AIAA JOURNAL
Vol. 30, No. 10, October 1992

Application of a Parallel Direct Simulation Monte Carlo Method to Hypersonic Rarefied Flows

Richard G. Wilmoth*
NASA Langley Research Center, Hampton, Virginia 23665

Application of a method for performing direct simulation Monte Carlo calculations using parallel processing to several hypersonic, rarefied flow problems is presented. The performance and efficiency of the parallel method are discussed in terms of some simple benchmark problems. The applications described are the flow in a channel and the flow about a flat plate at incidence. The benchmark results show significant advantages of parallel processing over conventional scalar processing and demonstrate the scalability of the method to large problems. The applications to hypersonic rarefied flows demonstrate the capabilities of the method, and the results demonstrate the need to adapt the parallel decomposition to the flowfield in order to achieve good load balancing.

Introduction

THE direct simulation Monte Carlo (DSMC) method developed by Bird[1] is one of the most widely used methods for analyzing hypersonic rarefied flows. The method has been applied to a wide range of problems, and recent advances have made practical the application of the method to the full Space Shuttle Orbiter geometry.[2] These advances provide a more efficient scheme for dealing with complex three-dimensional geometry as well as significant reductions in the computational effort. As currently implemented, the computer memory requirements for the new scheme increase rapidly as the freestream density increases. Therefore, existing DSMC codes that have been developed for treating higher density entry flows with complex real-gas effects remain in use for many problems. Although these more general codes have schemes for dealing efficiently with large variations in density, they still require significant amounts of computer time for near-continuum conditions because of the increase in the collision frequency with increasing density. With some planned improvements, the DSMC codes based on the new scheme will undoubtedly replace the older codes for most applications. However, further reductions in computational time are needed to make the treatment of high-density, large-scale problems more tractable.

Recent advances in computer hardware for parallel processing together with the development of efficient algorithms for doing DSMC calculations in parallel[3-6] offer the potential for significant reductions in computational times. Parallel methods can achieve and even surpass the performance of current supercomputers such as the Cray-2 for certain types of problems.[6] The implementation of these parallel algorithms does not require any fundamental changes to the simulation algorithm of Bird and, therefore, can, in principle, be applied to the general codes. Such an implementation will hopefully allow a wider range of flight conditions to be studied and make possible a direct comparison of DSMC results with continuum Navier-Stokes solutions for selected problems.

The purpose of the present paper is to present some initial results for a parallel DSMC method applied to hypersonic rarefied flow problems of interest. The parallel method is described, and some benchmark results are presented to show the potential performance relative to other supercomputers. Results are then presented for the rarefied hypersonic flow in a two-dimensional channel and for the flow about a flat plate at angle of attack to demonstrate the capability of the method. These calculations used an adaptive procedure for distributing the computational work load among the processors and were performed on an Intel iPSC/860 hypercube with up to 128 processors.

Parallel Direct Simulation Monte Carlo Method

The parallel DSMC method is based on domain decomposition in which a portion of the physical space along with its associated cells and molecules is allocated to each processor. (See Fig. 1.) A complete DSMC code is loaded on each processor, and the simulation in each subdomain proceeds essentially independently of the other subdomains. Molecules that cross the subdomain boundaries during the movement portion of the algorithm are sent to the appropriate processor using the message-passing facilities of the hypercube. Thus, the simulations are synchronized between the movement and collision routines at each time step. Since each processor contains a complete copy of the basic simulation code, the addition to or modification of the basic simulation algorithm to include more complex geometry or collision physics presents no problem, at least in principle.

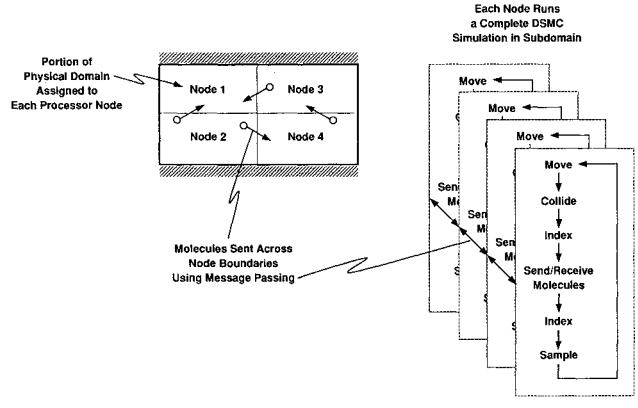


Fig. 1 Schematic of parallel DSMC method.

Presented as Paper 91-0772 at the AIAA 29th Aerospace Sciences Meeting, Reno, NV, Jan. 7-10, 1991; received June 21, 1991; revision received March 7, 1992; accepted for publication March 12, 1992.

*Aerospace Engineer, Aerothermodynamics Branch, Space Systems Division. Senior Member AIAA.

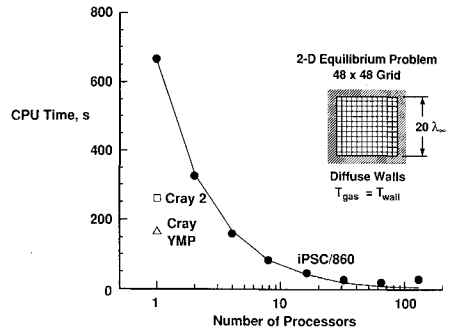
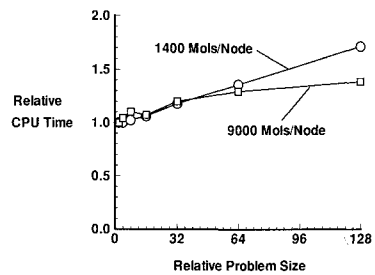
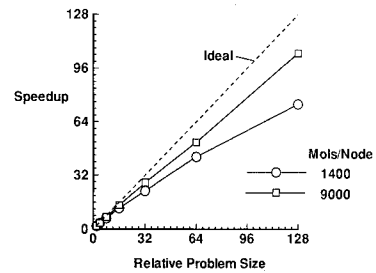


Fig. 2 Parallel DSMC benchmark results for a two-dimensional equilibrium gas simulation.



a) Relative CPU time



b) Speedup

Fig. 3 Scalability of parallel DSMC method as a function of problem size (relative problem size = number of processors).

The basic performance of the parallel DSMC method is illustrated in Fig. 2. This figure shows the computational time (average CPU time per processor) as a function of the number of processors compared with the time for the same calculation using an unvectorized, scalar code on single processing units of the Cray-2 and Cray-YMP. The problem was a simulation of equilibrium conditions in a stationary box with the wall temperature equal to the gas temperature. Since the mean velocity is zero and the temperature is uniform, there are no mean-flow gradients in the problem. The subdomains allocated to each processor were of equal size, and all contained the same number of cells and the same average number of molecules. Therefore, the parallel calculation achieves relatively good load balancing, and the computational times decrease as the number of processors increases up to about 64 processors. With 128 processors, the communication time and synchronization overhead have increased to the point where the average time spent in each processor starts to rise. Thus, for a fixed-size problem, the speedup reaches a maximum at some number of processors, and increasing the number of processors beyond this number will lead to unacceptably low efficiency.

The ratio of computational time to communication time can be controlled by properly scaling the size of the problem to the number of processors, as discussed in Ref. 5. Thus, if the problem of Fig. 2 is scaled upward in proportion to the number of processors, the efficiency of the parallel method should remain relatively constant. Results are shown in Fig. 3 for a study in which the problem size was scaled by increasing the size of the physical domain and the corresponding number of cells and molecules in direct proportion to the number of processors. The relative problem size is just the ratio of the total number of molecules (or cells) to that for a reference problem solved on a single i860 processor. Two reference problems were considered—one having an average of 1400 molecules/node and the other having an average of 9000 molecules/node. Figure 3a shows that the relative CPU time (average CPU time/node ratioed to that for the reference problem) increased by less than a factor of 2 as the relative problem size increased by a factor of 128. Figure 3b shows the speedup for both cases where the dashed line is the ideal speedup. The larger reference problem (9000 molecules/node) gave the best overall performance and shows an almost linear increase in speedup with relative problem size (or equivalently with the number of processors). Such a linear scalability is a key requirement of any parallel algorithm in order that larger problems may be solved without a proportional increase in CPU time.

For the largest benchmark problems solved (using 128 nodes), over 1 million particles were advanced through 200 time steps with over 40 million collisions in a wall-clock time of 105 s. This yields an effective computational rate of approximately 2 million particle time steps/s which is of the same order as that achieved by McDonald[7] using a highly vectorized particle simulation code on the Cray-2. However, in order to make a meaningful comparison between parallel performance on the iPSC/860 and performance on vector architectures like the Cray-2, it is important to look at potential gains that might be obtained by vectorizing the current code. For example, over 4000 CPU s were required to solve the largest benchmark problem using an unvectorized, scalar version of the present DSMC code on a Cray-2. However, Boyd[8] has shown that speedups due to vectorization of about 5 were achievable with moderate changes to the FORTRAN coding of Bird's no-time-counter DSMC algorithm. Since the present study used the same basic DSMC algorithm as Boyd, it is reasonable to expect that similar vector improvements could be made to the present code for running on the Cray-2. Even with such improvements, it would appear that the iPSC/860 still offers significantly lower computing times for the larger problems when using the current DSMC algorithm.

Adaptive Domain Decomposition

For real problems of interest, the flowfields will be nonuniform, and the simple domain decomposition used for the benchmark equilibrium problem (equal cells/domain) will not yield good parallel load balancing. To achieve reasonable load

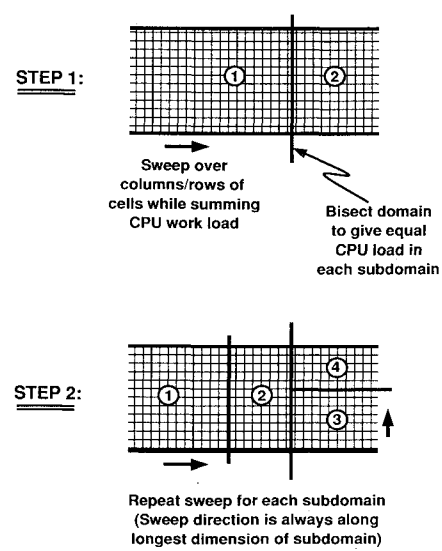


Fig. 4 Adaptive domain decomposition method.

balancing, an adaptive scheme was used to redistribute the cells among the processors (at some arbitrary time-step interval) based on successive binary bisections of the domain (and resulting subdomains). The adaptive decomposition method is shown schematically in Fig. 4. The domain (and subdomain) bisection is determined by sweeping through columns/rows of cells while summing over a quantity that provides a measure of the computational work. In the present study, the unit of measure was the actual CPU time required to perform the movement and collisions in each cell. The sweep is terminated at the column/row where the sum is one-half of the total work for that domain (or subdomain). The direction of the sweep is selected such that it is always in the direction of the longest dimension of the subdomain in order that a reasonable aspect ratio can be maintained.

Because of the computational overhead associated with the adaptive decomposition procedure, it is generally applied only at some arbitrary fixed time-step interval and then only during the transient portion of the simulation. Once a steady-state condition has been reached, variations in the computational work among processors occur due to statistical fluctuations that arise naturally in the simulation. However, it is generally more efficient to ignore these variations than to apply the adaptive load-balancing procedure during the steady-state portion of the simulation. In previous studies,[6] the procedure described here provided improvements of up to a factor of 2 in parallel efficiency and generally gave parallel speedups that were about 70–90% of the theoretical maximum. For the problems presented in the next section, it was not always practical to measure the speedup directly since the problems were too large to be run on a single processor. Therefore, the overall speedup and efficiency were calculated for each problem based on profile timings that were recorded for each portion of the simulation on each processor.

Results and Discussion

The results presented in this section are intended mainly to demonstrate the capabilities of the parallel DSMC method and in particular the efficiency of the adaptive domain decomposition method for load balancing on the iPSC/860 hypercube. Flowfield results are presented to show that the simulation gives results that are at least qualitatively consistent with more conventional DSMC methods. The gas is treated as single-species, nonreacting particles with no internal degrees of freedom. A simple hard-sphere model was used for gas-gas collisions, and a diffuse reflection model was used for gas-surface collisions. Flow parameters were nondimensionalized by appropriate freestream quantities and physical dimensions by the freestream mean free path. All calculations used cells that were of constant size throughout the flowfield. These limitations are not fundamental to the parallel method, and in principle, the parallel algorithm can be readily extended to treat more general problems with more realistic physical modeling.

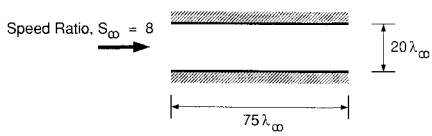


Fig. 5 Sketch of channel flow problem.

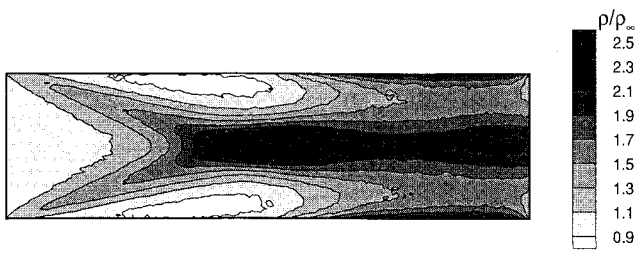


Fig. 6 Density contours for hypersonic channel flow.

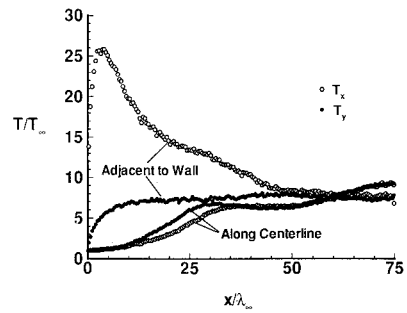


Fig. 7 Nonequilibrium temperature distributions for hypersonic channel flow.

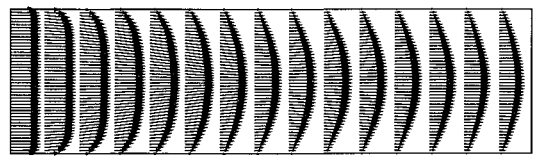


Fig. 8 Velocity vectors in hypersonic channel flow.

Hypersonic Channel Flow

The first problem presented is the flow in a two-dimensional channel. (See Fig. 5.) The channel has dimensions of 75 × 20 mean free paths, and the freestream speed ratio S_∞ was 8. (The Mach number is $S_\infty \cdot \sqrt{2/\gamma} = 8.6$, where γ is the ratio of specific heats and equals 5/3 for a monatomic gas.) The wall temperature was 5.4 times the freestream temperature. The number of cells was 180 × 48 in the x and y directions, respectively. Flowfield results are shown in Figs. 6–8. Gray-shaded density contours (see Fig. 6) show the shocks emanating from the leading edges of the channel quite clearly. The intersection of these shocks near the middle of the channel is also evident. Figure 7 shows the temperatures T_x and T_y associated with the thermal spread in velocities in the x and y directions, respectively, along the centerline and adjacent to the wall. The flow remains fairly close to thermal equilibrium along the centerline but shows a high degree of nonequilibrium adjacent to the wall near the entrance of the channel. The behavior near the wall is due to wall collisions that initially cause the flow to slow down and give rise to widely disparate velocities in the x direction. The flow achieves near thermal equilibrium at a distance of about 50 freestream mean-free-path lengths downstream from the entrance. Velocity vectors are shown in Fig. 8 and show a high degree of slip near the channel entrance, which is further evidence of the rarefied, nonequilibrium behavior. These results along with additional results presented in Ref. 6 are qualitatively similar to those obtained by Yasuhara et al.[9] over a similar range of conditions.

Flat-Plate Flow

A sketch of the computational arrangement for the flat-plate problem is given in Fig. 9. A plate of length L is placed in the computational region such that the left and lower boundaries are assumed to be inflow, and flow at the right and upper boundaries is assumed to exit into a vacuum. Calculations were performed for a freestream having a speed ratio of 3.16 and an angle of incidence to the plate of 18 deg. The plate was 20 λ_∞ long and the computational region was 40 × 40 λ_∞ with 96 × 96 cells. The temperature of the plate was equal to the freestream gas temperature. Flow results are shown in Figs. 10–13.

The density contours show the leading-edge bow shock and wake region quite clearly in Fig. 10. The density varies from

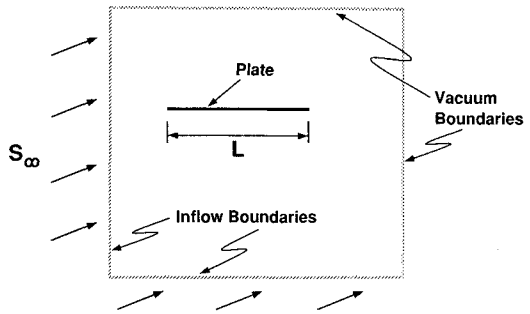


Fig. 9 Sketch of flat-plate problem.

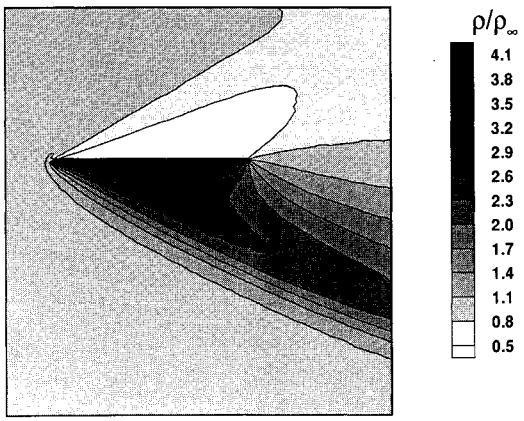
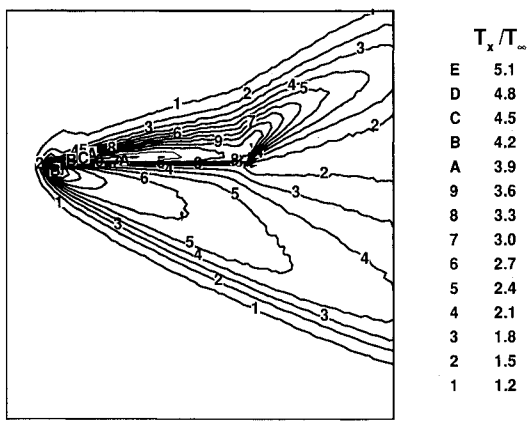
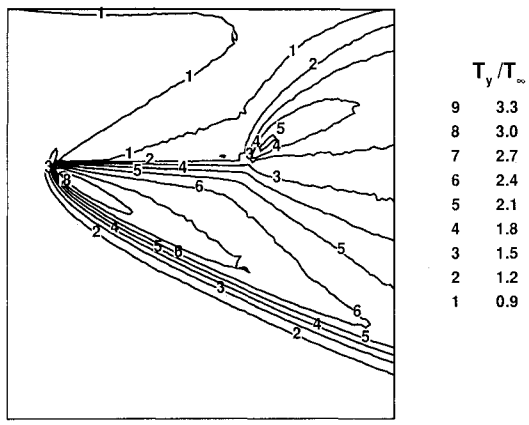


Fig. 10 Density contours about flat plate at angle of incidence of 18 deg.



a) T_x



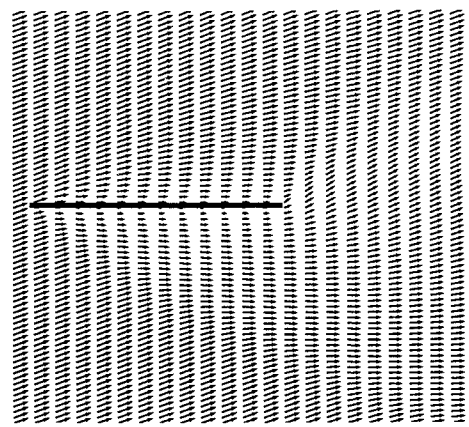
b) T_y

Fig. 11 Nonequilibrium temperature contours about flat plate.

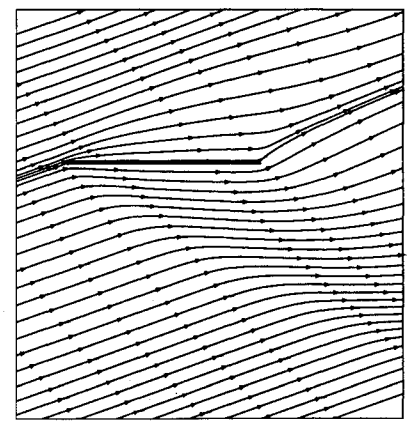
less than one-half the freestream density in the wake to over four times the freestream density near the leading edge on the windward side. The thermal nonequilibrium behavior is illustrated in Fig. 11 by the temperatures T_x and T_y, with T_x showing the largest overall variation. Although there are differences noted between T_x and T_y on the windward side, the most obvious differences occur on the leeside of the plate where the flow is much more rarefied and has a wide dispersion in x velocities. Mean velocity vectors and streamlines are shown in Fig. 12. (Because of the large number of cells, only about one-fifth of the vectors are shown.) The boundary layer and wake development as well as the turning of the flow through the bow shock are clearly evident. At the trailing edge of the plate, the windward flow expands rapidly into the lower density leeside flow, creating a plume-like streamline. Pressure and heat transfer rate distributions along the surface are shown in Fig. 13. On the windward side, both the pressure and heat transfer rates are at a maximum near the leading edge. The maximum values are slightly lower than the values computed for free-molecule flow for each quantity. Both the pressure and heat transfer decrease along the plate with the pressure dropping sharply near the trailing edge as one would expect.[10]

Parallel Decomposition and Load Balancing

The domain boundaries representing the parallel space decomposition for the two previous problems are shown in Fig. 14. The cell boundaries are not shown. However, because the cell size was constant, the area of each domain is representative of the number of cells it contained. For the channel problem, there is not a large variation in density and the domain size is relatively constant. For the flat plate, the density variations are much larger, and the domain sizes vary



a) Velocity vectors

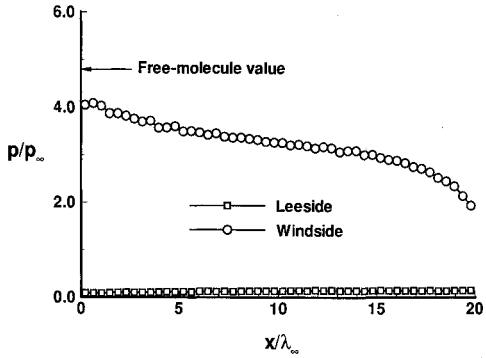


b) Streamlines

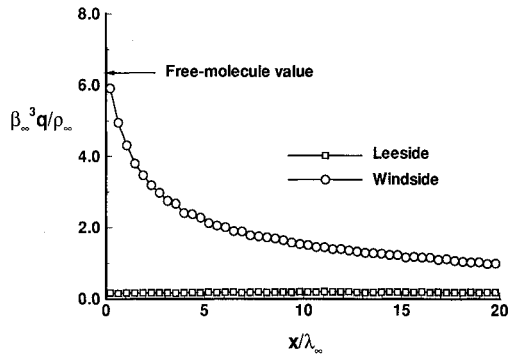
Fig. 12 Velocity vectors and streamlines about flat plate.

accordingly. (Since the domain decomposition was based on CPU time rather than directly on density, the correspondence is not exact.) Note also that the plate actually cuts across some of the domains demonstrating the versatility of having a description of the geometry and boundary conditions for the complete computational domain in each processor. Of course, for large problems, that would be impractical due to the limited memory available on each processor.

The distribution of CPU time among the processors is given in Fig. 15 for each problem. The portion of time attributed to the collision and movement part of the calculation is shown as the solid portion of each vertical bar, and the time for indexing, sampling, and communications is shown as the light shaded portion. (For the present algorithm, indexing time is at least twice as large as that for a nonparallel calculation since

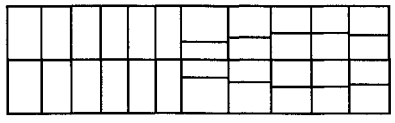


a) Pressure (nondimensionalized by freestream pressure p_∞)

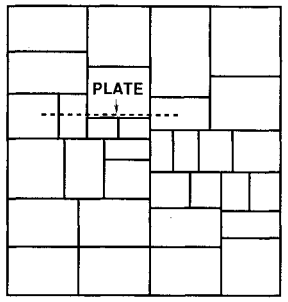


b) Heat transfer rate q (nondimensionalized by freestream density, ρ_∞ and most probable speed $1/\beta_\infty$)

Fig. 13 Pressure and heat-transfer rate on flat-plate surface.

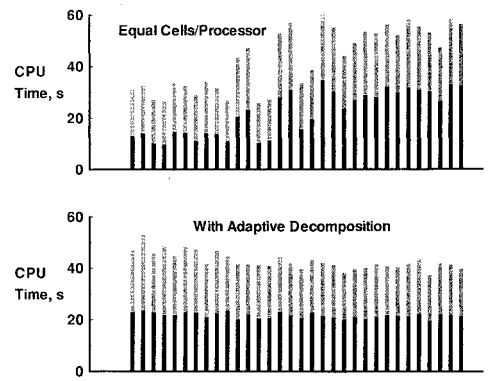


a) Channel flow problem

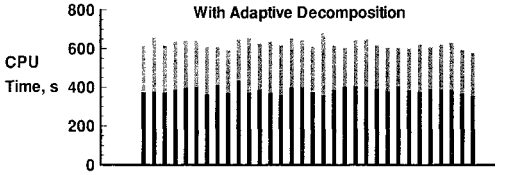


b) Flat-plate problem

Fig. 14 Domain boundaries used to partition the physical space for computation on 32 processors.



a) Channel flow problem



b) Flat-plate problem

Fig. 15 Distribution of CPU times among the individual processors. Collision-plus-movement time shown as solid portion of each vertical bar. Indexing, sampling, and communication time is shown as light shaded portion. (32 processors)

the indexing is performed before and after molecules are exchanged among processors.) For the channel problem, a distribution is also given for the case in which the domains had an equal number of cells (equal number of cells/processor). For the flat-plate problem, the load imbalance encountered using equal cells/processor was such that a solution could not be obtained without using adaption due to excessive communications.

Since the adaptive procedure is based only on the collision-plus-movement portion, it does a good job of balancing that part of the computational time as expected for both problems. For the channel problem, the maximum deviation in CPU time (max CPU time–min CPU time) was reduced from 85% of the average with equal cells to 37% of the average with adaptive decomposition. However, adaption produced a reduction of only about 10% in the total computational time (which includes all communication and synchronization times) from that obtained with equal cells/processor. The reason is that the remaining work (indexing, sampling, etc.) was not as evenly distributed as the collision and movement computations (see Fig. 15a). Furthermore, this problem was not that greatly imbalanced even with equal cells/processor, so that the largest reduction possible through adaption was only about 30%. On the other hand, the flat-plate problem produced much larger variations in density such that, without adaption, processors containing cells on the windward side of the plate would contain many more molecules than processors containing cells on the leeside of the plate. As stated previously, this imbalance was such that a solution could not be obtained when using equal cells/processor. However, using the adaptive procedure allowed the solution to proceed without difficulty and produced good overall load balancing (see Fig. 15b).

For the channel problem, the parallel speedup was about 20.4 using 32 processors, giving an efficiency of about 0.64. Using 16 processors, the speedup was about 11.6, giving an efficiency of 0.73. For the flat-plate problem, the speedup was 24.5 with an efficiency of 0.77 using 32 processors. McDonald[11] has implemented an adaptive load-balancing scheme in which preference is given to elongating the domains in the streamwise direction and has obtained an efficiency of 0.87 for a channel flow problem similar to that presented here. Although the binary bisection method used in the present study

provided reasonably good load balancing in most cases, it would appear that communications overhead might be further reduced by using alternate decomposition schemes. Further study of decomposition schemes is needed in order to provide more optimum efficiency for massively parallel processing.

Concluding Remarks

A method for implementing Bird's DSMC algorithm using parallel processing has been described, and the performance of the parallel method has been presented for simple benchmark problems. Applications of the code to simulations of the hypersonic, rarefied flow through a channel and about a flat plate at incidence have been presented.

The performance of the method for the benchmark problems demonstrates that parallel processing can clearly provide significant reductions in computational times for DSMC calculations. Spatial decomposition provides reasonable load balancing and allows good scaling behavior as the problem size is increased. In fact, the advantages of parallel processing over conventional scalar processing are greatest when the problem is very large.

Application of parallel methods to real hypersonic flow problems requires that the spatial decomposition be adapted to the flowfield in order to provide adequate load balancing. This is especially true for problems where there are large density variations in the flowfield. The adaptive decomposition scheme used in this study appeared to provide reasonable load balancing for the problems tested, although significant improvements in overall performance were not always evident. For the channel flow problem, the improvement was small; whereas for the flat-plate problem, adaption was necessary before a solution could even be obtained. However, in both cases, significant speedups over single processor computational rates were obtained.

The block decomposition of the DSMC computational space was relatively simple to implement on the iPSC/860. The majority of the DSMC code was unchanged from that used to solve the same problems on single processor machines. The changes were mainly in the form of added routines to send and collect molecule data at each time step and routines needed to perform the adaptive decomposition procedure. The movement, collision, and sampling routines were identical to those used for scalar processing. This approach has the advantage of allowing changes to the geometry, boundary conditions, and flow physics to be incorporated quite readily.

References

[1]Bird, G. A., "Monte Carlo Simulation in an Engineering Context," *Rarefied Gas Dynamics,* edited by S. S. Fisher, Vol. 74, Pt. 1, Progress in Astronautics and Aeronautics, AIAA, New York, 1981, pp. 235–239.

[2]Bird, G. A., "Application of the Direct Simulation Monte Carlo Method to the Full Shuttle Geometry," AIAA Paper 90-1692, June 1990.

[3]Furlani, T. R., and Lordi, J. A., "A Comparison of Parallel Algorithms for the Direct Simulation Monte Carlo Method II: Application to Exhaust Plume Flowfields," AIAA Paper 89-1167, June 1989.

[4]Goldstein, D., and Sturtevant, B., "Discrete Velocity Gasdynamics Simulations in a Parallel Processing Environment," AIAA Paper 89-1668, June 1989.

[5]Wilmoth, R. G., "Direct Simulation Monte Carlo Analysis on Parallel Processors," AIAA Paper 89-1666, June 1989.

[6]Wilmoth, R. G., "Adaptive Domain Decomposition for Monte Carlo Simulations on Parallel Processors," *Proceedings of the 17th International Symposium on Rarefied Gas Dynamics,* edited by A. E. Beylich, VCH Publishers, Weinheim, Germany, 1991, pp. 709–716.

[7]McDonald, J. D., "A Computationally Efficient Particle Simulation Method Suited to Vector Computer Architectures," Ph.D. Dissertation, Dept. of Aeronautics and Astronautics, Stanford Univ., Stanford, CA, Dec. 1989.

[8]Boyd, I. D., "Vectorization of a Monte Carlo Simulation Scheme for Nonequilibrium Gas Dynamics," *Journal of Computational Physics,* Vol. 96, No. 2, Oct. 1991, pp. 411–427.

[9]Yasuhara, M., Nakamura, Y., and Takanaka, J., "Monte Carlo Simulation of Flow into Channel with Sharp Leading Edge," *Rarefied Gas Dynamics: Theoretical and Computational Techniques,* edited by E. P. Muntz, D. P. Weaver, and D. H. Campbell, Vol. 118, Progress in Astronautics and Aeronautics, AIAA, Washington, DC, 1989, pp. 582–596.

[10]Dogra, V. K., Moss, J. N., and Price, J. M., "Rarefied Flow Past a Flat Plate at Incidence," *Rarefied Gas Dynamics: Theoretical and Computational Techniques,* edited by E. P. Muntz, D. P. Weaver, and D. H. Campbell, Vol. 118, Progress in Astronautics and Aeronautics, AIAA, Washington, DC, 1989, pp. 567–581.

[11]McDonald, J. D., "Particle Simulation in a Multiprocessor Environment," AIAA Paper 91-1366, June 1991.