Paralelización en Origin 2000 del cálculo de niveles de energía vibracionales de moléculas triatómicas *

Adolfo Bastida[1], Domingo Giménez[2], Alberto Requena[1] y José Zúñiga[1]

Resumen— **En esta comunicación se analiza la paralelización en Origin 2000 de un programa de Química Cuántica para el cálculo de niveles de energía vibracionales de moléculas triatómicas usando el método adiabático DVR.**

Palabras Clave— **Química Física, Computación Paralela, Origin 2000.**

I. Introducción

DURANTE los últimos veinte años se han desarrollado enormemente las técnicas de cálculo variacional para determinar niveles de energía vibracionales fuertemente excitados de moléculas triatómicas [1]. Uno de los métodos más eficaces utilizados es el método adiabático DVR. El método es válido para distintos tipos de coordenadas, pero veremos cómo se desarrolla para coordenadas hiperesféricas. En primer lugar especificamos las bases primitivas finitas utilizadas para generar las funciones de base discretas. Llamamos a las bases finitas FBR $\Phi_l(\rho)$, $\Phi_m(\phi)$ y $\Phi_n(\theta)$. Las bases discretas DVR se obtienen a partir de las finitas, diagonalizando la representación de la coordenada correspondiente a cada una de ellas, dando lugar a la resolución de tres problemas de valores propios "pequeños". Las ecuaciones de transformación entre ambas funciones de base son

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

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

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

donde $|\rho_i>$, $|\phi_j>$ y $|\theta_k>$ son las funciones DVR y R_{li}, S_{mj} y T_{nk} los correspondientes coeficientes de transformación.

Tomamos como coordenada adiabática la coordenada angular θ. Para cada valor discreto del ángulo θ_k definimos los operadores bidimensionales $\hat{H}^{2D}(\rho,\phi;\theta_k)$ de la forma

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

La ecuación de valores propios para este operador es

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

donde, tanto las funciones propias $\psi_n^{2D}(\rho,\phi;\theta_k)$ como los valores propios $E_n^{2D}(\theta_k)$ dependen paramétricamente de los puntos discretos θ_k. Para resolver esta ecuación escribimos las funciones propias $\psi_n^{2D}(\rho,\phi;\theta_k)$ como combinación lineal de la base bidimensional formada por los productos de representaciones DVR unidimensionales de las coordenadas ρ y ϕ, es decir

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

y diagonalizamos las correspondientes matrices Hamiltonianas para cada valor θ_k (problemas "medianos" de valores propios). La dimensión de estas matrices es $N_\rho \times N_\phi$, y puede crecer demasiado como para que la diagonalización sea practicable. Resulta razonable incluir en el desarrollo (6) solamente aquellos puntos DVR que proporcionen valores apreciables para las funciones propias $\psi_n^{2D}(\rho,\phi;\theta_k)$. En los puntos en los que la función de energía potencial tome valores muy elevados, la densidad de probabilidad será despreciable y podremos excluirlos del desarrollo. Para ello fijamos un valor máximo del potencial, V_{max}, y nos quedamos únicamente con aquellas funciones de base $|\rho_i>|\phi_j>$ que satisfacen la condición

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

*Los experimentos han sido realizados en el Origin 2000 del Centro Europeo de Paralelismo de Barcelona

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

[2]Parcialmente subvencionado por la Comisión Interministerial de Ciencia y Tecnología, projecto TIC96-1062-C03-02. Departamento de Informática, Lenguajes y Sistemas Informáticos, Universidad de Murcia, 30071 Murcia. domingo@dif.um.es

Llamamos $N^{2D}(\theta_k)$ al número de funciones de base bidimensionales DVR que cumplen esta condición, que es diferente para cada valor del angulo θ_k y que cumplen que $N^{2D}(\theta_k) < N_\rho \times N_\phi$.

Una vez determinadas las funciones bidimensionales $\psi_n^{2D}(\rho,\phi;\theta_k)$, las utilizamos, multiplicadas por sus correspondientes bases discretas angulares $|\theta_k >$, para construir el conjunto de funciones de base tridimensional $|\theta_k > \psi_n^{2D}(\rho,\phi;\theta)$ con el que resolvemos las ecuación de valores propios completa (problema "grande"). Escribimos así las funciones propias vibracionales tridimensionales como sigue

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

donde $D_{k,n,l}$ son los coeficientes de las funciones propias que determinaremos, junto con los valores propios de la energía, diagonalizando la matriz Hamiltoniana en la base $|\theta_k > \psi_n^{2D}(\rho,\phi;\theta_k)$. Los elementos de matriz correspondientes vendrán dados por

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

donde hemos utilizado la ecuación de valores propios bidimensional (5) para simplificar.

Tal como hemos definido las funciones de base discretas tridimensionales, la talla final de la matriz Hamiltoniana será $\sum_{k=1}^{N_\theta} N^{2D}(\theta_k)$. De nuevo aquí es deseable realizar una nueva selección de las funciones de base porque su número puede hacerse demasiado grande. Para ello usamos en el desarrollo completo (8) sólo aquellas funciones de base bidimensionales cuyas energías propias $E_n^{2D}(\theta_k)$ sean menores que cierta energía de corte bidimensional E_{corte}^{2D}, es decir las que cumplan la condición

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

Las funciones propias vibracionales vienen dadas finalmente por

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

donde el sumatorio se extiende sobre las N^{3D} parejas de índices k y n que satisfacen la condición (10), y el número de estas funciones propias es considerablemente inferior al número inicial de funciones de base discretas primitivas $N_\rho \times N_\phi \times N_\theta$.

Este metodo ha sido utilizado recientemente por un Grupo de Investigación del Departamento de Química Física de la Universidad de Murcia [2] para determinar el espectro vibracional completo del ión molecular FHF^-.

Debido al alto coste computacional del método cuando se pretende resolver problemas de grandes dimensiones, es interesante su optimización para poder resolver estos problemas eficientemente en máquinas paralelas modernas como el Origin 2000.

II. El programa secuencial

Analizaremos las cuatro partes más costosas en la resolución del problema para pasar posteriormente a su paralelización:

- La resolución del problema de valores propios que aparece en el cálculo de los cortes, que llamaremos D2D. En esta parte se utiliza la rutina `dsyevx` de LAPACK [3]. Los problemas son independientes y se conocen una vez resueltos los tres problemas "pequeños" iniciales, por lo que se pueden resolver en paralelo uno tras otro o varios al mismo tiempo, que es la opción que se ha elegido por ser la de mayor eficiencia. Se resuelven problemas reales simétricos obteniendo parte del espectro: los valores menores que la energía de corte V_{max} y sus vectores propios asociados.
- El almacenamiento de los resultados obtenidos al llevar a cabo las diagonalizaciones, que llamaremos A2D. En esta parte los valores y vectores propios calculados en cada paso se mezclan con los de los pasos anteriores, manteniendo ordenandos del mismo modo los vectores propios y los números cuánticos de cada valor. El coste es cuadrático y por lo tanto mucho menor que el de las otras partes que estamos analizando.
- La formación del Hamiltoniano final, que llamaremos G3D. Cada elemento i, j se forma a partir de los ángulos i, j almacenados. Para ángulos que vienen del mismo corte se hace una especie de producto escalar de los vectores asociados pero afectado cada término por un valor dependiente del número cuántico. Cuando los valores propios no vienen del mismo corte las longitudes de los vectores propios son distintas y también se hace una especie de producto escalar pero en este caso multiplicando cada elemento del primer vector por un elemento del segundo vector cuya posición no conocemos a priori sino que hay que calcular en función de los números cuánticos y los ángulos. Como la matriz es simétrica se calculan $n\frac{n+1}{2}$ entradas con n el número de valores propios almacenados. El trabajo que se realiza es proporcional al tamaño de los vectores (m), por lo que el coste es del orden $\frac{n^2}{2}m$, y el número de operaciones en coma flotante en casa paso es de 12.
- La resolución del problema de valores propios asociado al Hamiltoniano, que llamaremos D3D.

TABLA I

Tiempo de ejecución, en segundos, en las diferentes partes del programa, con paralelización automática y uso de la rutina **dsyevx** de la librería secuencial y paralela.

DVR=30 , E^{2D}_{corte}=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}_{corte}=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

También aquí se utiliza la rutina `dsyevx` para resolver el problema de valores propios. En esta parte el problema suele ser único y de mayor tamaño que en D2D, por lo que el paralelismo debe obtenerse paralelizando la rutina de cálculo de valores propios. Al igual que en D2D, se resuelven problemas reales simétricos obteniendo parte del espectro: los valores menores que la energía de corte E^{2D}_{max} y sus vectores propios asociados.

III. Paralelización automática

El programa se ha modificado para obtener un programa paralelo en Origin 2000. El sistema proporciona un compilador paralelizante. Antes de ejecutar el programa paralelo se asigna a la variable de entorno `MP_SET_NUMTHREADS` el número de threads que se asume que se ponen en marcha en una ejecución paralela. También se dispone de rutinas de cálculo de valores propios, tanto secuenciales, en la librería `complib.sgimath`, como paralelas, en la librería `complib.sgimath_mp`.

Para empezar analizando las posibilidades de paralelización en el programa hemos realizado algunos experimentos con paralelización automática y usando las librerías paralelas. En la tabla I se muestran los resultados de dos de los experimentos que corresponden a una dimensión de los problemas "pequeños" (DVR) 30 y 40, a valores de corte de los problemas "medianos" (V_{max}) 20000 y 25000, y a valores de corte del problema "grande" (E^{2D}_{corte}) 15000 y 20000, y con un número máximo de funciones de corte de 1200 en los dos casos. La tabla muestra tiempos de ejecución con 1, 2, 3 y 4 procesadores en las distintas partes del programa.

Vemos que en todos los casos, en la resolución de los problemas de valores propios no se produce una reducción clara en el tiempo de ejecución, lo que nos lleva a considerar que para reducir el tiempo de ejecución en D2D y D3D deberemos rediseñar los algoritmos usados para adecuarlos a la arquitectura del sistema con que estamos trabajando.

El tiempo invertido en A2D es muy pequeño en comparación con los de las otras partes. Esta parte se paralelizará junto con D2D.

En G3D se obtiene una reducción en el tiempo no demasiado grande, con un speed-up entre 1.5 y 2.3. Esta parte es la más costosa del programa y se puede obtener fácilmente un speed-up mayor realizando una asignación dinámica del trabajo a los procesadores.

Una vez analizado el comportamiento del programa con paralelización automática, pasamos a ver en las secciones siguientes algunas posibilidades de paralelización explícita en cada una de las partes del programa.

IV. Paralelización de la generación del Hamiltoniano

La generación del Hamiltoniano se hace formando la parte triangular inferior de la matriz utilizando los vectores propios almacenados en la parte anterior de la ejecución. Se genera el Hamiltoniano utilizando dos bucles, el primero para recorrer las filas de la matriz y el segundo para las columnas (sólo la parte triangular inferior), y para cada elemento se recorren uno por uno los elementos de dos de los vectores propios almacenados, que pueden corresponder a la misma o a distinta matriz. El esquema del código es:

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

La paralelización automática sobre el programa inicial sólo ha paralelizado el más interno de los bucles, lo que hace que las prestaciones no sean demasiado buenas al crearse threads $\frac{1+max}{2}max$ veces para trabajar con datos de tamaño $dim(vector)$. Con paralelización explícita se puede paralelizar el bucle más externo y realizar una asignación dinámica de los índices a los threads. En este caso, se crea un determinado número de threads y se asigna a cada uno el trabajo correspondiente a un i, conforme los threads van acabando el trabajo que tienen asignado toman un nuevo índice. De esta manera se evita el desbalanceo, pero se genera un coste adicional de asignación de índices a los threads. En la tabla II se comparan los tiempos de ejecución en G3D para distinto número de procesadores y con paralelización implícita y explícita. Los resultados con paralelización explícita pueden considerarse satisfactorios, y siguen siéndolo al aumentar el número de procesadores.

TABLA II

Tiempo de ejecución, en segundos, en G3D.

DVR=30, E^{2D}_{corte}=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^{2D}_{corte}=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

V. Paralelización de la resolución de los problemas 2D

Como vemos en la tabla I, la resolución de problemas de valores propios en el Origin 2000 no da buenos resultados, siendo preferible resolver los problemas usando un único procesador.

Para evitar esta mala paralelización se podría intentar diseñar métodos de resolución de problemas de valores propios en paralelo que funcionen de manera eficiente o, en la parte de resolución de problemas "medianos" de valores propios, asignar distintos problemas a threads distintos de manera que los problemas de valores propios se resuelvan en paralelo y cada uno de ellos en secuencial. Hay que tener en cuenta que tenemos unas restricciones que harán que no podamos resolver al mismo tiempo un número ilimitado de problemas, sino que este número estará limitado por el número de procesadores y por la memoria ocupada. Cuando los problemas son muy grandes puede que sólo quepan en memoria unos pocos o incluso sólo uno. En D3D se resuelven problemas de valores propios de "gran" dimensión, y normalmente un número reducido, incluso sólo uno, por lo que en esa parte del programa tendría más sentido la paralelización de la resolución de un problema de valores propios. Por el contrario, en D2D los problemas son normalmente más pequeños, por lo que pueden ejecutarse más al mismo tiempo al caber en memoria, y además su tamaño puede no ser lo suficientemente grande como para que la paralelización de la resolución de manera individual dé resultados satisfactorios.

Por tanto, empezaremos analizando la paralelización de la resolución "global" de los problemas "medianos" de valores propios. Se trata de determinar una asignación temporizada de los trabajos y la memoria a los threads, de manera que en cada momento los trabajos asignados tengan memoria suficiente para poder ejecutarse. Esta asignación se ha realizado de manera dinámica poniendo en marcha los threads y accediendo estos en exclusión mutua a las variables que describen la asignación de memoria y trabajos en cada momento. Se trata de un problema de administración de memoria [4].

Cada thread, además de resolver el problema de valores propios a él asignado, accedería después de la resolución a mezclar los valores y vectores propios calculados con los calculados por otros threads.

```
En cada thread:
MIENTRAS queden trabajos sin realizar
   C$OMP CRITICAL (memoria)
   buscar trabajo que quepa en la memoria
      SI hay trabajo
         reservar zona de memoria
   C$OMP END CRITICAL (memoria)
   SI hay trabajo
      resolver el PVP correspondiente
      C$OMP CRITICAL (mezcla)
         mezclar valores y vectores propios
      C$OMP END CRITICAL (mezcla)
      C$OMP CRITICAL (memoria)
         liberar zona de memoria en el mapa
      C$OMP END CRITICAL (memoria)
```

Fig. 1. Esquema de la paralelización de 2D.

El esquema de la paralelización se da en la figura 1. Cada thread, mientras quedan trabajos por hacer, consulta en exclusión mutua el mapa de memoria libre y busca un trabajo que quepa en la memoria, si no lo encuentra libera la exclusión en el acceso al mapa de memoria y vuelve a consultar, y si encuentra un trabajo que quepa en la memoria, reserva esa zona de memoria modificando el mapa y marca el trabajo como asignado. Una vez realizado el trabajo se accede en exclusión mutua a mezclar los valores y vectores calculados con los ya almacenados, y después se accede en exclusión mutua al mapa de memoria para indicar la memoria que ya no se está usando. Los trabajos se van intentando asignar del mayor al menor, de manera que queden los trabajos más pequeños para el final, con lo que se pretende balancear la asignación de trabajos.

En la tabla III se muestran los tiempos de ejecución y las eficiencias obtenidas en la resolución de 2D cuando varía el número de procesadores y el tamaño de la memoria utilizada en la resolución de problemas de valores propios. En la tabla IV se muestra la eficiencia. En las tablas se muestran resultados con los problemas con los parámetros usados en las tablas anteriores (DVR=30 corresponde al caso 1 y DVR=40 al caso 2). Aparece el número de millones de elementos en coma flotante usados para el almacenamiento de las matrices de que se calculan los valores propios, y otra cantidad de memoria de igual tamaño se utiliza para los vectores propios. Además de esa memoria se utilizan otras zonas para variables auxiliares, para zonas de trabajo temporales, para los valores propios, etc. Los tamaños de memoria utilizados para 1.44, 2.56, 3.24 y 4.00 millones de datos son aproximadamente 34 Mb, 60 Mb, 77 Mb y 96 Mb, respectivamente. Estos tamaños pueden parecer pequeños,

TABLA III
Tiempo de ejecución, en segundos, en la resolución de 2D.

Pr.	caso 1 1.44	caso 2 1.44	caso 2 2.56	caso 2 3.24	caso 2 4.00
1	5.35	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
8	0.92	27.75	17.76	16.38	14.19

TABLA IV
Eficiencia en la resolución de 2D.

Pr.	caso 1 1.44	caso 2 1.44	caso 2 2.56	caso 2 3.24	caso 2 4.00
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
8	0.73	0.28	0.44	0.47	0.55

pero el método es general y sirve para tamaños más grandes, y además, en algunos casos puede ser preferible no utilizar toda la memoria de la que podemos disponer por varios motivos: más usuarios en el sistema que también necesitan utilizar memoria, mejor tiempo de respuesta de la cola a la que se han mandado los trabajos si se demandan menos recursos, mejor utilización de la jerarquía de memorias, ...

El comportamiento es el esperado: al aumentar el número de procesadores se reduce el tiempo de ejecución pero esta reducción está limitada por el tamaño de la memoria y el de los problemas, lo que hace que se llega a un punto a partir del cual no se reduce el tiempo aumentando el número de procesadores, pero sí se reduce aumentando la memoria utilizada.

VI. Paralelización de la resolución del problema 3D

Después de formar el Hamiltoniano se resuelve uno o varios problemas "grandes" de valores propios. En algunos casos, los problemas de valores propios D2D, que hemos llamado "medianos", son también de gran dimensión y no es posible resolver varios a la vez debido a la memoria necesaria. Por todo esto, merece la pena intentar paralelizar la resolución de un problema de valores propios.

Una posibilidad sería utilizar la rutina `dsyevx` de LAPACK utilizando la librería paralela `complib.sgimath_mp`, pero ya hemos visto (tabla I) que el uso de esa rutina no reduce el tiempo de ejecución con respecto a la versión secuencial, por lo que la paralelización obteniendo buenas prestaciones se presenta como una tarea difícil.

TABLA V
Tiempo de ejecución, en segundos, y porcentaje respecto al tiempo total de las distintas rutinas de `dsyevx`.

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

La rutina `dsyevx` trabaja reduciendo la matriz inicial a forma tridiagonal, utilizando la rutina `dsytrd`; después de esto se trabaja con la matriz reducida calculando sus valores propios (que serán los de la matriz inicial) en el intervalo deseado, utilizando la rutina `dstebz`; se calculan los vectores propios asociados a los valores calculados en la matriz reducida, usando `dstein`; y finalmente se calculan los vectores propios de la matriz inicial con la rutina `dormtr`.

En [5] se estudia detalladamente la rutina `dsyevx`. En ese artículo podemos ver los costes de cada una de las rutinas que se utilizan en el caso en que se calcule todo el espectro: `dsytrd` tiene un coste $\frac{4}{3}n^3$, de los que la mitad corresponden a operaciones de nivel 3 y la otra mitad a operaciones de nivel 2; `dstebz` tiene un coste $120n^2$ y `dstein` un coste $400n^2$, todas ellas de nivel 2; y `dormtr` un coste $2n^3$, con operaciones de nivel 3. Las partes más costosas de la ejecución son las de orden n^3 (reducción a forma tridiagonal y cálculo de los vectores de la matriz inicial), aunque para matrices de dimensión no muy grande las partes correspondientes al trabajo con la matriz reducida pueden tener gran importancia debido al valor de la constante que las afecta. Además, como en nuestro caso no se calcula todo el espectro, el tiempo de ejecución de las tres últimas rutinas se reducirá, mientras que el de la rutina `dsytrd` no varía independientemente del número de valores que se calculen. Por todo esto, la parte más costosa debe ser la de reducción a forma tridiagonal. En la tabla V se muestra el tiempo de ejecución de las distintas rutinas y su porcentaje en el tiempo total para dos de los problemas que estamos considerando. Comprobamos que la rutina `dsytrd` es la más costosa y que su porcentaje aumenta al aumentar el tamaño del problema. Dado que estamos interesados en la resolución de "grandes" problemas, debemos intentar inicialmente la optimización de dicha rutina.

Para paralelizar la rutina `dsytrd` consideramos la matriz dividida en bloques de filas con b filas en cada bloque. Desde el último bloque hasta el primero se reduce cada uno de los bloques a forma tridiagonal utilizando la rutina `dlatrd`, y con la información obtenida en esta reducción se actualizan los bloques superiores de la matriz usando la rutina `dsyr2k`.

Una versión paralela puede consistir en asignar el trabajo sobre los bloques de manera cíclica a los distintos procesadores, de manera que en cada paso en `dlatrd` trabajará el procesador que contenga el bloque y en la actualización de la matriz trabajarán todos los procesadores. De esta manera, el aumentar el tamaño de los bloques producirá un mayor desbalanceo pero también un mejor uso de la jerarquía de memorias al asignarse datos contiguos al mismo procesador.

Sería también posible paralelizar el cálculo de `dlatrd`, pero no se ha hecho así en esta primera aproximación.

En la tabla VI se muestran los tiempos de ejecución en D3D cuando varía el número de procesadores, con distintos tamaños de problema y de bloque, y con dos versiones en la asignación del trabajo en `dsytrd`. En la primera versión el cálculo de `dlatrd` se hace en el procesador al que corresponde según la asignación del trabajo indicada, necesitándose la gestión de más procesos pero habiendo un esquema fijo de acceso a los datos que puede hacer que se realice de forma más eficiente; en la segunda versión `dlatrd` se aplica siempre en el procesador principal, con lo que se gestionan menos procesos pero puede ser necesario un mayor movimiento de datos en la memoria. Se comprueba que se puede reducir algo el tiempo de ejecución al aumentar el número de procesadores, aunque la reducción obtenida no es muy grande, pero si comparamos con los tiempo de ejecución obtenidos con `complib.sgimath_mp` (tabla I) podemos considerarla satisfactoria para una primera aproximación. El tamaño de bloque influye en el tiempo de ejecución pareciendo el más adecuado un tamaño entre 10 y 18. El número de procesadores para el que se obtienen los menores tiempos de ejecución está entre 4 y 8. No se ve una diferencia clara entre las dos versiones.

TABLA VI

TIEMPO DE EJECUCIÓN, EN SEGUNDOS, EN `dsytrd` PARALELO. CON TAMAÑO 1600.

	Procesadores			
bl.	2	4	6	8
Versión 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
Versión 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

TABLA VII

TIEMPO DE EJECUCIÓN, EN SEGUNDOS, EN LAS DISTINTAS PARTES DEL PROGRAMA.

Proc:	1	2	4	6	8
	DVR=30, E^{2D}_{corte}=15000, V_{max}=20000				
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}_{corte}=20000, V_{max}=25000				
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

VII. CONCLUSIONES

En este trabajo se ha mostrado cómo es posible la paralelización en Origin 2000 de un programa de cálculo de niveles de energía vibracionales de moléculas triatómicas usando el método adiabático DVR. En la paralelización del programa se han usado distintas técnicas que dan lugar a reducciones en el tiempo de ejecución distintas para distintas partes del programa: asignación dinámica de memoria, utilización de directivas de compilación, y paralelización de problemas de valores propios. En la tabla VII se muestran los tiempos de ejecución para las distintas partes del programa y para el total, con los dos problemas utilizados en las tablas anteriores y variando el número de procesadores. La reducción del tiempo de ejecución alcanzada en el conjunto del programa podemos considerar que es satisfactoria.

REFERENCIAS

[1] Z. Bačic y J. C. Light. *Annu. Rev. Phys. Chem.*, **40**, 469, 1989.

[2] J. Vives. *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 y D. Sorensen. *LAPACK Users' Guide*. SIAM, 1995.

[4] Aho, Hoptcroft y Ullman. *Estructuras de datos y algoritmos*. Adison-Wesley, 1988.

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