JOURNAL OF COMPUTATIONAL PHYSICS **138,** 286–301 (1997)
ARTICLE NO. CP975819

Computation of Rovibrational Eigenvalues of van der Waals Molecules on a CRAY T3D

Xudong T. Wu,* Prakashan P. Korambath,* Edward F. Hayes,*,[1] and Danny C. Sorensen†

**Department of Chemistry, Ohio State University, Columbus, Ohio 43210; †Department of Computational and Applied Mathematics, Rice University, Houston, Texas 77001*

Received March 18, 1997; revised August 5, 1997

Two algorithms for computing rovibrational eigen solutions for van der Waals molecules are presented. The performance and scalability of these algorithms are evaluated on a CRAY T3D with 128 processors using Ar–HO as the test molecule. Both algorithms are based on a discrete variable representation (DVR) of the rovibrational Hamiltonian for van der Waals molecules. The first algorithm applies the implicitly restarted Lanczos method (IRLM) of D. C. Sorensen directly to the DVR Hamiltonian to obtain the eigenpairs of interest. The second algorithm transforms the DVR Hamiltonian using the sequential diagonalization and truncation (SDT) approach of Light and co-workers to a reduced order SDT Hamiltonian prior to applying the IRLM. Both algorithms make use of Chebychev polynomial preconditioning to speed up the convergence of the IRLM. An important factor in the performance of the two algorithms is the efficiency of the matrix–vector product operation. Both algorithms make use of a Sylvester-type transformation to convert most DVR matrix–vector operations into a series of significantly lower order matrix–matrix operations. The basic trade-offs between the two algorithms are that the first algorithm has a significantly higher percentage of level-3 BLAS operations, which allows it to achieve higher Mflops, whereas the second algorithm involves the lower order SDT Hamiltonian, which makes the IRLM converge faster. The implementation details (e.g., the distribution of with the different submatrices that form the tensor representation of the DVR Hamiltonian) are central to achieving maximum efficiency and near linear scalability of the algorithms for large values of the total angular momentum.

[1] Corresponding author.

0021-9991/97 $25.00

I. INTRODUCTION

The theoretical prediction of rovibrational energy levels for polyatomic molecules becomes increasingly difficult as the number of atoms in the system increases. While several groups have developed approaches that work well on current scalar and vector machines for three-atom systems, there is considerable interest in finding more powerful approaches that can be used to solve problems that involve more than three active degrees of freedom [1–6]. The focus of this paper is on the performance and scalability of two algorithms that we have developed for obtaining the rovibrational eigenvalues of van der Waals molecules on a CRAY T3D. Van der Waals molecules have received a good deal of attention recently because of the wealth of high resolution spectroscopic and dynamical data that is becoming available for these weakly bonded molecules [7–12]. Previously we reported rovibrational energy calculation using serial algorithms for van der Waals molecule such as Ar–HS, Kr–HS [13], Ar–HCl, and Ar–HO [14]. The Ar–HO molecule was selected for this study to generate detailed performance information that is typical for these atom–diatom van der Waals molecules.

The computation time required to determine the rovibrational eigenvalues of a molecule increases considerably as one goes from an atom–diatom system to atom–polyatom systems because more degrees of freedom must be included to obtain accurate theoretical predictions. In this paper we will evaluate two algorithms for computing rovibrational eigen solutions for van der Waals molecules. The performance and scalability of these algorithms will be evaluated on a CRAY T3D with 128 processors using results from the study of the Ar–HO van der Waals molecule. Both algorithms to solve the rovibrational Hamiltonian for van der Waals molecules are based on a discrete variable representation (DVR) approach originally introduced by Light and co-workers [15]. The first algorithm applies the implicitly restarted Lanczos method (IRLM) of Sorensen [16] directly to the DVR Hamiltonian to obtain the eigenpairs of interest. The second algorithm transforms the DVR Hamiltonian using the sequential diagonalization and truncation (SDT) approach of Light and co-workers [17] to a reduced-order SDT Hamiltonian prior to applying the IRLM. Both algorithms make use of Chebychev polynomial preconditioning to speed up the convergence of the IRLM. One factor in the performance of the two algorithms is the efficiency of the matrix–vector product operation. Both algorithms make use of a Sylvester-type [18] transformation to convert most DVR matrix vector operations into a series of significantly lower order matrix–matrix operations. The basic trade-offs between the two algorithms are that the first algorithm has a significantly higher percentage of level-3 BLAS operations, which allows it to achieve higher Mflops, whereas the second algorithm involves the lower order SDT Hamiltonian, which makes the IRLM converge faster. The implementation details (e.g., the distribution of the different submatrices that form the tensor representation of the DVR Hamiltonian and the processor-to-processor communication strategy) are central to achieving maximum efficiency and near linear scalability of the algorithms for large values of the total angular momentum. These will be discussed in detail in the next several sections.

II. DVR HAMILTONIAN

The Hamiltonian for an atom–diatom system in a DVR was originally given in Jacobi coordinates (R, r, θ) and total angular momentum representation of the body-fixed reference frame by Choi and Light [19],

$$H^{\alpha'\beta'k'}_{\alpha\beta k} = T_{\alpha'\alpha}\cdot\delta_{\beta'\beta}\cdot\delta_{k'k} + \frac{\hbar^2}{2\mu}\left(\frac{1}{R_\alpha^2}+\frac{1}{r_e^2}\right)[\mathscr{A}_k]_{\beta'\beta}\cdot\delta_{\alpha'\alpha}\cdot\delta_{k'k}$$
$$+\frac{\hbar^2}{2\mu R_\alpha^2}\{[J(J+1)-2k^2]\cdot\delta_{\alpha'\alpha}\cdot\delta_{\beta'\beta}\cdot\delta_{k'k} - (1+\delta_{k0})^{1/2}\Lambda^+_{Jk}[\mathscr{B}^+_k]_{\beta'\beta}\cdot\delta_{\alpha'\alpha}\cdot\delta_{k'k+1}$$
$$-(1+\delta_{k'0})^{1/2}\Lambda^-_{Jk}[\mathscr{B}^-_k]_{\beta'\beta}\cdot\delta_{\alpha'\alpha}\cdot\delta_{k'k-1}\} + V(R_\alpha,\theta_{k\beta})\cdot\delta_{\alpha'\alpha}\cdot\delta_{\beta'\beta}\cdot\delta_{k'k} \quad (1)$$

in which the $\Lambda^{\pm}_{Jk}[\mathscr{B}^{\pm}_k]_{\beta'\beta}$ are the Coriolis coupling terms that satisfy

$$\Lambda^+_{Jk}\mathscr{B}^+_k = \Lambda^-_{J(k+1)}\mathscr{B}^-_{k+1} \quad (2)$$

and $V(R_\alpha, \theta_{k\beta})$ is the potential energy term. In mass-scaled Jacobi coordinates R is used to represent the distance between the atom and the center of mass of the diatom. Since the diatom is treated as having a fixed internuclear separation, the equilibrium distance r is fixed at r_e (the equilibrium value of the diatom in its ground vibrational state); θ is the angle formed between **R** and **r** (which lies along the diatom bond); J is the total angular momentum and k is the projection of the total angular momentum onto the body-fixed frame; and $T_{\alpha'\alpha}$ is the kinetic-energy matrix due to radial motion; while $[\mathscr{A}_k]_{\beta'\beta}$ is the kinetic-energy matrix due to angular motion. In Eq. (1), we have used the α index to label the quadrature points in R and β to label the quadrature points in θ.

In this paper, the approach to obtaining the eigen-solutions to Eq. (1), has been influenced by two recent advances. First, we make use of several ideas developed [14] for finding the eigenpairs of this Hamiltonian on a sequential computer using IRLM. Second, we follow the idea suggested by Hu and Sorensen [20]. They pointed out the value of using a Sylvester-type transformation to convert a high-order matrix–vector operation into a series of lower order matrix–matrix operations, whenever the initial matrix can be written in a tensor-product form. This converts many level-2 BLAS operations into level-3 BLAS operations and makes it practical to take advantage of the higher performance level-3 BLAS routines on a CRAY T3D.

For example, if a matrix, M, has the tensor product representation

$$M = B \otimes A, \quad (3)$$

where $A \in \mathscr{R}^{m\times m}$ and $B \in \mathscr{R}^{n\times n}$, then

$$y = Mx, \quad (4)$$

where $x, y \in \mathscr{R}^{mn}$, can be calculated through

$$Y = AXB^T, \tag{5}$$

where $X, Y \in \mathscr{R}^{m\times n}$.

The DVR Hamiltonian, Eq. (1), can also be written as a tensor product,

$$H^{\mathrm{DVR}} = I_K \otimes I_{n_\theta} \otimes T + \sum_k E^{K,K}_{k,k} \otimes A_k \otimes D_1 + V + \sum_k E^{K,K}_{k,k+1} \otimes B_k \otimes D_2 + \sum_k E^{K,K}_{k+1,k} \otimes B_k^T \otimes D_2 \tag{6}$$

where $T \in \mathscr{R}^{n_R\times n_R}$ and $A_k \in \mathscr{R}^{n_\theta\times n_\theta}$ are given by Eq. (1). And $D_1 = \hbar^2/2\mu(1/R^2 + 1/r_e^2)$, $D_2 = \hbar^2/2\mu R^2$ are diagonal matrices. B_k, on the other hand, relates to $\mathscr{B}_k^+$ through

$$[B_k]_{\beta'\beta} = -(1 + \delta_{k0})^{1/2}\Lambda^+_{Jk}[\mathscr{B}^+_k]_{\beta'\beta}. \tag{7}$$

The diagonal matrix V' is the effective potential which includes the centrifugal potential

$$(V')^{\alpha'\beta'k'}_{\alpha\beta k} = V(R_\alpha, \theta_{k\beta})\cdot\delta_{\alpha'\alpha}\cdot\delta_{\beta'\beta}\cdot\delta_{k'k} + \frac{\hbar^2}{2\mu R_\alpha^2}\{[J(J+1) - 2k^2]\cdot\delta_{\alpha'\alpha}\cdot\delta_{\beta'\beta}\cdot\delta_{k'k}. \tag{8}$$

Finally, $E^{KK}_{k,k}$, $E^{KK}_{k+1,k}$, and $E^{KK}_{k,k+1}$ are $K\times K$ matrices which have only one nonzero element. For $E^{KK}_{k,k}$, $E^{KK}_{k+1,k}$, and $E^{KK}_{k,k+1}$ the nonzero element is the (k, k) element, the $(k+1, k)$ element, and the $(k, k+1)$ element, respectively.

III. APPLICATION OF SYLVESTER-TYPE TRANSFORMATION

To write down the matrix equivalent to $y = H^{\mathrm{DVR}}x$, we need to define the operators $\tilde{T}$, $\tilde{A}_k$, $\tilde{B}_k$, and $\tilde{B}'_k$. $Y = \tilde{T}(X)$ is defined as

$$Y(i, :, :) = \sum_{j=1}^{n_R} T_{i,j}X(j, :, :); \tag{9}$$

$Y = \tilde{A}_k(X)$ is defined as

$$Y(:, i, l) = \begin{cases} 0, & \text{if } l \neq k, \\ \sum_{j=1}^{n_\theta} [A_k]_{i,j}X(:, j, k), & \text{if } l = k; \end{cases} \tag{10}$$

$Y = \tilde{B}_k(X)$ is defined as

$$Y(:, i, l) = \begin{cases} 0, & \text{if } l \neq k+1, \\ \sum_{j=1}^{n_\theta} [B_k]_{i,j}X(:, j, k), & \text{if } l = k+1; \end{cases} \tag{11}$$

and finally $Y = \tilde{B}'_k(X)$ is defined as

$$Y(:,i,l) = \begin{cases} 0, & \text{if } l \neq k, \\ \sum_{j=1}^{n_\theta} [B_k]_{j,i} X(:,j,k+1), & \text{if } l = k-1. \end{cases} \tag{12}$$

With these definitions, $y = H^{\mathrm{DVR}}x$ can be calculated through

$$Y = \tilde{T}(X) + D_1\left(\sum_{k=1}^{K} \tilde{A}_k(X)\right) + D_2\left(\sum_{k=1}^{K} (\tilde{B}_k(X) + \tilde{B}'_k(X))\right) + V' \odot X, \tag{13}$$

where X and x follows the usual one-to-one correspondence from $\mathscr{R}^{n_R \times n_\theta \times K}$ to $\mathscr{R}^{n_R n_\theta K}$ so that we have

$$x = \begin{pmatrix} X(:,1,1) \\ X(:,2,1) \\ \vdots \\ X(:,n_\theta,1) \\ \vdots \\ X(:,1,K) \\ \vdots \\ X(:,n_\theta,K) \end{pmatrix} \tag{14}$$

and the operation denoted by "$\odot$" is the element-by-element multiplication,

$$(V' \odot X)(i,j,k) = V'(i,j,k)X(i,j,k). \tag{15}$$

IV. IMPLEMENTATION ON THE CRAY T3D

The implementation of Eqs. (9)–(12) in a parallel fashion is accomplished by partitioning the X, Y, and other matrices in Eqs. (9)–(12) and assigning the submatrices to different processors. Locally each processor performs matrix–matrix multiplications using level-3 BLAS routines to the maximum extent possible. Periodically, processors must obtain the needed X or Y values from other processors through interprocessor data communications. In the next few paragraphs, we describe how the X and Y arrays are partitioned, what computation is done on each processor, and what kind of the communication is required for this implementation.

The implementation strategy assumes that the processors are "logically" sitting on a 3D torus network of processors, the size of the 3D torus is $p_R \times p_\theta \times p_K$ and the total number of the processors $p = p_R p_\theta p_K$. The 3D array $X = X(1 : n_R, 1 : n_\theta, 1 : K)$ is distributed on the torus as follows: On processor (i_R, i_θ, i_K), the local submatrix, X_{loc} corresponds to

$$\begin{aligned} &X_{\mathrm{loc}}(1 : m_R, 1 : m_\theta, 1 : m_K) \\ &\quad = X(i_R m_R + 1 : (i_R + 1)m_R, i_\theta m_\theta + 1 : (i_\theta + 1)m_\theta, i_K m_K + 1 : (i_K + 1)m_K), \end{aligned} \tag{16}$$

where m_R, m_θ, and m_K, the array dimensions on local processors, are

$$m_R = \frac{n_R}{p_R} \tag{17}$$

$$m_\theta = \frac{n_\theta}{p_\theta} \tag{18}$$

$$m_K = \frac{K}{p_K}. \tag{19}$$

The Y matrix also has the same processor distribution as X. The other matrices T, A_k, B_k are distributed in the following way:

$$T_{\text{loc}}(1:m_R, 1:n_R) = T(i_R m_R + 1 : (i_R+1)m_R, :) \tag{20}$$

$$(A_{\text{loc}})_k(1:m_\theta, 1:n_\theta) = A_{i_k m_K + k}(i_\theta m_\theta + 1 : (i_\theta+1)m_\theta, :) \tag{21}$$

$$k = 1, ..., m_K$$

$$(B_{\text{loc}})_k(1:m_\theta, 1:n_\theta) = B_{i_k m_K + k}(i_\theta m_\theta + 1 : (i_\theta+1)m_\theta, :) \tag{22}$$

$$k = 0, ..., m_K.$$

The computational scheme begins with processor (i_R, i_θ, i_K) assembling the following quantities:

1. $$Y(i_R m_R + 1 : (i_R+1)m_R, i_\theta m_\theta + 1 : (i_\theta+1)m_\theta, i_K m_K + 1 : (i_K+1)m_K)$$
$$= \tilde{T}_{\text{loc}}(X(:, i_\theta m_\theta + 1 : (i_\theta+1)m_\theta, i_K m_K + 1 : (i_K+1)m_K)) \tag{23}$$

2. $$Y(i_R m_R + 1 : (i_R+1)m_R, i_\theta m_\theta + 1 : (i_\theta+1)m_\theta, i_K m_K + 1 : (i_K+1)m_K)$$
$$= \sum_{k=i_K m_K+1}^{(i_K+1)m_K} (\tilde{A}_{\text{loc}})_k(X(i_R m_R + 1 : (i_R+1)m_R, :, k)) \tag{24}$$

3. $$Y(i_R m_R + 1 : (i_R+1)m_R, i_\theta m_\theta + 1 : (i_\theta+1)m_\theta, i_K m_K + 2 : (i_K+1)m_K + 1)$$
$$= \sum_{k=i_K m_K+1}^{(i_K+1)m_K} (\tilde{B}_{\text{loc}})_k(X(i_R m_R + 1 : (i_R+1)m_R, :, k)) \tag{25}$$

4. $$Y(i_R m_R + 1 : (i_R+1)m_R, i_\theta m_\theta + 1 : (i_\theta+1)m_\theta, i_K m_K : (i_K+1)m_K - 1)$$
$$= \sum_{k=i_K m_K+1}^{(i_K+1)m_K} (\tilde{B}'_{\text{loc}})_{k-1}(X(i_R m_R + 1 : (i_R+1)m_R, :, k)). \tag{26}$$

The results of Eqs. (23)–(26) are used to assemble Y of Eq. (13).

Since the computations implied by Eqs. (23)–(26) involve not only local array elements, data communication is necessary in order to complete these sums. In order for processor (I_R, i_θ, i_K) to calculate Eq. (23), it needs the portions of X that reside on p_R different processors $(1 \cdots p_R, i_\theta, i_K)$; and to calculate Eqs. (24)–(26), it needs the values of X that reside on p_θ different processors $(i_R, 1 \cdots p_\theta, i_K)$.

All the communications at this level are carried out in the following fashion. If processor i_1 needs the values of X from processor i_2, ..., i_h, it will first use the "shmen_get" routine to obtain the values of X from processor i_2. Then, processor i_1 uses i_2's portion of X to continue its summation. When this calculation is completed, processor i_1 will get the values of X that reside on processor i_3. This procedure continues until processor i_1 finishes all of the computations required by Eqs. (23)–(26).

We use the "shmem_get" routine in our implementation instead of the "faster" shared memory access routines (i.e., "shmem_put" routine) because "shmem_get" can perform safe data communications without synchronization between the processors. One difference between the CRAY T3D and other parallel computers (e.g., the Intel DELTA) is that one cannot control the underlying communication network on the T3D. Although our algorithm assumes that processors sit "logically" on a 3D torus, in practice the physical communication network can be very different from that of a real 3D torus. This might cause serious communication congestion problems because some communications might require the same key links at the same time. We minimize the chance of having different communications requiring the same key links by using asynchronous communication, so that few communications will overlap in time. The routines that make the asynchronous communication *safe* are the *blocking* routines such as "shmem_get." Our tests will examine and show that the performance of our implementation is efficient and reliable.

After the sums required by Eqs. (23)–(26) are completed, there is still one additional required step. Note that the indices of Y in Eq. (25) are

$$Y(i_R m_R + 1 : (i_R + 1)m_R, i_\theta m_\theta + 1 : (i_\theta + 1)m_\theta, i_K m_K + 2 : (i_K + 1)m_K + 1),$$

instead of

$$Y(i_R m_R + 1 : (i_R + 1)m_R, i_\theta m_\theta + 1 : (i_\theta + 1)m_\theta, i_K m_K + 1 : (i_K + 1)m_K).$$

In order to complete the required sums, processor (i_R, i_θ, i_K) must pass part of its Eq. (25) results to processor $(i_R, i_\theta, i_K + 1)$, except when i_K is not equal to $p_K - 1$. Similarly, processor (i_R, i_θ, i_K) needs to pass part of its Eq. (26) results to processor $(i_R, i_\theta, i_K - 1)$, unless i_K equals 0. These communications are performed using the "shmem_put" shared memory access routine.

V. SEQUENTIAL DIAGONALIZATION AND TRUNCATION

The sequential diagonalization and truncation (SDT) transformation can be used to reduce significantly the order of the matrix that one is working with to obtain the eigenpairs of interest. Rather than work directly on the DVR Hamiltonian to obtain the needed eigenpairs, the SDT method begins with a series of diagonalizations of lower dimensional problems followed by a truncation of the eigenvector representation based on specified criteria to ensure accuracy of the final result but yet reduce the dimension of the transformed matrix as much as possible. In the traditional implementation [17] the sparsity of the DVR matrix is not preserved. So the trade-off from a computational perspective is between a higher order sparse

DVR-matrix and a lower order dense SDT-matrix. In earlier work, Pendergast *et al.* [21] showed that if one is using an iterative eigen solver such as the IRLM, it is not necessary or desirable to explicitly carry out the transformation from the DVR to the SDT matrix representation. Since iterative methods only require that matrix–vector products be assembled, there is no need to transform the DVR matrix explicitly. Instead one can convert the vectors back and forth from the SDT to the DVR representation as necessary. Another benefit of working with the SDT matrix is that the eigenvalue spectrum in this truncated representation is much better conditioned for an iterative procedure like IRLM.

In the van der Waals case presented here, the SDT process is carried out in two stages. First, one forms a block diagonal matrix h which contains the diagonal blocks of the full Hamiltonian of Eq. (1),

$$\begin{aligned}\mathscr{H}_{\alpha\beta k}^{\alpha'\beta'k'} &= h_{\alpha'\alpha}^{(\beta k)}\delta_{\beta'\beta}\delta_{k'k} \\ &= T_{\alpha'\alpha}\cdot\delta_{\beta'\beta}\cdot\delta_{k'k} + V(R_\alpha, \theta_{k\beta})\cdot\delta_{\alpha'\alpha}\cdot\delta_{\beta'\beta}\cdot\delta_{k'k} \\ &\quad + \frac{\hbar \mathrm{h}^2}{2\mu R_\alpha^2}[J(J+1) - 2k^2]\delta_{\alpha'\alpha}\cdot\delta_{\beta'\beta}\cdot\delta_{k'k}. \end{aligned} \tag{27}$$

Then each diagonal block of $\mathscr{H}$ is diagonalized using a standard EISPACK or LAPACK diagonalizer to give

$$h^{(\beta k)} = C^{(\beta k)}\cdot E^{(\beta k)}\cdot [C^{(\beta k)}]^T$$

in which $C^{(\beta k)}$ is a $N_R \times N_R$ 1D-eigenvector matrix, and $E^{(\beta k)}$ is a diagonal matrix containing eigenvalues of $h^{(\beta k)}$ for each (β, k) blocks. In the SDT approach during this operation the high energy eigenvectors in $C^{(\beta k)}$ are truncated to speed up the diagonalization either by retaining a constant number of vectors or by retaining the states $C^{(\beta k)}$ that satisfy an energy cutoff condition such that

$$E^{(\beta k)} \le E_{\mathrm{cut}}^{1D}.$$

The truncated $C^{(\beta k)}$ matrix, denoted by $\tilde{C}^{(\beta k)}$ has a reduced dimension $N_R \times P_{\beta k}$, where $P_{\beta k}$ corresponds to the number of 1D eigenvectors that satisfy the required cutoff limit in the corresponding (β, k) block.

At this stage the original 3D DVR Hamiltonian matrix is transformed to a truncated Hamiltonian $\tilde{H}^{\mathrm{SDT}}$ in the representation of truncated 1D-eigenvectors as

$$\begin{aligned}\tilde{H}_{l\beta k}^{l'\beta'k'} &= \sum_{\alpha'\alpha}[\tilde{C}_{\alpha' l'}^{(\beta' k')}]^T\cdot H_{\alpha\beta k}^{\alpha'\beta'k'}\cdot\tilde{C}_{\alpha l}^{(\beta k)} \\ &= E_l^{\beta k}\cdot\delta_{l'l}\cdot\delta_{\beta'\beta}\cdot\delta_{k'k} + \sum_{\alpha'\alpha}[\tilde{C}_{\alpha' l'}^{(\beta' k')}]^T\cdot\left\{\left(\frac{1}{R_\alpha^2}+\frac{1}{r_e^2}\right)[\mathscr{A}_k]_{\beta'\beta}\cdot\delta_{\alpha'\alpha}\right\}\cdot\delta_{k'k}\cdot\tilde{C}_{\alpha l}^{(\beta k)} \\ &\quad + \sum_{\alpha'\alpha}[\tilde{C}_{\alpha' l'}^{(\beta' k')}]^T\cdot\left\{\frac{1}{R_\alpha^2}[-(1+\delta_{k0})^{1/2}\Lambda_{Jk}^{+}[\mathscr{B}_k]_{\beta'\beta}^{+}\cdot\delta_{k'k+1}\right. \\ &\quad \left. - (1+\delta_{k'0})^{1/2}\Lambda_{Jk}^{-}[\mathscr{B}_k]_{\beta'\beta}^{-}\cdot\delta_{k'k-1}]\right\}\cdot\tilde{C}_{\alpha l}^{(\beta k)}. \end{aligned}$$

In each $(\beta k, \beta' k')$ block, the transformation has reduced the $N_R \times N_R$ matrix to a $(P_{\beta k} \times P_{\beta' k'})$ matrix of 1D-eigenvector basis. Therefore,

$$\tilde{H}^{\mathrm{SDT}} = E + \left[\tilde{C}^T \left(\sum_k E^{K,K}_{k,k} \otimes A_k \otimes D_1 + \sum_k E^{K,K}_{k,k+1} \otimes B_k \otimes D_2 + \sum_k E^{K,K}_{k+1,k} \otimes B^T_k \otimes D_2 \right) \tilde{C} \right]. \tag{28}$$

So far the steps in the SDT transformation of Light and co-workers [17] and our IRLM/SDT approach are basically the same. However, in the next step instead of using a conventional diagonalizer to diagonalize the $\tilde{H}^{\mathrm{SDT}}$ we use the IRLM approach. This technique was first introduced by Pendergast *et al.* [21] in the 2D surface eigenfunction calculations needed by the APH hyperspherical coordinate approach by Parker and Pack [22]. Recently Korambath *et al.* [13] have found that this implicit SDT transformation to be effective for rovibrational eigen problems using sequential computers and Wu and Hayes [23] have applied the implicit SDT technique to obtain the HO_2 rovibrational energy levels on a CRAY T3D.

The key to calculating the eigenpairs of $\tilde{H}^{\mathrm{SDT}}$ is to calculate $u = \tilde{H}^{\mathrm{SDT}} w$, where u, w are vectors. The first step is to transfer the vector w from the SDT representation to the DVR representation by

$$x = \tilde{C} w. \tag{29}$$

One can map x and w into 3D arrays X and W, where x and X, w and W are related through Eq. (14). Now Eq. (29) can be rewritten as

$$X(:, \beta, k) = \tilde{C}^{(\beta k)} W(:, \beta, k) \tag{30}$$

because $\tilde{C}$ is a block diagonal matrix.

In the next step, one obtains

$$y = \left(\sum_k E^{K,K}_{k,k} \otimes A_k \otimes D_1 + \sum_k E^{K,K}_{k,k+1} \otimes B_k \otimes D_2 + \sum_k E^{K,K}_{k+1,k} \otimes B^T_k \otimes D_2 \right) x. \tag{31}$$

The procedure is very similar to that presented in Sections III and IV.

Finally, the product is converted back to the SDT representation and combined with the action of the first term in Eq. (28) on the initial vector, w, as

$$u = \tilde{C}^T y + E \odot w. \tag{32}$$

If one maps u and y into U and Y, following Eq. (14), this equation can be written as

$$U(:, \beta, k) = E(:, \beta, k) \odot W(:, \beta, k) + \tilde{C}^{(\beta k)T} Y(:, \beta, k). \tag{33}$$

The Sylvester transformation can also be employed in the SDT case to connect the matrix–vector operation in Eq. (31) to a series of lower order matrix–matrix operations.

The implementation of the SDT procedure on a parallel machine follows the same general approach as that presented previously for the DVR procedure. The main difference is that there are no partitions in the first dimension of X, Y, U, and W. On the other hand, the θ and K dimensions are partitioned in the same way for X, Y, U, and W arrays. On the processor (i_θ, i_K),

$$X_{\text{loc}}(:, 1:m_\theta, 1:m_K) = X(:, i_\theta m_\theta + 1 : (i_\theta + 1)m_\theta, i_K m_K + 1 : (i_K + 1)m_K) \quad (34)$$

and relationship between Y_{loc} and Y, U_{loc} and U, W_{loc} and W can be obtained by simply replacing X with Y, U, or W.

The last array that needs to be partitioned is $\tilde{C}$. On the processor (i_θ, i_K),

$$\tilde{C}_{\text{loc}}^{(1:m_\theta)(1:m_K)} = \tilde{C}^{(i_\theta m_\theta + 1 : (i_\theta + 1)m_\theta)(i_K m_K + 1 : (i_K + 1)m_K)}. \quad (35)$$

It is obvious that under the above partition, Eqs. (30) and (33) use only the *local* data. The only other required calculation is Eq. (31), whose parallel implementation is presented in Section IV.

VII. POLYNOMIAL PRECONDITIONING

Although an efficient matrix–vector product implementation is one of the most important factors in obtaining good performance, the computation time for the IRLM scales linearly with the number of iterations needed to obtain convergence. In general there are two factors that influence significantly the number of iterations necessary to achieve convergence. The first is the number of eigenvalues being sought. The second is the distribution of all the eigenvalues of the matrix. The IRLM takes fewer iterations when the desired eigenvalues are well separated. In this application, we need to obtain eigen-solutions for the lower end of the eigenvalue spectrum. This lower-energy part of the spectrum is tightly clustered, compared to the higher energy part of the eigenvalue spectrum. This is the opposite of what is desired for rapid convergence. A standard approach fixing this and accelerating iterative eigensolvers is to use a preconditioner such as a Chebychev preconditioner [24].

In our previous work [21], we found that Chebychev preconditioning can be used effectively to alter the eigen-problem so that the IRLM will converge rapidly to the desired eigenvectors. Since the eigenvectors for a polynomial of the Hamiltonian are the same as the eigenvectors of the original Hamiltonian one can use these converged eigenvectors to generate the desired eigenvalues using the Raleigh–Ritz quotient. Details to arrive at the optimum order for the Chebychev polynomial to get maximum speedups by using Chebychev preconditioning are given by Pendergast *et al.* [21]. For the present application, we first calculate the eigenvectors of $\phi_n([2H - (a_1 + a_0)]/[a_1 - a_0])$, where ϕ_n is a Chebychev polynomial of nth degree, a_1 is the highest eigenvalue of H, and a_0 is set a little higher than the upper bound

on eigenvalues to be determined. The highest eigenvalue, a_1 is determined by direct calculation without preconditioning. This is a very rapid calculation as the highest eigenvalue is well separated from the rest of the spectrum. As we are interested only in the bound-state eigenvalues below zero, we have found that taking a_0 equal to 0.1 eV works very nicely over a range of bound-state problems.

The product of the Chebychev polynomial and a vector, $\phi_n([2H - (a_1 + a_0)]/[a_1 - a_0])v$ which is needed to calculate the eigenvectors using IRLM is obtained from the 3-term recursion formula for $n \geq 2$,

$$\phi_{n+1}\left(\frac{2H - (a_1 + a_0)}{a_1 - a_0}\right) v = \left(\frac{2H - (a_1 + a_0)}{a_1 - a_0}\right) \times \phi_n \left(\frac{2H - (a_1 + a_0)}{a_1 - a_0}\right) v - \frac{1}{4} \phi_{n-1} \left(\frac{2H - (a_1 + a_0)}{a_1 - a_0}\right) v \tag{36}$$

and for $n < 2$ the recursion formula is

$$\phi_2\left(\frac{2H - (a_1 + a_0)}{a_1 - a_0}\right) v = \left(\frac{2H - (a_1 + a_0)}{a_1 - a_0}\right) \times \phi_1 \left(\frac{2H - (a_1 + a_0)}{a_1 - a_0}\right) v - \frac{1}{2},$$

$$\phi_1\left(\frac{2H - (a_1 + a_0)}{a_1 - a_0}\right) v = \left(\frac{2H - (a_1 + a_0)}{a_1 - a_0}\right) v.$$

We have found this recursion to be efficient and accurate for this application.

VIII. RESULTS AND DISCUSSION

One of the questions that we want to address is whether one can achieve better overall performance on the parallel architecture CRAY T3D by working with the DVR matrix—algorithm 1—or by working with the SDT representation of the Hamiltonian for the system—algorithm 2. In order to have as unbiased a comparison as possible it is important to take advantage of the underlying sparsity of the DVR matrix and use the Sylvester-like transformations to achieve high performance in generating the DVR matrix–vector product. On the CRAY T3D the assembly-coded BLAS routines are known to be effective in achieving high levels of performance on each processor. Moreover, the higher the level of the BLAS routine the better the performance is. In the calculations reported here we have made special efforts to make effective use of these routines. Since the Hamiltonian for the van der Waals molecules studied here has a tensor product form, the most time-consuming operations can be carried out using level-3 or level-2 BLAS operations. With this background, we can now take a closer look at the computational trade-off between the two algorithms.

For the Ar-OH van der Waals complex, we have carried out numerous comparisons of the DVR and SDT algorithms (algorithm 1 and algorithm 2, respectively). For both cases the underlying DVR representation is the same, namely $n_R = 96$ and $n_\theta = 24$.

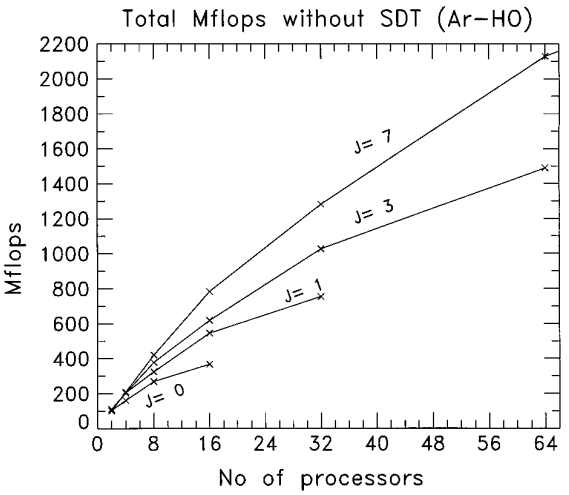

FIG. 1. Total Mflops for diagonalizing the Hamiltonian for Ar–HO molecule for $J = 0, 1, 3, 7$ without sequential diagonalization and truncation on different number of processors (number of radial functions $N_R = 96$, number of angular functions $N_\theta = 24$, number of eigenvalues = 12).

In Fig. 1, we plot the total Mflops of the DVR approach versus the number of processors for several different values of J. In Fig. 2, we have same kind of plot for the SDT approach. Both plots show almost linear performance up to 32 processors. The DVR algorithm achieves higher Mflops compared to the SDT algorithm. However, the total computation time for algorithm 1 is about twice the algorithm 2 computation time. So we have a situation in which the algorithm with the lower

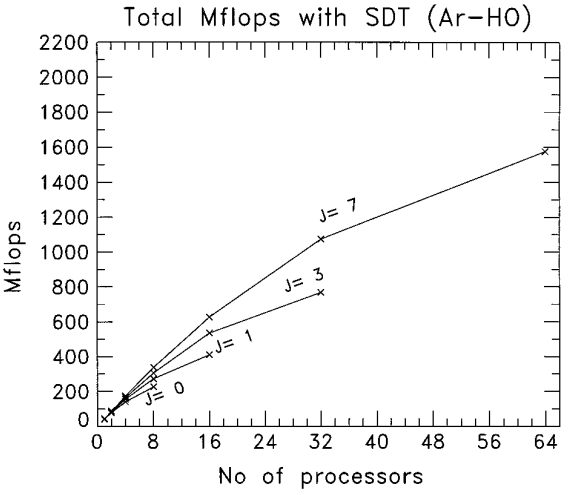

FIG. 2. Total Mflops for diagonalizing the Hamiltonian for Ar–HO molecule for $J = 0, 1, 3, 7$ with sequential diagonalization and truncation on different number of processors. Dimensions of the Hamiltonian and the other computational parameters are same as in Fig. 1 (number of truncated radial functions $P_{\beta k} = 72$).

TABLE I
Comparison of Percentage Time for Ar–HO System for Three BLAS Levels as well as Communication Overhead Time and Other Time[a]

Method	# PE	% Blas 3	% Blas 2	% Blas 1	% Comm.	% other
SDT/DVR/IRLM	2	16.4	72.2	6.3	0.0	5.1
SDT/DVR/IRLM	4	14.4	65.3	9.3	6.7	3.9
SDT/DVR/IRLM	8	16.2	60.5	10.2	8.8	4.0
SDT/DVR/IRLM	16	21.8	52.5	7.4	14.4	2.5
SDT/DVR/IRLM	32	31.4	39.4	3.1	20.6	1.6
DVR/IRLM	2	72.3	—	19.5	0.9	7.1
DVR/IRLM	4	58.2	—	20.2	15.5	6.0
DVR/IRLM	8	56.0	—	17.8	13.8	5.6
DVR/IRLM	16	54.7	—	14.2	25.9	4.8
DVR/IRLM	32	48.0	—	12.2	27.5	4.0
DVR/IRLM	64	57.5	—	8.3	28.4	3.2

[a] Total number of radial basis functions (N_R = 96 Lobatto functions defined in the range $R_{\min}$ = 2.0 Å, and $R_{\max}$ = 10.0 Å); the total angular functions (N_θ) = 24 and J = 3 truncated radial functions (for SDT) ($P_{\beta k}$ = 72).

Mflops performance is the most effective in terms of total run time. In the next few paragraphs we will provide some insight into why this is the case.

One factor contributing to the lower average Mflops for the SDT algorithm is the transformation of the SDT vector into a DVR vector. This step is dominated by level 2 BLAS operations. Table I contains a comparison of the percentage of the computation time required for each of the three BLAS levels as well as the communication overhead time and other computational time. These percentages were obtained using Apprentice to evaluate the performance of the matrix–vector program for the two cases. First, we note that for the DVR algorithm there are no level-2 BLAS operations. For the SDT algorithm the percentage of level-3 BLAS is reduced because the level-2 BLAS operations are taking an appreciable fraction of the matrix–vector processing time. For any given number of processors the percentage of communication time is about the same for the two cases. Since the performance of the level-3 BLAS algorithm exceeds that of the level-2 BLAS by about a factor of 2, the difference in the performance level of the two algorithms is almost entirely due to the difference in the percentage of level-3 BLAS operations.

The advantage that the SDT algorithm has can be seen in Table II, where the two algorithms are compared in terms of their total computation time and the number of matrix–vector operations required by IRLM to converge on the lowest 12 eigenpairs. Basically, the larger DVR matrix requires about 4 to 5 times the number of matrix–vector operations required by the smaller SDT matrix. This effect is much greater than the ratio of the Mflops performance. As a result, the time required to obtain the 12 eigenpairs on the same number of processors is decreased by about one-half for the SDT case compared to the DVR case. In this example, the R dimension is reduced from $n_R = 96$ to $P_{\beta k} = 72$ by the SDT process.

TABLE II
Diagonalization Time on CRAY T3D for Ar–HO Molecule for Several Values of the Total Angular Momentum, *J*, and Number of Processors[a]

		Without SDT		With SDT		
J value	No. processors	M × V[b]	Time[c]	M × V[b]	Time[c]	Time ratio
0	2		6.660		4.200	0.630
0	4		3.998		2.374	0.593
0	8	1250	3.180	348	1.500	0.471
1	2		12.580		6.748	0.536
1	4		6.964		3.584	0.514
1	8	1040	3.979	264	2.080	0.522
1	16		2.442		1.400	0.573
3	2		27.759		14.659	0.528
3	4		14.866		7.996	0.537
3	8	1040	7.844	288	4.256	0.542
3	16		4.652		2.490	0.535
3	32		2.823		1.704	0.603
7	2		64.769		33.826	0.522
7	4		31.745		16.891	0.532
7	8	1100	15.779	320	8.833	0.559
7	16		8.307		4.712	0.567
7	32		4.952		2.777	0.560
7	64		3.025		1.907	0.603

[a] Number of radial basis functions (N_R = 96 Lobatto functions defined in the range $R_{\min}$ = 2.0 Å, and $R_{\max}$ = 10.0 Å); number of angular functions (N_θ) = 24; and truncated radial functions ($P_{\beta k}$ = 72); number of eigenvalues = 12.
[b] M × V is the total number of matrix vector operations.
[c] Time in seconds.

In this example, we were able to obtain the required eigenpairs for a matrix of size 9216 × 9216 in 1.7 s. While this has some significance, the power of these two algorithms will become even greater when they are applied to systems with more degrees of freedom. Part of the reason for this is that the performance of the level-3 BLAS library routines such as ''sgemm'' is strongly related to the size of the matrices involved. The most favorable matrix size for ''sgemm'' is around 60 × 60, for this case the performance of ''sgemm'' can be over 90 Mflops for a single processor on the CRAY T3D. On the other hand, if the matrix size becomes very small (<10 × 10), the performance of ''sgemm'' can be reduced by a factor of 3 or more. Because the number of degrees of freedom in this study is quite small (3), and the number of basis function in each degree of freedom is also small (n_R = 96, n_θ = 24, K = 1, ..., 8), the size of the *local* matrices is about 10 when 64 or 128 processors are used. Therefore, the Mflops rating of ''sgemm'' is in the 20 to 40 Mflops range. However, for a system with more degrees of freedom such as a 4-atom system, the local matrices will be large enough even when over 100 processors are used that the performance of ''sgemm'' will be about 90 Mflops.

Another obstacle to achieving high performance is the data communication between the processors. In general, there are two keys to reducing the impact of data communication on the performance: (1) reducing the number of communications when designing the algorithm; (2) making each communication as efficient as possible. The implementation, presented in Section IV is very effective in reducing the number of communications involved. Here, we present one example from the DVR algorithm to make the point clearer. For $n_R = 96$, $n_\theta = 24$, and $K = 8$ case, we did the following partition in the 32-processor calculation: We partitioned n_R direction into four sectors, we did not partition the n_θ direction, and we partitioned K into eight sectors. With this partition scheme, five communications are performed during each matrix–vector operation. Of these, three communications are performed when computing Eq. (23), and two communications are performed to pass the results of Eqs. (25) and (26).

To make each communication as efficient as possible, we want to avoid communication congestion to the maximum extent possible. As pointed out in Section IV, the underlying physical communication network on the CRAY T3D is different from our ''logical'' network, therefore directly applying some communication templates of multidimensional meshes will not achieve favorable results on the CRAY T3D. The way we minimize communication congestion is to avoid many processors communicating at the same time. In our implementation we use asynchronous communications instead of synchronous communications to minimize congestion. The above strategy works very well in this study.

With the insights obtained from the application of the DVR and SDT algorithm to the van der Waals molecule Ar–HO, the SDT algorithm appears to be the most promising for applications to rovibrational problems involving additional degrees of freedom, provided one can reliably set the energy cutoff condition, E_{cut}^{1D}, without extensive testing for convergence.

ACKNOWLEDGMENTS

Partial support for this project came from National Science Foundation projects, ASC-9112693 and ASC-9408795; additional support was also provided by the National Science Foundation in the form of a postdoctoral fellowship grants ASC-9405161 and ASC-9504071. Computer time on the CRAY T3D was provided by a grant from the Ohio Supercomputer Center.

REFERENCES

1. D. Neuhauser, *J. Chem. Phys.* **93,** 2611 (1990).
2. Y. Wang, T. Carrington Jr., and G. C. Corey, *Chem. Phys. Lett.* **228,** 144 (1994).
3. D. J. Kouri, W. Zhu, G. Parker, and D. K. Hoffman, *Chem. Phys. Lett.* **238,** 395 (1995).
4. V. A. Mandelshtam, T. P. Grozdanov, and H. S. Taylor, *J. Chem. Phys.* **103,** 10074 (1995).
5. R. A. Friesner, J. A. Bentley, M. Menou, and C. Leforestier, *J. Chem. Phys.* **99,** 324 (1993).
6. R. E. Wyatt, *Phys. Rev.* **51,** 3643 (1995).
7. B. C. Chang, L. Yu, D. Cullin, B. Rehfuss, J. Williamson, T. A. Miller, W. M. Fawzy, X. Zheng, S. Fei, and M. C. Heaven, *J. Chem. Phys.* **95,** 7086 (1991).
8. M. C. Heaven, *Ann. Rev. Phys. Chem.* **43,** 283 (1992) and references therein.

9. M.-L. Dubernet and J. M. Hutson, *J. Chem. Phys.* **99,** 7477 (1993).
10. M. Yang and M. H. Alexander, *J. Chem. Phys.* **103,** 3400 (1995).
11. T. S. Ho, H. Rabitz, S. E. Choi, and M. I. Lester, *J. Chem. Phys.* **104,** 1187 (1996).
12. M. I. Lester, R. A. Loomis, L. C. Giancarlo, M. T. Berry, C. Chakravarthy, and D. C. Clary, *J. Chem. Phys.* **98,** 9320 (1993).
13. P. P. Korambath, X. T. Wu, E. F. Hayes, C. C. Carter, and T. A. Miller, *J. Chem. Phys*. **107,** 3460 (1997).
14. P. P. Korambath, X. T. Wu, and E. F. Hayes, *J. Phys. Chem.* **100,** 6116 (1996).
15. J. V. Lill, G. A. Parker, and J. C. Light, *Chem. Phys. Lett.* **89,** 483 (1982). [J. C. Light, I. P. Hamilton, and J. V. Lill, *J. Chem. Phys.* **82,** 1400 (1984); J. V. Lill, G. A. Parker, and J. C. Light, *J. Chem. Phys.* **85,** 900 (1986)]
16. D. C. Sorensen, *SIAM J. Matrix Anal. Appl.* **13,** 357 (1992).
17. Z. Bačić, R. M. Whitnell, D. Brown, and J. C. Light, *Comput. Phys. Commun.* **51,** 35 (1988). [R. M. Whitnell and J. C. Light, *J. Chem. Phys.* **90,** 1774 (1989); T. J. Park and J. C. Light, *J. Chem. Phys.* **90,** 2593 (1989)]
18. G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed. (Johns Hopkins Univ. Press, Baltimore, MD, 1989), p. 387.
19. S. E. Choi and J. C. Light, *J. Chem. Phys.* **92,** 2129 (1990).
20. D. Y. Hu and D. C. Sorensen, Dept. Computational and Applied Math Report TR94-10, Rice University, 1994.
21. P. Pendergast, Z. Darakjian, E. F. Hayes, and D. C. Sorensen, *J. Comput. Phys.* **113,** 201 (1994).
22. R. T. Pack and G. A. Parker, *J. Chem. Phys.* **87,** 3888 (1987).
23. X. T. Wu and E. F. Hayes, *J. Chem. Phys*. **107,** 2705 (1997).
24. Y. Saad, *Math. Comput.* **42,** 567 (1984) and the references therein.