New Complex Parallel Eigenvalue and Eigenvector Routines

Mark R. Fahey
Computational Migration Group
Computer Sciences Corporation

U.S. Army Engineer Research and Development Center
Major Shared Resource Center
Vicksburg, MS 39180

Two new codes for computing the eigenvalues and eigenvectors of a complex Hessenberg matrix are presented. The first computes the Schur decomposition of a complex Hessenberg matrix, while the second computes the eigenvectors of a complex upper triangular matrix. These subprograms have been developed to fill a void in the ScaLAPACK library. Now, the capability exists to compute the eigenvalues and eigenvectors of dense complex matrices using a parallel QR algorithm.

The parallel complex Schur decomposition routine was developed based on the parallel real Schur decomposition routine provided in ScaLAPACK. The real code was appropriately modified to make it work with complex arithmetic and implement a complex multiple bulge QR algorithm. This also required the development of new auxiliary routines that perform essential operations in the complex Schur decomposition, and that will provide additional linear algebra computation capability to the ScaLAPACK community.

A parallel eigenvector calculation routine was also developed. This routine can take the output from the parallel Schur decomposition subprogram and compute the eigenvectors for the original complex Hessenberg matrix.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming—*distributed programming, parallel programming*; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*eigenvalues and eigenvectors*

General Terms: Parallel programming

Additional Key Words and Phrases: Parallel QR algorithm, Schur decomposition, ScaLAPACK, complex matrices, eigenvalue, eigenvector

1. INTRODUCTION

In this work, two new parallel codes for computing the eigenvalues and eigenvectors of a complex Hessenberg matrix are discussed. The first computes the Schur decomposition of a complex Hessenberg matrix, while the second computes the left and/or right eigenvectors of a complex upper triangular matrix.

The conversion of the ScaLAPACK [Blackford et al. 1997] code `PDLAHQR` to a complex implementation is presented. `PDLAHQR` computes the Schur decomposition of a real matrix in parallel using a multiple bulge QR algorithm. Therefore, this new implementation, called `PZLAHQR`, computes the Schur decomposition of a complex matrix in parallel using a multiple bulge strategy. In addition, a parallel code was developed to compute the left and/or right eigenvectors of an upper triangular matrix. This code is denoted as `PZTREVC`. Since the output from `PZLAHQR` is

standard Schur form, PZLAHQR and PZTREVC can be used in conjunction with existing ScaLAPACK routines to compute eigenvalues and eigenvectors of a general complex matrix in parallel. The names of these new codes follow the ScaLAPACK convention.

The work by Henry, Watkins, and Dongarra [Henry et al. 1997] is the primary reference on the parallel nonsymmetric QR algorithm design. Their paper includes many details that will not be repeated here and should be used in addition to this work.

In Section 2, both the sequential single bulge and and multiple bulge QR algorithms are reviewed. In Section 3, highlights of the parallel nonsymmetric QR algorithm are presented from [Henry et al. 1997]. The conversion of the real parallel nonsymmetric QR algorithm to complex arithmetic is discussed in Section 4. In Section 5, a parallel complex eigenvector calculation routine is presented. Scalability results for PZLAHQR and PZTREVC are shown in Section 6. Concluding remarks are given in Section 7. Appendix A contains a list of all the codes discussed with a short description for each.

2. SEQUENTIAL QR ALGORITHM REVIEW

The implicit shifted QR algorithm has been a successful serial method for computing the Schur decomposition

$$H = QTQ^H$$

where H is a Hessenberg matrix, Q is an unitary matrix, and T is an upper triangular matrix. The initial matrix H is assumed to be Hessenberg for simplicity since the reduction to Hessenberg form is a well understood problem and has been shown to parallelize well [Berry et al. 1994; Dongarra and Van de Geijn 1992].

The Schur decomposition is computed by iteratively applying orthogonal similarity transformations to the Hessenberg matrix H until it becomes upper triangular. This process, known as the QR algorithm, performs QR iterations implicitly by chasing *bulges* down the subdiagonals of the upper Hessenberg matrix H [Golub and Van Loan 1996; Watkins 1991]. Each iteration begins by choosing shifts that accelerate convergence and using them to form a bulge. The bulge is then chased down the subdiagonals to complete the iteration.

2.1 Single Bulge

The Francis double implicit shifted QR algorithm has long been the standard serial version for real matrices. One step of the Francis QR algorithm is presented in Figure 1.

The Householder matrices in Figure 1 are unitary transforms of the form:

$$P_i = I - \tau v v^H \tag{1}$$

where $\tau \in \mathbb{C}$ and $1 \leq |\tau| \leq 2$ and $v \in \mathbb{C}^n$ [Lehoucq 1994].

The Francis QR step begins each iteration by choosing two shifts and using them to form a bulge of degree 2. The bulge is then chased from top to bottom to complete the iteration. The shifts are usually taken to be the eigenvalues of the 2×2 submatrix at the lower right; this is commonly referred to as the Wilkinson shift strategy. One could also use some larger number, say M, of shifts by computing

```
Francis QR Step
e = eig(H(n − 1 : n, n − 1 : n))
Let x = (H − e(1)I_n) ∗ (H − e(2)I_n)
Let P_0 ∈ C^{n×n} be a Householder matrix
      such that P_0 x is a multiplie of e_1
H ← P_0 H P_0
for i = 1, ..., n − 2
      Compute P_i so that P_i H has zero
            (i + 2, i) and (i + 3, i) entries
      Update H ← P_i H_i P_i
      Update Q ← Q P_i
endfor
```

Fig. 1. Sequential Single Bulge Francis QR Step

the eigenvalues of the lower right hand submatrix or order M. This leads to the multiple bulge QR algorithm discussed in the next section.

2.2 Multiple Bulge QR Algorithm

To chase multiple bulges simultaneously, the double-shift strategy is inadequate because the shifts for the next iteration cannot be calculated until the current bulge has been chased to the bottom of the matrix. A strategy proposed in [Bai and Demmel 1989] is to compute the eigenvalues of the lower $M \times M$ submatrix as the shifts. There are then enough shifts to chase $M/2$ bulges before new shifts are needed. In [Watkins and Elsner 1991], it was proved that this results in quadratic convergence like the double-shift QR algorithm. Each cycle of computing M shifts and chasing $M/2$ bulges is referred to as a *super-iteration*. One super-iteration of the general sequential multiple bulge algorithm is presented in Figure 2.

```
Multiple Bulge QR Super-Iteration
e = eig(H(n − m + 1 : n, n − m + 1 : n))
for k = 0, ..., n − 6 + 2m
    for j = m, m − 2, m − 4, ..., 2
        i = k − 2j + 4
        if i < 0 then P_i = I
        if i = 0
            Let x = (H − e(j − 1)I_n) ∗ (H − e(j)I_n)e_1
            Let P_i ∈ C^{n×n} be a Householder matrix
                  such that P_i x is a multiple of e_1
        if 1 ≤ i ≤ n − 2
            Compute P_i so that P_i H has zero (i + 2, i)
                  and (i + 3, i) entries
        if i > n − 2 then P_i = I
        H ← P_i H P_i, Q ← Q P_i
    endfor
endfor
```

Fig. 2. Sequential Multiple Bulge QR Super-Iteration

In Figure 2, the i index is similar in function to the i index in the algorithm shown in Figure 1. Since there are $M/2$ bulges, some start-up and wrap-up costs exist (see [Henry et al. 1997] for more details).

3. PARALLEL QR ALGORITHM

A parallel nonsymmetric QR algorithm for real matrices was implemented in the code PDLAHQR as part of ScaLAPACK. The algorithm used in PDLAHQR is similar to the LAPACK routine DLAHQR. However, unlike DLAHQR, instead of sending one double-shift through the largest unreduced submatrix, this algorithm sends multiple double-shifts. This allows all bulges to carry out up-to-date shifts and spaces them apart so that there can be parallelism across several processor row/columns. Another critical difference is that this algorithm applies multiple double-shifts in a block fashion, as opposed to DLAHQR, which applies one double-shift at a time. Note that the LAPACK code xHSEQR is a multiple shift, single bulge QR algorithm implementation.

This is the approach taken in [Henry et al. 1997] where M shifts are obtained from the lower $M \times M$ submatrix, where M is a fairly large even number (say 40), and used to form $S = M/2$ bulges of degree two and chase them one after the other down the subdiagonal in parallel.

Details of this approach are discussed by Henry, Watkins, and Dongarra [Henry et al. 1997]; their key observations pertaining to parallelization are as follows:

—The most critical difference between serial and parallel implementations of the QR algorithm is that the number of bulges must be chosen to keep the processors busy. The bulges must be separated by at least a block, and remain synchronized, to ensure that each row/column of processors remains busy. Usually the block size must be large; otherwise there will be too much boundary communication.

—The overall logic can be kept similar to the well-tested QR algorithm. The super-iteration can be implemented to complete before new shifts are determined and another super-iteration is begun. Information about the "current" unreduced submatrix must remain global to all nodes.

—The Householder transforms are of size 3, which means they are specified by sending 3 data items. The latency associated with sending many such small messages would be ruinous, so the information from several (e.g., 30) Householder transformations is bundled in each message.

—If many bulges are being chased simultaneously, there may be several bulges per row or column of processors. In that case, latency can be reduced further by combining the information from all bulges in a given row or column into a single message.

This concludes a brief review of a parallel multiple bulge QR algorithm and the solver PDLAHQR based on this approach. In what follows, a complex version of the multiple bulge nonsymmetric QR algorithm is developed.

4. CONVERSION OF PDLAHQR TO PZLAHQR

The original need to develop a parallel complex Schur decomposition routine arose during the migration of a researcher's code from a scalar to parallel platform. The

most time-consuming computational task in this code was computing the full eigendecomposition of a dense complex matrix. Without the time or resources to develop an entire parallel code from start to finish, the decision was made to convert the existing ScaLAPACK code PDLAHQR to a complex implementation. This would have the drawback of not developing a parallel code based on the multiple single-shift strategy usually associated with complex QR algorithms. Counterparts to the auxiliary codes developed with PDLAHQR had to be developed, as well as developing a serial complex double-shift QR algorithm counterpart ZLAHQR2 to the LAPACK code DLAHQR. LAPACK already has a routine named ZLAHQR that employs a single-shift strategy.

In the following subsections, new auxiliary routines needed in the development of PZLAHQR are discussed as well as PZLAHQR itself. For a full list of all codes associated with the routines presented here, see Appendix A.

4.1 Sequential Complex Double-Shift QR Algorithm

The first step in developing the parallel complex nonsymmetric Schur decomposition routine was to implement a serial complex double-shift QR algorithm (ZLAHQR2). This was done by converting the LAPACK single-shift routine ZLAHQR to a double-shift algorithm. This required minimal, but subtle changes.

Two of the more important modifications to implement this double-shift strategy are as follows:

(1) At the beginning of the QR step, if two consecutive small subdiagonals are found, update a subdiagonal element by the factor of $1 - \tau$ (see Equation 1) instead of waiting until the end of the QR step. This is similar to the real code, where the update factor is -1.

(2) Since the double-shift strategy only reduces the Hessenberg matrix to quasi-triangular form with 2×2 blocks on the diagonal, these blocks must be further reduced to triangular form. Then, if necessary, the reduction transformation (rotator matrix) must be applied to the appropriate matrix rows and columns.

The double-shift strategy produces eigenvalues that appear one at a time or in pairs. For real matrices, it is important that any QR algorithm implementation couples a complex eigenvalue with its complex conjugate, which is also an eigenvalue. In fact, most QR algorithm implementations take great care to produce complex conjugate eigenvalues whose real components are equal and whose imaginary components are equal in magnitude to full precision. However, for ZLAHQR2 and PZLAHQR to be presented later, each pair of eigenvalues it finds will not necessarily be a complex conjugate pair. In fact, even if a complex matrix has complex conjugate eigenvalues, this code does not pair them up and consequently does not ensure that the real components are identical or that the imaginary components have the same magnitude.

The LAPACK testing suite was used to experimentally test this new serial double-shift code. The results were then compared to those resulting from testing the LAPACK single-shift code ZLAHQR. The testing suite performs 12 tests on 21 different matrix types for the input parameters. Each of the 12 tests computes a residual value that should be of order one. For a threshhold of 10 or larger, both codes pass all tests. It is noteworthy that the computed residuals for a few tests, which

check orthogonality of the computed unitary matrices, are found to be larger than 5 for both codes. The development of this routine was necessary as a model for the parallel complex Schur decomposition code to be written. All the modifications necessary to transition the QR algorithm from a single-shift to a double-shift as well as those to move from a real to a complex double-shift would be utilized in the development of the new parallel code.

4.2 2×2 Schur Decomposition

In the previous section, it was noted that the 2×2 blocks that are formed along the diagonal must be further reduced to upper triangular form, i.e.,

$$\begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} \bar{w} & \bar{x} \\ 0 & \bar{z} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

A routine to compute this decomposition was developed and is denoted as `ZLANV2`. This routine is needed by `ZLAHQR2` and also by `PZLAHQR`, which is discussed below. For both codes, the rotator matrix (composed of cosine c and sine s values) is then applied to the corresponding rows and columns of the matrix where the reduced 2×2 block is located. For `ZLAHQR2`, one can use the LAPACK routine `ZROT` to perform this application. However, for `PZLAHQR`, there is no code to apply a rotator to distributed matirx. Therefore, the application of this rotator to a distributed matrix is addressed next.

4.3 Parallel Application of Rotator Matrix

The efficient application of a 2×2 rotator in parallel to two rows (or columns) of a distributed matrix requires some careful attention. The ScaLAPACK approach involves a 2-D block-cyclic distribution of the matrix over a 2-D process grid. The rows to be acted on by the rotator may reside on different process rows as well as being distributed column-wise. Although each processor will have the rotator matrix, each processor will not have all the necessary matrix information. To efficiently address this issue, a work array for every process is required. Each process, if needed, will receive a copy of the remote information it needs to update its locally owned row(s). Then each process can apply the rotator to its row(s) and work array in serial using `ZROT`. At this point, the matrix rows have been updated and no further communication is needed. This process has been implemented in the PBLAS-like routine `PZROT`. Since this code uses dynamic memory allocation for the work array, the routine is coded in C following the LAPACK and ScaLAPACK convention.

4.4 Parallel Complex QR Algorithm

With the aforementioned support routines and several auxiliary routines (listed in Appendix A) in place, all that remained was the implementation of `PZLAHQR` itself. To achieve this, the real code `PDLAHQR` was converted to a complex implementation. The two most important modifications were as follows:

(1) Similar to the sequential version, at the beginning of the QR step if two consecutive small subdiagonals are found, update a subdiagonal element by a factor of $1 - \tau$ (see Equation 1.) This is in comparison to `PDLAHQR` where the update

factor is always -1. In one particular location in the code, this meant introducing a coupled send and receive so that an up-to-date copy of τ can be used on the appropriate process.

(2) When 2×2 blocks are formed along the diagonal, they must be transformed to standard Schur form, which requires the use of ZLANV2. Then the corresponding rows and columns must be updated by PZROT.

As in PDLAHQR, PZLAHQR calls its serial counterpart ZLAHQR to compute eigenvalues of submatrices. The double-shift ZLAHQR2 or the LAPACK single-shift ZLAHQR subprogram may be used to do this task in PZLAHQR.

5. EIGENVECTOR CALCULATION

PZLAHQR computes the Schur form of a complex Hessenberg matrix. Although the eigenvalues are on the diagonal of the upper triangular matrix, the Schur decomposition does not produce the eigenvectors. Thus, additional calculations must be performed to compute the eigenvectors of the upper triangular matrix produced by PZLAHQR. The eigenvector matrix can then be postmultiplied by the unitary matrix obtained in the Schur decomposition to give the eigenvectors of the original Hessenberg matrix. Another parallel routine was developed, PZTREVC, that computes some or all left and/or right eigenvectors of a complex upper triangular matrix, i.e.,

$$Tx = \lambda x \qquad \text{and/or} \qquad y^H T = \lambda y^H$$

where T is upper triangular, x and y are right and left eigenvectors, respectively, and λ is an eigenvalue. Together, PZLAHQR and PZTREVC can be used to compute the eigenvalues and eigenvectors of a complex Hessenberg matrix. Since ScaLAPACK already has a routine to reduce a nonsymmetric complex matrix to Hessenberg form, one can now compute the eigendecomposition of a complex nonsymmetric matrix in parallel.

The eigenvalues of T are $t_{11}, t_{22}, \ldots, t_{nn}$, which are assumed to be distinct. To find a (right) eigenvector associated with t_{ii}, the homogeneous equation $(T - t_{ii}I)v = 0$ must be solved. With the partition

$$T - t_{ii}I = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \quad \text{and} \quad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

where $T_{11} \in \mathbb{C}^{i \times i}$ and $v_1 \in \mathbb{C}^i$, the equation $(T - t_{ii})v = 0$ becomes

$$\begin{aligned} T_{11}v_1 + T_{12}v_2 &= 0 \\ T_{22}v_2 &= 0 \end{aligned}$$

The submatrices T_{11} and T_{22} are upper triangular and T_{22} is nonsingular because its main diagonal entries are nonzero. Thus, the subvector v_2 must be 0. The equations then reduce to $T_{11}v_1 = 0$. Since the (i,i) entry of T_{11} is zero, this latest homogeneous equation can be written

$$\begin{bmatrix} \hat{T} & r \\ 0 \cdots 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{v} \\ w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

where $\hat{T}$ is the leading $(i-1, i-1)$ submatrix of T_{11} (and T), r is the leading $i-1$ subvector of column i of T_{11}, and $\hat{v}$ is the first $i-1$ elements of v. Let w be

any nonzero number, say 1, then back substitution can be used to solve for $\hat{v}$ and consequently yield an eigenvector. The eigenvector associated with t_{ii} is

$$v = \begin{bmatrix} -\hat{T}^{-1}r \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \end{matrix} i^{\text{th}} \text{ element}$$

The right eigenvectors are computed via backward substitution using the PBLAS. A similar technique is used to compute the left eigenvectors using forward substitution routines from the PBLAS.

Note that this algorithm may give inaccurate results if the eigenvectors are ill-conditioned. Furthermore, in the calculation of $-\hat{T}^{-1}r$, overflow may occur. This can be controlled by the use of scaling. The LAPACK developers implemented a routine named `ZLATRS` that controls the scaling of elements in the solution of a triangular linear system. `ZLATRS` estimates a bound for the solution vector. If the estimate is less than the overflow threshold,[1] then scaling is not necessary and the BLAS routine `ZTRSV` is used to compute the solution. Otherwise, an in-lined solver is used that checks for possible overflow, scaling the solution vector as necessary.

The LAPACK code `ZTREVC` uses `ZLATRS` to compute the left and/or right eigenvectors of complex upper triangular matrices. For `PZTREVC`, a corresponding parallel solver, `PZLATRS`, was implemented to control scaling in the computation of the eigenvectors.

6. NUMERICAL RESULTS

In this section, timings are presented to show the scalability of `PZLAHQR` and `PZTREVC`. The numerical tests were carried out on an SGI Origin 2000, IBM SP2, and IBM Power3 SMP at the U.S. Army Engineer Research and Development Center (ERDC) Major Shared Resource Center (MSRC). See Table 1 for more information on each machine.

Table 1. Parallel computing platforms at the ERDC MSRC used to test `PZLAHQR` and `PZTREVC`.

Machine	Processor Speed (MHz)	Mflops/s per processor	Processors Per node	Number Nodes	Peak Gflops/s
SGI Origin 2000	195	390	2	64	49.9
IBM SP2	135	540	1	255	137.7
IBM Power3 SMP	222	888	8[a]	64	454.6

[a]Current hardware limitation of a maximum of four MPI processes per node.

All routines are implemented in Fortran 77, except for `PZROT`, which is coded in C. All routines were compiled using optimization flags `-O3`, `-O3 -qarch=pwr2`

[1]The overflow threshold is one divided by the safe minimum as computed by the LAPACK routine `PDLAMCH`.

`-qtune=pwr2 -qmaxmem=-1`, and `-O3 -qarch=pwr3 -qtune=pwr3 -qmaxmem=-1` for the SGI Origin 2000, the IBM SP2, and the IBM Power3 SMP, respectively. For runs on the IBM Power3 SMP, the environment variable `MP_SHARED_MEMORY` was set to `YES`. All tests were run during normal operation hours in a nondedicated environment.

6.1 Fixed Problem Size Scalability of PZLAHQR

The timing results were performed using a modified version of the ScaLAPACK testing software for `PDLAHQR` to call `PZLAHQR` instead. All timing results are for the computation of the standard Schur form of a Hessenberg matrix with random entries.[2] A two dimensional block-cyclic data decomposition was used with a blocking factor of 100. Only square processor grids were used, but this is not a requirement. Any type of rectangular processor grid may be used. Note that the amount of data is $\mathcal{O}(N^2)$, and the run-time on a single processor is $\mathcal{O}(N^3)$.

In Tables 2, 3, and 4 the execution time in seconds for computing the complete Schur decomposition on an SGI Origin 2000, an IBM Power3 SMP, and an IBM SP2, respectively, is reported. The first line in the tables reports the timings for `ZHSEQR` from the LAPACK library. `ZHSEQR` is the sequential multiple single-shift QR algorithm for complex Hessenberg matrices. The remaining lines show the timings for `PZLAHQR` for various square processor grids.

Table 2. Execution time in seconds to compute a complex Schur decomposition on an SGI Origin 2000.

Execution time in seconds - SGI Origin 2000						
Proc. grid	Matrix Order					
mp × np	500	1000	1500	2000	2500	3000
1	64	521	1740			
1 × 1	66	552	1849			
2 × 2	23	155	568	1229	2417	4097
3 × 3	16	91	287	707	1278	2312
4 × 4	14	60	180	383	793	1123

Table 3. Execution time in seconds to compute a complex Schur decomposition on an IBM Power3 SMP.

Execution time in seconds - IBM Power3 SMP						
Proc. grid	Matrix Order					
mp × np	500	1000	1500	2000	2500	3000
1	40	321	1108	2568		
1 × 1	39	318	1106	2577		
2 × 2	14	106	340	764	1574	2514
3 × 3	9	58	198	428	813	1399
4 × 4	8	36	106	238	436	700

[2] See ScaLAPACK installation guide [Choi et al. 1995].

Table 4. Execution time in seconds to compute a complex Schur decomposition on an IBM SP2.

Execution time in seconds - IBM SP2						
Proc. grid	Matrix Order					
mp × np	500	1000	1500	2000	2500	3000
1	54	670	1500	3514		
1 × 1	54	448	1502	3471		
2 × 2	21	154	521	1387	2546	4279
3 × 3	13	83	269	632	1417	1957
4 × 4	12	52	154	339	862	1088

The results in Tables 2, 3, and 4 show that for fixed N, PZLAHQR scales well as the number of processors increases.

In addition, there are some timings that are missing from the tables for sequential runs. The missing data are due to memory limitations. The code written to test PZLAHQR creates one large array for which all matrix and vector storage is used. For the sequential cases, this array must be so large that it would require 64-bit addressing to access the entire array. The Basic Linear Algebra Communication Subprograms (BLACS) are built with MPI as the underlying communication interface on the machines tested. Only the SGI Origin 2000 currently supports 64-bit compilation of MPI programs. Thus, it would have only been possible to do the larger sequential tests on the Origin 2000 if the code had been recompiled with 64-bit addressing.

6.2 Scaled Problem Size Scalability of PZLAHQR

Next, the scalability of PZLAHQR is further investigated by calculating speedup and efficiency ratings based on the computed aggregate megaflop rate as the problem size and the number of processors increase. Assume that the flop count to compute the Schur decomposition is $18N^3$, where N is the order of the matrix [Blackford et al. 1997].

Let *efficiency* with respect to megaflop rate be defined as

$$E_F = \frac{M_p}{PM_s}$$

where M_p is the megaflop rate on P processors and M_s is the serial megaflop rate. The *speedup* with respect to megaflop rate

$$S_F = PE_F$$

is the factor by which execution time is reduced on P processors.

In Table 5, the time in seconds to compute the Schur decomposition for increasing larger problems and larger processor grids is shown. Notice that the efficiency ratings are much higher for the processor grids that are evenly divisible by four in Table 5. In fact, the megaflop rate per processor for the processor grids divisible by four stays approximately constant around 47. The IBM Power3 SMP currently has a maximum of 4 MPI processes per node, and intranode messages are communicated significantly faster than internode messages. For processor grids not evenly divisible by four, the batch system must be set to either use less MPI processes per node to balance the number of processes across nodes or to use four MPI processes per

Table 5. Speedups and efficiencies based on the megaflop rate for the IBM Power3 SMP.

Speedups and efficiencies for IBM Power3 SMP					
Proc. grid		IBM Power3 SMP			
mp × np	N	Time	Mflops/s	S_F	E_F
1×1	2000	2577	55.9	1.0	1.00
2×2	4000	5975	192.8	3.5	0.86
3×3	6000	13550	286.9	5.1	0.57
4×4	8000	12526	735.8	13.2	0.82
5×5	10000	23712	759.1	13.6	0.54
6×6	12000	18078	1720.4	30.8	0.85
8×8	16000	24471	3012.9	53.9	0.84

node with one node using less than four MPI processes. For example, with a 3×3 processor grid, an efficiency of 0.73 is obtained if the machine is set to use three MPI processes on three nodes instead of a four-four-one setup on three nodes that had an efficiency of 0.57 (as shown in Table 5). Thus, the variable efficiencies are a by-product of the hardware, not the code.

The data in Table 5 are also displayed in Figure 3 and clearly shows linear scalability for processor grids divisible by four.

6.3 Fixed Problem Size Scalability of PZTREVC

Similar to the PZLAHQR tests, PZTREVC was tested using a modified version of the ScaLAPACK testing software for PDLAHQR. All timing results are for the computation of left and right eigenvectors of an upper triangular matrix with random entries. A blocking factor of 100 was used for the 2D block-cyclic data decomposition. As before, only square processor test grids were used, but this is not necessary. In Table 6, the execution time in seconds for computing right and left eigenvectors on an SGI Origin 2000 are presented. The timings are shown for various processor grids.

Table 6. Execution time in seconds to compute the left and right eigenvectors of a complex upper triangular matrix using PZTREVC on an SGI Origin 2000.

Execution time in seconds			
Proc. grid	Matrix Order		
mp × np	500	1000	1500
1×1	5	27	
2×2	4	19	56
3×3	5	18	45
4×4	4	17	38

Larger problem sizes were tested, but these required the use of PZLATRS to control scaling (see section 5.) When solving the triangular system for matrices of order greater than 1500, some entries of the right-hand sides (eigenvectors) become very large and overflow. With PZLATRS, scaling was used to control overflow, but the timings were significantly longer than would have been observed if PZTRSV had been used to solve the system. Thus, timings are not reported for matrix orders larger than 1500 where use of PZLATRS is necessary.

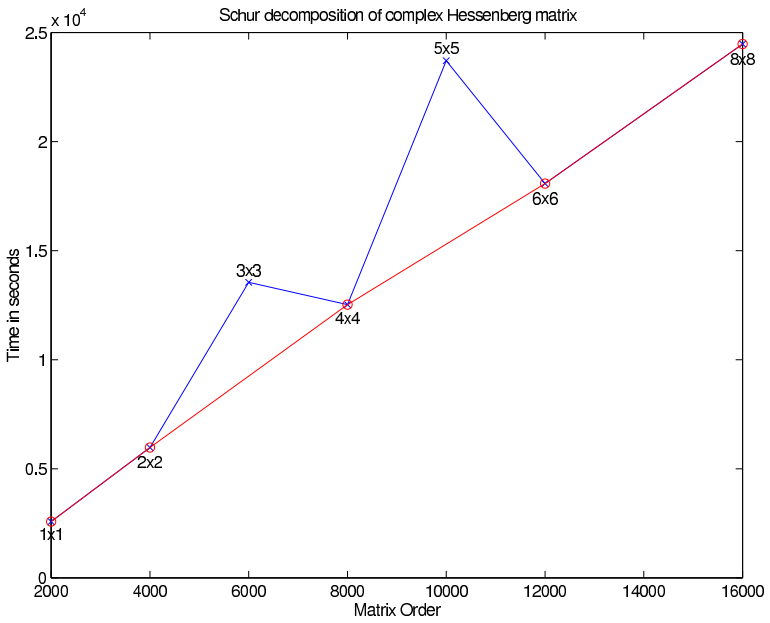

Fig. 3. Time in seconds to compute Schur decomposition of a complex Hessenberg matrix on an IBM Power3 SMP using increasing larger processor grids as the order of the matrix increases.

7. CONCLUDING REMARKS

A new parallel complex Schur decomposition routine PZLAHQR has been implemented based on the ScaLAPACK code PDLAHQR. Results were shown for PZLAHQR and showed that this routine scales nicely with the number of processors. Several auxiliary subroutines were developed to support this routine that will be useful outside the scope of PZLAHQR.

In addition, a parallel eigenvector calculation routine PZTREVC was developed for complex upper triangular matrices. Also, a new version of PZLATRS was developed that uses scaling to control potential overflow. However, PZLATRS requires further work to improve its scalability.

These codes are proposed additions to a future release of the ScaLAPACK library.[3]

[3]To obtain a copy of the codes, please send e-mail to Mark R. Fahey at mfahey@wes.hpc.mil.

ACKNOWLEDGMENTS

The author thanks Greg Henry for his advice early in this project. The author also thanks Dan Duffy and William Ward for their comments. Parallel runs were done on the SGI Origin 2000, IBM SP2, and IBM Power3 SMP at the U.S. Army Engineer Research and Development Center (ERDC) Major Shared Resource Center (MSRC). This work was supported in part by a grant of computer time from the DoD High Performance Computing Modernization Program at the ERDC MSRC, Vicksburg, MS.

APPENDIX

A. LIST OF CODES

A list of all the codes developed from this work is given below with short descriptions for each. Codes that begin with a `P` are parallel implementations. First, `PZLAHQR` and its auxiliary routines are listed, then `PZTREVC` and its auxiliary routines.

—`PZLAHQR` routine used to find the Schur decomposition and/or eigenvalues of a matrix already in Hessenberg form
  —`PZLACONSB` looks for two consecutive small subdiagonal elements checking the effect of starting a double shift QR iteration to see if this would make a subdiagonal negligible
  —`PZLASMSUB` looks for a small subdiagonal element from the bottom of the matrix that it can safely set to zero
  —`PZLAWIL` computes a transform given elements of Hessenberg matrix
  —`PZROT` applies a plane rotation
  —`ZLAHQR2` auxiliary routine to compute the Schur decomposition and/or eigenvalues of a matrix in Hessenberg form using a double-shift QR algorithm
  —`ZLAMSH` sends multiple shifts through a small matrix to see how consecutive small subdiagonal elements are modified by subsequent shifts in an effort to maximize the number of bulges
  —`ZLANV2` computes the Schur factorization of a complex 2×2 matrix
—`PZTREVC` computes some or all of the left and/or right eigenvectors of a complex upper triangular matrix
  —`PZLATRS` solves a triangular system with scaling to prevent overflow

REFERENCES

Bai, Z. and Demmel, D. 1989. On a block implementation of Hessenberg multishift QR iteration. *Int. J. High Speed Comput. 1*, 97–112. Also Argonne National Laboratory Technical Report ANL-MCS-TM-127, 1989.

Berry, M. W., Dongarra, J. J., and Kim, Y. 1994. A highly parallel algorithm for the reduction of a nonsymmetric matrix to block upper-Hessenberg form. Computer Sciences Dept. Technical Report CS-94-221 (Feb.), University of Tennessee, Knoxville, TN. LAPACK Working Note #68.

Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. 1997. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA.

Choi, J., Demmel, J., Dhillon, I., Dongarra, J., Ostrouchov, S., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. 1995. Installation guide for ScaLAPACK.

Computer Sciences Dept. Technical Report CS-95-280 (March), University of Tennessee, Knoxville, TN. LAPACK Working Note #93.

DONGARRA, J. J. AND VAN DE GEIJN, R. 1992. Reduction to condensed form on distributed memory architectures. *Parallel Computing 18*, 973–982.

GOLUB, G. H. AND VAN LOAN, C. F. 1996. *Matrix Computations* (Third ed.). The Johns Hopkins University Press, Baltimore, MD.

HENRY, G., WATKINS, D., AND DONGARRA, J. 1997. A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures. Computer Sciences Dept. Technical Report CS-97-352 (March), University of Tennessee, Knoxville, TN. LAPACK Working Note #121.

LEHOUCQ, R. 1994. The computation of elementary unitary matrices. Computer Sciences Dept. Technical Report CS-94-233, University of Tennessee, Knoxville, TN. LAPACK Working Note #72.

WATKINS, D. S. 1991. *Fundamentals of Matrix Computations*. John Wiley and Sons, New York, NY.

WATKINS, D. S. AND ELSNER, L. 1991. Convergence of algorithms of decomposition type for the eigenvalue problem. *Lin. Alg. Appl. 143*, 19–47.