Computation of vibrational DVR eigenvalues on Origin 2000*

Adolfo Bastida†, Domingo Giménez‡, Alberto Requena†, José Zúñiga†

Abstract

In this communication we analyse the parallelization on Origin 2000 of a program of Quantum Chemistry to compute vibrational energy levels of triatomic molecules using discrete variable representation DVR basis functions. The parallelism is achieved in three different ways: dynamic assignation of memory, utilization of compiler directives, and parallelization of eigensolvers.

Keywords: Quantum Chemistry, Parallel Computing, Origin 2000.

1 Introduction

During the last twenty years the techniques of variational computation to determine strongly excited vibrational energy levels of triatomic molecules have had a great development [1]. One of the most efficient methods used is the adiabatic DVR method. The method is valid for coordinates of different types, but we will explain how it works for hypersferical coordinates. First of all, we specify the primitive finite bases utilized to generate the functions of discrete base. We call the primitive base FBR $\Phi_l(\rho)$, $\Phi_m(\phi)$ and $\Phi_n(\theta)$. The discrete basis DVR are obtained from the finite basis, diagonalizing the representation of the coordinates corresponding to each one of them. In that way three "small" eigenproblems are solved. The equations of transformation of these base functions are:

$$|\rho_i> \;=\; \sum_{l=1}^{N_\rho} R_{li}\Phi_l(\rho) \tag{1}$$

$$|\phi_j> \;=\; \sum_{m=1}^{N_\phi} S_{mj}\Phi_m(\phi) \tag{2}$$

$$|\theta_k> \;=\; \sum_{n=1}^{N_\theta} T_{nk}\Phi_n(\theta) \tag{3}$$

where $|\rho_i>$, $|\phi_j>$ and $|\theta_k>$ are the DVR functions and R_{li}, S_{mj} and T_{nk} the corresponding transformation coefficients.

The angular coordinate θ is taken as adiabatic coordinate. For each discrete value of the angle θ_k the bidimensional operators $\hat{H}^{2D}(\rho,\phi;\theta_k)$ are defined as

$$\hat{H}^{2D}(\rho,\phi;\theta_k) = -\frac{\hbar^2}{2\mu}\left[\frac{\partial^2}{\partial\rho^2} + \frac{1}{4\rho^2} + \frac{1}{\rho^2}\frac{\partial^2}{\partial\phi^2}\right] + V(\rho,\phi;\theta_k) \tag{4}$$

*Communication for the 5th European Cray/SGI MPP-Workshop, Bologna (Italy), September 9-10, 1999. The experiments have been performed in the Origin 2000 at the European Center of Parallelism of Barcelona (CEPBA).

†Departamento de Química Física, Universidad de Murcia, 30071 Murcia. {bastida,rqna,zuniga}@fcu.um.es

‡Partially supported by Comisión Interministerial de Ciencia y Tecnología, project TIC96-1062-C03-02. Departamento de Informática, Lenguajes y Sistemas Informáticos, Universidad de Murcia, 30071 Murcia. domingo@dif.um.es. http://morgana.dif.um.es/~domingo

The eigenvalue equation for this operator is

$$\hat{H}^{2D}(\rho,\phi;\theta_k)\psi_n^{2D}(\rho,\phi;\theta) = E_n^{2D}(\theta_k)\psi(\rho,\phi;\theta_k) \tag{5}$$

where the eigenfunctions $\psi_n^{2D}(\rho,\phi;\theta_k)$ and the eigenvalues $E_n^{2D}(\theta_k)$ parametrically depend on the discrete points θ_k. To solve this equation the eigenfunctions $\psi_n^{2D}(\rho,\phi;\theta_k)$ are written as linear combination of the bidimensional basis, formed by the products of unidimensional DVR representations of the coordinates ρ and ϕ, what is

$$\psi_n^{2D}(\rho,\phi;\theta_k) = \sum_{i=1}^{N_\rho}\sum_{j=1}^{N_\phi} C_{i,j,n}^{2D}(\theta_k)|\rho_i> |\phi_j> \tag{6}$$

and the corresponding Hamiltonian matrices are diagonalized for each value θ_k (these are "medium" size eigenproblems). The dimension of these matrices is $N_\rho N_\phi$, which can become too high, making the diagonalization impracticable. To avoid this we can exclude in equation (6) those DVR points for which the eigenfunctions $\psi_n^{2D}(\rho,\phi;\theta_k)$ do not take appreciable values. These are the points where the potential energy function has a very high value and the probability density is negligible. A maximum value V_{max} of the potential is then fixed, and only those basis functions $|\rho_i>$, $|\phi_j>$ satisfying the condition

$$V(\rho_i,\phi_j,\theta_k) < V_{max} \tag{7}$$

are used in the calculations. We call $N^{2D}(\theta_k)$ the number of bidimensional basis functions DVR satisfying this condition. This number is different for each value θ_k, and it satisfies $N^{2D}(\theta_k) < N_\rho \times N_\phi$.

Once the bidimensional functions $\psi_n^{2D}(\rho,\phi;\theta_k)$ have been obtained, they are multiplied by the corresponding angular discrete functions $|\theta_k>$ to form the set of functions of tridimensional basis $|\theta_k>\psi_n^{2D}(\rho,\phi;\theta)$, and the complete eigenvalue problem is solved (it is the "high" eigenproblem). The vibrational tridimensional eigenfunctions are written as follows

$$\psi_l^{3D}(\rho,\phi,\theta) = \sum_{k=1}^{N_\theta}\sum_{n=1}^{N^{2D}} D_{k,n,l}|\theta_k>\psi_n^{2D}(\rho,\phi;\theta_k) \tag{8}$$

where $D_{k,n,l}$ are the coefficients of the eigenfunctions which are obtained together with the energy eigenvalues by diagonalizing the Hamiltonian matrix in the basis $|\theta_k>\psi_n^{2D}(\rho,\phi;\theta_k)$. The Hamiltonian matrix elements are given by the formula

$$\begin{aligned} <\theta_{k'}\psi_{n'}^{2D}(\rho,\phi;\theta_{k'})|\hat{H}(\rho,\phi,\theta)|\theta_k\psi_n^{2D}(\rho,\phi;\theta_k)> = -E_n^{2D}\delta_{n,n'} - \frac{\hbar^2}{2\mu} <\theta_{k'}|\frac{1}{\operatorname{sen}\theta}\frac{\partial}{\partial\theta}\operatorname{sen}\theta\frac{\partial}{\partial\theta}|\theta_k> \\ <\psi_{n'}^{2D}(\rho,\phi;\theta_{k'})|\frac{1}{\rho^2\operatorname{sen}^2\phi\cos^2\phi}|\psi_n^{2D}(\rho,\phi;\theta_k)> + <\theta_{k'}\psi_{n'}^{2D}(\rho,\phi;\theta_{k'})|V(\rho,\phi,\theta)|\theta_k\psi_n^{2D}(\rho,\phi;\theta_k)> \end{aligned} \tag{9}$$

where the bidimensional eigenvalue equation (5) has been utilized to simplify.

According to the definition of the discrete tridimensional basis functions, the final size of the Hamiltonian matrix is $\sum_{k=1}^{N_\theta} N^{2D}(\theta_k)$. Here again a selection of the basis functions is necessary since the number of these functions can become very high. To acomplish this we use in the equation (8) only those bidimensional basis functions with eigenenergies $E_n^{2D}(\theta_k)$ lesser than a certain bidimensional energy, i.e.

$$E_n^{2D}(\theta_k) < E_{cut}^{2D} \tag{10}$$

The vibrational eigenfunctions are given finally by

$$\psi_l^{3D}(\rho,\phi,\theta) = \sum_{k,n}^{N^{3D}} D_{k,n,l}|\theta_k>\psi_n^{2D}(\rho,\phi;\theta_k) \tag{11}$$

where the addition is over the pairs of indices k and n satistying (10). We note that the number of these eigenfunctions is considerably smaller than the initial number of discrete base functions ($N_\rho N_\phi N_\theta$).

This method has been recently used by the Quantum Chemistry Research Group of the Chemical-Physics Department at the University of Murcia in order to obtain the complete vibrational spectrum of triatomic molecules [2].

Due to the high computational cost of the method when high dimensional problems are solved, it is interesting the optimization of the method to solve efficiently those problems in moderm parallel machines as the Origin 2000.

2 The sequential progam

First, we will analyse the four more costly parts in the resolution of the problem, and then the parallelization of these parts will be analysed:

- The solution of the "medium" eigenproblems (we call this part D2D). The routine `dsyevx` of LAPACK [3] can be used in the solution of these problems. The problems are independent and they are known when the three "small" initial problems are solved. They can be solved in parallel one problem after another, or some problems can be solved in parallel at the same time. That second option has been selected because more efficiency is obtained in that way. Real symmetric eigenproblems are solved obtaining part of the spectrum: those eigenvalues lesser than the cutting energy V_{max} and the associated eigenvectors.

- The storage of the eigenvalues and eigenvectors obtained in D2D (this part is called A2D). In this part the eigenvalues and eigenvectors are merged with those obtained in previous steps. The eigenvalues are stored ordered, and the eigenvectors and associated quantum numbers are stored in the same order.

- The construction of the final Hamiltonian (we call this part G3D). Each element i, j is obtained from the stored angles i and j. When the angles correspond to the same cutting energy a modified "dot product" of the associated eigenvectors is performed, modifying each product by a value which depends on the quantum number. When the eigenvalues correspond to different cutting energies, the dimension of the eigenvectors are different, and a type of "dot product" is performed, but in this case each element in the first eigenvector is multiplied by an element of the second eigenvector in a position which is computed from the quantum numbers and the angles. Because the matrix is symmetric $n\frac{n+1}{2}$ entries are computed, where n is the number of stored eigenvalues. The computation on each one of these "dot products" is proportional to the dimension of the vectors (m), and the cost has an order $\frac{n^2}{2}m$, and the number of floating point operations on each step is 12.

- The solution of the eigenproblem associated to the Hamiltonian (we call this part D3D). The routine `dsyevx` is used to solve the eigenproblem. In this part we have normally only one eigenproblem and this is of bigger size than the eigenproblems in D2D. This is why the parallelism must be obtained in the routine solving the eigenproblem. As in D2D, real symmetric eigenproblems are solved obtaining a part of the spectrum: the eigenvalues lesser than the cutting energy E^{2D}_{max} and the associated eigenvectors.

Table 1: Execution time, in seconds, in the different parts of the program, with automatic parallelization and using the routine `dsyevx` of the sequential and parallel libraries.

DVR=30 , E^{2D}_{cut}=15000 , V_{max}=20000				
Proc:	1	2	3	4
D2D	**5.15**	5.30	5.48	6.92
A2D	0.20			
G3D	24.64	17.69	**16.09**	17.60
D3D	3.62	3.43	**3.39**	3.60
DVR=40, E^{2D}_{cut}=20000, V_{max}=25000				
Proc:	1	2	3	4
D2D	60.19	**57.16**	64.62	60.63
A2D	1.69			
G3D	183.50	122.12	99.36	**86.16**
D3D	**19.79**	19.92	22.06	19.88

3 Automatic parallelism

The sequential program has been modified to obtain a parallel program in Origin 2000. The system has a parallelizing compiler [4]. Before executing the parallel program, the number of desired threads in the execution is assigned to the environment variable `MP_SET_NUMTHREADS`. The system also proportionates routines to solve eigenproblems: sequential routines in the library `complib.sgimath`, and parallel routines in the library `complib.sgimath_mp`.

In order to begin the analysis of the parallelism in the program some experiments have been carried out using automatic parallelization and the parallel library. In table 1 the results of two experiments are shown. The experiments correspond to "small" eigenproblems of dimension (DVR) 30 and 40, cutting eigenvalues in the "medium" eigenproblems (V_{max}) 20000 and 25000, and cutting eigenvalues in the "high" eigenproblem (E^{2D}_{cut}) 15000 and 20000, and a maximum number of cutting functions 1200 in both cases. The table shows the execution time with 1, 2, 3 and 4 processors in the different parts of the program.

We can see on each case, a reduction in the execution time in the eigenproblem does not happen. This is why we consider it is necessary to redesign the algorithms to adapt them to the architecture where we are working and reduce the execution time on D2D and D3D.

The time of A2D is very small in comparison with the time of the other parts. This part will be parallelized together with D2D.

In G3D a small reduction in the execution time is achieved, with a speed-up between 1.5 and 2.3. This part is the most costly in the program and a bigger speed-up can be easily obtained by making a dynamic assignation of the work to the processors.

After the analysis of the automatic parallelization in the program, in the next sections we will analyse some possibilities of explicit parallelism on each one of the different parts in the program.

4 Paralellism in the generation of the Hamiltonian

The generation of the Hamiltonian is done forming the lower triangular part of the matrix using the previously stored eigenvectors. It is generated in two loops, the first for the rows and the second for

Table 2: Execution time, in seconds, in G3D.

DVR=30, E_{cut}^{2D}=15000, V_{max}=20000				
Proc:	1	2	3	4
impl.	24.64	17.69	16.09	17.60
expl.	24.75	12.87	8.80	6.35
DVR=40, E_{cut}^{2D}=20000, V_{max}=25000				
Proc:	1	2	3	4
impl.	183.50	122.12	99.36	86.16
expl.	196.25	95.28	69.89	52.08

the columns of the matrix (only the lower triangular part), and for each element in the matrix, each element of two of the stored eigenvectors are used in the computation (a type of "dot product"), with these eigenvectors corresponding to the same or different matrices. The scheme is:

for $i = 1$ to max
  for $j = 1$ to i
    for $k = 1$ to $dim(vector)$
      computation

The automatic parallelization on the initial program has only parallelized the inner loop, which produces the creation of $\frac{1+max}{2}max$ threads to work with $dim(vector)$, and the performance is not very good. With explicit parallelism the outer loop can be parallelized and a dynamic assignation of indices to threads can be done. In this case, a certain number of threads is created and to each thread the work corresponding to an index is assigned. When a thread finishes its work, it solicits a new index. In that way the imbalance is avoided, but we have an additional cost of assignation of indices to threads. In table 2 the execution time of G3D is compared when varying the number of processors and with implicit and explicit parallelism. The results with explicit parallelism can be considered satisfactory, even when the number of processors increases.

5 Parallelism in the problems 2D

As we can see on table 1, the solution of eigenproblems on Origin 2000 does not produce a reduction in the execution time when the parallel library is used, and it is preferable to solve the problems using only one processor.

To avoid this bad parallelization it is possible to design methods to solve the eigenproblem efficiently in parallel in this system or, in D2D where independent "medium" eigenproblems are solved, to assign several problems to different threads to solve them in parallel and each one of them sequentially. There are some restrictions impedeing us to solve at the same time an unlimited number of problems, and this number is limited by the number of processors and the available memory in the system. When the problems are big, only a few or even only one of them can be stored in the memory. In D3D eigenproblems of "high" dimension are solved, and normally only a small number of problems or only one, and in this part of the program the parallelism must be obtained by parallelizing the resolution of one eigenproblem. On the other hand, the problems D2D are normally smaller, and they can be solved some of them at the same time because the matrices can be stored at the same time in memory, and also the size of the problems is not big enough to produce a satisfactory reduction of the execution time the solution of the problem in parallel.

```
In each thread:
WHILE more jobs
   C$OMP CRITICAL (memory)
      look for a job it can be allocated in the free memory
      IF job found
         to reserve a memory zone
   C$OMP END CRITICAL (memory)
   IF job found
      to solve the corresponding eigenproblem
      C$OMP CRITICAL (merge)
         merge eigenvalues and eigenvectors
      C$OMP END CRITICAL (merge)
      C$OMP CRITICAL (memory)
         to free memory in the map
      C$OMP END CRITICAL (memory)
```

Figure 1: Scheme of parallelism in 2D.

Therefore, we begin analysing the parallelism on the solution of some of the "medium" eigenproblems at the same time. A temporal assignation of the jobs and the memory to the threads must be obtained, in such a way that at each moment the assigned jobs have enough memory to be executed. This assignation has been done in a dynamic way. The threads are created and they access in mutual exclusion to the variables they describe the memory and jobs assignation. It is a typical problem of memory management [5].

Each thread, after solving the eigenproblem it has, merges the eigenvalues and eigenvectors it has obtained with those previously computed.

The scheme of the parallelization is given in figure 1. Each thread, while there are more jobs to do, consults in mutual exclusion the map of free memory and looks for a job that can be stored in the memory. If it does not find such a job the thread goes out of the zone of mutual exclusion and continues with a new consult to the map after some time. If the thread finds a job it can store in the memory, this zone of memory is reserved by modifying the memory map, and the job is marked as assigned. When the job is finished the thread accesses in mutual exclusion to merge the computed eigenvalues and eigenvectors with the previously stored, and finally accesses in mutual exclusion to the memory map to mark as free the memory it is not using. The assignation is done from bigger to smaller job, in such a way that the smallest jobs are assigned the last and a better balance is obtained.

In table 3 the execution time obtained in the solution of 2D is shown, varying the number of processors and the memory size used in the solution of the eigenproblems. In table 4 the efficiency obtained is shown. Tables show data obtained with problems with the parameters used in the previous tables (DVR=30 corresponds to the case 1 and DVR=40 to the case 2). The number of millions of double precision elements used to store the matrices is shown, and a similar amount of memory is needed to store the eigenvectors. In addition to this memory more memory is needed for auxiliar variables, as temporal areas, for the eigenvalues, etc. The size of the memory used for 1.44, 2.56, 3.24, 4.00 and 9.00 millions of double precision data are approximately 34 Mb, 60 Mb, 77 Mb, 96 Mb and 216 Mb, respectively. These sizes can seem very small, but the method is general and it can be used also with

Table 3: Execution time, in seconds, in the solution of 2D.

Pr.	case 1 1.44	case 2 1.44	case 2 2.56	case 2 3.24	case 2 4.00	case 2 9.00
1	5.35	61.88	61.88	61.88	61.88	61.88
2	3.09	31.11	31.60	40.43	32.45	
4	1.63	27.74	19.54	19.12	18.17	
6	1.15	28.08	17.88	14.79	15.03	13.86
8	0.92	27.75	17.76	16.38	14.19	11.46
12						8.55
16						7.29

Table 4: Efficiency in the solution of 2D.

Pr.	case 1 1.44	case 2 1.44	case 2 2.56	case 2 3.24	case 2 4.00	case 2 9.00
2	0.87	0.99	0.98	0.77	0.95	
4	0.82	0.58	0.79	0.81	0.85	
6	0.78	0.38	0.58	0.70	0.69	0.74
8	0.73	0.28	0.44	0.47	0.55	0.67
12						0.60
16						0.53

bigger sizes. In addition, in some cases could be preferable not to use all the available memory due to different reasons: more users in the system which also need memory, a better answer time of the jobs queue where the jobs are sent if less resources are demanded, a better utilization of the memory hierarchy, ...

The behaviour is as we hoped: when the number of processors increases the execution time decreases, but this reduction is limited by the memory and problem sizes, and no reduction is achieved over a certain number of processors, but reduction is achieved by increasing the memory size.

An increment in the number of processors does not always produce a reduction in the execution time because more jobs can not be allocated in the memory, and an increment of the amount of memory does not produce a reduction because no more processors are available. Therefore, a simulation can be performed in order to obtain the minimum number of processors and necessary memory to obtain the maximum reduction of the execution time. Table 5 shows the results of this simulation for problems with E_{cut}^{2D}=20000 and V_{max}=25000, and DVR=40, 50, 60 and 70. The simulation can also be used to determine for a maximum number of processors and available memory the quantity to use.

6 Parallelism in the solution of the problem 3D

After the Hamiltonian has been formed one or more "high" eigenproblems are solved. In some cases the eigenproblems D2D, what we call "medium" problems, also have a high dimension and they can not be solved in parallel due to the necessary memory. This is why to parallelise the solution of one eigenproblem is worthwhile.

One possibility is to utilise the routine `dsyevx` of LAPACK using the parallel library `complib.sgimath_mp`,

Table 5: Number of processors and quantity of memory, in Mb, to achieve the maximum reduction in the execution time, and simulated speed-up. E^{2D}_{cut}=15000, V_{max}=20000.

	DVR=40	DVR=50	DVR=60	DVR=70
processors	15	12	19	26
memory	216	600	1536	3456
speed-up	13.82	11.12	16.79	24.17

Table 6: Execution time, in seconds, and percentage with respect to the total time in the different routines of `dsyevx`.

	`dsytrd`	`dstebz`	`dstein`	`dormtr`	Total
DVR=30, E^{2D}_{cut}=15000, V_{max}=20000					
Time	1.78	0.53	0.78	0.44	3.53
%	50.6	14.9	22.0	12.5	
DVR=40, E^{2D}_{cut}=20000, V_{max}=25000					
Time	14.14	1.34	2.99	2.00	20.47
%	69.1	6.5	14.6	9.8	

but we have seen (table 1) the use of this routine does not reduce the execution time with respect to the sequential version, and to parallelise the routine obtaining good performance seems to be a difficult task.

The routine `dsyevx` works by reducing the initial matrix to tridiagonal form, using the routine `dsytrd`. After that reduction, the routine works with the reduced matrix computing the eigenvalues (which are also the eigenvalues of the initial matrix) in the desired interval, using the routine `dstebz`. The eigenvectors of the reduced matrix associated to the computed eigenvalues are computed with the routine `dstein`. Finally, the eigenvectors of the initial matrix are computed using the routine `dormtr`.

In [6] the routine `dsyevx` is analysed in detail. In that article the cost of each one of the routines is analysed when the complete spectrum is computed: `dsytrd` has a cost $\frac{4}{3}n^3$, half corresponding to level 3 operations and half to level 2 operations; `dstebz` has a cost $120n^2$ and `dstein` a cost $400n^2$, all of level 2; and `dormtr` has a cost $2n^3$, with level 3 operations. The most costly parts in the execution are of order n^3 (reduction to tridiagonal form and computation of eigenvectors of the initial matrix). But when working with not to big matrices the parts corresponding to the work with the reduced matrix can be very important due to the high value of the constants. In addition, because only a part of the spectrum is computed, the execution time of the three last routines is smaller, while the routine `dsytrd` does not vary independently of the number of eigenvalues to compute. This is why the most costly part is the reduction to tridiagonal form. In table 6 the execution time of the different routines and the percentage in the total time is shown for the two problems we are considering. We can see the routine `dsytrd` is the most costly and the percentage increases with the problem size. Because we are interested in the solution of "high" size problems, we will first try to optimize that routine.

A parallel version can work by assigning the work in the blocks to the processors in a cyclic way. In that way, in each step the processor which contains the block will work in `dlatrd`, and in the updating of the matrix all the processors work. An increases in the size of blocks will produce a bigger imbalance but also a better use of the memory hierarchy because contiguous data are assigned to the same processor.

It is also possible to parallelize the computation in `dlatrd`, but we have not done that in this first

Table 7: Execution time, in seconds, in parallel `dsytrd`. Matrix size 1600.

bl.	Processors 2	4	6	8
Version 1				
6	38.98	33.54	27.49	32.99
10	31.81	32.65	27.85	28.43
14	32.12	28.51	27.33	27.23
18	31.33	26.53	22.80	25.23
Version 2				
6	32.76	31.27	28.88	26.02
10	33.03	22.97	26.33	27.73
14	34.61	28.01	23.91	23.85
18	32.29	25.74	35.20	27.41

approximation.

In table 7 the execution times in D3D are shown, for different number of processors, and different problem and block sizes, and with two different ways of work assignation in `dsytrd`. In the first version the computations in `dlatrd` are done in the corresponding processor using the assignation of blocks indicated; more processes are managed but there is a fixed scheme to access the data, which can do this access more efficient. In the second version `dlatrd` is applied always in the main processor; less processes are managed, but more data movements can be necessary. A small reduction of the execution time is achieved, but if we compare these times with those obtained with `complib.sgimath_mp` (table 1) this first approximation can be considered satisfactory. The block size influences in the execution time, and a block size between 10 and 18 seems to be optimal. The optimal number of processors is between 4 and 8. A clear difference between the two versions is not observed.

7 Conclusions

In this work we have shown it is possible the parallelization on Origin 2000 of a program to compute levels of vibrational energy of triatomic molecules using the adiabatic DVR method. Different techniques have been used in the parallelization of different parts of the program: dynamic assignation of memory, utilization of compiler directives, and parallelization of eigensolvers. In table 8 the execution time of different parts of the program and the total execution time are shown, with the problems utilized in previous tables and another problem of bigger dimension. The reduction of the execution time can be considered satisfactory, but there are more possibilities of execution time reduction:

- Other routines of `dsyevx` can be parallelized.
- In 2D some processors can be idle because no memory is available. To avoid that the jobs can be assigned to more than one processor. This is a more complex problem of resources assignation.
- In some cases the last eigenproblem is solved in successive steps increasing the size of the problem. In those cases G3D and D3D can be parallelized together working one processor in the solution of the eigenvalue problems and the others in the construction of the Hamiltonian.

Table 8: Execution time, in seconds, in the different parts of the program.

Proc:	1	2	4	6	8	12	16
	DVR=30, E^{2D}_{cut}=15000, V_{max}=20000, size=1200						
2D	5.35	3.09	1.63	1.15	0.92		
G3D	23.83	11.79	5.72	4.18	3.11		
D3D	3.63	3.97	3.47	3.52	3.78		
Total	32.81	18.85	10.82	8.85	7.81		
	DVR=40, E^{2D}_{cut}=20000, V_{max}=25000, size=1200						
2D	61.88	32.45	18.17	15.03	14.19		
G3D	189.18	92.35	47.97	32.02	25.31		
D3D	19.79	17.86	17.85	16.07	16.74		
Total	270.85	142.66	83.99	63.12	56.24		
	DVR=40, E^{2D}_{cut}=20000, V_{max}=25000, size=3000						
2D				13.86	11.46	8.55	7.29
G3D				141.09	107.22	68.03	50.64
D3D				141.32	191.10	112.81	105.92
Total				296.27	309.78	189.39	163.85

References

[1] Z. Bačic and J. C. Light. Theoretical Methods for Rovibrational States of Floppy Molecules, *Annu. Rev. Phys. Chem.*, **40**, 469, 1989.

[2] J. Vives. *Determinación discreta y análisis estadístico del espectro vibracional del ion FHF$^-$*. Tesina de Licenciatura, Universidad de Murcia, 1998.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorensen. *LAPACK Users' Guide*. SIAM, 1995.

[4] *MIPSpro Fortran 77. Programmer's Guide*. Silicon Graphics.

[5] Aho, Hoptcroft and Ullman. *Data structures and algorithms*. Addison-Wesley, 1983.

[6] J. Demmel and K. Stanley. The Performance of Finding Eigenvalues and Eigenvectors of Dense Symmetric Matrices on Distributed Memory Computers. In David H. Bailey, Petter E. Bjørstad, John R. Gilbert, Michael V. Mascagni, Robert S. Schreiber, Horst D. Simon, Virginia J. Torczon and Layne T. Watson, editor, *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pp 528–533. SIAM, 1995.