High performance solution of the complex symmetric eigenproblem

Ilan Bar-On [a] and Marcin Paprzycki [b]

[a] *Department of Chemistry, Technion, Israel Institute of Technology, Haifa 32 000, Israel*
E-mail: baron@cs.technion.ac.il

[b] *Department of Computer Science and Statistics, University of Southern Mississippi, Hattiesburg, MS 39406-5106, USA*
E-mail: marcin@orca.st.usm.edu

Received 7 May 1997; revised 3 February 1998
Communicated by J. Dongarra

Complex symmetric matrices often appear in quantum physics in the solution methods of partial differential equations such as the Schrödinger equation. We have recently introduced a new fast and efficient direct eigensolver for this problem in [4], and reported its performance in the eigenvalue calculation in [3]. In this paper, we further report on some benchmark tests for computing the full and partial eigenspectrum on a variety of super computing machines, i.e., the Cray J-932, the DEC Alfa 8400, and the SGI Power Challenge 8000 and 10000. We observe that in all cases the new algorithm is much faster than codes available in standard state of the art eigensolver packages such as LAPACK.

Keywords: complex symmetric, eigenvalues, eigenvectors, algorithms, performance

AMS subject classification: 15A18, 65F15, 65Y05

1. Introduction

Recent methods for studying chemical reaction problems, for example the complex scaled DVR method [2,10,13,14,16,17], require the diagonalization of very large (order 7000 and larger) dense complex symmetric (non-Hermitian) matrices. In a typical application first, about 10–20% of the eigenvectors of matrices of order ≈2000 are located about 50 times or more. These eigenvectors are used to construct the large Hamiltonian matrix for which the complete set of eigenvalues is calculated. Modern matrix software libraries, such as LAPACK [1], provide no special treatment of the complex symmetric eigenproblem and codes designed for general complex matrices have to be used. Thus, for example, the computation of the eigenvalues of complex symmetric matrices may be about 10 times slower than calculation of eigenvalues of complex Hermitian matrices [3]. Furthermore, the general eigensolver from the LAPACK library lacks the capability of calculating a subset of eigenvectors, as required

in the applications described above, i.e., the whole set of eigenvectors is computed (about 50 times), when only about 20% are required.

Apparently, the lack of special software for the direct solution of the complex symmetric eigenproblem stems from the following observations:

- Complex symmetric matrices are not necessarily diagonalizable, as is the case for Hermitian matrices, nor do their eigenvalues necessarily possess any special properties. In fact, any general complex matrix can be transformed into a complex symmetric matrix by a sequence of similarity transformations [8].
- The straight reduction of a dense complex symmetric matrix to a tridiagonal form is not always stable.
- There is no robust theory of the complex symmetric tridiagonal eigenproblem.

However, in recent years, some effective Lanczos based iterative algorithms that have been proposed in the literature have shed more light on the problems raised above. Look ahead, and block Lanczos techniques [7,12,15] have been effective in capturing the breakdown problem in the Lanczos process, and Cullum and Willoughby [5,6] have proposed a variant of the QR algorithm for the complex symmetric tridiagonal eigenproblem. However, although iterative methods are very effective for solving large and sparse eigenproblems, they are not very practical for the dense problems, especially when all the eigenvalues of the matrix are required. Hence, in our case, we have to look for direct methods. We have recently suggested a new fast algorithm in [4], which is similar in concept to the standard complex Hermitian eigensolver. The main stages of the algorithm can be summarized as follows:

Tridiagonal reduction. Reduce the complex symmetric matrix $A \in \mathbb{C}^n$ into a tridiagonal complex symmetric matrix $T \in \mathbb{C}^n$ by a sequence of complex orthogonal (yet stable) transformations.

Complex orthogonal QR. Compute the complete set of eigenvalues of the tridiagonal matrix, using the complex orthogonal QR algorithm, and extract the eigenvalues whose eigenvectors are of further interest.

Inverse iteration. Compute the eigenvectors of the tridiagonal matrix, corresponding to the subset of eigenvalues required, by inverse iteration.

Back transformation. Compute the corresponding eigenvectors of the original dense matrix by back transformations.

We have shown in [3] that the new algorithm considerably outperforms the general eigensolver of the LAPACK library when computing the eigenvalues of the complex symmetric matrix (which corresponds to the second stage of the DVR method). In this paper we proceed to study the performance characteristics of the algorithm when computing the complete or partial set of eigenvectors.

The remaining part of the paper is organized as follows: we begin by considering the performance of the general and Hermitian eigensolvers on the Cray J-932 vector computer in section 2. We then review the main features of the new algorithm and its performance on the Cray in section 3. Finally, in section 4, we compare the

performance of the three routines on the superscalar machines, the DEC 8400 and SGI Power Challenge 8000 and Power Challenge 10000. In concluding remarks some directions for future research are presented.

2. The standard algorithms

The standard approach to compute the partial or complete set of the spectrum of a general complex matrix could be described as follows [9]:

Stage (i). Reduce $A \in \mathbb{C}^n$ to an upper Hessenberg matrix $H \in \mathbb{C}^n$ by a sequence of Householder unitary transformations, i.e., $H = P_1^* A P_1$, $P_1^* P_1 = I$, where * denotes the conjugate transpose.

Stage (ii). Reduce $H \in \mathbb{C}^n$ to an upper triangular matrix $R \in \mathbb{C}^n$ by the QR algorithm, i.e., $R = P_2^* H P_2$, $P_2^* P_2 = I$, with the eigenvalues $\Lambda = (\lambda_1, \ldots, \lambda_n)$ on the main diagonal.

Stage (iii). Compute the eigenvectors of R corresponding to the selected set of eigenvalues by inverse iteration, i.e., $RV = V\Lambda$.

Stage (iv). Recover the corresponding eigenvectors of the original matrix by back transformations, i.e., $AQ = Q\Lambda$ with $Q = P_1 P_2 V$.

When the matrix is complex Hermitian, the algorithm simplifies, and computational performance improves as follows:

(a) the reduced matrix is tridiagonal so that stage (i) is $\approx$2.5 times faster,

(b) the QR step takes only $O(n^2)$ operations as compared to $O(n^3)$ for the general case,

(c) the computation of each eigenvector in stage (iii) takes only $O(n)$ operations as compared to $O(n^2)$ for the general case.

We illustrate this behavior in practice on the Cray J-932 vector machine (see tables 1–3) with the aid of the LAPACK routines CGEEVX (general eigensolver) and CHPEVX (Hermitian eigensolver). Both of these routines follow the general algorithm outlined above. In table 1 we consider the case of computing the eigenvalues of the matrix, in table 2 the complete spectrum, and in table 3 the partial set of the spectrum. We report both on the running times (in sec.), and the Mflop rates (in parentheses) for problems of size 400, 1000, 1600 and 2200. The performance data was collected using the Cray *perftrace* utility, and the results reported are averages of multiple runs.

In table 1 we observe a speedup of $\approx$10 of the Hermitian eigensolver over the general one. In addition, the Mflop rate of the Hermitian eigensolver is $\approx$1.28 higher and this is impressive even in terms of the practical peak performance of the Cray, 195 Mflops for the level 3 BLAS matrix–matrix multiplication (see table 8). We note also that the drop in performance of the general eigensolver when $n = 1600$ is likely due to the memory bank conflicts.

Table 1
Eigenvalue calculations in sec. (Mflops).

	$n = 400$	$n = 1000$	$n = 1600$	$n = 2200$
General	22(122)	289(139)	1275(129)	2920(146)
Hermitian	2.2(156)	30(178)	120(184)	304(187)
Speedup	10	9.6	10.6	9.6

Table 2
Complete eigensystem in sec. (Mflops).

	$n = 400$	$n = 1000$	$n = 1600$	$n = 2200$
General	49(139)	682(150)	3070(138)	7150(153)
Hermitian	11(122)	163(132)	687(128)	1670(137)
Speedup	4.5	4.2	4.5	4.3

Table 3
Speedup for partial calculation of the eigensystem set.

	$n = 400$	$n = 1000$	$n = 1600$	$n = 2200$
20%	12.4	14.7	16.2	16.2
40%	8.3	10.3	11.2	11.6
60%	6.0	7.7	8.1	8.7
80%	4.5	5.7	6.2	6.8

In table 2 we see that the speedup of the Hermitian solver over the general solver has now been decreased to only $\approx$4.4. This could be partially explained by the fact that the computation of the eigenvectors in stage (iv) is very time consuming and provides no particular advantage for the Hermitian eigensolver. However, we observe here also a change in the Mflop rates of the two routines. For the largest matrix (when $n = 2200$) the performance of the general solver improves from 146 to 153 Mflops while the performance of the Hermitian solver drops from 187 to 137 Mflops. This points to some deficiencies in the implementation of the eigenvector calculation stages of the Hermitian eigensolver. Finally, note again the drop in performance when $n = 1600$, this time in both routines.

In table 3, we further demonstrate the superiority of the Hermitian solver over the general one when computing only a partial set of the spectrum, as is the case for the DVR method. In this example, the speedup may become even more significant, as the general solver computes the whole spectrum nonetheless.

3. Complex symmetric matrices

3.1. An overview

Let us review the standard Hermitian process applied to dense matrices, i.e.,

$$H^{(k-1)} = Q_k^* H^{(k)} Q_k, \quad k = n, \ldots, 3, \tag{1}$$

where $H = H^{(n)}$, $Q_k^* Q_k = I$ and $H^{(2)}$ is tridiagonal Hermitian. Considering the kth step above, we have

$$H^{(k)} = \begin{pmatrix} H_k^{(k)} & \delta_{k+1} & & \\ \overline{\delta}_{k+1} & \gamma_{k+1} & \ddots & \\ & \ddots & \ddots & \delta_n \\ & & \overline{\delta}_n & \gamma_n \end{pmatrix}, \quad H_k^{(k)} = \begin{pmatrix} H_{k-1}^{(k)} & h \\ h^* & \gamma_k \end{pmatrix},$$

$$h \in \mathbb{C}^{k-1},\ \gamma_k \in \mathbb{R},\ \delta_k \in \mathbb{C}, \tag{2}$$

and

$$Q_k = \begin{pmatrix} \mathcal{Q}_{k-1} & \\ & I_{n-k+1} \end{pmatrix}$$

with

$$\mathcal{Q}_{k-1} = I - 2\frac{vv^*}{v^*v}, \quad v = h + \frac{h_{k-1}}{(h_{k-1}^* h_{k-1})^{1/2}} (h^* h)^{1/2} e_{(k-1)},$$

where $e_{(k-1)}$ is the $(k-1)$th standard unit vector. Hence,

$$H_{k-1}^{(k-1)} = \mathcal{Q}_{k-1}^* H_{k-1}^{(k)} \mathcal{Q}_{k-1}, \qquad \delta_k = -\frac{h_{k-1}}{(h_{k-1}^* h_{k-1})^{1/2}} (h^* h)^{1/2}.$$

Suppose that we try to apply this method to the complex symmetric matrices by replacing the conjugate transpose operation $(\cdot)^*$ with the more simple complex transpose operation $(\cdot)^{\mathrm{T}}$. We would then get

$$\mathcal{Q}_{k-1} = I - 2\frac{vv^{\mathrm{T}}}{v^{\mathrm{T}}v}, \quad v = h + \frac{h_{k-1}}{(h_{k-1}^{\mathrm{T}} h_{k-1})^{1/2}} (h^{\mathrm{T}} h)^{1/2} e_{(k-1)}.$$

However, this process may break down, as, for example, when

$$H = \begin{pmatrix} 0 & 3 & 4 & 5\mathrm{i} \\ 3 & 0 & 5\mathrm{i} & 4 \\ 4 & 5\mathrm{i} & 0 & 3 \\ 5\mathrm{i} & 4 & 3 & 0 \end{pmatrix}, \qquad X = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}, \qquad \begin{aligned} \lambda_1 &= 7 + 5\mathrm{i}, \\ \lambda_2 &= 1 - 5\mathrm{i}, \\ \lambda_3 &= -1 - 5\mathrm{i}, \\ \lambda_4 &= -7 + 5\mathrm{i}, \end{aligned}$$

with $HX = X\Lambda$. Here, $b^{\mathsf{T}} = (5\mathrm{i}, 4, 3)$ and $b^{\mathsf{T}}b = 0$ even though the spectral decomposition is very well defined. Similarly, the process may suffer from extremely large numerical errors when $b^{\mathsf{T}}b \approx 0$, or even when the Q's are of a bounded norm but their accumulated effect is no longer bounded. Thus, the straightforward application of the algorithm to complex symmetric matrices would not be stable in general. We shall now describe a modified approach that we found to be quite stable in practice [4]. Consider the kth step of (2), and let us denote the leading kth submatrix with $H = A + \mathrm{i}B$,

$$A = \begin{pmatrix} A' & a \\ a^{\mathsf{T}} & \alpha \end{pmatrix}, \qquad B = \begin{pmatrix} B' & b \\ b^{\mathsf{T}} & \beta \end{pmatrix}, \quad a, b \in \mathbb{R}^{k-1},\ \alpha, \beta \in \mathbb{R}.$$

Then, to reduce the column vector $a + \mathrm{i}b$ to the complex scalar δ, we apply the following three basic transformations. First, we apply a Householder transformation to reduce $b \in \mathbb{R}^{k-1}$, i.e., we apply

$$Q_I = I - 2ww^{\mathsf{T}}, \quad w = \frac{v}{\|v\|_2}, \qquad v = b + \operatorname{sign}(b_{k-1})\,\|b\|_2\, e_{(k-1)},$$

to both A and B yielding $H_I = A_I + \mathrm{i}B_I$ with

$$A_I = \begin{pmatrix} A'_I & \bar{a} \\ \bar{a}^{\mathsf{T}} & \alpha \end{pmatrix} = \begin{pmatrix} A''_I & \tilde{a} & \bar{a}' \\ \tilde{a}^{\mathsf{T}} & \alpha' & x \\ \bar{a}'^{\mathsf{T}} & x & \alpha \end{pmatrix}, \qquad B_I = \begin{pmatrix} B'_I & y \\ y & \beta \end{pmatrix} = \begin{pmatrix} B''_I & \tilde{b} & \\ \tilde{b}^{\mathsf{T}} & \beta' & y \\ & y & \beta \end{pmatrix},$$

where

$$A'_I = Q_I^{\mathsf{T}} A' Q_I, \qquad \bar{a} = Q_I^{\mathsf{T}} a, \qquad B'_I = Q_I^{\mathsf{T}} B' Q_I, \qquad y = \pm\|b\|_2\,.$$

Next, we apply a Householder transformation to reduce $\bar{a}' \in \mathbb{R}^{k-2}$ using

$$Q_{\mathrm{R}} = I - 2ww^{\mathsf{T}}, \quad w = \frac{v}{\|v\|_2}, \qquad v = \bar{a}' + \operatorname{sign}\left(\bar{a}'_{k-2}\right)\left\|\bar{a}'\right\| e_{(k-2)},$$

and get $H_{\mathrm{R}} = A_{\mathrm{R}} + \mathrm{i}B_{\mathrm{R}}$ with

$$A_{\mathrm{R}} = \begin{pmatrix} A''_{\mathrm{R}} & \hat{a} & z \\ \hat{a}^{\mathsf{T}} & \alpha' & x \\ z & x & \alpha \end{pmatrix}, \qquad B_{\mathrm{R}} = \begin{pmatrix} B''_{\mathrm{R}} & \hat{b} & \\ \hat{b}^{\mathsf{T}} & \beta' & y \\ & y & \beta \end{pmatrix},$$

where

$$A''_{\mathrm{R}} = Q_{\mathrm{R}}^{\mathsf{T}} A''_I Q_{\mathrm{R}}, \quad \hat{a} = Q_{\mathrm{R}}^{\mathsf{T}} \tilde{a}, \quad z = \pm\left\|\bar{a}'\right\|, \quad B''_{\mathrm{R}} = Q_{\mathrm{R}}^{\mathsf{T}} B''_I Q_{\mathrm{R}}, \quad \hat{b} = Q_{\mathrm{R}}^{\mathsf{T}} \tilde{b}.$$

Finally, we reduce $(z, x + \mathrm{i}y)^{\mathsf{T}}$ by the complex orthogonal transformation

$$Q_{\mathrm{C}} = \frac{1}{\delta}\begin{pmatrix} x + \mathrm{i}y & z \\ -z & x + \mathrm{i}y \end{pmatrix}, \quad \delta^2 = (x + \mathrm{i}y)^2 + z^2,\ Q_{\mathrm{C}}^{\mathsf{T}} Q_{\mathrm{C}} = I.$$

Note that the norm of Q_C is easily computable. In fact,

$$Q_C = \frac{1}{\delta} X D X^*, \qquad X = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ \mathrm{i} & -\mathrm{i} \end{pmatrix}, \qquad D = \begin{pmatrix} x + \mathrm{i}(y+z) & \\ & x + \mathrm{i}(y-z) \end{pmatrix},$$

with $X^*X = I$. Hence,

$$\|Q_C\|_2^2 = \frac{x^2 + (|y| + |z|)^2}{|\delta^2|} = \frac{x^2 + y^2 + z^2 + 2|yz|}{((x^2 - y^2 + z^2)^2 + (2xy)^2)^{1/2}}$$

$$= \frac{x^2 + y^2 + z^2 + 2|yz|}{((x^2 + y^2 + z^2)^2 - (2yz)^2)^{1/2}},$$

and

$$\|Q_C\|_2 = \left(\frac{1+\theta}{1-\theta}\right)^{1/4}, \quad \theta = \frac{2|yz|}{x^2 + y^2 + z^2} \leqslant 1.$$

We can, therefore, monitor the norm of Q_C by evaluating θ, proceeding with the above transformation as long as θ is less than some prescribed threshold $\tau < 1$. In this case, the reduced matrix satisfies

$$H' = Q^{\mathrm{T}} H Q, \quad Q = Q_I Q_R Q_C, \; Q^{\mathrm{T}} Q = I,$$

where Q_I and Q_R are real orthogonal and therefore stable, and Q_C is complex orthogonal but of a bounded norm. Furthermore, as Q_C applies to the next to last two columns and rows, of which only the second to last gets involved in the transformations that follows, rounding errors tend to remain small as is shown in [4]. The case of a breakdown, i.e., $\theta > \tau$, could be dealt with in several ways discussed in [4]. For example, when $\delta_{k+1} \approx 0$ we permute the matrix rows and columns or apply some Jacobi transformation. Otherwise, when $\delta_{k+1} \neq 0$ we apply a step of the QR algorithm to the trailing tridiagonal matrix (see (2)), which has again the effect of applying some orthogonal transformation to the original matrix [5,11]. As a breakdown in the algorithm is due to some special arrangement of the matrix elements, these transformations help to destroy that structure and continue with the standard algorithm. Thus, the number of breakdowns is small in practice, and the computational complexity remains practically the same. In place of the deterministic sequence (1), we obtain

$$H^{(t+1)} = \left(Q^{(t)}\right)^{\mathrm{T}} H^{(t)} Q^{(t)}, \quad t = 0, \ldots, m-1, \; m \approx n,$$

where $H^{(0)} = H$ and $T = H^{(m)}$ is tridiagonal and complex symmetric.

Considering the rounding errors, we get

$$\widehat{H}^{(t+1)} = \left(\widehat{Q}^{(t)}\right)^{\mathrm{T}} \widehat{H}^{(t)} \widehat{Q}^{(t)} + F^{(t)}, \quad \left\|F^{(t)}\right\|_2 = \mathrm{O}\left(\varepsilon \left\|\widehat{H}^{(t)}\right\|_2\right),$$

with ε the machine precision. Therefore, as $\| \widehat{H}^{(t)} \|_2 = \mathrm{O}(\varepsilon \| H \|_2)$, the computed eigenvalues are forward stable. Similarly, as

$$\widehat{T} = V^{\mathrm{T}}(H + E)V, \quad E = VFV^{\mathrm{T}}, \quad V = Q^{(0)} Q^{(1)} \ldots Q^{(m-1)},$$

$$F = \sum_{t=0}^{m-1} \left(V^{(t)}\right)^{\mathrm{T}} F^{(t)} V^{(t)}, \quad V^{(t)} = \widehat{Q}^{(t+1)} \ldots \widehat{Q}^{(m-1-t)},$$

the computed eigenpairs should be expected to be backward stable, and this could be verified a posteriori as follows: for each λ an eigenvalue of $H' = H + E$, there is a μ, an eigenvalue of H, such that

$$|\mu - \lambda| \leqslant \frac{|x^{\mathrm{T}} E x|}{|x^{\mathrm{T}} x|} + \mathrm{O}\left(\|E\|_2^2\right),$$

see [5, p. 198]. Hence, since

$$(H + E)x = \lambda x, \quad E = -\frac{r r^{\mathrm{T}}}{r^{\mathrm{T}} x}, \quad r = Hx - \lambda x,$$

we get to the first order $|\mu - \lambda| \leqslant |(x^{\mathrm{T}} r)/(x^{\mathrm{T}} x)|$ and the computed eigenpair is stable provided $|x^{\mathrm{T}} r|$ is relatively small. We conclude by saying that, for the chemical applications discussed in the introduction, we found that the accuracy of the new algorithm is practically the same as for the general solver of LAPACK [4].

3.2. Performance analysis

In this section, we proceed to investigate the performance of the new algorithm on the Cray machine. In tables 4 and 5 we compare the performance of the new algorithm with that of the general and Hermitian eigensolvers from the LAPACK library for matrices of order $n = 2200$. The times (in sec.) and Mflop rates (in parentheses) of the four stages of the new algorithm: the reduction to the tridiagonal matrix (Reduction), the calculation of the eigenvalues of the tridiagonal matrix using CMTQL1 (Eig.), the calculation of the eigensystem of the tridiagonal matrix using INVERM1 (Vec.), both routines from the Lanczos package [5,6], and the calculation of the eigenvectors of the original matrix by back transformation (Recover), are presented in table 4. In table 5, we depict the times for computing the eigenvalues only (column one), and the whole computation (column two) for the above three algorithms. We also represent here the speedup of the new algorithm over these standard routines (in parentheses).

We observe a relatively high performance in the reduction and back transformation stages, i.e., $\approx$175 Mflops or $\approx$91% of the practical peak performance. This performance has been obtained even though the code is implemented using calls to the level 1 and 2 BLAS kernels only. On the other hand, the performance of the intermediate stages, that require only $\mathrm{O}(n^2)$ flops, is very poor and especially so when compared to the performance on the RISC-based machines we review later, see table 9. We note here that the computation of the eigenvalues of tridiagonal Hermitian matrices is also performed much more efficiently [3]. Thus, we may attribute these phenomena to the scalar nature of the routines CMTQL1 and INVERM1 that we used, and conclude that more efficient codes should be used for future implementations. Observe also that the

Table 4
Performance of the new algorithm.

	Reduction	Eig.	Vec.	Recover
$n = 2200$	324(177)	163(14)	154(14)	489(172)

Table 5
Speedup with respect to the standard routines.

	Red. + Eig.	Total
New	487	1130
General	2920(6)	7150(6.3)
Hermitian	304(0.6)	1670(1.5)

Table 6
Percentage for the middle computations.

n	Red.	Trd.	Rec.	Tot.	Trd. / Tot.
2400	422	377	647	1446	26%
2800	666	514	1024	2204	23%
3200	988	672	1510	3170	21%
3600	1403	848	2142	4393	19%
4000	1923	1045	2968	5936	18%

increase in the dimension of the matrices tends to diminish this effect as illustrated in table 6. Here, we report on the relative time of the middle stages (Trd.) in terms of the total running time of the algorithm for increasing size of the matrices. We also depict here the times for the individual Reduction (Red.) and Recovery (Rec.) stages.

We further observe that the new algorithm outperforms the Hermitian solver by $\approx$1.5 times when computing the whole spectrum. Moreover, ignoring the time spent in the two middle stages, which as we noted above is peculiar to the tridiagonal code we use, the new algorithm is even twice as fast. Thus, this reassures our earlier observation (see also section 2, tables 1 and 2) that the Hermitian eigensolver of LAPACK is implemented rather inefficiently.

Finally, in table 7, we report the speedup of the new algorithm with respect to both standard solvers when only a part of the eigensystem set is computed. Here, as before (see table 3), we compare the computation of selected eigenvectors by the new algorithm with that of the complete set by the general eigensolver. We also depict for comparison the speedup with respect to the Hermitian eigensolver computing the same size of a subset. Thus, as is the case for the chemical reaction modeling problems discussed before, the new algorithm may substantially outperform the general eigensolver.

Table 7
Speedup with respect to a partial eigensystem set.

$n = 2200$	General	Hermitian
20%	11.6	0.7
40%	9.6	0.8
60%	8.2	0.9
80%	7.1	1.1

Table 8
Performance of BLAS level 2 and 3.

$n = 1000$	Cray	DEC	SGI1	SGI2
_GEMM	195	220	290	340
_GEMV	190	40	45	50

4. Superscalar machines

We proceed in this section to compare the performance of the three solvers on the superscalar machines: the DEC Alpha Server 8400 (DEC), the Silicon Graphics Power Challenge 8000 (SGI1), and the Silicon Graphics Power Challenge 10000 (SGI2). We also compare their performance with that of the Cray J-932. To obtain the running times we have used the appropriate Unix-based CPU-time-oriented timers. All codes have been run in double precision (which is compared to the Cray's single precision). Manufacturer provided BLAS kernels and LAPACK routines have been used. Each result presented is an average of multiple runs.

In table 8, we report typical measures of the practical performance of these machines for the level 2 and 3 BLAS kernels. Note that on the Cray, both levels are efficiently implemented, whereas on the RISC machines, the level 3 routines are much superior to those of level 2. Hence, the actual performance may be heavily dependent on the detailed implementation issues of the algorithm, and whether it is using level 2 or level 3 routines. In our case we note that the LAPACK routines were designed to take advantage of these features, see [1], whereas the new code is currently implemented using only level 2 routines. Nonetheless, the new code is faster than the general eigensolver and further improvements should be anticipated when implementing the algorithm with level 3 BLAS.

4.1. The classical algorithms

We report in tables 9, 10 and 11 on the performance of the standard LAPACK routines ZGEEVX and ZHPEVX for matrices of order $n = 2200$. We present their running times (in sec.) and their efficiency with respect to the performance on the Cray J-932 (in parentheses).

In table 9 we consider the case of computing the eigenvalues only. We present separately the running times of the reduction stage and the calculation of the spectrum

Table 9
Eigenvalue calculations in sec.

$n = 2200$	DEC	SGI1	SGI2
	General solver		
Reduction	1388	1130	1228
Spectrum	2196	3095	2263
Total	3584 (81%)	4225 (69%)	3491 (84%)
	Hermitian solver		
Reduction	689	677	720
Eigenvalue	3	6	2
Total	692 (44%)	683 (45%)	722 (42%)
Speedup	5.2	6.2	4.8

Table 10
Complete eigensystem in sec.

$n = 2200$	DEC	SGI1	SGI2
General	8292	11081	7588
	9680 (74%)	12211 (59%)	8816 (81%)
Hermitian	3321	2648	3378
	4010 (42%)	3325 (50%)	4098 (41%)
Speedup	2.4	3.7	2.2

stage. For the general solver, in the reduction stage the SGI1 is about 20% faster than the DEC (and about 10% faster than SGI2), but about 40% slower than these machines on the spectrum stage. In case of the Hermitian solver the DEC and the SGIs behave practically the same.

In table 10 we consider the case where we compute the complete set of eigenvectors. Here, the reduction stage remains essentially the same so that we depict the times for the remaining stages, i.e., the calculation of the eigenvalues and eigenvectors of the reduced matrix, and the computation of the eigenvectors of the original matrix. We also depict here the total time for the whole algorithm. Note that here SGI1 is the slowest machine for the general case, but the fastest for the Hermitian case.

In table 11 we finally consider the computation of only a partial set of the eigensystem (similar to table 3). We depict the speedup of the Hermitian solver calculating the partial eigensystem with respect to the general solver computing the complete eigensystem set. In addition, we report (in parentheses) the efficiency of these routines with respect to their performance on the Cray. We observe a decrease in the speedup of the Hermitian solver over the general solver as compared to the one on the Cray, see table 3. This is especially evident for the DEC, for which the efficiency does not reach 40% of the Cray. It should be noted that when the complete eigensystem is calculated the performance of the DEC (compared to the Cray) improves by about 17% while the

Table 11
Speedup for partial calculation of the eigensystem set.

$n = 2200$	DEC	SGI1	SGI2
20%	8.4 (38%)	12.5 (45%)	8.5 (43%)
40%	5.1 (32%)	9.3 (47%)	6.5 (46%)
60%	3.3 (28%)	8.0 (54%)	5.3 (50%)
80%	2.3 (25%)	6.6 (57%)	4.6 (55%)

Table 12
Computing the spectrum.

$n = 2000$	Reduction	Eig.	Total	General	Hermitian
DEC	848 (28%)	30 (440%)	878 (44%)	3.1	0.6
SGI1	1062 (23%)	54 (252%)	1116 (34%)	2.8	0.6
SGI2	1120 (22%)	20 (718%)	1140 (34%)	2.4	0.6

performance decreases by 7% for SGI1 and by 14% for SGI2 (see table 10 above).

Overall, these results indicate how inconsistent superscalar machines can be and how performance of such machines may depend on the implementation details. We may also conclude that the Cray vector machine significantly outperforms the superscalar machines, and this is quite surprising in view of the higher peak performance of the RISC machines on matrix multiplication. We also observe that on the RISC machines the speedup of the Hermitian eigensolver has been greatly reduced, as compared to that on the Cray (see tables 1 and 2), suggesting that the latter is implemented, using level 3 BLAS, more efficiently.

4.2. The new algorithm

We illustrate the performance of the new algorithm for matrices of order $n = 2000$ in tables 12–14. In table 12, we present the time for the reduction stage and for the calculation of the eigenvalues stage, and the corresponding speedup with respect to the general and Hermitian eigensolver. As before, we also report (in parentheses) the efficiency of the new algorithm with respect to the Cray. In table 13, we present the results for computing the complete eigensystem (the results in parentheses represent the efficiency as compared to the Cray). Finally, table 14 contains the speedup of computing a partial eigensystem set, with respect to computing the complete eigensystem set for the general eigensolver and the partial eigensystem set of the same size for the Hermitian eigensolver (in parentheses).

We note that SGI2 significantly outperforms SGI1 in the middle stages, dealing with the diagonalization of the complex symmetric tridiagonal matrix. This could be explained in terms of the improved scalar architecture introduced into the Power Challenge 10000, and the fact that the routines CMTQL1 and INVERM1 are implemented using a plain scalar code. At the same time, when the complete solution process is con-

Table 13
Complete eigensystem in sec.

$n = 2000$	Vec.	Recover	Total	General	Hermitian
DEC	34 (376%)	2104 (17%)	3016 (29%)	2.4	1.0
SGI1	52 (246%)	2136 (17%)	3304 (27%)	2.7	0.8
SGI2	24 (534%)	2854 (13%)	4018 (22%)	1.7	0.8

Table 14
Speedup for partial calculation of the eigensystem set.

$n = 2000$	DEC	SGI1	SGI2
20%	5.1 (0.6)	6.3 (0.5)	4.3 (0.5)
40%	4.1 (0.8)	4.7 (0.5)	3.3 (0.5)
60%	3.3 (1.0)	4.1 (0.5)	2.7 (0.5)
80%	2.8 (1.2)	3.3 (0.5)	1.8 (0.4)

sidered the SGI1 uses only 82% of the time required by the SGI2. Both SGI machines are slower than the DEC computer.

All three superscalar machines considerably outperform the Cray J-932 in the middle stages, indicating their superiority in terms of scalar operations. However, since the most time consuming stages of the algorithm, the reduction and recovery transformations, employ vector operations, the Cray remains considerably faster than the other machines. We finally note that in all cases the new code outperforms the general solver and that further improvement may be anticipated when implementing the new code using the level 3 BLAS routines.

5. Concluding remarks

We have presented in this paper some performance results for standard LAPACK routines for computing the eigensystem of complex general and Hermitian matrices on some state of the art supercomputer machines. We observed that performance, especially on the superscalar machines, may be significantly affected by the use of the different levels of the BLAS routines. Our experience suggests that for their better usage, software packages such as LAPACK should also indicate the expected performances of their routines.

We have further presented the performance of the new algorithm for computing the eigensystem of large and dense complex symmetric matrices that substantially outperforms the general eigensolver of LAPACK, the currently available routine for this problem. Our results indicate also that, for the time being, the vector processing Cray is the best choice for this type of problems.

Our future research will address a number of questions. First, the usage of blocking techniques will be investigated. This technique should improve the performance of the code primarily on the RISC-based computers. Second, the efficiency of diago-

nalization of complex symmetric tridiagonal matrices on the Cray needs to be studied. Finally, additional work may focus on the efficient implementation of the new code on parallel machines.

Acknowledgements

A computer time grant from NCSA in Urbana-Champaign, and from the Department of Chemistry in the Technion, Israel, is kindly acknowledged. We also thank Prof. Ian Gladwell for very helpful comments.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J.D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorensen, *LAPACK Users' Guide* (SIAM, Philadelphia, PA, 1992).

[2] E. Balslev and J. Combes, Spectral properties of many body Schrödinger operators with dilation analytic interactions, Commun. Math. Phys. 22 (1971) 280–294.

[3] I. Bar-On and M. Paprzycki, An efficient algorithm for finding eigenvalues of complex symmetric matrices, Comput. Assisted Mech. Engrg. Sci. (1997).

[4] I. Bar-On and V. Ryaboy, Fast diagonalization of large and dense complex symmetric matrices, with applications to quantum reaction dynamics, SIAM J. Sci. Comput. 18 (1997).

[5] J.K. Cullum and R.A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations* (Birkhäuser, Boston, 1985).

[6] J.K. Cullum and R.A. Willoughby, A QL algorithm for complex symmetric matrices, SIAM J. Matrix Anal. Appl. (1996).

[7] R.W. Freund, G.H. Golub and N.M. Nachtigal, Iterative solution of linear systems, Acta Numer. (1992) 1–44.

[8] F.R. Gantmacher, *The Theory of Matrices*, Vols. 1,2 (Chelsea, New York, 1959).

[9] G.H. Golub and C.F.V. Loan, *Matrix Computations* (Johns Hopkins Univ. Press, Baltimore, MD, 1989).

[10] N. Moiseyev, Resonances, cross-sections and partial widths by the complex coordinate method, Isr. J. Chem. 31 (1991) 311–322.

[11] B.N. Parlett, *The Symmetric Eigenvalue Problem* (Prentice-Hall, Englewood Cliffs, NJ, 1980).

[12] B.N. Parlett, D.R. Taylor and Z.A. Liu, A look-ahead Lanczos algorithm for unsymmetric matrices, Math. Comp. 44 (1985) 105–124.

[13] W. Reinhardt, Complex coordinates in the theory of atomic and molecular structure and dynamics, Ann. Rev. Phys. Chem. 33 (1982) 223–255.

[14] V. Ryaboy and N. Moiseyev, Three dimensional study of predissociation resonances by the complex scaled discrete variable representation method: HCO/DCO, J. Chem. Phys. 103 (1995).

[15] Y. Saad, *Numerical Methods for Large Eigenvalue Problems* (Halsted Press, New York, 1992).

[16] J. Simon, Quadratic form techniques and the Balslev–Combes theorem, Commun. Math. Phys. 27 (1972) 1–9.

[17] J. Simon, Resonances in n-body quantum systems with dilation analytic potentials and the foundations of time dependent perturbation theory, Ann. Math. 97 (1973) 247–274.