Parallel Complexity of Numerically Accurate Linear System Solvers.*

Mauro Leoncini† Giovanni Manzini‡ Luciano Margara§

August 8, 1997

Abstract

We prove a number of negative results about practical (i.e., work efficient and numerically accurate) algorithms for computing the main matrix factorizations. In particular, we prove that the popular Householder and Givens' methods for computing the QR decomposition are P-complete, and hence presumably inherently sequential, under both real and floating point number models. We also prove that Gaussian Elimination (GE) with a weak form of pivoting, which only aims at making the resulting algorithm nondegenerate (but possibly unstable), is likely to be inherently sequential as well. Finally, we prove that GE with partial pivoting is P-complete when restricted to Symmetric Positive Definite matrices, for which it is known that even plain GE does not fail. Altogether, the results of this paper give further formal support to the widespread belief that there is a tradeoff between parallelism and accuracy in numerical algorithms.

1 Introduction

Matrix factorization algorithms form the backbone of state-of-the-art numerical libraries and packages, such as LAPACK and MATLAB [2, 14]. Indeed, factoring a matrix is almost always the first step of many scientific computations, and usually the one which places the heaviest demand in terms of computing resources. In view of their importance, some authors have investigated the parallel complexity of the most popular matrix factorizations, namely the $(P)LU$ and $QR(\Pi)$ decompositions (see Appendix A for definitions and simple properties). A list of positive known results follows.

- LU decomposition is in arithmetic NC, whenever it exists, i.e., provided that the leading principal minors of the input matrix are nonsingular (in this case we will say that the matrix is *strongly nonsingular*) [16, 18].

*This work merges preliminary results presented at ESA '96 and SPAA '97.

†Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy, and IMC-CNR. via S. Maria 46, 56126 Pisa, Italy. Email: leoncini@di.unipi.it. Supported by Murst 40% funds.

‡Dipartimento di Scienze e Tecnologie Avanzate, Università di Torino, Via Cavour 84, 15100 Alessandria, Italy. Email: manzini@unial.it.

§Dipartimento Scienze dell'Informazione, Università di Bologna, Piazza Porta S. Donato 5, 40127 Bologna, Italy. Email: margara@cs.unibo.it.

- QR decomposition is in arithmetic NC for matrices with full column rank, since it easily reduces to LU decomposition of strongly nonsingular matrices [16].

- PLU decomposition is in arithmetic NC for nonsingular matrices [7]. The algorithm for finding a permutation matrix P such that P^TA is strongly nonsingular builds upon the computation of the Lexicographically First Maximal Independent Subset of the rows of a matrix, which is in NC^2 [3][1].

- $QR\Pi$ factorization of an arbitrary matrix A is in arithmetic NC [7]. A permutation Π such that the leftmost $n \times r$ submatrix of A has full column rank, for $r = \mathrm{rank}(A)$, can be found by computing LFMIS of sets of (column) vectors.

Unfortunately, none of the above algorithms has proved to be numerically accurate with respect to a realistic model of arithmetic, such as double precision floating point. Actually, finding a numerically stable NC algorithm to compute the LU (or QR) decomposition of a matrix can be regarded as one of the major open problems in parallel computation theory [10].

That a positive solution to the above problem is not just around the corner is confirmed by the negative results that can be proved of the *stable* algorithms in practical use for computing the LU and QR decompositions. Already in 1989 Vavasis proved that Gaussian Elimination with Partial pivoting (GEP), which is the standard method for computing the PLU decomposition, is P-complete over the real or rational numbers [20]. Note that, strictly speaking, membership in P could not be defined for the real number model. When dealing with real matrices and real number computations we then assume implicitly that the class P be defined to include the problems solvable in polynomial time on such models as the real RAM (see [17]). The result in [20] was proved by showing that a decision problem defined in terms of GEP's behavior was P-complete. For parallel complexity theory the P-completeness result implies that GEP is likely to be inherently sequential, i.e., admitting no NC implementations unless $P = NC$. One of the authors proved then that GEP is probably even harder to parallelize, in the sense that no $O(n^{\frac{1}{2}-\epsilon})$ time PRAM implementation can exist unless all the problems in P admit polynomial speedup [12].

In this paper we prove new negative results about the classical factorization algorithms. We consider the methods of Householder's reflections and of Givens' rotations to compute the QR decomposition of a matrix. The main use of the QR decomposition is within iterative methods for the computation of the eigenvalues of a matrix and to compute least squares solutions to overdetermined linear systems [9]. Moreover, both Householder's and Given's methods (hereafter denoted HQR and GQR, respectively) are potential competitors of Gaussian Elimination to solve systems of linear equations stably in parallel. To date, the fastest stable parallel solver is based on GQR and is characterized by $O(n)$ parallel time on an $O(n^2)$ processor PRAM [19], while both GEP and HQR run in $O(n \log n)$ time with $O(\frac{n^2}{\log n})$ processors. Also, GQR is especially suitable for solving large sparse systems, given its ability to annihilate selected entries of the input matrix at very low cost.

We also consider the application of Gaussian Elimination with Partial pivoting to special classes of matrices and a weaker form of pivoting which we will call *Minimal*

[1]Not to be confused with the analogous LFMIS problem of graph theory, which is known to be P-complete [10].

pivoting (GEM). Under minimal pivoting, the pivot chosen to annihilate a given column is the first nonzero on or below the main diagonal. Minimal pivoting is especially suitable for systolic-like implementations of linear system solvers (see, e.g., [11], although it is not called this way). Minimal pivoting can be regarded as the minimum modification required for Gaussian Elimination to be nondegenerate on arbitrary input matrices.

We prove the following results.

1. HQR and GQR are P-complete over the real or floating point numbers. We exhibit reductions from the NAND Circuit Value Problem (NANDCVP) with fanout ≤ 2. In particular, what we prove to be P-complete is to decide the sign of a given diagonal element of the upper triangular matrices R computed by either HQR or GQR. Our reductions seem to be more intricate than the simple one in [20]. This is probably a consequence of the apparently more complex effect of reflections and rotations with respect to the linear combinations of Gaussian Elimination. We would like to stress that the P-completeness proofs for the case of floating point arithmetic apply directly, and have been checked with, the algorithms available in the state-of-the-art package Matlab using the IEEE 754 standard for floating point arithmetic. In other words, the negative results apply to widely "in-use" algorithms.

2. We extend Vavasis' result proving that GEP is P-complete on input strongly nonsingular matrices. This class includes matrices which are important in practical applications, namely the diagonally dominant and symmetric positive definite ones. Note that plain Gaussian Elimination (no pivoting) is guaranteed not to fail on input a strongly nonsingular matrix. However, since it is usually unstable, one still uses GEP.

3. We prove that GEM is P-complete on general matrices.

4. We show that the known NC algorithm for computing a PLU decomposition of a nonsingular matrix corresponds to GE with a nonstandard pivoting strategy which only slightly differs from Minimal Pivoting. Also, we prove that GE with such a nonstandard strategy is P-complete on input arbitrary matrices, which somehow accounts for the difficulties of finding an NC algorithm to compute the PLU decomposition of possibly singular matrices.

The results of this paper give further evidence of the pervasiveness of a phenomenon that has been observed also by numerical analysts (from a more practical perspective). Namely that there is a "tradeoff" between the degree of parallelism, on the one hand, and nondegeneracy and accuracy properties of numerical algorithms, on the other [5].

The rest of this paper is organized as follows. In Section 2 we introduce a little notation and give some preliminary definitions. In Section 3 we describe the key ideas that are common to the P-completeness proofs for all the factorization methods considered in the paper. In Sections 4 and 5 we address QR decomposition via Householder's reflections and Givens' rotations, respectively. In Section 6 we prove our negative results about Gaussian Elimination with Partial and Minimal pivoting. In section 7 we show a correspondence between a known NC PLU decompostion algorithm and Gaussian Elimination. We conclude with some further considerations and open problems. In Appendix A we discuss

the algorithms considered in this paper. In Appendix B we give some basic definitions about the floating point number representation. Clearly, more material about these well known algorithms and the computer arithmetic can be found in many excellent textbooks (in particular, see [9]). Finally, we include one technical proof in Appendix C.

2 Preliminaries

The notations adopted here for matrices and matrix-related concepts are the standard ones (see [9]). Matrices are denoted by capital letters. The (i, j) entry of a matrix A is referred to by either a_{ij} or $[A]_{ij}$. Vectors are designated by lower case letters, usually taken from the end of the alphabet, e.g., x, y, etc. Note that the notation x refers to a *column* vector, i.e., an $n \times 1$ matrix, for some $n \geq 1$.

The i-th row (resp., column) of a matrix A is denoted by a_{i*} (resp., a_{*i}). A *minor* of a matrix A is any submatrix of A. A *principal* minor is any square submatrix of A formed by the same set of row and column indices.

The symbols I and O denote the identity matrix (with $[I]_{ij} = 1$ if $i = j$ and 0 otherwise) and the zero matrix (such that $[O]_{ij} = 0$), respectively. The zero vector is denoted using the symbol 0. The transpose of A is the matrix B such that $b_{ij} = a_{ji}$. B is usually denoted by A^T. A matrix Q is orthogonal when $Q^T Q = I$. A permutation matrix P is a matrix which is zero everywhere except for just one 1 in each row and column. Any permutation matrix is orthogonal. The transpose of a (column) vector x is the *row* vector x^T, i.e., a matrix of size $1 \times n$, for some n.

Let A be a square matrix of order n.

- The LU decomposition of A is a pair of matrices $\langle L, U \rangle$, such that L is lower triangular (i.e., $l_{ij} = 0$ for $j > i$) with unit diagonal elements, U is upper triangular, and $A = LU$. For an arbitrary (even nonsingular) matrix A the LU decomposition might not be defined. A sufficient condition for its existence (and unicity) is that A be strongly nonsingular.

- The PLU decomposition of A is a triple of matrices $\langle P, L, U \rangle$ such that L and U are as above, P is a permutation matrix, and $P^T A = LU$ (or, equivalently, $A = PLU$). The PLU decomposition is always defined but not unique.

- The QR decomposition of A is a pair of matrices $\langle Q, R \rangle$, such that Q is orthogonal, R is upper triangular, and $A = QR$. The QR decomposition always exists.

In all the above cases, if A is $m \times n$, with $m < n$, and when the factorization exists, we get a matrix that is properly said to be in *row echelon* form (rather than upper triangular). Its leftmost $m \times m$ minor is upper triangular while its rightmost $m \times (n - m)$ minor is in general a dense submatrix. However, when no confusion is possible, we will always speak of the triangular factor of a given factorization.

A detailed description of the algorithms considered in this paper can be found in Appendix A. The details, however, are not necessary to understand the common structure of the reductions. Some details will be required in the proofs of Theorems 4.3 and 5.3, which deal with the floating point version of the QR algorithms. Except for these, the following general description is sufficient. It defines a class $\mathcal{F}$ of matrix factorization

algorithms that includes, among the others, the classical QR algorithms and Gaussian Elimination.

Let A be the input matrix. The algorithms in $\mathcal{F}$ bring A to upper triangular form by applying a series of transformations that introduce zeros in the strictly lower triangular portion of A, from left to right. The notation $A^{(k)}$ is usually adopted to indicate the matrix obtained after $k-1$ transformations and its elements are referred to by $a_{ij}^{(k)}$. $a_{ij}^{(k)}$ is zero for $j < \min\{i,k\}$. A transformation is applied during one *stage* of the algorithm.

Every algorithm $\mathcal{A} \in \mathcal{F}$ satisfies the following properties.

p1 If $a_{ij} = 0$, for $j = 1, \ldots, s$, then $a_{i*}^{(k)} = a_{i*}$, $k = 1, \ldots, s$. In other words, if the first s entries in row i are zero, then the first s stages of $\mathcal{A}$ do not modify row i.

p2 If column k has complementary nonzero structure with respect to columns $1, \ldots, k-1$ (by which we mean $a_{ij} \neq 0 \Rightarrow a_{ik} = 0$ for any i), then $a_{*k}^{(j)} = a_{*k}$, i.e., column k is not affected by the first $k-1$ transformations.

p3 This is a property that we will call of *proper embedding* of a matrix A into a larger matrix A'. Let $A = (\bar{A}\ \hat{A})$ be a $k \times n$ matrix, with $\bar{A}$ of dimensions $k \times (k-1)$, and let R be the triangular factor computed by $\mathcal{A}$ on input A. Let E be a matrix having A as a minor and suppose that, as a consequence of the repeated applicability of **p1** and **p2**, the first $k-1$ stages of algorithm $\mathcal{A}$ on input E only affects the rows that identifies A. Then $E^{(k)}$ contains R as a minor. In other words, the factorization of A, viewed as a part of E, is the same as the factorization of A alone. Perhaps the simplest example of proper embedding is $E = \begin{pmatrix} \bar{A} & \hat{A} \\ O & B \end{pmatrix}$, with $E^{(k)} = \begin{pmatrix} \bar{A}' & \hat{A}' \\ O & B \end{pmatrix}$ and $R = (\bar{A}'\ \hat{A}')$.

p4 Stage k modifies the entry (i,j) only if $i, j \geq k$. In particular, it introduces zeros in the k-th column of $A^{(i-1)}$ without destroying the previously introduced zeros. In view of this, and to avoid some redundant descriptions, in the rest of this paper we will use the notation $A^{(k)}$ to indicate the submatrix of $A^{(k)}$ with elements from $a_{kk}^{(k)}$ rightward and downward.

3 A framework for reductions to $\mathcal{F}$

Our P-completeness results are all based on reductions from the NANDCVP, a restricted version of CVP (Circuit Value Problem) which we now briefly recall:

> **Input**: the description of a k-input boolean circuit C composed entirely of fanin 2 nand gates, and boolean values $x_1, \ldots, x_k$.
>
> **Output**: the value $C(x_1, \ldots, x_k)$ computed by C on input $x_1, \ldots, x_k$.

NANDCVP is P-complete, as reported in [10]. In order to simplify the proofs we will further assume, without loss of generality, that each gate of C has fanout at most two. What we shall prove in this section is the following general result, which applies to any factorization algorithm $\mathcal{A} \in \mathcal{F}$.

There is an encoding scheme of logical values and a log-space bounded transducer M with the following properties: given the description of a fanout 2 nand circuit C and boolean inputs $x_1, \ldots, x_k$ for C, M builds a matrix A_C of order N_C such that, if $A_C = XR$ is the factorization computed by algorithm $\mathcal{A}$, with R upper triangular, then $[R]_{N_C,N_C}$ is the encoding of $C(x_1, \ldots, x_k)$.

We shall prove (Theorem 3.1) that the transducer exists provided that there are certain elementary matrices with well defined properties. We will later show that such matrices do exist for the algorithms considered in this paper.

Unfortunately, a formal description and the proof of correctness of the transducer will require quite a large amount of details. In spite of this, the idea behind the construction is easy, namely to repeatedly apply the proper embedding property. Hence we first describe the reduction in an informal way and only afterward proceed to a formal derivation. Moreover, we have actually implemented the transducer as a collection of Matlab m-files. These and the elementary matrices for the floating point versions of Householder's and Givens' QR algorithms (the most interesting and technical ones) are electronically available through the authors.

3.1 Informal description

Let $\mathcal{A} \in \mathcal{F}$ and let $\mathbf{a}$ and $\mathbf{b}$ denote appropriate numerical encodings of the truth values $a, b \in \{\texttt{True}, \texttt{False}\}$. We need three kinds of (square) elementary matrices for $\mathcal{A}$.

The first such matrix is the *nand* N. It has $[N]_{11} = \mathbf{a}$ and $[N]_{22} = \mathbf{b}$ and, if we apply $\mathcal{A}$ to compute the factorization of N, we get the encoding of $\texttt{NAND}(a, b)$ in the right bottom entry of the upper triangular factor.

The second elementary matrix is the *duplicator* D. It has $[D]_{11} = \mathbf{a}$. If we compute an incomplete factorization of D, i.e., apply all but the last transformation of $\mathcal{A}$ to D, we get $\begin{pmatrix} \mathbf{a} & 0 \\ 0 & \mathbf{a} \end{pmatrix}$ in the right bottom corner of the incomplete triangular factor.

The third elementary matrix is the *copier* or *wire* W. It has $[W]_{11} = \mathbf{a}$, and if we compute the factorization of W we get $\mathbf{a}$ in the right bottom entry of the triangular factor.

Using these matrices as the building blocks we can construct a matrix A_C that simulates the circuit C. The structure of A_C is close to block diagonal, with one N block for each nand gate in the circuit C. Duplicator blocks are used to simulate fanout 2 nand gates, and wire blocks to route the computed values according to the circuit's structure. As the factorization of a block diagonal matrix could be performed by independently factoring the single blocks, a certain degree of overlapping between the blocks is necessary to pass the computed values around.

To illustrate how the preceding scheme can work in practice, and to see where the difficulties may appear, consider first the construction of a submatrix which simulates a fanout 2 nand gate. The basic idea is to append a duplicator to a nand block as pictorially shown in Figure 1 (left). The N block is the dark gray area, the D block is light gray, and the white zones contain zeros. The right bottom entry of N coincides with the top left entry of D. This is an example of proper embedding. Suppose N has order k. Then after $k-1$ stages of $\mathcal{A}$ the encoding of $\texttt{NAND}(a, b)$ is exactly where required, i.e., in the top left entry of D from where it can be duplicated. The light gray area in Figure 1 (right)

has not been modified by the transformations. The only changes in the submatrix still to be triangularized occurred in the first row of D. We already know that entry (k, k) has been modified properly, but this needs not be the case for the other entries, i.e., the black colored ones in Figure 1 (right). For the simulation to proceed correctly, it is required that the black entries store the rest of the first row of D *by the time the algorithm starts working on column* k. We cannot rely on their initial contents.

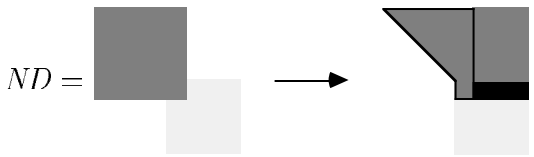

Figure 1: N-D matrix composition: effect of the first $k-1$ stages

As a second example, Figure 2 pictorially describes how a W block can be used to pass a value to a possibly far away place. The W block (the dark gray area in Figure 2) is split across non consecutive rows and columns. More precisely, if W is of order k, the top left dark gray area is intended to represent the principal minor of order $k-1$ of W. As before, the white zones contain zeros, while the light gray area is of arbitrary size and stores arbitrary values. This situation again represents a proper embedding, so that the first $k-1$ stages of $\mathcal{A}$ on input Ws (with w the order of W) will result in the factorization of the W block. This implies that the (encoding of) the logical value initially in the top left entry has been copied to the far away right bottom entry.

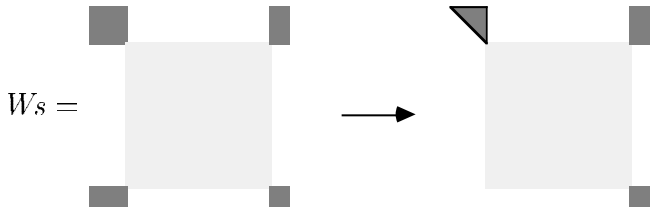

Figure 2: Splitting of a W block

To get an idea of what a complete matrix might looks like, see Figure 3, where the circuit C computing the exclusive or of two bits is considered. The corresponding matrix A_C has four N blocks, one D block and four W blocks denoted by different gray levels. Note, however, that Figure 3 does not incorporate yet the solution to the "black entries" problem mentioned above.

3.2 Elementary matrices

To prove the correctness of the transducer (in Theorem 3.1 below) it is convenient to introduce a block partitioning of the elementary matrices defined in the previous section.

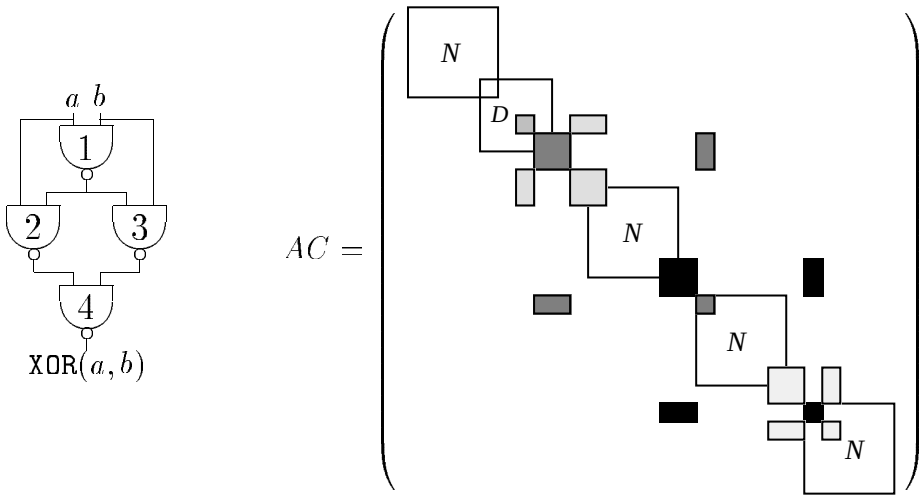

Figure 3: Circuit C computing $\texttt{XOR}(a,b)$ (left) and the structure of the corresponding matrix AC (right).

Let $E \in \{N, D, W\}$ denote one such matrix. We partition E as follows

$$E = \begin{pmatrix} E_I & \bar{E}_I & \hat{E}_I \\ \bar{E}_M & E_M & \hat{E}_M \\ \bar{E}_O & \hat{E}_O & E_O \end{pmatrix}, \tag{1}$$

where the diagonal blocks E_I, E_M, and E_O are square matrices and E_I and E_O also diagonal (i.e., with only zeros outside the main diagonal). We will refer to E_I and E_O as to the *input* and *output* submatrices and let i and o denote their order, respectively. Note that an elementary matrix actually defines a set of matrices. In fact, we regard the i diagonal entries of E_I as the *input places*. A particular elementary matrix is obtained by filling the input place(s) with the encoding of some logical value(s).

We can now formally define the "behavior" of the elementary matrices with respect to the factorization algorithm $\mathcal{A}$.

Nand matrix (N) We let ν denote the order of N, which has $i = 2$ and $o = 1$ in the block partitioning (1), and set $[N]_{11} = \mathbf{a}$, $[N]_{22} = \mathbf{b}$. If $XN = R$ is the factorization computed by $\mathcal{A}$, we have

$$R = \begin{pmatrix} * & \cdots & \cdots & * \\ & \ddots & & \vdots \\ & & * & * \\ & & & \mathbf{c} \end{pmatrix}$$

where $\mathbf{c}$ is the encoding of $\texttt{NAND}(a,b)$. If, as often required in practice, R overwrites the input matrix, the value $\mathbf{c}$ will replace N_O, and this is the reason why we defined E_O in (1) as the output submatrix. The same remark applies to the other elementary matrices. We also require that, for any real value x, an *auxiliary vector* $a_N = a_N(x) = (0, 0, a_3, \ldots, a_\nu)^T$ can be defined such that $Xa_N = (*, \ldots, *, x)^T$. Using the auxiliary

vectors we can solve the "black entries" problem outlined in Section 3.1. Intuitively, by appending auxiliary vectors to the right of N in A_C (instead of simply zeros, as in Figure 1 (left)), we can obtain the desired values in the black colored entries of Figure 1 (right). As we will see in the proof of Theorem 3.1, the initial zeros in the auxiliary vector prevents the problem from pumping up in the construction of A_C.

Duplicator matrix (D) We let d denote the order of D, which has $i = 1$ and $o = 2$ in the block partitioning (1), and set $[D]_{11} = \mathbf{a}$. If $XD = R$ is the incomplete factorization computed by $\mathcal{A}$, i.e., X represents the first $d-2$ transformations applied to D by $\mathcal{A}$, we obtain

$$R = \begin{pmatrix} * & \dots & \dots & \dots & * \\ & \ddots & & & \vdots \\ & & * & \dots & * \\ & & & \mathbf{a} & 0 \\ & & & 0 & \mathbf{a} \end{pmatrix}.$$

We also require that, for any pair of real numbers x and z, an auxiliary vector $a_D = a_D(x,z) = (0, a_2, \dots, a_d)^T$ can be defined such that $Xa_D = (*, \dots, *, x, z)^T$.

Wire matrix (W) We let w denote the order of W, which has $i = o = 1$ in the block partitioning (1), and set $[W]_{11} = \mathbf{a}$. If $XW = R$ is the factorization computed by $\mathcal{A}$, we get

$$R = \begin{pmatrix} * & \dots & \dots & * \\ & \ddots & & \vdots \\ & & * & * \\ & & & \mathbf{a} \end{pmatrix}.$$

We require again that, for any real number x, an auxiliary vector $a_W = a_W(x) = (0, a_2, \dots, a_w)^T$ can be defined such that $Xa_W = (*, \dots, *, x)^T$.

Elementary matrices (including auxiliary vectors) exist for both Householder's and Givens' methods and for Gaussian Elimination as well.

3.3 Construction and Correctness

In this section we present our main result (Theorem 3.1) on the existence of a single reduction scheme that works for any algorithm in $\mathcal{F}$. We still require a couple of definitions.

- Suppose that C has k input variables and let $k' \geq k$ be the number of places (inputs to the gates) where the variables are used. k' is the number of inputs counting multiplicities. Let the gates of C be sorted in topological order. Given a specific assignment of logical values to the input variables we may then refer to the i-th *actual input* as to the i-th value from the input set that is required at some gate, $1 \leq i \leq k'$.

- When we say that A_C *has an input row at position* i, we intend that initially row i is either

$$\left(\overbrace{0,\ldots,0}^{i-1},\mathbf{a},0,[\bar{N}_I]_{1*},[\hat{N}_I]_{1*},0^T\right), \tag{2}$$

or

$$\left(\overbrace{0,\ldots,0}^{i-2},0,\mathbf{a},[\bar{N}_I]_{2*},[\hat{N}_I]_{2*},0^T\right), \tag{3}$$

where $\mathbf{a}$ is the encoding of one of the actual inputs to C.

Theorem 3.1 *Let elementary matrices N, D, and W be given for $\mathcal{A} \in \mathcal{F}$. For any fanout 2 nand circuit C with input variables $x_1,\ldots,x_k$ and any truth assignment to $x_1,\ldots,x_k$, we can build a square matrix A_C such that the following holds*

(a) A_C has order $\bar{n} = O(n)$, where n is the number of gates in C.

(b) If $A_C = XR$ is the factorization computed by $\mathcal{A}$, then $[R]_{\bar{n},\bar{n}}$ is the encoding of $C(x_1,\ldots,x_k)$.

(c) A_C has a number of input rows which equals the number $k' \geq k$ of actual inputs to C; the i-th such row has either the structure (2) or (3) depending on whether the i-th actual input to C enters the first or the second input of some gate.

(d) Any actual input affects A_C only through one input place.

The construction can be done by using $O(\log n)$ work space.

Proof. We prove the result by induction on n. Let $z_1,\ldots,z_{k'}$, for $k' \geq k$, be the actual inputs to C and let $\mathbf{a}$ and $\mathbf{b}$ be the encodings of z_1 and z_2, respectively. The case $n = 1$ is easy. We only have to set $A_C = N$, with $[A_C]_{11} = \mathbf{a}$ and $[A_C]_{22} = \mathbf{b}$. Clearly $\bar{n} = O(1)$. Property (b) follows from the definition of N. Properties (c) and (d) are easily verified as well. In particular, A_C has exactly 2 input rows at positions 1 and 2, with structure (2) and (3), respectively, and this clearly matches the number of actual inputs to C. Finally, the actual inputs affect A_C only through the input places $[A_C]_{11}$ and $[A_C]_{22}$.

Now suppose that the number of gates in C is $n > 1$, and let $g_1,\ldots,g_n$ be a topological ordering of the DAG representing C. Clearly, all the inputs to g_1 are actual inputs to C. Let C' be the circuit with g_1 removed and any of its outputs replaced with x_1, the first input variable. Since C' has $n-1$ gates, we may assume that $A_{C'}$ can be constructed which satisfies the induction hypothesis. To build A_C we simply extend $A_{C'}$ to take g_1 into account. There are two cases, depending on the fanout of g_1. We fully work out only the case of fanout 1, the other being similar but just more tedious to develop (and for the reader to follow) in details.

1. Let g_h be the gate connected to the output of g_1. Suppose, w.l.o.g., that g_1 provides the first input to g_h. By the induction hypothesis (in particular, by (c)) $A_{C'}$ has the following structure.

$$A_{C'} = \begin{array}{c} \\ i_h \\ \downarrow \\ \end{array} \begin{pmatrix} X_1 & x_1 & x_2 & X_2 & X_3 & X_4 \\ 0^T & \mathbf{a} & 0 & [\bar{N}_I]_{1*} & [\hat{N}_I]_{1*} & 0^T \\ y_1^T & 0 & \gamma & y_2^T & y_3^T & y_4^T \\ Z_1 & z_1 & z_2 & Z_2 & Z_3 & Z_4 \end{pmatrix} \leftarrow i_h$$

$A_{C'}$ has an input row at some position i_h corresponding to the first input to gate g_h and $\mathbf{a}$ is the encoding of x_1. Note that, by property (d), the actual logical value encoded by $\mathbf{a}$ only affects the definition of $A_{C'}$ through the entry (i_h, i_h). Using $A_{C'}$ and N and W elementary matrices we define A_C as follows

$$A_C = \begin{pmatrix} N_I & \bar{N}_I & \hat{N}_I & \boxed{O} & O & \boxed{0} & 0 & O & O & O \\ \bar{N}_M & N_M & \hat{N}_I & \boxed{A_1} & O & \boxed{a_1'} & 0 & O & O & O \\ \bar{N}_O & \hat{N}_O & N_O & \boxed{a_1^T} & 0^T & \boxed{\alpha} & 0 & 0^T & 0^T & 0^T \\ O & O & \bar{W}_M & W_M & O & \hat{W}_M & 0 & A_2 & A_3 & O \\ O & O & 0 & O & X_1 & x_1 & x_2 & X_2 & X_3 & X_4 \\ 0^T & 0^T & \bar{W}_O & \hat{W}_O & 0^T & W_O & 0 & a_2^T & a_3^T & 0^T \\ 0^T & 0^T & 0^T & 0^T & y_1^T & 0 & \gamma & y_2^T & y_3^T & y_4^T \\ O & O & 0 & O & Z_1 & z_1 & z_2 & Z_2 & Z_3 & Z_4 \end{pmatrix}.$$

We set $[A_C]_{11} = \mathbf{a}$ and $[A_C]_{22} = \mathbf{b}$. The minor enclosed in boxes is a set of $w-1$ (w is the order of W) auxiliary column vectors for N that we choose such that

$$X \begin{pmatrix} O & 0 \\ A_1 & a_1' \\ a_1^T & \alpha \end{pmatrix} = \begin{pmatrix} * & * \\ * & * \\ \bar{W}_I & \hat{W}_I \end{pmatrix},$$

where X is the matrix that factorizes N. Observe that only the i_h-th row of $A_{C'}$ has been modified by replacing $\mathbf{a}$, $[\bar{N}_I]_{1*}$, and $[\hat{N}_I]_{1*}$ with W_O, a_2^T, and a_3^T, respectively.

In what follow we regard A_C as a block 8×10 matrix, and when we refer to the i-th row (or column) we really intend the i-th block of rows (columns). Nonetheless, A_C is square, if $A_{C'}$ is, with order $\nu + w - 2$ plus the size of $A_{C'}$. Using part (a) of the induction hypothesis we then see that $\bar{n} = O(n)$. It is easy to prove that A_C enjoys properties (b) through (d) as well. Assume C has k' actual inputs. Since g_1 has fanout 1 C' has $k'-1$ actual inputs and, by induction, $A_{C'}$ has $k'-1$ input rows. Now, by the above construction A_C has exactly $(k'-1)-1+2 = k'$ input rows, which proves (c). Property (d) also easily holds. To prove (b) we use the properties of $\mathcal{F}$. By **p1**, the application of $\nu - 1$ stages of $\mathcal{A}$ to A_C only affects the first 3 (block of) rows. Thus N (including its auxiliary vectors) is properly embedded in A_C and hence after the first $\nu - 1$ stages of $\mathcal{A}$ we get

$$A_C^{(\nu-1)} = \begin{pmatrix} W_I & \bar{W}_I & 0^T & \hat{W}_I & 0 & \boxed{0^T} & \boxed{0^T} & 0^T \\ \bar{W}_M & W_M & O & \hat{W}_M & 0 & \boxed{A_2} & \boxed{A_3} & O \\ 0 & O & X_1 & x_1 & x_2 & X_2 & X_3 & X_4 \\ \bar{W}_O & \hat{W}_O & 0^T & W_O & 0 & \boxed{a_2^T} & \boxed{a_3^T} & 0^T \\ 0^T & 0^T & y_1^T & 0 & \gamma & y_2^T & y_3^T & y_4^T \\ 0 & O & Z_1 & z_1 & z_2 & Z_2 & Z_3 & Z_4 \end{pmatrix},$$

where $W_I = \mathbf{c}$ is the encoding of $\mathtt{NAND}(z_1, z_2)$. The submatrix enclosed in boxes is a set of $\nu - 2$ auxiliary vectors for W that we choose such that

$$X \begin{pmatrix} 0^T & 0 \\ A_2 & A_3 \\ a_2^T & a_3^T \end{pmatrix} = \begin{pmatrix} * & * \\ * & * \\ [\bar{N}_I]_{1*} & [\hat{N}_I]_{1*} \end{pmatrix},$$

where X is the transformation matrix that triangularizes W. Note that the entries corresponding to the first elements of auxiliary vectors contain zero, as required. If the first element (in the definition) of auxiliary vectors were not zero, we would be faced with the additional problem of guaranteeing that the first $\nu - 1$ stages would set these entries to the required values.

It is again easy to see that W (including its auxiliary vectors) is properly embedded in $A_C^{(\nu-1)}$ so that additional $w - 1$ stages of $\mathcal{A}$ leads to

$$A_C^{(n+w-2)} = \begin{pmatrix} X_1 & x_1 & x_2 & X_2 & X_3 & X_4 \\ 0^T & N_I & 0 & [\bar{N}_I]_{1*} & [\hat{N}_I]_{1*} & 0^T \\ y_1^T & 0 & \gamma & y_2^T & y_3^T & y_4^T \\ Z_1 & z_1 & z_2 & Z_2 & Z_3 & Z_4 \end{pmatrix},$$

with $N_I = W_I = \mathbf{c}$. The correctness now follows from the induction hypothesis and property **p4**.

2. The full description of the fanout 2 case is definitely more tedious but introduces no new difficulties. A_C extends $A_{C'}$ by means of an initial N block, followed by a D block, followed by two W blocks. Taking the partial overlappings into account, it immediately follows that the order of A_C is $\nu + d - 1 + 2(w - 2)$ plus the order of $A_{C'}$.

The construction of matrix A_C can be done in space proportional to $\log n$ by simply reversing the steps of the above inductive process. That is, instead of constructing $A_{C'}$ and using it to build A_C, which would require more than logarithmic work space, we compute and immediately output the first $\nu + w - 2$ (or $\nu + d - 1 + 2(w - 2)$ in case of a first gate with fanout 2) rows and columns. We also compute and output row i_h (or the two rows where the output of a fanout 2 gates has to be sent). All of this can be done in space $O(\log n)$ essentially by copying the elementary matrices N, D, and W to the output medium. The only possible problem might be the computation of i_h, but this is not the case. In fact, for any $1 \leq i \leq n$, let $f_2(i)$ be the number of fanout 2 nand gates preceding gate i in the linear ordering of C. This information can be obtained, when required, by repeatedly reading the input and only using $O(\log n)$ work space for counting. It easily follows from the above results that the index of the top left entry $\pi(h)$ of the h-th N block is

$$\begin{aligned} \pi(h) &= f_2(h)(2(w-2) + \nu + d - 1) + (h - 1 - f_2(h))(\nu + w - 2) + 1 \\ &= (h-1)(\nu + w - 2) + f_2(h)(d + w - 3) + 1. \end{aligned}$$

Hence i_h will be either $\pi(h)$ or $\pi(h) + 1$, depending on whether the value under consideration is the first or second input to g_h.

The Matlab program that implements the transducer is indeed log-space bounded. It only uses the definition of the blocks and simple variables (whose contents never exceed the size of A_C) in magnitude. No data structure depending on n is required. Clearly, as it is implemented using double precision IEEE 754 arithmetic, it can properly handle only the circuits with up to approximately 2^{53} gates.

□

4 Householder's QR decomposition algorithm

In this section we prove that HQR is presumably inherently sequential under both exact and floating point arithmetic. This is done by proving that a certain set H, defined in terms of HQR's behavior is logspace complete for P.

$$H = \{A : \ A \text{ is } n \times n,\ A = QR \text{ is the factorization computed by HQR, and } [R]_{nn} > 0\}.$$

Note that by HQR we intend the classical Householder's algorithm presented in many numerical analysis textbooks. In particular we refer to the one in [9]. This is also the algorithm available as a primitive routine in scientific libraries (such as LINPACK's `ZQRDC` [6]) and environments (like Matlab's `qr` [14])

We begin by the ready to hand result about the membership in P.

Theorem 4.1 *H is in P under both exact and floating point arithmetic.*

Proof. Follows from standard implementations, which perform $O(n^3)$ arithmetic operations and $O(n)$ square root computations (see, e.g., [9]). □

According to the result of Section 3, to prove that H is also logspace hard for P it is sufficient to exhibit an encoding scheme and the elementary matrices required in the proof of Theorem 3.1. As we will see, however, the floating point case asks for additional care to rule out the possibility of fatal roundoff error propagations.

Theorem 4.2 *H is logspace hard for P under the real number model of arithmetic.*

Proof. We simply list the three elementary matrices required by Theorem 3.1. For each elementary matrix E, the corresponding auxiliary vector a_E is shown as an additional column of E. That the matrices enjoy the properties defined in Section 3.2 can be automatically checked using any symbolic package, such as Mathematica©.

N It is the 9×10 matrix of Figure 4, where $\mathbf{a}, \mathbf{b} \in \{-1, 1\}$ are the encoding of logical values (1 for `True` and -1 for `False`) and x is an arbitrary real number. Performing 8 steps of HQR on input N gives $N = QR$ with $[R]_{9,9} = \mathbf{c}$ and $[R]_{9,10} = x$, where $\mathbf{c} = \frac{\mathbf{1-a-b-ab}}{2}$ is the arithmetization of $\mathtt{NAND}(a, b)$ under the selected encoding.

D It is the 6×7 matrix shown in Figure 5 (left). Performing 4 steps of HQR on input D gives $D = QR$ with $[R]_{5,5} = [R]_{6,6} = \mathbf{a}$, $[R]_{5,6} = [R]_{6,5} = 0$, $[R]_{5,7} = z$, and $R_{6,7} = x$, where z and x are arbitrary real numbers.

$$N = \begin{pmatrix} \mathbf{a} & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{5}{4} & 0 & 0 \\ 0 & \mathbf{b} & -1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & -\frac{579}{145} & -\frac{211}{145} & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & -\frac{23}{116} & -\frac{70}{29} & 0 & 0 & 0 \\ 0 & 0 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{4}{3} & 1 & 0 & 0 & 0 & 0 & \frac{2575}{3552} & 0 \\ 0 & 0 & \frac{4}{3} & \frac{33}{8} & 0 & 0 & 0 & 0 & -\frac{38525}{10656} & 0 \\ \frac{4}{3} & 0 & 0 & 0 & 0 & 0 & \frac{5}{4} & 0 & \frac{125}{72} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{4}{3} & 0 & 1 & \frac{25}{24} & \frac{25}{12}x \end{pmatrix}.$$

Figure 4: The N block for HQR.

$$D = \begin{pmatrix} \mathbf{a} & -1 & -2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & -\frac{579}{145} & -\frac{211}{145} & 0 \\ 2 & 0 & 0 & 0 & -\frac{23}{116} & -\frac{70}{29} & 0 \\ 0 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{4}{3} & 1 & 0 & 0 & 0 & \frac{27\,x+254\,z}{222} \\ 0 & \frac{4}{3} & \frac{33}{8} & 0 & 0 & 0 & \frac{4767\,x+256\,z}{1776} \end{pmatrix} \qquad W = \begin{pmatrix} s & -\frac{5}{4} & 0 \\ \frac{4}{3} & 0 & \frac{5\,x}{3} \end{pmatrix}$$

Figure 5: The D and W blocks for HQR.

W It is the 2×3 matrix of Figure Figure 5 (right). Performing 1 step of HQR on input W gives $W = QR$ with $[R]_{2,2} = \mathbf{a}$ and $[R]_{2,3} = x$, for x an arbitrary real number.

□

Applying a floating point implementation of HQR to any single block defined in Theorem 4.2[2] results in approximate results. For instance, we performed the QR decomposition of the four N matrices using the built-in function `qr` available in Matlab. We found that the relative error affecting the computed encoding of $\texttt{NAND}(a, b)$ ranged from a minimum of 0.5ϵ to a maximum of 3ϵ. Here ϵ is the roundoff unit and equals $2.2204 \cdot 10^{-16}$ under IEEE 754 standard arithmetic. These might appear insignificant errors. However, for a matrix containing an arbitrary number of blocks, the roundoff error may accumulate to a point where it is impossible to recover the exact (i.e., 1 or -1) result. Clearly, direct error analysis is not feasible here, since it should apply to an infinite number of reduction matrices. Our solution is to control the error growth by "correcting" the intermediate results as soon as they are "computed" by nand blocks. Note that, by referring to the values *computed* by a certain elementary matrix E, we properly intend the non zero values one finds in the last row of the triangular factor computed by HQR on input E (including the auxiliary vectors). Analogously, the input values to E are the ones computed by the elementary matrix preceding E in A_C.

Theorem 4.3 *H is logspace hard for P under finite precision floating point arithmetic.*

Proof. We take duplicator and wire blocks as in Theorem 4.2, and provide a new definition for nand blocks so that they always compute exact results. To do this, we have to consider again the structure of A_C, as resulting from Theorem 3.1.

[2]More precisely, to the best possible approximations of the blocks under the particular machine arithmetic.

Let g_i be the i-th gate in the topological ordering of C, and let g_j and g_k be the gates providing the inputs to g_i. Let $N^{(l)}$ denote the N block of A_C corresponding to g_l, according to the construction of Theorem 3.1. To prove the result we maintain the invariant that the values computed by $N^{(1)}, N^{(2)}, \ldots, N^{(i-1)}$ are exact. This is clearly true for $i = 1$. Using the invariant we first verify that the errors affecting the values computed by $N^{(i)}$ can be bounded by a small multiple of the roundoff unit. We then use the bound to show how to redefine N so that it computes exact results, thus extending the invariant to $N^{(i)}$.

From the proof of Theorem 3.1 we know that the output of $N^{(j)}$ (and similarly of $N^{(k)}$) is placed in one of the input rows of $N^{(i)}$ as a consequence of the factorization of possibly a D followed by a W block. It follows that the error affecting the output of $N^{(i)}$ is only due to the above factorizations *and* to the factorization of $N^{(i)}$ itself. Since there is a limited number of structural cases (depending on the fanout of gates j and k) and considering all the possible combinations of logical values involved, the largest error ever affecting the output of $N^{(i)}$ can be determined by direct (but tedious) error analysis or more simply by test runs. For the purpose of the following discussion we may safely assume that the relative errors affecting the computed quantities are bounded by $c\epsilon$, for some constant c of order unit (c is actually smaller than 10). In other words, we may assume that the actual outputs of $N^{(i)}$ are $\mathbf{a}(1+\delta)$ and $x(1+\eta)$, with $|\delta|, |\eta| \leq c\epsilon$. Recall that x is the last entry of the generic auxiliary vector $a_N(x)$ of N after the factorization (see the definition of N is Section 3.2). Here, however, we require that x be a machine number (i.e., a rational number representable without error under the arithmetic under considerations).

Having a bound on the error, we are now ready to show how to "correct" the erroneous outputs. The new nand block, denoted by N_{corr}, extends N with two additional rows and columns, as shown below

$$N_{\text{corr}} = \begin{pmatrix} N & a_N(-1) & 0 \\ b^T & 0 & -(2^m+1) \\ 0^T & 2^m & 0 \end{pmatrix},$$

where $b^T = (\overbrace{0, \ldots, 0}^{8}, 2^m)$ and m is some positive integer (to be specified below). Note that $a_N(-1)$ is precisely that auxiliary vector for the old N that produces -1 as output, i.e., $a_N(-1) = (\overbrace{0, \ldots, 0}^{8}, -\frac{25}{12})^T$. The auxiliary vector for N_{corr} is $(\overbrace{0, \ldots, 0}^{10}, (2^m+1)x)$. A first requirement on m is thus that the quantity $(2^m+1)x)$ be a computer number. As the length of the significand of 2^m+1 is $m+1$, we see that a sufficient condition is that the length of the significand of x does not exceed $t-m-1$. Now, let us apply HQR to N_{corr} extended by its auxiliary vector. As N is properly embedded in N_{corr}, after 8 stages of HQR we get (using the above result on the error)

$$N^{(9)}_{\text{corr}} = \begin{pmatrix} \mathbf{a}(1+\delta) & -1+\eta & 0 & 0 \\ 2^m & 0 & -(2^m+1) & 0 \\ 0 & 2^m & 0 & (2^m+1)x \end{pmatrix}.$$

A second condition on m is that we want that $2^m + \mathbf{a}(1+\delta) = 2^m + \mathbf{a}$ to get rid of the error δ. An easy argument shows that this implies $m > \lceil \log c \rceil$. Thus, recalling the bound

on $|\delta|$ and $|\eta|$, we see that $m \geq 5$ is sufficient. As a consequence, the length of x cannot exceed $t-6$. The actual reflection matrix applied to $N_{\mathrm{corr}}^{(9)}$ is then

$$I - \frac{1}{2^m(2^m+1)} \begin{pmatrix} \mathbf{a}(2^m+1) \\ 2^m \end{pmatrix} \begin{pmatrix} \mathbf{a}(2^m+1) & 2^m \end{pmatrix},$$

which, by easy floating point computation, gives

$$N_{\mathrm{corr}}^{(10)} = \begin{pmatrix} \mathbf{a}(1-\delta) & -1 & 0 \\ 2^m & 0 & (2^m+1)x \end{pmatrix}.$$

Applying one more stage now leads to the correct results $\mathbf{a}$ and x. The above requirement on x is by no means a problem. In fact, the auxiliary values ever required are the non zero elements in the input rows of the blocks that possibly follow nand elementary matrices, i.e., D and W blocks. These are simply -1, -2, and $-5/4$, all of which can be represented exactly with a 3 bit significand. □

The elementary matrices of Theorem 4.3 are available for the general transducer implemented in Matlab. In particular, N_{corr} is defined with $m = 30$.

5 QR decomposition through Givens' rotations

In this section we prove that the following set

$$G = \{A : \ A \text{ is } n \times n,\ A = QR \text{ is the factorization computed by GQR, and } [R]_{nn} > 0\},$$

is logspace complete for P. The way we present the results of this section closely follows the metodology of Section 4. Here, however, we have to spend some more words about the particular algorithm considered. In fact, the computation of the QR decomposition can be done in various ways using plane (or Givens') rotations. Differently from Householder's reflections, a single plane rotation annihilates only one element of the matrix to which it is applied, and different sequences of annihilations result in different algorithms. By the way, this degree of freedom has been exploited to obtain the currently faster (among the known accurate ones) parallel linear system solvers [19, 15]. We also outline that there is no QR algorithm available in Matlab (nor in libraries as LINPACK or LAPACK), but it just provides the primitive `planerot` that computes a plane rotations. The hardness results of this section apply to the particular algorithm that annihilates the subdiagonal elements of the input matrix by proceding downward and rightward. This choice places GQR in the class $\mathcal{F}$ defined in Section 2, with the position that one stage of the algorithm is the sequence of plane rotations that introduce zeros in one column.

Theorem 5.1 *G is in P under both exact and floating point arithmetic.*

Proof. See, e.g., [9]. We only point out that the membership in P holds independently of the annihilation order. □

Theorem 5.2 *G is logspace hard for P under the real number model of arithmetic.*

Proof. As in Theorem 4.2, we simply list the three elementary matrices extended with the generic auxiliary vector. The matrices are shown in Figures 6 through 8, where $\mathbf{a}, \mathbf{b} \in \{-1, 1\}$ are encodings of logical values (1 for `True` and -1 for `False`) and x and z are arbitrary real numbers. Again, that the matrices enjoy the properties defined in Section 3.2 can be verified with the help of a symbolic package. □

$$N = \begin{pmatrix}
\mathbf{a} & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & \mathbf{b} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 3 & 0 & 0 & 0 & \frac{3}{16} - \frac{3\sqrt{65}}{16} & \frac{-3+\sqrt{65}}{4} & -\frac{183}{64} - \frac{25\sqrt{65}}{64} & -\frac{241}{128} - \frac{125\sqrt{65}}{384} & 0 \\
0 & 1 & 0 & 2 & 0 & \frac{39\sqrt{\frac{13}{7}}}{2} + 4\sqrt{30} & \frac{-19\left(7\sqrt{10}+\sqrt{371}\right)}{14} & 0 & 0 & \alpha & 0 \\
0 & 1 & 0 & 3 & 0 & 6\sqrt{\frac{13}{7}} + \sqrt{30} & -4\sqrt{\frac{53}{7}} - 4\sqrt{10} & 0 & 0 & \beta & 0 \\
0 & 1 & 0 & 4 & 0 & \frac{29\sqrt{\frac{13}{7}}}{2} + 3\sqrt{30} & \frac{-17\left(7\sqrt{10}+\sqrt{371}\right)}{14} & 0 & 0 & \gamma & 0 \\
0 & 1 & 0 & 5 & 0 & \frac{7\sqrt{30}+5\sqrt{91}}{2} & -10\sqrt{10} - \frac{3\sqrt{371}}{2} & 0 & 0 & \delta & 0 \\
2 & 0 & 4 & 0 & 0 & 0 & \frac{5}{8} - \frac{3\sqrt{65}}{8} & \frac{-1+\sqrt{65}}{2} & -\frac{97}{32} - \frac{25\sqrt{65}}{32} & -\frac{119}{64} - \frac{125\sqrt{65}}{192} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \frac{-5x}{3} \\
0 & 0 & 0 & 0 & 0 & \frac{4}{3} & 0 & -1 & 0 & -\frac{125}{96} & 0
\end{pmatrix}.$$

$$\alpha = \left(\frac{-16625\sqrt{\frac{5}{2}}}{36} + \frac{875\sqrt{\frac{15}{2}}}{8} - \frac{4875\sqrt{91}}{128} + \frac{2375\sqrt{371}}{72}\right)/14, \quad \beta = \frac{-375\sqrt{\frac{13}{7}}}{64} - \frac{125\sqrt{\frac{5}{2}}}{9} + \frac{125\sqrt{\frac{15}{2}}}{64} + \frac{125\sqrt{\frac{53}{7}}}{18},$$

$$\gamma = \left(\frac{-14875\sqrt{\frac{5}{2}}}{36} + \frac{2625\sqrt{\frac{15}{2}}}{32} - \frac{3625\sqrt{91}}{128} + \frac{2125\sqrt{371}}{72}\right)/14, \quad \delta = \left(\frac{-625\sqrt{\frac{5}{2}}}{9} + \frac{875\sqrt{\frac{15}{2}}}{64} - \frac{625\sqrt{91}}{128} + \frac{125\sqrt{371}}{24}\right)/2$$

Figure 6: The N block for GQR.

$$D = \begin{pmatrix}
\mathbf{a} & 1 & 0 & 0 & 0 & 0 \\
1 & 2 & 0 & \frac{39\sqrt{\frac{13}{7}}}{2} + 4\sqrt{30} & \frac{-19\left(7\sqrt{10}+\sqrt{371}\right)}{14} & \left(\frac{-39\sqrt{\frac{13}{7}}}{2} + 4\sqrt{30}\right)x + \left(-19\sqrt{\frac{5}{2}} + \frac{19\sqrt{\frac{53}{7}}}{2}\right)z \\
1 & 3 & 0 & 6\sqrt{\frac{13}{7}} + \sqrt{30} & -4\sqrt{\frac{53}{7}} - 4\sqrt{10} & \left(-6\sqrt{\frac{13}{7}} + \sqrt{30}\right)x + \left(4\sqrt{\frac{53}{7}} - 4\sqrt{10}\right)z \\
1 & 4 & 0 & \frac{29\sqrt{\frac{13}{7}}}{2} + 3\sqrt{30} & \frac{-17\left(7\sqrt{10}+\sqrt{371}\right)}{14} & \left(\frac{-29\sqrt{\frac{13}{7}}}{2} + 3\sqrt{30}\right)x + \left(-17\sqrt{\frac{5}{2}} + \frac{17\sqrt{\frac{53}{7}}}{2}\right)z \\
1 & 5 & 0 & \frac{7\sqrt{30}+5\sqrt{91}}{2} & -10\sqrt{10} - \frac{3\sqrt{371}}{2} & \left(7\sqrt{\frac{15}{2}} - \frac{5\sqrt{91}}{2}\right)x + \left(-10\sqrt{10} + \frac{3\sqrt{371}}{2}\right)z
\end{pmatrix}$$

Figure 7: The D block for GQR.

$$W = \begin{pmatrix}
\mathbf{a} & 1 & 1 & 0 \\
2 & 3 & \frac{-3+\sqrt{65}}{4} & -\frac{15x}{4} - \frac{\sqrt{65}x}{4} \\
2 & 4 & \frac{-1+\sqrt{65}}{2} & -\frac{9x}{2} - \frac{\sqrt{65}x}{2}
\end{pmatrix}.$$

Figure 8: The W block for GQR.

We now switch to the more delicate case of finite precision arithmetic.

Theorem 5.3 *G is logspace hard for P under finite precision floating point arithmetic.*

Proof. We apply the same ideas of Theorem 4.3. That is, we extend the definition of N so that it always computes the exact results. Here, however, we cannot reuse the D block adopted for the exact arithmetic case. There is a subtle problem that urges for a different definition of D. Let us see in details. If we apply a floating point implementation of GQR to D we clearly get approximate results (note that the matrices for GQR contain irrational numbers). In particular, instead of $\begin{pmatrix} \mathbf{a} & 0 \\ 0 & \mathbf{a} \end{pmatrix}$, in the bottom right corner of R we get $\begin{pmatrix} \mathbf{a}(1+\delta') & \epsilon' \\ \epsilon'' & \mathbf{a}(1+\delta'') \end{pmatrix}$. Even if δ', δ'', ϵ', and ϵ'' are of the order of the roundoff unit ϵ, the fact that ϵ'' is not zero causes the whole construction to fail. Note that the same kind of approximate results are obtained under HQR, but with no damage there. To get to the point, suppose that ϵ'' is in column k of A_C and let us proceed by considering stage k of the algorithm. In HQR one single transformation annihilates the whole column so that the contribution of a tiny ϵ'' to the k-th transformation matrix is negligible. On the other hand, in GQR the elements are annihilated selectively and, since ϵ'' is not zero, one additional plane rotation is required between rows k and $k+1$ to place zero in the entry $(k+1,k)$. Unfortunately this has the effect of making the element in position (k,k) positive, which is a serious trouble since this entry contained the encoding, with a small perturbation, of a truth value. The result is that, when the subsequent plane rotations (the ones simulating the routing of the logical value) are applied, the value passed around is always the encoding of `True`, and the simulation fails in general.
We thus need to replace the duplicator with one that returns a true zero in the entry $(d,d-1)$ of the incomplete factor R, which will clearly exploit the properties of floating point arithmetic.

noindent Let m and M denote the length of the significand and the largest exponent e such that 2^e can be represented in the arithmetic under consideration, respectively. For the standard IEEE 754 $m = 53$ and $M = 1023$. The nonzero elements of the new duplicator are only powers of 2. In this way any operation is either exact or is simply a no operation.

$$D = \begin{pmatrix}
\mathbf{a} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2^{-m} & 2^{-M+2*m+1} & -2^{M-1} & 0 & 0 & 0 & 0 & 0 \\
2^{-m} & 2^{-3*m} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2^{-M+3*m+2} & 1 & 2^{M-1} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2^{-M+1+m} & 0 & 2^{-M+2*m+1} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 2^{-m} & 2^{M-1-m} & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 2^{-\lfloor \frac{m}{2} \rfloor} & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 2^{-\lfloor \frac{m}{2} \rfloor} & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 2^{-\lceil \frac{m}{2} \rceil} & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2^{-\lceil \frac{m}{2} \rceil} & 0 & 1
\end{pmatrix}.$$

As the new auxiliary vector we define

$$a_D(x,y) = (0, y2^{M-2}, 1, 1, x2^m, 2^{-M}, 2^m, 2^m, 2^{m-\lceil \frac{m}{2} \rceil}, 2^{m-\lceil \frac{m}{2} \rceil})^T.$$

Nothe that the only possible assignments to x and y are 0 and 1 or 1 and 0.
The rest of the proof is now similar to that of Theorem 4.3. We show how to correct the slightly erroneous values computed by an N block assuming that the previous N blocks

return exact results. Let N stand for the nand block adopted for the exact arithmetic version of GQR (Figure 6). The new nand block is then

$$N_{\mathrm{corr}} = \begin{pmatrix} N & 0 & 0 \\ c^T & 1 & 0 \\ 0^T & 2^{-\lceil \frac{m}{2} \rceil} & 1 \end{pmatrix},$$

where $c^T = (\overbrace{0, \ldots, 0}^{9}, 2^{-\lfloor \frac{m}{2} \rfloor})$. As the new auxiliary vector we take

$$a_{N_{\mathrm{corr}}}(x) = (\overbrace{0, \ldots, 0}^{8}, -\frac{5}{3}x2^m, 0, 2^m, 2^{m-\lceil \frac{m}{2} \rceil})^T,$$

i.e., the first 10 entries of $a_{N_{\mathrm{corr}}}(x)$ coincide with $a_N(x2^m)$. Now, let us apply GQR to N_{corr} extended by its auxiliary vector. As N is properly embedded in N_{corr}, after 9 stages of GQR we get

$$N_{\mathrm{corr}}^{(10)} = \begin{pmatrix} \mathbf{a}(1+\delta) & 0 & 0 & x2^m(1+\eta) \\ 2^{-\lfloor m/2 \rfloor} & 1 & 0 & 2^m \\ 0 & 2^{-\lceil m/2 \rceil} & 1 & 2^{m-\lceil m/2 \rceil} \end{pmatrix},$$

where $|\delta|, |\eta| \leq c\epsilon$, for some small constant c of order unit. The plane rotation to annihilate the entry $(2,1)$ of $N_{\mathrm{corr}}^{(10)}$ is represented by

$$G_1 = \frac{1}{\sqrt{\mathbf{a}^2(1+\delta)^2 \oplus 2^{-2\lfloor m/2 \rfloor}}} \odot \begin{pmatrix} \mathbf{a}(1+\delta) & 2^{-\lfloor m/2 \rfloor} \\ -2^{-\lfloor m/2 \rfloor} & \mathbf{a}(1+\delta) \end{pmatrix} = \begin{pmatrix} \mathbf{a} & \frac{2^{-\lfloor m/2 \rfloor}}{1+\delta} \\ -\frac{2^{-\lfloor m/2 \rfloor}}{1+\delta} & \mathbf{a} \end{pmatrix},$$

and its application in floating point gives

$$N_{\mathrm{corr}}^{(10)} = \begin{pmatrix} \mathbf{a} & 0 & \mathbf{a}2^m \ominus x2^{\lceil m/2 \rceil}(1+\zeta) \\ 2^{-\lceil m/2 \rceil} & 1 & 2^{m-\lceil m/2 \rceil} \end{pmatrix},$$

where $1+\zeta = \frac{1+\eta}{1+\delta}$ and $|\zeta| \leq 2\epsilon + O(\epsilon^2)$. The crucial point is that, if x can be represented with no more than $2^{\lfloor m/2 \rfloor}$ significant bits, the alignment of the fraction part performed during the execution of $\mathbf{a}2^m \ominus x2^{\lceil m/2 \rceil}(1+\zeta)$ will simply cause the contribute $x2^{\lceil m/2 \rceil}\zeta$ to be lost. Hence, the computed element in the entry $(1,3)$ will be $\mathbf{a}2^m - x2^{\lceil m/2 \rceil}$. But then one more rotation produces the exact values $\mathbf{a}$ and x in the last row. Note that the only value required in place of x is 1. □

6 Gaussian Elimination with Pivoting

In this section we consider the algorithm of Gaussian Elimination with partial pivoting, or simply *GEP*, a technique that avoids nondegeneracies and ensures (almost always) numerical accuracy. We also consider the less-known Minimal pivoting technique, *GEM*, one that only guarantees that a PLU factorization is found. Minimal pivoting has been adopted for systolic-like implementations of Gaussian Elimination [11]. A brief description of these algorithms is reported in Appendix A.

We prove that GEM is inherently sequential, unless applied to strongly nonsingular matrices while GEP is inherently sequential even when restricted to strongly nonsingular matrices.

6.1 Partial Pivoting

The proof we give here builds on the original proof in [20], and hence does not share the common structure of the other reductions in this paper. Essentially we show that, with little additional effort with respect to Vavasis' proof, we can exhibit a reduction in which the matrix obtained is strongly nonsingular. As already pointed out, strongly nonsingular matrices are of remarkable importance in practical applications. This class contains symmetric positive definite (SPD) and diagonally dominant matrices, which often arise from the discretization of differential problems. Observe that, on input such matrices, plain GE (no pivoting) is nondegenerate, but it is not stable in general and hence is not the algorithm of choice.

As in [20], that GEP is inherently sequential follws from the proof that the folllowing set is P-complete.

$$L = \{(i, j, A) : \ A \text{ is strongly nonsingular and, on input } A, \text{ GEP uses row } i \text{ to eliminate column } j \ \}.$$

Theorem 6.1 *The set L is log-space complete for P.*

We postpone the technical proof of Theorem 6.1 to the Appendix C, but give an example that shows the way the matrix given in [20] is modified. Figure 9 depicts the reduction matrix $\mathbf{M}_C$ which would be obtained according to the rules in [20] on input the description of the circuit of Figure 3. The matrix is nonsingular; however, it can be seen that the leading principal minor of order 2 is singular. The matrix we obtain, according to Theorem 6.1, is shown in Figure 10. It can be easily seen that our matrix is strongly diagonally dominant by rows, and hence strongly nonsingular.

$$\begin{pmatrix}
-3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\mathbf{a} & \cdot & 1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \mathbf{b} & 1 & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & 1 & 4.0 & 1 & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & 1 & 4.0 & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & 1 & \cdot & 4.0 & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 1 & 1 & 4.0 & \cdot & \cdot & \cdot & \cdot & \cdot & 1
\end{pmatrix}$$

Figure 9: Matrix $\mathbf{M}_C$ corresponding to the exclusive or circuit. The symbol $\cdot$ stands for a zero entry.

6.2 Minimal Pivoting

The technique of minimal pivoting, i.e., selecting as the pivot row at stage k the first one with a nonzero entry (below or on the main diagonal) in column k, is probably the simplest modification that allows GE to cope with degenerate cases. However, such

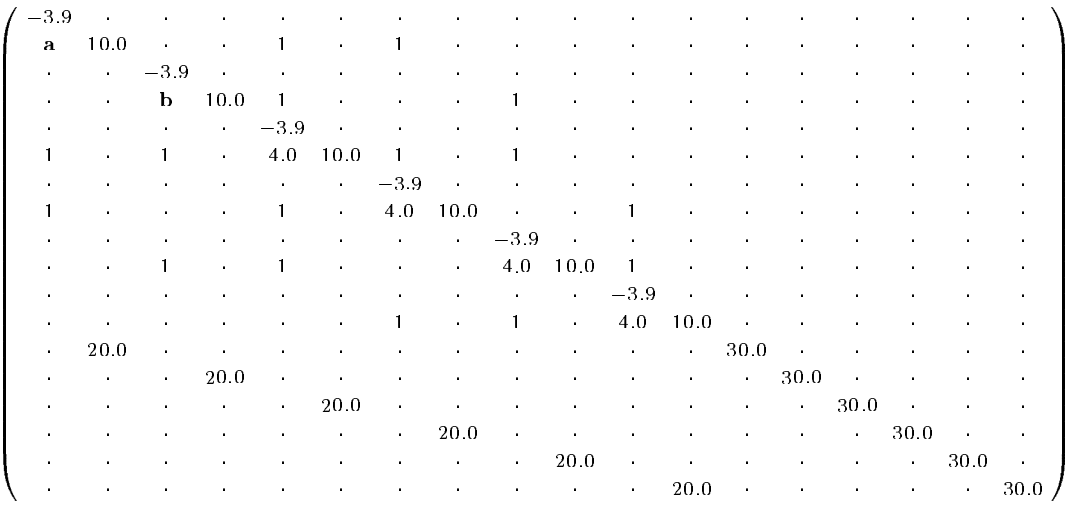

Figure 10: The matrix $\mathbf{A}_C$ for the computation of $\texttt{XOR}(a, b)$.

a simple technique is sufficient to make the Gaussian Elimination algorithm inherently sequential. Note that, even if no formal error analysis is available for GEM, it is not difficult to exhibit matrices (that can plausibly appear in real applications) such that the error incurred by GEM is very large. Actually, GEM is likely to be as unstable as plain GE.

Consider the following set.

$$L' = \{A : \ A \text{ is } n \times n,\ PA = LU \text{ is the factorization computed by GEM, and } [U]_{nn} > 0\}.$$

Theorem 6.2 *The set L' is logspace complete for P under both real and finite precision floating point arithmetic.*

Proof. The set is clearly in P, as GEM runs in time $O(n^3)$ under both models of arithmetic. We first show that L' is also logspace hard for P when the input matrices are singular, and then show how to restrict the input set. As GEM belongs to the class $\mathcal{F}$, to prove the hardness of L' we simply list the three elementary matrices required by Theorem 3.1. Note that the matrices are the same for both models of arithmetic, as the operations performed by GEM in floating point are exact. The encoding of logical values here is 0 for `False` and 1 for `True`. The matrices are depicted in Figures 11 and 12.

$$N = \begin{pmatrix} \mathbf{a} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{b} & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 & 0 & x \end{pmatrix} \qquad W = \begin{pmatrix} \mathbf{a} & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & x \end{pmatrix}$$

Figure 11: The nand (left) and wire (right) blocks for GEM.

$$D = \begin{pmatrix}
\mathbf{a} & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & z \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & z \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Figure 12: The D block for GEM.

A_C is clearly singular. Now consider the following matrix B_C of order $2n_C$, where N_C is the order of A_C.

$$B_C = \begin{pmatrix} A_C & E \\ E & O \end{pmatrix}, \tag{4}$$

where E is the matrix with 1 on the antidiagonal and 0 elsewhere. The determinant of B_C can be easily proved to be ± 1. Moreover, if $PB_C = LU$ is the factorization computed by GEM, then $\left[U\right]_{n_C,n_C}$. Note then that what we prove to be P-complete is not exactly L', but a set with a little more complicate definition (which is still in terms of GEM's behavior).

As usual, in the following the notation $A_C^{(k)}$ will be used to denote the matrix obtained after $k-1$ stages of GEM on input A_C, and considering only the entries (i,j) with $i,j \geq k$. However, by writing $B_C^{(k)}$ we intend the submatrix obtained after $k-1$ stages of GEM (on input B_C), and considering only the entries (i,j) such that $k \leq i,j \leq n_C$. With this position, to prove that the output of C can be read off entry (n_C, n_C) of the U factor of B_C, we show that the executions of GEM on input A_C and on input B_C result in identical submatrices $A_C^{(k)}$ and $B_C^{(k)}$, for $0 \leq k \leq n_C$. The proof is by induction. Initially, for $k = 1$, the equality follows from the definition of B. Consider stage $k \geq 1$. If column k of $A_C^{(k)}$ contains a nonzero element below or on the main diagonal, say at row index i, then the selected pivot row is the i-th under both executions. The results follows then from the induction hypothesis and the fact that exactly the same operations are performed on the elements of the submatrices. If no nonzero element is found in column k of $A_C^{(k)}$, then stage k of the first execution has no effect, and hence $A_C^{(k+1)} = A_C^{(k)}$. Under the second execution (the one on input B_C) by construction the pivot is taken from row $2n_C - k + 1$. However, the pivot is the only nonzero element in row $2n_C - k + 1$ and thus the effect of this step is simply the exchange of rows k and $2n_C - k + 1$. But then once more $B_C^{(k+1)} = B_C^{(k)}$. □

We conclude by observing that the set L' of Theorem 6.2 is clearly NC computable when the input set is the class of strongly nonsingular matrices. In fact, in this case, GEM and plain Gaussian Elimination behave exactly the same.

7 On NC algorithms for the PLU decomposition

In this section we show that a known NC algorithm for computing a PLU decomposition of a nonsingular matrix (see [7]) corresponds to GE with a nonstandard pivoting strategy which is only a minor variation of Minimal pivoting. This result seems to be just a "curiosity"; however, we can prove that the same strategy is inherently sequential on input arbitrary matrices, which can be seen as a further evidence of the difficulties of finding an NC algorithm to compute the PLU decomposition of possibly singular matrices.

The new strategy will be referred to as Minimal pivoting with circular Shift, and the corresponding elimination algorithm simply as GEMS. The reason for its name is that GEMS, like GEM, searches the current column (say, column k) for the first nonzero element. Once one is found, say in row i, a circular shift of row k through i is performed to bring row i in place of row k (and the latter in place of row $k+1$).

Theorem 7.1 *Computing the PLU factorization returned by GEMS on input a nonsingular matrix is in arithmetic NC^2.*

Proof. We consider the algorithm of Eberly [7]. Given A, nonsingular of order n, let A_i denote the $n \times i$ matrix formed from the first i columns of A, $i = 1, \ldots, n$. If S_i denotes the set of indices of the lexicographically first maximal independent subset of the rows of A_i, then $|S_i| = i$, since A_i has full column rank. Moreover, $S_i \subseteq S_{i+1}$, $i = 1, \ldots, n-1$. Note that the computation of all the S_i is in NC^2 (see [3]). Now, let $S_1 = \{j_1\}$, and, for $i = 2, \ldots, n$, $S_{i+1} - S_i = \{j_{i+1}\}$. Then a permutation P such that $P^T A$ has LU factorization is simply

$$P = (e_{j_1} | e_{j_2} | \ldots | e_{j_n}),$$

where e_i is the ith unit (column) vector. Clearly, once P has been determined, computing the LU factorization of $P^T A$ can be done in polylogarithmic parallel time using known algorithms. We now show by induction on the column index k that P is the same permutation determined by GEMS. The basis is trivial, since j_1 is the index of the first nonzero element in column 1 of A. Now, for $k > 1$, let

$$(e_{j_1} | \ldots | e_{j_k} | e_{l_{k+1}} | \ldots | e_{l_n})^T A = L_k U_k = L_k \begin{pmatrix} R_k & B_k \\ O & \mathcal{A}_k \end{pmatrix} \tag{5}$$

be the (partial) factorization computed by GEMS, where R_k is upper triangular with nonzero diagonal elements (since A is nonsingular) and the unit vectors $e_{l_{k+1}}, \ldots, e_{l_n}$ extend $e_{j_1}, \ldots, e_{j_k}$ to form a permutation matrix. Clearly, Minimal Pivoting ensures that $l_{k+1} < \ldots < l_n$. Now, the next pivot row selected by GEMS is the one corresponding to the first nonzero element in the first column of $\mathcal{A}_k$. Let $k+1 \leq i \leq n$ denote the index of the pivot row. Since Gaussian Elimination does nothing but linear combinations between rows, it follows that the initial matrix A_{k+1} satisfies

$$\det\left((e_{j_1} | \ldots | e_{j_k} | e_{l_m})^T A_{k+1}\right) = 0,$$

for any $m \in \{k+1, \ldots, i-1\}$, and

$$\det\left((e_{j_1} | \ldots | e_{j_k} | e_{l_i})^T A_{k+1}\right) \neq 0.$$

This in turn implies that $S_{i+1} = \{j_1, \ldots, j_k, l_i\}$, i.e. that $l_i = j_{k+1}$. □

We now show that GEMS is inherently sequential by proving that the set

$$L'' = \{A : \ A \text{ is } n \times n,\ PA = LU \text{ is the factorization computed by GEMS, and } [U]_{nn} > 0\}.$$

is P-complete. Clearly, that L'' is in P is obvious, so what remains to prove is the following.

Theorem 7.2 *L'' is logspace hard for P.*

Proof. Once more GEMS is in the class $\mathcal{F}$. So we simply give the elementary matrices. This is very easy. Everything is the same as in the first part of the proof of Theorem 6.2, *except for the auxiliary vector od D.* The new definition for $a_D(x, z)$ is

$$a_D(x, z) = (x - z, z, -x, x - z, 0, 0, 0, 0, 0, 0, 0, 0)^T.$$

□

It is an easy but interesting exercise to understand why the second part of Theorem 6.2, which extend the P-completeness result to nonsingular matrices, here does not work (we know that it cannot work, in view of Theorem 7.1).

8 Conclusions and Open Problems

The matrices corresponding, for both Householder's and Givens' algorithms, to a circuit C are singular, in general. More precisely, the duplicator elementary matrix is singular, so that all the matrices that do not correspond to simple formulas (fanout 1 circuits) are bound to be singular. All the attempts we made to extend the proofs to nonsingular matrices failed. The deep reasons of this state of affairs could be an interesting subject per se. To see that the reasons for these failures might be deeper than simply our technical inability, we mention a result of Allender et al. [1] about the "power" of singular matrices. They prove that the set of singular integer matrices is complete for the complexity class $\mathrm{C_{=}L^3}$. The result extends to the problem of verifying the rank of integer matrices. Of course, our work is at a different level: we are essentially dealing with presumably inherently sequential algorithms for problems that parallelize very well (using different approaches). However, the coincidence suggests that nonsingular matrices might not have enough power to map a general circuit. This is the major open problem for the QR algorithms.

Also, for general matrices, it would be interesting to know the status of Householder's algorithm with column pivoting, which is particularly suitable for the accurate rank determination under floating point arithmetic.

For what concerns Givens' rotations, an obvious open problem is to determine the status of other annihilation orderings, especially the ones that proved to be very effective in limited parallelism environments [19, 15]. We suspect that these lead to inherently sequential algorithms as well.

	general matrices	nonsingular matrices	strongly nonsingular matrices
GEP	Inherently Seq.	Inherently Seq.	**Inherently Seq.**
GEM	**Inherently Seq.**	**Inherently Seq.**	NC
GEMS	**Inherently Seq.**	**NC**	NC

Table 1: Parallel complexity of GE with different pivoting strategies and for different classes of input matrices. The results proved in this paper are in boldface.

Finally, Table 1 provides a summary of the known results for the three pivoting strategies investigated in this paper for Gaussian Elimination.

As already mentioned, the results of this paper support the belief that there is a tradeoff between parallelism, on the one hand, and nondegeneracy and accuracy, on the other, in numerical algorithms [5]. We suspect that far deeper work is needed to either prove such a tradeoff on a solid theoretical ground or to exhibit stable algorithms substantially more efficient than the ones adopted by numerical analysts for decades.

References

[1] Allender, E., Beals, R., and Ogihara, M., The complexity of matrix rank and feasible systems of linear equations, in: *Proc. 28th STOC* (1996), 161–167.

[2] Anderson, E. et al. *Lapack User's Guide*, (Society for Industrial and Applied Mathematics, Philadelphia, 1992).

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

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

[5] Demmel, J. W., Trading off parallelism and numerical accuracy, Tech. Rep. CS-92-179, Univ. of Tennessee, June 1992 (Lapack Working Note 52).

[6] Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., *LINPACK Users' Guide*, (Society for Industrial and Applied Mathematics, Philadelphia, 1979).

[7] Eberly, W., Efficient Parallel Independent Subsets and Matrix Factorizations, in: *Proc. 3rd IEEE Symposium on Parallel and Distributed Processing* (1991), 204–211.

[8] Von zur Gathen, J., Parallel Linear Algebra, in: J. Reif, ed., *Synthesis of Parallel Algorithm* (Morgan and Kaufmann Publishers, San Mateo, 1993) 573–617.

[3] A set A is in $C_=L$ provided that there is a nondeterministic logspace bounded Turing machine M such that $x \in A$ iff M has the same number of accepting and rejecting computations on input x.

[9] Golub, G. H. and C. F. Van Loan, *Matrix Computations*, third edition (The Johns Hopkins University Press, Baltimore, 1996).

[10] Greenlaw, R., H. J. Hoover, and W. L. Ruzzo, *Limits to Parallel Computation*, (Oxford University Press, New York, 1995).

[11] Leighton, F. T., *Introduction to Parallel Algorithms and Architectures: Arrays Trees Hypercubes* (Morgan Kaufmann, San Mateo, CA, 1992).

[12] Leoncini, M., On the Parallel Complexity of Gaussian Elimination with Pivoting, *Journal of Computer and System Sciences* **53** (1996), 380–394.

[13] Leoncini, M., Manzini, G., and Margara, L., Parallel complexity of Householder QR factorization, in: *Proc. European Symp. on Algorithms* (1996), Lecture Notes in Computer Science **1136**, 290–301.

[14] *Matlab 5.1 Users' Guide*, The MATHWORKS Inc., 1997.

[15] Modi, J. J., and Clarke, M. R. B., An alternative Givens ordering, *Numer. Math.* **43** (1984), 83–90.

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

[17] Preparata, F. P., and Shamos, M. I., *Computational Geometry* (Springer-Verlag, New York, 1985).

[18] Reif, J. H., $O(\log^2 n)$ Time Efficient Parallel Factorization of Dense, Sparse Separable, and Banded Matrices, in: *Proc. 6th ACM Symp. on Parallel Algorithms and Architectures* (1994) 278–289.

[19] Sameh, A. H. and D. J. Kuck, On Stable Parallel Linear System Solvers, *J. ACM* **25** (1978), 81–91.

[20] Vavasis, S.A., Gaussian Elimination with Pivoting is P-complete, *SIAM J. Disc. Math.* **2** (1989), 413–423.

[21] Wilkinson, J. H., *The Algebraic Eigenvalue Problem* (Clarendon Press, Oxford, 1965).

A Algorithms for matrix factorization

Gaussian Elimination (GE). GE computes the LU decomposition of A (whenever it exists) by determining a sequence of $n-1$ elementary transformations $M^{(k)}$ with the following properties:

$$\begin{cases} A^{(1)} &= A \\ A^{(k+1)} &= M^{(k)} A^{(k)}, & k = 1, \ldots, n-1, \\ a_{ij}^{(k)} &= 0 & i > j \text{ and } j < k, \\ U &= A^{(n)}, \\ L &= \prod \left(M^{(k)} \right)^{-1}. \end{cases}$$

In other words, the transformation $A^{(k+1)} = M^{(k)}A^{(k)}$ sends to zero the elements in column k of $A^{(k)}$ below the main diagonal, leaving the already introduced zeros unchanged. The k-th transformation $M^{(k)}$ is defined as $I - \tau e_k^T$, where

$$\tau^T = (0, \ldots, 0, \tau_{k+1}, \ldots, \tau_n)$$

and $\tau_i = a_{ik}^{(k)}/a_{kk}^{(k)},\ i = k+1, \ldots, n$. If, for some k, $a_{kk}^{(k)} = 0$ the algorithm fails. However, it can be proved that, if A is strongly nonsingular, $a_{kk}^{(k)} \neq 0,\ k = 1, \ldots, n$.

GE with Partial pivoting (GEP). GEP computes a PLU decomposition of A. GEP never fails. As in GE, the matrices L and U are built using a sequence of elementary transformations. However, before applying $M^{(k)}$ to $A^{(k)}$, GEP determines the minimum index h such that

$$|a_{hk}^{(k)}| = \max_{k \leq i \leq n} |a_{ik}^{(k)}|,$$

and swaps the rows k and h of $A^{(k)}$. If the maximum above is 0, the algorithm sets $A^{(k+1)} = A^{(k)}$. The rule used for choosing the index h is an example of *pivoting strategy*, and the row h itself is called the *pivot row*.

GE with Minimal pivoting (GEM). As GEP, with the only difference that h is the minimum among the indices i, $k \leq i \leq n$, such that $a_{ik}^{(k)} \neq 0$. If no such index exists the algorithm sets $A^{(k+1)} = A^{(k)}$.

QR factorization via Householder's reflections (HQR). HQR applies a sequence of $n-1$ elementary orthogonal transformations $Q^{(k)}$ to A, i.e.,

$$\begin{cases} A^{(0)} & = & A & \\ A^{(k+1)} & = & Q^{(k)}A^{(k)} & k = 1, \ldots, n-1, \\ a_{ij}^{(k)} & = & 0 & i > j \text{ and } j < k, \\ R & = & A^{(n-1)}, & \end{cases}$$

Since the $Q^{(k)}$ are orthogonal, we can write

$$A = (Q^{(1)})^T (Q^{(2)})^T \ldots (Q^{(n-1)})^T R = QR,$$

which represents the factorization computed by the algorithm. The matrix $Q^{(k)}$ is defined as follows. Let

$$a = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{kk}^{(k-1)} \\ \vdots \\ a_{nk}^{(k-1)}, \end{pmatrix}, \quad \text{and define} \quad v = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{kk}^{(k-1)} \\ \vdots \\ a_{nk}^{(k-1)}, \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \theta\sqrt{a^T a} \\ \vdots \\ 0 \end{pmatrix}, \tag{6}$$

where $\theta = a_{kk}^{(k-1)}/|a_{kk}^{(k-1)}|$ is the sign of $a_{kk}^{(k-1)}$. Then

$$Q^{(k)} = \begin{cases} I - \frac{2}{v^T v} vv^T & \text{if } v \neq 0, \\ I & \text{otherwise.} \end{cases} \tag{7}$$

The important point to observe is that there are other possible strategies for choosing θ in (6) that would produce the same effect of annihilating the kth column. However, the choice adopted here is to be preferred for stability reasons.

QR factorization via Givens rotations (GQR). GQR applies to general real matrices. It computes a sequence of $\frac{n(n-1)}{2}$ transformations (called *rotations*), such that each transformation annihilates one element below the main diagonal, leaving all the already introduced zeros unchanged. GQR annihilates the subdiagonal part of the matrix in the natural order (left to right and top to bottom).

The rotation used to annihilate a selected entry a_{ji} of a matrix A is the orthogonal matrix G_{ij} defined as follows:

$$G_{ij} = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \begin{matrix} \\ \\ \longleftarrow i \\ \\ \longleftarrow j \\ \\ \\ \end{matrix}$$

where $c = \frac{a_{ii}}{\sqrt{a_{ii}^2+a_{ji}^2}}$ and $s = \frac{-a_{ji}}{\sqrt{a_{ii}^2+a_{ji}^2}}$. One can easily verify that G_{ij} is indeed orthogonal and that the entry j,i of $G_{ij} \cdot A$ is zero.

B Floating point number representation

A floating point system $\mathcal{F}$ is characterized, on a particular computer, by four integers: the *base* b of the representation (usually $b = 2$), the *precision* t, and the *range* of the exponent $[\ell, L]$. A number $f \in \mathcal{F}$ has the form

$$f = \pm .b_1 b_2 \ldots b_t \times b^e,$$

where $0 \leq b_i < b$ and $\ell \leq e \leq L$. It is also required that $b_1 \neq 0$, which is a normalization condition. The sequence $.b_1 b_2 \ldots b_t$ is the *significand* while e is the *exponent*.

If $\mathrm{fl}(x)$ is the (rounded or chopped) floating point representation of a real x, then

$$\mathrm{fl}(x) = x(1+\delta),$$

where $|\delta| \leq \epsilon$ and

$$\epsilon = \begin{cases} \frac{1}{2}b^{1-t} & \text{for rounded arithmetic} \\ \\ b^{1-t} & \text{for chopped arithmetic.} \end{cases}$$

ϵ is the so called *machine precision* or *roundoff unit* and is used for roundoff error analysis. In particular, let $\odot$ denote the floating point implementation of the arithmetic operation $\cdot \in \{+, -, \times, /\}$. If $x, y \in \mathcal{F}$, then

$$x \odot y = \mathrm{fl}(x \cdot y) = (x \cdot y)(1+\eta), \tag{8}$$

where $|\eta| \leq \epsilon$. (8) is known as the *standard model* of arithmetic. A property that plays a crucial role in our reduction is the following. Let a and b be floating point numbers such that $|b| < \epsilon|a|$. Then $a \oplus b = a$.

C Proof of Theorem 6.1

The set is clearly in P. Now, given a NAND circuit C with n inputs and gates, we exhibit a strongly nonsingular matrix $\mathbf{A}_C$ such that, for $k = 1, \ldots, n$, if the output of node k of C (either an input or a NAND gate) is `True` the pivot for step $2k-1$ will be taken from row $2k-1$, or else the pivot will be taken from row $2k$.

Let $\mathbf{M}_C$ denote the matrix corresponding to the circuit C according to Vavasis' proof [20] (see Figure 9). If the circuit has n inputs and gates, the matrix $\mathbf{M}_C$ has order $2n$. However, the circuit simulation takes place while performing the elimination of the first n columns, with columns $n+1, \ldots, 2n$ having the only purpose of ensuring nonsingularity. For example, in the matrix of Figure 9 the first two columns correspond to the inputs a, b, while, for $i = 3, \ldots, 6$, column i corresponds to gate $i-2$. For the circuit inputs a logical value `False` is represented by the value 4.0, while a logical value `True` by the value 3.75. When one executes GEP on $\mathbf{M}_C$, a logical value `False` for a certain input or gate yields a pivot 4.0 in the corresponding column, whereas a logical value `True` yields a pivot -3.9. Therefore, the sequence of rows selected by GEP provides us the output of each gate in the circuit C.

Vavasis' proof is based on the observation that a NAND gate outputs `False` unless one of its inputs is `False`. This fact is mirrored in the matrix $\mathbf{M}_C$ where the pivot of each column is 4.0 unless that value is modified in a previous elimination step. The structure of the matrix is such that the processing of a pivot 4.0, corresponding to a `False` output, changes the status of the gates receiving that output from `False` to `True`. This change of status is achieved by subtracting 0.25 from an entry initially set to 4.0. This ensures that in the due time the pivot in the corresponding column will be -3.9. For example, in the matrix of Figure 9 the selection of the pivot 4.0 in column 4 (corresponding to a `False` output for gate 2) reduces the 4.0 entry in column 6 by 0.25 ensuring that the pivot for that column will be -3.9 (corresponding to a `True` output for gate 4). Note that the 1's in the first n columns of $\mathbf{M}_C$ corresponds to the wires of the circuit, one pair of 1's for each wire. Below and to the right of each 4.0 entry there is a number of 1's equal to the fan-out of the corresponding gate which in the following we assume is at most two.

Our matrix $\mathbf{A}_C$ has order $3n$. The $2n \times 2n$ leading principal submatrix of $\mathbf{A}_C$, denoted by $\mathbf{A'}_C$, is the *main submatrix*. The odd-numbered columns of $\mathbf{A'}_C$ are precisely the columns of the first half of the matrix $\mathbf{M}_C$. The even-numbered columns of $\mathbf{A'}_C$ are called the *auxiliary columns*; for $k = 1, \ldots, n$, column $2k$ of $\mathbf{A'}_C$ contains the entry 10.0 in position $2k$ and zero elsewhere. Outside the main submatrix, the entries a_{ij} of $\mathbf{A}_C$ are all zero except for $a_{2n+k,2k} = 20.0$ and $a_{2n+k,2n+k} = 30.0$, $k = 1, \ldots, n$. For example, for the circuit of Figure 9, the corresponding matrix $\mathbf{A}_C$ is shown in Figure 10. Note that $\mathbf{A}_C$ is strongly diagonally dominant, hence strongly nonsingular.

Define the *circuit area* of the matrix $\mathbf{A}_C$ to be the set of odd-numbered columns of the main submatrix. Also, for $i = 1, \ldots, n$, let

$$w_i = \sum_{j=2i+1}^{2n} |a_{2n+i,j}|.$$

The value w_i can be seen as the *weight* of certain entries in row $2n+i$ (see Fig. 13).

The proof of the theorem is now a consequence of the following two lemmas.

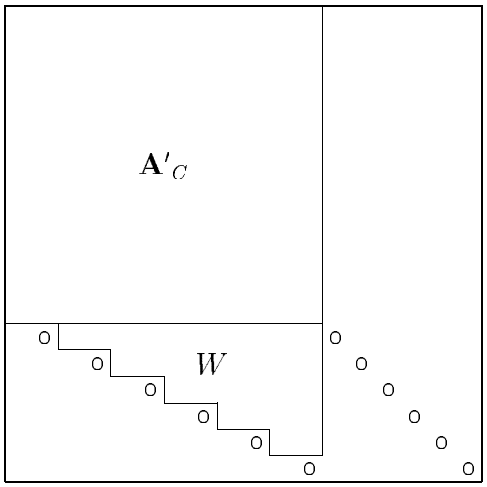

Figure 13: The structure of the matrix $\mathbf{A}_C$. The area labeled W contains the entries which contribute to the weights w_i's. The symbol o denotes the position of the initial nonzero entries outside the main submatrix $\mathbf{A'}_C$.

Lemma C.1 *Let A be a matrix such that the first nonzero element of row i is a_{ij} with $j \leq i$. Then, the first $j-1$ steps of GEP do not affect row i.* □

Lemma C.2 *For $k = 1, \ldots, n$ the following facts hold of GEP on input A_C:*

(a) the pivot for step $2k-1$ is either -3.9 or 4.0;

(b) step $2k-1$ modifies the circuit area as the kth step in the elimination of matrix $\mathbf{M}_C$;

(c) step $2k-1$ does not affect the auxiliary (even numbered) columns with index greater than $2k$;

(d) the pivot for step $2k$ is 20.0;

(e) step $2k$ does not affect the circuit area nor the auxiliary columns with index greater than $2k$;

(f) at the end of step $2k$, $w_i \leq 2.5$, for $i = 1, 2, \ldots, n$.

Proof. The proof is by complete induction on k. The basis is trivial. For the induction hypothesis, let $k > 1$ and suppose that (a) through (f) hold for any $i < k$.

Consider step $2k-1$ of GEP. By induction hypothesis, the entries in column $2k-1$ with row index less than $2n$ contain the same values generated during the elimination process on the matrix $\mathbf{M}_C$. This means

$$a_{2k-1,2k-1} = -3.9 \quad \text{and} \quad a_{2k,2k-1} = \begin{cases} 4.0 & \text{if the output of gate } k \text{ is } \texttt{False}, \\ 3.75 \text{ or } 3.50 & \text{otherwise.} \end{cases}$$

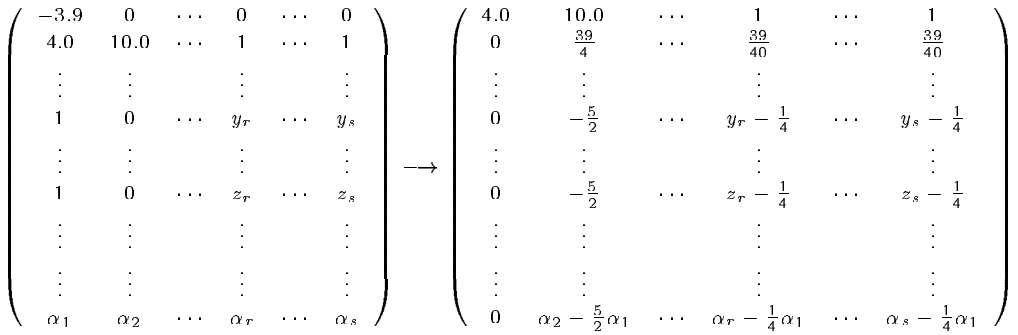

Figure 14: The effect of the elimination of column $2k-1$ over columns $2k, r, s$ when the pivot is 4.0. Since the fan-out of each gate is at most 2 no other column in the main submatrix is affected. We also show how step $2k-1$ modifies a row outside the main submatrix.

Moreover, for $2k < i \leq 2n$, we have $|a_{i,2k-1}| \leq 1.5$ (see part (d) of Lemma 3.2 in [20]). For the entries with row index larger than $2n$, we have, for $2n < i < 2n+k$,

$$|a_{i,2k-1}| \leq w_{i-2n} \leq 2.5,$$

because of (f), and, for $i \geq 2n+k$, we have $a_{i,2k-1} = 0$ by Lemma C.1. It follows that the pivot is 4.0 if the output of the gate k is `False`, and -3.9 otherwise, hence proving part (a) and, consequently, part (b).

To prove (c) we first observe that the pivot at step $2k-1$ is either $a_{2k-1,2k-1}$ or $a_{2k,2k-1}$. By induction hypothesis we know that the first $2k-2$ elimination steps did not affect the auxiliary columns $2k, 2k+2, \ldots, 2n$. Hence, for j even, $2k < j \leq 2n$, at the beginning of step $2k-1$ both $a_{2k-1,j}$ and $a_{2k,j}$ are zero. It follows that, regardless of the pivot, step $2k-1$ does not affect column j.

In order to prove (d), consider step $2k$ of GEP. We need to show that the element with the largest modulus in column $2k$ at the beginning of step $2k$ is $a_{2n+k,2k}$ which, by Lemma C.1, is equal to 20.0. Also by Lemma C.1, we know that at step $2k$ we have $a_{i,2k} = 0$ for $2n+k < i \leq 3n$. Hence, we only need to prove that $|a_{i,2k}| < 20.0$ for $2k \leq i < 2n+k$. We consider two cases. If at step $2k-1$ the pivot is -3.9, the annihilation of column $2k-1$ does not affect column $2k$. Hence, at the beginning of step $2k$ we have: $a_{2k,2k} = 10.0$, $a_{i,2k} = 0$, for $2k < i \leq 2n$, and, for $2n < i < 2n+k$, $|a_{i,2k}| \leq w_{i-2n} \leq 2.5$. Vice versa, if at step $2k-1$ the pivot is 4.0, the elimination of column $2k-1$ does affect column $2k$. As a result, at the beginning of step $2k$ we have $a_{2k,2k} = 39/4$, and two entries in column $2k$ are equal to $-5/2$. For the entries $a_{i,2k}$, with $2n < i < 2n+k$, we have (see Figure 14) $a_{i,2k} = \alpha_2 - \frac{5}{2}\alpha_1$, where $|\alpha_i| \leq 2.5$ (by part (f) of the induction). Hence, $a_{2n+k,2k} = 20.0$ is the largest entry in column $2k$ and is the pivot chosen by GEP.

To prove (e) we simply note that all the entries in the pivot row with column index less than or equal to $2n$ are zero. Therefore, the annihilation of column $2k$ does not affect the main submatrix outside column $2k$.

To prove (f), we fix an index i and we analyze how steps $2k-1$ and $2k$ affect the weight $w_i = \sum_{j=2i+1}^{2n} |a_{2n+i,j}|$. Again we consider two cases depending on the pivot chosen at step

$2k-1$. If this is -3.9, GEP sends $a_{2n+i,2k-1}$ to zero without affecting the other entries in row $2n+i$. Similarly, step $2k$ sends $a_{2n+i,2k}$ to zero without side effects. Thus, the weight w_i remains bounded by 2.5. A special case is when $i=k$. In fact, at the beginning of step $2k$ GEP swaps rows $2k$ and $2n+k$. In this case the new value w_k depends on the entries $a_{2k,j}$, with $2k < j \leq 2n$. By parts (b), (c), and (e) of the induction, one can see that at the beginning of step $2k$ there are at most four entries $a_{2k,j} \neq 0$ and that $\sum_{2k<j\leq 2n} |a_{2k,j}| \leq 2.5$. Hence, at the end of step $2k$ we have $w_k \leq 2.5$, as claimed. Suppose now that the pivot at step $2k-1$ is 4.0. In this case the effects of step $2k-1$ on row $2n+i$, $i \neq k$, are shown in Figure 14. Since step $2k$ simply sends to zero the entry in column $2k$ (i.e. $\alpha_2 - \frac{5}{2}\alpha_1$), the weight w_i decreases by an amount at least $|\alpha_1|/2 + |\alpha_2|$. Finally, for $i=k$, reasoning as for the previous case we get $w_k \leq 2.5$. □