Parallel Monte Carlo Simulation of Three-Dimensional Flow over a Flat Plate*

Robert P. Nance H. A. Hassan
Department of Mechanical and Aerospace Engineering
North Carolina State University
Raleigh, NC 27695-7910
[rpnance, hassan]@jupiter1.mae.ncsu.edu

Richard G. Wilmoth
Aerothermodynamics Branch, Gas Dynamics Division
NASA Langley Research Center
Hampton, VA 23681-0001
wilmoth@monte.larc.nasa.gov

Bongki Moon Joel Saltz
Institute for Advanced Computer Studies and Department of Computer Science
University of Maryland
College Park, MD 20742
[bkmoon, saltz]@cs.umd.edu

Abstract

This paper describes a parallel implementation of the direct simulation Monte Carlo method. Runtime library support is used for scheduling and execution of communication between nodes, and domain decomposition is performed dynamically to maintain a favorable load balance. Performance tests are conducted using the code to evaluate various remapping and remapping-interval policies, and it is shown that a one-dimensional chain-partitioning method works best for the problems considered. The parallel code is then used to simulate the Mach 20 nitrogen flow over a finite-thickness flat plate. It will be shown that the parallel algorithm produces results which are very similar to previous DSMC results, despite the increased resolution available. However, it yields significantly faster execution times than the scalar code, as well as very good load-balance and scalability characteristics.

Nomenclature

M_∞	Freestream Mach number
Re	Freestream Reynolds number
p_0	Stagnation pressure, bars
p_w	Surface pressure, Pa

* This work was supported by NASA Cooperative Agreement NCCI-112, the Mars Mission Research Center funded in part by NASA Grant NAGW-1331, a National Defense Science and Engineering Graduate Fellowship, NASA Grant NAG-1-1560, ARPA/NASA Grant NAG-1-1485, and NSF/NASA Grant ASC 9213821.

q_w Surface heat flux, W/m^2

t Time required to compute 1 time step

T_0 Stagnation temperature, K

T_w Surface temperature, K

$W(n)$ System degradation function

Z_r Rotational relaxation number

1 Introduction

The direct simulation Monte Carlo (DSMC) method of Bird [1] has become the standard method for the analysis of hypersonic rarefied flows. Since its inception, the method has been applied to more and more complex configurations, including the Space Shuttle orbiter geometry [2] and the Upper Atmosphere Research Satellite [3]. Furthermore, many DSMC analyses carried out today include physical phenomena such as thermal and chemical nonequilibrium [4]. The combination of complicated geometries and complicated flow physics leads to large processor-time and storage requirements, even for low-density calculations. For near-continuum DSMC applications, the resource requirements can render a meaningful simulation infeasible on current scalar architectures.

A new computational resource which may be brought to bear on DSMC problems is found in the advent of parallel computing. While parallel programming is still in its formative stages, parallel architectures show promise in being able to complete tasks in a fraction of the time required by contemporary sequential machines. This new architecture thus represents an opportunity to simulate flows at higher densities, or to perform many simulations in the time previously required for one simulation.

A considerable amount of effort has already been put into the parallelization of DSMC algorithms [5-10]. One of the aims of the present work is to utilize one such parallel implementation to analyze a problem of practical interest. The problem considered is described below.

2 CNRS Experiment

Figure 1 shows the pertinent dimensions of a finite-thickness flat plate with truncated leading edge tested at zero angle of attack in the SR3 low-density nitrogen tunnel at the Centre National de la Recherche Scientific (CNRS), Meudon, France [11]. The experimental results for the surface heat-transfer rate were compared to Navier-Stokes and DSMC results by the CNRS researchers; the computational-fluid-dynamics (CFD) results were shown to match the test data quite well, while the DSMC results overpredicted the heat flux across the length of the plate. Further efforts to correct the discrepancy in the DSMC results made little difference, as shown in the paper by Hash *et al.* [12] This further study included the consideration of effects such as collision model, grid refinement, and nonuniformities in the upstream test section. However, one possible physical phenomenon not considered in any of these solutions was the possibility of a three-dimensional relief effect, which could lower the heat flux to the plate. Therefore, one motivation for pursuing the problem is to obtain a solution for the complete flowfield. To this end, the flat plate was modeled first with the modified F3 code of Rault [13] and then with a modified version of the DSMC3 code of Bird [14].

The results from these computations suggested that, while no relief effect appeared to be present, increases in grid resolution tended to increase the agreement between the computed and experimental data. Both of these simulations were conducted on scalar machines; it was reasoned that solution on a parallel architecture would allow a simulation with greater resolution to be pursued with considerably less turnaround time.

3 Flow Conditions and Flow Physics

Table 1 lists the flow conditions for the CNRS experiment. Note that the stagnation temperature of 1100 K is quite low; therefore, vibrational excitation and dissociation are not expected to take place, and the only species considered was N_2. The Variable Hard Sphere (VHS) model of Bird [1] was utilized with a viscosity-temperature exponent of 0.75. Energy exchange between translational and rotational modes was determined through use of the Larsen-Borgnakke method [15] and a rotational relaxation number $Z_r = 5$. The surface of the plate was assumed to be diffusely reflective with full thermal accommodation. It should also be noted that the test conditions correspond to a freestream Knudsen number based on plate length of about 5.5×10^{-3}. Since rarefied-flow conditions are typically assumed to prevail above Knudsen numbers of around 0.01, these conditions correspond to a near-continuum flow.

Note in Figure 1 that the plate is partitioned into four equal sections. In the DSMC results to be discussed later, only one of these portions is considered. This simplification may be made because of the fact that the plate possesses two planes of symmetry, and that the experiment was carried out at zero angle of attack. Thus, consideration of the entire plate is unnecessary.

One of the objectives of this work was to develop a scalable parallel method while leaving the physics modeled in the original scalar code intact. To this end, no modifications have been made to the physical models employed by the original code for the sake of parallelization. Additionally, the possibilities explored in the earlier work on this problem (effect of collision model, upstream flow nonuniformity, *etc*.) have not been considered here, since the earlier studies showed that these variations made little difference in the results.

4 Parallel Algorithm

The method of parallelization used here utilizes runtime library support to carry out communication and data structure manipulations associated with molecule lists, as well as provide routines for remapping. This library—the CHAOS library—was developed at the University of Maryland, and is based upon the Parallel Automated Runtime Toolkit at ICASE (PARTI) library developed at the Institute for Computer Applications in Science and Engineering (ICASE) at NASA Langley Research Center [16-18]. The PARTI library was originally built for use with static irregular problems. These are problems where the data access patterns are known only at runtime, but the data access pattern is invariant once it has been defined. The data access patterns in irregular problems are determined through *indirection arrays*, which are arrays whose elements point to elements in another array, as shown in the FORTRAN code fragment below.

```
do 10 i=1,n
    sum=sum+x(icg(i))
```

```
 10   continue
```

The `icg` array in the above is an example of an indirection array.

DSMC represents a dynamic (or adaptive) irregular problem: data access patterns are known only at runtime, and can change as execution progresses. The data access patterns change because molecules move from cell to cell during the simulation, and molecule information is frequently referenced with respect to the cell in which a molecule resides. CHAOS was developed with such problems in mind, and utilizes a series of preprocessing steps in order to facilitate efficient computation [19].

First of all, CHAOS determines how data arrays are to be partitioned. This step involves the generation of a *translation table* which maps elements of the data arrays to their owner processors. This table is globally accessible. In this particular application, the table is replicated on each processor because the problem size is relatively small. However, memory considerations make it clear that it is not always feasible to replicate the table, so the translation table must be distributed across processors in some applications. Moon *et al.* [20] have recently developed new index translation schemes which use software caching techniques so that extra memory can be exploited adaptively for changeable data access patterns and communication latency can be avoided. However, as mentioned above, simple table replication was utilized for this study.

The second step is the actual remapping of the data; this remapping is carried out through (1) generation of an optimized interprocessor communication schedule and (2) use of scatter-append type procedures to move the data to the appropriate locations. These entities are discussed below.

Since molecules may move from cells owned by one processor to cells owned by another, it is necessary to communicate molecule-based data between processors even if the problem partition does not change. Since such communication is required every time step, communication optimization is crucial for efficient parallel computation in DSMC. Communication optimization is achieved by CHAOS in two aspects. First, the overhead of communication-schedule construction is further reduced by exploiting the data independence characteristic of DSMC. Standard CHAOS communication schedules specify information such as data placement order of off-processor elements, a list of local elements required by the other processors, and so on. Since the communication pattern is irregular and determined at runtime, it is necessary to build a communication schedule at every time step; the standard communication schedule therefore tends to be impractical for DSMC computations. Considering that the order of molecules in a cell is not important, the amount of information stored can be reduced by omitting the placement order of the molecules. We call this type of communication schedule a *lightweight* communication schedule. Second, the actual communication using the lightweight schedules is optimized by *communication vectorization*. Suppose the columns in a multidimensional array are distributed in the same manner and the access pattern is the same for each column. Then, with communication vectorization, the CHAOS library allows data to be moved from all the columns by a single invocation of a data-transfer function using a single communication schedule.This operation does not reduce the communication volume, but reduces the latency by the number of columns.

Scatter-append operations are very useful in DSMC because, once the movement phase is completed, DSMC cells can be operated on in any order (unlike CFD, where knowledge of the cell order and an orderly sweep through the domain are very important). Thus, data need only be appended to existing data lists for each cell, and costly reordering of the data is unnecessary. Instead, a cross-reference array is used to associate molecules with the corresponding cells.

As discussed earlier, DSMC is a highly dynamic method; that is, the molecules simulated by the code are not uniformly distributed, and the distribution varies considerably as the simulation progresses. To help maintain an acceptable load balance, CHAOS also supports several methods for repartitioning the domain. The basic premise of any load-balancing algorithm is to partition the domain so that each processor must perform approximately the same amount of work. However, a criterion must be selected as the basis for measuring the amount of work owned by each processor; in this study, the amount of compute time expended per cell was used as a workload measure. Several options are available for decomposing the domain; three possibilities investigated here are recursive coordinate bisection (RCB) [21], recursive inertial bisection (RIB) [22], and one-dimensional chain partitioning [23].

In the first two algorithms, the domain is recursively halved (with each new portion of the domain possessing an equal amount of "work") until there are as many subdomains as processors. The difference between RIB and RCB is that RIB chooses the partitioning direction as the direction with the minimum "inertia." In other words, if the data in the domain tend to be clustered around an axis different from one of the coordinate axes, RIB will find that axis and partition normal to it. On the other hand, RCB simply chooses bisecting directions from planes normal to the x, y, or z directions. The third choice—chain partitioning—is a very inexpensive method that works well for certain problems. Here, the domain is partitioned into many contiguous strips, or chains, with each of these chains containing about the same amount of work. The chain-partitioning method implemented here seeks to reduce the remapping cost even further by considering only the cost of computation in determining the amount of work owned by each processor; communication costs are not considered. An additional advantage of the chain partitioner over the two bisection methods is that it can be used with any number of processors, whereas the bisection methods require 2^N processors. Figures 2, 3, and 4 show problem domains decomposed using RCB, RIB, and chain partitioning, respectively. When any of these repartitioning methods are used, both cell-based and molecule-based data must be remapped as well. The same lightweight communication schedules and data-transfer procedures discussed earlier can be used to perform this remapping.

For dynamic problems such as DSMC, the workload distribution can change drastically during execution, leading to a high degree of load imbalance among the processors in use. Thus, the partitioning methods described above are reapplied at fixed or varying intervals. It has been shown that, for many problems solved using a load-balancing algorithm, remapping the domain at fixed intervals can lead to poor performance. Therefore, it is desirable to either determine the optimum interval for remapping, or employ a monitoring policy which actively decides when remapping is necessary. The former choice is not practicable for most problems, since it requires pre-runtime analysis to determine the optimal interval for remapping. Thus, in this study, a variable-interval remapping policy is investigated as well; the method employed is the Stop at Rise (SAR) policy suggested by Nicol and Saltz [24]. This remapping policy chooses to repartition the domain based on the value of a system degradation function W, which is defined as follows:

$$W(n) = \frac{\sum_{j=1}^{n} \left(t_{\max}(j) - t_{\mathrm{avg}}(j) \right) + C}{n} \tag{1}$$

In the above, n is the number of time steps since the last remapping (which occurred in the step just before step “1”), $t_{\max}(j)$ is the maximum amount of time required by any one processor during the jth time step (and thus the amount of time required to complete the jth time step), $t_{\text{avg}}(j)$ is the average time required by a processor to complete the jth time step, and C is the amount of time required to complete the remapping operation. This quantity is monitored during the computation, and represents the average processor idle time per step achieved by remapping immediately. Repartitioning is performed after the first value of n such that $W(n) > W(n-1)$ — that is, when the first local minimum is detected. The function $W(n)$ initially tends to decrease as n increases, because the remapping cost C is amortized over an increasing number of time steps. However, as n increases, the summation term in Equation (1) will eventually increase as well, indicating a loss of workload balance and a need to remap. This remapping method is advantageous in that no prior knowledge of the problem is necessary for the determination of the remapping interval, and the remapping interval can be expected to adapt to the dynamics of the problem.

An additional note about remapping is in order here. The DSMC algorithm under discussion was developed for steady flows; therefore, we expect to observe a transient phase leading to a steady state, after which the flowfield is sampled for a large number of time steps in order to remove scatter from the results. If we are indeed at a steady state, it would make sense to retain the steady-state mapping for the remainder of the calculation. However, for the cases considered herein, remapping was performed for the entire duration of the simulation.

These procedures were incorporated into the modified DSMC3 code of Bird and ported to the 72-node Intel Paragon recently brought on line at NASA Langley Research Center. The results presented herein were obtained using up to 64 nodes on this machine.

5 Results and Discussion

5.1 Parallel Performance Results

It was desired to evaluate the effect of different partitioning methods and remapping policies on the performance of the parallel algorithm. There are two frequently used criteria for making such an evaluation—speedup and scaleup—and both are examined here. Figures 5 through 7 show speedup results for the method applied to the flow over a zero-thickness flat plate at the CNRS test conditions, using a $48 \times 16 \times 2$ grid. The speedup is defined for a particular problem as:

$$\text{Speedup} = \frac{t_1}{t_p} \tag{2}$$

where t_1 is the time for the problem to be completed using the scalar code and t_p is the time for the *same* problem to run on p processors. These results are useful in delineating performance differences between the possible setups.

Figure 5 shows the speedup results for a fixed remapping interval and different partitioning methods. This plot shows that the chain partitioner yields slightly better speedup than the other partitioners. Note further that all three methods exhibit considerably less-than-ideal speedup. While this observation appears to indicate poor performance, it should be noted that perfect speedup should only be expected in perfectly parallel problems. While the degree of data independence in DSMC is quite high, interprocessor communications are still required when particles migrate between nodes. This migration is more frequent for a given problem when it is run on a larger number of processors, since the size of each subdomain is decreased. Therefore, we would expect that, as the number of processors is increased, the communication overhead would similarly increase, thereby reducing the relative benefit of increasing the number of processors, and speedup alone should not be used to judge the worth of a parallel algorithm. However, speedup results are still useful for detecting differences in performance, as Figure 5 demonstrates.

Another statement which may be made about the poor speedup data is that the speedup may be increased by reducing the communication overhead. Such a reduction may be realized by faster global operations (such as global sums and comparisons). Some performance studies have shown that the performance of the standard global operations supported by the current Paragon operating system suffers significantly for large arrays, and that the use of other communications libraries, such as the InterCom library [25], could yield a considerable increase in performance. This issue shall be addressed further in future work.

In Figure 6, we examine the performance of various remapping policies on a simulation partitioned using recursive inertial bisection. These results show that SAR performs better than the fixed-interval remapping, but also that, for this particular problem, the static partition (no remapping) actually offers the best performance for large numbers of nodes. This same phenomenon may be observed in Figure 7 also, and may be explained as follows. For these speedup tests, the problem considered is relatively small; for large numbers of nodes, the amount of time required to remap the domain is of the order of the time spent in computation between remaps. Thus, the cost of *any* remapping actually exceeds the cost incurred by load imbalance for large numbers of processors. Figures 6 and 7 also show that the fixed-interval remapping and SAR offer comparable speedup performance for the test case.

We may also examine the scalability of a parallel algorithm. To do so, we increase the problem size as the number of processors is increased, and compute scaleup in a manner similar to that for speedup:

$$\text{Scaleup} = \frac{t_1}{\tau_p} \tag{3}$$

where t_1 is the amount of time for the scalar code to run on one processor, and τ_p is the amount of time to run on p processors, where the problem size is greater than the original problem size by the factor p. Note that a scaleup value of unity indicates perfect problem scalability.

Figure 8 shows scaleup performance for the different partitioning methods and a fixed remapping interval. To obtain these results, the scalar code was first run using the geometry and conditions described above and a $48 \times 8 \times 2$ grid. Each time the number of processors was doubled, the resolution in the y direction was doubled as well. Since the number of particles per cell was held constant each time the resolution was doubled, the number of simulated particles was also approximately increased by a factor of two. The scaleup results show that each of the three partitioning methods offers good scalability. In each case, the scaleup decreases slightly as the number of processors is increased (for large numbers of processors), but the curve is relatively flat. Note also that some of the scaleup values exceed unity. This unusual characteristic may be explained by the fact that simply doubling the resolution in a particular direction will not precisely double the problem size. Since DSMC is not a deterministic method, it is very difficult to regulate the problem size with great precision. Therefore, the results in Figure 8 should be viewed as a qualitative measure of the scalability of the algorithm. In this respect, the present method does demonstrate very good scalability.

Having examined the overall performance characteristics of the code, let us now look at how the computational work is divided among the component subroutines. Table 2 lists such breakdowns for the test case, obtained using a remapping interval of 20 timesteps and the $48 \times 16 \times 2$ grid. (Note that routines consuming less than 0.5% of the total CPU time are not included in this table; therefore, the column sums do not equal 100%.) As the number of nodes is increased, it is apparent that the relative amount of simulation-related work (movement, collision, *etc*.) decreases, and the time spent in parallel-related operations (such as message-passing and wasted time due to synchronization) goes up. This behavior is not desirable, since we would expect the algorithm to be dominated by parallel operations for larger numbers of nodes; these trends could be a result of the relatively small problem size, or could be a sign that the fixed-interval remapping policy used here is not an optimal one.

5.2 CNRS Comparison Results

Prior to conducting a grid resolution study using the parallel code, it was desired to determine which code setup would be best for the CNRS problem. Therefore, an abbreviated set of performance-evaluation runs was conducted using a moderately dense ($52 \times 48 \times 26$) grid and about 697,000 molecules on average. The results of these tests are shown in Table 3; in each case, the code was run 1000 steady-state timesteps on 32 nodes. We can see that the combination of the chain partitioner and SAR remapping produces an overall compute time and load imbalance comparable to the best fixed-interval results obtained, but less computational effort per particle. (The load imbalance is defined as the ratio of the compute time required for the bottleneck processor to the average compute time for all the nodes.) This seemingly contradictory behavior may be attributed to the fact that a larger average number of particles was present in the first case than in the fixed-interval cases.

Let us further examine the application of the SAR policy to a full-scale case through the profile data shown in Table 4. In contrast to the results shown in Table 2 for the test case, we see that the amount of time spent in each type of operation remains fairly constant. This trend indicates that the scalability for this choice of problem size and remap policy is significantly improved over that for the test case. In light of the results shown in Tables 3 and 4, chain partitioning and SAR were used for the grid-refinement study.

In order to find a grid-independent solution, the parallel code was next used to obtain results for successively finer grids; the results for the final grid are shown in Figure 9. This case utilized a $52 \times 96 \times 26$ grid and approximately 1.4 million simulated molecules. It may be seen that the parallel DSMC results for the heat flux are nearly identical to the previous DSMC results. As stated earlier, it was thought that increases in grid resolution might lead to a reconciliation of the DSMC results and experimental data. However, as Figure 8 demonstrates, the difference between the computed and experimental results is still quite large, and the reason for this disparity remains a mystery.

We may also compare the surface pressure results, as shown in Figure 10. In this case, the computed results do not even follow the same trend as the experimental results. While the reason for the discrepancy is unclear, one possible explanation follows.

As described in Reference 11, the experimental wall-pressure measurements were made using 1-mm-diameter orifices connected to external pressure transducers. These transducers consist of a diaphragm separating two cavities—one connected to the pressure orifice, the other connected to a known reference vacuum—and an electrical current is passed through the diaphragm. The pressure differential causes a displacement of the diaphragm, and a corresponding change in the voltage across the diaphragm. This voltage change may be converted into a pressure measurement. However, one possible source of error lies in the fact that there is a finite length of piping connecting the orifice to the diaphragm; this piping could induce frictional losses in the flow between orifice and diaphragm, resulting in a lower measured pressure. From this standpoint, one might expect the calculated surface pressures to be greater than the measured surface pressures, which is the trend seen in Figure 10. A second possibility is that the flow had not reached steady state at the time the measurements were taken. If this was the case, the accuracy of the pressure measurements would certainly have been affected adversely.

It should be reiterated that the problem considered here required very simple flow physics; moreover, the geometry itself was quite simple. One may ask whether the present method could be readily expanded to more comprehensive flow physics and more complex surface geometries. As far as the latter is concerned, inclusion of other physical phenomena should be relatively straightforward. In DSMC, any physical process such as dissociation or ionization requires a collision between simulated particles, and the collision coding used herein is virtually unchanged from the scalar algorithm. The main difference would be that new arrays would be necessary to keep track of the additional particle information (such as vibrational energy state), and these arrays would have to be distributed in the same fashion as other data arrays. In order to incorporate more general geometries into the code capability, a nonuniform grid would most likely be necessary. Such a grid would necessarily complicate the movement and indexing phases of the method, and these added difficulties would translate into further difficulties with respect to parallelization. However, these problems can most likely be overcome, and will be part of the focus of further work.

An additional note regarding use of the parallel code on more complex configurations is in order. As mentioned before, the best choice of partitioning strategy is problem-dependent, and methods which worked well for the simple geometries considered herein may produce poor performance for other geometries. For instance, the chain partitioner discussed herein was quite useful for both the test case and the CNRS case. However, it may not be as useful in flows where a large percentage of the particles move in a direction other than the freestream flow direction, such as blunt-body wake flows. In any case, one should evaluate the available domain-decomposition options for a new problem prior to attempting a full-blown simulation of the problem.

6 Conclusions

The parallel implementation of the DSMC method presented here was successful in producing results which compared well with scalar DSMC results for the simple test case. Through use of the parallel code, it was possible to increase grid resolution and still obtain solutions for the CNRS comparison case in a reasonable amount of time. However, the increase in grid resolution did not improve the agreement between the experimental and computed results for the surface properties.

Performance results for the test case and the CNRS case indicate that the best partitioning and remapping policies are problem dependent; some of the load-balancing strategies which produced acceptable performance for the test case worked poorly for the CNRS case. Thus, it is important to know something about the flow under consideration before deciding what set of parallel-execution parameters to employ.

Acknowledgments

The authors would like to thank the Information Systems Division at NASA Langley Research Center for Intel Paragon access and technical support.

References

[1] Bird, G.A., "Monte Carlo Simulation in an Engineering Context," *Progress in Astronautics and Aeronautics: Rarefied Gas Dynamics*, Vol. 74, Pt. 1, AIAA, New York, 1981, pp. 239-255.

[2] Rault, D. F. G., "Aerodynamics of Shuttle Orbiter at High Altitudes," AIAA Paper 93-2815, July 1993.

[3] Woronowicz, M. S. and Rault, D. F. G., "On Predicting Contamination Levels of HALOE Optics aboard UARS Using Direct Simulation Monte Carlo," AIAA Paper 93-2869, July 1993.

[4] Taylor, J. C., Carlson, A. B., and Hassan, H. A., "Monte Carlo Simulation of Radiating Reentry Flows," *Journal of Thermophysics and Heat Transfer*, Vol. 9, No. 3, 1994, pp. 478-485.

[5] Wilmoth, R. G., "Adaptive Domain Decomposition for Monte Carlo Simulations on Parallel Processors," *Proceedings of the 17th International Symposium on Rarefied Gas Dynamics*, VCH Publishers, New York, 1991, pp. 709-716.

[6] McDonald, J. and Dagum, L., "A Comparison of Particle Simulation Implementations on Two Different Parallel Architectures," *Proceedings of the Sixth Distributed Memory Computing Conference*, IEEE Computer Society Press, Knoxville, TN, 1991, pp. 413-419.

[7] Fallavollita, M. A., McDonald, J., and Baganoff, D., "Parallel Implementation of a Particle Simulation for Modeling Rarefied Gas Dynamic Flow," *Computing Systems in Engineering*, Vol. 3, No. 1-4, 1992, pp. 283-289.

[8] Bartel, T. J. and Plimpton, S. J., "DSMC Simulation of Rarefied Gas Dynamics on a Large Hypercube Supercomputer," AIAA Paper 92-2860, June 1992.

[9] Boyd, I. and Dietrich, S., "A Scalar Optimized Parallel Implementation of the DSMC Method," AIAA Paper 94-0355, January 1994.

[10] Long, L. N., Wong, B. C., and Myczkowskifl, J., "Deterministic and Nondeterministic Algorithms for Rarefied Gas Dynamics," *Rarefied Gas Dynamics: Theory and Simulations*, Shizgal, B. D. and Weaver, D. P., editors; *Progress in Astronautics and Aeronautics*, Vol. 159, 1994, pp. 361-370.

[11] Allegre, J., Raffin, M., Chpoun, A., and Gottesdiener, L., "Rarefied Hypersonic Flow over a Flat Plate with Truncated Leading Edge," *Rarefied Gas Dynamics: Space Science and Engineering*, Shizgal, B. D. and Weaver, D. P., editors; *Progress in Astronautics and Aeronautics*, Vol. 160, 1994, pp. 285-295.

[12] Hash, D. B., Moss, J. N., and Hassan, H. A., "Direct Simulation of Diatomic Gases Using the Generalized Hard Sphere Model," *Journal of Thermophysics and Heat Transfer*, Vol. 6, No. 4, 1994, pp. 758-761.

[13] Rault, D. F. G., "Efficient Three-Dimensional Direct Simulation Monte Carlo Code for Complex Geometry Problems," *Rarefied Gas Dynamics: Theory and Simulations*, Shizgal, B. D. and Weaver, D. P., editors; *Progress in Astronautics and Aeronautics*, Vol. 159, 1994, pp. 137-154.

[14] Bird, G. A., *Molecular Gas Dynamics and the Direct Simulation of Gas Flows,* Clarendon Press, Oxford, 1994.

[15] Borgnakke, C., and Larsen, P. S., "Statistical Collision Model for Monte Carlo Simulation of Polyatomic Gas Mixtures," *Journal of Computational Physics*, Vol. 18, No. 3, 1975, pp. 405-420.

[16] Das, R., and Saltz, J., "Parallelizing Molecular Dynamics Codes Using the Parti Software," *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, 1993, pp. 187-192.

[17] Saltz, J., Mirchandaney, R., and Crowley, K., "Run-Time Parallelization and Scheduling of Loops," *IEEE Transactions on Computers*, Vol. 40, No. 5, May 1991, pp. 603-612.

[18] Saltz, J., Berryman, H., and Wu, J., "Multiprocessors and Run-Time Compilation," *Concurrency: Practice and Experience*, Vol. 3, No. 6, December 1991, pp. 573-592.

[19] Hwang, Y. S., Moon, B., Sharma, S., Ponnusamy, R., Das, R., and Saltz, J., "Runtime and Language Support for Compiling Adaptive Irregular Programs on Distributed Memory Machines," *Software Practice and Experience*, to appear.

[20] Moon, B., Uysal, M., and Saltz, J., "Index Translation Schemes for Adaptive Computations on Distributed Memory Multicomputers," *Proceedings of the Ninth International Parallel Processing Symposium*, IEEE Computer Society Press, Santa Barbara, CA, April 1995 (to be published).

[21] Berger, M. J., and Bokhari, S. H., "A Partitioning Strategy for Nonuniform Problems on Multiprocessors," *IEEE Transactions on Computers*, Vol. 36, No. 5, May 1987, pp. 570-580.

[22] Nour-Omid, B., Raefsky, A., and Lyzenga, G., "Solving Finite Element Equations on Concurrent Computers," *Parallel Computations and their Impact on Mechanics*, ASME, New York, 1987, pp. 209-227.

[23] Moon, B., and Saltz, J., "Adaptive Runtime Support for Direct Simulation Monte Carlo Methods on Distributed Memory Architectures," *Proceedings of the Scalable High Performance Computing Conference (SHPCC94)*, IEEE Computer Society Press, Knoxville, TN, May 1994, pp. 357-364.

[24] Nicol, D. M. and Saltz, J., "Dynamic Remapping of Parallel Computations with Varying Resource Demands," *IEEE Transactions on Computers*, Vol. 37, No.9, September 1988, pp.1073-1087.

[25] Barnett, M., Gupta, S., Payne, D., Shuler, L., van de Geijn, R., and Watts, J., "Interprocessor Collective Communication Library (InterCom)," *Proceedings of the Scalable High Performance Computing Conference (SHPCC94)*, IEEE Computer Society Press, Knoxville, TN, May 1994, pp. 357-364.

Tables

Table 1. CNRS test conditions.

Property	Value
M_∞	20
T_∞	14 K
Re	8380
p_0	10 bars
T_0	1100 K
T_w	290 K

Table 2. Runtime profiling data for test case (fixed-interval remapping).

Routine	% CPU (8 nodes)	% CPU (16 nodes)	% CPU (32 nodes)
Movement	27.0	24.1	16.6
Indexing	5.9	5.4	3.4
Collision	12.8	11.0	7.9
Sampling	2.0	1.9	1.4
Math operations	16.7	14.1	10.7
Parallel/system operations	30.9	37.8	51.7

Table 3. CNRS case performance results.

Partition/Remap interval	Execution time (seconds)	Degree of load imbalance	Computational efficiency (μsec/particle/ timestep)
Chain/SAR	1152	1.044	1.65
Chain/Fixed (75 timesteps)	1094	1.023	1.68
Chain/Fixed (50 timesteps)	1143	1.025	1.70
Chain/Fixed (25 timesteps)	1282	1.048	1.87
RIB/SAR	1261	1.089	1.81

Table 4. Runtime profiling data for CNRS case (SAR policy).

Routine	% CPU (8 nodes)	% CPU (16 nodes)	% CPU (32 nodes)
Movement	39.0	38.1	35.8
Indexing	9.5	9.1	7.9
Collision	10.6	10.9	10.8
Sampling	3.2	3.1	2.7
Math operations	11.0	11.6	11.9
Parallel/system operations	24.0	25.1	28.7

Figures

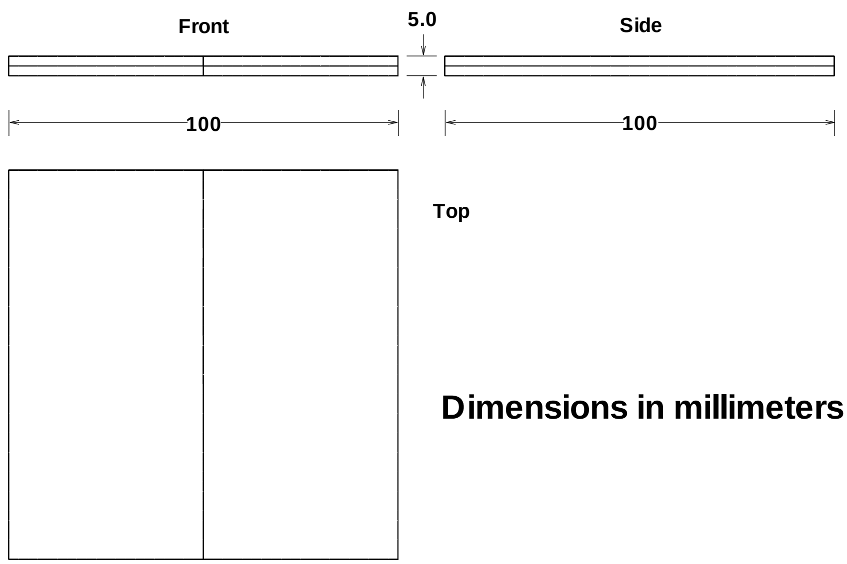

Figure 1: CNRS flat-plate geometry

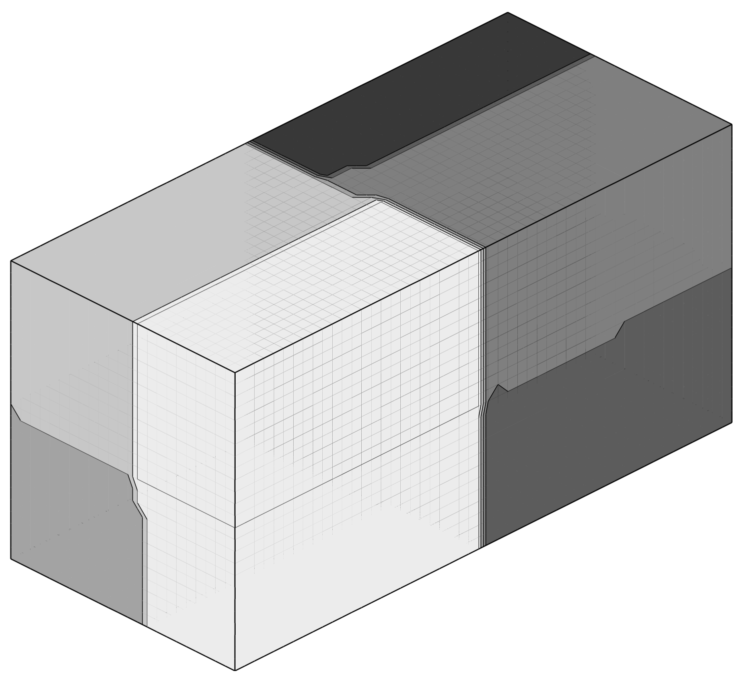

Figure 2: Domain decomposition with RCB

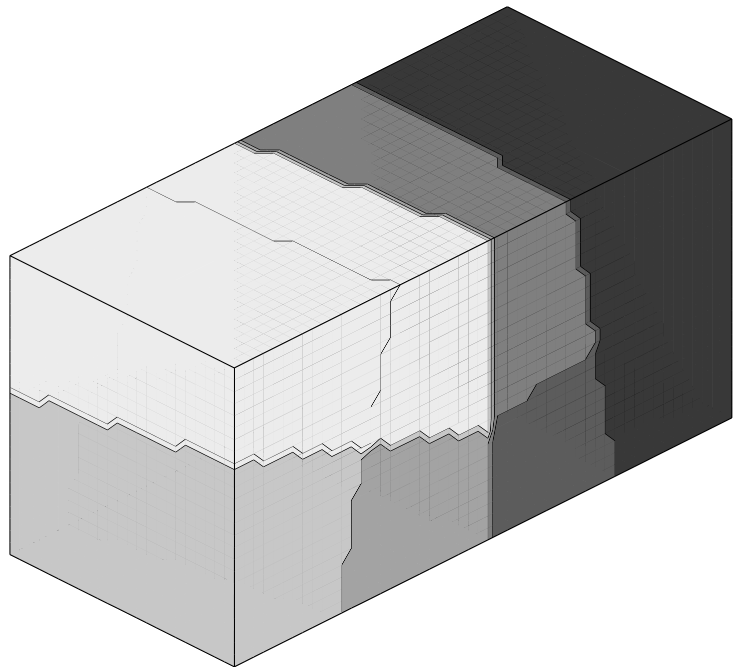

Figure 3: Domain decomposition with RIB

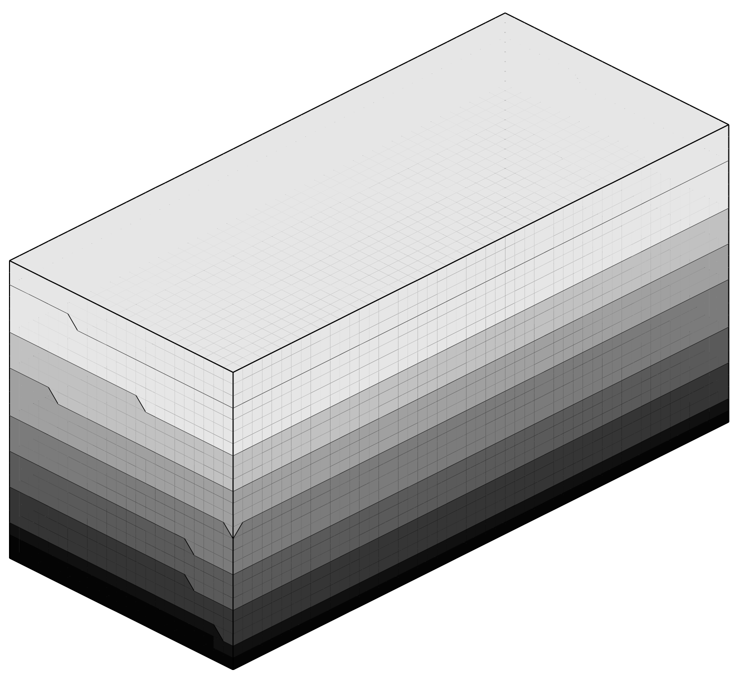

Figure 4: Domain decomposition with the chain partitioner

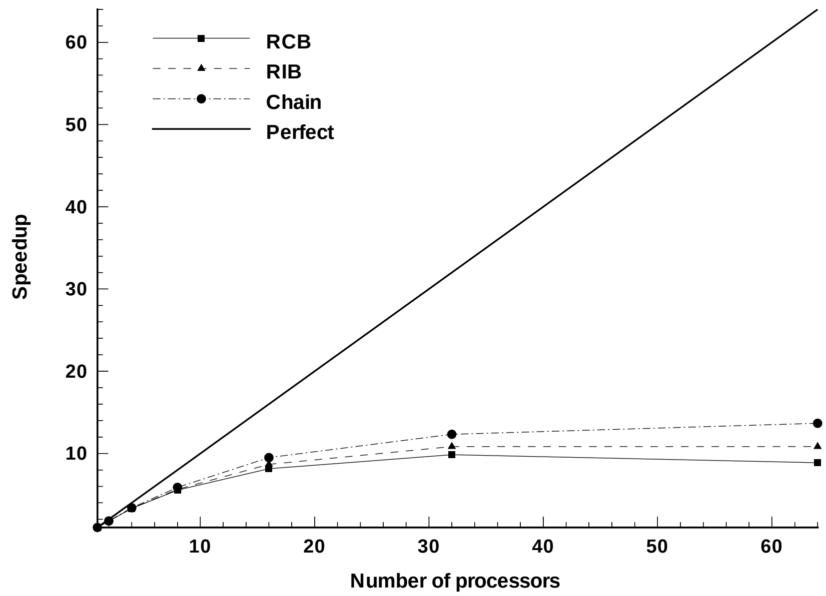

Figure 5: Speedup results: Effect of partition method

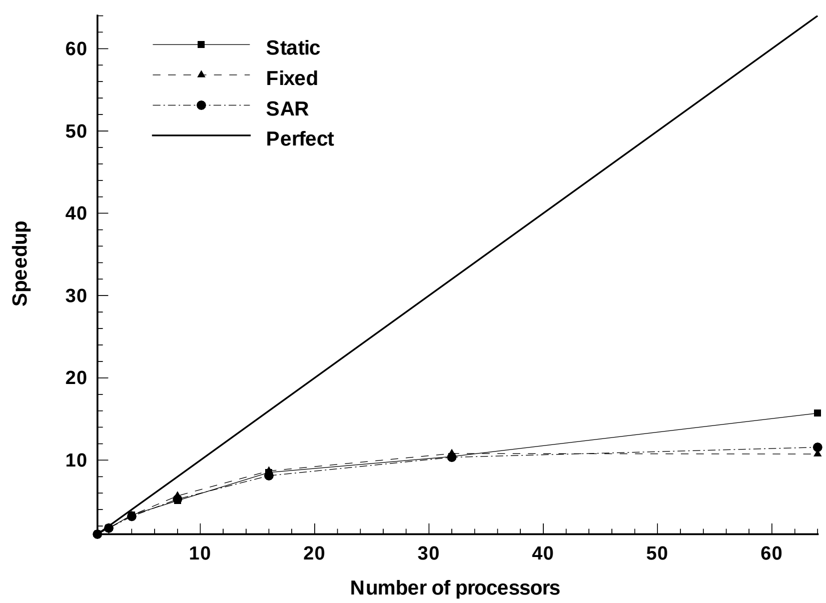

Figure 6: Speedup results: Effect of remapping method (RIB)

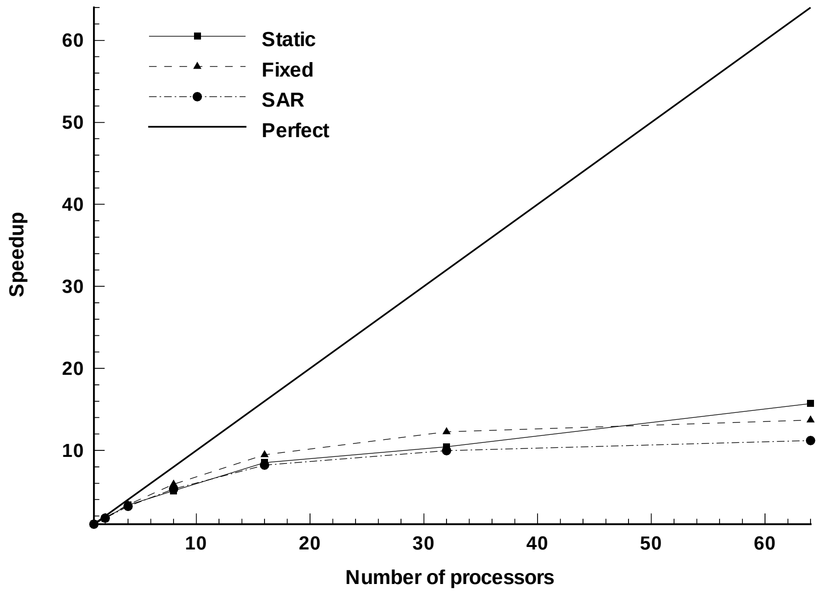

Figure 7: Speedup results: Effect of remapping method (chain partitioner)

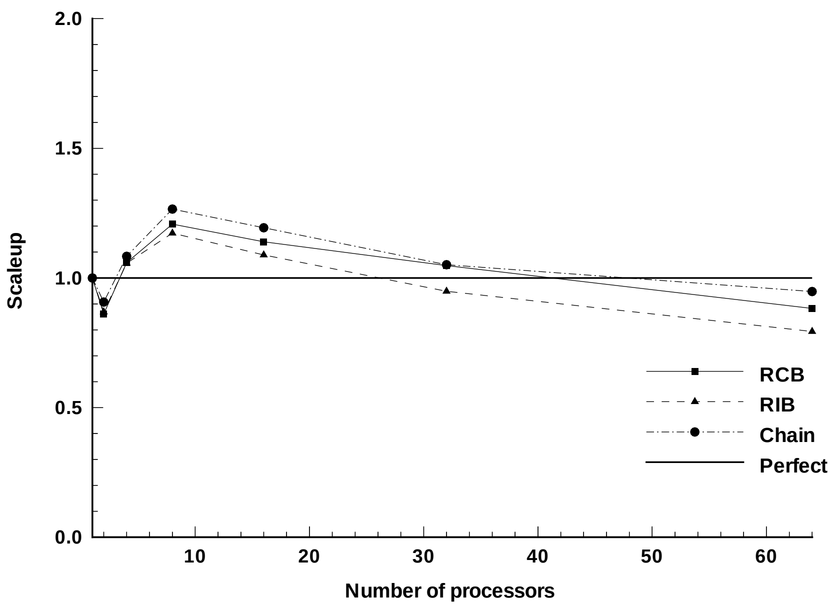

Figure 8: Scalability results: Effect of partition method

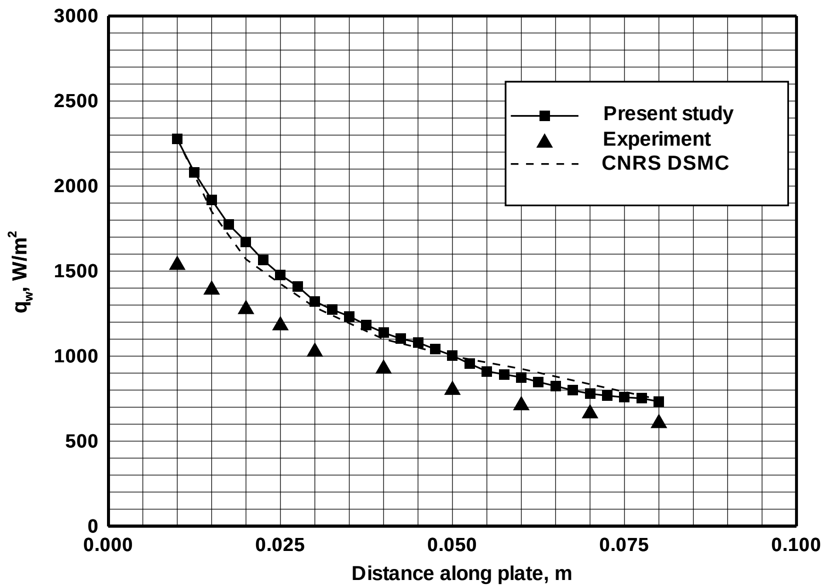

Figure 9: Heat flux results at $z = 0$ for the parallel code ($52 \times 96 \times 26$ grid)

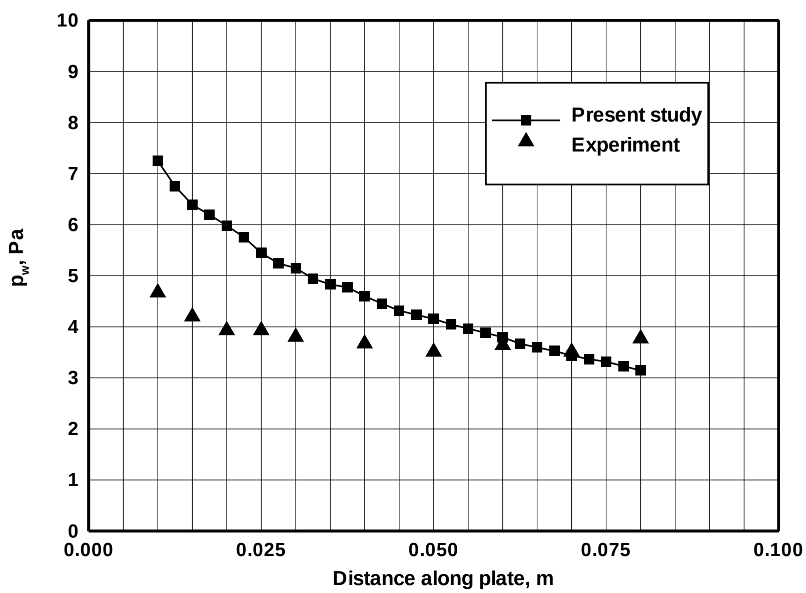

Figure 10: Surface pressure at $z = 0$ for the parallel code