Theor Chim Acta (1991) 79: 323–333

Theoretica Chimica Acta

$D + D_2$ Quasiclassical rate constant calculations on parallel computers

Antonio Laganà[1], Ernesto Garcia[2], Osvaldo Gervasi[3], Ranieri Baraglia[4], Domenico Laforenza[4], and Raffaele Perego[4]

[1] Dipartimento di Chimica, Università di Perugia, 1-06100 Perugia, Italy
[2] Departamento de Quimica Fisica, Universidad del Pais Vasco, Bilbao, Spain
[3] Centro di Calcolo, Università di Perugia, Perugia, Italy
[4] Centro Nazionale Universitario di Calcolo Elettronico, Pisa, Italy

Received October 7, 1990/Accepted November 13, 1990

Summary. The calculation of rate constant values of the $H + H_2$ reaction for an extended range of excited vibrational states of the diatomic molecule and temperatures is relevant to the modeling of H^- sources. To investigate the effect of isotopic substitutions on the efficiency of vibrational deexcitation processes, we extended the calculations to the $D + D_2$ system. These calculations were carried out using a program restructured to run on a shared memory vector and parallel computer. The dependence of the efficiency of vibrational deexcitation processes from both the initial vibrational state and temperature of reactants is reported. Restructuring strategies adopted for implementing the program on both shared and distributed memory computers as well as speedups achieved on both types of machines are also discussed.

Key words: Parallel computing – Shared memory – Distributed memory – Quasiclassical trajectories – Rate constants

1. Introduction

Recently, we presented an extensive quasiclassical trajectory study of the reactive vibrational deexcitation for [1]:

$$H + H_2(v, j) \rightarrow H_2(v', j') + H \tag{1}$$

In fact, although significant theoretical effort has been paid to the investigation of this system [2–11], still very little is known about the efficiency of reactive deexcitation when the hydrogen molecule is vibrationally excited. Vibrational deexcitation of H_2 by collision with H atoms plays an important role in determining the efficiency of negative hydrogen ion sources. Vibrational deexcitation, in fact, competes with the H^- production mechanism (a dissociative electron attachment to vibrationally excited hydrogen molecules) [12–14]. In a simplified modeling of the H^- source, the assumption that rotational (T_{rot}) and translational (T_{tr}) temperatures of plasma particles have the same value T, is usually made. For this reason, rate constants for Eq. (1) were first calculated for T ranging from 300 K to 4000 K and up to very high v values [13]. The calculations showed that, in the investigated range of temperature, deexcitation

to the next lower vibrational level is the most efficient process. They also showed that a temperature increase makes the rate constant larger and that deexcitation rate constant values can be parameterized as a function of the vibrational jump n ($n = v - v'$).

For a more realistic modeling of the H^- sources, T_{rot} and T_{tr} need not have the same value. For this reason, the $H + H_2$ calculations of Ref. [1] were made at different combinations of T_{rot} and T_{tr} values. The calculated rate constants showed that the two temperatures play an opposite role: the rotational temperature enhances the efficiency of the deexcitation processes, while the translational temperature favours adiabatic processes at the expense of vibrational deexcitation.

Another important element that a realistic modeling of the H^- sources should include is the effect of isotopic variations. For this reason, we investigated:

$$D + D_2(v, j) \rightarrow D_2(v', j') + D \tag{2}$$

In this case too, there is no possibility of getting from the experiment all the necessary information. This means that the use of extended calculations (as a numerical substitute of the experiment) is, again, the only viable means of providing a complete set of rate constant values. At low v values, a few thousand trajectories are usually needed to obtain a reasonably converged estimate of the reactive atom-diatom cross section. To have a statistically significant sample of collisions, this number has to become larger when either the reactant vibrational number or the temperature increases. The number of trajectories to be calculated has to be further increased when rate constants rather than cross sections have to be calculated. That means that a minimum of $5n_v n_{T_{rot}} n_{T_{tr}} 10^3$ trajectories (where n_v, $n_{T_{rot}}$, and $n_{T_{tr}}$ are the number of vibrational states, rotational and translational temperatures considered) has to be calculated.

For calculating the $H + H_2$ rate constant values reported in Ref. [1] a number of trajectories of the order of 10^6 had to be integrated. To extend the investigation to Eq. (2) the number of calculated trajectories was increased significantly. This is due to two main reasons: The narrower vibrational spacing of the D_2 molecule and the larger maximum impact parameter $b_{\max}$. In fact, because of the narrower vibrational spacing, D_2 has about 50% more vibrational levels than H_2. In addition, to have about the same number of trajectories for each interval of b regardless of the value of $b_{\max}$, the number of calculated trajectories was increased when considering higher internal energy levels.

Therefore, with even stronger motivations than for Eq. (1), we turned to parallel computers to carry out the planned investigation. In view of future similar (or computationally more intensive) work, the trajectory program was implemented also on parallel machines other than those used for production runs. Production runs were carried out on IBM and CRAY supercomputers; test runs were performed also on NCUBE and MEIKO machines.

The paper is organized as follows: Parallel features of the numerical approach and speedups obtained on shared-memory machines after parallel restructuring are given in Sect. 2. Parallel restructuring for distributed-memory architectures is discussed in Sect. 3. Rate constant values calculated for Eq. (2) are discussed in Sect. 4.

2. Parallelism in trajectory codes

The parallel restructuring of the trajectory code has been performed on both shared memory [15] and distributed-memory [16–19] machines. Shared-memory parallel architectures have the characteristic of sharing a common large memory among a few powerful (and expensive) processors. Because of this, shared-memory machines are equally suitable for coarse and fine grain [20] parallelism. At the same time, they can usually vectorize (i.e. they have segmented functional units which allow a pipelining (temporal parallelism) of vector and matrix elements while undergoing the same operation). Distributed-memory parallel architectures host up to 10000 more processors than shared memory machines. Each processor has direct access only to its local memory. Cooperation with other processors (spatial parallelism) is obtained via message passing on the communication network. Obviously, processors of distributed-memory machines are less powerful than those of shared memory machines and because of their simplicity can be produced using cheap VLSI technologies. In the case of cpu-intensive and limited-memory applications, distributed memory architectures do not run into computational bottlenecks. On the contrary, problems may arise when dealing with highly coupled problems and fine grain parallelism. However, recently, they have also been used for handling complex accurate 3D reactive scattering computational procedures [21].

Vibrationally state-to-state but rotationally summed and averaged rate constants for an atom-diatom system of reduced mass μ are defined as:

$$k_{v,v'}(T_{tr}, T_{rot}) = \frac{2^{3/2} \sum_{j=0}^{\infty} (2j+1) e^{-E_j/kT_{rot}}}{(k^3 T_{tr}^3 \pi\mu)^{1/2} Q_R(T_{rot})} \times \sum_{j'=0}^{\infty} \int_0^{\infty} E_{tr} e^{-E_{tr}/kT_{tr}} \sigma_R^{vj,v'j'}(E_{tr})\, dE_{tr} \tag{3}$$

where $\sigma_R^{vj,v'j'}(E_{tr})$ is the reactive cross section at the collision energy E_{tr}, $Q_R(T_{rot})$ is the BC rotational partition function, k is the Boltzmann constant and v, j, v', j' are the vibrational and rotational numbers of reactants (unprimed) and products (primed).

The quasiclassical estimate of the rate constant can be obtained from Eq. (3) once the quasiclassical value of the state-to-state cross section has been calculated by integrating a sufficient number of classical trajectories. The quasiclassical cross section is defined as a multidimensional integral over all the possible atom-diatom initial situations that can be evaluated using Monte Carlo techniques [1, 22–24]. As a result, the detailed state-to-state cross section can be written as:

$$\sigma_R^{vj,v'j'}(E_{tr}) = \frac{\pi b_{\max}^2}{N} \sum_{i=1}^{N} f_{vj,v'j'}^{i} \tag{4}$$

where $b_{\max}$ is the maximum value of the impact parameter considered for calculating the N trajectories and $f_{vj,v'j'}^{i}$ is a Boolean function valued 1 if the result of integrating the ith trajectory having initial vibrotational vj state can be boxed into the final vibrotational $v'j'$ state and 0 otherwise. The index i is the trajectory sequential number which also labels the set of initial conditions. Initial conditions are derived by an adequate subset of numbers out of a pseudorandom sequence. They specify the initial geometry of the triatom and the way energy is

partitioned among its various degrees of freedom. The task of generating a pseudorandom distribution is usually assigned to a specific routine that starting from a given integer (seed) generates a real number in the interval 0–1 and a new seed. To make the set of integrated trajectories reproducible (i.e. to make unique the correspondence between the value of initial conditions and the trajectory index i), the generation of the pseudorandom number series has to be strictly sequential.

To determine the value of the Boolean function or, in other words, to generate the value of $v'j'$, the equation of motions which determine the evolution in time of the reactive system have to be integrated. The section of the code performing this integration is a completely independent computational task. This means that, in principle, this section can be split into N parallel processes giving a typical example of high potential parallelism.

On the machines used for production runs both temporal and spatial parallelism can be exploited. Temporal parallelism is obtained by carrying out vector restructuring. A way of exploiting the temporal parallelism in atom-diatom trajectory calculations has been discussed in Ref. [25] by considering the vector of the $12N$ phase space points associated with the N trajectories to be integrated. As an alternative, our approach tends to exploit both temporal and spatial parallelism. The temporal parallelism is taken care of by the vector facility. To this purpose vector and matrix manipulations have been rearranged to optimize the vector speedup. For the same reason a more suitable functional representation of the potential energy surface was adopted [26] and use was made of the appropriate library routines.

The spatial parallelism was exploited only on the IBM 3090 by making use of the Parallel Fortran (PF) compiler [27] managed by the MVS/XA operating system [28]. The PF is an extension of the language and library routines of version 2.1 of the Fortran VS compiler. The application was restructured as a set of distinct processes each one considered as a working unit to be executed on a Fortran processor. Each unit is assigned to a MVS task when using the MVS/XA operating system or to a virtual processor of a virtual machine when using the operating system VM/XA. MVS tasks and virtual processors compete with all other active tasks to get access to real processors. Under PF both implicit and explicit parallelism can be activated. Implicit parallelism is exploited by invoking the appropriate compiling options activating automatic parallelization functions. Automatic parallelization applies only to inhibitor-free DO loops whose parallel execution is more convenient than a sequential or a vector one. Explicit parallelism can be activated directly by the user by inserting appropriate explicit primitives in the program operating at instruction, DO loop and subroutine level. Locking and synchronization routines are provided as library extensions. To carry out a hot-spot analysis use can be made of the Interactive Debugging (IAD) [29]. The IAD, in fact, contains not only debugging functions but also functions supplying information about cpu time consumption at main, subroutine and instruction level. This has the advantage of singling out the sections of the code on which concentrate the restructuring work. Additional support to the restructuring work can be given by the use of the Intercompilation Analysis (ICA) option to check compatibility between program units and obtain information about common blocks, calls to functions and subroutines, use of variables, etc.

There are no general recipes for carrying out a successful parallelization. Automatic parallelization requires no restructuring work, but in return quite

often gives negligible or no speedup. The introduction of explicit parallelism is, therefore, indispensable for reducing computing time even if the price to be paid is, in general, a loss of the program portability. The introduction of parallelism, however, must be such that the per processor efficiency is not compromised.

A hot-spot analysis of the program has confirmed that the time consuming section of the code is that integrating Hamilton equations to determine the final value of $v'j'$ for each trajectory. For this reason, the restructuring work was concentrated on transforming that section of the program into a routine (TRAJ). Related common blocks were reorganized into two different sets according to the role they play in the parallel task: *sharing commons* for input and output operations and *copying commons* for input operations only. *Copying commons* are copied into the address space of the parallel task executing the TRAJ routine before starting its execution. *Sharing commons* are broadcasted every time the calculation associated with the parallel task comes to an end. Synchronization is performed both to access sharing commons and at the end of the DO loop running over the trajectory index before performing the final analysis of the results.

Main program

```
.....
CALL PROCS(Kparal)
DO I = 1, Kparal
        ORIGINATE ANY TASK TASKID
END DO

.....
Read input data and evaluate constants of common use
Seed = Initiator
Vector of seeds generation
Partition of the trajectories among generated Fortran processors

.....
DO I = Initj, Ifinj, Igrain
        Generate a pseudorandom number and a new Seed
        SCHEDULE ANY TASK TASKID
        SHARING (globally shared common areas)
        COPYINGI (common areas mapped into the address space of the
                called subroutine)
        CALLING TRAJ (I,Igrain)
END DO

.....
WAIT FOR ALL TASKS

.....
Statistical analysis

.....
END Main Program
```

Subroutine TRAJ

.....

```
DO I = 1, Igrain
        DO J = 1, Icond
                Generate the jth initial condition
                Generate a pseudorandom number and a new seed
        END DO
        Integrate trajectory differential equations
        Evaluate product properties
        Update the statistical analysis
END DO
END Subroutine TRAJ
```

In the program the variables Initj, Ifinj, Igrain and Icond specify the initial, final, subset dimension of the trajectories to be integrated and the number of initial conditions to be defined. As a result of the restructuring, the parallelized part of the code amounts to about 85%. The following speedups have been obtained on a dedicated IBM 3090/400 VF by imposing the explicit parallelism: 2.76 for three processors, 1.99 for two processors. Such a result clearly confirms the high potential parallelism of trajectory calculations.

3. An implementation on distributed-memory architectures

As already mentioned, the program was also implemented on two distributed memory parallel architectures: a NCUBE/10 hypercube with 512 processors and a small MEIKO Computing Surface with 8 Transputers. Main features of the NCUBE machine are [30]: The machine has a hypercube architecture made of 2^p nodes each hosting a 32 bit custom processor and six DRAM chips for a total of 512 kbytes. Each node sits at the vertex of a p-dimensional cube and has direct links to p other processors (p is the dimension of the hypercube). Each node is given a p-bit binary address (Gray code). Addresses of neighbouring (connected by a direct physical link) nodes differ by a single digit (the differing bit gives the dimension along which the two nodes are connected). If needed, the hypecube can be divided into subcubes of smaller dimensions. Main features of the Computing Surface are [31]: The machine has an extensible network of computing elements. Each computing element has a 32 bit processor, a 1 Megabyte memory and 4 full duplex links. Communications to the System Supervisor and the management of the network are taken care by intelligent interfaces.

To structure the program for the distributed environment we have adopted on both machines a task farm model of cooperation consisting of a master program running on one processor and controlling a set of identical worker processors all executing the same code. As shown by the following scheme:

Master processor program

```
Read input data from file and do initializations
Broadcast initial data to all (P) worker processes
```

```
Ntraject = Initj
Seed = Initiator

FOR Pworker = 0 to P-1 DO
        CALL RANDOM(Seed, New seed, Rand)
        SEND (Ntraject, Seed) to Pworker
        Ntraject = Ntraject + 1
        Seed = New_seed
NEXT Pworker

WHILE Ntraject ⩽ Ifinj DO
        CALL RANDOM(Seed, New_seed, Rand)
        RECEIVE msg of type msgdone from worker M
        SEND (Ntraject, Seed) to worker M
        Ntraject = Ntraject + 1
        Seed = New_seed
END_WHILE

FOR Pworker = 0 to P − 1 DO
        RECEIVE msg of type msgdone from worker M
        SEND to worker M a msg of type msgend
NEXT Pworker

Receive statistical indicators
Perform statistical analysis
Write results on file
END Master Processor Program
```

the master processor executes the preliminary calculations of the main program, dispatches trajectory integrations to the worker processors, collects final results and performs the final statistical analysis. To optimize the load balancing, a self-scheduling method assigning only one trajectory at a time to each worker processor has been adopted.

As shown by the following scheme:

Worker program

```
Receive initial data from master and do initializations

End = False

WHILE End = False DO
        RECEIVE Msg of any type from master
        IF Msgtype = Msgend THEN End = True

        ELSE
        Use random seed received to generate other random quantities
        Integrate the trajectory and update statistical indicators
        SEND MSG of type Msgdone to host

END_WHILE

Receive statistical indicators from higher dimension nodes
Combine received and local statistical indicators
Send combined results to the lower dimension node
END Worker Processor Program
```

each worker processor, starting from the seed received from the master, generates the trajectory initial conditions, integrates motion equations, evaluates final quantum numbers and updates local statistical indicators. Once that all the N trajectories have been integrated, the worker processors cooperate to combine local statistical information into the global statistical quantities. These quantities are then sent to the master.

Also for distributed memory machines the FORTRAN common blocks had to be deeply reorganized. As a result, only four common blocks were left: two containing the real and integer variables storing initial conditions and data (these are broadcasted by the master processor to all worker processors at the beginning of the execution) and two others containing real and integer statistical indicators (these are sent by the worker processor to the master at the end of the trajectory integration).

For both shared- and distributed-memory machines the pseudorandom sequence generation had to be restructured. In a scalar run, the correspondence between the elements of the pseudorandom sequence and the value of the trajectory initial conditions is uniquely determined because one trajectory at a time is calculated. As a consequence, any change in time-length of a part of the program will not affect the correspondence between elements of the pseudorandom sequence and the value of initial conditions. On the contrary, on a parallel machine, the competition between different processors may alter the order in which trajectory requests for pseudorandom numbers are issued and, therefore, result in different sets of initial conditions. To make the assignment deterministic, we kept the generation of the first pseudorandom number of each trajectory from the existing seed (i.e. from the initiator or the seed generated by the previous call to the random generator) at the master processor level. As a consequence, the generation being strictly sequential is uniquely determined. The generation of all the subsequent pseudorandom numbers needed to generate the remaining initial conditions characterizing a given trajectory, is carried out at the level of the worker process carrying out the integration of the motion equations. Therefore, also the rest of sequence is generated in a strictly sequential way.

On the NCUBE, particular care was also paid to further reduce communications between the host and the worker nodes. To this purpose, the procedure for collecting trajectory results and carrying out the final statistical analysis was redesigned. To do this, use was made of the so-called dimensional collapsing algorithm [32]. Accordingly, rather than sending node partial results directly to the host, these were transferred in parallel from all worker nodes of a given level to the nodes of the next lower level in the same dimension. This process is repeated until node 0 is reached and, from it, results are transferred to the host. A more detailed description of the restructuring technique, speedup evaluation methods and parameters as well as a collection of data concerning the measured efficiency of the trajectory program on NCUBE machines, is given elsewhere [24]. Average computer times measured for integrating one trajectory are: 92 s, 5.8 s, 2.9 s, 1.5 s, 0.75 s, 0.39 s, and 0.23 s for a NCUBE/10 machine having respectively, 1, 16, 32, 64, 128, 256, and 512 processors. For a Meiko Computing Surface having respectively 1, 4, and 7 worker processors, related times are: 21 s, 5.3 s, and 3.1 s. These results clearly show that trajectory calculations are perfectly scalable on distributed memory parallel machines. In other words, these machines are almost insensitive to an increase of the number of trajectories to be calculated provided that the number of processors is proportionally increased.

4. The $D + D_2$ deexcitation rate constants

Quasiclassical trajectories were run on the same LSTH potential energy surface [33] used for calculations of Ref. [1]. For a given set of T_{tr} and T_{rot} values, at low v levels we ran about 6000 trajectories by setting b_{max} at 3 Å. However, because the interval of fully reactive impact parameters increases with the vibrational number, the value of b_{max} was gradually increased with v. The number of calculated trajectories was increased accordingly. For practical reasons, as for Eq. (1), several jobs of about half an hour rather than a few very long ones, were submitted. Each job was devoted to the calculation of one set of fixed v, T_{tr} and T_{rot} $k_{v,v'}(T_{tr}, T_{rot})$ rate constants.

For illustrative purposes, rate constant values (in units of 10^{-12} cm^3 molecule^{-1} s^{-1}) calculated at $T_{tr} = 4000$ K $T_{rot} = 500$ K are shown in Fig. 1. For the sake of clarity, rate constant values belonging to the same initial vibrational state are connected by straight lines. As a result, each curve shows the variation of the efficiency of vibrational deexcitation process as the gap between the reactant and product vibrational number increases. All curves show an increasing trend for low n values. Then, after reaching a maximum, they decrease almost linearly. Up to $v = 5$, the most favoured deexcitation process is that to the next lower product state. When vibrational energy increases, the most efficient deexcitation process is the one leading to $v' = v - 2$.

A comparison with results of the $H + H_2$ reaction shows that isotopic changes are quite effective in damping vibrational deexcitation effects. In fact, rate constant values calculated for the reaction $H + H_2$ at the same translational and rotational temperatures are larger than those of D_2 because of the smaller reduced mass of the $H + H_2$ reaction. This results in a less efficient redistribution of the reactant vibrational energy into that of products. This effect is emphasized by the fact that, because of the smaller energy spacing between D_2 vibrational levels, the same vibrational level has less vibrational energy to redistribute in a reactive collision. The regular shape of the curves reported in the figure confirms also that if rate constant values are plotted, as in our case, as a function of n, the shape of the curve is smooth and can be easily parameterized.

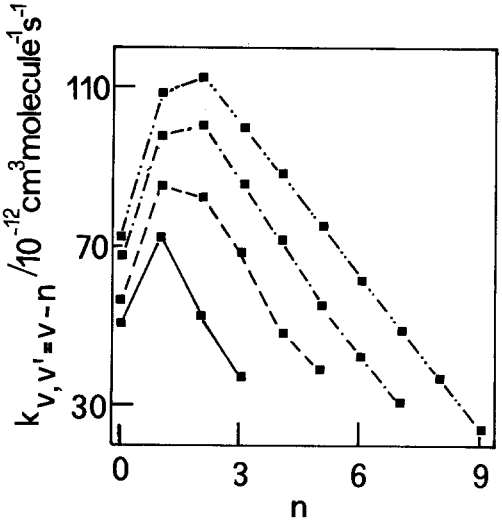

Fig. 1. Quasiclassical rate constants $k(v, v' = v - n)$ (in units of 10^{-12} cm^2 molecule^{-1} s^{-1}) plotted as a function of the vibrational jump n at $T_{tr} = 4000$ K and $T_{rot} = 500$ K. Only initial odd vibrational numbers are reported in the range $v = 9$ (upper curve, *dashed double dotted line*) – $v = 3$ (lower curve, *solid line*). Connecting lines have been drawn only for sake of clarity

5. Conclusions

The increasing need for massive trajectory calculations associated both with basic investigation of elementary chemical processes and with complex chemical systems modeling, can be satisfied only by making use of parallel computer architectures. In our case, the trajectory code calculating atom-diatom reactive rate constants restructured to run on both shared- and local-memory parallel computers, was used for evaluating rate constants of the $D + D_2$ reaction. Production runs, performed on the shared-memory IBM 3090/400 VF machine, allowed us to calculate all the rate constants needed to evaluate the efficiency of vibrational deexcitation for this reaction. Restructuring for the IBM 3090 was carried out by making use of the PF compiler. Other utilities designed to single out the most intensive computing parts of the application and to help the writing of cooperating computational tasks were also used. Significant parallel speedups were obtained by treating the trajectory integration as an independent task and dispatching its calculation to the different processors. The program was also restructured to run on a distributed memory computer. This was performed by copying into the worker nodes the trajectory integration routine. To minimize the amount of computational work assigned to the host, the generation of the initial conditions and the (partial) final statistical analysis were also transferred to the worker processors. In doing this, attention was paid to keep the generation of the initial conditions reproducible for different runs. A final effort has been dedicated to the efficient collection of results by using the dimensional collapsing algorithm.

A comparison of speedups obtained on shared and distributed memory parallel computers shows that comparable performances can be obtained when distributed memory machines can make use of a few hundred processors. Therefore, to carry out a reactive cross section calculation for the range of initial conditions we need to model H^- sources, machines with 10 or 100 more processors will be needed.

Acknowledgements. We wish to thank M. Capitelli for stimulating the investigation of the $H + H_2$ reaction. Thanks are also due to DELPHI (Viareggio, I), NCUBE (Portland, OR, USA), ACS MEIKO (Milano, I), CRAY (Spain) and CNUSC (Montpellier, F) for assistance and providing computer time. Partial financial support from CNR within the Progetto Finalizzato *Sistemi Informatici e Calcolo Parallelo* is acknowledged.

References

1. Laganà A, Garcia E, Mateos J (1991) Chem Phys Lett 176:273
2. Schatz GC (1986) In: Clary DC (ed) The theory of chemical reaction dynamics. Reidel, Dordrecht, p 1
3. Pack RT, Parker GA (1987) J Chem Phys 87:3888
4. Cuccaro SA, Kuppermann A (1989) Chem Phys Lett 154:155
5. Webster F, Light JC (1989) J Chem Phys 90:300
6. Linderberg J, Padkjaer S, Ohrn Y, Vessal B (1989) J Chem Phys 90:6254
7. Cuccaro SA, Kuppermann A (1989) Chem Phys Lett 157:440
8. Zhang JZH, Miller WH (1989) Chem Phys Lett 159:130
9. Manulopoulos DE, Wyatt RE (1989) Chem Phys Lett 159:519
10. Hancock G, Mead CA, Truhlar DG, Varandas AJC (1989) J Chem Phys 91:3492
11. Launay JM, Le Dourneuf ML (1989) Chem Phys Lett 163:178

12. Hiskes JR, Karo AM (1984) J Appl Phys 56:1827
13. Gorse C, Capitelli M, Bacal M, Bretagne J, Laganà A (1987) Chem Phys 117:177
14. Celiberto R, Cives P, Cacciatore M, Capitelli M, Lamanna U (1990) Chem Phys Lett 169:697
15. Hockney RW (1981) Parallel computers: architecture, programming and algorithms. Arrowsmith, Bristol
16. Seitz CL, Matisoo J (1984) Phys Today 37:38
17. Fox GC, Otto SW (1984) Phys Today 37:50
18. Fox GC, Lyzenga GA, Rogstad D, Otto S (1985) The Caltech Concurrent Computation Program Project Description. Proc 1985 ASME International Computers in Engineering Conference
19. Krishnamurthy EV (1989) Parallel processing: principles and practice. Addison-Wesley, Sydney
20. Gentzsch W, Szelenyi F, Zecca U (1988) Parallel Comput 9:107
21. Wu YM, Cuccaro SA, Hipes PG, Kuppermann A (1990) Chem Phys Lett 168:429
22. Bunker DL (1971) Methods Comput Phys 10:287
23. Alvariño JM, Garcia E, Laganà A (1989) In: Laganà A (ed) Supercomputer algorithms for reactivity, dynamics and kinetics of small molecules. Kluwer, Dordrecht, p 383
24. Baraglia R, Ferrini R, Laforenza D, Perego R, Laganà A, Gervasi O (1990) High Speed Computing (submitted)
25. Cochrane L, Truhlar DG (1988) Parallel Comput 6:63
26. Garcia E, Ciccarelli L, Laganà A (1987) Theor Chim Acta 72:253
27. Parallel Fortran Language and Libraries Reference. IBM Order No SC23-0431
28. MVS/XA General Information Manual IBM Order No GC28-1118
29. VS FORTRAN Version 2 Interactive Debug Guide and Reference, IBM Order No SC26-4223
30. Hayes JP, Mudge TN, Stout QF (1986) Architecture of a Hypercube Supercomputer. In: Proceedings of the 1986 International Conference on Parallel Processing, IEEE Computer Society Press, Washington, p 653
31. INMOS Ltd (1985) Transputer Reference Manual, INMOS
32. Gustafson JL, Montry GR, Benez RE (1988) SIAM J Sci Stat Comp 9:609
33. Truhlar DG, Horowitz CJ (1978) J Chem Phys 68:2468