On the Parallel Complexity of Matrix Factorization Algorithms*

Mauro Leoncini† Giovanni Manzini‡ Luciano Margara§

Abstract

We prove a number of negative results about practical (i.e., numerically accurate) algorithms for certain matrix factorizations. In particular, we prove that the popular Givens' method for computing the QR decomposition is inherently sequential over the realistic model of floating point arithmetic. We also prove a number of additional results concerning Gaussian Elimination for computing the LU decomposition. Altogether, the results of this paper support 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 [9, 12]. 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. Among the computations that involve matrix factorizations of some sort we recall linear system solution, eigenvalue and least squares approximation, and rank revealing transformations. In view of this, 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[1] [13, 15].
- QR decomposition is in arithmetic NC for matrices with full column rank, since it easily reduces to LU decomposition of strongly nonsingular matrices [13].
- PLU decomposition is in arithmetic NC for nonsingular matrices [5]. A permutation matrix P such that $P^T A$ is strongly nonsingular can be found with the algorithm for solving the Lexicographically First Maximal Independent Subset (LFMIS) problem [2].
- $QR\Pi$ factorization of an arbitrary matrix A is in arithmetic NC [5]. A permutation Π such that the leftmost $n \times r$ submatrix of A has full column rank, for $r = \text{rank}(A)$, 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 (say, double precision floating point). The widespread belief in numerical analysis circles is that these methods (as well as other fast parallel solvers, not based on factorizations [3]) are highly unstable, and in fact that there is a tradeoff between accuracy and the degree of achievable parallelism [4]. The results of this paper, which apply to the classical, numerically stable algorithms for matrix factorization, confirm this state of affairs.

Already in 1989 Vavasis proved that Gaussian Elimination with Partial pivoting (GEP), which is the standard method for computing the PLU decomposition in practice, is inherently sequential[2]. [17, 10]. This was only a starting point, however. For instance, to solve systems of linear equations there were other algorithms that were as stable as GEP and that appeared more advantageous from the viewpoint of parallelism. We refer, in particular, to the classical Householder's and Givens' QR decomposition methods (HQR and GQR, for short [7]). Actually, GQR is to date the best choice for solving dense systems of linear equations efficiently and stably in parallel [16]. 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. Recently, we proved that HQR is inherently sequential on input general matrices [11].

*This work has been supported by Esprit Project 9072 GEPPCOM, and by Murst 40% and 60%funds.

†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.

‡Dipartimento di Scienze e Tecnologie Avanzate, Università di Torino, Via Cavour 84, 15100 Alessandria, Italy, and IMC-CNR, Via S. Maria 46, 56126 Pisa, Italy. Email: manzini@unial.it.

§Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy. Email: margara@cs.unibo.it.

[1] In this case we will say that the matrix is *strongly nonsingular*.

[2] As is well known, P-complete problems are the hardest ones to parallelize in a precise technical sense. For this reason they are sometimes called "presumably inherently sequential" or just "inherently sequential" (see, e.g., [14]).

In this paper we prove a number of new results that give a more complete picture of the pervasiveness of the tradeoff phenomenon mentioned above.

1. We prove that GQR is inherently sequential. We exhibit a reduction from NAND Circuit Value Problem (NANDCVP) with fanout ≤ 2. The main technical difficulty of our proof stems in the arithmetic model. For HQR it was possible to exhibit a reduction in which the matrix had rational entries and prove that the execution of HQR on a rational model of arithmetic returned the exact result (to be interpreted as the output of the circuit) [11]. We have been unable to prove that this nice state of affairs holds for GQR as well. This made it necessary a more involved analysis of the behavior of the algorithm under finite arithmetic. Our facing this obstacle eventually had a positive byproduct, since the present result holds for the very realistic model of floating point arithmetic (indeed, it applies to the implementations of GQR actually available in both LAPACK and MATLAB). This result shows that all the practical options for solving linear systems share the same status of inherent sequentiality.

2. We extend Vavasis' result by proving that GEP is inherently sequential on 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 GE (which admits NC implementations) does not fail on input these matrices, but it is often unstable. Hence our result shows once more that there is a price to pay for accuracy.

3. We study weaker forms of pivoting for Gaussian Elimination. Our goal is to investigate whether *any* PLU decomposition could be computed fast in parallel. We define GE with Minimal pivoting (GEM), in which the selected pivot at step k is the lowest indexed nonzero element in column k of the submatrix being triangularized. We prove that, independently of the row permutation strategy adopted, GEM is inherently sequential, thus suggesting that we pay a price for the algorithm nondegeneracy. We prove, however, that a nonstandard permutation strategy allows NC implementation of GE on input nonsingular matrices. We call GEMS (for GEM with circular Shift) this last version of Minimal pivoting.

Table 1 provides a summary of the known results for the three pivoting strategies investigated in this paper for Gaussian Elimination. The results proved in this paper are in boldface.

The rest of this paper is organized as follows. In Section 2 we describe the key ideas that are common to the P-completeness proofs for all the factorization methods considered in the paper. In Section 3 we address Gaussian Elimination and in Section 4 we take QR decomposition into account. We present some basic linear algebra material in Appendix A.

2 A framework for reductions to matrix computations

The P-completeness results we will prove in the following sections are all based on reductions from the NANDCVP, a restricted version of CVP which we now briefly recall:

Input: the encoding 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 $C(x_1, \ldots, x_k)$ that would be computed by C on input $x_1, \ldots, x_k$.

NANDCVP is log-space complete for P (P-complete for short), as reported in [8]. In order to simplify the proofs, we assume that each gate has fanout at most two. This is not a loss of generality; in fact, for any vertex v whose fanout exceeds two, we may replace the edges from v with a binary tree with v as the root and v's sons as the leaves. Let S be the size of the original circuit (possibly with unbounded fanout); then it can be easily proved that the size of the modified circuit is $O(S^2)$. Moreover, the transformation is clearly log-space computable.

Hence, to prove that a given factorization algorithm $\mathcal{A}$ is P-complete, we will show that, given an instance of (fanout 2) NANDCVP, we can efficiently build a matrix A_C such that the execution of $\mathcal{A}$ on input A_C "simulates" the computation of C on input $x_1, \ldots, x_k$. More precisely, we will show that there is a log-space computable function f such that, given a pair $\langle \mathbf{C}, \mathbf{x} \rangle$ where $\mathbf{C}$ is the encoding of a fanout 2 NAND circuit C with k inputs and n gates, and $\mathbf{x}$ is the encoding of a k-ary boolean vector x, the following holds:

1. $f(\langle \mathbf{C}, \mathbf{x} \rangle) = \langle A_C, \nu \rangle$, where A_C is an $\nu \times \nu$ matrix, for some $\nu = n^{O(1)}$;

2. $\mathcal{A}$ on input A_C leaves the encoding of $C(x)$ in the entry $A_C(\nu, \nu)$.

The reductions share a common structure, based on the definition of certain submatrices, or blocks (corresponding to circuit components), and the rules according to which the blocks are pieced together to form the matrix A_C (corresponding to the assembly rules of the circuit components). The description of this general framework is the goal of the rest of this section.

2.1 Functional blocks

Let $\mathbf{a}$ denote the encoding of the logical value $a \in$ {`True`, `False`}.

NAND block (N) An N block computes the encoding of the NAND of two logical values whose encodings are $\mathbf{a}$ and $\mathbf{b}$. That is, if $N_{11} = \mathbf{a}$ and $N_{22} = \mathbf{b}$, then executing $\mathcal{A}$ on input N leaves in the bottom right entry the encoding of $\mathtt{NAND}(a, b)$.

Duplicator block (D) A block of type D duplicates the encoding $\mathbf{a}$ of a logical value. That is, if $D_{11} = \mathbf{a}$, executing $\mathcal{A}$ on input D leaves in the bottom right corner the 2×2 block $\begin{pmatrix} \mathbf{a} & 0 \\ 0 & \mathbf{a} \end{pmatrix}$.

Wire block (W) A block of type W simply copies the encoding of one or two logical values to different diagonal entries.

2.2 Block assembly

The simple idea behind the simulation of a NAND circuit C consists essentially of having an N block for any NAND gate of C. A fanout 2 NAND gate is simulated by placing a D block just after the corresponding N block, and a connection between a NAND gate (either fanout 1 or 2)

	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.

and the next NAND gate(s) is represented by placing a W block after the N (or D) block. Altogether, any pair of two consecutive diagonal blocks of A_C is of one of the following kinds: $\langle N, W\rangle$, $\langle N, D\rangle$, $\langle D, W\rangle$, and $\langle W, N\rangle$.

Of course, the above scheme cannot work in that simple way. The blocks must be partially overlapped, for otherwise no value could be passed around. Also, W blocks will be in general split apart in the matrix A_C, to make it possible to copy values to arbitrarily distant entries. Fig. 1 illustrates the basic composition rules to form a fanout 2 NAND gate and to route the output of fanout 1 and 2 NAND gates to subsequent NAND gates. In particular, note that the W block is a principal minor of the matrix but not one formed by contiguous rows and columns.

To illustrate the implications of these assembly rules on the block definition, consider the factorization algorithm working on a given diagonal block. When managing to send to zero the entries in the last row(s) of some block (marked with $*$ in Fig. 1), the algorithm will in general affect the entries of the first row(s) of the next block. This is exactly what we want for the first such entries (denoted by $\circ$), which are the places where results are passed between blocks. However, this is unwanted for the remaining entries (marked with $\times$). Hence, when specifying a given block we will have to show that the factorization algorithm leaves, in the entries marked with $\times$ in Fig. 1, the appropriate values required for the elimination of the next block.

3 Gaussian Elimination with Pivoting

In this section we consider three pivoting strategies for Gaussian elimination, namely *Partial Pivoting*, *Minimal Pivoting*, and *Minimal Pivoting with circular Shift*. The resulting algorithms will be referred to as *GEP*, *GEM*, and *GEMS*, respectively. GEP is well-known (see also the Appendix A). According to Minimal Pivoting and Minimal Pivoting with circular Shift, the pivot row is the lowest indexed one having a nonzero value in column k. Once the pivot row is found, GEM swaps it with row k (like GEP); on the other hand, GEMS brings it to position k, without altering the order of the other rows (this amounts to performing a circular shift of the rows with indices between k and the index of the pivot row).

We prove here that (1) GEMS is inherently sequential, unless the input matrix is nonsingular; (2) GEM is inherently sequential, unless applied to strongly nonsingular matrices; (3) GEP is inherently sequential even on strongly nonsingular matrices.

3.1 Minimal Pivoting

Minimal Pivoting is intended to be a weak form of pivoting, with the only aim at making the resulting Gaussian Elimination algorithm nondegenerate, but with absolutely no guarantee of numerical accuracy. However, the nondegeneracy guarantee is sufficient to make the resulting GE algorithm presumably very hard to parallelize.

Theorem 3.1 *The algorithms GEMS and GEM are inherently sequential on general matrices.*

Proof. According to the general framework discussed in Section 2, we show the specific functional blocks for GEMS and GEM. The reductions use the encodings 0 and 1 for the boolean values `False` and `True`, respectively.

- D blocks for GEMS and GEM are depicted in Figure 2. Note that the only difference is in the additional (rightmost) column, the diagonal blocks being the same. After 9 steps of either algorithm (on the appropriate matrix) we have $D_{10,10} = D_{11,11} = \mathbf{a}$, $D_{11,10} = D_{10,11} = 0$, $D_{10,12} = v$, and $D_{11,12} = w$.
- An N block is represented by the matrix depicted in Figure 3.

  Recall that, in view of the rules given in Section 2, we must prove that the execution of the factorization algorithms leaves the appropriate values in the last row, i.e., the ones required by the next (partially overlapped) block. For this reason we always add one column to the description of the blocks. In case of N blocks it is easy to see that after 4 steps of either GEMS or GEM $N_{5,5}$ contains the encoding of $\texttt{NAND}(a, b)$ while $N_{5,6}$ still contains v.
- The W block, which applies to both GEMS and GEM, is depicted in Figure 3. It "connects" the (one) output of a NAND gate to another NAND gate. After 2 steps of either GEMS or GEM we obtain $W_{3,3} = \mathbf{a}$ and $W_{3,4} = v$.

The matrix A_C can be built using $O(\log n)$ space. Clearly, the only difficulty in generating each block stems in the determination of its position in the matrix A_C. To generate a W block it is required to know the positions of the two blocks it connects (either two N blocks or a D and an N block). In turn, to generate a D block it is required to know the position of the preceding N block. Overall, the problem is the efficient (i.e. log-space) computation of the position of all the N blocks. Now, referring to Fig. 1, it is easy to see that the top left entry of the N block corresponding to the jth NAND gate of the circuit is placed at position (p_j, p_j), where

$$p_j = (s(N)+1)f_j^1 + (s(N)+S(D)+1)f_j^2,$$

$s(\cdot)$ is the order of the referred block, and f_j^1 and f_j^2 denote the number of fanout 1 and fanout 2 NAND gates with indices smaller than j. This computation can be clearly performed in $O(\log n)$ space. Assuming, that the output of the

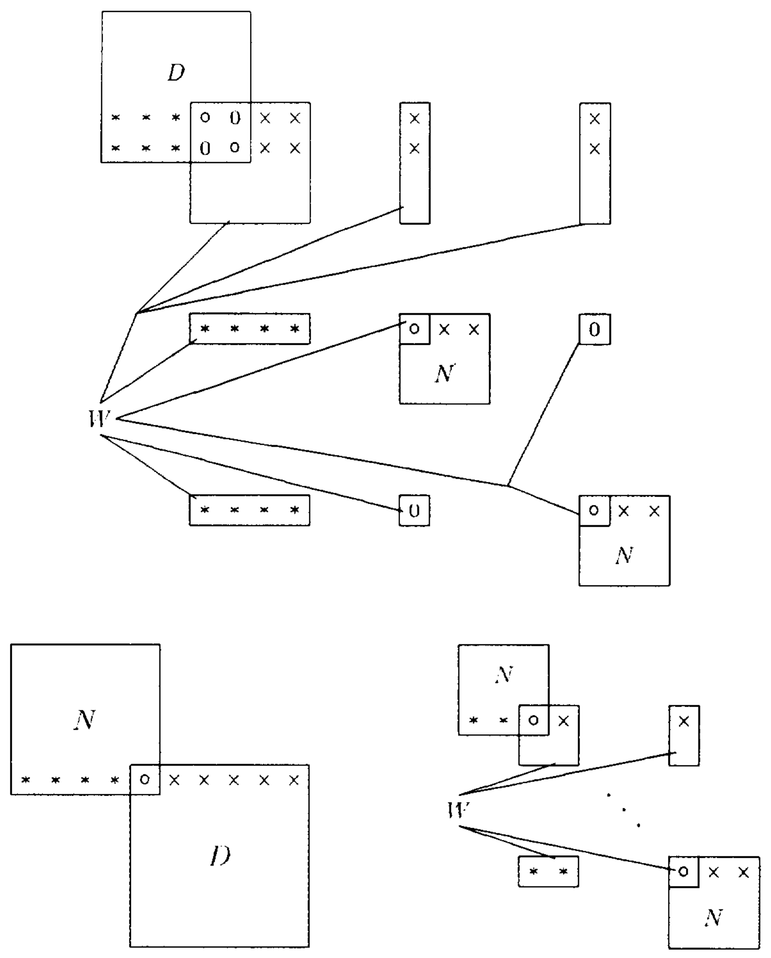


Figure 1: Block assembly: fanout 2 NAND gate (left), wiring for a fanout 1 NAND gate (right), and wiring for a fanout 2 NAND gate (top).

circuit is always taken from the output of the nth NAND gate, the order of A_C is simply given by p_{n+1}. □

The matrix A_C of Theorem 3.1 is clearly singular (having a number of zero columns). Interestingly, however, when restricted to nonsingular matrices the two algorithms appear to have quite different properties with respect to parallelism.

Corollary 3.2 *GEM is inherently sequential when restricted to nonsingular matrices.*

Proof. Given (the encoding) of C we apply the same reduction as in Theorem 3.1, obtaining a singular matrix A_C of order ν. We then consider the following matrix A'_C of order 2ν.

$$A'_C = \begin{pmatrix} A_C & E \\ E & O \end{pmatrix}, \qquad (1)$$

where O is the zero matrix of order ν and E is the matrix (of order ν) with 1 on the antidiagonal and 0 elsewhere. The determinant of A'_C can be easily proven to be ± 1. Moreover, the execution of GEM on input A'_C leaves the encoding of the output of C in the entry (ν, ν). This can be seen by observing that the first $\nu - 1$ steps of GEM modifies A_C (the interesting portion of A'_C for what concerns the simulation) in the same way as in the reduction of Theorem 3.1. To this end, consider step k of GEM. If column k contains a nonzero element with index not greater than ν, then clearly the kth elimination step modifies A_C exactly as in the previous reduction. Otherwise the pivot is taken from the row with index $\nu + k$, and the elimination step has no effect on A_C since the pivot is the only nonzero element in row $\nu + k$. Then, as the sole consequence of this step, row k is brought to position $\nu + k$, where it cannot affect the simulation any more (due to the structure of A'_C). Again this is exactly what happens in the previous reduction. □

Theorem 3.3 *Computing the PLU factorization returned by GEMS on input a nonsingular matrix is in arithmetic NC^2.*

Proof. An NC^2 algorithm to compute the PLU factorization of nonsingular matrices has been given by Eberly [5]. 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 [2]). 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 $P^T A$

$$D = \begin{pmatrix} \mathbf{a} & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & v - w \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -v \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & v - w \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad D = \begin{pmatrix} \mathbf{a} & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & v \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Figure 2: The D blocks for GEMS (left) and GEM (right).

$$N = \begin{pmatrix} \mathbf{a} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{b} & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 & 0 & v \end{pmatrix}, \quad W = \begin{pmatrix} \mathbf{a} & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & v \end{pmatrix}.$$

Figure 3: The N and W blocks for GEMS and GEM.

has LU factorization is simply

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

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

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

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 $e_{l_{k+1}}, \dots, e_{l_n}$ extend $e_{j_1}, \dots, e_{j_k}$ to form a permutation matrix. Clearly, Minimal Pivoting ensures that $l_{k+1} < \dots < l_n$. 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

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

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

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

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

Clearly GEMS and GEM behave the same when fed with strongly nonsingular matrices. Actually they compute the (unique) LU factorization, which is guaranteed to exist, without performing any row exchange. Hence, both GEMS and GEM, as well as plain GE, can be placed in NC under the assumption of strong nonsingularity.

3.2 Partial Pivoting

We now prove that GEP is inherently sequential on strongly nonsingular matrices. Our proof builds on the original proof in [17], 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. From the numerical point of view this is important, since, as already remarked in the introduction, for strongly nonsingular matrices plain GE (no pivoting) is known not to fail.

Theorem 3.4 *The set $L = \{(i, j, A)$: On input A, GEP uses row i to eliminate column j.$\}$, where A is strongly nonsingular, is log-space complete for P.*

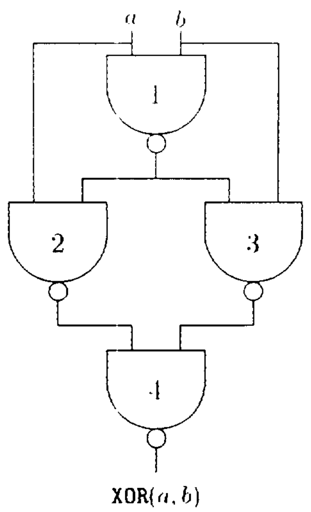
While we postpone the technical proof of Theorem 3.4 to the full paper, we give an example which shows the way the matrix given in [17] is modified. Figure 4 depicts a circuit for computing the Exclusive Or of two boolean values and the corresponding matrix $\mathbf{M}_C$ obtained according to the reduction in [17]. 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 3.4, is shown in Figure 5. It can be easily seen that our matrix is strongly diagonally dominant, hence strongly nonsingular.

4 QR decomposition through Givens' rotations

We backtrack to our general framework of Section 2 and prove that GQR is inherently sequential under the most realistic model of arithmetic.

Theorem 4.1 *GQR is inherently sequential under a fixed size floating point model of arithmetic.*

Proof. GQR is clearly in P under the model considered. We prove that the output of a fanout 2 NANDCVP instance can be read off the entry in position (n, n) of the triangular factor computed by GQR on input a given matrix A of order n (determined according to the rules of Section 2). The



$$\left(\begin{array}{cccccccccccc}
-3.9 & & & & & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\mathbf{a} & & 1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
 & -3.9 & \cdot & & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \mathbf{b} & 1 & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
 & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & 1 & 4.0 & 1 & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & 1 & 4.0 & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & 1 & \cdot & 4.0 & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 1 & 1 & 4.0 & \cdot & \cdot & \cdot & \cdot & \cdot & 1
\end{array}\right)$$

Figure 4: Circuit for computing **XOR**(a,b) (left) and the corresponding matrix $\mathbf{M}_C$ according to Vavasis' reduction (right). The symbol $\cdot$ denotes a zero entry.

$$\left(\begin{array}{cccccccccccccccccc}
-3.9 & & & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\mathbf{a} & 10.0 & & \cdot & 1 & & 1 & & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \mathbf{b} & 10.0 & 1 & \cdot & & & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & 1 & \cdot & 4.0 & 10.0 & 1 & & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & \cdot & \cdot & 1 & & 4.0 & 10.0 & & & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & \cdot & 1 & & & & 4.0 & 10.0 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & & 1 & & 4.0 & 10.0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 20.0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 30.0 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 20.0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 30.0 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 20.0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 30.0 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 20.0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 30.0 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 20.0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 30.0 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 20.0 & \cdot & \cdot & \cdot & \cdot & \cdot & 30.0
\end{array}\right)$$

Figure 5: The matrix $\mathbf{A}_C$ for the computation of **XOR**(a,b).

reduction uses the encodings -1 and 1 for the boolean values `False` and `True`, respectively. We first define the functional blocks for *the real number model*, then study the behavior of the algorithm in floating point arithmetic.

- A D block is represented by the matrix of Figure 6. If $\mathbf{a}$ is either 1 or -1, after 4 steps of exact GQR we get $D_{4,4} = D_{5,5} = \mathbf{a}$, $D_{4,5} = D_{5,4} = 0$, $D_{4,6} = x$, and $D_{5,6} = y$.
- After 2 steps of GQR applied to the block of Figure 7 we get $W_{2,2} = \mathbf{a}$ and $W_{2,3} = x$.
- The N block is depicted in Figure 8. Performing 10 steps of GQR on the matrix of Figure 8 leaves with $N_{10,10} =$ **NAND**(a,b) and $N_{10,11} = x$.

Applying a floating point implementation of GQR to any of the above blocks[3] leads to approximated results. For instance, the relative error affecting the sign of the result of an N block ranges from a minimum of ϵ to a maximum of 13ϵ on a PC version of MATLAB (with ϵ, the roundoff unit, equal to $2.2204 \cdot 10^{-16}$). Clearly, for matrices simulating circuits with many gates, the error will in general amplify to make it impossible to recover the exact (i.e., 1 or -1) result. Clearly, classical error analysis cannot help in general, since it should be performed for all the possible matrices representing NANDCVP instances.

Our solution is to slightly modify the exact blocks given above so that they always return the exact results when the elimination is performed under machine arithmetic. We take advantage of the following crucial properties of the floating point arithmetic:

1. $a + b = a$, if $|b| < \epsilon|a|$;
2. $|x| < \omega \Rightarrow x$ is a "machine zero".

Here ϵ is the roundoff unit while ω is the smallest magnitude representable number. The other crucial fact that helps in our analysis is that only the values 1 and -1 need to be passed from one block to another during the elimination (this can be seen by considering the first row in each of the exact blocks).

[3]More precisely, to the exact up to machine precision approximations of the blocks

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

Figure 6: The D block for GQR.

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

Figure 7: The W block for GQR.

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

$$\begin{aligned} \alpha &= \left(\frac{-16625\sqrt{\frac{5}{2}}}{36} + \frac{875\sqrt{\frac{15}{2}}}{8} - \frac{4875\sqrt{91}}{128} + \frac{2375\sqrt{371}}{72}\right)/14, \\ \beta &= \frac{-375\sqrt{\frac{13}{7}}}{64} - \frac{125\sqrt{\frac{5}{2}}}{9} + \frac{125\sqrt{\frac{15}{2}}}{64} + \frac{125\sqrt{\frac{53}{7}}}{18}, \\ \gamma &= \left(\frac{-14875\sqrt{\frac{5}{2}}}{36} + \frac{2625\sqrt{\frac{15}{2}}}{32} - \frac{3625\sqrt{91}}{128} + \frac{2125\sqrt{371}}{72}\right)/14, \\ \delta &= \left(\frac{-625\sqrt{\frac{5}{2}}}{9} + \frac{875\sqrt{\frac{15}{2}}}{64} - \frac{625\sqrt{91}}{128} + \frac{125\sqrt{371}}{24}\right)/2. \end{aligned}$$

Figure 8: The N block for GQR.

We show our solution for the W block (the same technique applies to the other blocks as well). Let m' be the number of bits of the mantissa for the particular arithmetic under consideration and set $m = m' + 10$ [4]. Consider the following modified W block:

$$W = \begin{pmatrix} \mathbf{a} & 1 & 1 & 0 & 0 & 1 \\ 2 & 3 & \frac{-3+\sqrt{65}}{4} & 0 & 0 & 3 - \frac{15\, 2^m}{4} - \frac{\sqrt{65}\, 2^m}{4} \\ 2 & 4 & \frac{-1+\sqrt{65}}{2} & 0 & 0 & 4 - \frac{9\, 2^m}{2} - \frac{\sqrt{65}\, 2^m}{2} \\ 0 & 0 & 2^{-\lfloor m/2 \rfloor} & 1 & 0 & 2^m \\ 0 & 0 & 0 & 2^{-\lceil m/2 \rceil} & 1 & 2^{m-\lceil m/2 \rceil} \end{pmatrix}$$

where we have set $x = 2^m$. After columns 1 and 2 have been eliminated we get

$$W_1 = \begin{pmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & \mathbf{a}(1+\eta) & 0 & 0 & 2^m(1+\zeta) \\ 0 & 0 & 2^{-\lfloor m/2 \rfloor} & 1 & 0 & 2^m \\ 0 & 0 & 0 & 2^{-\lceil m/2 \rceil} & 1 & 2^{m-\lceil m/2 \rceil} \end{pmatrix}$$

where $*$ denotes an arbitrary nonzero entry and $|\eta|, |\zeta| < c\epsilon$, for some small constant c. Using property 1 above, after one more step of GQR we obtain

$$W_2 = \begin{pmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & 0 & * \\ 0 & 0 & 0 & \mathbf{a} & 0 & \mathbf{a}2^m \ominus 2^{m-\lfloor m/2 \rfloor}(1+\zeta) \\ 0 & 0 & 0 & 2^{-\lceil m/2 \rceil} & 1 & 2^{m-\lceil m/2 \rceil} \end{pmatrix}$$

where $\ominus$ denotes the machine version of the $-$ operation. The crucial fact is that $\mathbf{a}2^m \ominus 2^{m-\lfloor m/2 \rfloor}(1+\zeta) = \mathbf{a}2^m - 2^{m-\lfloor m/2 \rfloor}$, which is a consequence of the choice of m and the fact the error ζ is of the order of ϵ. But then, after one more step of GQR we get the exact values, i.e.,

$$W_3 = \begin{pmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & 0 & * \\ 0 & 0 & * & * & 0 & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & \mathbf{a} & 1 \end{pmatrix}.$$

We conclude our proof by addressing the uniformity of the matrix generation process. Using arguments similar to those in Theorem 3.1, we can prove that the generation of the matrix A_C corresponding to a circuit/input pair can be performed in $O(\log n)$ space, where n is the number of gates in the circuit C. □

5 Open problems

The matrices arising from the reduction in Section 4 and the one in [11] are singular, and all the attempts we made to extend the proofs to nonsingular matrices failed. While the deep reasons of this state of affairs could be an interesting subject per se (with possible relationships with the results in [1]), it clearly leaves us with an open problem. For general matrices, it would be interesting to know the status of HQR with column pivoting, the algorithm of choice for determining the rank of a matrix in practice.

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 [4]. 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.

[4]The constant 10 is rather arbitrary. What is important is that $2^{-m} < \epsilon$ and that $m - \lfloor m/2 \rfloor < m'$

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

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

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

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

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

[7] Golub, G. H. and C. F. Van Loan, *Matrix Computations* (The Johns Hopkins University Press, Baltimore, 1989).

[8] Greenlaw, R., H. J. Hoover, and W. L. Ruzzo, A Compendium of Problems Complete for P, Technical Report 91-05-01, Dept. of Computer Science and Engineering, University of Washington (1991).

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

[10] Leoncini, M., How Much Can We Speedup Gaussian Elimination with Pivoting? in: *Proc. 6th ACM Symp. on Parallel Algorithms and Architectures* (1994) 290–297. *Journal of Computer and System Sciences*, to appear.

[11] 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.

[12] Sigmon, K., *Matlab Primer*, The MATH WORKS Inc., 1994.

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

[14] Reif, J. H., Depth-first search is inherently sequential, *Inf. Proc. Lett.* **20** (1985) 229–234.

[15] 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.

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

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

A Gaussian elimination and Givens rotations

Let A be an $n \times n$ matrix and let a be a column vector. i.e., an $n \times 1$ matrix. The elements of A are denoted by a_{ij}, $i, j = 1, \dots, n$, while the elements of a are denoted by a_i, $i = 1, \dots, n$.
- The transpose of A is the matrix B such that $b_{ij} = a_{ji}$. B is usually denoted by A^T.
- The transpose of the column vector a is the row vector (i.e., an $1 \times n$ matrix) a^T with the same components as a.
- A matrix Q is orthogonal when $Q^T Q = I$, where I is the identity matrix.
- 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 LU decomposition of A is a pair of matrices, L and U, 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 all the leading principal minors of A be nonsingular.
- The PLU decomposition of A is a triple of matrices P, L, and U such that L and U are as above, P is a permutation matrix, and $P^T A = LU$. The PLU decomposition is always defined (even for singular matrices), but not unique.
- The QR decomposition of A is a pair of matrices Q and R, such that Q is orthogonal, R is upper triangular, and $A = QR$. The QR decomposition always exists.

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 (in which the $a_{ij}^{(k)}$s are the elements of $A^{(k)}$):

$$\begin{cases} A^{(0)} = A \\ A^{(k)} = M^{(k)} A^{(k-1)}, \quad k = 1, \dots, n-1, \\ a_{ij}^{(k)} = 0 \text{ for } i > j \text{ and } j \leq k, \\ U = A^{(n-1)}, \\ L = \left(M^{(n-1)} M^{(n-2)} \dots M^{(1)}\right)^{-1}. \end{cases}$$

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

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

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

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 $M^{(k+1)}$ to $A^{(k)}$, GEP determines the 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 *pivot row*.

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:

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

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.