239

MOLECULAR ROTATION-VIBRATION CALCULATIONS USING MASSIVELY PARALLEL COMPUTERS

Hamse Y. Mussa,[1] Jonathan Tennyson,[1] C.J. Noble,[2] and R.J. Allan[2]

[1]Department of Physics and Astronomy, University College London, London WC1E 6BT, UK
[2]Daresbury Laboratory, Daresbury, Warrington WA4 4AD, UK

Introduction

Calculations of rotation-vibration spectra of small molecules by direct solution of the nuclear motion Schrödinger equation are beginning to make a major impact in the area of experimental spectroscopy[1]. However, although the techniques used to solve these problems are now well developed, there are applications, even for three atom systems on a single potential energy surface, which present a formidable computational challenge. Such systems may possess $10^5 - 10^6$ bound states.

One class of methods which have proved very successful at treating large nuclear motion problems are based on a particular finite element representation of the problem known as the Discrete Variable Representation[2]. Here we describe how the DVR3D package of Tennyson and co-workers[3], has been ported to massively parallel computers in manner appropriate for tackling large problems. Wu and Hayes [4] have also addressed the problem of performing rotation-vibration calculations for three atom systems on a Cray T3D machine. Our method is designed to tackle larger problems than theirs and is therefore represents a more radical change from algorithms used on sequential machines. A preliminary version of the algorithm discussed here has been published[5].

PDVR3DR : an overview

The DVR3D program suite[3] contains a number of modules. Program DVR3DRJ solves the rotationless $J = 0$ vibrational problems using either atom-diatom Jacobi coordinates or Radau coordinates. It also solves the effective 'vibrational' problems required for a two-step treatment of the rotational excitation problem[6]. Programs ROTLEV3 and ROTLEV3B are driven by DVR3DRJ and solve the fully coupled rotation-vibration problem respectively for Jacobi and a symmetrized grid in Radau coordinates. A final module, DIPOLE3, computes transition dipoles but will not concern us here.

In this article parallel versions of the programs DVR3DRJ and ROTLEV3B are presented. However parallelization requires a number of changes in the solution algorithm which were best achieved by splitting program DVR3DRJ. The parallel version thus consists of PDVR3DR and PROTLEV3R which solve the nuclear motion problem for symmetrized Radau coordinates with a bisector embedding[7], and PDVR3DJ and PROTLEV3J which are appropriate for Jacobi coordinates. Parallelizing the Radau co-ordinate program is the more difficult task and the only one we discuss here.

When determining parallelization procedures it is necessary to consider both the construction of the scientific problem being addressed and the architecture of the computer being employed. The codes themselves are not molecule or problem specific but it is helpful to use specific examples as these define typical limits of a calculation. Here we consider water at its dissociation limits as the prototype system to be treated in symmetrized Radau coordinates. The DVR3D programs have been adapted relatively easily to run very efficiently on parallel shared memory architectures. However these machines have few processors and the benefits in throughput are limited.

The intended architecture is a parallel message passing computer. PDVR3DR and PROTLEV3R are designed for parallel and distributed memory computers which use a Single Program Multiple Data (SPMD) loading strategy. Particular architectures used are the 26-processor IBM SP2 machine at Daresbury Laboratory, the Edinburgh Parallel Computer Centre (EPCC) Cray Machines – the T3D which has 512 processors, and the T3E which has 256 application processors –, and finally the 128 processor Cray-T3E in Bologna. For algorithmic reasons, described below, PDVR3DR program is usually run on 64 or 128 processors of the T3D, 16 to 64 of the EPCC Cray-T3E, 8 to 16 of the Duresbury IBM SP2 and 16 to 128 of the Bologna Cray-T3E while PROTLEV3R can use any number of processors.

Originally the programs were written using PVM[8] but we switched to MPI[9] when this became available. The DVR3D program suite was written in FORTRAN77, but modifications use Fortran 90 constructs.

The T3D and the SP2 were used for development purposes. Since the intention of the work is to build and diagonalize large Hamiltonian matrices, our work has been done mainly on the T3D and T3E's. The results presented here are for the Cray-T3D, similar results were obtained on the IBM SP2 when the size of the problem matched the available memory.

Vibrational Calculations

For an AB_2 molecule, Radau coordinates[10] consist of two symmetry related stretching coordinates and an included angle. For molecules, such as water, where the central atom is significantly heavier than the others ($m_A >> m_B$), these coordinates are similar to those given by the two AB bond distances, r_1 and r_2, and the $A\hat{B}A$ bond angle, θ. Coordinates (r_1, r_2, θ) are represented using DVR grids (α, β, γ).

The DVR3D programs use sequential diagonalization and truncation[2, 3] to create a compact final Hamiltonian matrix. For such schemes, the order in which this diagonalization and truncation is performed can be crucial in determining computational requirements[11]. Because the Radau coordinate symmetrization procedure used by DVR3DRJ is based on symmetrizing DVR grid points and not basis functions[12], the stretches need to be treated equivalently and hence together, and the logical choice is to treat the angular θ coordinate last.

The basic method of distributing the DVR calculation over N processors was to place $\frac{n_\gamma}{N}$ (n_γ = number of angular grid points) on each processor. Since $\frac{n_\gamma}{N} = 1$ is faster than $\frac{n_\gamma}{N} > 1$, this coordinate was usually represented by either $n_\gamma = 64$ or 128 'active' grid points, but calcualtions where $n_\gamma = 48, 80, 96$ have also been performed. In the standard energy-selected diagonalization and truncation procedure, the number of functions used to construct the final Hamiltonian varies with γ. This algorithm leads to an poorly load-balanced calculation, so the selection criterion was modified. In the modified version, an equal number of eigenstates of the two-dimensional problem, n^{2D}, were retained for each angular 'active' grid point. Inactive grid points were simply dropped from the calculation. One could include all γ grid points in a calculation, but for water geometries near linear OHH configurations are very high energy, and indeed the Tennyson and Sutcliffe[7] strategy for implementing the bisector embedding involves dropping at least the point nearest linearity. In practice our water calculations drop about 10% of the points, but tests we have performed on ozone involved dropping more than half. For ozone points are dropped from both ends of the angular grid.

Construction and diagonalization of intermediate, 2D Hamiltonians is performed simultaneously on each processor using a standard diagonalizer. We use NAG routine F02ABF [13] for this purpose. In this approach, labelled SDTA for Sequential Diagonalization Truncation Approach, the size of the final Hamiltonian, $\mathbf{H^{SDTA}}$, is given by

$$N^{3D} = n^{2D} \times n_\gamma \tag{1}$$

and

$$\mathbf{H^{SDTA}} = \mathbf{E^{2D}} + \mathbf{C^{(2D)}LC^{(2D)T}} \tag{2}$$

Where $\mathbf{E^{2D}}$ are the eigenvalues of the 2D Hamiltonian in (α, β) at each angular grid point γ and $\mathbf{C^{(2D)}}$ are the corresponding eigenvectors. $\mathbf{L}$ represents the angular kinetic energy terms[3, 12].

Construction of the final 3D Hamiltonian matrix requires all the vectors, $C_j^{2D}(\alpha\beta : \gamma), j = 1, n^{2D}$, obtained for the n_γ active angular grid points. A strip of the Hamiltonian is constructed on each processor which guarantees that one set of vectors is always correctly located. For blocks off-diagonal in γ, a second set of vectors has to be imported from another processor if they are not locally available.

Time-wise the most critical step is diagonalization of the final 3D Hamiltonian. A survey of available parallel diagonalizers has been conducted by Allan[14] and discussed by us[5].

Standard real symmetric diagonalisers, of which PeIGS[15] proved the best, limit the size of calculation which could be attempted due particularly to workspace considerations[5]. We therefore explored the possibility of using iterative diagonalisation procedure. Such an approach has been advocated by a number of other workers[4, 16], although, in contrast to many other studies, we require eigenvectors as well as eigenvalues. The parallel iterative eigensolver PARPACK[17] is designed to obtain a few eigenvalues of a large sparse matrix. To diagonalize the Hamiltonian matrix, $\mathbf{H}$, iteratively, as is usual, one needs to perform the matrix-vector product

$$\mathbf{y} = \mathbf{Hx} \tag{3}$$

However, there are three options as far as this operation is concerned:
a) $\mathbf{H}$ is obtained without any intermediate diagonalisation and truncation. In this approach the Hamiltonian is mainly block diagonal. Operation (3) can therefore be performed without forming $\mathbf{H}$ by using only the non-zero elements since the zero elements do not contribute. This operation is now given by $\mathbf{y} = (\mathbf{elements} > \mathbf{0})\mathbf{x}$ and takes advantage of the natural sparseness of the DVR based Hamiltonian. Such an approach has been used very successfully on sequential machines[16]. It involves treating very sparse matrices of dimension 100,000 or larger. On parallel machines vector $\mathbf{x}$ is distributed over the processors and broadcast to the other processors. BLAS routines are used to give $\mathbf{y}$. This procedure is very efficient and requires little interprocessor

communication.
b) **H** is obtained by the SDTA procedure. It is possible to use the iterative diagonaliser to diagonalize the same matrix as we tested the real symmetric diagonalisers on[5]. **x** is distributed as in (a) and **y** is obtained by using BLAS2 routine sgemv.
c) **x** is distributed as in (a) and **H** is obtained by the SDTA procedure, not formed explicitly. Thus

$$\mathbf{y} = \mathbf{H^{SDTA}x} = (\mathbf{E^{2D}} + \mathbf{C^{(2D)}LC^{(2D)T}})\mathbf{x} \tag{4}$$

$$\mathbf{y} = \mathbf{E^{2D}x} + (\mathbf{C^{(2D)}LC^{(2D)T}})\mathbf{x} \tag{5}$$

One virtue of method (a) is that in its simplest form it is very easy to program. However the symmetrisation of the radial grids points employed by DVR3D presents a significant complication as it destroys much of the structural simplicity of the uncontracted Hamiltonian matrix. This simplicity is important for the efficiency of this algorithm[16]. For this reason we implemented this method without using the radial symmetry. In this case the size of the matrix is given as the product of the number of grid points $n_\alpha n_\beta n_\gamma$. Distributing the matrix as before over the angular grid, γ, there is a local memory balance between the number of eigenvectors required and the size of the matrix stored on each processor. The maximum size for which we performed calculations was for 100 eigenstates of a matrix of dimension 102400. These proved considerably more expensive to obtain than method (b). One reason for the difference is that the speed of PARPACK convergence is dependent on the eigenvalue distribution of the Hamiltonian matrix[4]. For us the speed of convergence is related to the $\mathbf{H^{SDTA}}$ Hamiltonian eigenvalues distribution via

$$r = \frac{\mid \lambda_{suw} - \lambda_{lw} \mid}{\mid \lambda_{max} - \lambda_{suw} \mid} \tag{6}$$

Where *suw* is the smallest unwanted eigenvalue, *lw* is the largest eigenvalue required, and *max* is the largest eigenvalue of the Hamiltonian matrix. Larger r gives faster the convergence. The SDTA affects neither λ_{suw} nor λ_{lw}, but it reduces the λ_{max}. So the SDTA increases r and one can say then that the SDTA procedure not only reduces the size of the Hamiltonian, but is also improves the conditioning of the matrix and by doing so it speeds up convergence.

An advantage of (c) over (a) is faster convergence, and its advantage over (b) is less memory requirement. However, it has a disadvantage over (b) in that the number of matrix-vector operations needed is much bigger. Since the calculation is a trade-off between memory requirement and CPU-time, and our main concern when using PARPACK as the eigensolver is the CPU-time rather than memory, only (a) and (b) have been implemented. Implementing (c) would be straightforward and can be done if needed.

Using the PARPACK to diagonalise the contracted Hamiltonian, method (b), has one immediate advantage over use of PeIGS. PARPACK has little workspace requirement meaning that, in practice, matrices of twice the size could be diagonalized.

Rotational Excitation

Eigenvectors from the vibrational step of the calculation are required as a basis for computing rotationally excited states. It is necessary to transform the eigenvectors obtained from the final STDA diagonalization back to vectors which give the amplitude of the wavefunction on the original, raw grid points. As only h ($< N^{3D}$) eigenvectors need to be transformed, it is necessary to redistribute the eigenvectors so that those required are spread equally between the processors.

The transformation can be written

$$d^{ik}_{\alpha\beta\gamma} = \sum_{j=1}^{n^{2D}} C^{2D}_j(\alpha\beta : \gamma) C^{3D}_i(j,\gamma), \qquad i = 1, 2, 3..., h. \tag{7}$$

The 2D vectors, $\underline{C}^{2D}$ are mapped on to processors using γ as they are generated during the Hamiltonian construction. The 3D vectors, $\underline{C}^{3D}$, are mapped by distributing vectors between processors. The multiplication is performed using MPI broadcast-reduce routines. If one wants to analyse the wavefunctions of the vibration, the results are saved on disk. Since I/O is not parallel, one processor is used to perform the I/O to avoid bottle-necks. When considering states of rotational excitation J, it is necessary to repeat all the above steps for each k, the projection of J on the body-fixed z-axis, required. In general, if both even and odd Wang parity rotational states are to be calculated using the same first step vectors, then $J + 2$ separate calculations are performed, one for $k = 0, 1, \ldots J$ plus an extra $k = 1$ calculation as this must be treated as a special case[7].

Saving the transformed eigenvectors in eq. (7) for each k [5] causes disk space problems for even low J calculations. For example, we have access to 2GB only of disk space and to accurately calculate $J = 2$ up to dissociation requires 3.5 GB for the transformed eigenvectors. Therefore, PDVR3DR has been modified to keep transformed eigenvectors in core and then construct the off-diagonal blocks in k, which are contributed by the Coriolis operators, $\hat{H}_{kk'}$. In addition to overcoming the disk space problem, there is another benefit to this – computationally, there is no limit to how many J's one can calculate provided there is enough CPU-time. However, in real memory terms this costs 30 to 50 MBs extra for a typical calculation.

For symmetrized Radau coordinates, symmetry considerations dictate that the Cartesian axes of the system should be placed so x bisects the angle θ and the z axis is then in the plane of the molecule and perpendicular to x. This is the so-called bisector embedding[7]. In this embedding, the Coriolis coupling links blocks of the matrix labeled by k with those for with $k \pm 1$ and $k \pm 2$[7].

The block construction step consists of a series of transformations of the form

$$B^{kk'}_{ii'} =< k, i|\hat{H}_{kk'}|k', i' >= \sum_{\alpha\beta\gamma\gamma'} d^{ik}_{\alpha\beta\gamma} H_{kk'}(\alpha\beta\gamma\gamma') d^{i'k'}_{\alpha\beta\gamma'} \tag{8}$$

using the vectors created in eq. (7). The Coriolis operators are diagonal in the radial DVR's, $(\alpha\beta)$, but not in the angular DVR, γ. Full details are given elsewhere[3, 7]. The transformation uses h ($< N^{3D}$) eigenvectors from each k calculation[18]. In the present procedure it was assumed that h is the same for all k.

Because a four-dimensional transformation, eq. (8), is used to construct each matrix element, this step can be quite time consuming. It is therefore necessary to consider carefully how one might parallelize it. Goldfield and Gray[19] mapped the triatomic rearrangement reaction problem using k (or Ω) blocks. However in our case there is insufficient memory to treat more than one J at a time and only for large J are there enough k blocks to make distributing over the processors a possibility. Alternatively one can distribute over the stretching coordinates $(\alpha\beta)$ or the angle γ or the eigenvectors h. Parallelizing over $(\alpha\beta)$ is difficult because of the symmetrization. Conversely parallelizing over γ is superficially attractive as this is done in the first step and the number of γ's has already been chosen to map conveniently onto the number of processors. However there are problems with distributing on γ. In this mode, each processor

Figure 1. Structure of the fully coupled Hamiltonian diagonaliseded by PROTLEV3R. Only non-zero elements. those on the diagonal and shaded blocks, for the lower triangle are computed. The blocks are spread over the processors as indicated in the enlargements.

will build contributions to every element of the block, which will have to be assembled on a single processor once constructed. More seriously, a typical set of h vectors $\underline{d}^{ik}$ takes about 0.5 GB of memory, so these vectors must be distributed. This method leads to an excessive amount of inter-processor communication. We therefore distribute the building of the blocks over the M processors by placing a $\frac{h}{M}$ bra and $\frac{h}{M}$ ket vectors on each processor. Each processor thus builds rows (columns) of each block, see eq. (8). This is done using the vectors in local memory for the bra and for portion of the ket and vectors from other processors for the other portion of the ket. These following steps show how this is done:

Step 1: For $k = 0$; create $\mathbf{d}^{k=0}$ using eq. (7).
Step 2: For $k = 1$; create $\mathbf{d}^{k=1}$ using eq. (7).
form $\mathbf{B}^{k=0,k'=1}$ using eq. (8), $k' = k + 1$.
Step 3: For $k = 2$ to J; create $\mathbf{d}^{k}$ using eq. (7).
form $\mathbf{B}^{k,k'}$ using eq. (8), $k' = k + 2$.
replace $\mathbf{d}^{k-2}$ by $\mathbf{d}^{k-1}$.
form $\mathbf{B}^{k,k'}$ using eq. (8). $k' = k + 1$.

This algorithm ensures that not more than three sets of vectors, $\mathbf{d}^{k}$, are retained in memory at one time.

The diagonal elements. which correspond to the eigenvalues generated in each k calculation, and the built blocks, $\mathbf{B}^{k,k'}$, are sent to one processor which then writes them to disk. Both the data transfer between processors and writing on the disk are done in big chunks to avoid any an unnecsssary overheads.

In the final step of the solution procedure, PROTLEV3B, off-diagonal and diagonal elements are read back from the disk. The fully coupled Hamiltonian is then formed by distributing the blocks and the diagonal elements over the processors. The resulting structure, see Fig. 1, is appropriate when the matrix is to be diagonalized iteratively(suitable for both (a) and (b)). This algorithm for treating rotationally excited states is an improvement on our previous procedure[5] because it considerably reduces I/O and disk storage both of which are a serious problem with the old procedure.

Performance

As shown in ref. 5, PDVR3DR scales very well when building the Hamiltonians as the number of processors is increaded. However, it scales less well when diagonalizing the Hamiltonian on 128 processors or more. This is due to that PeIGS(the diagonaliser used for the scaling test)which is based on Householder transformation. There are several reasons for this:

- Most time is spent in the Householder reduction of the Hamiltonian to tridiagonal form.
- Repeated inverse iteration and orthogonalisation is used to give orthogonal eigenvectors. This can result in both poor load balancing in its algorithm and in creating large overheads.
- Finally, a matrix of 3200 is used for the scaling so the message-passing latency must also be an other factor as the number of processors increases. This factor must be less important as the size of the Hamiltonian increases beccause of the trade-off between the latency and the bandwidth.

Conclusion

We have developed parallel programs for treating the vibration-rotation motion of three-atom systems using either Jacobi or Radau coordinates. These programs are based on the published DVR3D program suite[3] which is designed for computers with traditional architectures. Significant algorithmic changes were required, in particular to reduce I/O and disk usage in the original programs and to produce a load balanced algorithm. The new suite shows good scalability and can be used for more challenging calculations. However diagonalization remains a bottleneck in these calculations. Tests of presently available software favour the use of iterative diagonalisation procedures, although further significant improvements in real symmetric matrix diagonalizers could alter this view.

Our parallel DVR suite has enabled us to calculate all the bound vibrational states of water in one wall clock hour using the EPCC Cray T3D. Using the modifications discussed above we have been able to calculate bound rotation-vibration states of water all the way to dissociation for $J > 0$. Previously such calculations took times measured in days or even in weeks. Results for both rotationless and rotationally excited states are being prepared for publication[20].

Acknowledgments

This work was performed as part of the ChemReact High Performance Computing Initiative (HPCI) Consortium. We thank members of HPCI Centre at Daresbury Laboratory for their help.

REFERENCES

1. O.L. Polyansky, N.F. Zobov, S. Viti, J. Tennyson, P.F. Bernath and L. Wallace, Science, 277 (1997) 346.
2. Z. Bacic and J. C. Light, Ann. Rev. Phys. Chem. 40 (1989) 469.
3. J. Tennyson, J.R. Henderson and N.G. Fulton, Computer Phys. Comms. 86 (1995) 175.
4. X.T. Wu and E.F. Hayes, J. Chem. Phys. 107 (1997) 2705.
5. H.Y. Mussa, J. Tennyson, C.J. Noble, R.J. Allan, Computer Phys. Comms. 108 (1998) 29
6. J. Tennyson and B.T. Sutcliffe, Mol. Phys. 58 (1986) 1067.
7. J. Tennyson and B.T. Sutcliffe, Intern. J. Quantum Chem. 42 (1992) 941.
8. "Parallel Virtual Machine". The PVM3 Users' Guide and Reference Manual is available from netlib2.cs.utk.edu/pvm3/ug.ps via anonymous ftp
9. "Message Passing Interface". The MPI Standard is available from netlib2.cs.utk.edu by anonymous ftp. Note: it is a very large document!
10. B.R. Johnson and W.P. Reinhardt, J. Chem. Phys. 85 (1986) 4538.
11. J.R. Henderson, C.R. Le Sueur, S.G. Pavett and J. Tennyson, Computer Phys. Comms. 74 (1993) 193.
12. J. Tennyson and J.R. Henderson, J. Chem. Phys. 91 (1989) 3815.
13. NAG Fortran Library Manual, Mark 17, Vol. 4 (1996).
14. R.J. Allan and I.J. Bush, *Parallel Application Software on High Performance Computers: Parallel Diagonalization Routines.* Edition 3 (Daresbury Laboratory HPCI Centre, 22/8/96) available on WWW URL http://www.dci.clrc.ac.uk/Publications
15. G. Fann, D. Elwood and R.J. Littlefield, PeIGS Parallel Eigensolver System, User Manual available via anonymous ftp from pnl.gov.
16. M.J. Bramley and T. Carrington Jr, J. Chem. Phys. 99 (1993) 8519.
17. K. Maschhoff and D. Sorensen *A portable implementation of ARPACK for distributed memory parallel architectures* Preliminary proceedings, Copper Mountain Conference on Iterative Methods (1996)
18. J. Tennyson, J. Chem. Phys. 98 (1993) 9658.
19. E.M. Goldberg and S.K. Gray, Computer Phys. Comms. 98 (1996) 1.
20. H.Y. Mussa and J. Tennyson, to be published.