Complexity of Parallel *QR* Factorization

M. COSNARD AND Y. ROBERT

Laboratory TIM 3, Université de Grenoble, France

Abstract. An optimal algorithm to perform the parallel QR decomposition of a dense matrix of size N is proposed. It is deduced that the complexity of such a decomposition is asymptotically $2N$, when an unlimited number of processors is available.

Categories and Subject Descriptors: F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems—*computations on matrices*; G.1.0 [**Numerical Analysis**]: General—*parallel algorithms*; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*linear systems*

General Terms: Algorithms, Theory, Verification

Additional Key Words and Phrases: Givens triangularization, SIMD and MIMD computers

1. *Introduction*

Solving dense systems of linear equations on parallel computers has been studied by various authors. On traditional architectures, as well as on systolic networks (see [1], [3], [5], and [6] among others for this last approach), most parallel algorithms are variations of well-known direct sequential methods. We refer the reader to the survey papers of Heller [4] and Sameh [8, 9]. It is now generally admitted that the best available method is the *QR* decomposition using square-root-free Givens transformations for reasons of stability and simplicity.

In [10] Sameh and Kuck propose a parallel scheme of computation for such a decomposition. They consider a single instruction, multiple data (SIMD) computer with $O(N^2)$ processors, where N is the size of the matrix to be decomposed. Their algorithm takes $2N - 3$ steps, each step being the time necessary to achieve a set of independent Givens transformations. It is based on a clever parallelization of the sequential algorithm and, since the total number of transformations does not exceed $N(N-1)/2$, it is as stable as the sequential one. A slight modification of their method produces an algorithm that uses the same number of processors but takes $2(N-1) - \lfloor \log N \rfloor$[1] steps. This seems to be the best known upper bound for the complexity of the *QR* decomposition of a dense matrix.

Our goal is to pursue the work of Sameh and Kuck. We first present an optimal algorithm (there is not a unique one) generating at most $\lfloor N/2 \rfloor$ rotations simultaneously and study the complexity of this algorithm. Our main result is an expression

[1] Throughout this paper, log n denotes $\log_2 n$, and $\lfloor u \rfloor$ is the greatest integer lower than or equal to u.

Authors' address: CNRS, Laboratory TIM3, Institute IMAG, Université de Grenoble, BP 68, 38402 Saint Martin d'Heres Cedex, France.

of the asymptotic complexity as N goes to infinity: Let T_N be the number of steps required by an optimal algorithm; we have

$$T_N = 2N - o(N),$$

where $o(N)/N$ tends to zero as N goes to infinity.

The optimal algorithm we propose is a greedy-type method that removes as many elements as possible for each step, and its numerical results indicate a slight improvement in speed over Sameh and Kuck's modified scheme. However, it is not very tractable on a parallel computer because of the complicated calculation of the indices. With respect to this fact, we conclude that the Sameh and Kuck algorithm, although not the fastest but asymptotically optimal (according to the result of complexity we give), is the best one.

2. *Setting of the Problem*

In the following, A is a square real matrix of size N. The problem is to obtain the orthogonal factorization of A, $A = QR$ (Q orthogonal and R upper triangular). We consider a single-instruction, multiple data computer (SIMD, [2]) with an unlimited number of available processors.

Let us recall that a Givens transformation is a plane rotation that combines two rows of A in order to annihilate one element. In the sequential algorithm the elements may be annihilated one at a time by column starting from the bottom. Hence each step of the computation uses only two rows of A. In order to parallelize this reduction, the basic idea is to annihilate more than one element at a time combining various rows, but in such a way that previously introduced zeros are not destroyed.

We assume that a Givens rotation can be realized in one step and do not allow duplication of rows. Since a rotation alters both rows, at most $\lfloor N/2 \rfloor$ rotations in one column can be performed at the same time. Various reduction schemes are possible. In order to represent such a scheme, we use the following notation (see [10]): We denote each element annihilated in the kth step by the integer k.

The following scheme has been proposed by Sameh and Kuck. The total number of steps is equal to $2N - 3$:

*														
14	*													
13	15	*												
12	14	16	*											
11	13	15	17	*										
10	12	14	16	18	*									
9	11	13	15	17	19	*								
8	10	12	14	16	18	20	*							
7	9	11	13	15	17	19	21	*						
6	8	10	12	14	16	18	20	22	*					
5	7	9	11	13	15	17	19	21	23	*				
4	6	8	10	12	14	16	18	20	22	24	*			
3	5	7	9	11	13	15	17	19	21	23	25	*		
2	4	6	8	10	12	14	16	18	20	22	24	26	*	
1	3	5	7	9	11	13	15	17	19	21	23	25	27	*

A slight modification of the preceding scheme (see below) produces an algorithm that requires only $2(N-1) - \lfloor \log N \rfloor$ steps (for $N = 15$ we obtain 25):

```
*
11  *
10  13  *
 9  12  14  *
 8  11  13  15  *
 7  10  12  14  16  *
 6   9  11  13  15  17  *
 5   8  10  12  14  16  18  *
 4   7   9  11  13  15  17  19  *
 3   6   8  10  12  14  16  18  20  *
 2   5   7   9  11  13  15  17  19  21  *
 1   4   6   8  10  12  14  16  18  20  22  *
 1   3   5   7   9  11  13  15  17  19  21  23  *
 1   2   4   6   8  10  12  14  16  18  20  22  24  *
 1   2   3   5   7   9  11  13  15  17  19  21  23  25  *
```

However, this is not the best possible scheme. For $N = 15$ we can obtain $T_N = 24$:

```
*
4  *
3  6  *
3  5  8  *
2  5  7  10  *
2  4  7   9  12  *
2  4  6   9  11  14  *
2  4  6   8  10  13  16  *
1  3  5   8  10  12  15  18  *
1  3  5   7   9  11  14  17  19  *
1  3  5   7   9  11  13  16  18  20  *
1  3  4   6   8  10  12  15  17  19  21  *
1  2  4   6   8  10  12  14  16  18  20  22  *
1  2  4   5   7   9  11  13  15  17  19  21  23  *
1  2  3   5   6   8  10  12  14  16  18  20  22  24  *
```

3. *Reduction to a Particular Class of Parallel Algorithms*

In this section we show that it is not worthwhile to annihilate an element that will be destroyed later on. Moreover, we prove that the elements can be annihilated from left to right and from bottom to top.

We begin by providing some notation and definitions: (i, j, k), $i \neq j$, denotes the rotation in plane (i, j) that annihilates the element in position (i, k). $\mathrm{Id}(i, j)$ and $\mathrm{Perm}(i, j)$ denote the rotations in plane (i, j) that correspond, respectively, to the identity and permutation of rows i and j. Since we deal with general matrices, it is assumed that rotation (i, j, k) cannot zero more than one element in row i and cannot zero any element in row j (this assumption is implicit in [10]). A parallel algorithm M of length T is represented by $M = (M(1), \ldots, M(T))$, where, for $t \leq T$, $M(t)$ is a set of $r(t)$ independent rotations. We set $A(0) = A$ and

for $1 \leq t \leq T$, $A(t)$ is the matrix obtained by applying in parallel the rotations in $M(t)$ to $A(t-1)$. The total number of rotations of the algorithm M is $R = r(1) + \cdots + r(T)$. For short we use the notation (M, T, R). Finally, M is said to be a T-algorithm if $A(T)$ is upper triangular.

PROPOSITION 1. *Let (M, T, R) be a T-algorithm. There exists a T-algorithm $(M\#, T, N(N-1)/2)$ that annihilates the elements in the following order: the element in position (i, j) is not annihilated before the element in position (k, m) if and only if $(k = i$ and $m < j)$ or $(m = j$ and $k > i)$.*

PROOF. This proof is divided into four parts:

Step 1. First, we construct (M', T, R), which annihilates the elements in any row from left to right.

Let us call $p(1, t), \ldots, p(N, t)$ the number of annihilated elements of $A(t)$ in rows $1, \ldots, N$.

We construct M' and the sequence $A'(t)$ (recall that $A'(0) = A'$ and for $1 \leq t \leq T$, $A'(t)$ is obtained by applying in parallel the rotations in $M'(t)$ to $A'(t-1)$) by induction on t so that $A'(t)$ has $p'(1, t), \ldots, p'(N, t)$ zeros with the following properties:

—$p'(i, t) \geq p(i, t)$; $i = 1, \ldots, N$,
—the first $p'(i, t)$ elements of row i in $A'(t)$ are zeros.

At time 1, if (i, j, k) belongs to $M(1)$, let $(i, j, 1)$ belong to $M'(1)$. The preceding induction hypothesis is clearly satisfied for $A'(1)$. We assume that $M'(t)$ has been constructed so that $A'(t)$ has the required properties. Because the rotations in $M(t+1)$ act on disjoint pairs of rows, we can consider each pair separately. Let (i, j, k) belong to $M(t+1)$. We must construct an operation in $M'(t+1)$ that preserves the induction properties.

There are two cases:

—If a new zero is required in row i of $A'(t)$, we use a rotation to introduce it or a permutation to bring it up from row j.
—If row i of $A'(t)$ already has enough zeros, we do nothing (identity).

What is actually done is determined by an analysis of cases, to which we now proceed.

Assume that the induction hypothesis is valid at time t. If (i, j, k) belongs to $M(t+1)$, discuss the following cases:

(a) The positions of the zeros in rows i and j of $A(t)$ are the same. Thus $p(i, t) = p(j, t)$.

(a1) $p'(i, t) = p'(j, t) = p(i, t)$.

Replace (i, j, k) in $M(t+1)$ by $(i, j, p'(i, t) + 1)$ in $M'(t+1)$.

(a2) $p'(i, t) > p(i, t)$.

Replace (i, j, k) in $M(t+1)$ by $\mathrm{Id}(i, j)$ in $M'(t+1)$.

(a3) $p'(i, t) = p(i, t)$ and $p'(j, t) > p(j, t)$.

Replace (i, j, k) in $M(t+1)$ by $\mathrm{Perm}(i, j)$ in $M'(t+1)$.

(b) The positions of the zeros in rows i and j of $A(t)$ are not the same.
Thus $p(j, t+1) \leq p(j, t)$.

(b1) $p(i, t+1) \leq p(i, t)$

Replace (i, j, k) in $M(t+1)$ by $\mathrm{Id}(i, j)$ in $M'(t+1)$.

(b2) $p(i, t+1) = p(i, t) + 1$.

Thus the set of the indices of the zeros in row i of $A(t)$ is strictly included into the one of row j, which implies

$$p(j, t+1) = p(i, t) < p(j, t).$$

—$p'(i, t) \geq p(i, t) + 1$.
Replace (i, j, k) in $M(t+1)$ by $\mathrm{Id}(i, j)$ in $M'(t+1)$.
—$p'(i, t) = p(i, t)$.
Replace (i, j, k) in $M(t+1)$ by $\mathrm{Perm}(i, j)$ in $M'(t+1)$.

Clearly, the induction hypothesis is satisfied at time $t+1$. Since M is a T-algorithm, M' is a T-algorithm too.

Step 2. A rotation that strictly increases the number of zeros is called efficient. Since no zero is destroyed in M', the number of efficient rotations in M' is $N(N-1)/2$. We now show the existence of an algorithm $(M'', T, N(N-1)/2)$ that annihilates the elements from left to right and such that $P''A''(T)$ is upper triangular for some permutation matrix P''.

At each time step t we introduce a permutation $s(t)$ of the rows of the matrix:

—$s(0)$ is the identity.
—Assume $s(t)$ and $A''(t)$ are defined:
—If (i, j, k) is an efficient rotation of $M'(t+1)$, then $(s(t)(i), s(t)(j), k)$ is a rotation of $M''(t+1)$.
—If $\mathrm{Id}(i, j)$ is an element of $M'(t+1)$, then M'' performs no rotation in plane (i, j) at time $t+1$.
—If $\mathrm{Perm}(i, j)$ is an element of $M'(t+1)$, then set

$$s(t+1) = s(t) \circ \mathrm{trans}(i, j),$$

where $\mathrm{trans}(i, j)$ is the permutation that exchanges i and j.

Let P'' be the permutation matrix associated with $s(T)$. Clearly, $P''A''(T)$ is lower triangular.

Step 3. We derive from $(M'', T, N(N-1)/2)$ an algorithm $(M''', T, N(N-1)/2)$ by the following equivalence:

$$(i, j, k) \in M''(t) \Leftrightarrow (s(T)(i), s(T)(j), k) \in M'''(t), \qquad 1 \leq t \leq T.$$

Clearly, M''' is an algorithm that reduces A to an upper triangular matrix by using $N(N-1)/2$ rotations and annihilating the elements from left to right.

Step 4. We construct an algorithm $M\#$ that has the same properties as M''' and annihilates the elements from bottom to top. Let $E'''(i, t)$ be the set of rows of $A'''(t)$ that have exactly i zeros, for $1 \leq i \leq N$ and $1 \leq t \leq T$. We show by induction that $E\#(i, t)$ and $E'''(i, t)$ have the same number of elements for all i and t:

—At time 1, M''' performs $k(0, 1)$ rotations using $2k(0, 1)$ rows of the matrix to annihilate $k(0, 1)$ elements in column 1. Then $M\#$ will annihilate the lowest

$k(0, 1)$ elements of column 1, using for instance the last $2k(0, 1)$ rows of the matrix

—If at time $t + 1$ M''' performs $k(i, t)$ rotations using $2k(i, t)$ rows of $E'''(i, t)$ to annihilate $k(i, t)$ elements in column $i + 1$, then $M\#$ will annihilate the $k(i, t)$ elements of column $i + 1$ located in the last $k(i, t)$ rows of $E\#(i, t)$, using, for instance, the last $2k(i, t)$ rows of $E\#(i, t)$. □

As a consequence, $M\#$ is an algorithm that satisfies the conditions of Proposition 1.

4. *An Optimal Algorithm*

In what follows we consider algorithms that use $N(N-1)/2$ Givens rotations and that annihilate an element a_{ik} only if $a_{ih} = 0$ for all $h < k$. Moreover, the elements in a given column will be annihilated from bottom to top.

Let us now introduce some notation and definitions. A column (of an annihilation scheme) of length n is a sequence of n integers:

$$a = a_1^{n_1} \cdots a_q^{n_q}$$

where power means concatenation with the following restrictions:

$$a_1 \geq 0; \quad a_{i+1} > a_i, \quad 1 \leq i \leq q-1;$$
$$n_i > 0, \quad i \leq i \leq q; \quad n_1 + \cdots + n_q = n.$$

We define on the set of columns of length n the classical partial ordering of $\mathbb{R}^n$:

$$x \leq y \Leftrightarrow (x_i \leq y_i,\ 1 \leq i \leq n).$$

The s-truncate ($1 \leq s \leq n$) of a is a column of length s composed of the s first elements of a and is denoted a^s.

$b = b_1^{m_1} \cdots b_p^{m_p}$ is called an iterate of a, or $b = \text{iter}(a)$, if

(i) b is a column of length $n - 1$.
(ii) $a_1 + 1 \leq b_1$.
(iii) $-a_1 + 1 \leq b_1 \leq a_2 \Rightarrow m_1 \leq \lfloor n_1/2 \rfloor$;
$-a_{k-1} + 1 \leq b_h \leq a_k$
$\Rightarrow m_h \leq \lfloor (n_1 + \cdots + n_{k-1} - m_1 - \cdots - m_{h-1})/2 \rfloor$
$2 \leq k \leq q$ and $1 \leq h \leq p$ $(m_0 = 0)$;
$-a_{p+1} \leq b_h \Rightarrow m_h \leq \lfloor (n - m_1 - \cdots - m_{h-1})/2 \rfloor$.

Consider now an algorithm that reduces A. We associate with it the triangular array $U = (u_{i,j})$, where $u_{i,j}$ is the step at which $a_{n+1-i,j}$ is annihilated: $u_{1,1} = 1$ means that $a_{n,1}$ is annihilated at step 1 (examples are given in Section 2).

We have the following relations:

$$u_{1,1} \geq 1; \quad u_{i-1,j} \leq u_{i,j}; \quad u_{i,j} < u_{i,j+1}.$$

Moreover the preceding considerations imply that the number of elements in column $j + 1$ that can be annihilated at step $t + 1$ is less than or equal to the half of the difference between the number of elements in column j and in column $j + 1$ annihilated at step t. We derive from this the following definition:

Definition 1. A triangular array U is a *scheme of computation* if

$$U_j = u_{1,j} \cdots u_{N-j,j}$$

is such that $U_j = \text{iter}(U_{j-1})$, $1 \leq j \leq N$, with $U_0 = O^N$.

Let us now introduce a special type of iterated column for a "greedy" computation scheme.

Definition and Proposition 2. *Let a be an iterated column of length n:*

$$a = a_1^{n_1} \cdots a_q^{n_q}.$$

The sequence $b = b_1^{m_1} \cdots b_p^{m_p}$ *defined as follows is an iterated column of a.*

We give the value of b_i, m_i and show that $b^{m_1+\cdots+m_i}$ is a column of length $m_1 + \cdots + m_i$, which is an iterate of the column $a^{m_1+\cdots+m_i+1}$.

Construction of b_1 *and* m_1

—If $n_1 = 1$, then $b_1 = a_2 + 1$ and $m_1 = \lfloor (n_1 + n_2)/2 \rfloor$.
—If $n_1 > 1$, then $b_1 = a_1 + 1$ and $m_1 = \lfloor n_1/2 \rfloor$.

Clearly, b^{m_1} is iterated from a^{m_1+1}.

Construction of b_i *and* m_i

We assume that $b_1, \ldots, b_{i-1}$ and $m_1, \ldots, m_{i-1}$ are known and, moreover, that

$$b^{m_1+\cdots+m_{i-1}}$$

is an iterated column of

$$a^{m_1+\cdots+m_{i-1}+1}.$$

We also assume that $m_1 + \cdots + m_{i-1} < N - 1$.

(i) If there exists k such that $a_{k-1} + 1 \leq b_{i-1} \leq a_k$, then

$$r_{i-1} = (n_1 + \cdots + n_{k-1}) - (m_1 + \cdots + m_{i-1}) \geq 1.$$

(i1) If $b_{i-1} < a_k$ and $r_{i-1} > 1$, then $b_i = b_{i-1} + 1$, and $m_i = \lfloor r_{i-1}/2 \rfloor$;
(i2) else $b_i = a_k + 1$, $m_i = \lfloor (n_k + r_{i-1})/2 \rfloor$.

(ii) If $b_{i-1} > a_q$, then $b_i = b_{i-1} + 1$,
and $m_i = \lfloor n - (m_1 + \cdots + m_{i-1})/2 \rfloor$.

It is clear that in either (i) or (ii), $b^{m_1+\cdots+m_i}$ is an iterated column of $a^{m_1+\cdots+m_i+1}$.
We use the notation $b = \text{optiter}(a)$ in order to denote the preceding iterated column.

PROPOSITION 3

(*i*) *Let* a_n *be a column of length n and* $c_{n-1} = iter(a_n)$ *an iterated column of* a_n. *Then*

$$b_{n-1} = optiter(a_n) \leq iter(a_n) = c_{n-1}.$$

(*ii*) *Let* a_n *and* c_n *be two columns of length n such that* $a_n \leq c_n$. *Then*

$$optiter(a_n) \leq optiter(c_n).$$

PROOF

(i) From the preceding construction and the definition of the iterated column, we have that

$$b_{n-1}^{m_1+\cdots+m_i} \leq c_{n-1}^{m_1+\cdots+m_i}, \qquad 1 \leq i \leq p.$$

Hence $b_{n-1} \leq c_{n-1}$.
(ii) Follows from the same argument. □

In order to illustrate Propositions 2 and 3, consider the following example:

$$a = 1^3\ 3^4\ 4^2 \leq c = 1^3\ 3^2\ 4^4,$$
$$\text{optiter}(a) = 2\ 3\ 4^2\ 5^2\ 6\ 7 \leq \text{optiter}(c) = 2\ 3\ 4\ 5^3\ 6\ 7,$$
$$\text{iter}(a) = 2\ 3\ 4\ 6\ 7^2\ 8\ 9 \geq \text{optiter}(a).$$

Definition 3. We call V the scheme of computation associated with optiter:

$$V_0 = O^N,\ V_i = \text{optiter}(V_{i-1}), \qquad 1 \leq i \leq N-1.$$

We say that a scheme of computation U is better than another one U' if $U_{N-1} \leq U'_{N-1}$.

Clearly U_{N-1} is the number of steps in order to achieve the reduction using the scheme of computation U. Hence, U is better than U' if it uses fewer steps than U'.

THEOREM 1. *Let N be given. V is an optimal scheme of computation*

$$\forall U,\ V_{N-1} \leq U_{N-1}.$$

PROOF. Proposition 1 implies that $U_0 = O^N$; $U_i = \text{iter}(U_{i-1})$, $1 \leq i \leq N-1$. Clearly, $V_0 \leq U_0$. Assume that $V_{i-1} \leq U_{i-1}$. Then, applying Proposition 3,

$$V_i = \text{optiter}(V_{i-1}) \leq \text{optiter}(U_{i-1}) \leq \text{iter}(U_{i-1}) = U_i.$$

Hence $V_{N-1} \leq U_{N-1}$.

Thus the optimal number of steps to achieve a QR reduction using plane rotations is V_{N-1}, and V is optimal. Note that the optimal scheme is not unique:

$$V = \begin{array}{cccccccc} * & & & & & & & \\ 3 & * & & & & & & \\ 2 & 5 & * & & & & & \\ 2 & 4 & 7 & * & & & & \\ 1 & 3 & 6 & 8 & * & & & \\ 1 & 3 & 5 & 7 & 9 & * & & \\ 1 & 2 & 4 & 6 & 8 & 10 & * & \end{array} \qquad U = \begin{array}{cccccccc} * & & & & & & & \\ 5 & * & & & & & & \\ 4 & 6 & * & & & & & \\ 3 & 5 & 7 & * & & & & \\ 2 & 4 & 6 & 8 & * & & & \\ 1 & 3 & 5 & 7 & 9 & * & & \\ 1 & 2 & 4 & 6 & 8 & 10 & * & \end{array} \qquad N = 7. \quad \square$$

5. *Bounds for the Complexity*

In this section we concentrate on the evaluation of the complexity of V, that is, $T_N = V_{N-1}$. We do not succeed in obtaining a simple formula for T_N. However, we give the asymptotic complexity of the optimal scheme. Let us begin by some remarks and experiments.

LEMMA 1

(*i*) $T_2 = 1$ *and* $N \geq 2$, $T_N + 1 \leq T_{N+1} \leq T_N + 2$.
(*ii*) $N - 1 \leq T_N \leq 2N - 3$.

PROOF

(i) Let V be the optimal scheme for $N+1$. Let U be the scheme obtained by deleting the last row and column of V. U is a scheme of computation for N and $T_N \leq U_{N-1}$. From the construction of U we obtain that

$$U_{N-1} \leq V_N - 1 = T_{N+1} - 1.$$

Now let V be the optimal scheme for N, and let U be the scheme obtained by adding a diagonal to V:

$$u_{N+1-j,j} = v_{N-j,j} + 1, \qquad u_{1,N} = v_{1,N-1} + 2.$$

U is a scheme of computation for $N+1$ and $T_{N+1} \leq U_N = T_N + 2$, which concludes the proof.

(ii) Follows directly from (i). □

Only part (ii) of the lemma is of importance in what follows; moreover, it can be obtained directly (the lower bound follows from the need to perform $N(N-1)/2$ rotations with at most $\lfloor N/2 \rfloor$ rotations per step, and the upper bound follows from the scheme by Sameh and Kuck). However, part (i) points out the main difficulty in the study of the time complexity of the greedy algorithm: determining when $T_{N+1} - T_N$ is equal to 1 or 2.

The table below shows T_N for some values of N. Clearly, $N-1$ is not a realistic lower bound, but $2N-3$ is not so bad: for instance, with $N = 4096$, $T_N = 8129$, whereas $2n - 3 = 8189$.

N	3	4	5	6	7	8	9	10	14	15	16	17	18
T_N	3	4	6	8	10	11	13	15	23	24	26	28	30

N	20	32	40	50	64	128	256	512	1024	2048	4096
T_N	34	56	72	91	118	243	495	1000	2015	4051	8129

We now discuss the asymptotic complexity of the parallel QR factorization. We want to prove that

$$T_N = 2N - o(N).$$

Consider the optimal scheme of computation V and let X_j^r be the number of times that $j + r - 1$ appears in column r. From the definition of V, we deduce that

$$X_1^0 = N; \qquad X_j^0 = 0, \qquad j > 1;$$
$$X_j^r = \left\lfloor \frac{\sum_{i=1}^{j} X_i^{r-1} - \sum_{i=1}^{j-1} X_i^r}{2} \right\rfloor, \qquad j, r > 0.$$

It is worth noting that some X_j^r can be equal to zero. The preceding formulas define a set of recurrence relations acting on N. In order to evaluate the X_j^r that this scheme defines, we introduce the associated real scheme:

$$Y_1^0 = N; \qquad Y_j^0 = 0, \qquad j > 1;$$
$$Y_j^r = \frac{\sum_{i=1}^{j} Y_i^{r-1} - \sum_{i=1}^{j-1} Y_i^r}{2}, \qquad j, r > 0.$$

Let us finally define partial sums of X's and Y's as

$$S_k^r = \sum_{j=1}^{k} X_j^r,$$
$$T_k^r = \sum_{j=1}^{k} Y_j^r.$$

S_k^r is the number of elements in column r that are less than or equal to $r + k - 1$.

Call $K(N)$ the first nonzero index in column $N-1$, that is

$$X_{K(N)}^{N-1} = 1; \qquad X_j^{N-1} = 0, \qquad j < K(N).$$

This is equivalent to $S_{K(N)-1}^{N-1} = 0$ and $S_{K(N)}^{N-1} = 1$.

The total number of steps required to achieve the QR reduction is then $N - 2 + K(N)$. We want to show that this total number of steps is asymptotically equivalent to $2N$. We thus have to prove that

$$\lim_{N\to+\infty} \frac{K(N)}{N} = 1.$$

From the preceding section we have $K(N) \leq N$ so that $\limsup_{N\to+\infty} K(N)/N \leq 1$. In order to prove that $\liminf_{N\to+\infty} K(N)/N \geq 1$, we first show that $S_k^r \leq T_k^r$, then that $\lim_{N\to+\infty} T_{(1-e)N}^{N-1} = 0$, $e > 0$, and finally that $\lim_{N\to+\infty} S_{(1-e)N}^{N-1} = 0$.

LEMMA 2. *Given k and r: $S_k^r \leq T_k^r$.*

PROOF. By definition,

$$\begin{aligned} S_0^r &= T_0^r = 0 \qquad \text{for all} \quad r \geq 0, \\ S_k^o &= T_k^o = N \qquad \text{for all} \quad k \geq 1. \end{aligned}$$

Therefore, by induction

$$\begin{aligned} S_k^r &= S_{k-1}^r + X_k^r \\ &= S_{k-1}^r + \lfloor \tfrac{1}{2}(S_k^{r-1} - S_{k-1}^r) \rfloor \\ &\leq S_{k-1}^r + \tfrac{1}{2}(S_k^{r-1} - S_{k-1}^r) \\ &= \tfrac{1}{2}(S_{k-1}^r + S_k^{r-1}) \\ &\leq \tfrac{1}{2}(T_{k-1}^r + T_k^{r-1}) \\ &= T_{k-1}^r + \tfrac{1}{2}(T_k^{r-1} - T_{k-1}^r) \\ &= T_{k-1}^r + Y_k^r \\ &= T_{k-1}^r. \end{aligned}$$

□

It should be noted that even though the sums satisfy the above inequality, this does not hold for individual quantities Y_j^r and X_j^r.

We deduce that $K'(N) \leq K(N)$, where for the real scheme $K'(N)$ is such that $T_{K'(N)-1}^{N-1} < 1 \leq T_{K'(N)}^{N-1}$.

LEMMA 3

$$\forall j, r,\ Y_j^r = \binom{j + r - 2}{j - 1} \cdot 2^{-j-r+1} \cdot N.$$

PROOF. First of all we have the boundary conditions

$$Y_j^1 = N \cdot 2^{-j} \quad \text{and} \quad Y_1^r = N \cdot 2^{-r}.$$

From the definition of Y_j^r, we deduce that for $j, r \geq 2$, we have

$$Y_j^r = Y_{j-1}^r + \tfrac{1}{2}(Y_j^{r-1} - Y_{j-1}^r) = \tfrac{1}{2}(Y_j^{r-1} + Y_{j-1}^r),$$

which leads easily to the result (this formula is similar to the one for combinations with repetition [7]). □

LEMMA 4. *There exists $E > 0$ such that for any positive number $e < E$*

$$\lim_{N\to+\infty} S_{\lfloor (1-e)N \rfloor}^{N-1} = 0.$$

PROOF. We set $\lfloor(1-e)N\rfloor = (1-a)N$; thus $a - 1/N \le e \le a$. Since from Lemma 3

$$Y_j^{N-1} \le Y_{j+1}^{N-1}, \qquad 1 \le j \le N-2,$$

we have

$$S_{(1-a)N}^{N-1} \le T_{(1-a)N}^{N-1} \le (1-a)N \cdot Y_{(1-a)N}^{N-1}.$$

Hence, we have to study

$$L = \lim_{N\to+\infty} f(N),$$

where

$$f(N) = (1-a)N \cdot \binom{(2-a)N-3}{(1-a)N-1} \cdot 2^{-(2-a)N+2} \cdot N.$$

We write first $f(N)$ as

$$f(N) = \binom{(2-a)N}{(1-a)N} \cdot 2^{-(2-a)N} \cdot g(N),$$

where $g(N)$ is a rational fraction of N of degree 2.[2]

Now, using Stirling's formula,

$$L = \lim_{N\to+\infty} h(N^{1/2}) \cdot [(2-a)^{2-a}(1-a)^{-(1-a)}2^{-(2-a)}]^N,$$

where h is a rational fraction of $N^{1/2}$ of degree 3. Let

$$A = (2-a)^{2-a}(1-a)^{-(1-a)}2^{-(2-a)}$$
$$= \left(1-\frac{a}{2}\right)^{(2-a)}(1-a)^{-(1-a)}.$$

We evaluate $\ln A$

$$\ln A = (2-a)\cdot \ln\left(1-\frac{a}{2}\right) - (1-a)\cdot \ln(1-a)$$
$$= (2-a)\cdot\left[-\frac{a}{2} - \frac{a^2}{8} + O(a^3)\right] - (1-a)\cdot\left[-a - \frac{a^2}{2} - O(a^3)\right]$$
$$= -\frac{a^2}{4} + O(a^3).$$

Therefore there exists $E' > 0$ such that $A < 1$ for any $a \le E'$. Since $a \le e + 1/N$, we may choose $E = E'/2$ to ensure that

$$\forall e < E, \qquad \lim_{N\to+\infty} S_{\lfloor(1-e)N\rfloor}^{N-1} = 0.$$

Now let $e < E$. From the above lemma, there exists $N(e)$ such that

$$K(N) > \lfloor(1-e)N\rfloor \qquad \text{for} \quad N \ge N(e).$$

Therefore

$$\liminf_{N\to+\infty} \frac{K(N)}{N} \ge 1.$$

[2] The degree is the difference between the degrees of the numerator and the denominator.

We have proved the following theorem:

THEOREM 2. *$\lim_{N\to+\infty} T_N/2N = 1$, and hence $T_N = 2N - o(N)$.*

6. *Concluding Remarks*

The *QR*-factorization is the most currently used algorithm to solve linear problems in various fields of scientific computation. The $O(N^2)$ transformations required to achieve the factorization motivate its parallelization. In this paper we have shown that Sameh and Kuck's scheme is the best possible, from both points of view:

—Practically, No significant improvement of the time of computation can be obtained without a prohibitive amount of complexity.
—Asymptotically, It is optimal.

Note Added in Proof. The article by J. J. Modi and M. R. B. Clarke, "An alternative Givens ordering," *Numerische Mathematik 43* (1984), 83–90, has been pointed out to us by the referees during the second revision of our paper. The authors introduce the greedy algorithm to triangularize a rectangular matrix A of size $M \times N$, with $M \gg N$. The proofs we give in Section 3 and 4 may be straightforwardly extended to rectangular matrices. Hence the greedy method is optimal for any rectangular matrix, not only in the class of "Givens sequences" (i.e., "any sequence of Givens rotations in which zeros once created are preserved," as was conjectured by Modi and Clarke), but more generally for the class of all possible parallel algorithms based on Givens rotations. Furthermore, when $M \gg N$, Modi and Clarke show that the number of parallel steps is asymptotically $\log M + (N-1)\log\log M$: Their time analysis for the greedy algorithm can now be viewed as a result of complexity.

ACKNOWLEDGMENTS. We are greatly indebted to the referees for pointing out that the proof of Proposition 1 was not complete, for their careful reading of the paper, and for their helpful suggestions, which greatly improved the quality of the original manuscript.

REFERENCES

1. AHMED, H. M., DELOSME, J. M., AND MORF, M. Highly concurrent computing structures for matrix arithmetic and signal processing. *Computer Magazine* (Jan. 1982), pp. 65–82.
2. FLYNN, M. J. Very high-speed computing systems. *Proc. IEEE 54* (1966), 1901–1909.
3. GENTLEMAN, W. M., AND KUNG, H. T. Matrix triangularization by systolic arrays. *Proc. SPIE 298*, Real time signal processing 4 (Aug. 1981), 19–26.
4. HELLER, D. A survey of parallel algorithms in numerical linear algebra. *SIAM Rev. 20* (1978), 740–777.
5. HELLER, D., AND IPSEN, I. Systolic networks for orthogonal equivalence transformations and their applications. In *Proceedings of the 1982 Conference on Advanced Research in VLSI*. MIT, Cambridge, Mass., 1982, pp. 113–122.
6. KUNG, H. T., AND LEISERSON, C. E. Systolic arrays for (VLSI). In *Sparse Matrix Proceedings 1978*, I. Duff and G. W. Stewart, Eds. SIAM, Philadelphia, Pa., 1979, pp. 256–282.
7. RIORDAN, J. *An Introduction to Combinatorial Analysis*. Wiley, New York, 1958.
8. SAMEH, A. Numerical parallel algorithms—A survey. In *High-Speed Computer and Algorithm Organization*, D. Kuck, D. Lawrie, and A. Sameh, Eds. Academic Press, Orlando, Fla., 1977, 207–228.
9. SAMEH, A. An overview of parallel algorithms. *Bull. EDF C1* (1983) 129–134.
10. SAMEH, A., AND KUCK, D. On stable parallel linear system solvers. *J. ACM 25*, 1 (1978), 81–91.

RECEIVED JULY 1983; REVISED DECEMBER 1985; ACCEPTED JANUARY 1986