

# ornl

**OAK RIDGE  
NATIONAL  
LABORATORY**

**MARTIN MARIETTA**

MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES



3 4456 0278988 2

ORNL/TM-10581

## QR Factorization of a Dense Matrix on a Shared-Memory Multiprocessor

Eleanor Chu  
Alan George

OAK RIDGE NATIONAL LABORATORY  
CENTRAL RESEARCH LIBRARY  
CIRCULATION SECTION  
400N TROON ST

**LIBRARY LOAN COPY**

DO NOT TRANSFER TO ANOTHER PERSON

If you wish someone else to see this  
report, send in name with report and  
the library will arrange a loan.

OPERATED BY  
MARTIN MARIETTA ENERGY SYSTEMS, INC.  
FOR THE UNITED STATES  
DEPARTMENT OF ENERGY

Printed in the United States of America. Available from  
National Technical Information Service  
U.S. Department of Commerce  
5285 Port Royal Road, Springfield, Virginia 22161  
NTIS price codes—Printed Copy: A03; Microfiche A01

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ORNL/TM-10581

Engineering Physics and Mathematics Division

Mathematical Sciences Section

QR Factorization of a Dense Matrix  
on a Shared-Memory Multiprocessor

Eleanor Chu  
Department of Computer Science  
University of Waterloo  
Waterloo, Ontario, Canada N2L 3G1

Alan George†  
Mathematical Sciences Section  
Oak Ridge National Laboratory  
Oak Ridge, Tennessee 37831

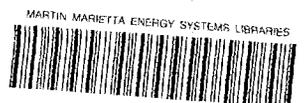
---

†Also a member of the Departments of Computer Science and  
Mathematics, University of Tennessee, Knoxville, Tennessee 37996.

Date Published - October 1987

The work was supported by the  
Applied Mathematical Sciences subprogram  
of the Office of Energy Research,  
U.S. Department of Energy

Prepared by the  
Oak Ridge National Laboratory  
Oak Ridge, Tennessee 37831  
operated by  
Martin Marietta Energy Systems, Inc.  
for the  
**U.S. DEPARTMENT OF ENERGY**  
under Contract No. DE-AC05-84OR21400



3 4456 0278988 2



## CONTENTS

Abstract .....	v
1. Introduction .....	1
2. The Algorithm .....	4
2.1 The Independent Annihilation Phase .....	5
2.2 The Cooperative Annihilation Phase .....	5
3. Implementation Issues .....	11
4. Analysis of Synchronization Cost .....	14
5. Analysis of Work Loan Distribution .....	15
6. Performance Analysis .....	17
7. Numerical Experiments .....	20
References .....	24



### Abstract

A new algorithm for computing an orthogonal decomposition of a rectangular  $m \times n$  matrix  $A$  on a shared-memory parallel computer is described. The algorithm uses Givens rotations, and has the feature that its synchronization cost is low. In particular, for a multiprocessor having  $p$  processors, an analysis of the algorithm shows that this cost is  $O(n^2/p)$  if  $m/p \geq n$ , and  $O(mn/p^2)$  if  $m/p < n$ . Note that in the latter case, the synchronization cost is smaller than  $O(n^2/p)$ . Therefore, the synchronization cost of the algorithm proposed in this article is bounded by  $O(n^2/p)$  when  $m \geq n$ . This is important for machines where synchronization cost is high, and when  $m \gg n$ . Analysis and experiments show that the algorithm is effective in balancing the load and producing high efficiency (speed-up).



# 1 Introduction

In this article we present an algorithm for reducing an  $m \times n$  ( $m \geq n$ ) matrix to upper triangular form on a shared-memory multiprocessor having  $p$  identical processors. The use of Givens transformations has been widely studied in the literature for parallel implementation on systolic arrays [1,2,8,10], shared-memory multiprocessors [5,7,9,11,13], and local-memory multiprocessors [3,12]. The parallel algorithm we describe is also based on Givens rotations. Since our target machine is a shared-memory multiprocessor, a brief review of the schemes in [5,7,9,11,13] will provide useful background information.

The mathematical computation we consider is usually formulated as

$$QA = \begin{pmatrix} R \\ 0 \end{pmatrix},$$

where  $A$  is an  $m \times n$  ( $m \geq n$ ) matrix with full column rank,  $Q$  is an  $m \times m$  orthogonal matrix and  $R$  is an upper triangular matrix of order  $n$ . The matrix  $Q$  is formed as the product of Givens rotations such that the elements of  $A$  below the main diagonal are annihilated one at a time. There is much freedom in the order of applying the Givens rotations. For a particular Givens ordering, the *theoretically* minimum number of parallel steps is obtained by assuming that all *independent* (or *disjoint*) rotations can be computed simultaneously in one step. The parallel algorithms presented in [5,7,9,11,13] are all based on "Givens sequences", that is, sequences of Givens rotations in which zeros once created are preserved. The question of whether temporarily annihilating elements and introducing zeros that will be destroyed later can lead to any additional parallelism is discussed in [5]. The odd-even ordering used in the rotation method proposed by Luk in [10] has this property of creating redundant zeros.

Figures 1-3 illustrate the different Givens sequences used in [5,7,9,11,13] for an  $8 \times 8$  matrix. For each sequence illustrated, the disjoint rotations are identified by the same step number. Since the elimination order is from left to right for all three Givens sequences, the first  $k$  ( $k < 8$ ) columns of each matrix illustrate the annihilation ordering for an  $8 \times k$  rectangular matrix.

Note that for the Givens sequences illustrated in Figures 1-3,  $\lfloor m/2 \rfloor$  processors are required to factor an  $m \times n$  matrix using the minimum parallel steps. The availability of  $\lfloor m/2 \rfloor$  processors is assumed in the algorithm analyses in [5,11,13], and the optimality of the greedy Givens sequence is established in [5] under the same condition. The parallel algorithm proposed

$$\begin{pmatrix} \times & \times \\ 7 & \times \\ 6 & 8 & \times & \times & \times & \times & \times & \times \\ 5 & 7 & 9 & \times & \times & \times & \times & \times \\ 4 & 6 & 8 & 10 & \times & \times & \times & \times \\ 3 & 5 & 7 & 9 & 11 & \times & \times & \times \\ 2 & 4 & 6 & 8 & 10 & 12 & \times & \times \\ 1 & 3 & 5 & 7 & 9 & 11 & 13 & \times \end{pmatrix}$$

Figure 1: Standard Givens Sequence [13]

$$\begin{pmatrix} \times & \times \\ 1 & \times \\ 2 & 3 & \times & \times & \times & \times & \times & \times \\ 3 & 4 & 5 & \times & \times & \times & \times & \times \\ 4 & 5 & 6 & 7 & \times & \times & \times & \times \\ 5 & 6 & 7 & 8 & 9 & \times & \times & \times \\ 6 & 7 & 8 & 9 & 10 & 11 & \times & \times \\ 7 & 8 & 9 & 10 & 11 & 12 & 13 & \times \end{pmatrix}$$

Figure 2: Dongarra et al. [7] and Lord et al. [9]

$$\begin{pmatrix} \times & \times \\ 3 & \times \\ 2 & 5 & \times & \times & \times & \times & \times & \times \\ 2 & 4 & 7 & \times & \times & \times & \times & \times \\ 1 & 3 & 6 & 8 & \times & \times & \times & \times \\ 1 & 3 & 5 & 7 & 9 & \times & \times & \times \\ 1 & 2 & 4 & 6 & 8 & 10 & \times & \times \\ 1 & 2 & 3 & 5 & 7 & 9 & 11 & \times \end{pmatrix}$$

Figure 3: Greedy Givens sequence [5,11]

by Sameh and Kuck [13] is based on the standard Givens sequence (Figure 1), for which  $(m + n - 2)$  steps are required to factor an  $m \times n$  matrix using up to  $\lfloor m/2 \rfloor$  processors. A parallel algorithm based on the greedy Givens sequence (Figure 3) was proposed independently by Modi et al. [11] and Cosnard et al. [6]. While there is no exact analysis, by assuming  $m$  goes to infinity with  $n$  fixed, Modi and Clarke's approximate analysis in [11] gives the asymptotic complexity of

$$\log_2 m + (n - 1) \log_2 \log_2 m$$

parallel steps. Although the reduction in the number of steps from  $(m+n-2)$  to  $(\log_2 m + (n - 1) \log_2 \log_2 m)$  is impressive, the efficiency of this algorithm is not satisfactory since  $\lfloor m/2 \rfloor$  processors are used. For  $n$  fixed,  $m \rightarrow \infty$ , Cosnard et al. [5] derive an efficiency of

$$2n/\log_2 m + o(1/\log_2 m).$$

They assume that all rotations in the serial Givens scheme or in the parallel scheme take the same amount of time.

Cosnard et al. also derive the asymptotic complexity of  $2n$  parallel steps for the case  $m/n^2$  tending to zero as  $m$  and  $n$  go to infinity. In view of the relatively small number of processors on a shared-memory multiprocessor, it is unlikely that the assumption of  $\lfloor m/2 \rfloor$  available processors will hold for any problem of reasonably large size. Therefore, these complexity results are mainly of theoretical interest.

The parallel algorithms proposed in [7] and [9] are designed for a shared-memory multiprocessor with low synchronization overhead. The Denelcor HEP computer is an example. The parallel ZIGZAG scheme proposed by Lord et al. in [9] can be viewed as implementing the Givens sequence shown in Figure 2 on a square matrix of order  $n$  using  $\lfloor n/2 \rfloor$  processors. The asymptotic efficiency of this algorithm is 44.4%. Even for  $n$  as small as 17, the efficiency is 45%, which is already quite close to the asymptotic value [9]. Thus, the predicted (and actual) efficiency of this algorithm is low, although this is not surprising given the large number of processors employed. The COLSWP (column-sweep) Givens scheme proposed in [9] and the pipelined Givens method proposed in [7] assume a relatively small number of processors; i.e., the number of processors is assumed to be much less than  $n$ . When  $p$ , the number of processors, is much less than  $\lfloor m/2 \rfloor$ , the parallelism allowed by a particular Givens sequence can be exploited in a variety of ways. Indeed the COLSWP algorithm can be viewed as

implementing the Givens sequence shown in Figure 2 in a column-by-column manner so that the zero elements in each column are created by the same processor. For the pipelined Givens method, the same Givens sequence is implemented in a row-by-row manner so that the zero elements in each row are created by the same processor.

On a shared-memory computer with multiple processes running in parallel, the processes must be synchronized in order to prevent their simultaneously updating shared data and thereby corrupting it. This synchronization can be achieved through the use of “locks”. A lock ensures that only one process at a time can access a shared data structure. A lock has two values: locked and unlocked. Before attempting to access a shared data structure, a process waits until the lock associated with the data structure is unlocked. The process then *locks* the lock, accesses the data structure, and *unlocks* the lock. In this article we measure synchronization cost by the number of times a lock is accessed.

The synchronization cost of the parallel schemes in [7] and [9] is not analyzed. This is reasonable because *low* synchronization overhead was assumed. An analysis of the algorithm in [7] shows that the synchronization cost is a function of  $m$ ,  $n$ , and  $p$  [4]. The dependence of the synchronization cost on the row dimension  $m$  is undesirable when  $m \gg n$ , which is not uncommon. This prompted us to devise a parallel algorithm for which the synchronization cost is *independent* of the row dimension  $m$ . Such a scheme is particularly suitable for multiprocessors whose synchronization overhead is significant.

## 2 The Algorithm

The algorithm we propose has been designed for shared-memory multiprocessors. The main objective of our design is to reduce the synchronization cost and processor idle time by assigning the processors to work on disjoint sets of rows as much as possible. The algorithm has two phases: an independent annihilation phase (IAP) and a cooperative annihilation phase (CAP). For ease of exposition, we first describe the algorithm for factoring an  $m \times n$  matrix  $A$  using  $p$  processors, where  $m$  and  $n$  are integral multiples of  $p$ , and  $m/p \geq n$ . For example, assuming  $p = 4$ , a matrix of dimension  $32 \times 8$  satisfies the above condition.

## 2.1 The Independent Annihilation Phase

In the IAP, each processor is assigned a block of  $m/p$  consecutive rows of  $A$ . Each processor reduces its own block of rows to an upper triangular submatrix of order  $n$  ( $n \leq m/p$  by assumption). Figure 4 depicts the action by four processors on a  $32 \times 8$  matrix.

For this example, each processor performs 28 rotations *independently* and *simultaneously*. Note that in the IAP, each processor does equal work, and there is no idle time or synchronization cost.

## 2.2 The Cooperative Annihilation Phase

In the CAP, the rows in the  $p$  upper triangular submatrices are assigned to groups by collecting the rows with leading nonzero in column  $j$  into a group  $G_j$ ,  $1 \leq j \leq n$ . For the case  $m/p \geq n$ , at the end of the IAP we have exactly  $p$  rows in each  $G_j$  for  $1 \leq j \leq n$ . Note that the collection of rows in  $G_j$  can be viewed as a  $p \times (n - j + 1)$  rectangular submatrix. If a Givens rotation is applied to the  $i^{\text{th}}$  row and the  $j^{\text{th}}$  row to annihilate the leading nonzero  $a_{j,k}$  in the  $j^{\text{th}}$  row, then row  $i$  is referred to as the “pivot row”. By choosing one row in each group as the pivot row, the task of eliminating the leading nonzeros in the remaining  $(p - 1)$  rows in one group is *independent* of the same task in another group. We shall arbitrarily choose the lowest numbered row in each group as the pivot row. In order to have these independent tasks performed by the  $p$  processors simultaneously, and also maintain the work load balance, we assign groups  $G_1$  to  $G_p$  to the  $p$  processors in order, with the assignment of group  $G_{p+1}$  “wrapping around” to processor 1. Figure 5 illustrates the *initial* data assignment for the example in Figure 4. The matrix elements assigned to the  $i^{\text{th}}$  processor are labelled by  $P_i$ .

×	×	×	×	×	×	×	×
1	×	×	×	×	×	×	×
2	3	×	×	×	×	×	×
4	5	6	×	×	×	×	×
7	8	9	10	×	×	×	×
11	12	13	14	15	×	×	×
16	17	18	19	20	21	×	×
22	23	24	25	26	27	28	×
×	×	×	×	×	×	×	×
1	×	×	×	×	×	×	×
2	3	×	×	×	×	×	×
4	5	6	×	×	×	×	×
7	8	9	10	×	×	×	×
11	12	13	14	15	×	×	×
16	17	18	19	20	21	×	×
22	23	24	25	26	27	28	×
×	×	×	×	×	×	×	×
1	×	×	×	×	×	×	×
2	3	×	×	×	×	×	×
4	5	6	×	×	×	×	×
7	8	9	10	×	×	×	×
11	12	13	14	15	×	×	×
16	17	18	19	20	21	×	×
22	23	24	25	26	27	28	×
×	×	×	×	×	×	×	×
1	×	×	×	×	×	×	×
2	3	×	×	×	×	×	×
4	5	6	×	×	×	×	×
7	8	9	10	×	×	×	×
11	12	13	14	15	×	×	×
16	17	18	19	20	21	×	×
22	23	24	25	26	27	28	×

Figure 4: Independent Annihilation by Four Processors

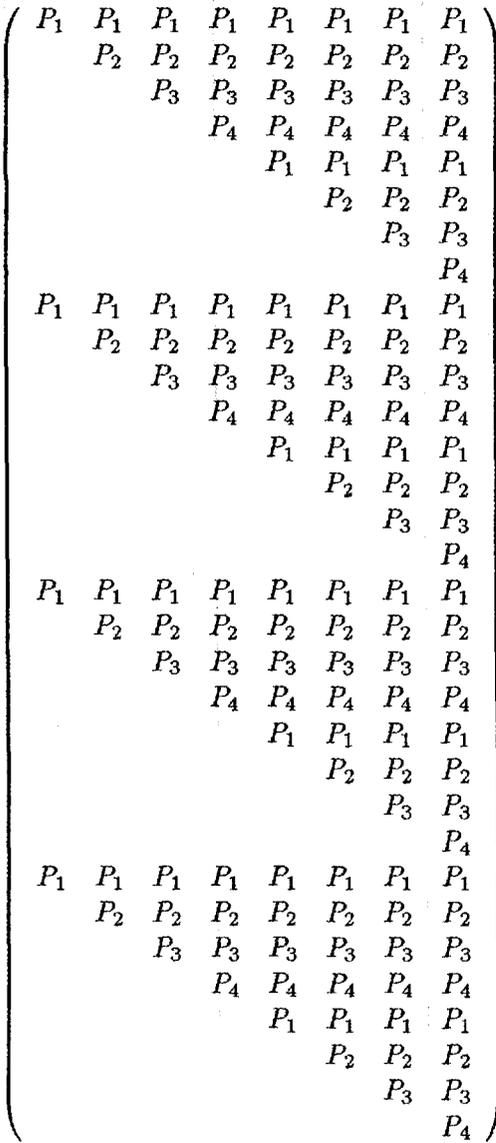


Figure 5: CAP-Initial Data Distribution among Four Processors

Note that the “top” submatrix in Figure 5 contains the pivot rows, and the main diagonals of the remaining  $(p - 1)$  submatrices correspond to the nonzero elements which can be eliminated by the four processors independently and simultaneously. Recall that using the wrap mapping, each processor is assigned  $n/p$  groups, and each group has  $(p - 1)$  leading nonzeros to be eliminated. Therefore, after the  $p$  processors each perform  $n(p - 1)/p$  rotations (in parallel), we have eliminated the  $(p - 1)$  main diagonals of the “bottom”  $(p - 1)$  triangular submatrices. The remaining nonzeros are depicted in Figure 6. We then apply the same idea to eliminate the elements along the first superdiagonal in each of the  $(p - 1)$  submatrices, using the rows in the top submatrix as the pivot rows.

The elimination of the diagonals can be implemented in several ways. In the implementation we describe in the next section, the pivot rows in the top submatrix are statically assigned to the  $p$  processors using a wrap mapping as explained earlier, and the remaining data are accessed in groups by each processor. To be accurate in what follows, we redefine group  $G_j$  initially to contain  $(p - 1)$  rows with leading nonzero in column  $j$  excluding the  $j^{\text{th}}$  pivot row. We adopt the convention of labelling the elements of pivot rows by  $P_i$  if they are assigned to processor  $P_i$  in the static mapping, and the remaining rows are labelled by their respective group number  $G_j$ . Referring to Figure 7, we see that after the annihilation of main diagonals, the leading nonzero position in group  $G_j$  becomes  $(j + 1)$ . Therefore, the processor which is assigned the  $j^{\text{th}}$  ( $j > 1$ ) pivot row will now access group  $G_{j-1}$  to eliminate the current leading nonzero elements in column  $j$ . We illustrate this mapping in Figure 7, where the three superdiagonals correspond to the elements to be eliminated in this step.

After the elements on the first superdiagonals are eliminated, the processor which is assigned the  $j^{\text{th}}$  ( $j > 2$ ) pivot row can now annihilate the current leading nonzero elements in group  $G_{j-2}$ . The elements to be eliminated in this step lie on the second superdiagonals. Thus, it is clear that if the  $p$  processors simply synchronize with each other before starting annihilation of elements along each diagonal, the  $(p - 1)$  submatrices will be eliminated one diagonal at a time without any other synchronization cost. Note in particular that all processors will be accessing disjoint sets of pivot rows and disjoint groups when eliminating elements along the same diagonal. Thus, there are *no* shared data. Since each  $n \times n$  upper triangular submatrix has  $n$  diagonals, the synchronization cost for the parallel algorithm is clearly  $O(n)$ . Such an implementation is particularly suitable for



$$\begin{pmatrix}
 P_1 & P_1 \\
 & P_2 \\
 & & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 \\
 & & & P_4 & P_4 & P_4 & P_4 & P_4 \\
 & & & & P_1 & P_1 & P_1 & P_1 \\
 & & & & & P_2 & P_2 & P_2 \\
 & & & & & & P_3 & P_3 \\
 & & & & & & & P_4 \\
 0 & G_1 \\
 & 0 & G_2 & G_2 & G_2 & G_2 & G_2 & G_2 \\
 & & 0 & G_3 & G_3 & G_3 & G_3 & G_3 \\
 & & & 0 & G_4 & G_4 & G_4 & G_4 \\
 & & & & 0 & G_5 & G_5 & G_5 \\
 & & & & & 0 & G_6 & G_6 \\
 & & & & & & 0 & G_7 \\
 & & & & & & & 0 \\
 0 & G_1 \\
 & 0 & G_2 & G_2 & G_2 & G_2 & G_2 & G_2 \\
 & & 0 & G_3 & G_3 & G_3 & G_3 & G_3 \\
 & & & 0 & G_4 & G_4 & G_4 & G_4 \\
 & & & & 0 & G_5 & G_5 & G_5 \\
 & & & & & 0 & G_6 & G_6 \\
 & & & & & & 0 & G_7 \\
 & & & & & & & 0 \\
 0 & G_1 \\
 & 0 & G_2 & G_2 & G_2 & G_2 & G_2 & G_2 \\
 & & 0 & G_3 & G_3 & G_3 & G_3 & G_3 \\
 & & & 0 & G_4 & G_4 & G_4 & G_4 \\
 & & & & 0 & G_5 & G_5 & G_5 \\
 & & & & & 0 & G_6 & G_6 \\
 & & & & & & 0 & G_7 \\
 & & & & & & & 0
 \end{pmatrix}$$

Figure 7: CAP-Data mapping to eliminate the first superdiagonals

a machine with *high* synchronization overhead. We refer to this version of implementation as the *synchronous* implementation. In the next section we discuss some implementation details and describe an *asynchronous* implementation which can further reduce the processor idle time by increasing the synchronization cost to  $O(n^2/p)$ .

### 3 Implementation Issues

The implementation of the independent annihilation phase (IAP) is straightforward. We therefore concentrate on the implementation of the cooperative annihilation phase (CAP) in this section. Note that the data each individual processor will access during the entire elimination process is *dictated* by the initial static allocation of pivot rows. For example, if processor  $P_i$  is assigned pivot rows

$$\{k_1, k_2, \dots, k_{\mu_i}\},$$

then processor  $P_i$  will participate in eliminating the elements along the main diagonal by accessing data in groups

$$\{G_{k_1}, G_{k_2}, \dots, G_{k_{\mu_i}}\}.$$

To eliminate elements along the  $j^{\text{th}}$  superdiagonal, processor  $P_i$  will access data in groups

$$\{G_{k_1-j}, G_{k_2-j}, \dots, G_{k_{\mu_i}-j}\}.$$

Note that encountering a group  $G_\rho$  with  $\rho \leq 0$  simply indicates that there are no more rows with leading nonzero in the same position as the corresponding pivot row. For example, if  $(k_1 - j) \leq 0$ , then the pivot row  $k_1$  will not be modified any more and is row  $k_1$  of the upper triangular factor  $R$ .

Observe that the group  $G_\rho$  must be eliminated against pivot rows  $\rho, \rho+1, \dots, n$  in strict order. Whether  $G_\rho$  can be reduced against pivot row  $(\rho + \eta)$  can be determined by simply checking whether the current position of its leading nonzero elements is in column  $(\rho + \eta)$ . Therefore, if we associate with each group  $G_j$  a shared variable  $first[j]$  to indicate the current position of its leading nonzeros, all processors can proceed by themselves to complete their share of work in the entire CAP process using the following synchronization mechanism. For convenience in describing the algorithm, we assume that processor  $P_i$  is assigned pivot rows  $\{k_1, k_2, \dots, k_{\mu_i}\}$ . The pivot row numbers for  $P_i$  are stored in a local array  $pvt_s[j], 1 \leq j \leq \mu_i$ . We also need a global array  $first$  to record the current position of the leading nonzeros for each

group, and a local array *map* to identify the groups which are currently due to be processed by each individual processor. Note that *first* is shared among multiple processors and its exclusive access must be protected. The basic algorithm executed by processor  $P_i$  can now be expressed in the following form.

```

for  $j = 1, 2, \dots, \mu_i$  do
     $pvt_s[j] \leftarrow k_j$ 
     $map[j] \leftarrow k_j$ 
 $jstr_t \leftarrow 1$ 
while  $jstr_t \leq \mu_i$  do
    for  $j = jstr_t, jstr_t + 1, \dots, \mu_i$  do
         $\rho \leftarrow map[j]$ 
        wait until  $first[\rho] = pvt_s[j]$ 
        reduce rows in group  $G_\rho$  using pivot row  $pvt_s[j]$ 
         $first[\rho] \leftarrow first[\rho] + 1$ 
         $map[j] \leftarrow map[j] - 1$ 
    if  $map[jstr_t] = 0$  then
         $jstr_t \leftarrow jstr_t + 1$ 

```

The algorithm and its implementation can easily handle the case where  $m/p < n$ . For  $p = 4$ , a matrix of order  $16 \times 8$  is such an example. For this example, in the IAP each processor will reduce its block of rows to an  $m/p$  by  $n$  upper trapezoidal submatrix. The number of rotations performed by each processor is  $(m/p - 1)m/2p$ . The remaining nonzero elements are shown in Figure 8 for an  $16 \times 8$  example.

When  $m/p < n$ , the top upper trapezoidal submatrix does not contain a full set of pivot rows. For the example in Figure 8, the pivot rows 5, 6, 7, and 8 are all to be generated during the elimination process in the CAP. An important observation which facilitates a clean implementation is that the currently non-existent pivot rows will each be generated from the data in an existing group. Therefore, we can still statically assign  $n$  pivot rows to the  $p$  processors and record them in the *pvt\_s* and *map* arrays as before. In addition, each processor records whether group  $G_i$  exists for  $1 \leq i \leq n$ . For each entry in  $map[i]$ , the processor will process  $G_{map[i]}$  if it exists, and will do nothing other than updating  $map[i]$  to be  $(map[i] - 1)$  if  $G_{map[i]}$  does not exist. Since  $map[i]$  is updated each time, eventually  $G_{map[i]}$  refers to existing data with leading nonzero in position  $pvt_s[i]$ . One row will now be

$$\begin{pmatrix}
 P_1 & P_1 \\
 & P_2 \\
 & & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 \\
 & & & P_4 & P_4 & P_4 & P_4 & P_4 \\
 G_1 & G_1 \\
 & G_2 \\
 & & G_3 & G_3 & G_3 & G_3 & G_3 & G_3 \\
 & & & G_4 & G_4 & G_4 & G_4 & G_4 \\
 G_1 & G_1 \\
 & G_2 \\
 & & G_3 & G_3 & G_3 & G_3 & G_3 & G_3 \\
 & & & G_4 & G_4 & G_4 & G_4 & G_4 \\
 G_1 & G_1 \\
 & G_2 \\
 & & G_3 & G_3 & G_3 & G_3 & G_3 & G_3 \\
 & & & G_4 & G_4 & G_4 & G_4 & G_4
 \end{pmatrix}$$

Figure 8: End of IAP – A  $16 \times 8$  Example

taken from  $G_{map[i]}$  to become the pivot row, the remaining rows (if there are any) will be further reduced.

## 4 Analysis of Synchronization Cost

The actual implementation of the synchronization mechanism for the general case ( $m/p < n$  or  $m/p \geq n$ ) needs to maintain two attributes for each group  $G_i$ , namely  $first[i]$ , the current position of the leading nonzero elements in  $G_i$ , and  $nrows[i]$ , the current number of rows in  $G_i$ . Both items are shared data to be updated and read by multiple processors. The exclusive access of these two items is ensured by a lock associated with group  $G_i$ , namely  $glock[i]$ . We let  $\beta$  denote the total time of *acquiring* and *releasing* a lock. The synchronization mechanism described in the last section requires  $\beta$  time for each updating of the array  $first$ . Note that for the general case,  $nrows[i]$  can be updated together with  $first[i]$  under the same lock.

We now analyze the synchronization cost for the cases  $m/p \geq n$  and  $m/p < n$  separately. For convenience, we assume  $m$  and  $n$  are integral multiples of  $p$ . If  $m/p \geq n$ , we have  $n$  pivot rows and  $n$  groups of data. Since the  $i^{th}$  pivot row is used to eliminate the leading nonzero elements in groups  $G_i, G_{i-1}, \dots$ , and  $G_1$  in strict order, there are exactly  $i$  groups to be processed by the  $i^{th}$  pivot row. The synchronization cost associated with the  $i^{th}$  pivot row is therefore  $i\beta$ . Recalling that the pivot rows are assigned to the  $p$  processors using a wrap mapping, we can therefore compute the synchronization cost of processor  $P_k$  by

$$\begin{aligned} S(P_k) &= \beta \sum_{\ell=0}^{\frac{n}{p}-1} (k + p\ell) \\ &= \beta \left( \frac{n^2}{2p} - \frac{n}{2} + k\frac{n}{p} \right), \end{aligned} \tag{1}$$

where  $1 \leq k \leq p$ . For the case  $m/p < n$ , the synchronization cost associated with the first  $m/p$  pivot rows is  $i\beta$ , where  $1 \leq i \leq m/p$ . For  $m/p < j \leq n$ , the synchronization cost associated with row  $j$  is at most  $\beta m/p$ . Assuming further that  $m/p$  is also an integral multiple of  $p$ , the synchronization cost of processor  $P_k$  is as shown in (2).

$$\begin{aligned}
S(P_k) &\leq \beta \left( \sum_{\ell=0}^{\frac{m}{p}-1} (k + p\ell) \right) + \beta \left( \frac{m(n - m/p)}{p} \right) \\
&= \beta \left( \frac{mn}{p^2} - \frac{m^2}{2p^3} + k \frac{m}{p^2} - \frac{m}{2p} \right) \tag{2}
\end{aligned}$$

## 5 Analysis of Work Load Distribution

We now examine how the computational work is distributed among the  $p$  processors in the CAP. We denote the work performed by the  $k^{\text{th}}$  processor by  $W_p(P_k)$ ,  $1 \leq k \leq p$ , and first consider the case  $m/p \geq n$ . Processor  $P_k$  is assigned the set of pivot rows  $\{k, k + p, k + 2p, \dots, n - k + p\}$ . As shown in the last section, the number of groups to be processed by the  $i^{\text{th}}$  pivot row is exactly  $i$ , and there are  $(p - 1)$  rows in each group. The elimination of one of the  $(p - 1)$  leading nonzeros in column  $i$  requires one rotation applied to a pair of rows of length  $(n - i + 1)$ , which amounts to  $4(n - i + 1)$  multiplicative operations. Thus,

$$\begin{aligned}
W_p(P_k) &= 4(p - 1) \sum_{\ell=0}^{\frac{n}{p}-1} (k + \ell p)(n - k - \ell p + 1) \\
&= \frac{(p - 1)}{p} \left( \frac{2}{3}n^3 + 2n^2 - n \left( 2p + \frac{2}{3}p^2 - 4k - 4kp + 4k^2 \right) \right) \tag{3}
\end{aligned}$$

where  $1 \leq k \leq p$ . Note that the total serial work in the CAP can be computed by assuming that the  $(p - 1)$  upper triangular submatrices will be eliminated by a single processor one submatrix at a time, using the rows in the top submatrix as pivot rows. This yields

$$\begin{aligned}
W_1 &= 4(p - 1) \sum_{i=1}^n \frac{i(i + 1)}{2} \\
&= (p - 1) \left( \frac{2}{3}n^3 + 2n^2 + \frac{4}{3}n \right) \cdot \tag{4}
\end{aligned}$$

The optimal work load distribution is thus

$$\frac{W_1}{p} = \frac{(p - 1)}{p} \left( \frac{2}{3}n^3 + 2n^2 + \frac{4}{3}n \right) \cdot \tag{5}$$

Comparing equations (3) and (5), we see that the work distribution of the CAP differs from the optimal distribution in only the low order  $O(n)$  terms.

For the case  $m/p < n$ , we further assume that  $m/p$  is an integral multiple of  $p$ , and that  $n = s\frac{m}{p}$ , where  $s$  is an integer in  $[2, p]$ . The following observations are useful in deriving the work load of processor  $P_k$ .

**Observation 1.**  $\frac{m}{p} < n$  and  $n = s\frac{m}{p}$  imply that  $\frac{m}{p}$  pivot rows exist at the beginning of the CAP, and that the other  $(s-1)\frac{m}{p}$  pivot rows are to be taken from the remaining  $\frac{m}{p}(p-1)$  rows during the CAP.

**Observation 2.** The wrap mapping dictates that pivot rows

$$\left\{ \begin{array}{l} \frac{m}{p} + k, \frac{m}{p} + p + k, \dots, \frac{m}{p} + \left(\frac{m}{p^2} - 1\right)p + k, \\ 2\frac{m}{p} + k, 2\frac{m}{p} + p + k, \dots, 2\frac{m}{p} + \left(\frac{m}{p^2} - 1\right)p + k, \\ \vdots \\ (s-1)\frac{m}{p} + k, (s-1)\frac{m}{p} + p + k, \dots, (s-1)\frac{m}{p} + \left(\frac{m}{p^2} - 1\right)p + k \end{array} \right\}$$

are to be generated by processor  $P_k$ .

**Observation 3.** The total number of rows to be reduced by processor  $P_k$  by pivot row  $\left(i\frac{m}{p} + \ell p + k\right)$  is  $\left(\frac{m}{p}(p-i) - \ell p - k\right)$ , and the length of each row is  $\left(n - i\frac{m}{p} - \ell p - k + 1\right)$ .

Applying these observations, we obtain

$$\begin{aligned} W_p(P_k) &= 4(p-1) \sum_{\ell=0}^{\frac{m}{p^2}-1} (k + p\ell)(n - k - p\ell + 1) + \\ &\quad 4 \sum_{i=1}^{s-1} \sum_{\ell=0}^{m/p^2-1} \left(\frac{m}{p}(p-i) - \ell p - k\right) \left(n - i\frac{m}{p} - \ell p - k + 1\right) \\ &= \left(2s^2 - 2s + \frac{2}{3}\right) \frac{m^3}{p^3} - \frac{2}{3}s^3 \frac{m^3}{p^4} + (4s-2) \frac{m^2}{p^2} - 2s^2 \frac{m^2}{p^3} \\ &\quad + O(m). \end{aligned} \tag{6}$$

The total CAP serial work for the case  $m/p < n$  can be computed by subtracting the total IAP serial work from the total work:

$$W_1 = 4 \sum_{i=1}^n (m-i)(n-i+1) - 4 \sum_{i=1}^{\frac{m}{p}-1} p \left(\frac{m}{p} - i\right) (n-i+1)$$

$$\begin{aligned}
&= 2mn^2 - \frac{2}{3}n^3 - 2\frac{m^2n}{p} + 4mn - 2n^2 + \frac{2}{3}\frac{m^3}{p^2} - 2\frac{m^2}{p} \\
&\quad + \frac{4}{3}(m - n) .
\end{aligned} \tag{7}$$

The optimal work load distribution is  $W_1/p$ . In order to compare with  $W_p(P_k)$ , we simplify  $W_1/p$  by substituting  $n = s\frac{m}{p}$ . This yields

$$\begin{aligned}
\frac{W_1}{p} &= \left(2s^2 - 2s + \frac{2}{3}\right) \frac{m^3}{p^3} - \frac{2}{3}s^3 \frac{m^3}{p^4} + (4s - 2) \frac{m^2}{p^2} - 2s^2 \frac{m^2}{p^3} \\
&\quad + O(m).
\end{aligned} \tag{8}$$

Comparing  $W_p(P_k)$  with  $W_1/p$  shows that

$$W_p(P_k) - \frac{W_1}{p} = O(m). \tag{9}$$

Thus, for both cases ( $m/p < n$ ,  $m/p \geq n$ ),  $W_p(P_k)$  and  $W_1/p$  differ only in their low order terms.

## 6 Performance Analysis

To analyze the performance of the parallel algorithm in the CAP, first note that the nonzero elements in the same group are eliminated by *different* processors in strict order. It is clear that when  $p$  groups of data are processed in parallel, the time is bounded by the processing time of the group which requires the largest amount of computation. We consider the parallel time (in units of multiplicative operations) for factoring an  $m \times n$  matrix using  $p$  processors. As usual we assume  $m$  and  $n$  are integral multiples of  $p$ .

We first consider the case  $m/p \geq n$ . In this case, there are  $n$  groups of data, each of  $(p-1)$  rows, to be eliminated entirely in the CAP. Because the  $n$  groups are assigned to the  $p$  processors using a wrap mapping, an upper bound of the CAP parallel time  $T_p$  is given by assuming that the nonzeros in groups  $G_1, G_{p+1}, G_{2p+1}, \dots, G_{n-p+1}$  are eliminated sequentially. Letting  $\Lambda(T_p)$  denote the upper bound of  $T_p$ , we obtain

$$\begin{aligned}
\Lambda(T_p) &= 4(p-1) \sum_{\ell=0}^{\frac{n}{p}-1} \frac{1}{2} (n - p\ell)(n - p\ell + 1) \\
&= \frac{(p-1)}{p} \left( \frac{2}{3}n^3 \right) + pn^2 - \frac{n^2}{p} + O(n).
\end{aligned} \tag{10}$$

Note that  $T_p = \Lambda(T_p)$  for the synchronous implementation of the CAP.

For the asynchronous implementation described in detail in Section 3, note that when  $P_\alpha$  is processing group  $G_{n-p+1}$  to eliminate its leading nonzeros in the  $j^{\text{th}}$  column, it is possible for a different processor  $P_\gamma$  to start its processing of group  $G_1$  to eliminate its leading nonzeros in the  $(j+1)^{\text{st}}$  position. We thus expect  $T_p < \Lambda(T_p)$ . Nevertheless, since the serial time for the CAP is given by

$$T_1 = (p-1) \left( \frac{2}{3} n^3 \right) + O(n^2),$$

$\Lambda(T_p)$  is *optimal* in its leading term. We therefore cannot expect dramatic improvement in  $T_p$  using the asynchronous version of the implementation. Experimental results given in Section 7 confirm this expectation.

For the case  $m/p < n$ , we again assume that  $m/p$  is an integral multiple of  $p$ , and that  $n = s \frac{m}{p}$ , where  $s$  is an integer in  $[2, p]$ . The following observations are helpful in analyzing the performance of the algorithm.

**Observation 1.**  $\frac{m}{p} < n$  and  $n = s \frac{m}{p}$  imply that  $(s-1) \frac{m}{p}$  pivot rows are absent at the beginning of the CAP.

**Observation 2.** Since processor  $P_k$  is assigned pivot rows

$$\left\{ k, k+p, \dots, \frac{m}{p} - p + k \right\},$$

the  $(p-1)$  rows in group  $G_{\frac{m}{p}}$  will initially be reduced by processor  $P_p$  and have the position of their leading nonzeros updated to be  $\text{first} \left( \frac{m}{p} \right) = \frac{m}{p} + 1$ . Because pivot row  $\left( \frac{m}{p} + 1 \right)$  does not exist, when group  $G_{\frac{m}{p}}$  is next processed by processor  $P_1$ , one row will be taken to serve as the pivot row. In general, the wrap mapping dictates that the first  $(p-1)$  pivot rows are taken from group  $G_{\frac{m}{p}}$ , and the next  $(p-1)$  pivot rows are taken from group  $G_{\frac{m}{p}-1}$ , and so on, until all of the pivot rows are present.

**Observation 3.** Based on observation 2, an upper bound for the parallel time of the CAP is the total time to process the groups  $G_1, G_{p+1}, G_{2p+1}, \dots$  and  $G_{\frac{m}{p}-p+1}$  sequentially. Note that because the number of rows in each group could change dynamically during the CAP, we have to keep track of the actual number of rows in our analysis in order to have a sufficiently tight upper bound.

The following lemmas are needed to prove Theorem 4.

**Lemma 1** *If all of the  $n$  pivot rows are already present, then the total number of multiplicative operations required to eliminate the nonzeros in groups  $G_1, G_{p+1}, G_{2p+1}, \dots$  and  $G_{\frac{m}{p}-p+1}$  is given by*

$$\begin{aligned}
\hat{T}_p &= 4(p-1) \sum_{\ell=0}^{\frac{m}{p^2}-1} \frac{1}{2} (n-p\ell)(n-p\ell+1) \\
&= 2\frac{mn^2}{p} - 2\frac{m^2n}{p^2} + \frac{2m^3}{3p^3} + 2mn + \frac{2}{3}m - \frac{m^2}{p} + \frac{1}{3}pm - \\
&\quad 2\frac{n^2m}{p^2} - 2\frac{mn}{p^2} + 2\frac{m^2n}{p^3} + \frac{m^2}{p^3} - \frac{m}{p} - \frac{2m^3}{3p^4} \\
&= (2s^2 - 2s + \frac{2}{3})\frac{m^3}{p^3} - (2s^2 - 2s + \frac{2}{3})\frac{m^3}{p^4} + O(m^2). \quad (11)
\end{aligned}$$

**Lemma 2** *If we assume for convenience that  $\mu = \frac{(s-1)\frac{m}{p}}{p(p-1)}$  is an integer, then  $(\mu-1)(p-1)$  pivot rows will be taken from groups  $G_{\frac{m}{p}-p+1}, G_{\frac{m}{p}-2p+1}, \dots$ , and  $G_{\frac{m}{p}-(\mu-1)p+1}$  in order. Moreover, the size of the  $\ell^{\text{th}}$  row taken from group  $G_{\frac{m}{p}-ip+1}$  is given by*

$$f(m, n, i, p, \ell) = n - \frac{m}{p} - (p-1)^2 - (i-1)(p-1)p - \ell + 1.$$

**Proof:** This follows directly from observation 2.  $\square$

**Lemma 3** *The number of multiplicative operations to be saved by not eliminating a row of size  $\eta$  is  $\eta(\eta+1)/2$ .*

**Theorem 4** *An upper bound for the parallel time for the CAP is given by*

$$\Lambda(T_p) = \left(2s^2 - 2s + \frac{2}{3}\right) \frac{m^3}{p^3} - \frac{2}{3}s^3 \frac{m^3}{p^4} + O(m^2).$$

**Proof:** From Lemmas 1 to 3, we have

$$\begin{aligned}
\Lambda(T_p) &= \hat{T}_p - 4 \sum_{i=1}^{\mu-1} \sum_{\ell=1}^{p-1} \frac{1}{2} f(m, n, i, p, \ell) (f(m, n, i, p, \ell) + 1), \\
&= \left(2s^2 - 2s + \frac{2}{3}\right) \frac{m^3}{p^3} - \frac{2}{3}s^3 \frac{m^3}{p^4} + O(m^2). \quad (12)
\end{aligned}$$

□

The total CAP serial work  $W_1$  for the case  $m/p < n$  was derived in the last section. We use it here to represent the serial time  $T_1$ . Comparing  $\Lambda(T_p)$  in Theorem 4 with  $T_1/p$  from equation (8), we have

$$\Lambda(T_p) - \frac{T_1}{p} = O(m^2). \quad (13)$$

Since  $T_p \leq \Lambda(T_p)$ , we have shown again that the CAP parallel time  $T_p$  is *optimal* in its leading term. The actual performance of the algorithm is reported in the next section.

## 7 Numerical Experiments

Our experiments were performed on an 8-processor Sequent Balance 8000 parallel computer, and our algorithms were implemented in FORTRAN. Since the use of Givens rotations is row-oriented and a two-dimensional array is stored column by column in FORTRAN, the transpose of the coefficient matrix was stored and operations on the matrix were done in a column-oriented manner. The execution times reported below exclude only the time for generating the coefficient matrix. In particular, for the parallel algorithm, the execution time includes the overhead in creating multiple processes. Since spawning a child process on the Balance 8000 is an expensive operation (30-50 milliseconds), this overhead is significant for small problems and large numbers of processes.

The execution times (in seconds) of the serial and parallel algorithms are denoted by  $T_s$  and  $T$  respectively, and as in previous sections,  $p$  denotes the number of processors.

Table 1 gives some timing results of the serial algorithm and the asynchronous implementation of the parallel algorithm. The entries of the  $m \times n$  test matrices were generated using a random number generator. The reported efficiency is computed using

$$efficiency = \frac{T_s}{p \times T}.$$

Table 2 compares the execution time of the asynchronous implementation with the synchronous implementation for the same set of matrices. The results show that the former is slightly faster than the latter for all test matrices. This is not surprising because the synchronization overhead on

$m$	$n$	$p$	$T_s$	$T$	<i>efficiency</i>
100	100	1	47.17		
		2		24.07	98%
		3		16.25	97%
		4		12.13	97%
		5		9.82	96%
		6		8.47	93%
		7		7.63	88%
200	200	1	368.95		
		2		187.80	98%
		3		125.60	98%
		4		93.90	98%
		5		75.20	98%
		6		63.60	97%
		7		54.70	96%
200	40	1	22.80		
		2		11.63	98%
		3		7.87	97%
		4		5.98	95%
		5		4.85	94%
		6		4.18	91%
		7		3.68	89%
500	100	1	336.30		
		2		168.90	99.6%
		3		113.00	99.3%
		4		84.80	99.3%
		5		67.90	99.0%
		6		57.27	97.8%
		7		49.30	97.5%

Table 1: Execution time on the Balance 8000

the Balance 8000 is very low, so the saving from reducing synchronization cost is evidently less significant than the saving from reducing idle time. However, as our analysis in Section 6 predicted, the difference is not very great.

$m$	$n$	$p$	Asynchronous $T$	Synchronous $T$
100	100	2	24.07	24.10
		3	16.25	16.73
		4	12.13	12.87
		5	9.82	10.87
		6	8.47	9.82
		7	7.63	8.88
		200	200	2
3	125.60			127.07
4	93.90			96.88
5	75.20			79.38
6	63.60			69.13
7	54.70			61.50
200	40			2
		3	7.87	7.97
		4	5.98	6.07
		5	4.85	5.03
		6	4.18	4.48
		7	3.68	3.95
		500	100	2
3	113.00			113.27
4	84.80			85.37
5	67.90			69.02
6	57.27			58.58
7	49.30			51.62

Table 2: Execution time on the Balance 8000

To study the effect of high synchronization cost, we simulated that situation by executing a dummy assignment statement five hundred times whenever a lock was accessed. We then compared the asynchronous implementation, the synchronous implementation, and our implementation of the pipelined Givens method given in [7] for problems of three different sizes. For the  $1000 \times 100$  matrix, since  $m \gg n$ , the effect of high synchronization

cost would be expected to be most dramatic for the pipelined Givens method, since its synchronization cost is  $O(mn/p)$  [4]. The difference between the pipelined Givens method and the asynchronous Givens method would be expected to diminish as  $m$  approaches  $n$ . Table 3 confirms these expectations. The synchronous Givens algorithm has the lowest synchronization cost, and performs best among the three when the synchronization cost is high and  $m \gg n$ .

$m = 1000, n = 100, p = 5, T_s = 696.8 \text{ sec}$		
Parallel Algorithm	Execution Time	Efficiency
Pipelined Givens	211.80 sec	66%
Asynchronous Givens	144.03 sec	97%
Synchronous Givens	141.77 sec	98%
$m = 500, n = 100, p = 5, T_s = 336.3 \text{ sec}$		
Parallel Algorithm	Execution Time	Efficiency
Pipelined Givens	101.57 sec	66%
Asynchronous Givens	71.58 sec	94%
Synchronous Givens	70.52 sec	98%
$m = 100, n = 100, p = 5, T_s = 47.2 \text{ sec}$		
Parallel Algorithm	Execution Time	Efficiency
Pipelined Givens	13.87 sec	68%
Asynchronous Givens	10.52 sec	90%
Synchronous Givens	11.60 sec	81%

Table 3: The Effect of High Synchronization Cost

## References

- [1] H.M. Ahmed, J-M. Delosme, and M. Morf. Highly concurrent computing structures for matrix arithmetic and signal processing. *Computer*, 15:65–82, 1982.
- [2] A. Bojanczyk, R.P. Brent, and H.T. Kung. Numerically stable solution of dense systems of linear equations using mesh-connected processors. *SIAM J. Sci. Stat. Comput.*, 5, 1984.
- [3] R.M. Chamberlain and M.J.D. Powell. *QR Factorization for Linear Least Squares Problems on the Hypercube*. Technical Report CCS 86/10, Dept. of Science and Technology, Chr. Michelsen Institute, Bergen, Norway, 1986.
- [4] E. C. H. Chu. *Parallel Algorithms for Linear Equations and Least Squares Problems*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1, 1987.
- [5] M. Cosnard, J.-M. Muller, and Y. Robert. Parallel QR Decomposition of a Rectangular Matrix. *Numer. Math.*, 48:239–249, 1986.
- [6] M. Cosnard and Y. Robert. Complexité de la factorisation QR en parallèle. *C.R. Acad. Sci.*, 297:549–552, 1983.
- [7] J.J. Dongarra, A.H. Sameh, and D.C. Sorensen. Implementation of some concurrent algorithms for matrix factorization. *Parallel Computing*, 3:25–34, 1985.
- [8] W.M. Gentleman and H.T. Kung. Matrix triangularization by systolic arrays. In *Real Time Signal Processing IV: SPIE Proceeding*, pages 19–26, Society of Photo-Optical Instrumentation Engineers, Bellingham, WA, 1981.
- [9] R. Lord, J. Kowalik, and S. Kumar. Solving linear algebraic equations on an MIMD computer. *J. Assoc. Comput. Mach.*, 30:103–117, 1983.
- [10] Franklin T. Luk. A rotation method for computing the QR-decomposition. *SIAM J. Sci. Stat. Comput.*, 7:452–459, 1986.
- [11] J.J. Modi and M.R.B. Clarke. An Alternative Givens Ordering. *Numer. Math.*, 43:83–90, 1984.

- [12] A. Pothan, J. Somesh, and U. Vemulapati. Orthogonal factorization on a distributed memory multiprocessor. In M.T. Heath, editor, *Proc. Hypercube Multiprocessors 1987*, pages 587–596, SIAM, Philadelphia, PA, 1987.
- [13] A.H. Sameh and D.J. Kuck. On stable parallel linear system solvers. *J. ACM*, 25:81–91, 1978.



## INTERNAL DISTRIBUTION

1-5.	E. Chu	32-36.	R. C. Ward
6.	J. B. Drake	37.	D. G. Wilson
7.	E. L. Frome	38.	A. Zucker
8.	G. A. Geist	39.	P. W. Dickson (Consultant)
9-13.	J. A. George	40.	G. H. Golub (Consultant)
14.	L. J. Gray	41.	R. M. Haralick (Consultant)
15-16.	R. F. Harbison	42.	D. Steiner (Consultant)
17.	M. T. Heath	43.	Central Research Library
18-22.	J. K. Ingersoll	44.	K-25 Plant Library
23-27.	F. C. Maienschein	45.	ORNL Patent Office
28.	T. J. Mitchell	46.	Y-12 Technical Library
29.	E. G. Ng		Document Reference Station
30.	G. Ostrouchov	47.	Laboratory Records - RC
31.	C. H. Romine	48-49.	Laboratory Records Department

## EXTERNAL DISTRIBUTION

50. Dr. Donald M. Austin, Office of Scientific Computing, Office of Energy Research, ER-7, Germantown Building, U.S. Department of Energy, Washington, DC 20545
51. Dr. Robert G. Babb, Department of Computer Science and Engineering, Oregon Graduate Center, 19600 N.W. Walker Road, Beaverton, OR 97006
52. Dr. Jesse L. Barlow, Department of Computer Science, Pennsylvania State University, University Park, PA 16802
53. Prof. Ake Bjorck, Department of Mathematics, Linkoping University, Linkoping 58183, Sweden
54. Dr. James C. Browne, Department of Computer Sciences, University of Texas, Austin, TX 78712
55. Dr. Bill L. Buzbee, C-3, Applications Support & Research, Los Alamos National Laboratory, P.O. Box 1663, Los Alamos, NM 87545
56. Dr. Donald A. Calahan, Department of Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI 48109
57. Dr. Tony Chan, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520
58. Dr. Jagdish Chandra, Army Research Office, P.O. Box 12211, Research Triangle Park, North Carolina 27709
59. Dr. Paul Concus, Mathematics and Computing, Lawrence Berkeley Laboratory, Berkeley, CA 94720
60. Dr. Jane K. Cullum, IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598

61. Dr. George Cybenko, Department of Computer Science, Tufts University, Medford, MA 02155
62. Dr. George J. Davis, Department of Mathematics, Georgia State University, Atlanta, GA 30303
63. Dr. Jack J. Dongarra, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
64. Dr. Stanley Eisenstat, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520
65. Dr. Howard C. Elman, Computer Science Department, University of Maryland, College Park, MD 20742
66. Dr. Albert M. Erisman, Boeing Computer Services, 565 Andover Park West, Tukwila, WA 98188
67. Dr. Geoffrey C. Fox, Booth Computing Center 158-79, California Institute of Technology, Pasadena, CA 91125
68. Dr. Paul O. Frederickson, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545
69. Dr. Fred N. Fritsch, L-300, Mathematics and Statistics Division, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550
70. Dr. Robert E. Funderlic, Department of Computer Science, North Carolina State University, Raleigh, NC 27650
71. Dr. Dennis B. Gannon, Computer Science Department, Indiana University, Bloomington, IN 47405
72. Dr. David M. Gay, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
73. Dr. C. William Gear, Computer Science Department, University of Illinois, Urbana, Illinois 61801
74. Dr. W. Morven Gentleman, Division of Electrical Engineering, National Research Council, Building M-50, Room 344, Montreal Road, Ottawa, Ontario, Canada k1A 0R8
75. Prof. Gene H. Golub, Department of Computer Science, Stanford University, Stanford, CA 94305
76. Dr. Joseph F. Grear, Division 8331, Sandia National Laboratories, Livermore, CA 94550
77. Dr. Don E. Heller, Physics and Computer Science Department, Shell Development Co., P.O. Box 481, Houston, TX 77001
78. Dr. Robert E. Huddleston, Computation Department, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550
79. Dr. Ilse Ipsen, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520
80. Dr. Harry Jordan, Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO 80309

81. Dr. Linda Kaufman, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
82. Dr. Robert J. Kee, Applied Mathematics Division 8331, Sandia National Laboratories, Livermore, CA 94550
83. Ms. Virginia Klema, Statistics Center, E40-131, MIT, Cambridge, MA 02139
84. Dr. Richard Lau, Office of Naval Research, 1030 E. Green Street, Pasadena, CA 91101
85. Dr. Alan J. Laub, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106
86. Dr. Robert L. Launer, Army Research Office, P.O. Box 12211, Research Triangle Park, North Carolina 27709
87. Prof. Peter D. Lax, Director, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012
88. Dr. Michael R. Leuze, Computer Science Department, Box 1679 Station B, Vanderbilt University, Nashville, TN 37235
89. Dr. Joseph Liu, Department of Computer Science, York University, 4700 Keele Street, Downsview, Ontario, Canada M3J 1P3
90. Dr. Franklin Luk, Electrical Engineering Department, Cornell University, Ithaca, NY 14853
91. Dr. Thomas A. Manteuffel, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545
92. Dr. Paul C. Messina, Applied Mathematics Division, Argonne National Laboratory, Argonne, IL 60439
93. Dr. Cleve Moler, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006
94. Dr. Dianne P. O'Leary, Computer Science Department, University of Maryland, College Park, MD 20742
95. Maj. C. E. Oliver, Office of the Chief Scientist, Air Force Weapons Laboratory, Kirtland Air Force Base, Albuquerque, NM 87115
96. Dr. James M. Ortega, Department of Applied Mathematics, University of Virginia, Charlottesville, VA 22903
97. Prof. Chris Paige, Basser Department of Computer Science, Madsen Building F09, University of Sydney, N.S.W., Sydney, Australia 2006
98. Dr. John F. Palmer, NCUBE Corporation, 915 E. LaVieve Lane, Tempe, AZ 85284
99. Prof. Beresford N. Parlett, Department of Mathematics, University of California, Berkeley, CA 94720
100. Prof. Merrell Patrick, Department of Computer Science, Duke University, Durham, NC 27706
101. Dr. Robert J. Plemmons, Departments of Mathematics and Computer Science, North Carolina State University, Raleigh, NC 27650

102. Dr. John K. Reid, CSS Division, Building 8.9, AERE Harwell, Didcot, Oxon, England OX11 0RA
103. Dr. John R. Rice, Computer Science Department, Purdue University, West Lafayette, IN 47907
104. Dr. Garry Rodrigue, Numerical Mathematics Group, Lawrence Livermore Laboratory, Livermore, CA 94550
105. Dr. Donald J. Rose, Department of Computer Science, Duke University, Durham, NC 27706
106. Dr. Ahmed H. Sameh, Computer Science Department, University of Illinois, Urbana, IL 61801
107. Dr. Michael Saunders, Systems Optimization Laboratory, Operations Research Department, Stanford University, Stanford, CA 94305
108. Dr. Robert Schreiber, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180
109. Dr. Martin H. Schultz, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520
110. Dr. David S. Scott, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006
111. Dr. Lawrence F. Shampine, Numerical Mathematics Division 5642, Sandia National Laboratories, P.O. Box 5800, Albuquerque, NM 87115
112. Dr. Danny C. Sorensen, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
113. Prof. G. W. Stewart, Computer Science Department, University of Maryland, College Park, MD 20742
114. Capt. John P. Thomas, Air Force Office of Scientific Research, Building 410, Bolling Air Force Base, Washington, DC 20332
115. Prof. Charles Van Loan, Department of Computer Science, Cornell University, Ithaca, NY 14853
116. Dr. Robert G. Voigt, ICASE, MS 132-C, NASA Langley Research Center, Hampton, VA 23665
117. Dr. Andrew B. White, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545
118. Mr. Patrick H. Worley, Computer Science Department, Stanford University, Stanford, CA 94305
119. Dr. Arthur Wouk, Army Research Office, P.O. Box 12211, Research Triangle Park, North Carolina 27709
120. Dr. Margaret Wright, Systems Optimization Laboratory, Operations Research Department, Stanford University, Stanford, CA 94305
121. Office of Assistant Manager for Energy Research and Development, Department of Energy, Oak Ridge Operations Office, Oak Ridge, TN 37830
- 122-152. Technical Information Center