[9] G. H. Golub and C. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 1983.

[10] C. C. Paige, The Gatlinburg X Meeting talk, October 1987.

[11] W. Kahan, private communication, April,1988.

[12] B. T. Smith, J. M. Boyle, Y. Ikebe, V. C. Klema and C. B. Moler, *Matrix Eigensystem Routines: EISPACK Guide*, 2nd ed. Springer-Verlag, New York, 1970.

[13] G. W. Stewart, *Introduction to Matrix Computations*, Academic Press, New York, 1973.

[14] G. W. Stewart, *A Parallel Implementation of the QR-algorithm*, Parallel Computing 5(1987), pp.187-196.

[15] D. Watkins, *On the GR Algorithms for the Eigenvalue Problem.* Presented at the NATO Advanced Study Institute, 1988.

9 Acknowledgements

The first author acknowledges the financial support of DARPA, grant F49620-87-C0065, and the second author the support of NSF, grant ASC-8715728. The second author is also a Presidential Young Investigator.

References

[1] D. Boley, *Solving the Generalized Eigenvalue Problem on a Synchronous Linear Processor Array*, Parallel Computing 3, pp.152-166,1986

[2] G. J. Davis, R. E. Funderlic and G. A. Geist, *A Hypercube Implementation of the Implicit Double Shift QR Algorithm*, Hypercube Multiprocessors'87, M. T. Heath ed. SIAM publisher, 1987.

[3] J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling and D. Sorensen, *Prospectus for the Development of a Linear Algebra Library for High-Performance Computers,* Argonne National Laboratory, ANL-MCS-TM-97, September, 1987.

[4] J. J. Dongarra, J. Du Croz, S. Hammarling and R. J. Hanson, *An Extended Set of the Fortran Basic Linear Algebra Subprograms.* ANL-MCS-TM 41, 1986.

[5] J. J. Dongarra, J. Du Croz, I. Duff and S. Hammarling, *A Proposal for a Set of Level 3 Basic Linear Algebra Subprograms*, ANL-MCS-TM 88,April 1987.

[6] J. J. Dongarra, S. J. Hammarling, and D. C. Sorensen, LAPACK Working Notes # 2:*Block Reduction to Condensed Forms for Eigenvalue Computations*, Mathematics and Computer Science Division, Argonne National Lab, Sep. 1987

[7] J. G. F. Francis, *The QR Transformation - A Unitary Analogue to the LR Transformation*, Computer J. 4(1961/1962), pp.265-271 and 332-345.

[8] R. A. van de Geijn, *Implementation the QR-Algorithm on an Array of Processors*, Univ. of Maryland, Dept. of Computer Science Technical Report CS-TR-1897, August 1987.

sweeps are necessary. The Eispack QR routine *hqr* takes about $2n$ sweeps to find all eigenvalues, but *shseqr* takes only about $\frac{1}{4}n$ sweeps. We observe empirically that in *shseqr*, after several QR sweeps, a small subdiagonal element appears near the position $n-k$, i.e. a submatrix of approximate size $k \times k$ is split out, so we can find its eigenvalues directly by calling Eispack QR algorithm. For example, for a 300×300 [-1, 1] uniformly distributed matrix, using *shseqr* with $k = 26$ and $p_2 = 4$, the number of QR sweeps and corresponded size of the deflated submatrix blocks are shown in Table 5:

iter	31	2	1	1	5	4	4	5	4	4	3	3	4
block	26	22	3	10	26	26	24	26	26	25	26	25	35

Table 5

This is what we expected. More tests and analysis will be needed to see whether this is true in more general cases. We still need to analyze the asymptotic convergence rate of the algorithm.

Remark 4: If we can consistently deflate k by k subblocks as suggested in Remark 3, this could be used to design a parallel divide and conquer scheme.

Remark 5: The *shseqr* with scheme S4 (cf. Section 3) took about double the time of *shseqr* with shift scheme S1.

8 Future Work

This is the first report of work in progress. Future work will include optimizing our *shseqr* codes, numerical tests on CRAY and Alliant FX/8 machines, and finding optimal parameters k, p_1 and p_2 for different size matrices and specific machine architectures. The asymptotic convergence rate as a function of the multishift number k will be discussed in detail (see [Watkins'88]), The final version of this code will be a part of the LAPACK linear algebra library (see [J. Demmel *et al*' 87]).

Figure 1: The timing cost depending on k, $p_2 = 4$

Figure 2: The timing cost depending on p_2, $k = 14$

n	matrices	*hqr* timing	*shseqr*(k, p_2) timing	speedup
200	[0,1]	15.63	9.21(18,6)	1.70
200	[-1,1]	17.23	7.25(18,7)	2.38
200	normal	17.62	8.78(18,6)	2.01
256	[0,1]	29.81	17.50(12,6)	1.70
256	normal	34.75	14.28(14,6)	2.43
300	[0,1]	45.11	23.94(12,6)	1.88
300	[-1,1]	45.26	23.26(12,6)	1.94
400	[-1,1]	132.31	50.71(14,6)	2.61
400	normal	127.46	51.53(14,6)	2.47

Table 4. QR iteration timing on Cyber 205

¿From our preliminary numerical tests, we make the following remarks. Much analysis remains to be done.

Remark 1: In our test examples, the *shseqr* produces the same eigenvalues as the EISPACK codes to at least ten decimal places even for very ill-conditioned eigenproblems. *Shseqr* is, of course, backward stable.

Remark 2: In table 1, the parameters k and p_1 and p_2 are chosen by local optimization. For example, let matrix A be 128×128 having entries chosen independently from the uniform distribution on [0,1]. Using *shseqr*, we first fix parameter p_2 and vary k to find the k minimizing the running time (Figure 1). Then for this locally optimal k, we vary p_2 to minimize the running time (Figure 2). How to characterize the optimal parameters k and p_2 is still not very clear. We expect these parameters to be machine-dependent, depending on the number of vector registers, cache size etc.

Remark 3: By 'iter', we mean the total number of QR sweeps to find all eigenvalues. In general, it is seen that the more shifts used, the fewer QR

n	matrices	*orthes* timing	*shseqr*(p_1) timing	speedup
200	[0,1]	4.81	0.58(12)	8.29
200	[-1,1]	4.80	0.58(12)	8.28
200	normal	4.80	0.59(12)	8.14
256	[0,1]	10.50	1.01(8)	10.40
256	normal	10.50	0.99(8)	10.60
300	[0,1]	17.21	1.46(12)	11.79
300	[-1,1]	17.19	1.44(12)	11.93
400	[-1,1]	42.07	2.92(12)	14.41
400	normal	42.65	2.88(12)	14.81

Table 3. Reduction timing on Cyber 205

n	matrices	*hqr* iter.	timing (sec)	*shseqr* iter.	timing(k, p_2) (sec)	speedup
200	[0,1]	342	15.65	44	13.02(18,6)	1.20
200	[-1,1]	372	15.83	40	11.90(18,7)	1.33
200	normal	386	17.03	50	15.78(18,6)	1.20
256	[0,1]	478	30.73	87	26.34(12,6)	1.17
256	normal	481	31.47	70	21.45(14,6)	1.47
300	[0,1]	559	47.15	106	38.53(12,6)	1.22
300	[-1,1]	551	46.22	100	35.15(12,6)	1.31
300	normal	573	46.67	104	36.54(12,6)	1.28
400	[-1,1]	702	96.65	122	69.68(14,6)	1.39
400	normal	711	97.27	127	75.15(14,6)	1.29

Table 2. QR iteration timing on CONVEX C-1

The timing costs on CYBER 205 in 32-bit arithmetic of reduction to upper Hessenberg form and QR iteration are listed in Tables 3 and 4 respectively. The last column of the tables are the speedups. We beat EISPACK by a factor 8-14 in reduction to Hessenberg form and a factor 1.7-2.6 in QR iteration.

trices (each matrix entry chosen independently as follows):

normal : standard normal distribution with mean 0, variance 1
$.[-1,1]$: uniform distribution on $[-1,1]$
$.[0,1]$: uniform distribution on $[0,1]$

The sizes of the test matrices range from 100 up to 400. The timing in seconds for finding all eigenvalues and no eigenvectors of full real matrices by EISPACK *orthes* and *hqr*, and *sgehrd*(p_1) and *shseqr*(k,p_2) in 64 bit precision on the CONVEX are listed in Table 1 and 2, where the EISPACK codes are also optimized using the $-O2$ compiler option. We see from Table 2 that the block multishift QR algorithm is about 20% to 45% faster than the EISPACK codes. We expect that *shseqr* with more careful coding would be better. Evidence for this is shown in that the *shseqr* with optimal choices of the parameters k, p_1 and p_2 is 2 to 4 times faster than *shseqr* with $k = 2$, $p_1 = p_2 = 1$, which should be equivalent to the "standard QR".

n	matrices	*orthes* timing	*sgehrd*(p_1) timing	speedup
200	[0,1]	7.30	2.83(12)	2.58
200	[-1,1]	6.93	3.03(12)	2.29
200	normal	7.53	2.85(12)	2.54
256	[0,1]	14.35	5.90(8)	2.43
256	normal	14.90	5.55(8)	2.68
300	[0,1]	22.45	8.87(12)	2.53
300	[-1,1]	23.00	8.78(12)	2.62
300	normal	21.95	8.58(12)	2.56
400	[-1,1]	52.13	19.75(12)	2.64
400	normal	50.85	19.88(12)	2.56

Table 1. Reduction timing on CONVEX C-1

7 Numerical Tests

Numerical tests of the block multishift QR iteration were carried out on the CONVEX C-1 computer at Courant Institute, New York University and CYBER 205 at John von Neumann National Supercomputer Center. The CONVEX has a vector architecture with register to register operations and pipelined functional units, and has a cycle time of 100 ns which results in a theoretical peak performance of 10 MFLOPS for simple operations and 20 MFLOPS for compound add/multiply operations assuming 64-bit arithmetic. The memory is managed on a fixed-size page basis. There is a FORTRAN vectorizing compiler. CONVEX rates their machine as 1/6 of the CRAY 1-S in speed.

The Control Data Corporation CYBER 205 is a vector computer like the Cray-1, but does not contain vector registers. Hence any data to be processed is transferred directly from memory to the designated vector functional unit and back to memory. The cycle time of the CYBER 205 is 20 nsec. Vector units may run in parallel under certain circumstances.

All the codes of block implementation of reduction to upper Hessenberg form (in short: *sgehrd*) and multishift QR iteration (in short: *shseqr*) for Hessenberg form are written in standard Fortran 77, with as many matrix-vector(Level 2 BLAS) and matrix-matrix(Level 3 BLAS) operations as possible in order to exploit the memory hierarchy.

For the test results reported on the CONVEX in this paper we used VECLIB, a collection of FORTRAN-callable subprograms providing basic mathematical software including the BLAS. We use the $-O2$ option in the CONVEX FORTRAN compiler fc to perform machine-independent local and global optimizations plus vectorization.

On CYBER 205, we have so far programmed all the BLAS codes ourselves in Fortran, and so expect future performance improvements.

The parameters k, p_1 and p_2 denote the number of shifts in each QR sweep and block sizes for Hessenberg reduction and bulge chasing, (see [Dongarra *et al.*'87] for details on the Hessenberg reduction algorithm). As stated in section 3, the k shifts are chosen as the k eigenvalues of the $k \times k$ trailing principle submatrix. So for $k = 2$ and $p_1 = p_2 = 1$, our algorithm can be regarded as the "standard" implicit double shift QR iteration.

Our preliminary experiments are based on the following classes of ma-

Counting the floating-point operations reveals that if we chase the k by k bulge one column at a time ($p = 1$), one sweep costs $2kn^2$ flops. k sweeps of the usual single shift QR or $k/2$ sweeps of the usual double shift QR algorithm also cost approximately $2kn^2$ flops. In aggregating transformations to perform the block chasing, additional work is required to form y_j and z_j. The additional work amounts to:

$$(k + \frac{1}{2}p)n^2 + O(n).$$

Note that we use matrix-vector operations (Level BLAS 2) in the j loop for computing y_j and z_j, and matrix-matrix operations (Level 3 BLAS) for the l loop updating. Thus, the new algorithm must have an execution rate at least $\frac{3}{2} + \frac{p}{4k}$ times as great as the standard algorithm in order to have a speed up (assuming approximately equal convergence rates).

6 Convergence Criterion

We recall that in the standard QR algorithm, the convergence test first looks for a negligible subdiagonal element to set to zero and deflate a submatrix (called deflation technique I), and then looks for two small consecutive subdiagonal elements whose product is negligible (called deflation technique II). The QR iteration then works on the smaller submatrices. The approximate eigenvalues are computed one by one for real eigenvalues, or pair by pair for complex conjugate eigenvalues.

The motivation of multishift QR iteration is to deflate several eigenvalues simultaneously, i.e. to find a negligible subdiagonal element near subdiagonal $n-k$ rather than $n-1$ or $n-2$ as in standard QR iteration. If a deflated submatrix has dimension smaller than some n_0 (which depends on k), we will simply use standard QR (*hqr* from EISPACK) to compute its eigenvalues. Thus the algorithm is a hybrid of the standard and block multishift QR algorithms.

Experience with MATLAB indicates that deflation technique II introduces extra flops and data movement exceeding the benefit of the faster convergence, so we have chosen not to implement it in our code (although it is retained in *hqr*, which our code calls).

composed of the ith to jth columns of matrix X_k is denoted by $X_k^{(i:j)}$, and $x_k^{(i:j)}$ denotes a sub-vector composed by the ith to jth components of vector x_k.

Algorithm 1(*Bulge Chasing*). Let A be an upper Hessenberg matrix with a k by k bulge. This algorithm chases the bulge to reduce A to upper Hessenberg form. In particular, to reduce a full real general matrix to upper Hessenberg form, choose $k = n - 2$. p is the block size.

$N = (n-2)/p$; N is the number of the blocks.
for $l = 1, .., N$; outer loop.
  $s = (l-1)p + 1$; s is the first column index in each block.
  for $j = s, ..., s+p-1$; inner loop.
    $q = s + \mathrm{mod}(j-1, p) + 1$;
    $kq = q + k - 1$;
    $b_j := b_j^{(q:kq)} - \sum_{i=s}^{j-1} (v_j^{(i)} u_i^{(q:kq)} + u_j^{(i)} w_i^{(q:kq)})$;
    compute $\hat{u}_j$ such that $(I - \hat{u}_j \hat{u}_j^{\mathrm{T}}) b_j = \sigma_j e_1$;
    let $u_j = (0, ..., 0, \hat{u}_j, 0, ..., 0)^{\mathrm{T}}$; where $\hat{u}_j$ occupies the qth to kqth components of u_j.
    $y_j = (B - U_l^{(s:j-1)} V_l^{(s:j-1)^{\mathrm{T}}} - W_l^{(s:j-1)} U_l^{(s:j-1)^{\mathrm{T}}})^{\mathrm{T}} u_j$;
    $z_j = (B - U_l^{(s:j-1)} V_l^{(s:j-1)^{\mathrm{T}}} - W_l^{(s:j-1)} U_l^{(s:j-1)^{\mathrm{T}}}) u_j$;
    $v_j = y_j - \frac{1}{2}(z_j^{\mathrm{T}} u_j) u_j$;
    $w_j = z_j - \frac{1}{2}(y_j^{\mathrm{T}} u_j) u_j$;
    $U_l^{(j)} = u_j$;
    $V_l^{(j)} = v_j$;
    $W_l^{(j)} = w_j$;
  end of j loop
  $B = B - U_l V_l^{\mathrm{T}} - W_l U_l^{\mathrm{T}}$;
end of l loop

Note that by choosing $k = n - 2$, the above algorithm reduces a dense matrix to upper Hessenberg form. Thus, QR iteration and reduction to Hessenberg form can be thought of as special cases of the same general algorithm.

Thus restructuring QR to chase a k by k bulge p columns at a time lets us use Level 3 BLAS in the innermost loop of the algorithm. The aggregation idea was proposed in [Dongarra *et al.*'87], which showed how to reduce a full matrix to upper Hessenberg form.

In the process, B is repeatly modified. At each stage, the matrix is updated by a rank-2 matrix, and only $k+2$ rows and $k+2$ columns of the matrix are changed. To achieve better memory utilization, we can aggregate a sequence of transformations, say p of them, so that the matrix is updated by a rank $2p$ matrix, and so $(p+k+1)$ rows and $(p+k+1)$ columns of the matrix will be updated simultaneously. This works as follows: Let us start from $B_1 = B$ for simplicity. Instead of explicitly updating the matrix with a rank two change, we only form the second column of B_2, say $\hat{b}_2$, i.e.

$$\hat{b}_2 := b_2 - v_1^{(2)} u_1 - u_1^{(2)} w_1,$$

where $v_1^{(2)}$, $u_1^{(2)}$ are the second elements of the vector v_1 and u_1. From $\hat{b}_2$ we can compute u_2, and construct y_2 and z_2 as follows:

$$\begin{aligned} y_2 &= (B_1 - u_1 v_1^{\mathrm{T}} - w_1 u_1^{\mathrm{T}}) u_2, \\ z_2 &= (B_1 - u_1 v_1^{\mathrm{T}} - w_1 u_1^{\mathrm{T}}) u_2. \end{aligned}$$

We then explicitly form B_3 as follows:

$$\begin{aligned} B_3 &= B_1 - u_1 v_1^{\mathrm{T}} - w_1 u_1^{\mathrm{T}} - u_2 v_2^{\mathrm{T}} - w_2 u_2^{\mathrm{T}} \\ &= B_1 - (u_1, u_2)(v_1, v_2)^{\mathrm{T}} - (w_1, w_2)(u_1, u_2)^{\mathrm{T}} \\ &= B_1 - U_2 V_2^{\mathrm{T}} - W_2 U_2^{\mathrm{T}}. \end{aligned}$$

We can continue the process and in general find for a rank-$2p$ updating:

$$B_{p+1} = B_1 - U_p V_p^{\mathrm{T}} - W_p U_p^{\mathrm{T}}.$$

e.g. starting from the matrix in (1) with $p = 2$, after the first block updating the matrix looks like

$$\begin{pmatrix} \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times & \times & \times & \times \\ & & \underline{\times} & \times & \times & \times & \times & \times & \times \\ & & \underline{\times} & \underline{\times} & \times & \times & \times & \times & \times \\ & & \underline{\times} & \underline{\times} & \underline{\times} & \times & \times & \times & \times \\ & & & & & & \times & \times & \times \\ & & & & & & & \times & \times \end{pmatrix}.$$

These considerations may be summed up in the following algorithm, where the jth column of a matrix X_k is denoted by x_j or $X_k^{(j)}$, a submatrix

transformations to chase a k by k bulge column by column. We will see that this leads to an algorithm with matrix-vector (Level 2 BLAS) operations in the inner loop. But since our purpose in developing the multishift QR algorithm is to make the computation rich in matrix-matrix operations, we can do block chasing. The idea is to partition the matrix by columns into n/p blocks:

$$B = (\bar{B}_1, \bar{B}_2, \ldots, \bar{B}_{n/p})$$

where each $\bar{B}_i$ has roughly p columns, and then chase the bulge block by block, p columns at a time. At each block, we aggregate the Householder transformations and apply them in a blocked fashion.

The process with $p = 1$ can be described as follows: At the ith step, the bulge has been chased to ith column. e.g. for $n = 9$, at the 3th step, the matrix is of the form

$$\begin{pmatrix} \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times & \times & \times & \times \\ & & \underline{\times} & \times & \times & \times & \times & \times & \times \\ & & \underline{\times} & \underline{\times} & \times & \times & \times & \times & \times \\ & & \underline{\times} & \underline{\times} & \underline{\times} & \times & \times & \times & \times \\ & & & & & & \times & \times & \times \\ & & & & & & & \times & \times \end{pmatrix}.$$

¿From the ith column of the reduced matrix B_i, we can construct the Householder transformation $P_i = I - u_i u_i^{\mathrm{T}}$, ($\| u_i \| = \sqrt{2}$) to "chase" the bulge from the ith column to the $(i+1)$-st column. Applying P_i to the matrix B_i can be expressed as a rank-2 update (Level 2 BLAS):

$$\begin{aligned} B_{i+1} &= P_i B_i P_i = (I - u_i u_i^{\mathrm{T}}) B_i (I - u_i u_i^{\mathrm{T}}) \\ &= B_i - u_i v_i^{\mathrm{T}} - w_i u_i^{\mathrm{T}} \end{aligned}$$

where

$$\begin{aligned} y_i &= B_i^{\mathrm{T}} u_i, \\ z_i &= B_i u_i, \\ v_i &= y_i - (z_i^{\mathrm{T}} u_i) u_i, \\ w_i &= z_i \end{aligned}$$

The scheme S1 is a generalization of the double shift QR scheme. The schemes S2 and S3 can be regarded as the generalized Rayleigh shifts. The scheme S4 generalizes Wilkinson's shift. All the schemes were numerically tested using MATLAB. The test results eliminated the schemes S2 and S3 as noncompetitive. The schemes S1 and S4 are competitive for small matrices, but for large matrices (e.g., larger than 100), the scheme S1 is more efficient.

Kahan claims the scheme S5 raises the order of convergence, and makes cycling much less likely. But our preliminary tests for scheme S5 have not shown much benefit over scheme S1. More analysis is necessary for this scheme.

In practice, we will always take k even to avoid complex arithmetic and simplify the logic. It is well-known that to find all eigenvalues of a $k \times k$ real matrix using standard QR, about $8k^3$ flops are necessary, see [Golub and Van Loan'83].

4 Determining the First Column of $\Pi^{(k)}$

We recall that $\Pi^{(k)}$ is defined as

$$\Pi^{(k)} = (A - \mu_1 I)(A - \mu_2 I) \cdots (A - \mu_k I).$$

or

$$\Pi^{(k)} = (A - \mu_k I)(A - \mu_{k-1} I) \cdots (A - \mu_1 I),$$

since the $(A - \mu_i I)$ commute. It is easy to see that $\Pi^{(k)}$ is a lower banded matrix, with lower bandwidth $k + 1$ because A is upper Hessenberg. A trivial way to compute the first column $\pi_1^{(k)} = \Pi^{(k)} e_1$ is to use matrix-vector multiplications directly to form the first column of $\Pi^{(k)}$; this takes $\frac{1}{6}k^3 + O(k^2)$ flops, if we assume that all shifts are real. But this would be rather susceptible to overflow and underflow. Thus, after each multiplication by $(A - \mu_i I)$, we must rescale the resulting vectors to have approximately unit norm, since we only need the direction of the final vector, no essential information is lost.

5 Block k by k Bulge Chasing

¿From section 2, we see that chasing the k by k bulge is the core of the algorithm. Of course, instead of using Givens rotations to chase 1 by 1 or 2 by 2 bulges as in the usual QR algorithm, we can use Householder

Hessenberg matrix with a k by k bulge, e.g. for $n = 9, k = 3$

$$\begin{pmatrix} \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \underline{\times} & \times & \times & \times & \times & \times & \times & \times & \times \\ \underline{\times} & \underline{\times} & \times & \times & \times & \times & \times & \times & \times \\ \underline{\times} & \underline{\times} & \underline{\times} & \times & \times & \times & \times & \times & \times \\ & & & & \times & \times & \times & \times & \times \\ & & & & & \times & \times & \times & \times \\ & & & & & & \times & \times & \times \\ & & & & & & & \times & \times \end{pmatrix}. \tag{1}$$

Then a sequence of Householder transformations P_i are used to reduce B to upper Hessenberg form. After each application $P_i(\cdot)P_i$, the k by k bulge moves one step down the subdiagonal; this is called "chasing". It is easy to see that

$$\tilde{P}_{n-1}e_1 = P_1 e_1.$$

Since P_1 is determined by the first column of $\Pi^{(k)}$, we know that $B_1 = A_k$ by Fact 2.

The above algorithm defines a single QR sweep. Then we take A_k as A_0 to start another sweep. Now we will consider how to choose k shifts simultaneously, how to efficiently compute the first column of $\Pi^{(k)}$, and how to chase the k by k bulge of B. All of these problems will be discussed in detail in the following sections.

3 Multishift Schemes

How to choose k shifts $\{\mu_i\}_{i=1}^k$ in a QR sweep is one of the basic problems of the algorithm. The motivation of using multishifts rather than 1 or 2 shifts is to deflate a small submatrix, presumably about k by k. The prospective shifting schemes are as follows:

S1. k eigenvalues of the $k \times k$ trailing principle submatrix.

S2. The k diagonal elements of the $k \times k$ trailing principle submatrix.

S3. $\mu_1 = \mu_2 = ... = \mu_k = a_{n,n}$.

S4. $\mu_1 = \mu_2 = ... = \mu_k =$ the eigenvalue of the $k \times k$ trailing principle matrix that is nearest to $a_{n,n}$.

S5. Compute $k+1$ eigenvalues of the bottom $k+1$ by $k+1$ matrix, and pick k of them closest to the last set of k shifts, see [Kahan'88].

shifts, determining the first column of the $\Pi^{(k)}$, k by k bulge chasing and the convergence criterion are presented. Preliminary numerical tests of the algorithm on the CONVEX C-1 and CYBER 205 vector machines are reported in section 7. Finally, we discuss future work.

2 The Implicit Multishift QR Algorithm

The usual explicit or implicit QR algorithm uses 1 or 2 shifts to compute A_1 or A_2 from $A_0 = A$. If we assume k shifts $\{\mu_i\}_{i=1}^k$ are available, we will show that the following algorithm computes A_k directly from A_0.

1. Find the first column $\pi_1^{(k)}$ of $\Pi^{(k)}$, where

$$\Pi^{(k)} = (A_0 - \mu_1 I)(A_0 - \mu_2 I)\cdots(A_0 - \mu_k I).$$

2. Determine a Householder transformation $P_1 = I - u_1 u_1^{\mathrm{T}}$ such that

$$P_1 \pi_1^{(k)} = \sigma e_1.$$

   where e_1 is the first column of the identity matrix I.

3. Premultiply and postmultiply A by P_1

$$P_1 A_0 P_1 = B.$$

4. Reduce B to upper Hessenberg form

$$P_{n-1}\cdots P_2 B P_2 \cdots P_{n-1} = \tilde{P}_{n-1}^{\mathrm{T}} A_0 \tilde{P}_{n-1} = B_1.$$

   where $\tilde{P}_{n-1} = P_1 P_2 \cdots P_{n-1}$, and for $i > 1$, P_i is a Householder matrix chosen to zero out rows $i+1$ through $\min(i+k, n)$ of column $i-1$.

We will show that $B_1 = A_k$. To see this, note that $\pi_1^{(k)}$ is a vector with the last $n-k-1$ elements zero, since $\{A_0 - \mu_i I\}$ is a set of upper Hessenberg matrices. Then Householder matrix P_1 transforms $\pi_1^{(k)}$ to σe_1. After P_1 pre- and post-multiplies A_0 , it is easy to see that B is an upper

arithmetic for real matrices, the implicit double shift QR algorithm is used, resulting in 2 by 2 bulge chasing in each QR sweep, see [Francis'61,62, Stewart'73].

The QR iteration has two important properties: Let us first introduce some additional notation. Let

$$\tilde{Q}_{k-1} = Q_0 Q_1 \cdots Q_{k-1}$$

and

$$\tilde{R}_{k-1} = R_{k-1} R_{k-2} \cdots R_0.$$

Then from the fact $A_k = Q_{k-1}^{\mathrm{T}} A_{k-1} Q_{k-1}$, it follows that

$$A_k = \tilde{Q}_{k-1}^{\mathrm{T}} A_0 \tilde{Q}_{k-1}$$

or since $\tilde{Q}_{k-1}$ is orthogonal

$$A_k - s_{k-1} I = \tilde{Q}_{k-1}^{\mathrm{T}} (A - s_{k-1} I) \tilde{Q}_{k-1}.$$

Denote

$$\Pi^{(k)} = (A - s_0 I)(A - s_1 I) \cdots (A - s_{k-1} I).$$

Our QR algorithm, which we call *block multishift QR*, depends on the following two well-known facts about QR iteration:

Fact 1: $\tilde{Q}_{k-1} \tilde{R}_{k-1} = \Pi^{(k)}$, see [Stewart'73].

Fact 2: Let A be an unreduced upper Hessenberg matrix, and suppose that it is transformed by an orthogonal matrix W into $W^{\mathrm{T}} A W$, which is also an unreduced upper Hessenberg matrix. Then if the first column of W is given by $w_1 = \frac{1}{\|\pi_1^{(k)}\|} \pi_1^{(k)}$, where $\pi_1^{(k)}$ is the first column of $\Pi^{(k)}$, the resulting matrix is identical to the kth matrix generated by shifted QR iteration, i.e. $A_k = W^{\mathrm{T}} A W$, see [Francis'61].

Fact 1 reveals the connection between the QR iteration and the power method applied to $\Pi^{(k)}$. Fact 2 is a restatement of a theorem from [Francis'61] in more modern terminology. It is usually called the " implicit Q" theorem as presented in [Stewart'73, Golub and Van Loan'83, Paige'87]. Taken together, they show that if we can compute just the first column $\pi_1^{(k)}$ of $\Pi^{(k)}$, A_k can be computed directly from A_0 by finding an orthogonal W such that $W^{\mathrm{T}} A W$ is upper Hessenberg and the first column of W is proportional to $\pi_1^{(k)}$.

The rest of this paper is organized as follows. In section 2, we describe the implicit multishift QR algorithm. Then, in sections 3 to 6, more details about the implementation of the algorithm, such as the choice of the

and matrix-matrix operations efficiently on high performance machines [see Dongarra *et al*'86,87]. Matrix-vector operations include matrix-vector multiplication, rank-1 matrix update, and solving triangular systems of equations; matrix-matrix operations include matrix-matrix multiplication, rank-k matrix update, and solving triangular systems with many right hand sides.

The vector-vector operations (Level 1 BLAS) on which the usual QR algorithm is based can generally not be implemented as efficiently as the larger granularity matrix-vector and matrix-matrix operations. This inefficiency is usually a result of poor memory hierarchy utilization. This leads us to attempt to restructure the usual QR algorithm to have matrix-vector and matrix-matrix operation inside the inner loops. Even if more floating point operations are introduced by restructuring, the more efficient inner loops may provide a faster overall algorithm.

In this paper, we will describe such a restructured QR algorithm. Briefly, instead of a single or double shift providing a 1 by 1 or 2 by 2 "bulge" which is "chased" one column at a time as in the usual QR algorithm, we will use k simultaneous shifts, resulting in a k by k bulge which is then chased p columns at a time. p and k are architecture dependent parameters, which can be tuned to optimize performance. We choose the k shifts to be the eigenvalues of the bottom right k by k principal sub-matrix.

We have performed preliminary numerical tests of the algorithm on the CONVEX C-1 and CYBER 205. Using straightforward Level 2 and Level 3 BLAS implementations, our algorithm performs better then the vectorized usual QR algorithm (*hqr* from the EISPACK library [Smith'76]). When we use optimized BLAS for these architectures and polish our codes, we expect to get even better performance.

For simplicity, it is assumed that the reader is familiar with the sequential QR algorithm(see [Stewart'73]). The following is an outline of the algorithm. Let $A \in R^{n\times n}$, the explicitly shifted QR algorithm produces a sequence $A_0, A_1, ...$, of similar matrices as follows:

$$
\begin{aligned}
&A_0 = A \\
&\text{for } \ k = 1, 2, ... \\
&\quad A_{k-1} - s_{k-1} I = Q_{k-1} R_{k-1}; \\
&\quad A_k = R_{k-1} Q_{k-1} + s_{k-1} I;
\end{aligned}
$$

The scalars s_{k-1} are called origin shifts. Q_{k-1} is orthogonal and R_{k-1} is upper triangular. For accelerating convergence and avoiding complex

On a Block Implementation of Hessenberg Multishift QR Iteration

Z. Bai and J. Demmel
Courant Institute of Mathematical Sciences
New York University
251 Mercer Street
New York, NY 10012

Abstract

The usual QR algorithm for finding the eigenvalues of a Hessenberg matrix H is based on vector-vector operations, e.g. adding a multiple of one row to another. The opportunities for parallelism in such an algorithm are limited. In this report, we describe a reorganization of the QR algorithm to permit either matrix-vector or matrix-matrix operations to be performed, both of which yield more efficient implementations on vector and parallel machines. The idea is to chase a k by k bulge rather than a 1 by 1 or 2 by 2 bulge as in the standard QR algorithm. We report our preliminary numerical experiments on the CONVEX C-1 and CYBER 205 vector machines.

1 Introduction

The usual QR algorithm for finding the eigenvalues of a Hessenberg matrix H is based on vector-vector operations, e.g. adding a multiple of one row to another, or one column to another. The computed eigenvalues are deflated one by one for real eigenvalues, or pair by pair for complex conjugate eigenvalues. The opportunities for parallelism in such an algorithm are limited.

There are several papers proposing parallel implementations of the QR algorithm, e.g. [Boley'86, Stewart'87, Van de Geijn'87 and Davis *et al*'87]. All these implementations perform vector-vector operations in their innermost loops.

Our approach is motivated by the availability of the Level 2 and Level 3 BLAS (Basic Linear Algebra Subroutines) for performing matrix-vector