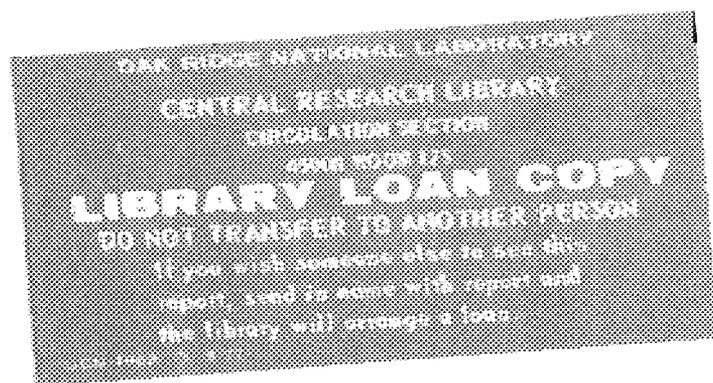


**OAK RIDGE
NATIONAL
LABORATORY**

MARTIN MARIETTA

**Finding Eigenvalues and Eigenvectors
of Unsymmetric Matrices Using
A Distributed-Memory Multiprocessor**

G. A. Geist
G. J. Davis



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.

Engineering Physics and Mathematics Division

Mathematical Sciences Section

**FINDING EIGENVALUES AND EIGENVECTORS
OF UNSYMMETRIC MATRICES USING
A DISTRIBUTED-MEMORY MULTIPROCESSOR**

G. A. Geist *

G. J. Davis **

* Oak Ridge National Laboratory
Mathematical Sciences Section
P.O. Box 2009, Bldg. 9207-A
Oak Ridge, TN 37831-8083

** Mathematics and Computer Science
Georgia State University
Atlanta, Georgia 30303

Date Published: November, 1988

Research was supported by the Applied Mathematical Sciences Research
Program 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-AC-05-84OR21400

MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES



3 4456 0283946 3

Contents

1 INTRODUCTION	1
2 PARALLEL HESSENBERG REDUCTION	2
3 FINDING EIGENVALUES	3
4 FINDING EIGENVECTORS	10
5 RESULTS	13
6 CONCLUSIONS	14
REFERENCES	15

**FINDING EIGENVALUES AND EIGENVECTORS
OF UNSYMMETRIC MATRICES USING
A DISTRIBUTED-MEMORY MULTIPROCESSOR**

G. A. Geist

G. J. Davis

Abstract

Distributed-memory parallel algorithms for finding the eigenvalues and eigenvectors of a dense unsymmetric matrix are given. While several parallel algorithms have been developed for symmetric matrices, little work has been done on the unsymmetric case. Our parallel implementation proceeds in three major steps: reduction of the original matrix to Hessenberg form, application of the implicit double-shift QR algorithm to compute the eigenvalues, and back transformations to compute the eigenvectors. Several modifications to our parallel QR algorithm, including ring communication, pipelining and delayed updating are discussed and compared. Results and timings are given.

1. INTRODUCTION

In the past few years several parallel algorithms have been developed for the solution of the symmetric eigenvalue problem, including bisection, multisection, and Cuppen's method [2,5]. Until recently, little has been published about developing algorithms for the more difficult unsymmetric case. Since the symmetric methods do not extend to the unsymmetric problem in any natural way, other algorithms must be developed. R. van de Geijn [8] has explored variants of the QR algorithm on dense matrices for an array of processors. Boley and Maier [1] have investigated the QR algorithm for $O(n)$ processors, where n is the order of the matrix, and they are developing hypercube implementations.

This paper describes a parallel implementation of an efficient serial algorithm on a hypercube multiprocessor. The most efficient serial algorithm known for finding all the eigenvalues and eigenvectors of a dense $n \times n$ matrix involves the following steps [9]. First, the original matrix, A , is reduced to upper Hessenberg form, H i.e. it is zero below the first subdiagonal. Having the matrix in Hessenberg form greatly reduces the amount of computation in the following steps. There are several methods for reducing A to upper Hessenberg form. The most popular methods use either orthogonal or elementary similarity transformations. The methods that use orthogonal similarity transformations have better stability properties, but they are slower by a factor of two than the methods that use elementary similarity transformations. Because instabilities using elementary similarity transformations with pivoting are rarely seen in practice, we chose to implement a parallel method that uses these transformations.

The second step of the serial algorithm is to apply implicit double-shift QR iterations to H until all the eigenvalues are found. This variation of QR iteration was originally proposed by Francis [3] and has the advantage of using only real arithmetic throughout the computation. Since complex arithmetic often requires more than twice as much work as real arithmetic, this variation saves time. Researchers have found that the iteration method converges in an average of two iterations, usually to the eigenvalue of smallest modulus remaining in the matrix.

In the third step, inverse iteration is performed using the known eigenvalues to find the eigenvectors of H . Serially, this step is similar to doing n triangular solutions (one for each eigenvector). Finally, we apply to these eigenvectors the inverses of the

transformations used to reduce A to H . The result is all the eigenvectors of the original matrix A .

In the following sections we will describe parallel implementations of each of the above steps. We assume the reader is familiar with the details of the operations required to annihilate individual matrix entries (see, e.g.,[9]), and we will therefore give only a high-level description of the parallel algorithms. We will also describe several enhancements that were incorporated into step 2. In the final sections we summarize the results obtained to date and comment on future research areas.

2. PARALLEL HESSENBERG REDUCTION

The use of elementary similarity transformations resembles Gaussian elimination with two important modifications: elements on the first subdiagonal are not eliminated, and both row and column operations are performed to preserve similarity. Nevertheless, much of our previous work on the parallel solution of linear systems applies. In particular LU factorization can be implemented quite efficiently with row or column storage on distributed-memory multiprocessors[4].

The need to preserve similarity affects the choice of storage scheme in Hessenberg reduction. In a column-mapped scheme, the search for the pivot and the formation of the multipliers must be done entirely on the processor that holds the pivot column. This serial phase can seriously degrade the performance of a parallel implementation unless this sequential thread is masked. If the pivot search is effectively masked, a column-oriented LU decomposition can be competitive with a row-oriented version. However, Hessenberg reduction requires that both row and column interchanges be done. This hampers attempts at the kind of masking used in LU factorization. Since the column oriented LU algorithm performs poorly without masking, we chose to implement the reduction to Hessenberg form using row storage.

The algorithm for Hessenberg reduction is presented in Figure 1, and is described below. In the following discussion, processors will be numbered from 0 to $p - 1$, and rows of the matrix will be numbered from 1 to n . The parallel Hessenberg reduction is implemented by initially having the host processor send rows of the matrix to the processors in a wrap mapping, i.e., row i of A is sent to processor $(i - 1) \bmod p$. Unless the matrix is diagonally dominant, this mapping is changed due to pivoting during the

```
for  $k = 1$  to  $(n - 2)$  do
  determine pivot row:
    scan my portion of column  $k$ , find local max
    determine global maximum and which processor has the pivot
  do column interchanges and update row permutation map
  if I have the pivot row
    broadcast pivot row to all other processors
  else
    await pivot row
  compute multipliers to eliminate my pieces of column  $k$ 
  apply transformation to my rows
  assemble row multipliers which I have into a local vector
  concatenate multipliers from all processors into one global vector
  apply inverse transformation to my pieces of the unreduced columns
```

Figure 1: Algorithm for Hessenberg reduction.

reduction phase. Thus, wrap mapping cannot be assumed as the algorithm proceeds.

The algorithm loops over the $n - 2$ unreduced columns of the matrix. At step k , all the processors search their portions of column k and send their local maxima up a spanning tree of the processor topology rooted at processor 0. Processor 0 determines the global maximum and fans this information back down the spanning tree. All processors perform an explicit column interchange and an implicit row interchange by updating a map vector. The map vector keeps track of which processors contain which rows. The pivot row is broadcast, and the processors apply row modifications to their respective parts of the matrix. Next, the processors again work together through the spanning tree to concatenate their elements of column k . This column is fanned back down the spanning tree, where all the processors use it to complete the column modification for this step.

3. FINDING EIGENVALUES

In Figure 2 we present an algorithm for finding the eigenvalues of the Hessenberg matrix. This algorithm has two major loops. The outer loop checks whether one real eigenvalue, a pair of complex conjugate eigenvalues, or no eigenvalues have been found. The outer loop also checks whether the problem can be deflated at this step. (Deflation

```
While(  $n > 2$  )
  If problem can be deflated
    Adjust problem size
  If root(s) found
    decrease  $n$ 
     $numits = 0$ 

  Inner Loop: one QR iteration
   $numits = numits + 1$ 
  If I have column 2
    Send column modification information to processor with column 1
  Else if I have column  $n - 1$  or  $n$ 
    Send shift information to processor with column 1
  Else if I have column 1
    Await shift and column modification information
  For  $k = 0$  to  $(n - 3)$  of currently active submatrix do
    If ( $k = 0$  and I have column 1) or (I have column  $k$ )
      Calculate Householder vector  $pqr$ 
      Broadcast vector  $pqr$ 
    Else
      Await vector  $pqr$ 
    Modify my parts of rows  $k + 1$ ,  $k + 2$  and  $k + 3$ 
    If I have column  $k + 1$ 
      Await columns  $k + 2$  and  $k + 3$ 
      Form vector  $v$  as linear combination of the three columns
      Send  $v$  to processors holding columns  $k + 2$  and  $k + 3$ 
      Modify column  $k + 1$ 
    Else if I have columns  $k + 2$  or  $k + 3$ 
      Send column to processor with column  $k + 1$ 
      Await  $v$ 
      Modify column  $k + 2$  or  $k + 3$ 
    End of iteration

  If  $numits > 30$  then print message and stop
End while
```

Figure 2: Synchronous Algorithm for Finding Eigenvalues.

is a process by which the original problem is split into two or more smaller problems.) The inner loop performs one implicit double-shift QR iteration on H .

All the operations in this algorithm are symmetrical so there is no advantage in storing H by rows or by columns. The amount of communication and the potential for parallelism are the same for either storage method. We have implemented this algorithm assuming H is stored by columns in a wrap mapping. We will see that this storage method will improve the efficiency of the third step, finding the eigenvectors, but it requires us to transpose the matrix H .

Eigenvalues are found when one of two things occurs. If $H_{n,n-1}$ has converged to zero, then $H_{n,n}$ is a single real eigenvalue. On the other hand, if $H_{n-1,n-2}$ has converged to zero, then this signals that a pair of complex conjugate eigenvalues has been found. In order to determine their values it is necessary for the two processors that contain elements of the isolated 2×2 submatrix to solve the quadratic equation.

Checking for possible deflation requires a large amount of communication, which is often wasted, since most of the time the problem does not deflate. The two main advantages of checking for deflation are that a significant amount of computational work can be avoided, and if the matrix is known to have split, separate shifts can be calculated for each part. We implemented the following test for deflation. The subdiagonal is searched in parallel for "zero" elements. If any elements are "small" relative to the diagonal entries around them, then they are taken as zero. If such a "zero" is found, this signifies that the matrix can be split into two smaller matrices. The parallel implementation of this test has each processor send its diagonal entries to its left and right neighbors in the wrap mapping. Each processor then compares the subdiagonals it holds with the appropriate diagonal entries and sends the highest column number that has a "zero" subdiagonal (sending a 1 if it has none) to the processor that holds the last column of the active matrix. The latter processor chooses the maximum of these values and broadcasts this information to all the other processors. All the processors assume the matrix has split at this column and adjust their work accordingly.

The implicit double shift QR iteration on a Hessenberg matrix can be regarded as a loop with the index k going from 0 to $n-2$. At each step k , except for the last, three consecutive rows and then three consecutive columns are modified. To understand our

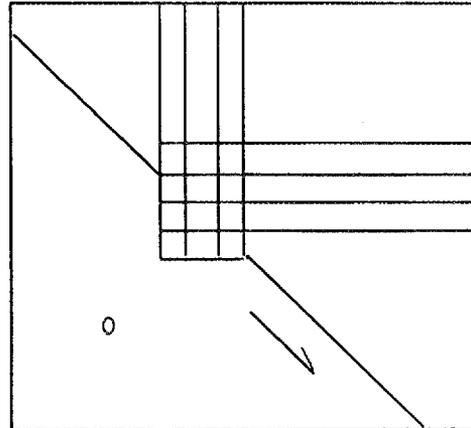


Figure 3: A bulge is chased down the diagonal during the QR iteration.

parallel implementation, it is important to understand how and when the operations are done. Figure 3 shows a snapshot of the matrix being modified during the iteration.

There are two communication phases at each step of the iteration. Before the row or column modifications can be performed, the Householder vector, which we call pqr , must be formed and broadcast to all the processors. Once the processors have pqr , they modify their parts of the three rows in parallel. The second communication phase occurs before the column modifications are performed. A linear combination of the three columns, v , must be formed. This vector is formed on the processor holding the leftmost of the three columns after it receives information from the other two processors. The formation of v is serial but, once formed, it is sent back out and used in parallel by the three processors to modify the three columns.

While finding the eigenvalues, about half of the computational work is performed in the column modifications and half in the row modifications. As described in the synchronous algorithm above, only three processors work on the column modifications at one time. By Amdahl's Law, if half the work in an algorithm is performed by only three processors, then the maximum speedup obtainable with that algorithm is about 6. This is unacceptable for distributed-memory architectures, which may have thousands of processors. Since the performance bottleneck is attributable to the serial nature of our initial implementation of the column modifications, we concentrated our efforts on developing implementations that were more parallel.

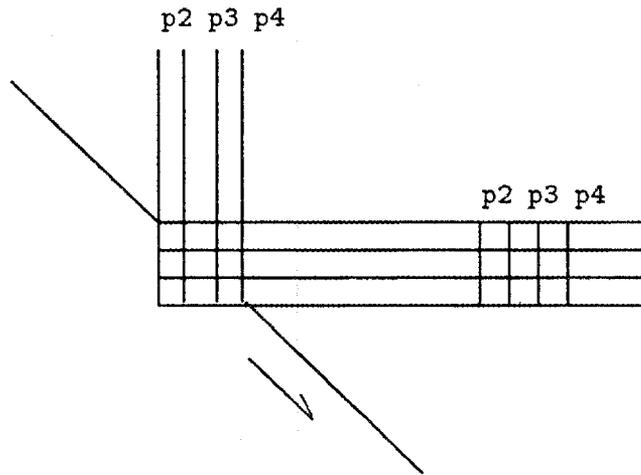


Figure 4: Pipelining requires knowing which processors contain active elements.

Several methods were employed in an attempt to mask the column modification phase of the algorithm. One of these methods involved using a ring architecture. In this method the roles of the three processors computing the column modification were switched. In a ring it makes more sense to let the middle processor compute v and send it back out, since this involves only nearest neighbor connections. Since we restricted ourselves to a strict ring architecture, the broadcast of pqr used a ring broadcast. In a ring broadcast a message is passed serially around the ring. While this method of broadcast requires $O(p)$ steps, it is often possible to mask this work completely by pipelining the operations around the ring.

We define pipelining to be a method in which a processor calculates and sends needed information before finishing the rest of its work for the current step. Thus, when the other processors finish their work the information they need is already available to them. Up to this point we have been describing synchronous algorithms. In synchronous algorithms computation and communication are parallelized, but they are not overlapped. In our case, either or both the row and column modifications can be pipelined. This pipelining can be applied to the original algorithm or the ring algorithm. To understand how this pipelining is implemented we will refer to Figure 4. In Figure 4, p_2 , p_3 , and p_4 are the processors that contain the three active columns of H at step k .

We first consider the pipelining of the column modifications. In the synchronous algorithm, p2 modifies all of column k , and the next row modification is started when p2 calculates and broadcasts new values for pqr . In the pipelined algorithm, p2 modifies the last three nonzero elements in column k , then calculates and broadcasts pqr before finishing modifications to the column. Since row modifications can start as soon as pqr is known, all processors except p2, p3, and p4 will begin the next step while these three processors finish the previous step. The effect is that now all processors are doing computations during part of the time when only three processors were busy in the original algorithm.

Next we consider pipelining the row modification. In the pipelined version, p2, p3, and p4 perform row modifications only on elements inside the leftmost 3×3 block in Figure 4, then complete column modifications (sending out pqr) before finally going back to finish the rest of their row modifications. Again, the idea is to have needed information available when processors finish the current stage. Pipelining the row modification masks the serial work of calculating v and modifying the last three elements in the pipelined column modification scheme.

Figure 5 presents the inner loop of a node algorithm for finding the eigenvalues of a Hessenberg matrix in which the row and column updates are pipelined.

There is still a problem even with these pipelining schemes. In the column modification step of our example, p2, p3, and p4 will finish the column modification while the other processors are doing row modifications. But now these processors will wait while p3 and p4 do the next row and column modification. It is not clear how to eliminate this problem, but it can be improved by using a scheme we call *delayed updating*. The basic idea is for key processors to delay calculating any values that are not immediately useful until later, when these processors would otherwise be idle.

Like pipelining, delayed updating can be applied to either the column modification or the row modification. During the k th column modification, elements with row indices less than $k - 2$ will not be needed until the next iteration is started. In Figure 4, since p3 and p4 must perform key calculations at this step, the updating of these column elements is delayed until step j , where $p \pmod{j} = 0$. In the above example, p2 is now in this position so it will modify the appropriate elements in its column with the last three modifications.

```
Inner Loop one QR iteration
  If I have column 2
    Send column modification information to processor with column 1
  Else if I have column  $n - 1$  or  $n$ 
    Send shift information to processor with column 1
  Else if I have column 1
    Await shift and column modification information
    Calculate householder vector  $pqr$ 
    Broadcast vector  $pqr$ 
  For  $k = 0$  to  $(n - 3)$  of currently active submatrix do
    If  $k \neq 0$  and I do not have column  $k$ 
      Await vector  $pqr$ 
    Modify the first three elements in rows  $k + 1$ ,  $k + 2$  and  $k + 3$ 
    If I have column  $k + 1$ 
      Await columns  $k + 2$  and  $k + 3$ 
      Form vector  $v$  as linear combination of the three columns
      Send  $v$  to processors holding columns  $k + 1$  and  $k + 2$ 
      Modify last three elements of column  $k + 1$ 
      Calculate  $pqr$  for the next value of  $k$ 
      Broadcast  $pqr$ 
      Finish row modification on rows  $k + 1$ ,  $k + 2$  and  $k + 3$ 
      Finish column modification on column  $k + 1$ 
    Else if I have columns  $k + 2$  or  $k + 3$ 
      Send column to processor with column  $k + 1$ 
      Finish row modification on rows  $k + 1$ ,  $k + 2$  and  $k + 3$ 
      Await  $v$ 
      Modify column  $k + 2$  or  $k + 3$ 
    Else
      Finish row modification on rows  $k + 1$ ,  $k + 2$  and  $k + 3$ 
  End of iteration
```

Figure 5: Inner loop of Pipelined Algorithm for Finding Eigenvalues.

Similarly, when several columns are contained on each processor, the row modifications can be delayed. In our example, p2, p3, and p4 must perform a row modification on their respective parts of the active columns, but not on any other columns they hold. (We are assuming that $p \geq 4$). After p2 has finished broadcasting the next pqr , it can finish the row modifications it has delayed. At subsequent steps, p3 and p4 will do the same.

The idea behind delayed updating in this algorithm is to allow the 3×4 block of computations to move down the diagonal as quickly as possible. The delayed work will tend to string out behind this block and allow more than three processors to be active during the column modifications. However, it is possible that delayed updating of the column modifications can hurt performance. Using the above scheme, the speed of the 3×4 block is increased at the expense of increasing the serial work of otherwise idle processors by a factor of three. Depending on the size of the problem, the number of processors, and the machine parameters, the result could be an increase in execution time.

4. FINDING EIGENVECTORS

When the QR iteration phase is complete, the Hessenberg matrix has been reduced to a quasi-upper triangular matrix, T , and the eigenvalues of A are known. The eigenvectors of T are found by inverse iteration. Although the parallel inverse iteration algorithm is similar to the column-oriented backsolve algorithm given in Romine and Ortega [6], the presence of the 2×2 blocks on the diagonal of T complicates the process. The algorithm is given in Figure 6.

For each eigenvalue, the algorithm calls a backsolve routine for a matrix of appropriate size. We illustrate this with the matrix T of order eight shown in Figure 7. The 2×2 blocks on the diagonal of T correspond to either a real or a complex conjugate pair of eigenvalues. Diagonal elements of T that are not in a 2×2 block correspond to real eigenvalues.

Proceed down the list of eigenvalues, beginning with $T(1, 1)$. This eigenvalue is real and its eigenvector is trivially $(1, 0, 0, 0, 0, 0, 0, 0)^T$. However, the next two eigenvalues of T are found in a 2×2 block. The backsolve algorithm must be called with the 3×3 principal submatrix of T , and this step will find two eigenvectors.

Main program:

```

For  $k = 1$  to  $n$  do
  Broadcast block structure corresponding to eigenvalue  $k$ 
  If column  $k$  is the first part of a  $2 \times 2$  block for eigenvalue  $k$ 
    Backsolve the  $(k + 1) \times (k + 1)$  submatrix for vectors  $k$  and  $(k + 1)$ 
  Else if column  $k$  is the second part of a  $2 \times 2$  block for eigenvalue  $k$ 
    Increment  $k$  (work is already done)
  Else
    Backsolve the  $k \times k$  submatrix for eigenvector  $k$ 

```

Backsolve algorithm for $k \times k$ submatrix:

```

For  $c = k$  to 1 do
  If I have column  $c - 1$  and it is first part of a  $2 \times 2$  block then
    Send multiplier information to processor with column  $c$ 
  Accumulate backsubstitution summation from all processors
  If I have column  $c$ 
    If column  $c$  is the second part of a  $2 \times 2$  block then
      Receive multiplier information from column  $c - 1$ 
    Compute entry  $c$  of the eigenvector

```

Figure 6: Backsolve Algorithm for Finding Eigenvectors of Quasi-Triangular Matrix.

$$T = \begin{pmatrix} \times & \times \\ & \times \\ & & \times & \times & \times & \times & \times & \times \\ & & & \times & \times & \times & \times & \times \\ & & & & \times & \times & \times & \times \\ & & & & & \times & \times & \times \\ & & & & & & \times & \times \\ & & & & & & & \times \end{pmatrix}$$

Figure 7: Quasi-triangular matrix of order eight.

Although the fourth eigenvalue is real, when the backsubstitution is performed on the 4×4 principal submatrix, additional communication is again required to pass through the 2×2 block.

During the QR iteration phase, the transformations that are applied to H are also accumulated in a two-dimensional array, Z . The eigenvectors of H are found by applying these transformations to the eigenvectors of T . Since Z is stored by columns, each processor contains n/p rows of Z^T and n/p columns of eigenvectors of T . Thus the parallel matrix-matrix multiplication can be performed efficiently and with perfect load balance.

To obtain the eigenvectors of A , we must finally apply the inverses of the transformations that reduced the original A to Hessenberg form. This algorithm is outlined in Figure 8.

```
For  $k = (n - 2)$  to 1 do
  Concatenate and broadcast multipliers for column  $k$ 
  Apply multipliers to my pieces of column  $k$ 
  Do column interchange to complete the similarity transformation
```

Figure 8: Algorithm for Transforming Eigenvectors of H to Eigenvectors of A .

The permutation information has been recorded during the Hessenberg reduction, and the multipliers have been stored in the lower triangle of A . Storage of the eigenvectors of H is by columns. If we were to calculate each of the eigenvectors of A one at a time using all the processors, then the calculation would be essentially sequential. Our implementation calculates the first component of all the eigenvectors, then the second component and so on, allowing a large degree of parallelism.

This phase begins with all the processors working together to concatenate a column of multipliers. Once each processor has received the assembled vector, it has all the information necessary to both calculate the first component of each eigenvector for which it is responsible, and to permute the respective rows of the eigenvectors in the same way that A was originally permuted. All processors work independently and in parallel during this phase. These three processes of concatenating the next column of multipliers, calculating the next component of the eigenvectors, and then permuting

the eigenvectors continue until all the components are found.

5. RESULTS

This section gives performance results for our implementation of parallel Hessenberg reduction, four versions of parallel eigenvalue algorithms, an implementation of a parallel eigenvalue/eigenvector code, and the final parallel eigenvector transformation routine on an Intel iPSC/2 hypercube. The square test matrices are dense with entries randomly distributed between [0,1]. Three matrix sizes and up to 32 processors were used in the experiments. Table 1 shows the execution time in seconds for each of the algorithms on various size problems. We use the same names as the analogous serial codes in EISPACK [7] to indicate the function of the various parallel codes.

p	n	ELMHES	HQR SYNCH	HQR PIPE	HQR RING	HQR DELAY	HQR2 SYNCH	ELMBAK
4	64	1.2	16	13	13	12	38	0.9
	128	7.1	88	70	69	68	213	5.2
8	64	1.0	17	13	13	11	41	0.7
	128	4.9	85	67	65	65	216	3.3
	256	28	418	346	342	352	1171	21
16	64	1.0	17	13	12	12	43	0.6
	128	3.6	87	66	65	65	223	2.4
	256	17	409	333	330	351	1168	13
32	64	1.1	17	13	13	12	45	0.6
	128	3.3	89	66	65	65	232	2.1
	256	13	410	329	326	354	1196	8.7

Table 1: Execution times in seconds for test problems.

The parallel reduction to Hessenberg form (ELMHES) is very efficient. The time required for the reduction is less than 10% of the total time to find the eigenvalues of the matrix. The execution time decreases as p increases, and efficiencies as high as 90% have been observed for large matrices.

The parallel algorithms labeled HQR in Table 1 are implicit double shift QR algorithms to find only the eigenvalues of a matrix. Several versions were developed because the original synchronous version performed poorly. The execution times of the synchronous code are displayed along with the times for a pipelined version, a pipelined version using only ring communication, and a delayed updating version. While there

was a significant improvement in going from the synchronous to a pipelined implementation, variations in the pipelining and delayed updating made surprisingly little improvement. Moreover, there is almost no decrease in execution time as p increases. The efficiencies of the improved versions of HQR are about 50% for $p = 8$ and grow slowly with n . Thus the performance of these parallel algorithms is still not satisfactory.

Finding the eigenvectors serially requires about the same amount of time as finding the eigenvalues serially. Thus, a parallel algorithm that also finds eigenvectors should require about twice as much time as HQR to execute. We expanded the synchronous code HQR to include eigenvector calculations and called it HQR2. Our timings reveal that HQR2 ran 2.5 to 3 times longer than HQR. After profiling HQR2, it was found that in the $n = 256$ case approximately 10% of the extra time is spent doing the n backsubstitutions, and 90% of the time is spent applying transformations to Z . Since HQR2 is a direct modification of the synchronous HQR code, the serial portions of HQR are being exaggerated in HQR2 by also applying the transformations to Z serially. It may be possible to mask much of the formation of Z in the pipelined codes, and a future project is to develop a pipelined version of HQR2.

The parallel algorithm for transforming the eigenvectors of H to those of A is called ELMBAK. The performance properties of ELMBAK are similar to those of ELMHES. It is not only highly efficient, but as p increases it consumes less than 1% of the total time for finding the eigenvalues and eigenvectors of a matrix.

6. CONCLUSIONS

In this paper we have presented a parallel approach to the solution of the eigenvalue/eigenvector problem for a dense, unsymmetric matrix. The reduction to Hessenberg form and the final transformation for the eigenvectors were direct extensions of previous work on linear systems. The performance of these routines is very much as we expected. First, the combined time for ELMHES and ELMBAK is a small percentage of the total work load. More importantly, for fixed problem size n , execution times decrease as the number of processors p increases.

After implementing the synchronous algorithm for eigenvalues (HQR), we discovered that the sequential thread of the 3×4 block proceeding down the diagonal of the matrix was the major bottleneck. Considerable time and effort were put into masking

this bottleneck. Although improvements were made, the timing results revealed the apparent independence of execution time with number of processors. The primary reason for this independence is due to the large communication to computation ratio of existing hypercubes. On the iPSC/2, sending a single message takes about 60 times as long as a single floating point operation. The modification of the elements in the 3×3 block requires communication with two other processors. While we have concentrated on improving the computational part of the sequential thread, the communication part still remains as the next likely target for parallelization. Other mappings of columns to processors (e.g. block mapping) could reduce the communication, but at the expense of more serial arithmetic. Various methods of improving the iteration step are under investigation.

7. References

- [1] D. Boley and R. Maier. A parallel QR algorithm for the nonsymmetric eigenvalue problem. July 1988. Talk at SIAM Annual Meeting, Minneapolis Minnesota.
- [2] J. J. Dongarra and D. C. Sorensen. A fully parallel algorithm for the symmetric eigenvalue problem. *SIAM J. Sci. Statist. Comput.*, 8:139–154, March 1987.
- [3] J. G. F. Francis. The QR transformation - Part 2. *The Computer Journal*, 4:332–345, 1961.
- [4] G. A. Geist and C. H. Romine. LU factorization algorithms on distributed-memory multiprocessor architectures. *SIAM J. Sci. Statist. Comput.*, 9:639–649, 1988.
- [5] I. C. F. Ipsen and E. R. Jessup. *Solving the Symmetric Tridiagonal Eigenvalue Problem on the Hypercube*. Tech. Rept. YALEU/DCS/RR-548, Dept. of Comp. Sci., Yale University, New Haven, CT, 1987.
- [6] C. H. Romine and J. M. Ortega. Parallel solution of triangular systems of equations. *Parallel Comput.*, 6:109–114, 1988.
- [7] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garabow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystem Routines - EISPACK Guide*. Springer, Heidelberg, 1974.

- [8] R. A. van de Geijn. Storage schemes for efficient parallel methods for solving linear eigenvalue problems. May 1988. Dept. of Computer Science, U. of Texas at Austin.
- [9] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, Oxford, 1965.

INTERNAL DISTRIBUTION

- | | | | |
|--------|-------------------|--------|-------------------------------------------------------|
| 1. | B. R. Appleton | 30. | P. H. Worley |
| 2. | J. B. Drake | 31. | A. Zucker |
| 3-7. | G. A. Geist | 32. | J. J. Dorning (Consultant) |
| 8-9. | R. F. Harbison | 33. | R. M. Haralick (Consultant) |
| 10. | M. T. Heath | 34. | Central Research Library |
| 11-15. | J. K. Ingersoll | 35. | ORNL Patent Office |
| 16. | M. R. Leuze | 36. | K-25 Plant Library |
| 17-21. | F. C. Maienschein | 37. | Y-12 Technical Library/
Document Reference Station |
| 22. | E. G. Ng | 38. | Laboratory Records - RC |
| 23. | G. Ostrouchov | 39-40. | Laboratory Records Department |
| 24. | C. H. Romine | | |
| 25-29. | R. C. Ward | | |

EXTERNAL DISTRIBUTION

41. Dr. Donald M. Austin, Office of Scientific Computing, Office of Energy Research, ER-7, Germantown Building, U.S. Department of Energy, Washington, DC 20545
42. Dr. Robert G. Babb, Department of Computer Science and Engineering, Oregon Graduate Center, 19600 N.W. Walker Road, Beaverton, OR 97006
43. Lawrence J. Baker, Exxon Production Research Company, P.O. Box 2189, Houston, TX 77252-2189
44. Dr. Jesse L. Barlow, Department of Computer Science, Pennsylvania State University, University Park, PA 16802
45. Dr. Chris Bischof, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
46. Prof. Ake Bjorck, Department of Mathematics, Linkoping University, Linkoping 58183, Sweden
47. Dr. James C. Browne, Department of Computer Sciences, University of Texas, Austin, TX 78712
48. Dr. Bill L. Buzbee, Scientific Computing Division, National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307
49. Dr. Donald A. Calahan, Department of Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI 48109
50. Dr. Tony Chan, Department of Mathematics, University of California, Los Angeles, 405 Hilgard Avenue, Los Angeles, CA 90024

51. Dr. Jagdish Chandra, Army Research Office, P.O. Box 12211, Research Office, P.O. Box 12211, Research Triangle Park, NC 27709
52. Dr. Eleanor Chu, Department of Computer Science, University of Waterloo, Ontario, Canada N2L 3G1
53. Prof. Tom Coleman, Department of Computer Science, Cornell University, Ithaca, NY 14853
54. Dr. Paul Concus, Mathematics and Computing, Lawrence Berkeley Laboratory, Berkeley, CA 94720
55. Prof. Andy Conn, Department of Combinatorics, and Optimization, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
56. Dr. Jane K. Cullum, IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598
57. Dr. George Cybenko, Computer Science Department, University of Illinois, Urbana, IL 61801
58. Dr. George J. Davis, Department of Mathematics, Georgia State University, Atlanta, GA 30303
59. Dr. Jack J. Dongarra, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
60. Dr. Iain Duff, CSS Division, Harwell Laboratory, Didcot, Oxon OX11 0RA, England
61. Prof. Pat Eberlein, Department of Computer Science, SUNY/Buffalo, Buffalo, NY 14260
62. Dr. Stanley Eisenstat, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520
63. Dr. Lars Elden, Department of Mathematics, Linkoping University, 58183 Linkoping, Sweden
64. Dr. Howard C. Elman, Computer Science Department, University of Maryland, College Park, MD 20742
65. Dr. Albert M. Erisman, Boeing Computer Services, 565 Andover Park West, Tukwila, WA 98188
66. Dr. Peter Fenyves, General Motors Research Laboratory, Department 15, GM Technical Center, Warren, MI 48090
67. Prof. David Fisher, Department of Mathematics, Harvey Mudd College, Claremont, CA 91711
68. Dr. Geoffrey C. Fox, Booth Computing Center 158-79, California Institute of Technology, Pasadena, CA 91125
69. Dr. Paul O. Frederickson, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545

70. Dr. Fred N. Fritsch, L-300, Mathematics and Statistics Division, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550
71. Dr. Robert E. Funderlic, Department of Computer Science, North Carolina State University, Raleigh, NC 27650
72. Dr. Dennis B. Gannon, Computer Science Department, Indiana University, Bloomington, IN 47405
73. Dr. David M. Gay, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
74. Dr. C. William Gear, Computer Science Department, University of Illinois, Urbana, IL 61801
75. Dr. W. Morven Gentleman, Division of Electrical Engineering, National Research Council, Building M-50, Room 344, Montreal Road, Ottawa, Ontario, Canada K1A 0R8
76. Dr. Alan George, Vice President, Academic and Provost, Needles Hall, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
77. Dr. John Gilbert, Xerox Palo Alto Research Center, 3333 Coyote Hill Road Palo Alto, CA 94304
78. Dr. Jacob D. Goldstein, The Analytic Sciences Corporation, 55 Walkers Brook Drive, Reading, MA 01867
79. Prof. Gene H. Golub, Department of Computer Science, Stanford University, Stanford, CA 94305
80. Dr. Joseph F. Grcar, Division 8331, Sandia National Laboratories, Livermore, CA 94550
81. Dr. Per Christian Hansen, Copenhagen University Observatory, @Ø@ster Voldgade 3, DK-1350 Copenhagen K, Denmark
82. Dr. Don E. Heller, Physics and Computer Science Department, Shell Development Co., P.O. Box 481, Houston, TX 77001
83. Dr. F. J. Helton, GA Technologies, P.O. Box 81608, San Diego, CA 92188
84. Dr. Charles J. Holland, Air Force Office of Scientific Research, Building 410, Bolling Air Force Base, Washington, DC 20332
85. Dr. Robert E. Huddleston, Computation Department, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550
86. Dr. Ilse Ipsen, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520
87. Ms. Elizabeth Jessup, Department of Computer Science, Yale University, P.O. Box 2158, Yale Station, New Haven, CT 06520
88. Prof. Barry Joe, Department of Computer Science, University of Alberta, Edmonton, Alberta, Canada T6G 2H1

89. Dr. Harry Jordan, Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO 80309
90. Dr. Bo Kagstrom, Institute of Information Processing, University of Umea, 5-901 87 Umea, Sweden
91. Dr. Hans Kaper, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
92. Dr. Linda Kaufman, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
93. Dr. Robert J. Kee, Applied Mathematics Division 8331, Sandia National Laboratories, Livermore, CA 94550
94. Ms. Virginia Klema, Statistics Center, E40-131, MIT, Cambridge, MA 02139
95. Dr. Richard Lau, Office of Naval Research, 1030 E.Green Street, Pasadena, CA 91101
96. Dr. Alan J. Laub, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106
97. Dr. Robert L. Launer, Army Research Office, P.O. Box 12211, Research Triangle Park, NC 27709
98. Dr. Charles Lawson, Applied Mathematics Group, Jet Propulsion Laboratory, California Institute of Technology, M/S 506-232, 4800 Oak Grove Drive, Pasadena, CA 91103
99. Prof. Peter D. Lax, Director, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012
100. Dr. John G. Lewis, Boeing Computer Services, P.O. Box 24346, M/S 7L-21, Seattle, WA 98124-0346
101. Dr. Heather M. Liddell, Director, Center for Parallel Computing, Department of Computer Science and Statistics, Queen Mary College, University of London, Mile End Road, London E1 4NS, England
102. Dr. Joseph Liu, Department of Computer Science, York University, 4700 Keele Street, Downsview, Ontario, Canada M3J 1P3
103. Dr. Franklin Luk, Electrical Engineering Department, Cornell University, Ithaca, NY 14853
104. James G. Malone, General Motors Research Laboratories, Warren, MI 48090-9055
105. Dr. Thomas A. Manteuffel, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545
106. Dr. Bernard McDonald, National Science Foundation, 1800 G Street, NW, Washington, DC 20550
107. Dr. Paul C. Messina, California Institute of Technology, Mail Code 158-79, Pasadena, CA 91125

108. Dr. Cleve Moler, Ardent Computers, 550 Del Ray Avenue, Sunnyvale, CA 94086
109. Dr. Brent Morris, National Security Agency, Ft. George G. Meade, MD 20755
110. Dr. Dianne P. O'Leary, Computer Science Department, University of Maryland, College Park, MD 20742
111. Maj. C. E. Oliver, Office of the Chief Scientist, Air Force Weapons Laboratory, Kirtland Air Force Base, Albuquerque, NM 87115
112. Dr. James M. Ortega, Department of Applied Mathematics, University of Virginia, Charlottesville, VA 22903
113. Prof. Chris Paige, Department of Computer Science, McGill University, 805 Sherbrooke Street W., Montreal, Quebec, Canada H3A 2K6
114. Dr. John F. Palmer, NCUBE Corporation, 915 E. LaVieve Lane, Tempe, AZ 85284
115. Prof. Roy P. Pargas, Department of Computer Science, Clemson University, Clemson, SC 29634-1906
116. Prof. Beresford N. Parlett, Department of Mathematics, University of California, Berkeley, CA 94720
117. Prof. Merrell Patrick, Department of Computer Science, Duke University, Durham, NC 27706
118. Dr. Robert J. Plemmons, Departments of Mathematics and Computer Science, North Carolina State University, Raleigh, NC 27650
119. Dr. Alex Pothen, Department of Computer Science, Pennsylvania State University, University Park, PA 16802
120. Dr. John K. Reid, CSS Division, Building 8.9, AERE Harwell, Didcot, Oxon, England OX11 0RA
121. Dr. John R. Rice, Computer Science Department, Purdue University, West Lafayette, IN 47907
122. Dr. Garry Rodrigue, Numerical Mathematics Group, Lawrence Livermore Laboratory, Livermore, CA 94550
123. Dr. Donald J. Rose, Department of Computer Science, Duke University, Durham, NC 27706
124. Dr. Ahmed H. Sameh, Computer Science Department, University of Illinois, Urbana, IL 61801
125. Dr. Michael Saunders, Systems Optimization Laboratory, Operations Research Department, Stanford University, Stanford, CA 94305
126. Dr. Robert Schreiber, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180
127. Dr. Martin H. Schultz, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520

128. Dr. David S. Scott, Intel Scientific Computers, 15201 N.W.Greenbrier Parkway, Beaverton, OR 97006
 129. Dr. Lawrence F. Shampine, Mathematics Department, Southern Methodist University, Dallas, TX 75275
 130. Dr. Kermit Sigmon, Department of Mathematics, University of Florida, Gainesville, FL 32611
 131. Dr. Danny C. Sorensen, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
 132. Prof. G. W. Stewart, Computer Science Department, University of Maryland, College Park, MD 20742
 133. Dr. Kosmo D. Tatalias, Atlantic Aerospace Electronics Corporation, 6404 Ivy Lane, Suite 300, Breenbelt, MD 20770-1406
 134. Prof. Charles Van Loan, Department of Computer Science, Cornell University, Ithaca, NY 14853
 135. Dr. Robert G. Voigt, ICASE, MS 132-C, NASA Langley Research Center, Hampton, VA 23665
 136. Dr. Andrew B. White, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545
 137. Dr. Arthur Wouk, Army Research Office, P.O. Box 12211, Research Triangle Park, NC 27709
 138. Dr. Margaret Wright, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
 139. Dr. A. Yeremin, Department of Numerical Mathematics of the USSR Academy of Sciences, Gorki Street 11, Moscow, 103905, USSR
 140. Office of Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Operations Office, P.O. Box 2001 Oak Ridge, TN 37831-8600
- 141-150. Office of Scientific & Technical Information, P.O. Box 62, Oak Ridge, TN 37831