Theor Chim Acta (1987) 72: 253-264

Theoretica Chimica Acta

A vectorizable potential energy functional for reactive scattering*

Ernesto Garcia,[1] Luigi Ciccarelli[2] and Antonio Laganà[2]

[1] Departamento de Quimica Fisica, Facultad de Ciencias, Universidad del Pais Vasco, E-48000 Bilbao, Spain
[2] Dipartimento di Chimica, Università di Perugia, I-06100 Perugia, Italy

(Received April 15, revised and accepted May 27, 1987)

Critical steps in trajectory programs for vector restructuring are analysed. The crucial effect of potential energy routine design on the efficiency of trajectory calculations performed on supercomputers have been investigated using a test program for which time consumptions when different algorithms are implemented are compared. Comparisons have been extended to more realistic computational situations by inserting related routines in a trajectory program and calculating reactive properties of some systems of chemical interest.

Key words: Potential energy functional — Reactive scattering — Supercomputer

1. Introduction

In a classical mechanics approach [1], the motion of three particles A, B and C having masses m_A, m_B and m_C on a given potential energy surface $U(|\boldsymbol{q}_A-\boldsymbol{q}_B|, |\boldsymbol{q}_B-\boldsymbol{q}_C|, |\boldsymbol{q}_A-\boldsymbol{q}_C|)$ ($\boldsymbol{q}_I$ is the vector describing the location of particle I with respect to a Cartesian frame of arbitrary origin and $|\boldsymbol{q}_I-\boldsymbol{q}_J|$ is the internuclear distance r_{IJ} between atoms I and J) is described by the Hamiltonian

$$H = T(\boldsymbol{p}_A, \boldsymbol{p}_B, \boldsymbol{p}_C) + U(r_{AB}, r_{BC}, r_{AC}) \tag{1}$$

where the kinetic operator T is defined as

$$T(\boldsymbol{p}_A, \boldsymbol{p}_B, \boldsymbol{p}_C) = \frac{\boldsymbol{p}_A^2}{2m_A} + \frac{\boldsymbol{p}_B^2}{2m_B} + \frac{\boldsymbol{p}_C^2}{2m_C} \tag{2}$$

* This paper was presented at the International Conference on 'The Impact of Supercomputers on Chemistry', held at the University of London, London, UK, 13-16 April 1987

with $p_A = m_A\dot{q}_A$. U is generated by the solution of the related electronic Schrödinger equation. The time evolution of the system can be followed by integrating nine pairs of Hamilton equations of the type

$$\dot{q}_{Iw} = \frac{\partial H}{\partial p_{Iw}} \quad \dot{p}_{Iw} = -\frac{\partial H}{\partial q_{Iw}} \tag{3}$$

for the three projections ($w = x, y, z$) of the conjugated quantities q_I and p_I on the axes of the Cartesian frame and for the three particles ($I = \mathrm{A, B, C}$). By working in the centre of mass, the number of equations to be solved can be reduced to twelve. No advantage is usually taken from total energy (E) and total angular momentum (J) constancy to further reduce the number of equations to be integrated. On the contrary, their constancy is frequently used for checking the accuracy of the numerical integration of Eqs. (3).

The large amount of cpu time required by trajectory integration makes a systematic investigation of the properties of reactive systems quite difficult. This problem can be relieved by making use of supercomputers (an IBM 3090/200 VF has been used for calculations) and by restructuring classical trajectory programs so as to take advantage of their special architecture. To do this, an analysis of the program structure has been made and the most time consuming routines isolated (Sect. 2). Speed up obtainable from restructuring the crucial section of the code using different algorithms has been estimated using a test program (Sect. 3). The efficiency of restructuring for a typical trajectory study of realistic chemical reactions is discussed in Sect. (4).

2. Structure of trajectory programs

The general structure of a classical trajectory program is sketched in Fig. (1). The program consists of an initial part in which the initiator of the random sequence, physical constants of the problem (such as colliding masses, translational and internal energies, initial distances, potential parameters, number of trajectories to be computed, etc.) are read in. The outermost DO loop running over the trajectory index is then opened. This determines the set of initial values to be selected randomly from a uniform distribution (phase of the initial oscillator, impact parameter, in plane and outplane angles of orientation of the target diatom, the angle formed between the initial direction of the velocity and the plane containing the three particles at initial time).

Nested inside the main loop are respectively the DO loop over integration steps and the DO loop over the 12 projections of the conjugated quantities on the Cartesian frame. Within the innermost DO loop the routine POTDER is called to evaluate the potential energy derivatives with respect to internuclear distances. In the same DO loop the integration (INTEGR), chain rule (CHAINR), and coordinate transformation (COORDT) routines are also called.

Earlier dynamical studies were carried out using simple functional forms [2] (such as the LEPS potential). Because related routines consist of few instructions

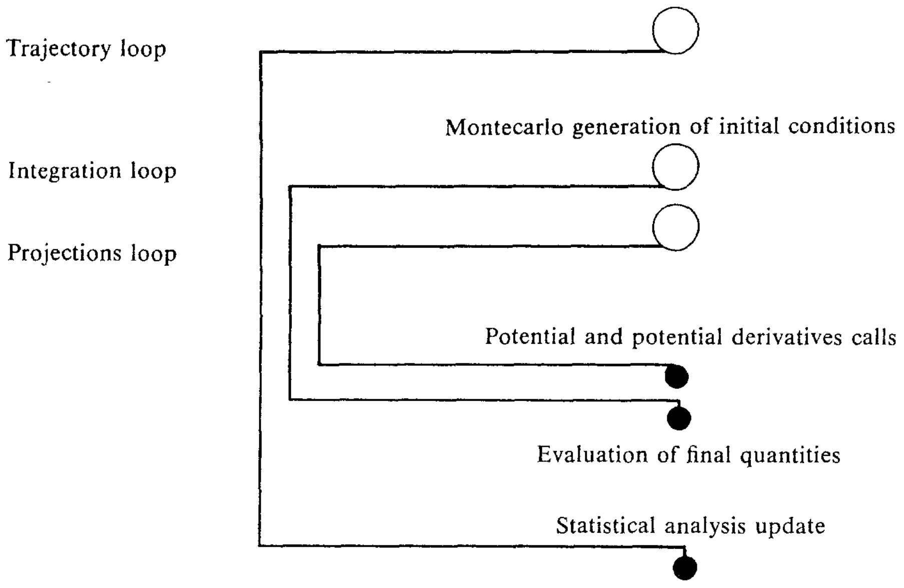
Trajectory loop
Integration loop
Projections loop
Montecarlo generation of initial conditions
Potential and potential derivatives calls
Evaluation of final quantities
Statistical analysis update

Fig. 1. Sketch of the program structure. *Open circles* indicate opening of DO loops. *Solid circles* indicate closing of DO loops

they take a little share of the overall cpu time. More recently, quite accurate calculations [3] and experiments [4] have produced such a wealth of information [5-7] on potential energy surfaces of reactive systems that, in general, they can only be reproduced by flexible but complicated functional forms [8]. To deal with the problem of an accurate representation of the atom diatom potential energy surface, we have proposed a new functional representation of the interaction, following the approach of Murrell and co-workers [9]. They suggest dividing the potential energy into the Multi-Body components

$$U(r_1, r_2, r_3) = V2_1(r_1) + V2_2(r_2) + V2_3(r_3) + V3_{123}(r_1, r_2, r_3) \tag{4}$$

where, for simplicity, the energy zero has been set at triatomic dissociation and the AB, BC and AC pairs have been numbered as 1, 2 and 3. At first, we represented the $V2_i$ terms as polynomials in the related Bond Order (BO) variable n_i ($n_i = \exp\{-b_i(r_i - r_{ei})\}$, b_i is a parameter and r_{ei} is the equilibrium distance of the diatom)

$$V2_i(r_i) = \sum_{k=1}^{K} a_{ik} n_i^k \tag{5}$$

where K (the degree of the polynomial) is usually 4 so as to obtain the b_i and a_{ik} coefficients of the polynomial by solving a system of algebraic equations generated by the requirement of reproducing the spectroscopic force constants of the diatom. This functional form is highly flexible and, at the same time, easy to compute. The use of BO variables allows a mapping of the interaction in a space that ensures its linearization in a form naturally converging to asymptotic limits for diatomic terms [10]. The BO representation has been successfully adopted also for the three body term [11] although in this case it does not converge to the right limit at short distances. A variable designed to reproduce both short and long range limits of the three body interaction is the product of the BO variable times the corresponding internuclear distance (XBO) [12]. In addition to being simple to calculate, BO and XBO functional representations of $V3_{123}$ have the advantage that their best fit coefficients can be estimated using a linear regression.

After evaluating potential derivatives in POTDER, time increments of conjugated quantities for the three particle system are calculated in the INTEGR routine. Usually an exit from this intermediate DO loop indicates either that the trajectory being calculated has reached an asymptotic limit (of reactants or products type) or that it has been trapped or slowed down in some region of the potential energy surface so that the maximum allowed number of steps has been exceeded. At this point, an analysis of quantities describing the system is performed and an association with the nearest quantum state is made. In this way, quantum-like scattering properties can be evaluated.

3. Vectorization of a test program

Parallel restructuring of a classical trajectory program is not a difficult task owing to the fact that each trajectory is an independent event. Therefore, provided that the initial conditions and final quantities are respectively generated and stored in matrices (so that trajectory integrations connect two pre-defined array elements) speed up factors obtainable are roughly equivalent to the number of cpus available.

The restructuring of trajectory programs for vectorization is more complicated. As already pointed out in the previous section, the key point for vector restructuring of a trajectory program are in routines evaluating potential energy derivatives. We have also already mentioned that BO and XBO polynomial representations of the potential energy surface are easy to compute. Therefore, it is an interesting problem to investigate the saving of computing time that can be obtained when running in vector mode programs which make use of this formulation of the potential energy.

In order to investigate in depth this problem, we have isolated the potential energy calculation from the program context and designed several algorithms for its evaluation. For simplicity, attention has been focussed upon the potential energy value rather than on its several partial derivatives and the study has been confined to BO representations. However, it can be easily shown that for BO and XBO polynomials routines for calculating partial derivatives with respect to internuclear distances have structure and performances similar to those of routines evaluating potential energy. For these test runs a simple main program (VTEST) has been written in which diatomic equilibrium distances (RO(*)), exponential parameters (B(*)), the number of terms of the bond order expansion (JKL), related coefficients (C(*), in vector form) and powers (JJ(*), KK(*) and LL(*))

of the BO expansion of the potential are read in. In VTEST C(*) as well as JJ(*), KK(*) and LL(*) are transformed respectively into the matrix A(*,*,*) and the vectors JG(*), KG(*) and LG(*) for use in nested DO loops. In addition, maximum values of JG(*), KG(*) and LG(*) vector elements are stored in JM, KM and LM. Also the maximum value of JM, KM and LM is stored for further use (JKLMAX). Finally, routines calculating the potential energy using different algorithms are called several time with different values of the internuclear distances R(*) in order to estimate the average time consumption per call.

The original potential energy routine POTJKL has been written in the form usually adopted for scalar computers

```
      SUBROUTINE POTJKL(VJKL)
C
      COMMON/C/JM,KM,LM,JKL,JKLMAX
      COMMON/V/B(3),R(3),RO(3),C(99),A(0:8,0:8,0:8),JJ(99),KK(99),LL(99)
C
      VJKL=0.0
      EN1=EXP(-B(1)*(R(1)-RO(1)))
      EN2=EXP(-B(2)*(R(2)-RO(2)))
      EN3=EXP(-B(3)*(R(3)-RO(3)))
      DO 103 L=0, LM
      EN3X=EN3**L
      DO 102 K=0, KM
      EN2X=EN2**K
      DO 101 J=0, JM
      VJKL=VJKL+A(J,K,L)*EN3X*EN2X*EN1**J
101   CONTINUE
102   CONTINUE
103   CONTINUE
      RETURN
      END
```

In order to have a more compact design of the routine, the potential has been redefined using the more general formulation

$$U(r_1, r_2, r_3) = \sum_{j=0}^{JM} \sum_{k=0}^{KM} \sum_{l=0}^{LM} A_{jkl} n_1^j n_2^k n_3^l \tag{6}$$

where diatomic terms have been incorporated into the three-body expansion by setting equal to zero the power of BO variables other than that relative to the considered diatom. Accordingly, A_{j00}, A_{0k0} and A_{00l} are set equal to the corresponding diatomic coefficients. When POTJKL runs in vector mode, only one small DO loop (out of three) vectorizes automatically. In consequence, little advantage is gained from the vector architecture.

In order to extend the vectorization to all terms of the summation, we have transformed the coefficient matrix into a vector and stored related exponents of BO variables into separate vectors. Accordingly, the restructured code (POTPOW) contains a single DO loop.

```
      SUBROUTINE POTPOW(VPOW)
C
      COMMON/C/JM, KM, LM, JKL, JKLMAX
      COMMON/V/B(3),R(3),RO(3),C(99),A(0:8,0:8,0:8).JJ(99),KK(99),LL(99)
C
      VPOW=0.0
      EN1=EXP(-B(1)*(R(1)-RO(1)))
      EN2=EXP(-B(2)*(R(2)-RO(2)))
      EN3=EXP(-B(3)*(R(3)-RO(3)))
      DO 99 I=1, JKL
      VPOW=VPOW+C(I)*EN1**JJ(I)*EN2**KK(I)*EN3**LL(I)
99    CONTINUE
      RETURN
      END
```

In our test case, the BO diatomic polynomials are all of the fourth order whilst the three-body term is of the sixth order. Therefore, JKL is 77 a more suitable value for taking advantage of vectorization. The average cpu time per call of both these routines are shown in the second and third line of Table 1. In the first two columns times measured on the IBM 3090/200 VF are reported. Time consumptions recorded during previous calculations performed on a CRAY X-MP.12 are reported in the third and fourth columns. A direct comparison of IBM and Cray times is meaningless. In fact, different precisions (single on Cray and double on IBM) and different time analysers used on the two machines significantly affect measured cpu times. Therefore, only meaningful comparison can be performed between scalar and vector performances measured on the same machine.

Time figures obtained when inhibiting vectorization (a single cpu was used all the time for these test runs) are shown in columns 1 and 3 respectively for the two machines. Related cpu times measured while allowing vectorization are reported in the second and fourth columns. When examining the cpu times shown in the table, a few comments are in order. As expected, no speed-up could be obtained when vectorizing VTEST. On a small scale, this is typically what occurs for the MAIN section for any package dealing with a realistic problem. Sometimes it also happens that the largest fraction of cpu time is spent in sections of the program which are not suitable for vectorization. In these cases, even if significant speed-ups are obtained for single routines, the global time saving can be percentually very small.

Table 1. Per call CPU time[a] for program VTEST

Routine	IBM scalar	IBM vector	Cray scalar	Cray vector
VTEST	17.00	17.00	14.00	14.00
POTJKL	0.80	0.97	0.35	0.24
POTPOW	0.47	0.62	0.21	9.06
POTEX	0.37	0.10	0.21	0.02
POTCUR	0.13	0.04	0.06	0.08
POTGAT			0.08	0.05

[a] Time in ms

In addition, a blind application of vectorization can lead to a speed decrease. This is what occurs for POTJKL on the IBM 3090/200 VF. The related speed-up ratio (SUR) is 0.82 (the speed-up ratio is defined as SUR = S/V where S and V are the cpu times spent in scalar and vector mode respectively). Similarly, for POTPOW the SUR factor is 0.76. However, from a comparison of cpu times for scalar runs of POTJKL and POTPOW, it is apparent that an effort to restructure a computer code for running on a vector computer quite often pays off also in terms of scalar speed.

Results obtained by running the same program on the Cray X-MP.12 are of a different type. In fact, for both POTJKL and POTPOW the SUR calculated on the Cray X-MP.12 is always larger than one. In particular, for POTJKL the value SUR is 1.46 meaning that, though not large, some advantage is still obtained if this routine is used for potential energy calculations. Clearer advantage is obtained when the POTPOW routine runs on a Cray X-MP.12. In fact, even in scalar mode, time spent per call for POTPOW is about half that of the original routine. On top of that a quite large SUR value (3.50) is still achieved when running in vector mode.

In order to further improve the vector efficiency of the code we have performed a deeper restructuring of the original routine and left inside the DO loop only a single exponentiation.

```
      SUBROUTINE POTEX(VEX)
C
      COMMON/C/JM,KM,LM,JKL,JKLMAX
      COMMON/V/B(3),R(3),RO(3),C(99),A(0:8,0:8,0:8),JJ(99),KK(99),LL(99)
C
      VEX=0.0
      X1=-B(1)*(R(1)-RO(1))
      X2=-B(2)*(R(2)-RO(2))
      X3=-B(3)*(R(3)-RO(3))
      DO 99 I=1, JKL
      VEX=VEX+C(I)*EXP(X1*JJ(I)+X2*KK(I)+X3*LL(I))
99    CONTINUE
      RETURN
      END
```

In this case, the efficiency of IBM 3090/200 VF improves by a factor of two with comparison to that of the original routine. On top of that, the SUR value is 3.70 making this routine eight times faster than the original one. Efficiency of the Cray is not very good when the routine is run in scalar mode (efficiency gain with comparison to the original routine is lower than a factor 2 as in the POTPOW case). However, the strength of the restructured code can be fully appreciated when it runs in vector mode. In fact, the SUR factor is 8.89. The POTEX routine is 15 times faster than POTJKL.

A different approach has been also followed in order to find a more efficient way of calculating the BO potential energy values on the IBM vector machine. By taking advantage of the possibility of making use of indirect addressing on the

IBM 3090/200 VF, powers of bond order terms have been evaluated outside the DO loop. Inside it, the needed power is referenced using a vector index.

```
      SUBROUTINE POTCUR(VCUR)
C
      COMMON/C/JM,KM,LM,JKL,JKLMAX
      COMMON/V/B(3),R(3),RO(3),C(99),A(0:8,0:8,0:8),JJ(99),KK(99),LL(99)
      COMMON/GATV/JG(99),KG(99),LG(99)
      DIMENSION FJ(6),FK(6),FL(6)
C
      VCUR = 0.0
      ENJ = EXP(-B(1)*(R(1) - RO(1)))
      ENK = EXP(-B(2)*(R(2) - RO(2)))
      ENL = EXP(-B(3)*(R(3) - RO(3)))
      FJ(1) = 1.0
      FK(1) = 1.0
      FL(1) = 1.0
      DO 100 I = 2, JKLMAX
      II = I - 1
      FJ(I) = FJ(II)*ENJ
      FK(I) = FK(II)*ENK
      FL(I) = FL(II)*ENL
100   CONTINUE
      DO 105 I = 1, JKL
      VCUR = VCUR + C(I)*FJ(JG(I))*FK(KG(I))*FL(LG(I))
105   CONTINUE
      RETURN
      END
```

In this case the IBM 3090/200 VF reaches its top performance. In fact, although the value of the SUR factor is similar to that obtained for POTEX (3.25 here against 3.70 for POTEX), the overall performance is superior because the scalar speed is much better. When POTCUR is run on the Cray X-MP.12, the scalar performance of this routine is excellent (0.06 ms). Unfortunately, when running in vector mode, no further time saving can be obtained due to the inability of this machine to vectorize codes making use of indirect addressing. The Cray X-MP.12 can deal with indirect addressing by calling GATHER and SCATTER routines. These routines in fact reorder vectors according to a predefined frame

```
      SUBROUTINE POTGAT(VGAT)
C
      COMMON/C/JM,KM,LM,JKL,JKLMAX
      COMMON/V/B(3),R(3)RO(3),C(99),A(0:8,0:8,0:8),JJ(99),KK(99),LL(99)
      COMMON/GATV/JG(99),KG(99)LG(99)
      DIMENSION FJ(99), FK(99), FL(99), EJ(99), EK(99), EL(99)
C
      VGAT = 0.0
      ENJ = EXP(-B(1)*(R(1) - RO(1)))
      ENK = EXP(-B(2)*(R(2) - RO(2)))
      ENL = EXP(-B(3)*(R(3) - RO(3)))
      DO 101 J = 0, JM
      EJ(J + 1) = ENJ**J
101   CONTINUE
      DO 102 K = 0, KM
      EK(K + 1) = ENK*K
```

```
102   CONTINUE
      DO 103 L=0, LM
      EL(L+1)=ENL**L
103   CONTINUE
      CALL GATHER(JKL,FJ,EJ,JG)
      CALL GATHER(JKL,FK,EK,KG)
      CALL GATHER(JKL,FL,EL,LG)
      DO 105 I=1, JKL
      VGAT=VGAT+C(I)*FJ(I)*FK(I)*FL(I)
105   CONTINUE
      RETURN
      END
```

In the above routine the DO loop actually calculating the potential energy value vectorizes because, before entering it, calls to the GATHER routine fill the FJ, FK and FL vectors with the proper sequences. Although this procedure looks quite complicated, the figures reported in the table indicate that this is the best scalar algorithm for computing BO potential energies on a Cray. However, when running in vector mode the speed gain is not as good as for POTCUR (the SUR factor is only 1.65) and POTGAT is outperformed by POTEX by a factor of 2.

4. Vectorization tests for real cases

As already mentioned, saving in computer time obtained from the use of vectorizable routines often, in real cases, is but a modest fraction of the overall time. In fact, excellent peak performances of a few routines can be quenched out by non vectorizable parts of the program. For this reason, after performing bench-like calculations using the VTEST program, we have carried out calculations of potential energy derivatives in their natural context of classical trajectory integrations for realistic reactive systems using a program having the structure described in Sect. 2 [13]. As a first case, we have examined the

$$\mathrm{Li}+\mathrm{HCl}(v,j)\rightarrow \mathrm{LiCl}(v',j')+\mathrm{H} \tag{7}$$

reaction by running batches of 50 trajectories starting from initial conditions reproducing experimental conditions [4]. Accordingly, the collision energy has been set at 9.2 kcal/mole, and the target molecule has been assumed to be in its ground vibrational state and at a rotational temperature of 60 K. Since the test was performed on the IBM 3090/200 VF, we have used derivative routines having the same structure as POTCUR because of the excellent vector performance of this routine on the IBM machine. The cpu time spent in scalar mode for integrating the 50 trajectories is 137.2 s while in vector mode it is 43.2 s. Related SUR factor is 3.2.

An account of the efficiency of the restructured computer code is given in Table 2.

In the table, the individual and fractional (related to total) cpu times spent by the most time consuming routines are shown for both the scalar and vector runs. As already pointed out in Sect. (3), the data of Table 2 demonstrates that the

Table 2. Absolute and relative CPU times[a] for a trajectory calculation

Routine	Scalar time	Scalar time %	Vector time	Vector time %
POTDER	100.0	72.9	24.7	57.1
INTEGR	22.6	16.5	5.8	13.4
CHAINR	9.6	7.0	9.3	21.6
COORDT	3.0	2.2	2.2	5.0

[a] Time in s

critical point of a classical trajectory calculation is in the evaluation of potential derivatives (POTDER) which accounts for about 70% of the cpu time in a scalar run. The second largest time consuming routine (when the program runs in scalar mode) is INTEGR which accounts for about another 20% of the global cpu time. It is not surprising, therefore, that the time saving is very large even when vectorization is involved for just these two routines. By vectorizing only POTDER time consumption halves. When vectorization is extended to INTEGR time consumption reduces to 46.6 s. This account for 96.4% of the time saving obtained when running the whole program in vector mode. This means that it is worthless paying additional effort for vectorizing the rest of the program.

On the contrary, additional effort has been paid in analysing whether part of the total speed up is due to factors not directly relevant to vectorization. For this reason, we have repeated our calculations in scalar mode using level 3 optimization (opt (3)) instead of the opt (0) level adopted in the previous run. In this way, the optimization level automatically built into compilation by the vector option is adopted also for scalar runs. As a result, again for a run of 50 trajectories, the needed cpu time almost halves (it lowers from 137.2 s to 83.8 s). This fact demonstrates that a large part of the time saving obtained when running the program in vector mode is due to a better organization of the scalar part of the code. As a further check of the influence of the optimization of scalar efficiency in determining the trajectory code speed up, we have run this program using opt (3) and allowing vectorization only for POTDER. As a result, cpu time lowers to 44.1 getting quite close to the value obtained when running the whole program in vector mode (43.2). This confirms the hypothesis that the main (and almost unique) vectorization contribution to computing time saving can be obtained by properly redesigning POTDER. Additional time saving is due to an appropriate reorganization of the scalar code and by the use of appropriate optimization levels.

It is also interesting to notice that, because POTDER vectorizes nicely, when running in vector mode its time share, though still large, lowers significantly. On the contrary, non vectorizing routines (such as CHAINR and COORDT) increase significantly their time share. This seems to indicate that the success of a restructuring process is evidenced by a levelling off of time fractions assigned to each routine.

The clear advantage of running restructured trajectory programs in vector mode, even for realistic reactive investigations, has enabled us to undertake a systematic

Table 3. Reactive cross sections[a] for Mg+HF(v,j)

	$v=2$	$v=3$	$v=4$	$v=5$
j=0	0.15	6.22	11.9	16.1
j=1	0.17	6.39	11.9	16.1
j=4	0.65	6.35	11.5	15.7
j=7	1.25	5.90	10.5	14.3
j=10	1.64	5.69	10.0	14.1

[a] In Å^2

investigation of reaction

$$\mathrm{Mg}+\mathrm{HF}(v,j)\rightarrow \mathrm{MgF}(v',j')+\mathrm{H} \tag{8}$$

on the same machine. *Ab initio* calculations of the potential energy have been recently carried out for this system [14]. The study of this process is part of a systematic analysis of reactivity and reactive dynamics of the family of alkaline earth hydrogen fluoride systems [15]. By carrying out calculations on a supercomputer it is possible to perform a crossed analysis of the effect of vibrational and rotational energy of the reactant molecule on the reactivity of the system in reasonable time. An example of results obtainable from this kind of calculation is reported in Table 3 for $E_{tr}=20$ kcal/mole (cpu time needed for these calculations has been about 35 hr).

Although a full rationalization of the dynamical behaviour of this reactive system requires additional computational effort for running more trajectories (both in order to have a wider variation of initial parameter and a graphical analysis of reactive paths), data shown in the table already indicate that the reactive cross section is a sensitive function of the vibrational and rotational energy of the reactants. In fact, as for other members of this family of systems [6, 16], reactivity increases with increasing vibrational energy, as expected from the late location of the (high) barrier in the exit channel. On the contrary, and partially unexpected, the effect of rotational energy on reactivity depends upon vibrational energy. In fact, while rotational energy favours reactivity at low reactant vibrational energy, it quenches reaction when the reagent diatom is highly vibrationally excited.

5. Conclusions

The main conclusion of our work is that vectorization of classical trajectory programs can be profitably performed only when the potential energy representation is reformulated in a way that allows vectorization. having isolated this problem from the bulk of the program, we have used a test program for studying the suitability of different algorithms in speeding up the integration of classical trajectories in both scalar and vector mode. In this way, we have found that speed ups of an order of magnitude can be easily obtained and that indirect addressing is the best technique for dealing with this problem on the IBM 3090/200 VF. In addition, we have found that time savings obtained in test calculations are, in large part, transferred to realistic cases. As a consequence, trajectory programs can take advantage not only of parallelism and of the higher scalar speed of supercomputers, but they can also reduce time requirements by using vectorizable potential energy functional forms. In this way, it is possible to confine the computing time needed for investigating reactive properties of realistic chemical systems to within reasonable limits.

During this investigation we have also found that a careful restructuring is in any case of great advantage for accelerating scalar calculations and that a thought balance between effort for vector restructuring and time saving obtained has to be made. Moreover, in some cases vectorization overheads have shown to be so heavy that the program runs faster in scalar mode.

In our calculations, we have also found that, in general, the Cray X-MP.12 improves its performance when going from scalar to vector mode. On the contrary, our experience suggests that this is not the case for the IBM 3090/200 VF. At the same time, we have found that in the case of indirect addressing the IBM 3090/200 VF gives its best performance. Meanwhile, for the same case the Cray X-MP.12, although running very fast in scalar mode, has to be helped by calls to special routines aimed at reordering vectors to run in vector mode.

Acknowledgments. Authors wish to thank IBM for generous allocation of free computer time on IBM 3090/200 VF at ECSEC (Roma, Italy). Calculations on the Cray X-MP.12 have been performed at Cineca (Casalecchio, Italy).

References

1. Truhlar DG, Muckerman JT (1979). In: Bernstein RB (ed) Atom molecule collision theory. New York, Plenum, p 105
2. Connor JNL (1979) Comput Phys Commun 17:117
3. See for example Schaefer III HF (1975). In: Bernstein RB (ed) Atom molecule collision theory. New York, Plenum, p 45
4. See for example Becker C, Casavecchia P, Tiedeman P W, Valentini JJ, Lee YT (1980) J Chem Phys 73:2833
5. Garcia E, Laganà A, Hernandez ML, Alvariño JM (1986) J Chem Phys 84:3059
6. Jaffe RL, Pattengill MD, Mascarello FG, Zare RN J Chem Phys (in press)
7. Palmieri P, Garcia E, Laganà A J Chem Phys (in press)
8. Sathyamurthy N (1985) Comput Phys Rep 3:1
9. Murrell JN, Carter S, Huxley P, Farantos SC, Varandas AJC (1984) Molecular potential energy functions. London, Wiley; Varandas AJC (1985) J Mol Struct 120:401
10. Garcia E, Laganà A (1985) Mol Phys 56:621
11. Garcia E, Laganà A (1985) Mol Phys 56:629
12. Dini M, Laganà A (unpublished results)
13. A program derived from QCPE n. 273, Department of Chemistry, Bloomington, Indiana
14. Paniagua M, Garcia della Vega JM, Alvarez Collado JR, Alvariño JM, Laganà A (1986) Chem Phys 101:55
15. Alvariño JM, Laganà A, Paniagua M Stud Chem (in press); Chapman S, Dupuis M, Green S (1983) Chem Phys 78:93
16. Chapman S (1984) J Chem Phys 81:262