JOURNAL OF COMPUTATIONAL PHYSICS **94**, 454–466 (1991)

Optimization of Direct Simulation Monte Carlo (DSMC) Codes for Vector Processing

G. PRISCO

Analysis and Programming Section, Mathematical Support Division, ESTEC, European Space Agency, Noordwijk, The Netherlands

Received May 8, 1989; revised December 27, 1989

The use of direct simulation Monte Carlo (DSMC) codes in the analysis of low-density flow or, more generally, flows containing regions of strong translational non-equilibrium, is justified by briefly reviewing the limitations of codes based upon a macroscopic approach. The general structure of a DSMC algorithm is described briefly, and a class of algorithms used within DSMC codes for the simulation of molecular interactions is discussed in greater detail. Basic principles of vector processing are then reviewed, and it is finally shown that the simulation of molecular interactions, which was previously thought to be non-vectorizable, can on the contrary be vectorized with high efficiency. © 1991 Academic Press, Inc.

1. INTRODUCTION

Direct simulation Monte Carlo (DSMC) codes are a powerful tool for the computer simulation of low-density flows [1]. These flows, such as those encountered by a reentry vehicle (space shuttle, Hermes) in the upper atmosphere or those found in low-density gas-dynamic devices, are characterized by a large value of the Knudsen number for which the macroscopic equations (Euler, Navier–Stokes) do not hold. The Knudsen number is defined as the ratio of the mean free path in the flow to the characteristic scale of length of spatial gradients of macroscopic flowfield variables and may be regarded as a measure of the degree at which translational non-equilibrium effects are relevant to the analysis of the flowfield. A convenient rule of thumb for estimating the order of magnitude of the Knudsen number of a reentry flow is that of taking the Knudsen number as the ratio of the mean free path to a characteristic dimension of the reentry vehicle (it is assumed that the latter is of the same order as the characteristic length of macroscopic gradients). Typically, for a reentry flow the Knudsen number increases as the density of the flow decreases, and consequently the analysis of low-density reentry flows lies outside the range of validity of the macroscopic approach. The reverse is not always true because even within flows which are almost everywhere well described by the macroscopic equations there may exist local regions of strong non-equilibrium, as is the case across thin shock waves at high Mach numbers. For non-vanishing values of the Knudsen number the flow must be considered as

0021-9991/91 $3.00

composed of individual molecules, and the relevant equation is the Boltzmann equation, a convenient form of which is

$$\frac{df}{dt} = L(f), \tag{1}$$

where $f(x, v, t)$ is the single-particle distribution function defined in the six-dimensional phase space spanned by the position x and velocity v of a typical particle in the three-dimensional physical space, and the collision integral L is a rather complex integral functional of f accounting for the effect of 2-molecular interaction. When L is very small the effect of collisions may be neglected and the flow is said to be in the free-molecular regime. regime. When L increases the effect of collisions must be considered (but the number of collisions is still too low to justify the use of the macroscopic approach) and the flow is said to be in the transition regime. It can be shown with sufficient generality [2, 3] that, in the limit of low Knudsen numbers, the kinetic approach may be collapsed to the macroscopic approach. In this case the number of collisions is sufficienctly high to relax the flow to local translational equilibrium in a time much shorter than any characteristic scale of time of the macroscopic evolution of the flow. The task of computing a solution of the Boltzmann equation by using the standard techniques of numeric analysis is difficult because of the large number of independent variables (six instead of three).

DSMC codes permit the solution of the Boltzmann equation to be obtained without explicitly manipulating it, but rather by tracing the phase-space evolution of a set of representative molecules. The molecules are generated randomly and followed during their lifetime in the computational domain, and they may interact among themselves and with a body in the flow according to specified interaction models. Flow resolution is obtained by referring the molecules to a spatial lattice and computing the thermodynamic variables of the flow (density, pressure, temperature, chemical composition) by means of averages performed over the molecules in the neighborhood of each lattice point.

The interaction models employed must be sufficiently complex in order to account for real-gas effects (chemical reactions, excitation of vibrational levels, ionization, and consequent electromagnetic effects) associated with the high temperatures encountered in typical reentry flows [4, 5]. Since DSMC codes may require a very long processing time, it is extremely important to design them in such a way as to take full advantage of the capabilities of vector computers, such as the IBM 3090 Vector Facility. This requirement has caused many authors to discard the most established method for the simulation of the gas–gas interaction (Bird's algorithm) on the basis that it seems to be non-vectorizable, and that the gas–gas interaction accounts for a more and more significant fraction of the overall processing time when the simulated flow advances in the transition regime (more individual molecular interactions must be simulated). On the other hand, Bird's method is the one which models more closely the physics which is actually going on, so that it seems unjustified to discard it on the basis of the efficiency of its computer implementation without having more extensively analyzed the issue. It will be shown in this article that, in the most interesting cases, Bird's algorithm can be vectorized with high efficiency.

2. RGFLOW—A Program for the Simulation of Low-Density Flows

RGFLOW is a FORTRAN 77 program which has been developed by the author within the framework of the effort made at the European Space Agency (ESA) to develop efficient simulation software for the Hermes project. It simulates a three-dimensional low-density, electically neutral, multicomponent, and chemically reacting gas flow past a body by making use of a DSMC algorithm. The program has been extensively tested with respect to the thermodynamical and chemical behaviour of the simulated flow, and the results have always been in excellent agreement with theoretical results (in those cases when theoretical results are available) and other simulation programs. Initial positions and velocities of the molecules are sampled from the appropriate statistical distributions. A very concise list of the main computational steps performed by the program is given below.

All molecules are moved along straight segments corresponding to the time-step and their velocity vectors. The positions and velocities of the molecules crossing the boundaries of the computational domain are redefined in a way consistent with the boundary conditions. For instance, if the computational domain is a typical region in the freestream the velocities of the incoming molecules are sampled from the appropriate statistical distribution. If the computational domain is a typical region in a closed system (gas in a box) the velocities of the incoming particles coincide with those of the outgoing particles.

For all molecules hitting the surface of the body placed in the flow, the surface interaction is computed according to a diffuse reflection model (the reflected molecules are sampled from a statistical distribution at thermodynamic equilibrium with the wall). The body itself is a sphere in the current version of the program. A version under development will allow the user to specify the shape of the body by using the output of a CAD package (CAEDS is being considered). The same graphical interface will be used for display of results.

The gas-gas interaction is computed according to the model described in the next section.

Each molecule is assigned to the nearest lattice point. This can be done in a number of ways, for instance, by defining a set of local arrays containing pointers to positions in the particle array. This approach has the defect of leading to unnecessarily high storage requirements if the density fluctuations are high. In RGFLOW it has been preferred that the particle array at each time-step be reordered in such a way as to assign contiguous positions in the particle array to molecules assigned to the same lattice point. The processing time for the reordering is a linear function of the number of molecules, and consequently the time penalty is not high.

The thermodynamic variables at each lattice point are computed by means of averages performed over the molecules assigned to the lattice point. If requested, the time-average of the thermodynamic variables is computed.

The aerodynamic forces on the body in the flow are computed by means of averages performed over the molecules which have hit the surface of the body. If requested, the time-average of the aerodynamic forces is computed.

It is worth discussing some points concerning the time-averages mentioned above. If the flow is steady, as may be assumed to be the case for reentry flows, time-averaging the results obtained over a smaller statistical sample is equivalent to directly processing a larger statistical sample (ergodic flow). Consequently, for ergodic flows the limits in computer storage capabilities do not impose corresponding limits to the statistical significance of the simulated sample. This point should be considered in view of the fact that a general feature [15] of Monte Carlo methods is a convergence proportional to the square root of the dimension of the statistical sample (in order to gain one more significant digit in the output, one must increase by two orders of magnitude the number of particles in the simulation).

After a first development stage on an IBM 3090 mainframe, the program has been moved to the enhanced architecture IBM 3090 Vector Facility [6] and it has been extensively modified in order to take the best advantage of the vector hardware. The task of vectorizing the code not dealing with the gas–gas interaction has been rather undemanding. A large portion of the code vectorizes, and the program is faster than the scalar version by a factor of 4 to 6. The vectorization of the remaining part of the code is discussed in the following sections.

3. Physical Models for the Simulation of the Gas–Gas Interaction

RGFLOW is able to simulate flows where the following two different modes of chemical reaction take place,

$$A + B \leftrightarrow C + D \tag{2}$$

and

$$A + M \leftrightarrow B + C + M, \tag{3}$$

where A, B, C, and D are defined molecular species and M can be any molecule acting as a catalyzer for the dissociation-recombination reaction $A \leftrightarrow B + C$. The program can manage up to five different molecular species and any number of chemical reactions of the form (2) or (3). Note that the first reaction is 2-molecular in both directions, whereas the second is 2-molecular in the forward direction and 3-molecular in the backward direction. These two reaction modes are sufficient for simulating the chemistry of the earth and planetary atmospheres for which higher order reactions are highly improbable. For instance, the chemical kinetic model nomally used for the neutral atmosphere in the range of parameters relevant for the study of typical reentry flows contains the five molecular species O, O_2, N, N_2, NO (atomic and molecular oxygen, atomic and molecular nitrogen, nitric oxide) and the following six reactions:

$$\begin{array}{ll} \text{(a)} & O_2 + M \leftrightarrow O + O + M \\ \text{(b)} & N_2 + M \leftrightarrow N + N + M \\ \text{(c)} & NO + M \leftrightarrow N + O + M \\ \text{(d)} & NO + O \leftrightarrow N + O_2 \\ \text{(e)} & NO + N \leftrightarrow O + N_2 \\ \text{(f)} & NO + NO \leftrightarrow N_2 + O_2. \end{array} \tag{4}$$

The first three reactions above are dissociation/recombination reactions of the form (3), the other three are shuffle (atom exchange) reactions of the form (2).

The chemical kineties of the model is defined by assigning to each chemical reaction values for the activation temperature T_a, the reaction probability (steric factor) P_r, and the lifetime of the recombined molecule Lft (used only for 3-molecular recombination reactions). The energy corresponding to T_a (A, B) is the minimum required collision energy for a collision between molecules belonging to the species A and B to result in a chemical reaction producing specified molecular species. When the collision has the required or excess energy the chemical reaction occurs with probability $P_r(A, B)$. The required reaction probability is simulated by sampling a random number R_f from a uniform distribution between 0 and 1 (random fraction) and computing the chemical reaction only if

$$P_r(A, B) > R_f. \tag{5}$$

$P_r(A, B)$ is a weak function of the excess collision energy and other translational and internal parameters of the colliding pair and is considered as a constant in RGFLOW since, as even the order of magnitude of the steric factor is only poorly known for many relevant reactions, it would be pointless to try to follow slight variations in its value. In recombination reactions the recombined molecule A in the backward reaction (3) must be stabilized by another collision with a third molecule M acting as a catalyzer during a time Lft(B, C) of the order of the vibrational period of A. The third body carries away that part of the energy released in the recombination which would otherwise lead to a spontaneous redissociation.

One T_a, P_r, and Lft are specified, the chemical behaviour of the system in dictated by its collisional behaviour. This is a very central point in DSMC codes and can be specified by assigning to each molecular species values for the molar mass m, the molecular diameter d, and the number of particles N, which will generally be a function of the lattice point and the time. The unrealistically low number of particles which can be simulated, taking into account computer storage and speed limitations, can be compensated for by assigning unrealistically high values to the molecular diameters (and thus to the collision cross sections) in such a way as to define a mean free path of the desired magnitude.

The collisional model employed in RGFLOW is fundamentally similar to the hard-sphere models described in the DSMC literature [1, 11, 12]. At each lattice point, colliding pairs (1, 2) are selected randomly with probability proportional to the collision cross section and the relative velocity vr(1, 2). The collision cross section for hard-sphere molecules is defined as

$$s(1,2)=\frac{\pi}{4}(d(1)+d(2))^2. \tag{6}$$

The choice of colliding pairs distributed with the specified probability is made by accepting a random colliding pair only if

$$\frac{S(1,2)}{P(i)}>R_f, \tag{7}$$

where R_f is a random fraction, $S(1,2)$ is vr(1, 2) times the collision cross section, and $P(i)$ is the current value at the lattice point i of an estimate for the maximum value of S. $P(i)$ is dynamically updated by assigning to it the value of S when the latter is greater than the previous estimate. It remains to compute the total number of collisions occurring in a time-step, which is done by incrementing a local time counter at each collision until it exceeds the time-step. The necessary increment D_t of the time counter is given by

$$D_t=\frac{2h^3}{s(1,2)\,n(i)^2\,\mathrm{vr}(1,2)}, \tag{8}$$

where h is the local average lattice step and $n(i)$ is the number of molecules currently assigned to the lattice point. The use of the time counter avoids computing the local average value of the relative velocity which would be time-consuming (the processing time corresponding to this computation would be proportional to the square of the number of molecules). It should be noted that RGFLOW makes use of a single collisional time counter per lattice point, whereas most published DSMC algorithms use separate time counters for collisions between different molecular species (which would require a greater storage). Other authors avoid the use of time counters by using efficient (requiring a processing time proportional to the number of particles) methods for estimating the number of molecular collisions occurring within the time-step. These "no-time counter" methods are, however, equivalent to those employing time counters in respect of processing time [16], and consequently either method can be used. Moreover, as it will be shown in the following section, the use of a time counter does not prevent the code from being vectorized.

Three-molecular collisions must be considered in order to account for recombination reactions. In RGFLOW this is done as follows. After a collision (B, C) a third molecule M is selected randomly and the time increment D_t which would be added to the local time counter after a collision between M and the recombined molecule is computed. D_t is then divided by a factor $2/n(i)$ corresponding to the probability for the collision pair chosen randomly immediately after the (B, C) pair to include the recombined particle. This scaled value of D_t (DT) is an estimate of the total time which would be added to the time counter before the next collision involving the recombined molecule. The recombination is then filtered through an acceptance/rejection step based on the ratio of DT to $\mathrm{Lft}(B, C)$,

$$\frac{DT}{\mathrm{Lft}(B, C)} < R_f, \tag{9}$$

where R_f is a random fraction. If the test is pased the recombination is accepted, otherwise only the mechanical collision (B, C) is computed.

When a collision results in a chemical reaction, the energy of the outcoming pair is decremented by the energy corresponding to the activation temperature for endothermic reactions. For exothermic reactions the energy of the outcoming pair is incremented by an amout corresponding to the negative of the input value of the activation temperature, which for exothermic reactions is a negative number whose only function is to express the energy released (the activation temperature itself may generally be taken to be zero for exothermic reactions). The new values of the collision energy and of the molecular species of the colliding pair (these parameters also define a new value for the relative velocity) are used in the computation of the mechanical collision, which uses two random deflection angles. The idea of simulating the gas–gas interaction by means of individual interactions between random partners and that of using local time counters in order to keep track of the number of collisions, which together reproduce the correct non-equilibrium behaviour of the flow, are due to Bird [1], and numerical algorithms of this sort are called Bird's algorithms. In the literature it is often claimed that Bird's algorithm is non-vectorizable, this being the reason why many authors [13, 14] shifted to other methods (Nambu's method or low discrepancy codes). It seems, however, that Bird's method is still the most satisfactory one with respect to physics, because both the conservation laws and the requirement of random microscopic behaviour are strictly satisfied. It is therefore worthwhile, in view of the considerations exposed in the Introduction, to spend some effort in vectorizing the method.

4. Optimization for Vector Processing

In vector computers [6–10] vectorizable loops are executed on vector registers which permit processing different iterations of the loop in parallel. A single processor controls the computation (as opposed to truly parallel execution). As in scalar computers, a fast cache memory is used to hold data which have been recently used. Reusing data in cache leads to improvements in the processing speed, which can be particularly significant when executing in vector mode. In order to be able to make use of the vector hardware, a computational loop must be such that the processing pertaining to a given value of the iterative subscript is independent of the processing pertaining to previous values of the iterative subscript. This very basic limitation may be alternatively stated by saying that the otput of the loop must be independent of the order of execution. This requirement sometimes rules out the possibility of spontaneous vectorization of existing codes, in which case a significant amount of recoding may need to be done in order to take advantage of the vector hardware. Besides this and other limitations dictated by the logic of parallel execution, the software developed must often face limitations in the performance of vector compilers (even smart compilers are not smart enough sometimes). This may be the case, for instance, when the data access patterns are made complex by the extensive use of indirect addressing. In these cases, even if the code does not contain true data dependencies, the compiler may sometimes not be able to establish this fact by analyzing the code, and the loop is consequently flagged as probably non-vectorizable. However, most vector compilers offer the possibility of bypassing the decisions resulting from the compiler analysis by means of compile-time directives. Even when a section of code vectorizes, the speedup with respect to scalar execution sometimes does not reflect the full potentialities of vector processing. This may be due, for instance, to the presence of a significant amount of conditionally executed code (when a loop containing a conditionally executed block executes in vector mode the statements in the block are always processed, and undesired results are discarded after the execution by using a vector mask register containing truth values for the condition to be tested). Another limiting factor may be an inefficient data access pattern which does not optimize the use of cache (that is, requires unnecessary transfers of data from main memory to cache). In order to establish realistic hopes concerning the speedup which may result from vectorization, it must be remembered that the fraction of non-vectorizable code dictates an ultimate limit for the speedup of a program. This is summarized by Amdahl's law,

$$\text{vector/scalar speedup} = \frac{1}{1 - v + v/x}, \tag{10}$$

where v isthe fraction of vectorized code and x is the average local vector/scalar speedup. For instance, a program containing 20% of non- vectorizable code can execute faster on a vector processor only up to a factor 5, which would correspond to infinitely fast vector hardware. Taking into account Amdahl's law and the fact that complex software systems are bound to contain a sizeable fraction of non-vectorizable code, the IBM 3090 Vector Facility has been designed in such a way as to give its optimum cost/performance ratio to systems containing a fraction of non-vectorizable code of the order of 20–30%. In other words, the machine has been tuned having in mind an operating point corresponding to a global speedup for a complex software system between 3 and 5 (of course, the peak local speedup for well-vectorizable loops is much higher), and statistical analysis of the performances have shown that this speedup may sometimes require considerable effort (it is normally claimed that a reasonable vector program speedup is 1.5 to 3). It may be seen that vectorizing the algorithm for the simulation of the gas–gas interaction described in the previous section constitutes a task which is far from straightforward. The computational loop which is to be performed at each lattice point in order to deal with the local set of molecular interactions taking place in the time-step is a REPEAT ... UNTIL loop whose number of iterations is not known in advance. Moreover, the computational loop contains nested REPEAT ... UNTIL loops, the exit from which is triggered by the occurrence of conditions defined at run-time (for instance, the random choice of candidate colliding pairs must be repeated until a pair is accepted). The necessity of establishing an exit condition from a computational loop prevents vectorization (state of the art vector compilers can only vectorize FOR loops with a fixed number of iterations). Another problem arises from the use of a significant number of independent random fractions. On the one hand, using a subroutine call for the generation of each random fraction would, apart from preventing vectorization, be very inefficient. On the other hand, the use of a single subroutine call for filling a buffer of independent random fractions is made difficult by the fact that the number of independent random fractions in the buffer is not known in advance and its order of magnitude cannot be estimated with acceptable accuracy (both the uncertainty in the number of iterations of the loop over the set of accepted collisions and the uncertainty in the number of iterations of the internal loops over candidate colliding pairs cooperate in creating the uncertainty in the dimension of the buffer of random fractions). The only way out is that of drastically overdimensioning the buffer, which would unfortunately be a most inefficient solution with respect to both processing time and memory occupation.

Even if the computational loops dealing with the set of local molecular interactions do not appear to be vectorizable to any useful extent, the computation of the global set of molecular interactions may be very efficiently vectorized by noting that the logical structure of the local computational loop, however complex, is the same for all the lattice points (whose number will be significant in a relevant application) and that different data are processed at different lattice points (each molecule is assigned to a single lattice point). Consequently, a better approach is that of vectorizing the computation of the global set of molecular interactions over the set of lattice points. This can be done by means of assigning to each lattice point a status flag, whose function is that of remembering at which point the computation of local molecular interactions at each lattice point has been left after the last (vectorized) sweep over the set of lattice points. A convenient choice for the set of values for the status flag is described below:

1. A candidate colliding pair is to be chosen and checked to ensure that the same molecule has not been selected twice and that neither were deleted in a previous chemical recombination.

2. A candidate colliding pair (already defined) is to be checked for the requirement of a collision probability proportional to the cross section and the relative velocity.

3. A candidate third molecule acting as a catalyst for a recombination reaction is to be chosen and checked to ensure that it does not coincide with one of the two recombining molecules, and that it has not been deleted in a previous chemical recombination.

4. The chemical interction between two colliding molecules must be accounted for. This includes changing the molecular species of the pair as well as the collision energy and relative velocity, and may include creating or deleting molecules.

5. The mechanical collision between the two molecules of the colliding pair is to be computed. The parameters identifying the two molecules do not coincide at this point with their initial specifications.

6. The required number of local collisions has already been computed at the lattice point, and consequently no more processing is needed at the lattice point.

States 1 and 2 are considered as different because, if the simple test in status 1 fails, there is no need to perform the computationally heavier test in status 2. The computation to be done at lattice points in status 4 depends on the reaction mode, and consequently status 4 is more conveniently expanded into a set of three states

TABLE I

State Transition Diagram

Initial status	Final status	Triggering condition
1	2	Colliding couple accepted
	1	Colliding couple rejected
2	3	Colliding couple accepted and recombination occurring
	4	Colliding couple accepted and other reaction occurring
	5	Colliding couple accepted and no reaction occurring
	1	Colliding couple rejected
3	4	Recombination accepted
	5	Recombination rejected
4	5	Always
5	6	Time counter > time-step
	1	Time counter < time-step

in order to avoid the occurrence of conditionally executed code. However, the three resulting states behave in the same way with respect to the transition between different states, which are summarized in the state transition diagram in Table I. The vectorized sweep over the set of lattice points consists simply of performing at each lattice point the action appropriate to the current value of its status flag. The computation of the gas–gas interaction in the time-step is finished when all of the lattice points are in status 6. The number N_r of random fractions which must be fed to the process at each sweep is known in advance and is given by

$$N_r = N(1)\,N_s(1) + \cdots + N(5)\,N_s(5), \tag{11}$$

where $N(i)$ is the number of lattice points in status i and $N_s(i)$ is the number of random fractions which must be fed to the computational process at a single lattice point in status i. The possibility of using efficient vector subroutines for the generation of blocks of random fractions, such as those provided by the IBM library ESSL, gives rise to part of the gain in speed of the vector algorithm described here.

It is clear that this algorithm for the computation of the gas–gas interaction is vectorizable in principle since, as the sets of molecules assigned to different lattice points are disjoint, the processes pertaining to different lattice points operate on disjoint sets of data and are consequently independent. On the other hand, some effort is necessary in the coding stage in order to overcome limitations in the compiler's ability to analyze fully candidate loops for vectorization. Moreover, the details of the implementation affect the efficiency of the code and set a limit to the ultimate gain in speed with respect to scalar execution. It is consequently very important, as discussed above, to limit as much as possible the fraction of conditionally executed code (IF blocks) and to use data structures which optimize the reuse of data in cache. A number of possible templates for the implementation of the vector algorithm have been tested, and it has been found that the scheme described below gives the best results.

Each sweep is made over the set of lattice points pointed to by the array of pointers I, $I(n)$ being the nth lattice point in status <6. Clearly, the number N_{lp} of elements in I decreases as more and more lattice points reach status 6, and the computation is finished when I is empty. Moreover, the ordering of pointers in I corresponds to the ascending order of the status flags of the lattice points which they define. So, the first $N(1)$ elements in I contain pointers to lattice points in status 1, the next $N(2)$ contain pointers to lattice points in status 2, and so on. The computational process which is to be performed over each lattice point is consequently dictated by the order in which the lattice points are considered. This eliminates the need for IF blocks which would slow down the computation.

In Table II the implementation of the vector algorithm described above is explained in more detail, using the keywords REPEAT ... UNTIL and FOR ... TO to indicate respectively a loop which iterates until the condition following UNTIL is true, and a loop whose number of iterations is fixed.

It is interesting to compare the method described above to that described in [17]

TABLE II

Computational Algorithm

```
REPEAT
  $N(1)=0$
  ...
  $N(5)=0$
  FOR $n=1$ TO $N_{lp}$
    $N(\text{status}(I(n)))=N(\text{status}(I(n)))+1$
  END FOR
  Update the array $I$ and sort it
  in such a way as to have the pointers $I(n)$
  ordered in ascending values of status($I(n)$).
  $N_{lp}=N(1)+\cdots+N(5)$
  Generate $N_r$ random fractions.
  FOR $n=1$ TO $N_{lp}$
    Apply to relevant data at the lattice point $I(n)$ the
    process appropriate to the value of the status flag
    corresponding to $n$.
    Update status($I(n)$) as in the state transition diagram.
  END FOR
UNTIL $N_{lp}=0$
```

for the simulation of a single-component, non-reacting gas in a CRAY-1S vector environment, which also permits a significant vector/scalar speedup to be obtained by vectorizing the computation of the gas–gas interaction over the set of lattice points. This method does not use a status flag, and consequently does not permit one to perform simultaneously (in vector mode) the processing pertaining to lattice points in different states. Rather, the initial selection of colliding pairs is vectorized over a set of lattice points, and the new selections required at those points, where the chosen pair fails the acceptance test, are performed in scalar mode. Only after all of the lattice points in the set have been assigned a validated colliding pair is the final computation of the mechanical collision executed in vector mode.

5. Conclusions

Several test cases have been run in order to compare the performance of the vector algorithm to that of the scalar algorithm. The system used in the test runs has been an interacting gas with a few chemical components, represented by a set of molecules of size ranging from a few hundreds to a few tens of thousands an embedded in a computational grid with a number of lattice points ranging from a few tens to a few thousands. Since the gas–gas interaction is decoupled from the gas–surface interaction, the test system has not included a body interacting with the flow. The processing time has been accurately monitored by using the timing option provided by the interactive debugging facility of the IBM vector FORTRAN compiler.

The parameters which influence the vector/scalar speedup are the number of lattice points, the average number of molecules per lattice point, and the average number of collisions per lattice point (which, for given values of the other parameters, can be tuned by changing the value of the molecular diameter for each specie).

The test runs have been performed by tuning the parameters above and holding the other parameters fixed. The vector/scalar speedup has always been greater than 3 and is close to 4 in most cases. Moreover, the vector/scalar speedup becomes better as each of the three parameters quoted increases. Since these are obviously the more interesting cases (a run meant to produce reliable results must handle at least some thousands of lattice points and some tens of thousands of molecules; moreover, pushing the simulated system into the transition regime increases the number of collisions per lattice point). It is clear that the vector algorithm described here leads to a very significant improvement in the performance of DSMC codes using Bird's interaction model.

Acknowledgments

The author expresses his gratitude to G. L. Webb (ESA), J. C. Lewis (ESA), and R. Houweling (IBM Holland) for their helpful suggestions and comments.

References

1. G. A. Bird, *Molecular Gas Dynamics* (Clarendon Press, Oxford, 1976).
2. M. N. Kogan, *Rarefied Gas Dynamics* (Plenum, New York, 1969).
3. W. G. Vincenti and C. H. Kruger, *Introduction to Physical Gas Dynamics* (Wiley, New York, 1965).
4. J. D. Anderson, *Modern Compressible Flow with Historical Perspective* (McGraw–Hill, New York, 1982).
5. J. D. Anderson, *Hypersnic and High Temperature Gas Dynamics* (McGraw–Hill, New York, 1989).
6. S. G. Tucker, *IBM Syst. J.* **25**, 4 (1986).
7. D. H. Gibson *et al.*, *IBM Syst. J.* **25**, 36 (1986).
8. R. S. Clark and T. L. Wilson, *IBM Syst. J.* **25**, 63 (1986).
9. J. McComb and S. Schmidt, *IBM Syst. J.* **27**, 404 (1988).
10. H. Samukawa, *IBM Syst. J.* **27**, 453 (1988).
11. G. A. Bird, "Monte Carlo Simulation in an Engineering Context," in *Progress in Astronautics and Aeronautics, Vol. 74*, edited by S. Fisher (AIAA, New York, 1981), p. 239.
12. G. A. Bird, "Simulation of Multi-Dimensional and Chemically Reacting Flows," in *Rarefied Gas Dynamics*, edited by R. Campargue (CEA, Paris, 1979), p. 265.
13. K. Nambu, *J. Phys. Soc. Japan* **45**, 2042 (1980).
14. H. Babovski *et al.*, "Low-Discrepancy Method for the Boltzmann Equation," in *Progress in Astronautics and Aeronautics, Vol. 118*, edited by E. P. Muntz, D. P. Weaver, and D. H. Campbell (AIAA, Washington, DC, 1989), p. 85.
15. M. Rosenblatt, *Random Processes* (Springer-Verlag, New York, 1974).
16. G. A. Bird, "Perception of Numerical Methods in Rarefied Gasdynamics," in *Progress in Astronautics and Aeronautics, Vol. 118*, etited by E. P. Muntz, D. P. Weaver, and D. H. Campbell (AIAA, Washington, DC, 1989), p. 211.
17. E. Meiburg, "Vectorization of the Direct Simulation MonteCarlo Technique," in *Vectorization of Computer Programs with Application to Computational Fluid Dynamics*, edited by W. Gentzsch (Vieweg Verlag, Wiesbaden, 1984), p. 235.