Vectorization of the General Monte Carlo Classical Trajectory Program VENUS

Xiche Hu and William L. Hase*

Department of Chemistry, Wayne State University, Detroit, Michigan 48202

Tony Pirraglia

High Performance Computing Solution Development, IBM, Kingston, New York 12401

Received 3 January 1991; accepted 20 May 1991

The general chemical dynamics computer program VENUS is used to perform classical trajectory simulations for large polyatomic systems, with many atoms and complicated potential energy functions. To simulate an ensemble of many trajectories requires a large amount of CPU time. Since each trajectory is independent, it is possible to parallel process a large set of trajectories instead of processing the trajectories by the conventional sequential approach. This enhances the vectorizability of the VENUS program, since the integration of Hamilton's equations of motion and the gradient evaluation, which comprise 97.8% of the CPU, can each be parallel processed. In this article, the vectorization and ensuing optimization of VENUS on the CRAY-YMP and IBM-3090 are presented in terms of both global strategies and technical details. A switching algorithm is designed to enhance the vector performance and to minimize the memory storage. A performance of 140 MFLOPS and a vector/scalar execution rate ratio of 10.6 are observed when this new version of VENUS is used to study the association of CH_3 with the $H(Ar)_{12}$ cluster on the CRAY-YMP.

INTRODUCTION

Classical trajectory methods are widely accepted as a versatile approach to study microscopic molecular dynamics for a variety of chemical processes ranging from simple chemical bonding to protein folding, clustering, solvation, and polymerization.[1-12] The basic procedure is to choose a set of initial conditions and solve the classical equations of motion for a collection of particles representative of the physical system. For a system of N atoms, integrating the set of $6N$ first-order differential equations and evaluating the $3N$ components of the gradient are computationally the most time-consuming steps. The time needed to calculate a classical trajectory is determined by two important factors: the size of the interacting molecules and the complexity of the potential energy surface.

The practical implementation of classical trajectory methods to physically realistic systems, such as clustering and solvation, can be limited by the computer time required. As vector pipeline supercomputers have become more available,[13-18] attempts have been made to extend the applicability of classical trajectory methods[19-22] by vectorizing the computer codes in order to reduce the time required for propagating trajectories. Usually, trajectories are processed in a one by one sequential approach and, as a result, the iteration count for DO-loops is not large enough to achieve efficient vector performance. One of the necessary conditions for a DO-loop to be vectorizable is the independence of variables in sequential iterations. Since trajectory runs are independent of each other, a set of trajectories can be propagated simultaneously to enhance the vectorizability of the code.

VENUS is a generalized Monte Carlo classical trajectory program developed by Hase and co-workers and is widely used in dynamics studies.[23] In this article, the efficient vectorization of VENUS and the ensuing optimization of the vectorized code on the IBM-3090 and CRAY-YMP is discussed in both global strategies and technical details. By efficient we mean detailed consideration has been given to maximize execution speed and minimize memory storage. Due to the statistical nature of trajectory simulations, trajectory lengths are scattered. When vectorizing over the number of trajectories, simply terminating a trajectory once it is completed decreases the vector length. The result is particularly problematic when only a few long-lived trajectories exist, since short vector lengths reduce the performance gain of vector processing. This is especially true as the number of trajectories decreases to the vector break length, defined here as the vector length where the improved performance of vector processing is offset by the overhead associated with vector start-up. One approach to solve this problem is to increase the initial number of trajectories, so that a reasonably

*Author to whom all correspondence should be addressed.

CCC 0192-8651/91/081014-11$04.00

large number of long-lived trajectories will be left to process.[22] However, for many cases this may require an unreasonable amount of memory storage. To solve this problem we have designed a switching algorithm, the details of which are given in the next section, where the vectorization of VENUS is described. Test results for the association of CH_3 with the $H(Ar)_{12}$ cluster are given in the third section. The article concludes with a brief summary in the final section.

VECTORIZATION

In this section the vectorization of VENUS on both the IBM-3090 and CRAY-YMP is described and illustrated by typical examples.[24-29]

Structure of the Computer Program

In classical trajectory simulations, Hamilton's equations of motion are numerically solved. For a system of N particles, the Hamiltonian is written as

$$H = T(p_i) + V(q_i) \tag{1}$$

if the kinetic energy is coordinate independent. Hamilton's equations are then

$$\dot{q}_i = \frac{\partial H}{\partial p_i} = \frac{\partial T}{\partial p_i}$$
$$\dot{p}_i = -\frac{\partial H}{\partial q_i} = -\frac{\partial V}{\partial q_i} \qquad i = 1, 2, \ldots, 3N \tag{2}$$

In VENUS Cartesian coordinates and conjugate momenta are used in Hamilton's equations. The simulation consists of four steps: (1) representing the potential energy surface; (2) choosing starting conditions (initial coordinates and momenta); (3) integrating the classical equations of motion; (4) final states analysis (determining scattering angles, cross sections, rate constants, product energy distributions, etc.). As shown in Figure 1, the scalar version of VENUS processes the trajectories in a one by one sequential approach. Each trajectory is initialized with a Monte Carlo sampling procedure and a numerical integration algorithm is then used to solve Hamilton's equations of motion. VENUS uses a fixed step size sixth-order Adams–Moulton integrator, preceded by a fourth-order Runge–Kutta integrator. Analytic representations of the first-order derivatives of the potential energy with respect to Cartesian coordinates and of the kinetic energy with respect to Cartesian momenta are used in these integrations. A final states analysis is performed for the trajectory once it is completed.

It should be pointed out that, as a global strategy for transition from a scalar to vector code, the execution profile should be examined to identify the most time-consuming modules.[24] Although one should theoretically always try to vectorize as much

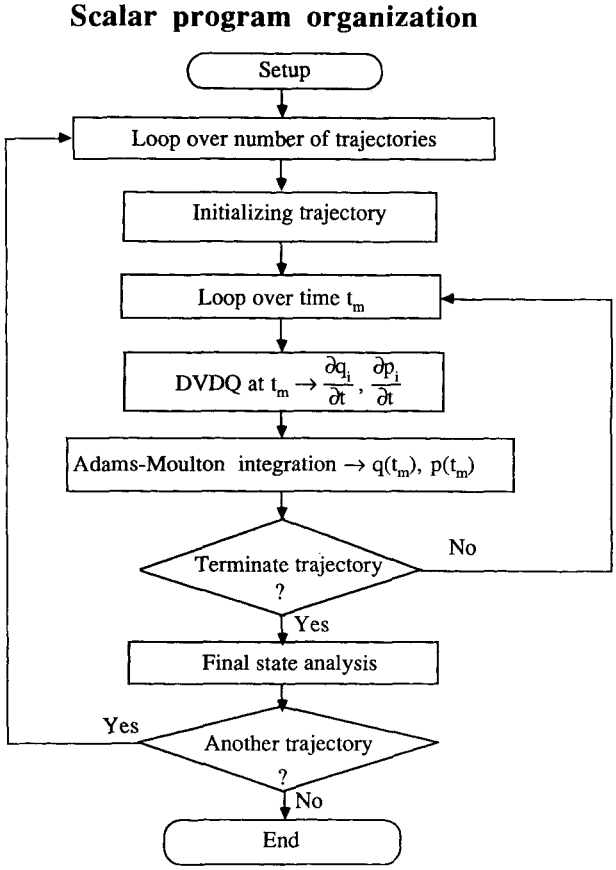

Figure 1. Flow chart of the scalar version of VENUS.

code as possible, only the most numerical intensive subroutines need to be vectorized to achieve a significant performance gain in a cost efficient manner. The computer program VENUS includes more than 50 subroutines. An execution profile for calculating $H(Ar)_{12} + CH_3$ trajectories with this program is plotted in Figure 2. More than 97.8% of the numerical processing is associated with subroutines TETRA, LENJ, and MORSE, which evaluate the first order derivatives $\partial V/\partial q_i$ in eq. (2) and the subroutine ADAMSM, which performs the numerical integration of Hamilton's equations of motion. Although such a profile is dependent upon the chemical system being studied, the $H(Ar)_{12} + CH_3$ example shown is typical and representative: the majority of CPU time is spent in the routines used in evaluating gradients and performing numerical integration. Naturally, we choose to vectorize the numerical integration and the $\partial V/\partial q_i$ derivative related subroutines. The structure of the vector program is presented in Figure 3. Here, a set of trajectories are propagated simultaneously instead of the one by one scalar approach, but the initialization and final states analysis are still executed in scalar.

Potential Energy Functions and Gradient

VENUS[23] is a versatile molecular dynamics program in the sense that it contains a variety of potential

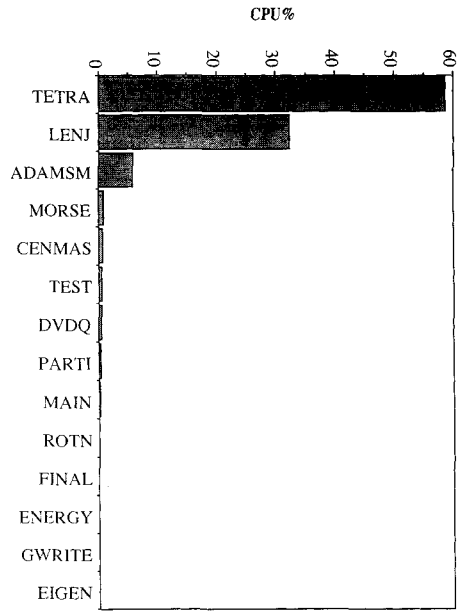

Figure 2. A flowtraced profile of the application of VENUS to association of CH_3 with $H(Ar)_{12}$ cluster. A total of 10 trajectories are evaluated, each for 7500 integration cycles.

energy functions that are characteristic of most physical systems of interest in molecular dynamics simulations. These potential functions are building blocks for potential surfaces used to model molecular systems. Listed in the following are the analytical expressions for these potential energy functions, and the names of the corresponding subroutines which evaluate the first order derivatives of the potential with respect to Cartesian coordinates (i.e., the gradients):

1. Harmonic stretch (STRET)

$$V = \tfrac{1}{2} f_o \, \Delta r^2 \tag{3}$$

2. Morse stretch (MORSE)

$$V = D[1 - \exp(-\beta \Delta r)]^2 \tag{4}$$

with $\beta = c_1 + c_2 \, \Delta r + c_3 \, \Delta r^2 + c_4 \, \Delta r^3$

3. Harmonic bend (HBEND)

$$V = \tfrac{1}{2} f_\theta (\theta - \theta_o)^2 \text{ with } f_\theta = f^o_\theta S(r_i) S(r_j) \tag{5}$$

where r_i and r_j define the bend and

$$S(r) = \exp(-c \, \Delta r^2) \text{ if } \Delta r > 0$$
$$= \quad 0 \quad \text{if } \Delta r < 0$$

4. Harmonic alpha bend or wag (HALPHA)

$$V = \tfrac{1}{2} f_a (a - \pi)^2 \tag{6}$$

5. Generalized Lennard–Jones (LENJ)

$$V = \frac{a}{r^n} + \frac{b}{r^m} + \frac{c}{r^l} \tag{7}$$

6. Twofold torsion

$$V = V_o(1 - \cos 2\tau)/2 \tag{8a}$$

or threefold torsion (HTAU)

$$V = V_o \cos^2(3\tau)/2 \tag{8b}$$

7. Generalized exponential repulsion or attraction (HEXP)

$$V = a \exp(-br) + c/r^n \tag{9}$$

8. Ghost pair interaction[30] (GHOST)

$$V = \frac{q_1}{r_{12}} + q_2 \left(\frac{1}{r_1} + \frac{1}{r_{1n}} + \frac{1}{r_{2i}} + \frac{1}{r_{2j}} \right) \tag{10}$$

9. Tetrahedral center[31] (TETRA)

$$V = \sum_{i=1}^{4} D_i (1 - e^{-\beta_i (r_i - r_i^0)})^2 + \tfrac{1}{2} \sum_{i=1}^{3} \sum_{j>1}^{4} f_{ij} (\theta_{ij} - \theta^o_{ij})^2 + \sum_{i=1}^{3} \sum_{j>1}^{4} g_{ij} (\theta_{ij} - \theta^o_{ij})^3 + \sum_{i=1}^{3} \sum_{j>1}^{4} h_{ij} (\theta_{ij} - \theta^o_{ij})^4 + \sum_{i=1}^{4} f_{\Delta i} \sum_{j=1, j\neq i}^{4} \Delta^2_{ij} + \sum_{i=1}^{4} h_{\Delta i} \sum_{j=1, j\neq i}^{4} \Delta^4_{ij} + g_{n4} C_4 \tag{11}$$

10. Nondiagonal stretch-stretch interaction (VRR)

$V = f_{ij}(r_i - r^o_i)(r_j - r^o_j)$ with

$$f_{ij} = f^o_{ij} S(r_i) S(r_j) \tag{12}$$

11. Nondiagonal stretch-bend interaction (VRT)

$V = f_{r\theta}(r_{ij} - r^o_{ij})(\theta_{klm} - \theta^o_{klm})$ with

$$f_{r\theta} = f^o_{r\theta} S(r_{ij}) S(r_{km}) \, S(r_{lm}) \tag{13}$$

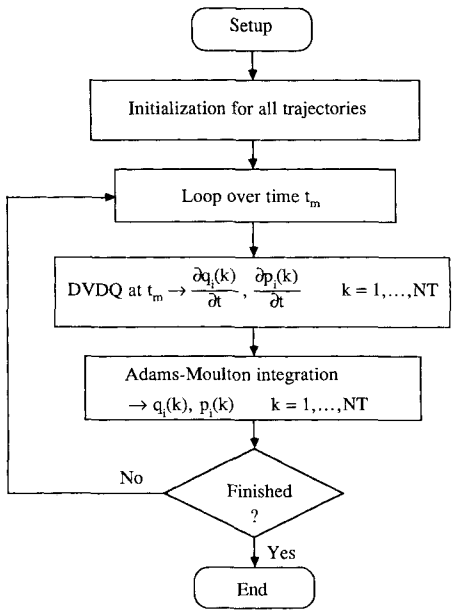

Figure 3. Flow chart of the vector version of VENUS.

12. Nondiagonal bend-bend interaction (VTT)

$$V = f_{\theta\theta}(\theta_{ijk} - \theta^o_{ijk})(\theta_{lmn} - \theta^o_{lmn}) \text{ with}$$
$$f_{\theta\theta} = f^o_{\theta\theta} S(r_{ij}) S(r_{jk}) S(r_{lm}) S(r_{mn}) \quad (14)$$

13. Torsion for dihedral angle (DANGLE)

$$V = \sum_n \frac{k^n_d}{2} [1 + \cos(n\phi - \gamma_n)] \quad (15)$$

All of the above potential energy functions are defined in terms of internal coordinates,[32] i.e., $V = V(r, \theta, a, \tau, \phi)$, where r represents an interatomic distance, and θ, a, τ, and ϕ represent bond angles. The derivatives of the potential with respect to Cartesian coordinates $\partial V/\partial q_i$ are determined from the chain rule

$$\frac{\partial V}{\partial q_i} = \frac{\partial V}{\partial r}\frac{\partial r}{\partial q_i} + \frac{\partial V}{\partial \theta}\frac{\partial \theta}{\partial q_i} + \frac{\partial V}{\partial a}\frac{\partial a}{\partial q_i} + \frac{\partial V}{\partial \tau}\frac{\partial \tau}{\partial q_i} + \ldots . \quad (16)$$

by writing the internal coordinates in terms of Cartesian coordinates.

Vectorization

Integration

As shown by eq. 2, for an N atom molecular dynamics system, Hamilton's equations of motion consist of $6N$ coupled differential equations. The choice of integration algorithms is determined by the nature of the forces existing in the molecular dynamics system. Consideration should be given to both the speed and stability of the algorithms.[33,34] For systems in which rapid and slow changes in the derivatives $\partial V/\partial q_i$ coexist, such as solute and solvent interactions, multiple time step size integration methods can be used.[35,36] In scalar mode, variable step size integrators may be used to minimize the number of gradient evaluations, and therefore, enhance the overall efficiency.[33] However, studies have shown that higher-order and variable step size methods are not always more efficient than lower order, fixed step size methods.[4] In VENUS, a fixed step size sixth-order Adams–Moulton integrator, initialized by a fourth–order Runge–Kutta integrator is used to solve Hamilton's equations. No attempt was made to choose a different integrator in vectorizing VENUS, in part because variable step size methods may cause difficulty in vectorization. VENUS is constructed so that it is easy for users to replace these integrators with their own.

Since both the predictor and corrector components to the Adams–Moulton integrator have the form[33]

$$y_n = y_{n-1} + h \sum_{i=0}^{k-1} \beta_i f_{n-i}, \quad (17)$$

a similar FORTRAN code can be used to vectorize each of the components. The vectorized corrector code is written as

```
C
C        ADAMS-MOULTON CORRECTOR
C
      DO 600 I=1,NI
      J1=2*NID+I
      J2=NID+J1
      J3=NID+J2
      J4=NID+J3
      J5=NID+J4
      DO 68 II=NTI,NTF
      ASUM=27.D0*TABLE(II,J1)-173.D0*TABLE(II,J2)
     *     +482.D0*TABLE(II,J3)-798.D0*TABLE(II,J4)
     *     +1427.D0*TABLE(II,J5)+475.D0*PDOT(II,I)
      P(II,I)=TABLE(II,I)+ASUM*ATIME
   68 CONTINUE
  600 CONTINUE
```

Index II is over trajectories and I over Cartesian coordinates. The II DO-loop is arranged as the innermost loop since most machines can only vectorize the innermost loops. For machines capable of vectorizing outer loops, such as the IBM-3090, the II DO-loop can easily be switched to an outer loop, since II and I are independent. We should also point out that for systems containing a large number of atoms, the size of the I DO-loop($3N$) can be significantly bigger than that of the II DO-loop. Switching the I DO-loop into an inner loop will ensure a maximum degree of vectorization in this particular case. Notice that the vector operations inside the II DO-loop are nearly all compound operations. Thus, the performance of the vectorized code is quite efficient.

Evaluation of the Gradient

As mentioned earlier, the VENUS program contains a variety of potential energy functions, from the simple Morse potential to a potential as complicated as that for a tetrahedral center. Finding the first-order derivatives of the potentials is the core of the trajectory calculation. All 13 types of potentials are solved analytically for their first-order derivatives with respect to Cartesian coordinates, then arranged into the gradient evaluation subroutines STRET, MORSE, HBEND, HALPHA, LENJ, HTAU, HEXP, GHOST, TETRA, VRR, VRT, VTT, DANGLE. Presenting a complete report on all of these subroutines is certainly beyond the scope of this article. Instead, we will use the Lennard–Jones potential as an example case to illustrate how to effectively transfer a scalar version of a module into a vector version. Listed are the scalar and vector versions of the FORTRAN code for LENJ.

```
C***        Scalar Version of LENJ
C
      SUBROUTINE LENJ(NL)
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON/QPDOT/Q(150),PDOT(150)
      COMMON/COORS/R(500)
      COMMON/LENJB/ALJ(500),BLJ(500),CLJ(500),N5J(500),N5K(500),NREP(500
     *),MREP(500),LREP(500)
      COMMON/FORCES/N
C        CALCULATE INDICES FOR COORDINATES
      J3=3*N5J(NL)
      J2=J3-1
      J1=J2-1
      K3=3*N5K(NL)
      K2=K3-1
      K1=K2-1
C        CALCULATE INDEX FOR R
      JK=(N5J(NL)-1)*(2*N-N5J(NL))/2+N5K(NL)-N5J(NL)
```

```
C        CALCULATE RELATIVE COORDINATES AND R
      T1=Q(K1)-Q(J1)
      T2=Q(K2)-Q(J2)
      T3=Q(K3)-Q(J3)
      R(JK)=DSQRT(T1*T1+T2*T2+T3*T3)
C        CALCULATE (DV/DQ)'S.
      DUM1=0.D0
      IF(NREP(NL).EQ.0)GO TO 1
      DUM1=-NREP(NL)*ALJ(NL)/R(JK)**(NREP(NL)+2)
    1 IF(MREP(NL).EQ.0)GO TO 2
      DUM1=DUM1-MREP(NL)*BLJ(NL)/R(JK)**(MREP(NL)+2)
    2 IF(LREP(NL).EQ.0)GO TO 3
      DUM1=DUM1-LREP(NL)*CLJ(NL)/R(JK)**(LREP(NL)+2)
    3 DUM2=T1*DUM1
      PDOT(K1)=PDOT(K1)+DUM2
      PDOT(J1)=PDOT(J1)-DUM2
      DUM2=T2*DUM1
      PDOT(K2)=PDOT(K2)+DUM2
      PDOT(J2)=PDOT(J2)-DUM2
      DUM2=T3*DUM1
      PDOT(K3)=PDOT(K3)+DUM2
      PDOT(J3)=PDOT(J3)-DUM2
      RETURN
      END
C***        Vector Version of LENJ
C
      SUBROUTINE LENJ(INL,LNL)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON/QPDOT/Q(63,150),PDOT(63,150)
      COMMON/COORS/R(63,500)
      COMMON/LENJB/ALJ(150),BLJ(150),CLJ(150),N5J(150),N5K(150),NREP(150
     *),MREP(150),LREP(150)
      COMMON/FORCES/N
      COMMON/LENJL/JK(500),RNA(500),RMB(500),RLC(500)
      COMMON/INTEGR/TIME,ATIME,NI,NID,NTI,NTF
      COMMON/SHARE/DIFFN,DIFFM,DIFFL
      LOGICAL FIRST,DIFFN,DIFFM,DIFFL
      DATA FIRST/.TRUE./
C
      IF (FIRST) THEN
      DO 9997 NL=INL,LNL
        JK(NL)=ISHFT((N5J(NL)-1)*(2*N-N5J(NL)),-1)+N5K(NL)-N5J(NL)
        RNA(NL)=-NREP(NL)*ALJ(NL)
        RMB(NL)=-MREP(NL)*BLJ(NL)
        RLC(NL)=-LREP(NL)*CLJ(NL)
 9997   CONTINUE
      ENDIF
C
C  Code used when NREP, MREP, or LREP values are not constant
C
      IF (DIFFN.OR.DIFFM.OR.DIFFL) THEN
        DO 9510 NL=INL,LNL
        J3=3*N5J(NL)
        J2=J3-1
        J1=J2-1
        K3=3*N5K(NL)
        K2=K3-1
        K1=K2-1
        JJ=JK(NL)
        DO 9500 II=NTI,NTF
        T1=Q(II,K1)-Q(II,J1)
        T2=Q(II,K2)-Q(II,J2)
        T3=Q(II,K3)-Q(II,J3)
        R(II,JJ)=DSQRT(T1*T1+T2*T2+T3*T3)
        RRJK=1.0/R(II,JJ)
        DUM1=RNA(NL)*RRJK**(2+NREP(NL))
        DUM1=DUM1+RMB(NL)*RRJK**(MREP(NL)+2)
        DUM1=DUM1+RLC(NL)*RRJK**(LREP(NL)+2)
        TDUM1=DUM1*T1
        TDUM2=DUM1*T2
        TDUM3=DUM1*T3
        PDOT(II,K1)=PDOT(II,K1)+TDUM1
        PDOT(II,K2)=PDOT(II,K2)+TDUM2
        PDOT(II,K3)=PDOT(II,K3)+TDUM3
        PDOT(II,J1)=PDOT(II,J1)-TDUM1
        PDOT(II,J2)=PDOT(II,J2)-TDUM2
        PDOT(II,J3)=PDOT(II,J3)-TDUM3
 9500   CONTINUE
 9510   CONTINUE
C
C   Code for 12-6-0 LENJ inputs
C
      ELSE IF (NEXP.EQ.12.AND.MEXP.EQ.6.AND.LEXP.EQ.0) THEN
        DO 9550 NL=INL,LNL
        J3=3*N5J(NL)
        J2=J3-1
        J1=J2-1
        K3=3*N5K(NL)
        K2=K3-1
        K1=K2-1
        JJ=JK(NL)
        DO 9540 II=NTI,NTF
        T1=Q(II,K1)-Q(II,J1)
        T2=Q(II,K2)-Q(II,J2)
        T3=Q(II,K3)-Q(II,J3)
        RRJK=1/DSQRT(T1*T1+T2*T2+T3*T3)
        DUM1=RNA(NL)*RRJK**14
        DUM1=DUM1+RMB(NL)*RRJK**8
        TDUM1=DUM1*T1
        TDUM2=DUM1*T2
        TDUM3=DUM1*T3
        PDOT(II,K1)=PDOT(II,K1)+TDUM1
        PDOT(II,K2)=PDOT(II,K2)+TDUM2
        PDOT(II,K3)=PDOT(II,K3)+TDUM3
        PDOT(II,J1)=PDOT(II,J1)-TDUM1
        PDOT(II,J2)=PDOT(II,J2)-TDUM2
        PDOT(II,J3)=PDOT(II,J3)-TDUM3
 9540   CONTINUE
 9550   CONTINUE
      END IF
      RETURN
      END
```

Given the way the scalar version is written, it is not vectorizable. The PDOT array contains an apparent recurrence. This is a common problem faced by all derivative (i.e., gradient) evaluation subroutines. One way to solve this problem is to break the NL DO-loop into two loops, isolating the recursive portion of the DO-loop, so that part of the DO-loop will be vectorizable. Even with this measure, one still cannot expect a significant gain from vectorization since the vector length over NL is too short in most gradient subroutines. Another difficulty peculiar to the scalar LENJ subroutine is that it contains numerous IF (conditional) statements. While IF statements can be vectorized under mask operations, the resultant executable code often does not provide optimal performance.[37] As a result, it is often desirable to avoid conditional statements wherever possible. How this was done for the LENJ routine is now described.

In the vector version of LENJ, the NL index runs over all Lennard–Jones pair potentials and II index runs over trajectories. As mentioned earlier, classical trajectories are independent of each other. By running a set of trajectories in parallel, a dimension is created by the number of trajectories, which is indexed by II. The long vector length over II will tremendously enhance the efficiency of the vector code. Since arrays in FORTRAN are stored in column-major order, care has been taken to arrange II as the leftmost subscripts of arrays so that the FORTRAN arrays are addressed by stride-1 storage. Since most machines can only vectorize the innermost loops, it is the II DO-loop that will be vectorized. Therefore, we do not need to deal with the variable recurrence over the NL DO-loop. Instead of processing "IF statements" inside the DO-loop, they are taken outside the loop. To accomplish this a one time pretreatment of the conditional statements is done to classify the input data. However, due to space considerations, the code for pretreatment is not shown above. Restructuring the program in this manner will also speed up execution in the scalar mode for the LENJ code.

As is done here for LENJ, whenever possible, a divide operation is converted into multiplication of the reciprocal of the dividends, since divide operations take more clock cycles to execute. To further enhance the speed of vectorized VENUS, a partial mathematical reduction of the equations of motion was carried out. A complete mathematical reduction of the equations will be considered in future work. The work of Noid et al.[20] will be very helpful in this regard.

In general, the code for evaluating first-order derivatives of the potential is very complicated in form. This is especially true for a generalized program like VENUS, which allows the user to arbitrarily assign indices to the atoms of the molecular system. The price paid for this user convenience is in the elaborate programming. The need for branching and indirect addressing generates many vector inhibitors. In most cases, these vector inhibitors can be removed by segmenting one DO-loop into several

loops. Also, to achieve the maximum extent of vectorization, some short inner DO-loops are unrolled.

It should be pointed out that subroutine calls are one of the most time consuming instructions. In consideration of the diversity of applications of the VENUS program, the gradient evaluations of potential energy functions are arranged into separate subroutines. Although it may not be a computationally economic structure, it gives users the option to make different combinations of potential energy functions and to add their own potential energy functions if necessary. As a matter of fact, the vectorized structure reduces the number of gradient evaluation subroutine calls by a factor II, where II is the number of trajectories propagated in one run. Analysis of CPU time profiles of vectorized VENUS shows that the CPU time spent on gradient-subroutine-calls is not substantial (less than 0.2% of total CPU).

Vector Length and Core Memory, Switching Algorithm

Classical trajectory calculations are statistical in nature. Dynamical properties of the chemical system are based on the average behavior of a large number of trajectories. As shown in Figure 4, classical trajectories have scattered lengths. This behavior has a direct consequence on the choice of vector length. However, before describing the approach taken in dealing with the problem of changing vector lengths, it is useful to review some general features of vector processors.

Vector processing reduces the machine cycles required to process the set of instructions (addition, multiplication, etc.) in a given FORTRAN DO loop by operating on a number of data elements simultaneously in a pipeline fashion.[24,25] However, the initiation of a vector pipeline is always accompanied by some overhead cost in machine cycles. This additional overhead is responsible for the existence of the vector break length, defined here as the vector length at which vector and scalar processing have equivalent execution times. Thus, for vector processing to yield any benefit the number of elements in the vector pipeline must be large enough so that the loss due to overhead is offset by the cycles saved in simultaneously operating on multiple elements. Vector break lengths are operation dependent, and are approximately 10 and 4 for the IBM 3090 and CRAY YMP, respectively.

Data elements in a vector pipeline are handled by vector registers. The number of elements each vector register is capable of holding is called the section size, and is denoted here as Z. Consequently, the maximum number of elements that may be simultaneously processed in a vector pipeline is equal to Z, the section size. For the IBM 3090, Z is 128 or 256; for the CRAY YMP, Z is 64.

It is important to note that the overhead associated with the start-up of a vector pipeline is the same whether the number of elements processed is 1 or Z, the maximum. When the iteration count of a DO-loop is large, the start up time becomes insignificant. However, for a DO-loop with a small iteration count, the overhead can be a significant fraction of the loop's execution time. As a result, it is often advantageous to set up DO loops so that the number of iterations in the loop approaches the section size, if

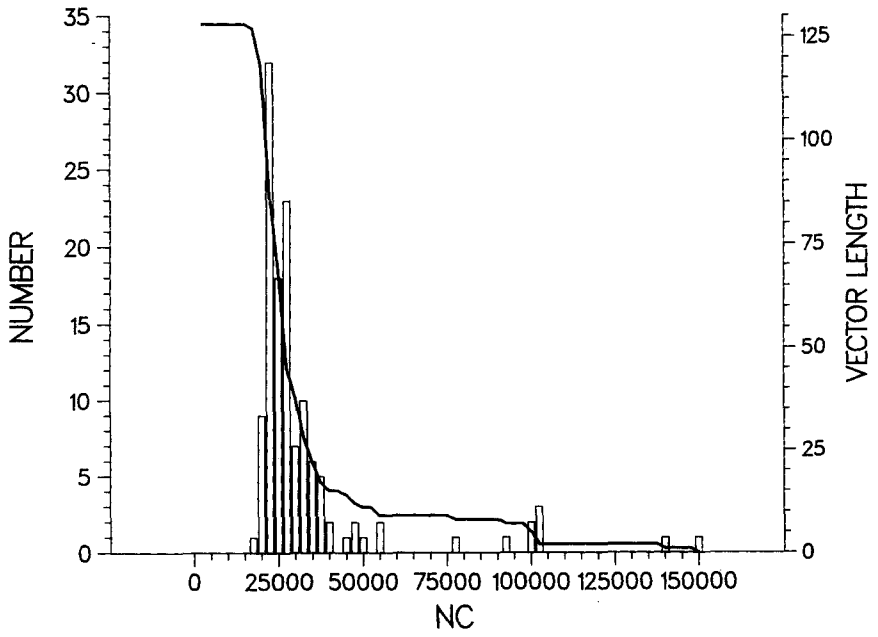

Figure 4. Histogram for distribution of trajectory lengths (number of trajectories vs integration cycles) sampled over association of $H(Ar)_2$ cluster with CH_3 and a plot of the vector length vs. integration cycles.

possible.[37] In this manner, a maximum number of elements can be processed once the overhead cost of starting the vector pipeline has already been incurred. For cases in which the number of iterations in a DO loop is greater than the section size Z, one section of Z elements is processed at a time until the loop is completed.

As discussed in a previous section, the desire to avoid short vector lengths by vectorizing over the number of trajectories was a major motivation for the restructuring of VENUS. In addition, the implementation of a switching algorithm ensures that the number of iterations in vectorized DO loops remains at or near Z even though some trajectories may terminate before others in a given run. How this was done is now discussed.

Assuming that a particular run starts with 120 trajectories, as for the case in Figure 4, the vector length will become shorter and shorter as the calculation progresses. Eventually, the vector break length is reached, but the trajectories must still be propagated until the last one is terminated. This has a deleterious effect on the execution, especially when scattered long-lived trajectories exist, as in the case for CH_3 association with $H(Ar)_{12}$. One way to deal with this problem is to start with a large number of trajectories so that long-lived trajectories will be processed in a reasonable vector length. However, this requires a large II dimension which may saturate the limit of core memory. Clearly, other approaches to solving this problem merit investigation.

A common theoretical approximation[16] for the time t_n needed to perform a calculation of vector length n is

$$t_n = (n + n_{1/2})/r_\infty \tag{18}$$

where r_∞ is the maximum asymptotic rate of computation, and $n_{1/2}$ is the vector length at which half the asymptotic rate is attained. This corresponds to the typical performance curve for a vector machine in Figure 5. It appears from this curve that the longer the vector the better the performance. However, it must be realized that the core memory that is available is limited. It has been mentioned in the second section that as the potential energy surface becomes more complicated the gradient becomes more difficult to define. This effects the computer code since more variables are required. One price for parallel processing trajectories is that the dimensionality of most variables is raised by one; i.e., the II dimension. Directly increasing the length of the II dimension of all array variables, in order to run more trajectories, is virtually inhibited by the amount of memory available in a machine.

Based on the above factors, a switching algorithm is designed. (From a rather exhaustive search of the literature we found no previous description or use of this algorithm. However, it is possible that this algorithm has been used before for simulations much different than the one performed here). Z-1 is chosen as the iteration count of the II DO-loop. This is done to ensure that the number of elements in the vector pipeline is close to the section size, Z. Z-1 is chosen in preference to Z since the arrays are optimally dimensioned to match the iteration count. However, in order to avoid possible memory bank conflict,

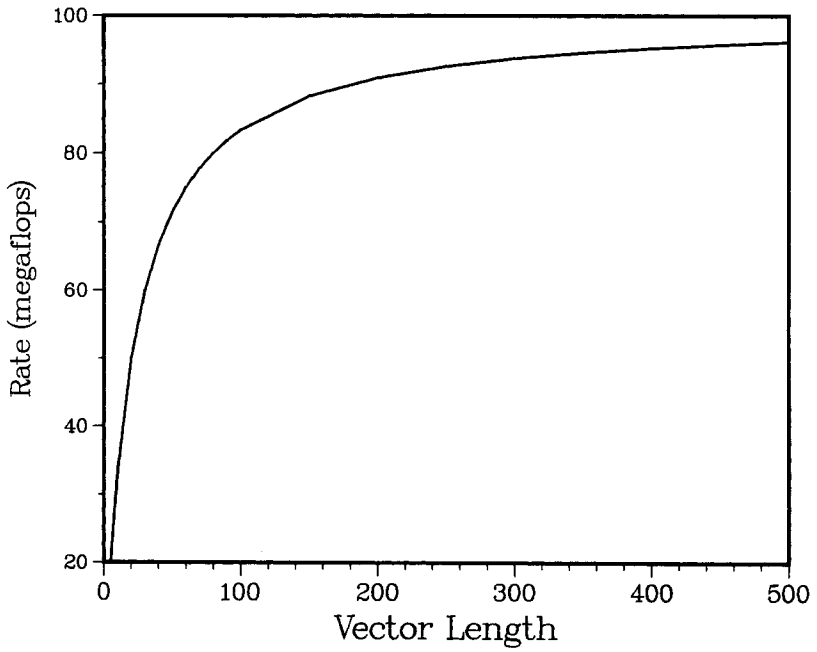

Figure 5. Typical performance curve for a vector machine with $n_{1/2} = 20$, $r_\infty = 100$ megaflops.

arrays should not be dimensioned to sizes which are powers of 2. This is avoided by the use of Z-1. The switching algorithm is depicted in Figure 6. To illustrate the algorithm, assume one wants to run a set of n trajectories, where n is greater than 64. At first, 63 trajectories are initialized and propagated. Once one of 63 trajectories is terminated the 64th trajectory starts. This continues until the nth trajectory is started. Finally, there are only 63 trajectories left to be processed. When one of the 63 trajectories is terminated the 63rd will be shifted to its position; that procedure is continued until only one trajectory remains. The advantages of this switching algorithm are: (1) the vector length remains as Z-1 while the n-(Z-1) trajectories are processed, so that the computer will always run with an efficient vector length; (2) The size of the II dimension is Z-1 instead of the impractical n, which overcomes the limitation of core memory; and (3) The algorithm is easily implemented and requires very little computer time.

TEST RESULTS

Before test results are presented, the three different versions of the program, VENUS, VENUSV, VENUSVS are described. VENUS is the original scalar version of the program, in which trajectories are processed sequentially. VENUSV is the vector version of the program, for which all the modifications mentioned in the above have been made. VENUSVS is the vector version of the program (VENUSV) running in scalar mode.

The VENUSV program is vectorized on both the IBM-3090 and CRAY-YMP. As far as their compilers are concerned, the major difference between these two machines is that the IBM-3090 FORTRAN version 2 compiler is capable of vectorizing outer loops, while the CRAY-YMP CFT77 3.1 compiler can only vectorize the innermost loops. The guideline for optimization is to achieve the maximum extent of vectorization. If not inhibited by a variable dependence, the II DO-loop is always set as the outer loop in the IBM-3090 version of VENUSV. Since only innermost DO-loops can be vectorized on the CRAY-YMP, a number of small inner loops are unrolled to arrange the II DO-loop as the innermost loop. The core of the VENUSV program is essentially the same for both machines, except the order of the DO-loops.

In this article, results are presented for running the different versions of VENUS on the CRAY-YMP. The molecular dynamics system considered here is the association of CH_3 with the $H(Ar)_{12}$ cluster. The potential energy function for this system consists of four Morse potentials, 126 pairwise Lennard–Jones interactions, and a tetrahedral center potential. The latter potential is the most complicated of all the

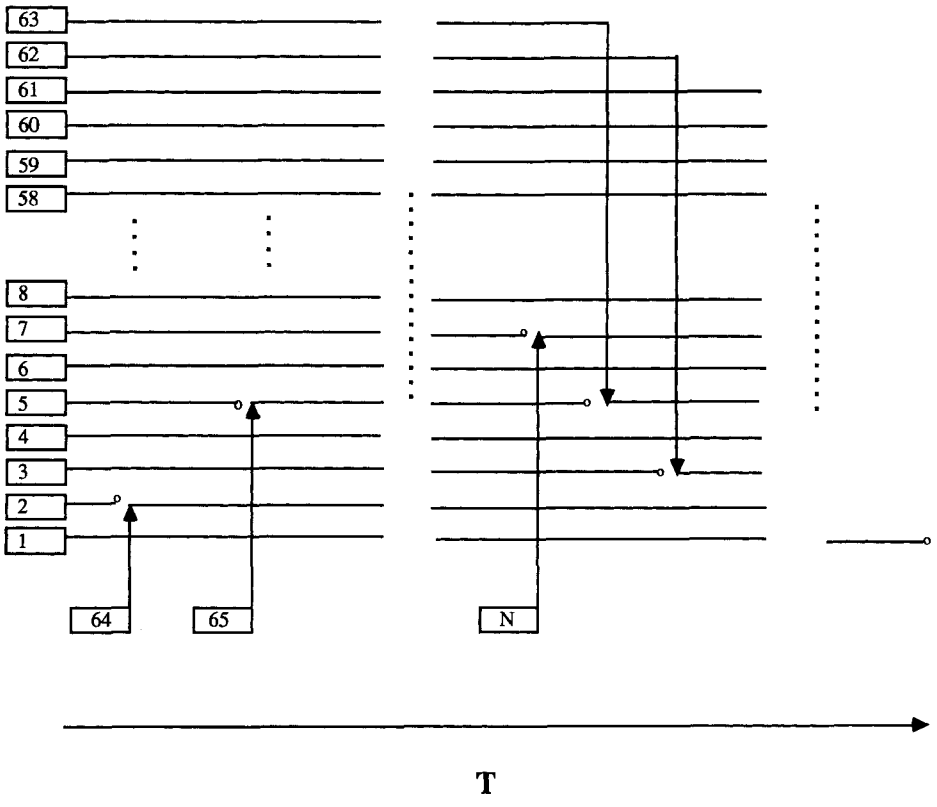

Figure 6. Switching algorithm ⊏⊐— : Trajectories 1 to 63 are initialized, vector length = 63, ⊏⊐↑ : A new trajectory is initialized to fill the position of a completed trajectory, —┘ : Shifting of a trajectory to fill the position of a completed trajectory, vector length decreases by 1, —o : A trajectory is terminated once it is completed.

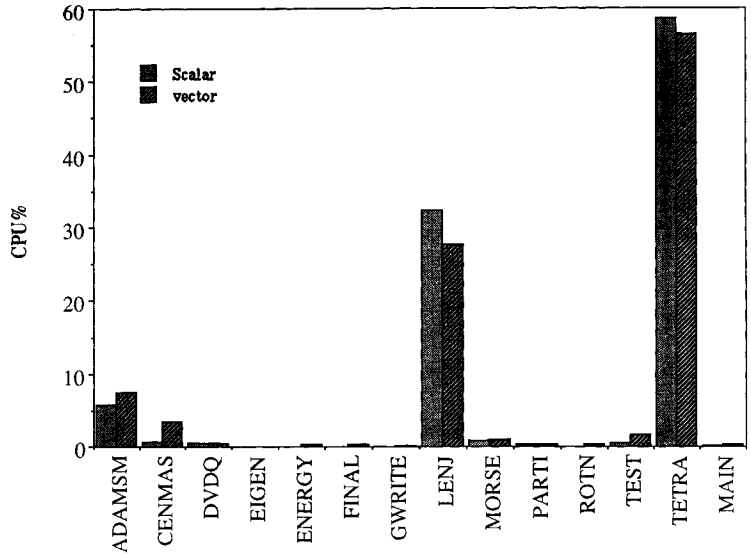

Figure 7. Profiles of execution for VENUSV (vector) and VENUSVS (scalar) applied to the association of CH_3 with $H(Ar)_{12}$ cluster, 63 trajectories are evaluated, each for 7500 cycles.

potentials in VENUS. It is an analytically fitted global potential energy surface to *ab initio* calculations for H association with CH_3 to form CH_4.[31] A complicated set of switching functions is used to describe the analytic potential in going from the reactants H and CH_3 to the product CH_4. Though the results presented here are system dependent, we feel the $H(Ar)_{12}$ + CH_3 system is representative.

Shown in Figure 7 is the relative percentage of CPU time spent on each module of VENUSV and VENUSVS in executing $H(Ar)_{12}$ + CH_3 trajectories on the CRAY-YMP. The utility flowtrace on the CRAY-YMP is used to find the CPU time used by each module. The left-most bar corresponds to VENUSV running in scalar mode (i.e., VENUSVS). To accomplish this the CFT77 compiler directive novector is used to turn off the vector processing. The right-hand side bar corresponds to vector version of the program with vector on (VENUSV). Notice that the two bars have almost the same height. Assuming each subroutine vectorizes to nearly the same extent, the equivalence of the height of the two bars indicate that the most important subroutines are all vectorized.

A comparison was also made between VENUSVS and VENUS. Essentially, this is a test of speed in nonvector machines to see how efficient the VENUSV code is in comparison to the original VENUS code. Again, we used the $H(Ar)_{12}$ + CH_3 system for our test. The execution time is based on 10 trajectories, each executed for 7500 integration cycles. The time spent on each module is analyzed by utility flowtrace. The results are listed in Table I. Overall, the restructuring of the program enhances the program's performance. The significant gain in LENJ comes from the systematic use of scalar loops to note the branching conditions as discussed earlier. The gain seen in MORSE is representative for all simple potentials. It is achieved by a noniterative treatment of the index. The slight decrease in the TETRA speed is typical for a complicated potential. As the potential becomes more sophisticated, the II DO-loop has to be segmented to remove some vector inhibitors. The segmentation of the II DO-loop creates an intermediate temporary storage requirement, which reduces the program's performance.

Since the goal of optimization is to speed up the execution of the program, the important question is what is the ratio of the vector and scalar execution rates? To answer this question, we performed the following test. A set of 63 trajectories for $H(Ar)_{12}$ + CH_3 are propagated for 7500 integration cycles with both VENUSV and VENUSVS. The ratio of the overall execution time is 10.6 (scalar/vector, or VENUSVS/VENUSV). Table II depicts the performance of each module. It should be pointed out that since all trajectories are not completed at 7500 cycles (Fig. 4), the vector program is running with

Table I. Execution time in CPU (s).

	Modules			
Program	LENJ	MORSE	TETRA	ADAMSM
VENUS	154.18	3.66	172.55	17.01
VENUSVS	96.18	2.64	176.94	16.25

Table II. Ratio of execution time (scalar/vector).

	Modules			
	LENJ	MORSE	TETRA	ADAMSM
Ratio	13.6	11.53	11.18	6.17

a full vector length of 63. Therefore, the above ratio may overestimate the efficiency of the vectorized code for trajectories with scattered long-lived complexes involved. One of the most important reasons for designing the switching algorithm is to achieve this maximum efficiency.

The switching algorithm is designed to optimize speed with the existing limit of core memory. How well will this algorithm perform for actual trajectory calculations? Figure 8 depicts the execution time per trajectory as a function of number of trajectories n. For $n > 63$, the order of execution is arranged by the switching algorithm. Considering the fact that the vector break length for the CRAY-YMP is short, a significant amount of speed is gained by the switching algorithm. It is worth mentioning that core memory storage should be treated as precious as CPU time. That is why most supercomputer centers charge memory usage! The switching algorithm speeds up the execution and saves memory.

To illustrate the performance of vectorized VENUS(VENUSV), a hardware performance analysis is carried out. It is done with the utility program HPM in CRAY-YMP. For a set of 63 trajectories propagated for 7500 integration cycles, a performance of 140 MFLOPS is achieved by the vectorized VENUS(VENUSV). It should be pointed out that this supervector performance is achieved by a single central processing unit (CPU).

SUMMARY

Based on the fact that classical trajectory runs are independent of each other, a set of trajectories may be processed simultaneously. The generalized Monte Carlo classical trajectory program VENUS has been reorganized to ensure a long vector processing, and therefore a large extent of vectorization. The most time-consuming subroutines in VENUS, which are the numerical integration and gradient evaluations, are all fully vectorized on both the IBM-3090 and CRAY-YMP. Care has been taken to make efficient use of the following vector features: (1) stride-1 storage is arranged; (2) the divide operation is converted into the multiplication of reciprocals of dividends whenever possible; (3) uses of an "IF statement" inside a DO-loop is minimized; and (4) small inner DO-loops are unrolled. A switching algorithm has been designed to ensure an efficient vector length and a minimal core memory requirement. On the CRAY-YMP the vector version of VENUS achieves a performance of 140 MFLOPS and executes 10.6 times faster than does the scalar version, which extends the range of applicability of trajectory methods to large systems such as clusters, solvation, etc. The methodology we are using here is not only applicable to VENUS,[38] but also to a wide spectrum of molecular dynamics programs. We believe that a combination of vectorization and multitasking for trajectory calculations could speed up the execution even further.

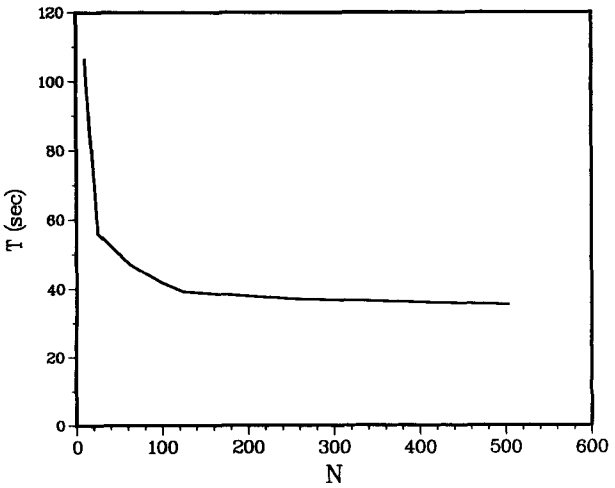

Figure 8. Execution time per-trajectory (T) as a function of total number of trajectories (N). For $N > 63$, the order of execution is arranged according to the switching algorithm as shown in Figure 6. The molecular dynamics system is association of CH_3 with the $H(Ar)_{12}$ cluster. All trajectories are propagated to completion.

This research was supported by the National Science Foundation and IBM Corporation. The calculations were performed at the NSF Pittsburgh Supercomputer Center and the IBM 3090 Vector Facility of the High Performance Computing Solution Development group at Kingston, New York. The authors would also like to acknowledge Dan Eisenhauer and Swamy Kandadai of IBM Kingston for their invaluable assistance.

References

1. D.L. Bunker, *Met Comput. Phys.*, **10,** 287 (1971).
2. R.N. Porter and L.M. Raff, in *Dynamics of Molecular Collisions,* Part B, W.H. Miller, Ed., Plenum, New York, 1976, p. 1.
3. M.D. Pattengill, in *Atom-Molecule Collision Theory,* R.B. Bernstein, Ed., Plenum, New York, 1979, p. 359.
4. D.G. Truhlar and J.T. Muckerman, in *Atom-Molecule Collision Theory,* R.B. Bernstein, Ed., Plenum, New York, 1979, p. 505.
5. W.L. Hase, in *Aspects of the Kinetics and Dynamics of Surface Reactions,* AIP Conference Proceedings No. 61, U. Landman, Ed., AIP, New York, 1980, p. 109.
6. H. Goldstein, *Classical Mechanics,* Addison-Wesley, Reading, Massachusetts, 1950.
7. R.D. Levine and R.B. Bernstein, *Molecular Reaction Dynamics and Chemical Reactivity,* Oxford, New York, 1987.
8. C.L. Brooks III, M. Karplus, and B.M. Pettitt, *Adv. Chem. Phys.*, vol. LXXI, J. Wiley and Sons, New York, 1988.
9. a. W.L. Jorgensen and J.K. Buckner, *J. Phys. Chem.*, **90,** 4651 (1986). b. J. Chandrasekhar, S.F. Smith, and W.L. Jorgensen, *J. Am. Chem. Soc.*, **107,** 154 (1985).
10. a. D. Fincham and D.M. Heyes, *Adv. Chem. Phys.*, **63,** 493 (1985). b. M.L. Klein, *Ann. Rev. Phys. Chem.*, **36,** 525 (1985).
11. X. Hu and W.L. Hase, in preparation.

12. L. Perera and F.G. Amar, *J. Chem. Phys.*, **90,** 7354 (1990).
13. S. Wilson, in *Methods in Computational Chemistry,* Vol. 3, S. Wilson, Ed., Plenum, New York, 1989, p. 1.
14. P.R. Taylor, C.W. Bauschlicher, and D.W. Schwenke, in *Methods in Computational Chemistry,* Vol. 3, S. Wilson, Ed., Plenum, New York, 1989, p. 63.
15. G.H.F. Diercksen, N.E. Gruner, and J. Steuerwald, in *Methods in Computational Molecular Physics,* G.H.F. Diercksen and S. Wilson, Eds., D. Reidel, Dordrecht, Holland, 1982, p. 335.
16. I.S. Duff, *Computer Physics Reports,* **11,** 1 (1989).
17. *Supercomputer Research in Chemistry and Chemical Engineering,* E.F. Jensen and D.G. Truhlar, Eds., ACS Symposium Series 353, American Chemistry Society, Washington, D.C. 1987.
18. P.R. Taylor and C.W. Bauschlicher, *Theor Chim Acta,* **71,** 105 (1987).
19. W.A. Kraus and A.F. Wagner, *J. Comp. Chem.*, **7,** 219 (1986).
20. D.W. Noid, B.G. Sumpter, B. Wunderlich, and G.A. Pfeffer, *J. Comp. Chem.*, **11,** 236 (1990).
21. D.C. Rapaport, *Computer Physics Reports,* **9,** 1 (1988).
22. D.L. Cochrane and D.G. Truhlar, *Parallel Computing,* **6,** 63 (1988).
23. W.L. Hase, R.J. Duchovic, X. Hu, K.F. Lim, D.-H. Lu, K.N. Swamy, S.R. Vande Linde, and R.J. Wolf, VENUS, to be submitted to *QCPE.* VENUS is an enhanced version of MERCURY. W.L. Hase, MERCURY, *QCPE,* **3,** 453 (1983).
24. *Designing and Writing FORTRAN Program for Vector and Parallel Processing* (Publication No. SC23-0337-00), IBM, 1986.
25. *IBM VS FORTRAN Version 2.4 Programming Guide* (Publication No. SC26-4222-04), IBM, 1989.
26. *IBM VS FORTRAN Version 2.4 Language and Library Reference Manual* (Publication No. SC26-4221-04), IBM, 1989.
27. *CFT77 Reference Manual* (Publication No. SR-0018), Cray Research Inc., Mendota Heights, Minnesota, 1987.
28. *UNICOS User Commands Reference Manual* (Publication No. SR-2011), Cray Research Inc., Mendota Heights, Minnesota, 1988.
29. *Cray CFT Optimization Guide,* On Line Documents, Pittsburgh Supercomputer Center.
30. G.C. Lie and E. Clementi, *J. Chem. Phys.*, **62,** 2195 (1975).
31. a. R.J. Duchovic, W.L. Hase, and H.B. Schlegel, *J. Phys. Chem.*, **88,** 1339 (1984) b. W.L. Hase, S.L. Mondro, R.J. Duchovic and D.M. Hirst, *J. Am. Chem. Soc.*, **109,** 2916 (1987).
32. E.B. Wilson, J.C. Decius, and P.C. Cross, *Molecular Vibrations,* McGraw-Hill, New York, 1955.
33. C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations,* Prentice-Hall, Englewood Cliffs, New Jersey, 1971.
34. L. Lapidus and J.H. Seinfeld, *Numerical Solution of Ordinary Differential Equations,* Academic Press, New York, 1971.
35. O. Teleman and B. Jönsson, *J. Comp. Chem.*, **7,** 58 (1986).
36. W.B. Street, D.J. Tildesley, and G. Saville, *Mol. Phys.*, **35,** 539 (1978).
37. Course materials from IBM vendor workshop at Kingston, Dallas.
38. We plan to ultimately make this vectorized VENUS available through QCPE. Until that is done, interested parties can receive copies of the program by contacting the authors.