|_\infty \leq E_i(\epsilon, s)$ and $\|I - B_\alpha^{(i)} A_\alpha\|_\infty \leq E_i(\epsilon, s)$.*

**Proof:** Lemma 4.13 is proved by induction. As in Lemma 3.4, in the Newton Iteration Algorithm for the RF of bounded displacement rank, at each depth $d = 1, \ldots, \log n$, and for each $\alpha \in \{0,1\}^d$, on the first stage of Newton iteration, the initial approximation of the RF matrices at depth $d$ have very small (with respect to $A$) $\infty$-norm error $\epsilon$. In particular, $\|A_\alpha^{(0)} - A_\alpha\|_\infty \leq \epsilon$ and $\|I - B_\alpha^{(0)} A_\alpha\|_\infty \leq \epsilon$. We use this as a basis for the induction. The inductive step follows directly from Lemma 4.12 applied to $\{A_\alpha^{(i)}\}$ and $\{\tilde{B}_\alpha^{(i)}\}$, which provides the required bounds on the errors of the $i$-th approximate RF $\{A_\alpha^{(i)}\}$. ∎

**Pipelined Newton Iterations for RFs of Bounded Displacement Rank.** We will again apply stream contraction to do pipelined Newton iterations for RFs of bounded displacement rank, as previously described in Subsection 3.5.

LEMMA 4.14. *Given a NIM($\hat{\delta}$) $n \times n$ matrix $A$ of displacement rank $\delta$ and with $mindiag(A) \geq 1$ we can compute, using bit precision $\leq O(\pi)$, an approximation to the RF of depth $d^*$ for $A$ within error $2^{-\Omega(\pi)}$ in parallel time $O(\log^2 n)$ using $\delta^2 P(n)$ processors.*

**Proof:** The pipelined algorithm for the approximate RF of $A$ described in Subsection 3.5 for a dense $A$ requires $O(\log n)$ iterations of simultaneous Newton iterations at every node of the tree. We can analyze the time and processor bounds of this pipelined algorithm for the approximate RF as we just did in Lemma 4.11 for the RF (mod $p^{2^k}$), as specialized to $A$ of displacement rank $\delta$. In particular, the approximate inverse matrices of the RF are also specialized to be of constant displacement rank. The asymptotic time and processor bounds are exactly as in Lemma 4.11 for $k = O(\log n)$, that is, in parallel time $O(\log^2 n)$ using $\delta^2 P(n)$ processors. The error bounds follow from Lemma 4.13. ▮

### 4.10. Analysis of the RF algorithm for bounded displacement rank

THEOREM 4.1. *Given an $n \times n$ SPD integer matrix $A$ of displacement rank $\delta$, our algorithm for the exact RF takes, with high likelihood $\geq 1 - 1/n^{\Omega(1)}$, parallel time $O(\log^2 n)$ using $\delta^2 P(n)$ randomized processors, and bit precision $O(\pi)$, using at most a linear number of random variables each ranging over a domain of size $(n\|A\|)^{O(1)}$.*

**Proof:** Fix as input an $n \times n$ integer nonsingular matrix $A$, of constant displacement rank $\delta$, and with integer entries of magnitude $\leq 2^\beta$, where $\beta \leq n^{O(1)}$. Our proof follows in large part from the proof of Theorem 3.1, except that here we have decreased processor bounds due to the bounded displacement rank of $A$. Therefore, we will only mention the modifications of the proof of Theorem 3.1 required to complete the proof of Theorem 4.1 (note in particular, again we need only bit precision $O(\pi)$).

Steps 1 and 2 clearly take $O(1)$ time using $n$ processors.

In step 3, we apply the approximate Newton Iteration of bounded displacement rank of Subsection 4.9, using bit precision $O(\pi)$, to compute a rational approximate RF tree of depth $d^* = \log n - \log\log n$, for $\bar{A}$ within accuracy $2^{-c\pi}$, for a sufficiently large positive constant $c$. Clearly $mindiag(\bar{A}) \geq 1$, so by Lemma 4.14, step 3 costs parallel time $O(\log^2 n)$ using $\delta^2 P(n)$ processors.

Step 4 is executed as follows: applying Proposition 2.6, (which states DETERMINANT has an efficient $O(\log n)$ time parallel reduction to RF using the recursive formulae), in time $O(\log n)$ using $\delta^2 P(n)$ processors, we can compute a rational approximation to $det(\bar{A}_\alpha)$ up to accuracy within $2^{-c\pi}$ (using bit precision $O(\pi)$) from the approximate $LU$ factorization of $\bar{A}_\alpha$. Next, using $\delta^2 P(n)$ processors, for each subtree rooted at depth $d^*$, we use known sequential $O(\log n)$ time algorithms to extend the computation of the rational approximation of RF sequences to a rational approximation of LU factorizations and determinants of all matrices from depth $d^*$ to depth $\log n$ of this approximate RF tree.

Recursive computation of the integer multipliers in step 5 and the multiplication and round off operations in steps 6, 7 and 8 clearly each cost at most parallel time $O(\log n)$ using $\delta^2 P(n)$ processors. The correctness of steps 5, 6, 7, and 8 again follows from the proof of Theorem 3.1.

The Newton-Hensel lifting of step 9 is done as described in Subsection 4.8. We do the Newton-Hensel lifting first as follows: **for** $i = 0, \ldots, k^* = \lceil \log(c\pi/\log p) \rceil$ compute the RF $\pmod{p^{2^i}}$ of $A$. Lemmas 4.10 and 4.11 imply that the cost to do this Newton-Hensel lifting by pipelining is time $O(\log^2 n)$ using $\delta^2 P(n)$ processors. By parallel use (for all the subtrees rooted at depth $d^*$) of known sequential algorithms for Newton-Hensel lifting, extend the computation of the RF $\pmod{p^{2^{k^*}}}$ of all matrices from depth $d^*$ to depth $\log n$.

The recursive computation in steps 10 and 11 of determinants of the RF matrices (mod $p^{2^{k^*}}$) and of the integer multipliers in step 12, and the exact RF in steps 13 costs at most parallel time $O(\log n)$ using $\delta^2 P(n)$ processors. The correctness of these steps 10-13 again follows from the proof of Theorem 3.1. Thus, by Lemmas 4.11 and 4.14, each major stage of the RF algorithm on the RF tree of depth $d^*$ takes at most time $O(\log^2 n)$ using $\delta^2 P(n)$ processors, with bit precision $O(\pi)$. Each matrix at depth $d^*$ of the exact RF tree for $A$ is of size $n_{d^*} \times n_{d^*}$ where $n_{d^*} = n/2^{d^*} = n/2^{\log n - \log\log n} = \log n$ and has displacement rank at most $\delta$. The known parallel algorithms [70, 64, 67, 68]) for the problems of inverse, determinant, and $LU$ factorization, for matrices of size $n_{d^*} \times n_{d^*}$ with displacement rank $\delta$, cost parallel time $O(\log^{O(1)} n_{d^*}) = O(\log^{O(1)} \log n)$ using at most $O(\delta^2 n_{d^*} P(n_{d^*}))$ $\leq O(\delta^2 \log^2 n) \log\log\log n$ processors, since $P(n_{d^*}) \leq O(n_{d^*} \log\log n_{d^*})$. By slow down by a factor of $O(\log n) \log\log\log n$, this can be done in time $O(\log^2 n)$ with reduced processor bound $O(\delta^2 \log n)$. There are $2^{d^*} = n/\log n$ nodes at depth $d^*$ of the RF tree for $A$. Thus the total cost of computing the exact RF sequences, LU factorizations, and determinants of all matrices at depth $d^*$ of this exact RF tree for $A$ is at most $O(\log^2 n)$ parallel time and $(n/\log n)\ O(\delta^2 \log n) \leq O(\delta^2 n)$ $\leq O(\delta^2 P(n))$ processors. Since there are only $O(1)$ such major stages, by slowing the time by a further constant factor, we require time $O(\log^2 n)$ using $\delta^2 P(n)$ processors. ∎

If the input matrix $A$ is not SPD, then we again apply the RF algorithm instead to the SPD $AA^T$, as described in Subsection 2.2 by use of the normal form reduction, which only at most squares the displacement rank.

## 5. PARALLEL RF COMPUTATION FOR MATRICES WITH SEPARABLE SPARSITY GRAPHS

Let $A$ be an $n \times n$ SPD matrix $A$ with an $s(n)$-separable sparsity graph (as defined in the introduction). Lipton, Rose, and Tarjan [53] and Pan and Reif [69, 73] define a nested dissection ordering $V_0, \ldots, V_D$ which is used to guide the Gaussian elimination process of the sparse matrix so as to minimize fill-in. The separation of the sparsity graph $G$ and recursive separation of its subgraphs defines a binary tree, known as the *separator tree* whose nodes are labeled with the induced separators. In particular, $G$ has an $s(n)$-separator $S$ whose deletion results in two disconnected subgraphs $G_1, G_2$ of size at most $2n/3$, so then the root of the separator tree is labeled with $S$ and each of the children of the root are labeled with the recursively defined separators of the subgraphs $G_1, G_2$, etc. The separator tree has depth $D \leq \log_{3/2} n$. For $d = 1, \ldots, D$ let $V_d$ be the union of all the separator nodes at depth $D - d$ in the separator tree. (Note in particular that $V_0$ is the union of all the separators at the leaves of the tree and $V_D$ is the separator $S$ of $G$.) We assume that the matrix has already been pre and post multiplied by a permutation matrix so that the resulting matrix $A$ has the rows and columns in this order.

Using this ordering $V_0, \ldots, V_D$ for sequential Gaussian elimination,

LEMMA 5.1. *(Lipton, Rose, and Tarjan [53]) Given an $n \times n$ SPD matrix $A$ with an $s(n)$-separable sparsity graph, a recursive LU factorization can be computed in sequential time $O(s(n)^3)$, exactly solving in the same time bound the problems DETERMINANT and LINEAR SYSTEM SOLVE.*

In the Parallel Nested Dissection algorithm of Pan and Reif [69, 73], this nested dissection ordering is specified by a vertex partition sequence $V_0,V_1,\ldots,V_D$, for $D \le \log_{3/2} n$ which is used to guide the parallel elimination process. Let $n_0 = n$ and let $n_{d+1} = n_d - |V_d|$ for $d = 0,\ldots,D$. Let $k_D = n$ and for $d < D$, $k_d \le \lfloor (2/3)k_{d+1} \rfloor$. (Note that $k_d$ is an upper bound on the number of vertices of the induced subgraph of $G$ whose associated separator is at depth $D - d$ in the separator tree.) Assuming an $s(n)$-separable sparsity graph (see Lipton, Rose, and Tarjan [53]), we define:

the **Nested Dissection(ND) Sequence Problem:** if possible, construct a sequence of matrices $A = A_0,\ A_1, A_2,\ \ldots,\ A_D$, where $D \le \log_{3/2} n$ and for $d = 0, 1, \ldots, D-1$, $A_d$ is an $n_d \times n_d$ matrix which is partitioned as

$$A_d = \begin{bmatrix} W_d & X_d \\ Y_d & Z_d \end{bmatrix}$$

where $n_{d+1} = n_d - |V_d|$ $W_d, X_d, Y_d, Z_d$ are matrices, $W_d$ is a block diagonal matrix of size $|V_d| \times |V_d|$ where each block is of size $s(k_d) \times s(k_d)$, $X_d$ is of size $|V_d| \times n_{d+1}$, $Y_d$ is of size $n_{d+1} \times |V_d|$ (if $A$ is symmetric, then $Y_d = (X_d)^T$), $Z_d$ is of size $n_{d+1} \times n_{d+1}$, and $A_{d+1} = Z_d - Y_d W_d^{-1} X_d$ is the *Schur complement*.

Note that Pan and Reif [69, 73] show that the norm and condition bounds of Proposition 2.2 hold for any ND RF sequence. Pan and Reif [69, 73] also show that an ND RF sequence can be computed in parallel time $O(\log^3 n)$ using $n + M(s(n))$ processors, also giving in the same time bounds solutions to the problems DETERMINANT and LINEAR SYSTEM SOLVE. The proofs in Pan and Reif [69, 73] show that the work for ND RF sequence is dominated by the computation of $O(|V_d|/s(k_d))$ products and inverses of a sequence of dense submatrices of size $s(k_d) \times s(k_d)$ each requiring $O(\log^2 n)$ time and $O((k_d + M(s(k_d)))|V_d|/s(k_d))$ processors for $d = 0,\ldots,D$, where $D \le \log_{3/2} n$. Thus the total time is $O(\log^3 n)$ and the processor bound is

$$\sum_{d=0}^{D} O((k_d + M(s(k_d)))|V_d|/s(k_d)) \le O(n + M(s(n))),$$

where $s(n)$ is of the form $n^\gamma$ for $\gamma > 0$.

We now derive a parallel algorithm in this case, and reduce the parallel time from $\Omega(\log^3 n)$ to $O(\log^2 n)$ while still using $n + M(s(n))$ processors. We use a modified form of our (balanced) RF algorithm on the appropriately permuted input matrix.

The partitioning of blocks is again altered depending on the separator structure, following the usual techniques used in the above ND RF sequence. Again the nested dissection ordering is specified by the vertex partition sequence $V_0, V_2, \ldots, V_D$, for $D \le \log_{3/2} n$ which is used to guide the parallel elimination process. Let $n_d$ be as defined above.

ND RF: If possible, compute a binary tree of depth $D = O(\log n)$ whose nodes are matrices. Each node at depth $d, 0 \le d \le D$ is a $n_\alpha \times n_\alpha$ matrix $A_\alpha$ where $\alpha \in \{0,1\}^d$ is a binary string of length $d$, and $n_\alpha$ is defined recursively below. The node is a leaf if $n_\alpha = 1$.

For each $d, 0 \le d \le D$, we specify the string $1^d$ to be *special*. For each $\alpha \in \{0,1\}^d$, if $\alpha$ is special and $n_\alpha > 1$, then we recursively decompose the matrix (using the

ND RF sequence), setting $n_{\alpha 0} = |V_d|$ and $n_{\alpha 1} = n_{d+1} = n_d - |V_d|$. Otherwise, if $\alpha \in \{0,1\}^d$ is not special but $n_\alpha > 1$, then we recursively decompose the matrix evenly (using the RF defined at the start of this paper). Let $n_{\alpha 1} = \lfloor n_\alpha/2 \rfloor$, $n_{\alpha 0} = \lceil n_\alpha/2 \rceil$. If $d = 0$ then $A_\alpha = A$ is the root of the tree and $\alpha$ is the empty string. For $0 \le d < D$, each matrix $A_\alpha$ at depth $d$ with $n_\alpha > 1$ has exactly two children in the tree, $A_{\alpha 1}$ and $A_{\alpha 0}$ at depth $d+1$ which will be defined by recursion. In particular, for $d = 0, 1, \ldots, D-1$, $A_\alpha$ is an $n_\alpha \times n_\alpha$ matrix which is partitioned as

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

where $A_{\alpha 0}, X_\alpha, Y_\alpha, Z_\alpha$ are matrices, $A_{\alpha 0} = W_d$ is of size $n_{\alpha 0} \times n_{\alpha 0}$, (where if $\alpha = 1^d$ then $n_{\alpha 0} = |V_d|$ and $A_{\alpha 0}$ is a block diagonal matrix of size $|V_d| \times |V_d|$ with each block of size $s(k_d) \times s(k_d)$), $X_\alpha$ is of size $n_{\alpha 0} \times n_{\alpha 1}$, $Y_\alpha$ is of size $n_{\alpha 1} \times n_{\alpha 0}$ (if $A$ is symmetric, then $Y_\alpha = (X_\alpha)^T$), $Z_\alpha$ is of size $n_{\alpha 1} \times n_{\alpha 1}$, and $A_{\alpha 1} = Z_\alpha - Y_\alpha A_{\alpha 0}^{-1} X_\alpha$ is the *Schur complement*.

Note: The above ND RF of $A$ is defined to be very similar to the ND sequence $A = A_0, A_1, A_2, \ldots, A_D$. In particular, $A_d = A_{1^d}$ for $d = 1, \ldots, D$. The only difference is that the ND RF also recursively factors the $W_d = A_{1^d 0}$ matrices appearing in the ND RF sequence. This takes $O(\log^2 n)$ time using $n + M(s(n))$ processors. Since this is an $s(k_d)$-block diagonal matrix of size $|V_d| \times |V_d|$ with each block of size $s(k_d) \times s(k_d)$), the processor bound is $O((k_d + M(s(k_d)))|V_d|/s(k_d)) \le O(n + M(s(n)))$. These are recursively factored evenly (rather than use the separator structure), using the (balanced) RF defined at the start of this paper. The results of Pan and Reif [69, 73] imply that the norm and condition bounds of proposition 2.4 hold also for any ND RF.

We now apply our RF algorithm as previously described in Section 3, only modified to execute on the $A_\alpha$ of the ND RF. Note that the depth of the ND RF tree is at most $\log_{3/2} n$, which is at most a factor of $1/\log(3/2)$ more than the depth of the previously defined RF tree, so in the modified algorithm the iterator $d$ must range up to $\log_{3/2} n$. Also note that the integer multipliers $m_\alpha$ are computed in $1 + 2$ (depth of the ND RF tree) stages, which is $1 + 2\log_{3/2} n$ in this case. The error analysis of the modified RF algorithm is similar, except that again we need to replace $\log n$ with $\log_{3/2} n$ in a number of places due to the constant factor increase in the depth of the ND RF tree.

We again use the exact same pipelining technique used in Subsection 3.4 to decrease the parallel time bounds from $O(\log^3 n)$ to $O(\log^2 n)$. We use both the analysis of (balanced) RF defined at the start of this paper as well as the analysis of ND RF sequence defined in Pan and Reif [69, 73]. In particular, the proof of our $n + M(s(n))$ processor bounds follows exactly from the results of Pan and Reif [69, 73]. Again in this sparse $s(n)$-separable case the work for ND RF is dominated by the computation of products and inverses of a sequence of dense submatrices of size $s(k_d) \times s(k_d)$ each requiring $O(\log^2 n)$ time and $O(k_d + M(s(k_d)))$ processors for $d = 0, \ldots, D$, where $D \le \log_{3/2} n$. However, due to our use of pipelining, all these inverses are computed simultaneously, so the total time is $O(\log^2 n)$ while the processor bound remains $\sum_{d=0}^{D} O(k_d + M(s(k_d)))$. This sum is $O(n + M(s(n)))$ if $s(n)$ is of the form $n^\gamma$ for $0 < \gamma < 1$. Note that if the sparsity graph of $A$ has constant degree or is planar, then the sparsity graph of symmetric positive definite $A^T A$ still has separator bound $O(s(n))$.

THEOREM 5.1. *Let $A$ be an $n \times n$ SPD matrix with an $s(n)$-separable sparsity graph, where $s(n)$ is of the form $n^\gamma$ for $0 < \gamma < 1$. If $A$ is nonsingular, then an ND RF can be computed in parallel time $O(\log^2 n)$ using $n+M(s(n))$ processors, and bit precision $O(\pi)$, where $\pi = O(n(\beta+\log n))$, with high likelihood $\geq 1-1/n^{\Omega(1)}$ (using a constant number of random variables ranging over a domain of size $(n\|A\|)^{O(1)}$).*

## 6. APPLICATIONS OF THE RF

### 6.1. Reduction of Matrix Problems to RF Computation

We will consider a reduction to be an *efficient parallel reduction* if it can be done in $O(\log^2 n)$ parallel time using $M(n)$ processors. In this Subsection we define various matrix problems and their efficient parallel reduction to RF computation (also, see Pan, p.69 [63]).

1. *LU FACTORIZATION:* If possible, factor $A = LU$ where $L$ is nonsingular lower triangular, and $U$ is nonsingular upper triangular, otherwise output NO LU FACTORIZATION. (If $A$ is symmetric, then $U = L^T$ and this problem is known as *CHOLESKY FACTORIZATION.*)

If $A$ is SPD, then $A$ always has a $LU$ factorization.

Given a RF tree of $A$, then the $LU$ factorization can be computed by $O(\log n)$ stages of matrix multiplication using the above recursive formula.

LEMMA 6.1. *There is an efficient parallel reduction from LU FACTORIZATION to the RF problem.*

2. *DETERMINANT:* Compute $det(A)$.

If $A$ has a factorization $A = LU$, then $det(A) = det(L)det(U)$. The determinant of a triangular matrix is obtained by multiplying all the elements on its principal diagonal. (This can be computed in $O(\log n)$ time and $n/\log n$ processors by using a balanced binary multiplication tree of size $O(n/\log n)$ whose leaves have $\log n$ elements each.) So $det(L) = \prod_{i=1}^n L_{ii}$ and $det(U) = \prod_{i=1}^n U_{ii}$.

3. $|$*DETERMINANT*$|$*:* Compute the magnitude $|det(A)|$ of $det(A)$.

Otherwise, $A^T A$ is SPD, and so has a factorization $A^T A = LU$. Then since $det(A) = det(A^T)$, we have $det(A)^2 = det(A^T A) = det(L)det(U)$, so $|det(A)| = \sqrt{det(L)det(U)}$.

LEMMA 6.2. *There is an efficient parallel reduction from $|$DETERMINANT$|$ (or from DETERMINANT if $A$ is SPD) to the RF problem costing $O(\log n)$ time and $n$ processors.*

4. *INVERSE:* If $A$ is nonsingular, then compute $A^{-1} = \frac{adj(A)}{det(A)}$, where $adj(A)$ is the adjoint matrix of $A$, otherwise output SINGULAR.

If $A$ has an RF, then $A^{-1}$ can be computed by the above RF sequence or tree formula. Otherwise, $A^T A$ is SPD, so if $A$ is nonsingular then $A$ has an RF. Then $(A^T A)^{-1} = (A)^{-1}(A^T)^{-1}$, can be computed by the above RF tree formula, and we can compute $A^{-1} = A((A)^{-1}(A^T)^{-1}) = A(A^T A)^{-1}$ by one more matrix product.

LEMMA 6.3. *INVERSE has an efficient parallel reduction to the RF problem.*

5. *LINEAR SYSTEM SOLVE:* If $A$ is nonsingular, then compute $A^{-1}v$, otherwise output SINGULAR.

Here we can apply the efficient parallel reduction given in our Subsection 1.2 to LU factorization of $AA^T$ (which we can compute by Lemma 6.1 by efficient parallel reduction to the RF), and to solution of triangular linear systems, for which there are known efficient parallel algorithms (see JáJá [42]).

LEMMA 6.4. *There is an efficient parallel reduction from LINEAR SYSTEM SOLVE to the RF problem.*

6. *QR FACTORIZATION:* If possible, factor $m \times n$ matrix $A = QR$ where $R$ is a nonsingular upper triangular matrix and $Q$ is an orthogonal matrix (so $Q^T Q = I$) of size $m \times n$, for $m \geq n$. If the $QR$ factorization is not possible, then output NO QR FACTORIZATION (Rank deficient).

The $QR$-factors of $A$ can be computed from the $LU$-factors of $A^T A$, where $R = U$ and $Q = AR^{-1}$.

LEMMA 6.5. *There is an efficient parallel reduction from QR FACTORIZATION to the RF problem.*

7. *HESSENBERG REDUCTION:* Compute a matrix $H = Q^T AQ$ having upper Hessenberg form $H_{ij} = 0$ if $i > j+1$, and compute $Q$, which is an orthogonal matrix. If $A$ is symmetric, then $H$ is tridiagonal.

The *Krylov matrix* of an $n \times n$ matrix $A$ and $n$-vector $v$, is an $n \times m$ matrix $K(A, v) = (v, Av, A^2 v, \ldots, A^{m-1} v)$. Borodin and Munro ([17]p.128) describe a well known algorithm for the Krylov matrix which can be used in the generic case where the minimal polynomial is the characteristic polynomial. Their algorithm requires $2 \log m$ multiplications of matrices of size at most $n \times \max(n, m)$. Their reduction is a Las Vegas randomized type of reduction. Using a random, independent choice of the elements of $n$-vector $v$ over a fixed set of polynomial size, the Schwartz-Zippel Lemma [84, 80] insures a failure probability $\leq 1/n^{\Omega(1)}$. They observe that the matrix powers $A, A^2, \ldots, A^{2^{\lfloor \log m \rfloor}}$ can be computed in $\lfloor \log m \rfloor$ stages of matrix products, and that $K(A, v)$ can be computed in $\log m$ further stages, where using the identity for $i = 1, \ldots, \lfloor \log m \rfloor$, $(A^{2^i} v, Av, A^2 v, \ldots, A^{2^{i+1}} v) = A^{2^i}(v, Av, A^2 v, \ldots, A^{2^i} v)$. Since

$A^k v$ in general converges to the eigenvector of $A$ with the largest magnitude eigenvalue, the Krylov matrix $K(A, v)$ is in general nearly singular. Thus this reduction to Hessenberg form, also used by Pan [63], must be used with care.

**Further Note:** Borodin and Munro's reduction fails [**?**, **?**] whenever A is a matrix whose minimal polynomial is a proper divisor of its characteristic polynomial, since the Krylov matrix $K(A, v)$ will be singular, for every choice of the vector $v$, in this case. If $K(A, v)$ is nonsingular with $QR$ factorization, $K(A, v) = QR$, then $H = Q^T AQ$ is in upper Hessenberg form.

LEMMA 6.6. *There is an efficient Las Vegas randomized parallel reduction (using n random numbers chosen from a fixed set of polynomial size), with success probability* $\geq 1-1/n^{\Omega(1)}$ *from HESSENBERG REDUCTION to the RF problem in the generic case where the minimal polynomial is the characteristic polynomial.*

### 6.2. A Randomized Reduction from RANK to RF

In this section we give an efficient parallel algorithm for RANK via the RF computation and a known randomized preconditioning method, but avoiding the usual computation of characteristic polynomials.

Borodin et al. ([16]) give a processor-efficient randomized parallel algorithm for the rank problem and the more general problem of the solution of singular linear systems. They give a randomized reduction from the solution of singular linear systems to the problems of (i) inverting a non-singular matrix and (ii) determining the rank of a matrix that has the added property that all its leading principal sub-matrices, of dimension no larger than its rank, are non-singular.

Consider an $n \times n$ singular matrix $A$. To avoid the problem where principal submatrices of $A$ have determinant 0, the results of [49] can be used to construct a matrix from $A$ which is expected to act like a generic matrix. Define a product matrix $A' = UAL$ with random $n \times n$ non-singular preconditioning multipliers $L, U$. The matrices $L$ and $U^T$ are unit lower triangular $n \times n$ Toeplitz matrices, with unit diagonal elements and with the strictly lower elements of the first column randomly and independently chosen over the values $\{i/n^c|$ where $i$ is an integer on the range from $-n^c$ to $n^c\}$ for a constant $c \geq 5$. For example, matrix $L$ is defined as follows:

$$L = \begin{cases} L_{i,j} = L_{1+(i-j),1} & \text{if } i > j \\ L_{i,j} = 1 & \text{if } i = j \\ L_{i,j} = 0 & \text{otherwise,} \end{cases}$$

where $L_{2,1}, L_{3,1}, \ldots, L_{n,1}$ are $(n-1)$ random numbers from this range. Then by Lemma 7.1, $Prob(cond_2(L) \geq n^c) \leq O(1/n^{c-4})$ and $Prob(cond_2(U^T) \geq n^c) \leq O(1/n^{c-4})$, so matrices $L, U$ are well conditioned, with condition number $\leq n^c$ with likelihood $\geq 1 - \Omega(1/n^{c-4})$.

Let $A'([r], [r])$ be the leading principal $r \times r$ submatrix of $A'$, (i.e., the submatrix of $A'$ indexed by rows $1, \ldots, r$ and by columns $1, \ldots, r$).

PROPOSITION 6.1. *Assuming $A$ has rank $r$, then with probability* $\geq 1 - 1/n^{\Omega(1)}$, *the rank of $A'$ is $r$ and moreover, $A'([r], [r])$ is positive definite and non-singular.*

This Proposition follows from Theorem 2 of [49], noting that the random elements of $U, L$ are chosen over over a fixed set of polynomial size, so the Schwartz-Zippel Lemma [84, 80] insures a failure probability $\leq 1/n^{\Omega(1)}$.

This randomized method is known to be *Las Vegas* (that is, the output can always be verified to be correct). The determined rank and the solvability of $Ax = b$ can be certified, as follows. Let superscripts to $0, 1$ indicate the size of matrices filled with $0, 1$ respectively. We have

$$A' = \begin{bmatrix} A'([r],[r]) & B \\ B' & B'' \end{bmatrix}.$$

Where $B$ is $r \times (n-r)$, $B'$ is $(n-r) \times r$, and $B''$ is $(n-r) \times (n-r)$. Let

$$E = \begin{bmatrix} I_r & -A'([r],[r])^{-1}B \\ 0^{(n-r)\times r} & I_{n-r} \end{bmatrix}.$$

If $A$ has rank $r$ then (see also [49] and [48], p 720)

$$A'E = \begin{bmatrix} A'([r],[r]) & 0^{r\times(n-r)} \\ B' & 0^{(n-r)\times(n-r)} \end{bmatrix},$$

and the right null space of $A$ is spanned by the columns of

$$LE \begin{bmatrix} 0^{r\times(n-r)} \\ I_{n-r} \end{bmatrix}.$$

If $b$ is a vector such that $Ax = b$ holds, and $b'$ is the vector formed by the first $r$ entries of $Ub$, then

$$ALE \begin{bmatrix} A'([r],[r])^{-1}b' \\ 0^{n-r} \end{bmatrix} = b,$$

thus certifying the determined rank and the solvability of $Ax = b$. The RANK problem requires that we actually find the rank $r$, not just verify that the rank is $r$. The RANK problem can be solved by using $O(\log n)$ stages of binary search for $r$, increasing the time bound by a logarithmic factor. This proves the known result:

LEMMA 6.7. *The RANK problem on a given matrix $A$ and the solvability of $Ax = b$ can be determined by a randomized parallel algorithm, using $2n$ random numbers over $\{1, \ldots, n^{O(1)}\}$, and with success probability $\geq 1 - 1/n^{\Omega(1)}$, using $O(\log n)$ calls to DETERMINANT of matrices derived from $A$ by multiplying $A$ by two triangular Toeplitz matrices.*

However we can do better. We now give a method that avoids this slow-down without utilizing asymptotically more processors (note [48] get similar results, but require the computation of characteristic polynomials). Suppose we are given an $n \times n$ matrix $A$ of rank $r < n$. Let $A' = UAL$ be the product matrix derived by multiplying $A$ with random $n \times n$ non-singular preconditioning multipliers $U, L$, as defined above. By Proposition 6.1, $A'([r],[r])$, the leading principal nonsingular $r \times r$ submatrix of $A'$, is nonsingular with probability $\geq 1 - 1/n^{\Omega(1)}$. Let us assume that this event holds, and $A'([r],[r])$ is positive definite. Note that $A'$ as defined above may not be symmetric. In Section 3, (RF Algorithm of Theorem 3.1) we note that the RF will still be constructed for inputs which are non-symmetric positive definite matrices, so symmetry of the input matrix is not essential. Alternatively, it is easy to show that Proposition 6.1 holds also if $U = L^T$ where again $L$ is a random unit lower triangular $n \times n$ Toeplitz matrix, so $A' = UAL$ is symmetric.

A more significant difficulty is that the RF algorithm on input matrix which is singular will not result in a complete RF. Since $A'$ is singular, if we attempt to construct the RF $\{A'_\alpha\}$ of $A'$, then the RF of $A'$ will not be complete, and instead the RF will be defined only on a submatrix of $A'$, in particular the leading principal nonsingular $r \times r$ submatrix $A'([r],[r])$.

Thus the rank $r$ of $A'$ can be found by examining the incomplete RF, and determining the completed portion of the RF corresponding to $A'([r],[r])$. Note that the leaves of the RF are $1 \times 1$ matrices $A'_\alpha$ where $\alpha \in \{0,1\}^{\log n}$, and so $A'_\alpha$ is nonsingular iff $A'_\alpha \neq 0$. Let NUMBER$(\alpha)$ be the binary number corresponding to the binary string $\alpha$. Note that the leaves are indexed by binary numbers $\alpha$ of length $\log n$, and that the completed portion of the RF corresponding to $A'([r],[r])$ will have the nonzero leaves $A'_\alpha$ for all indices $\alpha \in \{0,1\}^{\log n}$, such that NUMBER$(\alpha) \in \{0,1,\ldots,r-1\}$. These nonzero leaves are $1 \times 1$ non-singular matrices which are derived from the factored submatrices of the matrix $A'([r],[r])$. If $r < n$, then the next larger leading principal submatrix $A'([r+1],[r+1])$ is singular. Thus it follows that there is a zero leaf $A'_{\alpha^*} = 0$, for $\alpha^*$ such that NUMBER$(\alpha^*) = r$ (since otherwise if $A'_{\alpha^*} \neq 0$, then the next larger leading principal submatrix $A'([r+1],[r+1])$ would be nonsingular, so $A$ would have rank $r+1$, a contradiction). Thus we have:

LEMMA 6.8. *Let $\alpha^* \in \{0,1\}^{\log n}$ be the lexically smallest binary string of length $\log n$ such that $A'_\alpha = 0$. Then $r =$NUMBER$(\alpha^*)$.*

Summarizing the discussion above, given a matrix $A'$ which is singular, the RF Algorithm of Theorem 3.1 extends to allow us to construct the maximal partial RF of the matrix corresponding to the leading principal nonsingular submatrix $A'([r],[r])$ of the matrix $A'$. Thus by Lemma 6.8, and Theorem 3.1, we have:

THEOREM 6.1. *There is an efficient randomized parallel reduction from RANK to the RF problem. The RANK problem can be exactly solved in randomized parallel time, $O(\log^2 n)$ using $M(n)$ processes, using $2n$ random numbers over $\{1,\ldots,n^{O(1)}\}$, and with success probability $\geq 1 - 1/n^{\Omega(1)}$ and a constant number of random variables ranging over a domain of size $(n\|A\|)^{O(1)}$.*

### 6.3. Applications of the Matrix Reductions to Dense Matrices

By Theorem 3.1 and the efficient parallel reductions of Subsection 6.1, we have the following further results:

COROLLARY 6.1. *The problems (defined in Subsection 6.1) |DETERMINANT|, INVERSE, LINEAR SYSTEM SOLVE, RANK, QR FACTORIZATION and HESSENBERG REDUCTION can be exactly solved, with high likelihood $\geq 1 - 1/n^{\Omega(1)}$,* *in parallel time $O(\log^2 n)$ using $M(n)$ randomized processors, and bit precision $O(\pi)$, where $\pi = O(n(\beta + \log n))$. If the input matrix is SPD, we can also compute $DETERMINANT$ and LU FACTORIZATION, within these same parallel bounds. All of these problems require only a constant number of random variables ranging over a domain of size $(n\|A\|)^{O(1)}$, except RANK and HESSENBERG REDUCTION which require a further linear number of random variables ranging over a domain of size $n^{O(1)}$.*

**Note:** An input matrix which is nonsingular may not have a complete RF factorization. However, the RF will still be constructed for inputs which are nonsymmetric positive definite matrices, so symmetry of the input matrix is not essential (it just simplifies the proof in Lemma 3.6 of the pipelined Newton iterations described in Subsection 3.5).

Note that our parallel time bounds for the problems LU FACTORIZATION, QR FACTORIZATION and HESSENBERG REDUCTION are the best known. Previously Pan [63] had proved the same processor bounds for these listed problems with a time bound of $\Omega(\log^3 n)$.

The additional problems |DETERMINANT|, INVERSE, LINEAR SYSTEM SOLVE, and RANK were previously known to be exactly solvable (see [62, 63, 47, 48, 42]) in time $O(\log^2 n)$ using $M(n)$ processors, requiring reduction to the computation of the characteristic polynomial. Our techniques achieve the same bounds, with high likelihood $\geq 1 - 1/n^{\Omega(1)}$ (using again a constant number of random variables ranging over a domain of size $(n\|A\|)^{O(1)}$, except the rank computation which requires a further linear number of random variables ranging over a domain of size $n^{O(1)}$), without the need to compute the characteristic polynomial.

### 6.4. Applications of the Matrix Reductions to Sparse Matrices

Applying Theorem 5.1 and the above matrix reductions on sparse matrices, we have:

COROLLARY 6.2. *Let $A$ be an $n \times n$ SPD matrix with an $s(n)$-separable sparsity graph, where $s(n)$ is of the form $n^\gamma$ for $0 < \gamma < 1$. We can exactly solve the problems DETERMINANT and LINEAR SYSTEM SOLVE for a nonsingular $A$ in parallel time $O(\log^2 n)$ using $n + M(s(n))$ processors, and bit precision $O(\pi)$, where $\pi = O(n(\beta + \log n))$, with high likelihood $\geq 1 - 1/n^{\Omega(1)}$.*

### 6.5. Applications of the Matrix Reductions to Bounded Displacement Rank Matrices

By the reductions of Subsection 6.1, specialized to the case of bounded displacement rank, and the efficient algorithms for bounded displacement rank of section 2, we have:

COROLLARY 6.3. *Given an $n \times n$ integer matrix $A$ of displacement rank $\delta$, let $\delta' = \delta$ if $A$ is SPD, and else let $\delta' = \delta^2$. Then the problems $|DETERMINANT|$, INVERSE, LINEAR SYSTEM SOLVE, RANK can be exactly solved, with high likelihood $\geq 1 - 1/n^{\Omega(1)}$, in randomized parallel time $O(\log^2 n)$ using $\delta^2 P(n)$ processors, and bit precision $O(\pi)$, where $\pi = O(n(\beta + \log n))$, using at most a linear* *number of random variables each ranging over a domain of size $(n\|A\|)^{O(1)}$. If the input matrix is also SPD, we can compute DETERMINANT and LU FACTORIZATION, within these same parallel bounds.*

**The RANK problem for Bounded Displacement Rank Matrices.** Suppose we are given an $n \times n$ symmetric matrix $A$ of displacement rank $\delta$ which is singular and of rank $r < n$. Following the methods of Subsection 6.2, we choose random $n \times n$ non-singular lower triangular Toeplitz matrices $L, U^T$ with unit diagonal elements and with the strictly lower elements of the first column randomly chosen uniformly over the values $\{i/n^c|$ where $i$ is an integer on the range from $-n^c$ to $n^c\}$ for a constant $c \geq 5$. By Lemma 7.1, matrices $L, U$ are well conditioned, with condition number $\leq n^c$ with likelihood $\geq 1 - \Omega(1/n^{c-4})$. By Proposition 6.1, with likelihood $\geq 1 - 1/n^{\Omega(1)}$, $A' = UAL$ has a nonsingular $r \times r$ leading principal submatrix. Since multiplication of a matrix $A$ of displacement rank $\delta$ by these two triangular Toeplitz matrices results in a matrix of displacement rank at most $\delta$, it follows that $A' = UAL$ has this same displacement rank. Given a matrix $A'$ which is not positive definite, the RF Bounded Displacement Rank Algorithm of Theorem 4.1 easily extends, by the techniques of Subsection 6.2, to allow us to construct, in parallel time $O(\log^2 n)$ using $\delta^2 P(n)$ processors, the maximal partial RF of depth $d^*$ for the matrix corresponding to the leading principal nonsingular submatrix of the matrix $A'$.

Corollary 6.4. *The RANK problem for $n \times n$ symmetric matrices of displacement rank $\delta$ can be solved, with success probability $\geq 1 - 1/n^{\Omega(1)}$, in randomized parallel time $O(\log^2 n)$ using $\delta^2 P(n)$ processors.*

### 6.6. Applications to Polynomial Problems

The problems of computing the resultant and the greatest common denominator (GCD) of two polynomials is an important application of Toeplitz matrices.

**Parallel Computation of Polynomial Resultant.** Fix polynomials $A(x) = \sum_{i=0}^{n} a_i x^i$ and $B(x) = \sum_{i=0}^{m} b_i x^i$. A **Sylvester** or **resultant matrix** $S(A, B)$ of polynomials $A(x)$ and $B(x)$ is a size $(m + n) \times (m + n)$ matrix which is a block Toeplitz matrix, consisting of two block submatrices, of size $(m + n) \times m$ and

$(m+n) \times n$ each of which is Toeplitz. It is defined as follows:

$$S(A,B) = \begin{bmatrix} a_n & & & b_m & & \\ a_{n-1} & \ddots & & b_{m-1} & \ddots & \\ \vdots & \ddots & & \vdots & \ddots & \\ & & a_n & & & b_m \\ & & a_{n-1} & & & b_{m-1} \\ & & \vdots & & & \vdots \\ a_1 & & & b_1 & & \\ a_0 & \ddots & & b_0 & \ddots & \\ & \ddots & & & \ddots & \\ & & a_1 & & & b_1 \\ & & a_0 & & & b_0 \end{bmatrix}.$$

The **univariate resultant** (see [19, 16]) of $A(x)$ and $B(x)$ is the determinant of $S(A,B)$. It is well-known that the resultant is equal to 0 iff $A(x)$ and $B(x)$ have a common root. The resultant has many applications in computational algebra.

Though the Sylvester matrix $S(A,B)$ as a whole will not be Toeplitz, it consists of two block submatrices, each of which is Toeplitz (see JáJá [42]). So by Lemma 4.4, the Sylvester matrix will have constant displacement rank 3. Hence, the processor requirements and parallel time for computing the resultant is the same bounds as constructing the RF of a matrix bounded by displacement rank 3. Thus, by Theorem 4.1 and Corollary 6.3 and the partial RF computation of rank of Corollary 6.4, this implies:

COROLLARY 6.5. *The resultant problem for univariate polynomials with P-rational coefficients can be solved in randomized parallel time* $O(\log^2 n)$ *using* $P(n)$ *processors, with success probability* $\geq 1 - 1/n^{\Omega(1)}$.

**Parallel Computation of Polynomial Greatest Common Divisor.** Another application of Toeplitz block matrices occurs in the *GCD computation problem:* Again fix $A(x) = \sum_{i=0}^{n} a_i x^i$ and $B(x) = \sum_{i=0}^{m} b_i x^i$, where w.l.o.g., $m \leq n$. For uniqueness we define the GCD of $A(x)$ and $B(x)$ to be a monic (the coefficient of the highest degree term of $GCD(x)$ is 1) polynomial $GCD(A(x), B(x)) = GCD(x)$ which is the unique maximum-degree polynomial that divides both $A(x)$ and $B(x)$. Let $r$ be the degree of $GCD(x)$. The GCD satisfies the equation $GCD(x) = U(x)A(x) + V(x)B(x)$, for unique polynomials $U(x), V(x)$, where $U$ has degree $m-r-1$ and $V$ has degree $n-r-1$. The *extended GCD computation problem* is to compute $U(x), V(x)$ in addition to $GCD(x)$. The extended GCD computation can be done sequentially by the Extended Euclidean Algorithm ([1]) in $O(P(n)\log^2 n)$ time, but there were no previous similarly efficient parallel algorithms.

We now develop our parallel algorithms for GCD or extended GCD. We observe that the processor requirements and parallel time for computing polynomial GCD's of output degree $r$ is the same as that for finding the rank $r$ and the basis of a matrix of displacement rank 3 (see [22]). The extended polynomial GCD defines a submatrix of the Sylvester matrix. Let the $r$**-subresultant matrix** (see also Brown and Traub [19] and JáJá [42]) be a matrix of size $(n+m-2r)\times(n+m-2r)$, consisting of the first $n+m-2r$ rows of the Sylvester matrix defined for the resultant of $A(x)$ and $B(x)$. Consider the $r$-subresultant system defined for the $r$-subresultant matrix:

$$\begin{bmatrix} a_n & & & b_m & & \\ a_{n-1} & & & b_{m-1} & & \\ \vdots & & & \vdots & & \\ & \ddots & & & \ddots & \\ \vdots & \ddots & \ddots & \vdots & \ddots & \ddots \\ a_0 & & a_n & & & b_m \\ & & a_{n-1} & b_0 & & b_{m-1} \\ & & \vdots & & & \vdots \\ & a_0 \ \dots & a_r & & b_0 \ \dots & b_r \end{bmatrix} \begin{bmatrix} u_{m-r-1} \\ \vdots \\ \vdots \\ u_1 \\ u_0 \\ v_{n-r-1} \\ \vdots \\ \vdots \\ v_1 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ . \\ . \\ . \\ . \\ . \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

Again, by Lemma 4.4, the subresultant matrix will have constant displacement rank 3. The above $r$-subresultant system corresponds to the linear system derived from the equation: $GCD(x) = U(x)A(x) + V(x)B(x)$. The $r$-subresultant system has a solution if $r \geq deg(GCD(x))$. Thus we can use divide and conquer, in $O(\log n)$ stages, to find the degree $r$ of the GCD. Hence, the processor requirements and parallel time for computing polynomial GCD's of output degree $r$ is bounded by $O(\log n)$ stages of testing the singularity of $n \times n$ matrices of displacement rank 3 and a final stage of finding the and the basis of such a matrix. For bounded displacement rank matrices, we apply Lemma 4.4, Corollary 6.3 and the partial RF computation of rank used in Corollary 6.4. This implies:

COROLLARY 6.6. *The GCD problem for univariate polynomials with P-rational coefficients can be solved in randomized parallel time $O(\log^3 n)$ using $P(n)$ processors, with success probability $\geq 1 - 1/n^{\Omega(1)}$.*

**Padé approximants.** We next observe our parallel algorithms for extended GCD (Corollary 6.6) can be applied to derive an efficient parallel algorithm for finding Padé approximants. Consider a (formal) power series $A(x) = a_0 + a_1 x + a_2 x^2 + \dots$ with $a_0 \neq 0$. A Padé approximant is a rational function approximation to the power series $A(x)$. The $(m, n)$ *Padé approximant* to $A$ is defined (see [36]) to be the rational function $P_{mn}(x) = U(x)/V(x)$, where $U$ and $V$ are polynomials with $\deg(U) \leq m$ and $\deg(V) \leq n$ and $A(x)V(x) - U(x) = O(x^{m+n+1})$. Padé approximants are widely used in applications for rational function approximation. Frobenius [30] first defined Padé approximants and proved:

LEMMA 6.9. *The $(m, n)$ Padé approximant $P_{mn}(x) = U(x)/V(x)$ to $A$ is unique.*

Padé [61] popularized the use of Padé approximants. Gragg [36] gave quadratic time algorithms for Padé approximants. Brent, Gustavson, and Yun [18] developed an efficient sequential $O(P(n+m)\log^2(n+m))$ algorithm for Padé approximants, using a reduction to the solution of Toeplitz systems. The *Padé table* [36] is a two-dimensional array where the $(m, n)$ Padé approximant $P_{mn}(x)$ is in the position indexed by $(m, n)$. Brent, Gustavson, and Yun [18] show that the entries of the Padé table can be computed by the Extended Euclidean Algorithm ([1]).

Let $A(x)$ be a power series with $a_0 \neq 0$. Rational form $U(x)/V(x)$ is an $(m, n)$ Padé approximant to $A(x)$ if $A(x)V(x) - U(x) = O(x^{m+n+1})$. Note that this equation $A(x)V(x) - U(x) = O(x^{m+n+1})$ defines a linear system of $m + n + 1$ equations (each corresponding to a power of $x$) with $m + n + 2$ unknowns $v_i, i = 0, \ldots, n$, corresponding to $V(x) = \sum_{i=0}^{n} v_i x^i$ and $u_i, i = 0, \ldots, m$, corresponding to $U(x) = \sum_{i=0}^{m} u_i x^i$ where $a_v = 0$ if $v < 0$. These equations have the form:

$$\begin{bmatrix} a_0 & & \\ a_1 & a_0 & \\ \vdots & & \\ a_m & & a_{m-n} \\ \vdots & \ddots & \vdots \\ a_{m+n} & & a_m \end{bmatrix} \begin{bmatrix} v_0 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_0 \\ \vdots \\ u_m \\ 0 \\ . \\ 0 \end{bmatrix}$$

Observe that a rearrangement of this linear system gives the linear system:

$$\begin{bmatrix} a_m & a_{m+n-2} & a_{m+n-1} & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m-n+1} & a_{m-1} & a_m & 0 & 0 \\ a_{m-n} & a_{m-2} & a_{m-1} & -1 & \\ & a_0 & \vdots & \ddots & \\ & 0 & a_0 & & \ddots \\ 0 & & 0 & & -1 \end{bmatrix} \begin{bmatrix} v_n \\ \vdots \\ v_1 \\ u_m \\ \vdots \\ . \\ u_0 \end{bmatrix} = - \begin{bmatrix} a_{m+n} \\ \vdots \\ a_{m+1} \\ a_m \\ \vdots \\ . \\ a_0 \end{bmatrix} v_0.$$

By Lemma 6.9, there always exist solutions to this linear system, which is Block Toeplitz.

Hence our parallel algorithms for extended GCD of Corollary 6.6 provides an efficient parallel algorithm for finding Padé approximants:

Corollary 6.7. *The $(m, n)$ Padé approximant $P_{mn}(x) = U(x)/V(x)$ to a power series can be computed in randomized parallel time $O(\log^2 n)$ using $P(n)$ processors, with success probability $\geq 1 - 1/n^{\Omega(1)}$.*

**Applications to Sturm sequences and Finding Real Roots.** A *Sturm sequence* (See [20]) of polynomials $f_0(x), f_1(x)$ is the sequence of polynomials $f_0(x), f_1(x), \ldots, f_k(x)$ where for $i = 1, 2, \ldots k - 1$, $f_{i+1}(x) = q_i(x)f_i(x) - f_{i-1}(x)$, the $q_i(x)$ are linear, and $f_k(x)$ is constant. The Sturm sequence of $f_0(x), f_1(x)$ is similar to the remainder sequence generated by the Euclidean algorithm for the $GCD(f_0(x), f_1(x))$ except that $f_{i+1}(x)$ is the negative of the remainder of the division of $f_{i-1}(x)$ by $f_i(x)$. Therefore, the Sturm sequence can be computed in time $O(n \log^2 n)$ by simple modification (see Ben-Or and Tiwari [7]) of the usual HGCD algorithms (see [1]). Note: The precision required can be reduced by use of additional indeterminants as linear factors; see Collins [24] and [6, 7] for details. Pan [65] and others have observed that the linear system defining the Sturm sequence is block Toeplitz, with bounded displacement rank. Applying our results for bounded displacement rank matrices, by Lemma 4.4, and Corollaries 6.3 and 6.4 we have:

COROLLARY 6.8. *The Sturm sequence can be computed in parallel time $O(\log^2 n)$ using $P(n)$ processors, with success probability $\geq 1 - 1/n^{\Omega(1)}$.*

### 6.7. Real Root Isolation

The (standard) *Sturm sequence* of a single polynomial $f(x)$ of degree $n$ with derivative $f'(x)$ is defined to be the length $k \leq n$ Sturm sequence of $f_0(x) = f(x)$, $f_1(x) = f'(x)$. In the following, we assume that $f(x)$ has real coefficients and all roots are real. The applications of Sturm sequences use the following lemma, attributed to Rolle; see Marden, 1966 [56],

LEMMA 6.10. *If $f(x)$ has real coefficients and all roots are real, then the roots of $f'(x)$ are all real and they strictly interleave the roots of $f(x)$.*

For a real $a$, let $V_a$ be the number of sign variations of the Sturm sequence $f_0(a), f_1(a), \ldots, f_k(a)$, that is, the number of times $f_i(a) \cdot f_{i+1}(a) < 0$. It follows (Jacobson, 1974) from the result of Rolle that

LEMMA 6.11. *(Jacobson, 1974) For any interval $[a, b]$ of the real line, the number of real roots in this interval is $V_a - V_b$.*

Order the zeros of the linear terms $q_i(x)$ of the Sturm sequence $y_{i_1} \leq y_{i_2} \leq \ldots \leq y_{i_k}$ and order the roots of $f(x)$ $r_1 < r_2 < \ldots < r_n$. For any fixed $\epsilon, 0 < \epsilon < 1/2$, a point $s$ on the real line is an *$\epsilon$-splitting point* for the roots of $f(x)$ if $r_i < s < r_{i+1}$ for some $\epsilon n \leq i \leq (1 - \epsilon)n$. Ben-Or, Feig, Kozen and Tiwari [6] prove the remarkable result that:

LEMMA 6.12. *There is a $j$ such that $y_{i_j}$ is a 1/4-splitting point for the roots of $f(x)$*

Now we give parallel algorithm for finding a 1/4-splitting point for the roots of $f(x)$. The sign sequence of the Sturm sequence is computable by multipoint evaluation, which costs parallel time $O(\log^2 n)$ using $P(n)$ processors (see JáJá [42]). Using a binary search of $O(\log n)$ stages on the sequence $y_{i_1} \leq y_{i_2} \leq \ldots \leq y_{i_k}$ and applying Lemma 6.11 to count the number of roots of $f(x)$ in the appropriate interval considered at each stage of this binary search, we get

COROLLARY 6.9. *A 1/4-splitting point for the roots of a polynomial $f(x)$ with all real roots can be computed in parallel time $O(\log^2 n)$ using $P(n)$ processors, with success probability $\geq 1 - 1/n^{\Omega(1)}$.*

**Parallel Solution of the Real Roots Problem.** Given a univariate complex polynomial $f(x)$ of degree $n$ with rational coefficients expressed as a ratio of two integers $< 2^m$, the *root problem* is to find all the roots of $f(x)$ up to specified precision $2^{-\mu}$. The *real root problem* is the root problem where all the roots of the polynomial are real. Reif [76] gave a real root algorithm which has sequential arithmetic time cost of $O(P(n)\log^2 n(\log n + \log b))$, where $b = m + \mu$. In the following, we assume for simplicity the case of high precision $b = m + \mu \leq n^c$, for a constant $c > 1$. In that case, the sequential algorithm of Reif has time bound $O(P(n)\log^3 n)$. The real root algorithm of Reif [76] proceeds by splitting with very high accuracy, the degree $n$ polynomial into a product of two polynomials each with degree at least $n/12$. This splitting algorithm can be viewed as a reduction to the problems (for polynomials of degree $n$ or inputs of size $n$): (i) polynomial convolution and multiplication, (ii) polynomial division, (iii) $n$ point polynomial evaluation and interpolation, and (iv) Sturm Computation. By Corollaries 6.6, 6.8, 6.9 all these computations can be done in parallel time $O(\log^2 n)$ using $P(n)$ processors. Thus $O(\log n)$ stages of root splitting allow us to extract all the root with high accuracy $2^b = 2^{n^{-\Omega(1)}}$, implying:

Corollary 6.10. *Given a polynomial of degree $n$ with only real roots and coefficient bit-precision $b = n^{O(1)}$, we can approximate all the real roots within bit-precision $b = n^{O(1)}$, in parallel time $O(\log^3 n)$ using $P(n)$ processors, with success probability $\geq 1 - 1/n^{\Omega(1)}$.*

**The Symmetric Tridiagonal Eigenvalue Problem.** The *symmetric tridiagonal matrix eigenvalue problem* is the problem of finding all the eigenvalues of an $n \times n$ symmetric tridiagonal matrix

$$A = \begin{bmatrix} b_1 & a_2 & 0 & 0 & \dots & 0 & 0 & 0 \\ a_2 & b_2 & a_3 & 0 & \dots & 0 & 0 & 0 \\ 0 & a_3 & b_3 & a_4 & \dots & 0 & 0 & 0 \\ \vdots & & \ddots & \ddots & & & & \\ & & & \ddots & \ddots & & & \\ & & & & & & & \\ 0 & & & & & a_{n-1} & b_{n-1} & a_n \\ 0 & \dots & & & & 0 & a_n & b_n \end{bmatrix}.$$

The real roots problem has an efficient reduction to and from the symmetric tridiagonal matrix eigenvalue problem, which has been attributed to Hald [38] and described in [52, 11, 12, 10] and also by JáJá [42], p428, homework 8.37 (this relationship is well-known and is also occurs in many other computational problems as inverse eigenvalue problems, orthogonal polynomials, Sturm sequences, three-term recurrences, Euclidean scheme, and Lanczos algorithm). This reduction from the symmetric tridiagonal matrix eigenvalue problem for the above matrix $A$ to the real roots problem requires us to compute the characteristic polynomial det $(\lambda I - A)$. We sketch here this efficient reduction, with arithmetic cost $O(n\log^2 n)$. For each $i = 1, \dots, n$ let $p_i(\lambda) = \det(\lambda I - A^{(i)})$, where $A^{(i)}$ is the $i \times i$ submatrix consisting of the first $i$ rows and the first $i$ columns. Note that $p_0(\lambda) = 1$, $p_1(\lambda) = \lambda - b_1$, and $p_i(\lambda) = (\lambda - b_i)p_{i-1}(\lambda) - a_i^2 p_{i-2}(\lambda)$. This recurrence equation (see JáJá [42]) can be solved for $p_n(\lambda) = \det(\lambda I - A)$ within arithmetic work $O(P(n)\log^2 n)$, or in parallel time $O(\log^2 n)$ using $O(P(n)\log n)$ processors. The reverse reduction of the polynomial root-finding problem for a polynomial $f(x)$ to the symmetric tridiagonal matrix eigenvalue problem, by the Euclidean remainder scheme, is described in Hald [38]. This reverse reduction is used also in [11, 12, 10], and shown to have Boolean cost $O(M(n)M(nm)\log n)$. The arithmetic cost for this reduction is $O(n\log^2 n)$. The Euclidean scheme can be applied to $f(x)$ and $f'(x)$ or equivalently, to $f(x)$ and $g(x)$ where $f'(x_i)g(x_i) > 0$, $f(x_i) = 0$. The computation of this reduction can be done by the quotient-tree procedure of [7] (sect. 8.1). The quotient-tree procedure yields all the quotients and the leading coefficients of the remainders, which are the entries of the tridiagonal. Recovery of the coefficients of the polynomial from the entries of the matrix is described in [11, 12, 10] and can also done by the technique of [52]. Thus the arithmetic cost for these forward and reverse reductions is easily be seen to be $O(P(n)\log^2 n)$ sequential steps. Bini and Pan [9, 11, 12] have noted that this also gives a parallel reduction to and from the real root problem and the eigenvalue problem for symmetric tridiagonal matrices. These reductions require the computation of a Euclidean polynomial remainder sequence similar to polynomial GCD, which we can compute by Corollary 6.6 in parallel time $O(\log^2 n)$ using $P(n)$ processors. Bini and Pan [9, 11, 12] required $O(\log^2 n)$ time using $nP(n)$ processors for their parallel reductions. We can improve their reduction by applying our results for bounded displacement rank matrices. Using Lemma 4.4, and Corollaries 6.3 and 6.4, we can compute these reductions in parallel time $O(\log^2 n)$ using $P(n)$ processors. By Corollary 6.10,

Corollary 6.11. *Given a symmetric tridiagonal matrix of size $n \times n$ and coefficient bit-precision $b = n^{O(1)}$, we can approximate all the eigenvalues (which are all real in this case) within bit-precision $b = n^{O(1)}$, in parallel time $O(\log^3 n)$ using $P(n)$ processors, with success probability $\geq 1 - 1/n^{\Omega(1)}$.*

## 7. CONDITION BOUNDS FOR RANDOM MATRICES

Proposition 7.1. *(Demmel [26], Equation 4.20) Let $S$ be a real, homogeneous hypersurface over a given dimension $N$ defined by a single polynomial of degree $d$. Let $p$ be a random point chosen over a uniform distribution on the unit sphere of dimension $d$. Let $dist(p, S)$ the shortest Euclidean distance from $p$ to $S$. Then for any $\epsilon > 0$,*

$$Prob(dist(p, S) \leq \epsilon) \leq 2\sum_{j=1}^{N} \binom{N}{j} (2d\epsilon)^j.$$

Let $S_n$ be the set of real $n \times n$ matrices which are singular. Note that $S_n$ is a real, homogeneous hypersurface over dimension $N = n^2$ defined by a single polynomial of degree $d = n$ (that is, the determinant polynomial set to 0). Eckart and Young [28] (also see Demmel [26], Theorem 3.1) proved:

Proposition 7.2. *The distance $dist(A, S_n)$ from a matrix $A$ to the nearest singular matrix is $\|A^{-1}\|^{-1}$.*

Since by definition $cond_2(A) = \|A\|\ \|A^{-1}\|$, this distance is $dist(A, S_n) = \|A\|/cond_2(A)$. This implies that $cond_2(A)$ can be upper bounded by a lower bound on $dist(A, S_n)$ as given by Proposition 7.1.

The *Frobenius norm* of matrix $A$ is $\|A\|_F = (\sum_i |a_{ij}|)^{1/2}$. Let $A$ be a random $n \times n$ matrix with elements chosen so that $\|A\|/\|A\|_F$ uniformly distributed over the unit sphere. Propositions 7.2 and 7.1 can be immediately applied to bound the condition of $A$ by (see [26], Theorem 5.1):

$$Prob(cond_2(A) \geq x) \leq 2 \sum_{j=1}^{n^2} \left( \begin{array}{c} n^2 \\ j \end{array} \right) (\frac{2n}{x})^j.$$

Hence $Prob(cond_2(A) \geq x) \leq O(n^3/x)$ for $x > 4n^3$.

Similarly, let $LTS_n$ be the set of real, singular $n \times n$ lower triangular Toeplitz matrices with unit diagonal elements. Note that each such matrix is defined from only the $n-1$ strictly lower elements of the first column. Thus $LTS_n$ is a real, homogeneous hypersurface over dimension $N = n-1$ defined by a single polynomial of degree $d = n-1$ (again, this is the determinant polynomial set to 0). Let $L$ be a random $n \times n$ lower triangular Toeplitz matrix with unit diagonal elements and with strictly lower elements of the first column randomly chosen with a uniform distribution over the unit sphere. Let $L'$ be the lower triangular Toeplitz matrix derived from $L$, where the $n-1$ strictly lower elements of the first column are normalized to unit Frobenius norm. Propositions 7.2 and 7.1 immediately imply that

$$Prob(cond_2(L') \geq x) \leq 2 \sum_{j=1}^{n-1} \left( \begin{array}{c} n-1 \\ j \end{array} \right) (\frac{2(n-1)}{x})^j$$

and so $Prob(cond_2(L') \geq x) \leq O(n^2/x)$ for $x > 4n^2$. Since the renormalization to unit Frobenius norm can only decrease the condition by a factor of $\leq 1/n^2$, the bound

$$Prob(cond_2(L) \geq xn^2) \leq O(n^2/x)$$

holds for the (un-normalized) the matrix $L$. Since by Proposition 7.2, $dist(L, LTS_n) = \|L\|/cond_2(L)$, this is equivalent to the bound:

$$Prob(dist(L, LTS_n) \leq \|L\|/(xn^2)) \leq O(n^2/x).$$

This immediately also implies a similar bound on the condition for a lower triangular Toeplitz matrix $L$ whose elements are randomly chosen over a discrete uniform distribution, where the distance between discretization points is $1/n^c = o(\|L\|/(xn^2))$ for $c > 4$. Let $x = n^{c-2}$. In this case, with at most this same probability $O(n^2/x) = O(1/n^{c-4})$, we have $dist(L, LTS_n) \leq \|L\|/(xn^2) - 1/n^c \leq (\|L\|/n^c)(1 - o(1))$, and so with at most this same probability $O(1/n^{c-4})$, we have $cond_2(L) \geq xn^2(1+o(1)) = n^c(1+o(1))$. Absorbing the $o(1)$ additive factor into the $O(-)$ notation, with at most probability $O(1/n^{c-4})$, we have $cond_2(L) \geq n^c$.

LEMMA 7.1. *Let $L$ be a random $n \times n$ lower triangular Toeplitz matrix $L$ with unit diagonal elements and with the strictly lower elements of the first column randomly chosen uniformly over the values $\{i/n^c|$ where $i$ is an integer on the range from $-n^c$ to $n^c\}$, for a constant $c > 4$. Then with high likelihood, matrices $L, U$ are well conditioned, and in particular, $Prob(cond_2(L) \geq n^c) \leq O(1/n^{c-4})$.*

## 8. RANDOMIZED CONSTRUCTION OF DISPLACEMENT GENERATORS

Suppose $A$ is an $n \times n$ matrix with minimum displacement rank $\delta$, and let $S = \Delta_+(A) = A - ZAZ^T$. If $A$ is a generic matrix then the resulting $S$ has a nonsingular $\delta \times \delta$ leading principal submatrix $S([\delta],[\delta])$, so we can then apply Lemma 4.5 as given in Subsection 4.3 to construct $\delta$ displacement generators for $A$.

However, in general $S$ may have a singular $\delta \times \delta$ leading principal submatrix. Here we extend these techniques of Subsection 4.3 to specific rather than just generic matrices. To remedy this, we choose random $n \times n$ non-singular lower triangular Toeplitz matrices $L, U^T$ (see [47]) with unit diagonal elements and with the strictly lower elements of the first column randomly chosen uniformly over the values $\{i/n^c|$ where $i$ is an integer on the range from $-n^c$ to $n^c\}$ for a constant $c \geq 5$. Then by Lemma 7.1, the condition of matrices $L, U$ is $\leq n^c$ with likelihood $\geq 1 - \Omega(1/n^{c-4})$.

These preconditioning multipliers $U, L$, can be used to construct a matrix which is expected to act like a generic matrix; in particular, by Proposition 6.1 $S' = USL$ has a nonsingular $\delta \times \delta$ leading principal submatrix $S'([\delta],[\delta])$ with likelihood $\geq 1 - 1/n^{\Omega(1)}$. (In the case $S'([\delta],[\delta])$ or $U, L$ are singular, we repeat the random choice of $U, L$.) Now we consider the case where the leading principal submatrix $S'([\delta],[\delta])$ of $S'$ is nonsingular and also $U, L$ are nonsingular. Since $S$ has rank $\delta$, and $U, L$ are nonsingular, and $S' = USL$, it follows that $S'$ has rank $\delta$. By Proposition 4.1, $S' = S'(-,[\delta])S'([\delta],[\delta])^{-1}S'([\delta])$ so it follows that $S = U^{-1}S'L^{-1} = U^{-1}S'(-,[\delta])S'([\delta],[\delta])^{-1}S'([\delta])L^{-1}$. Therefore we redefine $g_1, \ldots, g_\delta$ to be the first $\delta$ columns of the product $U^{-1}S'(-,[\delta])$. We also redefine ${h_i}^T$ to be the $i$th row of the product $S'([\delta],[\delta])^{-1}S'([\delta])L^{-1}$ for $i = 1, \ldots, \delta$. (Note that this naturally generalizes the definition of the $h_i^T$ in Lemma 4.5, where previously these were defined to be $i$th row of the product $M^{-1}S([\delta])$.) For this redefinition of the $g_i$ and $h_i$ we have

$$\sum_{i=1}^{\delta} g_i {h_i}^T = (U^{-1}S'(-,[\delta]))(S'([\delta],[\delta])^{-1}S'([\delta])L^{-1}) = U^{-1}S'(-,[\delta])S'([\delta],[\delta])^{-1}S'([\delta])L^{-1} = S.$$

Thus we have the following extension of Lemma 4.5 to the general case.

LEMMA 8.1. *Let $A$ be an $n \times n$ matrix and let $GENERATOR_\delta(A)$ be the $\Delta_+$-displacement generator $n$-vectors $\{g_i\}, \{h_i\}$, $i = 1, \ldots, \delta$ where the $g_i$ and $h_i$ are as just redefined with a nonsingular $S'([\delta],[\delta])$. If $A$ has minimum displacement rank $\delta$ then $S = \Delta_+(A) = \sum_{i=1}^{\delta} g_i {h_i}^T$ and by Lemma 4.4, $A = \sum_{i=1}^{\delta} L(g_i)L^T(h_i)$.*

## 9. CONCLUSION

**Open Problems.** There are a number of further open problems remaining:

- Our RF algorithm includes the random selection of a prime whose binary representation's length is linear in the size of the input matrix. So there remains the open problem of finding similarly efficient parallel algorithms that do not use randomization.

• Also, our RF algorithm requires us to apply a homomorphism from $\mathbf{Q}$ to $\mathbf{Z}_p$. To apply this homomorphism, we have assumed a model of computation with unit cost division over a finite field (whereas prior works do not generally require this). Recall that NC is the class of Boolean circuits of polylog depth and polynomial size. Division over a finite field is NC reducible to extended integer GCD. However, no NC algorithm has yet been found for extended integer GCD, and we can not directly derive an efficient algorithm in the Boolean circuit model. We conclude that there remains the challenging open problem of finding similarly efficient parallel algorithms that use significantly smaller prime moduli or that work in the Boolean circuit model of parallel computation.

**Acknowledgments.** The author wishes to thank Shenfeng Chen, Chris Lambert, Arman Glodjo, K. Subramani, Aki Yoshida, Hongyan Wang, and Prokash Sinha for various edits and comments on the paper, Ken Robinson for excellent editorial assistance, and especially Deganit Armon for a very thorough technical reading of this paper and her insightful comments. We also gratefully acknowledge the referees as well as Wayne Eberly and Wei Zhang for suggesting substantial improvements to our presentation and proofs.

## REFERENCES

1. A.V.Aho, J.E.Hopcroft, and J.D.Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA (1974).
2. G.S.Ammar and W.G.Gragg, *Superfast solution of real positive definite Toeplitz systems*, SIAM J. Matrix Anal. Appl. **9** (1988), 61–76.
3. D.Armon and J.Reif, *Space and time efficient implementations of parallel nested dissection*, Proc. SPAA'92 (1992), 344–352. BS83
4. A.Ben-Israel, *A note on iterative method for generalized inversion of matrices*, Math. Comput. (1966), 439–440.
5. A.Ben-Israel and D.Cohen, *On iterative computation of generalized inverses and assorted projections*, SIAM J. Numer. Analysis (1966), 410–419.
6. M.Ben-Or, E.Feig, D.Kozen and P.Tiwari, *A Fast Parallel Algorithm for Determining All Roots of a Polynomial with Real Roots*, SIAM J. Comput. (1986).
7. M.Ben-Or and P.Tiwari, *Simple Algorithms for Approximating All Roots of a Polynomial with Real Roots*, Journal of Complexity **6** (1990), 417–442.
8. E.R.Berlekamp, *Algebraic Coding Theory* Ch.7, McGraw-Hill (1968).
9. D. Bini, *Divide and conquer techniques for the polynomial root-finding problem*, Proceedings of the International Congress of Nonlinear Analysts, Tampa, August (1992).
10. D. Bini, L. Gemignani, *On the complexity of polynomial zeros*, SIAM J. Comput., 21,4, 781-799, 1992.
11. Bini D., Pan V.,*Parallel Complexity of Tridiagonal Symmetric Eigenvalue Problem*, Proc. 2nd Annual ACM–SIAM Symp. on Discrete Algorithms, pp. 384–393, (1991).
12. Bini D., Pan V., *Practical Improvement of the Divide-and-Conquer Eigenvalue Algorithms*, Computing 48, pp. 109-123, (1992).
13. Bini D., Pan V., Polynomial and Matrix Computations, Volume 1, Birkhauser,Boston, MA, (1994).

14. R.R.Bitmead and D.O.Anderson, *Asymptotically fast solution of Toeplitz and related systems of linear equations*, Linear Algebra and its Applications **34**, (1980), 103–116.

15. R.R.Bitmead, S.-Y.Kung, D.O.Anderson, and T.Kailath, *Greatest common divisors via generalized Sylvester and Bezout matrices*, IEEE Trans. Automatic Control (1978), 1043–1047.

16. A.Borodin, J.von zur Gathen and J.Hopcroft, *Fast parallel matrix and GCD computations*, Information and Control **52** (1982), 241–256.

17. A.Borodin and I.Munro, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier (1975).

18. R.P.Brent, F.G.Gustavson, and D.Y.Y.Yun, *Fast solution of Toeplitz systems of equations and computation of Padé approximants*, J. Algorithms **1** (1980), 259–295.

19. W.S.Brown and J.F.Traub, *On Euclid's Algorithm and the Theory of Subresultants*, J. ACM **18** (1971), 505–514.

20. W.S.Burnside and A.W.Panton, *The Theory of Equations, Vols. 1 and 2*, Dover (1960).

21. D. G. Cantor and E. Kaltofen, *On fast multiplication of polynomials over arbitrary rings*, Acta Inf. **28** (1991), 697–701.

22. J.Cheriyan and J.H.Reif, *Parallel and Output Sensitive Algorithms for Combinatoral and Linear Algebra Problems*, Proc. of Symp. on Parallel Algorithms and Architectures '93 (1993).

23. J.Chun and T.Kailath, *Divide-and-conquer solution of least-squares problems for matrices with displacement structure*, SIAM J. Matrix Anal. Appl. (submitted).

24. G.E.Collins, *Polynomial Remainder Sequences and Determinants*, Amer. Math. Monthly **73** (1966), 708–712.

25. D. Coppersmith and S. Winograd. *Matrix Multiplication via Arithmetic Progressions*, J. Symbolic Comput. **9**:3 (1990), 251–280.

26. J. W. Demmel, *The probability a numerical analysis problem is difficult*, Mathematics of Computation **30**,182, (April 1988), 449–480.

27. W. Eberly, Personal communication, 1995.

28. C. Eckart and G. Young, *The approximation of one matrix by another of lower rank*, Psychometrika **1**, (1936), 211–218.

29. B.Friedlander, T.Kailath, M.Morf, and L.Ljung, *Extended Levinson and Chandrasekar equations for general discrete-time linear estimation problems*, IEEE Trans. Automatic Control **4** (1978), 653–659.

30. G.Frobenius, *Über Relationem zwischen den Naherungsbruchen von Potenzreihen*, J. fur Math. **90** (1881), 1–17.

31. H.Gazit and G.L.Miller, personal communication, (1990).

32. I.Gohberg, T.Kailath, and I.Koltracht, *Efficient solution of linear systems of equations with recursive structure*, Linear Algebra Appl. **80** (1986), 81–113.

33. I.Gohberg, T.Kailath, I.Koltracht, and P.Lancaster, *Linear complexity parallel algorithms for linear systems of equations with recursive structure*, Linear Algebra Appl. **88/89** (1987), 271–315.

34. I.C.Gohberg and A.A.Semencul, *On the inversion of finite Toeplitz matrices and their continuous analogs*, Mat. Issled. **2** (1972), 201–233 (in Russian).

35. G.H.Golub and C.F.van Loan, *Matrix Computations*, Johns Hopkins Univ. Press (1989).

36. W.B.Gragg, *The Padé table and its relation to certain algorithms of numerical analysis*, SIAM Rev. **14** (1972), 1–62.

37. F.G.Gustavson, *Analysis of the Berlekamp-Massey linear feedback shift-register synthesis algorithm*, IBM J. Res. Develop. **20** (1976), 204–212.

38. O.H. Hald, *Inverse Eigenvalue Problems for Jacobi Matrices, Linear Algebra Appl.*, 14 (1976) 63-85.

39. G.Heining and K.Rost, *Algebraic methods for Toeplitz-like matrices and operators*, Operator Theory **13** (1984).

40. H.Hotelling, *Some new methods in matrix calculation*, Ann. Math. Statist. (1943), 1–34.

41. H.Hotelling, *Further points on matrix calculations and simultaneous equations*, Ann. Math. Statist. (1943), 440–441.

42. J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley (1992).

43. T.Kailath, *A view of three decades of linear filtering theory*, IEEE Trans. Inf. Theory **20** (1974), 146–181.

44. T.Kailath and J.Chun, *Generalized Gohberg-Semencul formulas for matrix inversion*, Op. Theory: Advan. Appl. **40** (1989), 231–246.

45. T.Kailath, S.-Y.Kung, and M.Morf, *Displacement ranks of matrices and linear equations*, J. Math. Anal. Appl. (1979), 395–407.

46. T.Kailath, A.Viera, and M.Morf, *Inverses of Toeplitz operators, innovations, and orthogonal polynomials*, SIAM Rev. **20** (1978), 106–119.

47. E.Kaltofen and V.Pan, *Processor efficient parallel solution of linear systems over an abstract field*, Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, (1991), 180-191.

48. E.Kaltofen and V.Pan, *Processor-efficient parallel solution of linear systems II The positive characteristic and singular cases*, Proc. 33rd Annual IEEE Symposium on F.O.C.S. (1992), 714–723.

49. E.Kaltofen and B.D.Saunders, *On Wiedemann's method of solving sparse linear systems*, in Proc. AAECC-5, Lecture Notes in Computer Science **536**, Springer-Verlag, (1991), 216–226.

50. D.Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Addison-Wesley (1981).

51. D.Kozen, *Fast parallel orthogonalization*, SIGACT News (1987), 47.

52. A.S. Krishnakumar, M. Morf,*Eigenvalues of a symmetric tridiagonal matrix: a divide-and-conquer approach*, Numer. Math, 48 (1986), 349-368.

53. R.J.Lipton, D.J.Rose, and R.E.Tarjan, *Generalized Nested Dissection*, SIAM Jour. Num. Anal., **16** (1979) 346–358.

54. R.J.Lipton and R.E.Tarjan, *A Separator Theorem for Planar Graphs*, SIAM J. Applied Math. (1979), 177–189.

55. J.Makhoul, *Linear prediction: A tutorial review*, Proc. IEEE (1975), 561–580.

56. M.Marden, *Geometry of Polynomials*, American Mathematical Society

57. R.T.Moenck and J.H.Carter, *Approximate algorithms to derive exact solutions to systems of linear equations*, Proc. Eurosam, Lecture Notes in Computer Science (1979), 63–73.

58. M. Morf, *Doubling Algorithms for Toeplitz and Related Equations*, Proc.International Conference on ASSP, IEEE Computer Society Press, 954–959, (1980).

59. B.R.Musicus, *Levinson and fast Choleski algorithms for Toeplitz and almost Toeplitz matrices*, Internal Report, Lab of Elec., MIT (1981).

60. S.T.O'Donnell and V.Rokhlin, *A fast algorithm for numerical evaluation of conformal mappings*, Research Report RR-554, Yale Univ. Comp. Sci. Dept. (1987).

61. H.Padé, *Sur la representation approchee d'une fonction par des fractions rationelles*, Thesis, Ann. Ecole Nor. **9** (1892), 2–93.

62. V.Pan, *Fast and efficient parallel algorithms for the exact inversion of integer matrices*, Proc. 5th Conf. FST and TCS Lecture Notes in Computer Science (1985), 504–521.

63. V.Pan, *Complexity of Parallel Matrix Computations*, Theoretical Computer Science **54** (1987), 65–85.

64. V.Pan, *New effective methods for computations with structured matrices*, Tech. Report 88-28, Computer Science Dept., State Univ. of New York at Albany, Albany, NY.

65. V.Pan, *Fast and Efficient Parallel Evaluation of the Zeros of a Polynomial Having Only Real Zeros*, Computers Math. Applic. **17** (1989), 1475-1480.

66. V.Pan, *On computations with dense structured matrices*, *Mathematics of Computation*, Vol. 55, No. 191, (1990), 179–190.

67. V.Pan, *Parameterization of Newton's Iteration for Computations with Structured Matrices and Applications*, Tech. Report CUCS-032-90, Computer Science Dept., State Univ. of New York at Albany, Albany, NY.(1990).

68. V.Pan, *Parallel solution of Toeplitzlike linear systems*, Journal of Complexity, 8(1), (1992), 1-21.

69. V.Pan and J.Reif, *Efficient Parallel Solution of Linear Systems*, Proc. 17th ACM Symp. on Theory of Computing(STOC) (1985), 143–152.

70. V.Pan and J.H.Reif, *Some polynomial and Toeplitz matrix computations*, Proc. 28th Annual IEEE Symposium on Foundations of Computer Science(FOCS), (1987), 173–184.

71. V.Pan and J.H.Reif, *Fast and Efficient Parallel Solution of Dense Linear Systems*, Comp. Math. Applic. (1989), 1481–1491.

72. V. Pan and J.H.Reif, *The Parallel Computation of Minimum Cost Paths in Graphs by Stream Contraction*, Information Proc. Lett., **40**, (1991), pp. 79-83.

73. V.Pan and J.H.Reif, *Fast and Efficient Parallel Solution of Sparse Linear Systems*, SIAM Journal on Computing (1992).

74. V.Pan. I. Sibe, and A. Atinkpahoun, *On Parallel Computations with Band Matrices*, Information and Computation, 120, 2, 237–250, (1995).

75. J.H.Reif, ed., *Synthesis of Parallel Algorithms*, Morgan Kaufmann (1993).

76. J.H.Reif, *An* $O(n \log^3 n)$ *Algorithm for the Real Root Problem*, IEEE Symposium of Foundations of Computer Science (FOCS93), Palo Alto, CA (1993).

77. J.H.Reif, *Work Efficient Parallel Solution of Toeplitz Systems and Polynomial GCD*, Duke CS Technical Report Nov (1993), 27th Annual ACM Symposium of Theory of Computing (STOC95), Las Vegas, Nevada, May,1995.

78. J.H.Reif, $O(\log^2 n)$ *Time Efficient Parallel Factorization of Dense, Separable, and Banded Matrices*, 6th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'94), NJ, July 1994. Full paper: Duke Computer Science Dept. Technical Report, November, 1994.

79. J.H.Reif, *Parallel Output Sensitive Algorithms for Combinatorial and Linear Algebra Problems,* to appear in Journal of Computer and System Sciences, 2000.

80. J.T.Schwartz, *Fast probabilistic algorithms for verification of polynomial identities*, J. ACM **27** (1980), 701–717.

81. V.Strassen, *The Asmptotic Spectrum of Tensors and the Exponent of Matrix Multiplication*, Proc. 27th Annual IEEE Symposium on Foundations of Computer Science(FOCS), (1986), 49–54.

82. M.Tismenetsky, *Bésoutians, Toeplitz and Hankel matrices in the spectral theory of matrix polynomials*, Ph.D. Thesis, Technion, Hafia (1981).

83. W.F.Trench, *An algorithm for the inversion of finite Toeplitz matrices*, J.SIAM (1964), 515–522.

84. R. E. Zippel, *Probabilistic algorithms for sparse polynomials*, in "Proc. EUROSAM '79", Lecture Notes in Computer Science **72** (1991), 216–226.