JOURNAL OF COMPUTATIONAL PHYSICS **126,** 328–342 (1996)
ARTICLE NO. 0141

# Scalar and Parallel Optimized Implementation of the Direct Simulation Monte Carlo Method

STEFAN DIETRICH AND IAIN D. BOYD

*Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, New York 14853*

Received April 11, 1995; revised September 29, 1995

This paper describes a new concept for the implementation of the direct simulation Monte Carlo (DSMC) method. It uses a localized data structure based on a computational cell to achieve high performance, especially on workstation processors, which can also be used in parallel. Since the data structure makes it possible to freely assign any cell to any processor, a domain decomposition can be found with equal calculation load on each processor while maintaining minimal communication among the nodes. Further, the new implementation strictly separates physical modeling, geometrical issues, and organizational tasks to achieve high maintainability and to simplify future enhancements. Three example flow configurations are calculated with the new implementation to demonstrate its generality and performance. They include a flow through a diverging channel using an adapted unstructured triangulated grid, a flow around a planetary probe, and an internal flow in a contactor used in plasma physics. The results are validated either by comparison with results obtained from other simulations or by comparison with experimental data. High performance on an IBM SP2 system is achieved if problem size and number of parallel processors are adapted accordingly. On 400 nodes, DSMC calculations with more than 100 million particles are possible. 

## 1. INTRODUCTION

The direct simulation Monte Carlo (DSMC) method [1] has become a widely used computational tool for the simulation of gas flows in which molecular effects become important, e.g., in flows with thermal or chemical nonequilibrium. The advantage of using a particle method under these circumstances is that molecular models can be applied directly to the calculation of particle collisions, while continuum methods use macroscopic averages of such effects. Therefore, particle methods can predict these effects with higher accuracy. However, the main drawback of such direct methods is their large numerical cost.

The basic premise of the DSMC method is that particle motion may be decoupled from particle–particle collisions over a time step smaller than the mean time between collisions. The DSMC algorithm consists of the following basic steps: (1) move all particles through a computational grid spanning physical space according to the product of the velocity of each particle and a small time step; (2) compute collisions of particles with a solid surface; (3) determine the cell location of each particle; (4) on a statistical basis, compute collisions between particles occupying the same cell; and (5) gather information on the particles residing in each cell. Although the large number of molecules in a real gas is replaced with a reduced number of model particles, still hundreds of thousands to millions of particles must be simulated, leading to tremendous computer power requirements.

In the past it was most important to develop numerically efficient implementations of the DSMC algorithm, especially for vector supercomputers [2, 3]. However, the associated overhead of building long arrays for these machines restricts the speedup achieved to a factor of about five over unvectorized codes. Perhaps more significantly, while vector machines have reached a plateau in performance, workstations and parallel computers continue to see significant improvements. In addition, the vectorized codes are difficult to understand and to maintain. The DSMC method has become a standard technique for simulation of rarefied flows. Therefore, emphasis is placed on creating an implementation that allows calculation of many different flow problems without changing the code. Workstation processors are a much better platform for these new requirements. They are more accessible and allow interactive control of a simulation calculation. However, to make efficient use of modern workstation architectures, the traditional DSMC algorithm has to be reformulated using new data structures with higher data locality. This is also advantageous for the parallel use of many workstations as in the IBM SP2, whose combined performance is much superior to even today's largest vector supercomputers.

This paper describes the development of a large system named "MONACO" that is based on a new DSMC algorithm formulated specifically for workstation architectures that may be harnessed together in parallel. Strong emphasis is placed on developing a general implementation which can adapt to user needs. First, general concepts associated with implementation of the DSMC algorithm on any computing platform are considered from the standpoint of the



0021-9991/96 $18.00

desire for modular design and high software maintainability. This leads to use of a cell as the basic unit of the algorithm (rather than a particle in previous DSMC implementations) which lends itself naturally to the use of unstructured grids. Next, specific issues related to implementation of the DSMC algorithm on workstation processors are discussed. The approach adopted in the present study for using several scalar processors in parallel is then considered. These discussions lead to the introduction and description of the MONACO system itself. Results are then presented for three different flow cases which demonstrate some of the capabilities of MONACO. Finally, the performance of the new algorithm is compared with previous DSMC implementations. It is found that execution of MONACO on 400 nodes of an IBM SP2 computer allows the use of 100 million particles. Under these conditions the code provides the computing power of 75 single processor Cray C90 vector machines and achieves a parallel efficiency of about 90%.

## 2. IMPLEMENTATION PHILOSOPHY

For implementation of the DSMC technique it is advantageous to strictly separate physical modeling, geometrical calculations and organizational tasks, and to define communication interfaces between these modules. All data related to one module are strictly kept locally and can only be shared among subroutines in the same module. This encapsulated design ensures that source code changes cannot have side effects outside of the module, so that searching for errors is localized. In addition, a whole unit, e.g., for 2D or 3D geometries, can be exchanged easily, while the same modules for physical modeling and organizational tasks are used. Even single subroutines can be replaced simply by linking a new subroutine to the appropriate unit. Changes to the source code itself become rare.

To reach this goal it is further necessary to use an appropriate grid format not only including the numerical description of the geometry, but also the boundary conditions in a general form. It must be possible to specify where inflow, outflow, symmetry, and wall conditions apply and to specify varying inflow and wall conditions. In the past, structured grids were commonly chosen because they are relatively easy to generate and facilitate the vectorization of a program.

On workstation architectures where the principles of vectorization do not apply, unstructured grids may be used efficiently. These have become very popular in continuum flow mechanics solvers, and standard tools may be used for their generation. In addition, they allow a much better adaptation to the flow properties, with higher resolution of gradients in important regions while reducing the cost of calculations in areas of minor importance. Generally, any cell shape may be used; only for practical reasons polygonal cells, or just triangles and/or quadrilaterals are usually employed. However, the use of unstructured grids requires a data structure redesign and a reformulation of the DSMC algorithm which is described in the following section.

While investing tremendous efforts into computational efficiency, the importance of software maintainability is often underestimated. Especially in research projects, where codes are much more likely to be changed, for example, to test different physical models, it is important to take this aspect into account in an early design phase of the implementation. This issue is addressed in the present work through strict separation of geometric and physical aspects of the DSMC algorithm and by designing the code to be as general as possible.

## 3. IMPROVING DSMC PERFORMANCE ON WORKSTATIONS

The architectural differences between traditional vector supercomputers and workstations (see Fig. 1) lead to different requirements for algorithmic implementation. Supercomputers use expensive, fast memory systems organized in independently working banks which deliver data for vector and scalar registers at their computation speed. When data is repeatedly accessed from the same memory bank a delay of about a factor of five to ten is observed. However, these so-called bank conflicts are rare events for most applications. Workstations, in contrast, use inexpensive memory components that store the same amount of data in small physical dimensions. To overcome the associated limitations in bandwidths and access times, a small but fast cache memory is supplied as a buffer between processor and memory. Optimal speed is achieved with operations on data in the cache. As long as access of memory is predictable, the cache is reloaded with sequential sections of memory while the processor is working on other data in the cache. However, whenever data is accessed randomly, so-called cache misses occur. Data elements have to be continuously loaded from the main memory; otherwise less than one-tenth of the nominal processing speed is achieved.

Compared to bank conflicts on supercomputers, cache misses occur much more frequently and can only be avoided if appropriate data structures are used. These aspects were not previously considered in the historical development of the DSMC technique. For example, in the traditional DSMC implementation, a cross reference array is used to assign particles to a grid cell (see Fig. 2a). For the computations, data have to be loaded from a cell data array, which contains the starting point of the particle indices in the cross-reference array, which itself contains the actual location of the particle data in memory. This random access of memory elements through indirect ad-

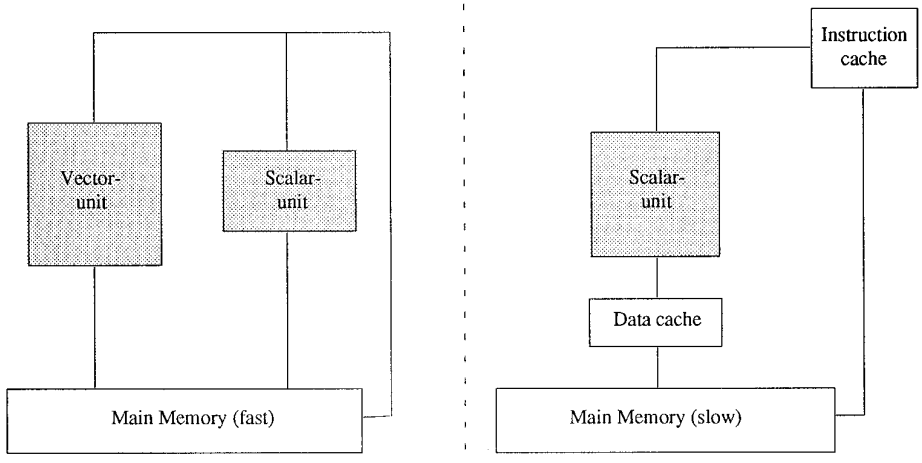


**FIG. 1.** Schematic comparison of vector supercomputer and workstation architectures.

dressing leads to poor performance on workstations. Therefore, on that platform, the data structure shown in Fig. 2b is better suited. All cell data is stored locally including particles residing in it. All computations on one cell are now performed with data available in the cache. During the particle movement phase in the DSMC algorithm, some particles may leave their cell. These are reassigned to the neighboring data structure and therefore implicitly sorted. The explicit calculation of a cross-reference array is not necessary anymore.

The new data structure increases the performance on workstations significantly, cutting execution time by 30–50%, compared to codes which use traditional data structures (see Table I and the associated discussion). This localization strategy is also advantageous for parallelization. Since each cell data structure contains all the information about the cell geometry, as well as the particle data, each cell can be freely assigned to any processor. Therefore, a grid partitioning is designed that accounts for the specific relations of computational to communication speed of different parallel computer systems for optimal load balancing and minimal communication between the computational domains. This was not possible with the adaptation of traditional implementations on parallel computers [4–6].

Nevertheless, some drawbacks are associated with the new data structure. First, it requires the use of a dynamical programming language supporting data structures, like C or Fortran90. Although dynamical memory allocation represents an advantage when using scarce memory resources, the real drawback is that a compiler cannot optimize the coding as well as for a static language like standard Fortran, since certain assumptions cannot be made. In addition, when using the new data structure, the average loop length

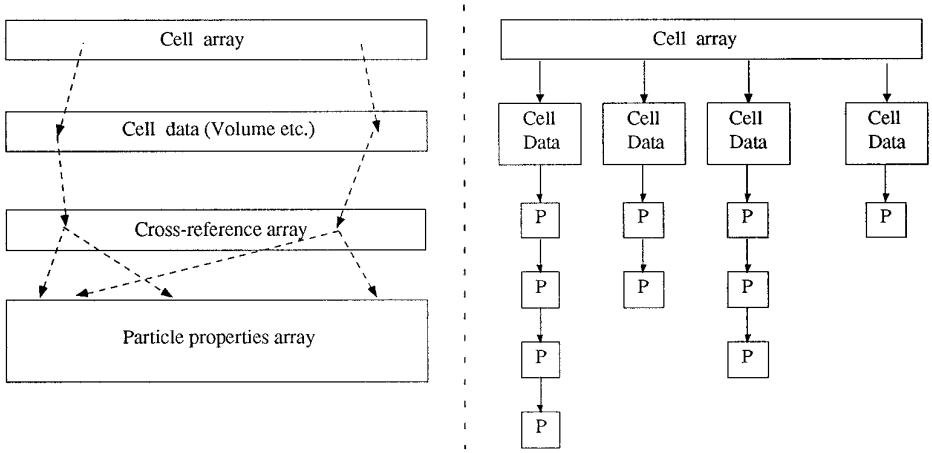


**FIG. 2.** Comparison between the traditional data structure design (left) and an object oriented data design (right) which allows an efficient implementation on workstation architectures.

**TABLE I**

MONACO Performance Compared to a Traditional Code

| System | MONACO | Traditional |
|---|---|---|
| SUN 10/50 | 20 $\mu$s | 28 $\mu$s |
| SGI Indigo II | 12 $\mu$s | 16 $\mu$s |
| RS 6000/990 | 6.5 $\mu$s | 15 (9[a]) $\mu$s |
| Cray C90 | 1.5 $\mu$s | 0.5 $\mu$s |
| SP2—400 nodes | 0.02 $\mu$s | — |

[a] This time is achieved if a special preprocessor is used which adapts the code better to the superscalar architecture.

is reduced. While most parts of the code can still be vectorized, performance degradations compared with traditional data structures are observed on supercomputers with very long vector registers.

Besides optimizing memory access, further performance gains in the DSMC algorithm may be achieved if the internal architecture of the processor itself is taken into account. Supercomputers and workstations internally use many different strategies such as to overlap several independent instructions in parallel. Generally, this task should be achieved by the compiler and optimizer system; however, there are always situations where certain constructs in a code cannot be translated correctly to achieve maximum performance. Bottlenecks occur which dominate the overall performance of the whole code. Hand-turning of parts of the code are nearly always necessary to approach optimum performance of a computer system. Thus, time critical parts of the assembler code produced by the compiler have to be analyzed and compared with the optimal scheduling scheme which would keep as many internal processor units busy as possible. Slight rearrangements of the source code can help the compiler to recognize better instruction scheduling schemes, leading to significantly improved performance. However, very good knowledge of compiler principles and hardware architecture is required. Further details are beyond the scope of this paper, but are described by Dowd [7].

## 4. IMPROVING PERFORMANCE THROUGH PARALLELIZATION

After having established an efficient serial implementation of the DSMC algorithm for a workstation architecture, higher performance can only be reached if several of them are used in parallel. Since today's parallel computers, like IBM's SP2, Cray's T3D, or Intel's Paragon, are also based on workstation technology on each node, the new data structure will be beneficial for them as well. The advantage of using a parallel computer compared to a workstation cluster is that they supply faster communication channels between the nodes.

Using the new data structure, the grid is directly reflected in the data structure (see Fig. 3). Particles are exchanged between cells simply by relinking them to the particle list of the neighbor cell. However, if cells are located on different parallel domains, it is not feasible to send each particle to the appropriate task. Here, "task" is the computer program running on a particular node or processor. The communication overhead due to the latency involved in each communication process would be too high. Therefore, each task computes the movement of all its particles and links those who are leaving the domain, together with a remaining moving time, to a list of particles to be transferred (see Fig. 4a). After this has been accomplished, each task holds a vector with the numbers of particles to be moved to each other task. From a global point of view, these vectors build a matrix which is stored rowwise on each task (see Fig. 4b). Now the total sum of all particles to be moved is calculated, first for all particles on each node and then globally.[1] If this global sum is zero, no more particles have to be communicated and each task starts over with a new DSMC cycle. Otherwise, the matrix is transposed,[2] which means that now all vectors on each task will contain the number of particles to be received from each other task. The particle data in each linked list is copied into a continuous communication buffer and an asynchronous send is submitted to transfer the particle data in one block for performance reasons. The MPL command employed in the present implementation is optimized to perform efficiently in parallel. After each task has asynchronously started all send operations, it waits for the particle data to be received according to the matrix vector. The order in which task data is received is irrelevant. Once a data block is received, the particles are moved while other particle data might still arrive. Some particles might leave the domain. They are stored again in the linked list of particles to be transferred and after all particle data is received, the loop starts over again.

This kind of parallelization has the advantage, that no explicit synchronization is required. Therefore, computations of the particle movement can be overlapped to a high degree with the communication of particle data if permitted by the hardware. In addition, besides the global communication while transposing the matrix, no other communication between tasks is necessary unless particle data needs to be exchanged between tasks. In this way, the overall amount of data for communication is kept low.

## 5. THE MONACO SYSTEM

The MONACO system is designed according to the principles described above. It focuses on the use of workstation

[1] Using MPL, the complete operation is performed with "mpc_combine."

[2] Using MPL, this all-to-all communication is advantageously performed with "mpc_index."

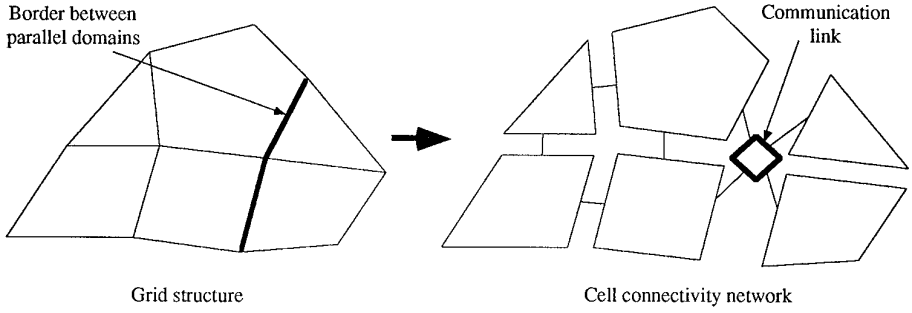


**FIG. 3.** The grid geometry is directly reflected in the data structure. A communication link is introduced between cells on different parallel domains.

processors, perhaps linked in a parallel configuration, as the primary calculation platform. Nevertheless, the most important objective is to create a generic, easily maintainable DSMC code that can be extended by new modules to meet specific requirements. The new data structure is used to achieve high efficiency on serial workstations or parallel workstation clusters. Physical modeling, geometry calculations and organizational tasks are strictly separated (see Fig. 5). A kernel library handles the organizational tasks associated with a DSMC simulation, such as the correct assignment of particles to their cell data structure; the geometric library handles such tasks as the movement of particles through the grid; and the physics library selects collision pairs and applies appropriate physical models to them. Each of these three units uses completely local data, which is initialized from externally supplied input files. The libraries are now described in more detail.

### 5.1. *The MONACO Kernel Library*

This set of subroutines provides organizational tasks necessary for a DSMC simulation. They are implemented in the programming language C, allowing dynamic memory allocation. This can reduce memory requirements significantly, compared to a static implementation in Fortran. Another advantage of using C is the possibility to include computer architecture dependent program statements, so that the same source code can be used on different serial, as well as on parallel, computers, simply by recompiling.

The main task of the kernel is to build the data structure described above and then to communicate with the subroutines in the geometry and physical modeling libraries. For example, for cells with inflow boundaries, calls to the routines for generation of new particles in the physics module are made. The new particles are stored together with particles already present in the cell close together in memory. Subroutines for the selection of collision pairs and the application of appropriate collision mechanics between them are called. If a steady state has been reached, routines for the sampling of macroscopic values are executed. For the movement of particles, the kernel software calls the move routine for each cell. Most particles will stay within the same cell, but some may cross the cell boundaries. These are returned to the kernel software and assigned to their new cell data structures with a remaining time step. Although many calls to the move routine are necessary, the associated overhead is kept small through extensive hand tuning of this portion of the code, and therefore an insignificant effect on the overall performance is observed.

On a parallel computer, the kernel software distributes the cells automatically or as prescribed by a decomposition file. Each program on each node performs all the necessary calculations for its cells independently. Only during the

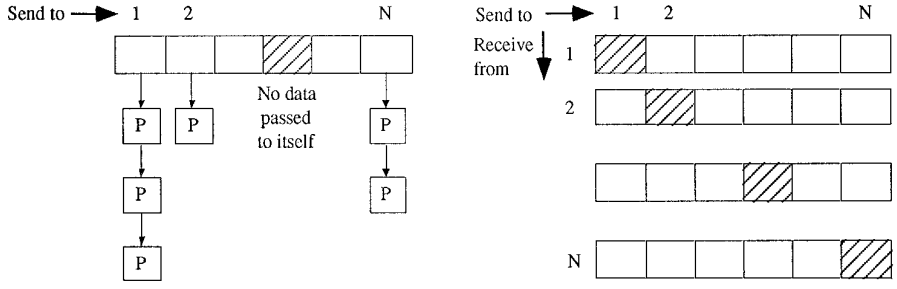


**FIG. 4.** Particles which have to be sent to another task are first stored in linked lists on each task. From a global view, the list of particles form a matrix with numbers of particles to be sent.

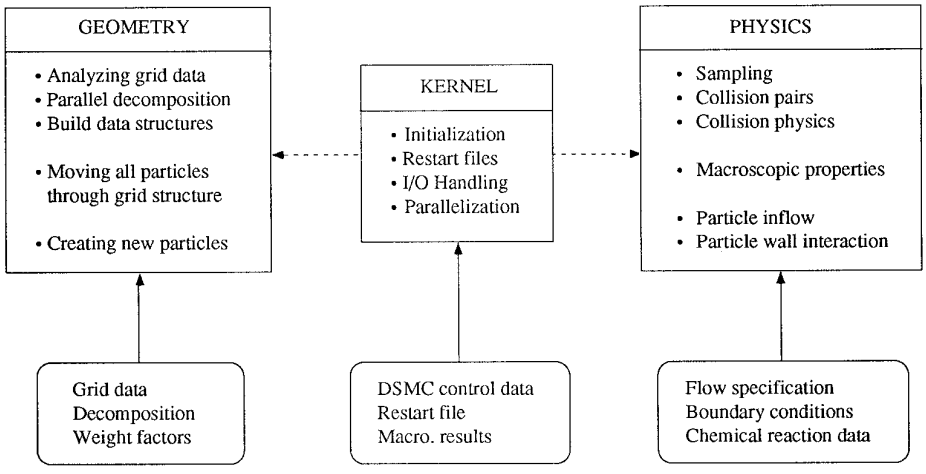


**FIG. 5.** The MONACO software system consists of three parts: a kernel for the management of the simulation, a geometry module, and a physics module.

movement phase, for particles that are assigned to a cell structure on a different processor, communication between the processors becomes necessary. Because of the latency involved in each exchange of data between processors, it is not feasible to send particles one by one. Instead, they are temporarily stored in communication buffers and sent asynchronously at once after all the movement calculations are completed. Then the movement computation is continued until all the particles have been moved by a full time step and no particles are left in the communication buffers. This has the side effect that no explicit synchronization of the tasks is necessary and therefore good parallel overlapping of computations is achieved.

The kernel software not only supplies continuous output about the status of the simulation and temporary results, but it is able to detect if user-supplied input files have been altered and to adjust the simulation accordingly. This feature gives a quasi-interactive control if the simulation is running in batch mode and some discrepancy to the presumed simulation behavior is observed. Then the parameters may be adjusted during a computation without having to cancel and to interrupt, restart, and requeue the simulation.

### 5.2. *The MONACO Geometry Library*

The geometry library supplies tasks related to the geometry itself such as finding the neighbors of each cell in a grid. Since these routines have to handle complex data structures, they are also implemented in C. Originally, different versions of this library for structured and unstructured grids were under consideration, but the performance on workstations using unstructured grids and localized data structures is about the same. Since unstructured grids allow a much higher flexibility and adaptability to flow problems, the development of a geometry library for structured grids was dropped.

The most important subroutine processes the movement of particles through a given grid structure. The move routine is called by the kernel module and passes a list of particles to be moved. To move the particles and to determine which particles leave the cell and have to be reassigned to other cells, a particle tracing method is used [8]. Tracing particles through an unstructured grid with high efficiency and accuracy is not a trivial task. The method used in the MONACO geometry module is described for 2D and axi-symmetric flows in the Appendix of this paper. For the future development of a 3D geometry library, a method described by Dietrich [2] may be used.

Within an axi-symmetric simulation it is very useful to introduce spatially varying weights for particles. This means that real molecules in some regions of the flow are represented by more model particles for better statistical calculation of the macroscopic properties, especially along the axis. Particles which cross regions with different weights may have to be destroyed or cloned accordingly. Although a weight may be associated with every cell, it is found advantageous to use different weight domains. In the parallel version, these domains can also be associated with the domains used for the decomposition of the problem. Then the required calculations can be overlapped with the communication overhead and the communication is reduced since particles are destroyed if necessary before sending and are cloned if necessary only while receiving. Although this association of weights with parallel domains limits somewhat their free selection, it has proven to be very efficient.

The data format used by the geometry library for the grid employs a standard from the ''National Grid Project (NGP)'' [9]. In this project, tools are developed for use in

engineering field simulations, especially for grid generation and adaptation to flow gradients and for data postprocessing. These tools can easily be used for DSMC calculations, avoiding the development of special grid generation algorithms. The advantage of the NGP format is that boundary conditions are part of the grid definition, making it possible to perform flow simulations of very different problems without any changes in the code.

### 5.3. *The MONACO Physical Modeling Library*

The routines in this library specify the physical aspects of the DSMC method itself. They are written in Fortran and implemented in serial. This allows straightforward inclusion of existing physical modeling implementations. Since each routine has a well-defined interface to the kernel module, a routine can easily be exchanged with another to test different models without requiring changes to the source code of the MONACO system itself. This greatly enhances the development and application of new physical models.

Currently, a set of state of the art subroutines which comply with the MONACO interface standard is available. The calculation of physical interactions between the molecules may be computed using the popular no-time-counter method introduced by Bird [10]. However, this scheme uses an updating process to establish the correct collision rate. This is associated with a poor performance if unusual values are generated in the transient stage of the simulation that may lead to a very inefficient selection calculation. For the MONACO system the collision pair selection mechanism by Baganoff and McDonald [11] using the VHS model [12] is implemented. This method gives the correct collision rate instantaneously. A new scalar optimized algorithm for the random selection of particles is developed to replace the half-deterministic schemes used by McDonald [6]. This algorithm shuffles an array which is initialized with sorted elements so that two successive indices give a random collision pair using each particle only once and using only one random number per particle pair:

```
do = i+1, nparticles
     shuffle(i) = i
enddo

do   i=1, nparticles2
     isave = shuffle(i)
     j = i + nranf(nparticles - i)
     shuffle(i) = shuffle(j)
     shuffle(j) = isave
enddo
```

The function nranf($n$) in this example returns a random integer between 1 and $n$. The variable ''nparticles2'' is the updated number of particles used to build collision pairs. Since it is not efficient to pair all particles in each cell if only a few collisions occur, this number has to be adjusted from time to time. However, if more pairs have to be built than there are particles available the simulation time step is too large. This provides instant feedback to the user about the correctness of the numerical parameters. In this case, the time step in the simulation should be reduced. MONACO is designed so that this may be performed as the code is executing.

Once this algorithm for an efficient selection of collision pairs is established, further physical modeling can be added. Currently, mechanisms are implemented using schemes developed by Borgnakke and Larsen [13] and by Bergemann and Boyd [14] for internal energy exchange, and by Dietrich [15] for energy exchange in chemical reactions.

### 5.4. *The MONACO Utility Library*

The utility library consists of several independent programs to provide additional functionality. Several filters are supplied to convert conventional structured or block-structured grids into the unstructured NGP-format using triangular or quadrilateral elements and to convert different formats for unstructured grids into the NGP-format.

When using a parallel computer, a utility program generates a suitable domain decomposition from a given grid and a primary solution to redistribute the cells over the processors to reach a better load balancing and to minimize communication between the processors. Strictly speaking, a minimum spanning tree algorithm known from operations research problems would be an optimal choice. However, limitations, such as the use of different weighting factors in different parallel domains and the requirement that the same number of particles be established in all domains, makes this algorithm very complex. Therefore, a simple, but functional approach was used [16]. Only strips of cells are chosen which contain the same number of particles. These horizontal regions typically consist of portions of the domain containing flow from the left to the right. Although these regions do not always follow the local flow velocity, as shown in the examples below, the method works sufficiently well without creating a communication bottleneck.

## 6. EXAMPLES OF FLOW SIMULATIONS WITH MONACO

Three different flow problems are chosen to demonstrate the generality and performance of the MONACO system. The first example, a two-dimensional flow through a diverging channel, illustrates the ability of the code to handle flows through an adapted triangulated grid. In the second example, a hypersonic flow around a planetary probe, the

problematic wake area is simulated with high accuracy. Finally, the neutral flow through a plasma contactor shows the ability to handle a variety of variable boundary conditions and a mixed triangular and structured grid. All these simulations are performed with the same program without recompiling the code. Once a suitable grid is generated, the average time to set up a simulation of one of these flow problems is just a few hours.

### 6.1. *Simulation of a Diverging Channel Flow*

Flows through nozzles in spacecraft thrusters continue to be an important application for the DSMC method [17]. They are difficult to calculate since they require the simulation of the whole transition from a very dense gas at the nozzle entry to a rarefied gas at the nozzle exit. Special attention has to be paid to the simulation of the nozzle entry. Here, a shock forms at the inflow edge and its location may influence the flow field in the downstream region. As a simple test case, a two-dimensional channel geometry is constructed which allows investigation of the typical phenomena without having to calculate a complete nozzle configuration.

The flow is simulated with a traditional DSMC code [3] based on structured grids and the new MONACO system. The resolution of the structured grid is 250 cells along the center line and 50 cells perpendicular to it. For the simulation with MONACO, diagonals are introduced randomly into the cells of the structured grid to form a triangular grid. The gas flow is molecular nitrogen with conditions at the inlet: $n_\infty = 10^{23}$ m$^{-3}$, $T_\infty = 300$ K, and $v_\infty = 1500$ m/s. The collision model is the variable hard sphere of Bird [12] and the parameters for nitrogen reported in Ref. [12] are used here. Constant collision numbers of 5 and 50 are employed for relaxation of the rotational and vibrational energy modes. Interaction of particles with the wall assumes full energy accommodation to a temperature of 300 K.

The traditional code reached a steady state containing about 220,000 particles after 5,000 time steps of $\Delta t_m = 1.94 \times 10^{-9}$ s, and another 10,000 steps are added to sample macroscopic values. With MONACO, about 5,000 time steps are needed to reach a steady state. While the simulation is running in batch mode on eight SP2 processors, $\Delta t_m$ is interactively decreased from $1 \times 10^{-8}$ s to $1 \times 10^{-9}$ s. Then the grid is adapted to the local mean free paths (see Fig. 6) and the total number of particles in the flow field is increased to 480,000. In Fig. 6, the length scales $x_b = y_b = 0.005$ m. After 1,000 additional time steps the simulation again reached a steady state and 20,000 sampling steps are added to obtain smooth macroscopic results.

Figure 7 shows the calculated density field. The shock forms on the inflow edge of the channel and progresses towards the center. Due to the expansion of the channel,

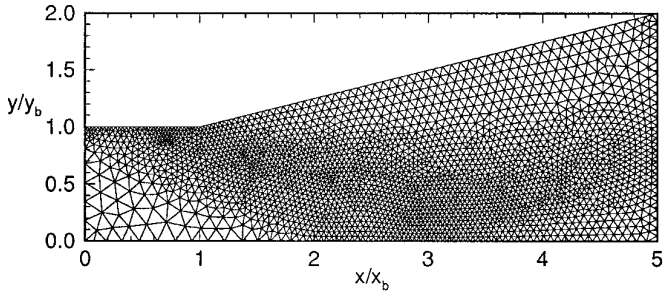


**FIG. 6.** Adapted grid generated from an initial grid for the channel flow.

it is not reflected and disperses. At the beginning of the slope the flow separates. The solutions of both codes show a good overall agreement in the simulation of the overall physical effects. However, the advantage of using an adapted grid and a large number of particles can be seen in the results along the center line and along the channel exit. The center line results (see Fig. 8), show that the strong jump in both temperature and density occurs earlier in the calculation on the structured grid. This is the result of the insufficient resolution of the structured grid in the shock area. The rotational temperature is hardly affected because it increases later and more slowly than the translational temperature. The comparison of the temperature and density results along the exit of the channel in Fig. 9 shows nearly perfect agreement. However, due to a lack of particles in the simulation with the structured grid, high statistical scatter is observed at the upper left exit area.

### 6.2. *Simulation of a Reentry Planetary Probe*

The heat and pressure distribution of hypersonic flight vehicles during reentry is characterized by strong thermo-

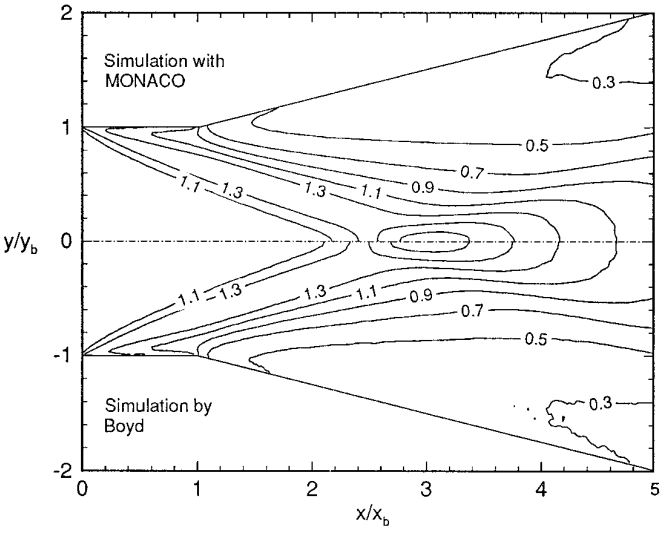


**FIG. 7.** Comparison of the results for the density field of the channel flow simulations.

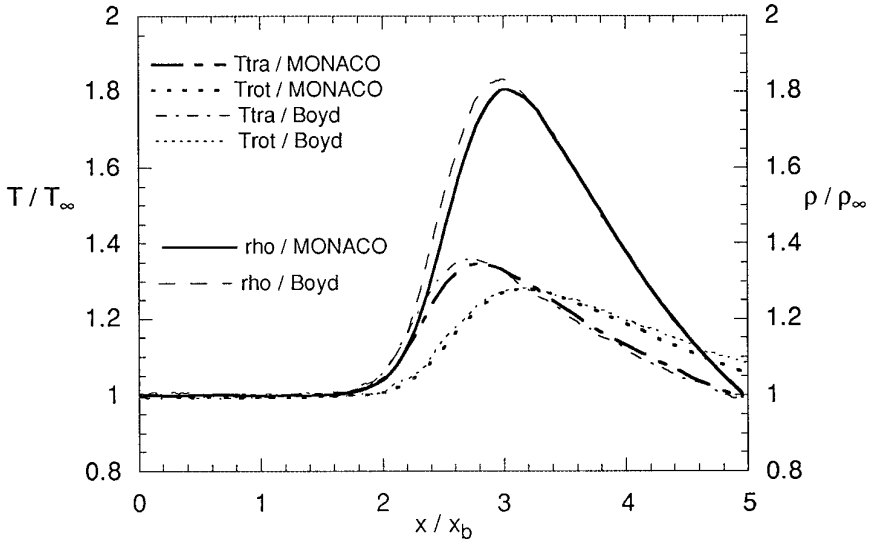


**FIG. 8.** Comparison of temperature and density along the center line of the channel flow.

chemical nonequilibrium effects. These can be simulated using the DSMC method, and a variety of appropriate models have been developed. However, these models have not been validated thoroughly due to a lack of experimental data. To address this issue experiments were conducted at the Large Enthalpy National Shock Tunnel (LENS), CALSPAN-University of Buffalo Research Center, using a planetary probe configuration [18]. This case is now simulated with MONACO to show its ability to handle such flow problems. The simulated flow conditions are: $n_{\infty,N2} = 2.808 \times 10^{21}$ m$^{-3}$, $n_{\infty,N} = 1.124 \times 10^{18}$ m$^{-3}$, $T_\infty = 295$ K, and $\nu_\infty = 3245.8$ m/s. The VHS model is again used with rotational and vibrational collision numbers of 5 and 1000, respectively. Full energy accommodation to a wall temperature of 295 K is assumed.

An unstructured grid with 15,000 nodes and 12,900 quadrilateral elements is generated around the probe body, whose geometry is described in [18]. The simulation is started on 20 SP2 processors, using a relatively large time step of $\Delta t_m = 1 \times 10^{-8}$ s. As soon as the whole flow field is filled with particles and a shock forms, the time step is decreased interactively to $\Delta t_m = 1.95 \times 10^{-9}$ s. The simula-

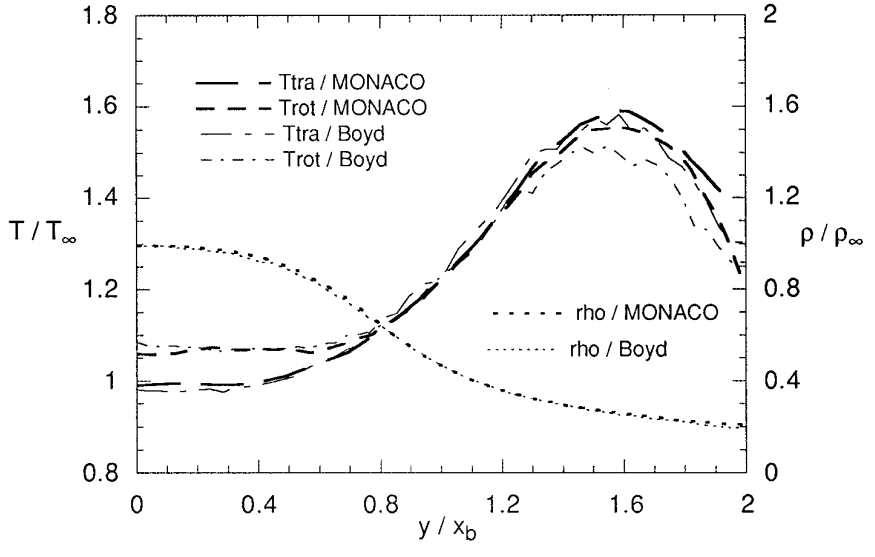


**FIG. 9.** Comparison of temperature at the channel exit.

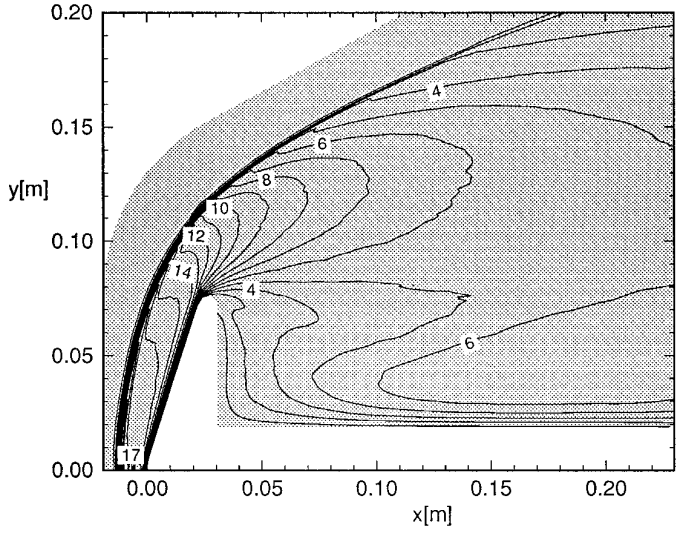


**FIG. 10.** Temperature contours for the planetary probe simulation computed with MONACO.

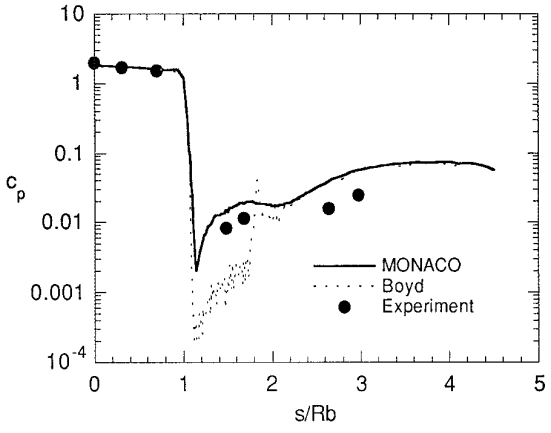


**FIG. 12.** Pressure coefficient along the surface of the planetary probe.

tion becomes stationary with about two million particles after about 30,000 steps, and 50,000 sampling time steps are added using 50 processors.

Contours of calculated translational temperature are shown in Fig. 10. A strong shock forms in front of the body, and a rapid expansion causes a wake structure in the back of the probe (see Fig. 11). Profiles of surface pressure and heat transfer along the body surface are shown in Fig. 12 and Fig. 13. They are compared with results obtained independently by Boyd and the experimental data [18]. A good overall agreement is achieved between the DSMC methods, although different models are used for the internal energy exchange and chemical reactions. Due to memory limitations in the simulation of Boyd, the number of particles in the wake area was too small and, therefore, some discrepancies are observed. However, considering the complexity of this case, both simulations show satisfactory agreement with the experimental data. This demonstrates the ability of the DSMC method to predict this type of flow problem.

### 6.3. *Simulation of a Neutral Contactor*

Electric propulsion systems are being developed for the control of satellites. These plasma physics applications open a new field for the DSMC method with two new principal problems: steep density gradients occur between the ion source and the surrounding flow field, and the electromagnetic field influences the particle movement. In the following, a neutral flow in a plasma contactor is simu-

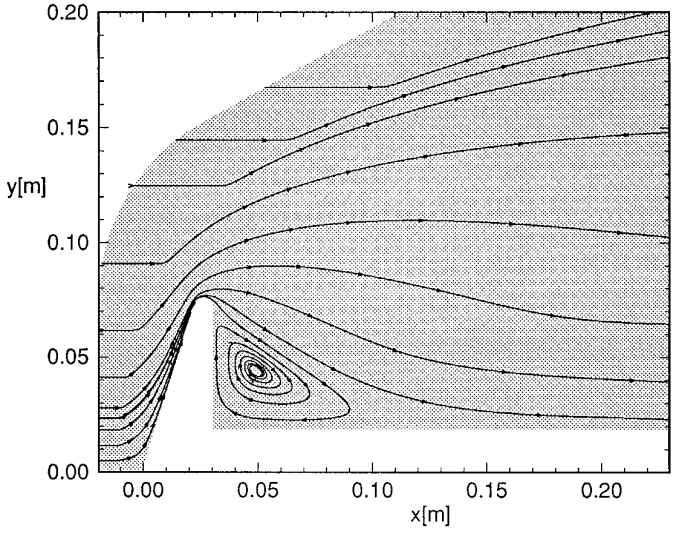


**FIG. 11.** Streamlines in the wake area of the planetary probe simulation computed with MONACO.

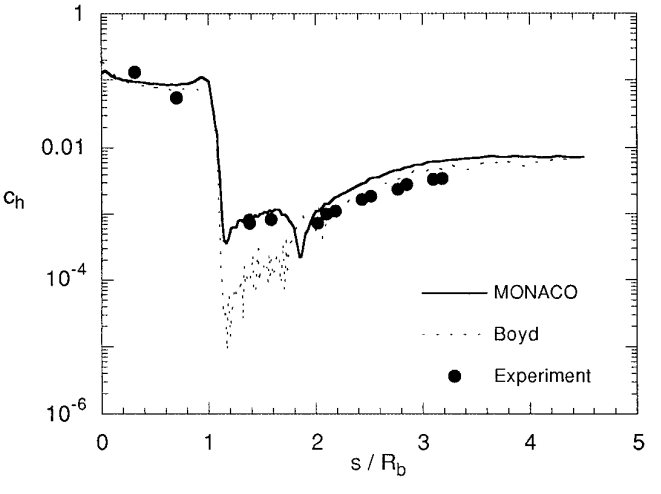


**FIG. 13.** Heat transfer coefficient along the surface of the planetary probe.

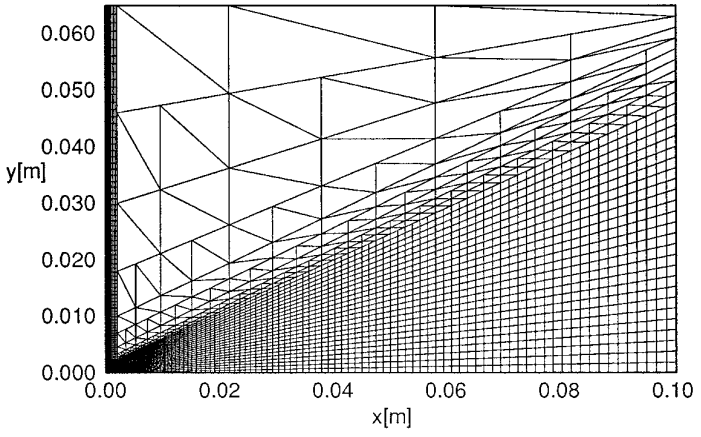


**FIG. 14.** Mixed structured/unstructured grid used for the plasma contactor simulation.

lated to show the ability of the new code to handle steep gradients with grids consisting of mixed cell types. As illustrated in Fig. 14, the mesh was generated using mainly quadrilateral cells. This is appropriate for the main flow of particles from the anode in the lower left area. However, behind the vertical anode wall an adaptation of the grid to the extremely rarefied conditions is necessary. Triangular elements were used in this area. The simulated flow conditions for the xenon gas employ a flow rate from the cathode of $4 \times 10^{-7}$ kg/s into a chamber pressure of $4 \times 10^{-4}$ Pa. Full energy accommodation is assumed to wall temperatures of 300 K and 1300 K for the anode and cathode, respectively. Note that variable inflow conditions have been applied at the cathode orifice which are a result of a separate DSMC computation of the flow within the anode.

The simulation is started, neglecting the main flow, to establish a stationary background gas. Then the main flow is introduced. On 20 processors 120,000 simulation time steps with $\Delta t_m = 9 \times 10^{-9}$ s are performed to establish a steady state with 200,000 particles. This flow expands quickly into the background gas and forms a stationary field whose density contours are shown in Fig. 15. The comparison of the density along the center line with the experimental data [19] in Fig. 16 shows very good agreement. In the near future, new modules will be developed for the MONACO system which incorporate movement of charged particles through electromagnetic fields and appropriate collision and wall models.

## 7. PERFORMANCE CONSIDERATIONS

To assess the benefits from the new data structure design in MONACO, timing studies are made and compared to a code by Boyd [3] based on the traditional data structure design. While the MONACO code uses unstructured grids and can run in parallel, the traditional code uses structured grids and is vectorizable. Performance results for the planetary probe flow are given in Table I. The units used are absolute time spent per particle and time step, since measurements in megaflops are mainly implementation dependent. As expected, the MONACO code always performs on workstation architectures better than the traditional design, whereas the performance of the traditional code is much better on a vector supercomputer. While the new code takes advantage of better cache management and some superscalar optimization on the RS6000 architecture, which was the development platform for the MONACO software, the traditional code employs large vector lengths and achieves a higher degree of vectorization. However, a gather/scatter hardware that allows fast indirect addressing is mandatory for high performance.

Besides running faster in serial mode, MONACO can also distribute the cells over several workstations running

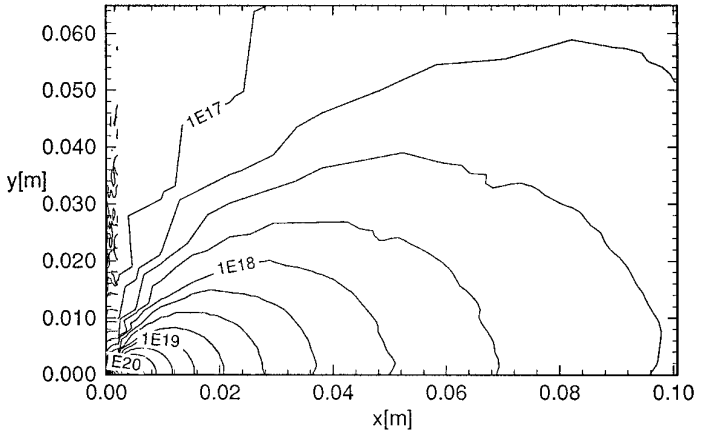


**FIG. 15.** Density field of the plasma contactor flow.

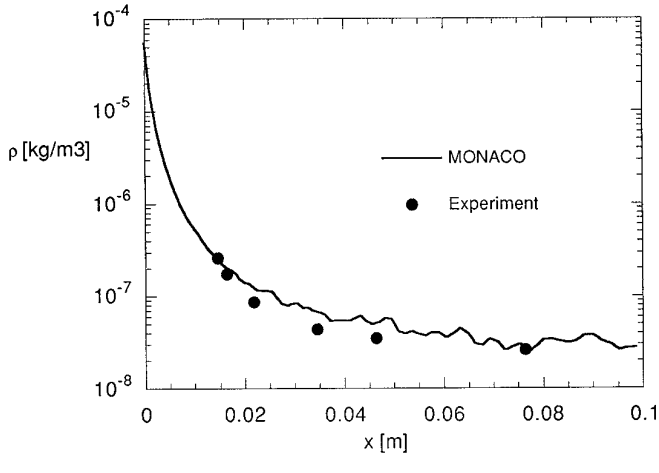


**FIG. 16.** Density along the contactor axis and comparison with experimental data.

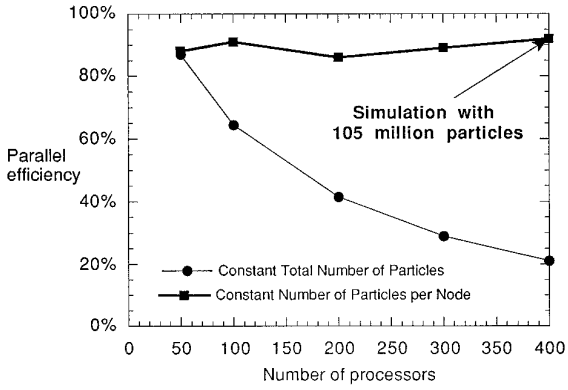


**FIG. 17.** Parallel efficiency of MONACO on the IBM SP2.

in parallel. As a test case, a planetary probe simulation with a density 10 times higher, compared to the flow described above, is used. The simulation is started on 50 processors with a default assignment of the cells to the processors. While particles are entering the flow field, their distribution does not agree with the default decomposition, and some processors start to hold more particles than others. The load balance is disturbed and the parallel efficiency drops as shown in Fig. 17. Here, parallel efficiency is defined as the ratio of computation time to the sum of computation plus communication time. Cell redistribution is necessary after 3,000 and 10,000 time steps to keep parallel efficiency higher than an acceptable value of 60%. However, as the simulation becomes more stationary, less distribution is necessary, and the parallel efficiency is able to reach a value of about 90%.

Using more processors does not necessarily accelerate the simulation adequately. If calculation of the test case is continued on more than 50 processors, the parallel efficiency drops significantly. On 400 processors, only 21% parallel efficiency is achieved, as shown in Fig. 17. In this case only about 30,000 particles are used on each node, leading to a large discrepancy between calculation time and communication setup time. During this communication setup phase, each processor has to communicate to all the other processors how much particle data is to be exchanged between the processors. On many processors this all-to-all communication is expensive on the SP2 system because of its specific communication hardware design. The actual exchange of particle data is, however, of less importance for the overall communication time. To reduce this discrepancy, a large number of particles must be employed on each processor. Figure 18 shows the average time spent per particle and the time step on a SP2 node for varying numbers of particles per processor. Two efforts overlap: first, with an increasing number of particles on each node, the relation between calculation time and communication setup time improves; second, the cache and the different scalar units on each processor are more efficiently used. To assess the influence of this effect during a simulation, a rough estimate is computed by the code, based on simple calculations of the possible number of instructions, compared to instructions actually performed on the processor. This is indicated by the dashed line in Fig. 18.

With about 250,000 particles per node and newly distributed cells, according to the current state of the simulation, high parallel efficiency may be obtained on any number of processors. On an IBM SP2, simulations were performed with up to 400 nodes (see Fig. 17), employing more than 100 million particles. This is, so far, the largest DSMC simulation reported in the literature. Compared to the performance on a supercomputer, e.g., a Cray C90, the MONACO code performs in parallel on five SP2 nodes as fast as in serial on one Cray processor. The performance of the specifically vectorized traditional code is exceeded by MONACO on 16 SP2 processors, and a run on 400 nodes is equivalent to a calculation capability about 75 times faster than running MONACO on a single Cray C90 processor. However, as long as a simulation has not reached a steady state, the cells have to be reassigned quite often, since a small deviation of the number of particles on one node from the average may lead to a significant drop in parallel efficiency. Reassigning cells is, however, expensive. Also, these results show that it is practically not possible to accelerate a specific DSMC simulation simply by using MONACO with more processors on an SP2 system, or to make small simulations on a large number of nodes. Instead, only the total problem size can be scaled with the number of available processors, and the total time to finish a simulation is about the same.

## 8. CONCLUSIONS

The MONACO system was developed with strong emphasis on using workstations as primary calculation plat-

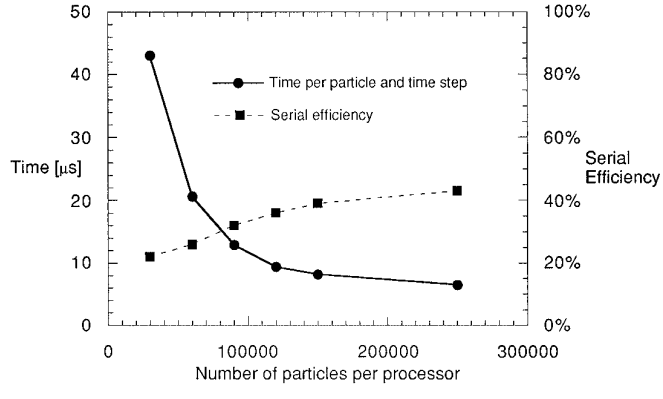


**FIG. 18.** Time spent per particle and time step on an SP2 node when running in parallel.

forms. These machines are readily accessible and, if codes are adapted accordingly, have performance capabilities formerly only achievable by supercomputers. A major design feature of MONACO was the separation of physical modeling, geometrical issues, and managerial tasks. This makes it possible to exchange or to enhance physical models simply by linking new modules to MONACO without having to change the code itself. In addition, this significantly improved the maintainability of the code. High performance on workstation architectures was achieved in MONACO using a new object-oriented data structure design. The design allows a better localization of the data in memory, thus improving cache management. This cannot be achieved when a code with the traditional DSMC data structure is used on workstations. The performance advantage of the traditional design on vector supercomputers is easily overcome due to the powerful parallelization capabilities of the new design, driving performances beyond the capabilities of current vector supercomputers.

Using a grid structure with incorporated boundary conditions also makes it possible to calculate a variety of flow problems without having to change the code. Three different flow problems were chosen for this paper and calculated with the same executable code. A flow through a channel was simulated to show the advantages of using adapted triangulated grids for a better resolution of the shock and its location. The simulation of a flow under thermochemical nonequilibrium around a planetary probe configuration showed the ability of the new implementation to handle large problems efficiently. Good agreement with other DSMC results and experiments was reached. Large gradients in density, as they appear in the third simulation of a neutral contactor flow can be mastered by combining structured and unstructured grids. In addition, the new implementation proved to be able to handle a variety of boundary conditions in this problem. Good agreement with experimental data was reached.

Performance studies were made to show the excellent performance of the MONACO code in serial as well as in parallel mode. Since the new data structure allows any grid cell to be assigned to any processor, communication can be kept to a minimum. On 16 SP2 processors the code was faster than DSMC codes specifically optimized for vector supercomputers. However, the research has shown that the benefits of using a large parallel computer will only make larger simulations possible. Small DSMC simulations with only a few hundred-thousand particles will not be significantly accelerated using many processors since the parallelization overhead is too large, compared to the computation time on each node. It is more economical to run these only on a few workstations, using the new data structure, or in serial on a supercomputer using the traditional data structure. However, really large DSMC simulations with millions of particles will greatly benefit from using a parallel computer. The research showed that it is already possible today to make simulations exceeding 100 million particles on an IBM SP2 computer with 400 nodes while achieving a parallel efficiency of 90%.

## APPENDIX: MOVING PARTICLES IN AXISYMMETRIC FLOWS

At first glance, particle tracking may seem a trivial task. However, a fast method is needed to determine if, during particle movement, walls or other boundaries are encountered, and in which grid cell the particle resides after being moved. Otherwise, this part of the DSMC algorithm becomes a severe bottleneck for the simulation. Since simple rectangular grids and body shapes are insufficient for modern simulations, methods must be developed for use in grids generated numerically.

Particle tracing has been shown to be an efficient method for 2D and 3D simulations [2]. In this method, for each particle the time is calculated in which it hits the next edge of its current cell (the time-to-hit). If the time-to-hit is larger than the simulation time step, the particle will not leave the cell, and the new position is calculated very quickly. This is typically true for more than 50% of all particles. If a particle enters the neighboring cell, the time calculation has to be repeated. This does not mean that the intersection point is actually calculated. In fact, generally, the intersection point should not be calculated since this may cause a particle to become trapped in the ''numerical thickness'' of the cell boundary. For example, a particle may be passed to a neighboring cell because the time-to-hit is smaller than the simulation time step. Then, after being repositioned by the time-to-hit, the coordinates of the particle show that it still resides in its old cell. This is avoided by using the original particle location as a reference and moving the particle to the final location only after the new cell is found.

Calculating the time until a particle hits the next cell edge can be a very time consuming process. To prevent this, in general, it is sufficient just to move each particle by the complete time step and perform a simple check to see if it is still in the same cell. For the relatively small subset of particles that are no longer found in the same cell, it must be determined through which edge they exited the cell. Before actually calculating the time until a particle hits each cell side, several checks are performed to see if they are able to hit a cell side. Only for those potential candidates is the actual time calculated.

While these algorithms are easy to implement efficiently in 2D and 3D calculations, some problems with low efficiency and numerical accuracy occur in axisymmetric simulations. In this case, the calculation of time-to-hit requires solution of a quadratic equation. Efficient solution of this

equation requires careful instruction scheduling. Our approach is outlined in the following.

As a first step, each particle in a cell is moved and the coordinates of the new position are determined as

$$\begin{aligned} x_p' &= x_p + u\Delta t \\ y_p' &= \sqrt{(y_p + v\Delta t)^2 + (w\Delta t)^2}. \end{aligned} \tag{1}$$

Note that the particle velocity vector need not be transformed at this point. Now for every side of the cell with start and end point indexed 0 and 1, respectively, the following expression is calculated which is simply a vector product of the particle position vector with the normal vector of a cell side:

$$\begin{aligned} \mu &= x_t(y_p' - y_0) - y_t(x_p' - x_0) \\ x_t &= x_1 - x_0; y_t = y_1 - y_0. \end{aligned} \tag{2}$$

If $\mu$ is positive for each cell side, then the particle stays in the same cell. If any $\mu$ is negative, the particle is located beyond that cell side. However, it cannot be concluded that it has actually crossed that side. It may have hit another side first. In this case it has to be determined which cell side the particle will hit first on its path by solving the following equations for $\lambda$ and $\delta t$:

$$\begin{aligned} x_p + u\delta t &= x_0 + \lambda x_t \\ \sqrt{(y_p + v\delta t)^2 + (w\delta t)^2} &= y_0 + \lambda y_t, \end{aligned} \tag{3}$$

where $\delta t$ represents the time until the particle hits and $0 < \lambda < 1$ represents a parameter locating where it hits the cell side. Accuracy considerations make it necessary to calculate $\lambda$ first. Equations (3) may be combined to give the following quadratic equation:

$$\begin{aligned} a\lambda^2 + b\lambda + c &= 0 \\ a = r_3^2 + r_4^2 - r_5^2;&\quad b = 2(r_3r_6 + r_4r_7 - r_1r_5); \\ c = r_6^2 + r_7^2 - r_1^2.& \end{aligned} \tag{4}$$

The following abbreviations were specifically hand-tuned to achieve a high degree of scalar optimization through instruction scheduling in the computational implementation:

$$\begin{aligned} &r_0 = uy_p; \quad r_1 = uy_0; \quad r_2 = x_0 - x_p; \\ &r_3 = vx_t; \quad r_4 = wx_t; \quad r_5 = uy_t; \\ &r_6 = r_0 + vr_2; \quad r_7 = wr_2. \end{aligned} \tag{5}$$

So far, only simple addition and multiplication operations were used. Further calculation of solutions of Equation (4) usually requires the use of expensive division and square root operations. However, in most cases Eq. (4) will have no solution with $0 < \lambda < 1$. Therefore, all cases which cannot lead to valid solutions are discarded. For a $c < 0$, only one solution with $\lambda > 0$ exists, and $\lambda < 1$, if

$$\begin{aligned} &ad < a^2, \\ &d = 0.5(-b + \text{sign}(a)\sqrt{b^2 - 4ac}). \end{aligned} \tag{6}$$

In the case a $b > 0$ or $b^2 - 4ac < 0$ no solutions exist for $\lambda$; otherwise, two solutions are possible, which in a simulation is a rare case occurring mostly for particles in cells close to the axis. In this case the validity of the solutions is checked without using expensive division calculations through

$$\begin{aligned} 0 < \lambda_1 < 1 &\leftrightarrow ad < a^2 \wedge ad > 0 \\ 0 < \lambda_2 < 1 &\leftrightarrow cd < d^2 \wedge cd > 0 \end{aligned} \tag{7}$$

in which $d$ is calculated using

$$\begin{aligned} b > 0&: d = -0.5(b + \sqrt{b^2 - 4ac}) \\ b < 0&: d = -0.5(b - \sqrt{b^2 - 4ac}) \end{aligned} \tag{8}$$

for high accuracy. Only for valid $\lambda$ results, the appropriate time is calculated with

$$\begin{aligned} \delta t_1 &= \frac{1}{u}\left(r_2 + \frac{d}{a}x_t\right), \\ \delta t_2 &= \frac{1}{u}\left(r_2 + \frac{c}{d}x_t\right) \end{aligned} \tag{9}$$

and if both are valid, the minimum is taken. However, the actual calculation of a time until the particle hits a side is rarely necessary since in most cases a valid $\lambda$ cannot be found.

## ACKNOWLEDGMENTS

Funds for this research were provided by DFG (German Science Foundation) and Cornell Theory Center who also supplied use of their facilities and time on the SP1 and SP2 systems. The development of such a large and complex computer code would not have been possible without the help of many people. Special thanks to Adolfy Hoisie, Cornell Theory Center, for helpful discussions on scalar optimization and parallelization. The authors are also grateful to students in the research group of Professor Boyd at Cornell University (Keith Kannenberg, Xiaoming Liu, Daniel Karipides, and Anand Srinivasan) for their suggestions and continuous testing to improve the software. The plasma contactor simulations were performed by Xiaoming Liu. And, finally, thanks also to Frank Bergemann, Rolf D. Boettcher, Thomas Sonar, and Oliver Friedrich from DLR (German Aerospace Establishment).

## REFERENCES


1. G. A. Bird, *Molecular Gas Dynamics* (Clarendon Press, Oxford, 1976).
2. S. Dietrich, ''An Efficient Computation of Particle Movement in 3D DSMC Calculations on Structured Body-Fitted Grids,'' in *Proceedings 17th Int. Symp. on Rarefied Gas Dynamics, Aachen, Germany, 1991,* edited by E. A. Beylich, p. 745.
3. I. D. Boyd, *J. Comput. Phys.* **96,** 411 (1991).
4. R. G. Wilmoth, ''Adaptive Domain Decomposition for Monte Carlo Simulations on Parallel Processors,'' *Proceedings, 17th Int. Symp. on Rarefied Gas Dynamics,* edited by E. A. Beylich, (VCH Press, Weinheim, 1991), p. 709.
5. L. Dagum, ''Comparisons of Two Approaches to Direct Particle Simulation on the Connection Machine,'' in *Proceedings, 18th Int. Symp. on Rarefied Gas Dynamics,* edited by B. D. Shizgal and D. P. Weaver (AIAA, Washington, 1994), p. 429.
6. J. D. McDonald, Report No. 589, Stanford University, Department of Aeronautics and Astronautics, 1989.
7. K. Dowd, *High Performance Computing* (O'Reilly, Sebastopol, 1993).
8. S. Dietrich and I. D. Boyd, ''Parallel Scalar Optimized Direct Simulation Monte Carlo Method for Rarefied Flows on the SP1,'' in *Lecture Notes in Computer Science,* Vol. 796, edited by W. Gentzsch and U. Harms (Springer-Verlag, Berlin 1994).
9. J. F. Thompson, *Comput. Systems Eng.* **3**(1–4), 393 (1992).
10. G. A. Bird, ''Perception of Numerical Methods in Rarefied Gasdynamics,'' in *Proceedings, 16th Int. Symp. on Rarefied Gas Dynamics,* edited by E. P. Muntz *et al.* (AIAA, Washington, DC, 1989), p. 221.
11. D. Baganoff and J. D. McDonald, *Phys. Fluids A* **2**(7), 1248 (1990).
12. G. A. Bird, ''Monte Carlo Simulation in an Engineering Context,'' in *Proceedings, 13th Int. Symp. on Rarefied Gas Dynamics,* edited by S. S. Fisher (AIAA, New York, 1981), p. 239.
13. C. Borgnakke and P. S. Larsen, *J. Comput. Phys.* **18,** 405 (1975).
14. F. Bergemann and I. D. Boyd, ''DSMC Simulation of Inelastic Collisions Using the Borgnakke–Larsen Method Extended to Discrete Distributions of Vibrational Energy,'' in *Proceedings, 18th Int. Symp. on Rarefied Gas Dynamics,* edited by B. D. Shizgal and D. P. Weaver (AIAA, Washington, 1994), Vol. 158, p. 174.
15. S. Dietrich, ''Improved DSMC-Modeling of Near-Continuum Flows with Chemical Reactions,'' in *Proceedings, 18th Int. Symp. on Rarefied Gas Dynamics,* edited by B. D. Shizgal and D. P. Weaver (AIAA, Washington, 1994), Vol. 159, p. 208.
16. S. Dietrich and I. D. Boyd, ''Loadbalancing for the DSMC Method on Parallel Computers,'' in *Proceedings, 19th Int. Symp. on Rarefied Gas Dynamics,* edited by J. K. Harvey (Oxford Univ. Press, Oxford, 1995), Vol. 2, p. 794.
17. I. D. Boyd, P. F. Penko, D. L. Meissner, and K. J. DeWitt, *AIAA J.* **30,** 2453 (1992).
18. M. S. Holden, J. Kolly, and K. Chadwick, AIAA Paper 95-0291, Reno, NV, Jan. 1995 (unpublished).
19. Williams, J. D. and Wilbur, P. J., *J. Spacecraft* **27**(6), 634 (1990).