Monte Carlo Particle Simulation of Low-Density Fluid Flow on MIMD Supercomputers

Steve Plimpton and Tim Bartel

Sandia National Laboratories, Albuquerque, NM 87185

Abstract

Direct Simulation Monte Carlo is a well-established technique for modeling low density fluid flows. The parallel implementation of a general simulation which allows for body-fitted grids, particle weighting, and a variety of surface and flow chemistry models is described. We compare its performance on a 1024-node nCUBE 2 to a serial version for the CRAY-YMP. Experiences with load-balancing the computation via graph-based heuristics (Kernighan and Lin) and the newer spectral techniques (Pothen, Simon, and Liou) are also discussed. This is a critical issue since density fluctuations can create orders-of-magnitude differences in computational loads as the simulation progresses.

1: Introduction

When modeling flow in a simulation geometry, the transport terms in the Navier-Stokes equations for continuum flow begin to break down at very low densities. The physical reason is that the difference equations used to discretize the PDE's are no longer valid when the distance particles move between collisions is comparable to typical length scales in the geometry. This is a characteristic of flows with a Knudsen number (ratio of mean-free path to length scale) on the order of 0.2. A commonly used method for modeling such flows is via particle simulation on a grid - a technique known as Direct Simulation Monte Carlo (DSMC), pioneered by Bird [1]. It has utility in aerospace applications (upper-atmosphere flows for "space"-planes, re-entry vehicles, rocket plumes, etc.) and in vacuum-related technologies for the semiconductor industry (modeling plasma etching or chemical vapor deposition).

* This work was partially supported by the Applied Mathematical Science Program, U. S. Department of Energy, Office of Energy Research, and was performed at Sandia National Laboratories which is operated for the DOE under contract number DE-AC04-76DP00789.

The key concept in DSMC is that particle motion is decoupled from particle collisions. During a timestep, particles flow through the grid independent of each other. At the end of the timestep the grid is used to locate particles that are close to each other and collisions between pairs of particles are performed via Monte Carlo techniques to produce a new velocity distribution. An attractive feature of DSMC is that complex gas and surface chemistry can be done straightforwardly as particles of different species collide with each other and with objects in the simulation geometry.

Large-scale DSMC computations on current supercomputers can easily be in the 10^6 particle, 10^4 grid cell, and thousands of timestep range. Researchers have made efforts to speed these computations for vector supercomputers [2,3] as well as to implement them on parallel MIMD supercomputers [4,5]. Reference [5] is particularly notable because it showed scalable speed-up can be obtained on a parallel machine and incorporated a dynamic load-balancing scheme whereby grid cells are reapportioned between processors. All of these optimization and parallelization methods were for DSMC done on a regular mesh which greatly simplifies the problem of moving and finding particles on the grid and (for the parallel case) also simplifies the dynamic load balancing problem.

In this work, we describe our parallel implementation of a more general kind of DSMC that uses a more complex gridding. It is a good match for MIMD parallel computers because many of the computations associated with finding particles on the complex grid and doing Monte Carlo chemistry and collisions do not vectorize well. In the next sections we discuss the features of our DSMC model, its performance on the nCUBE 2 parallel supercomputer vs. the CRAY-YMP, and the issues of static and dynamic load-balancing using different grid-partitioning algorithms.

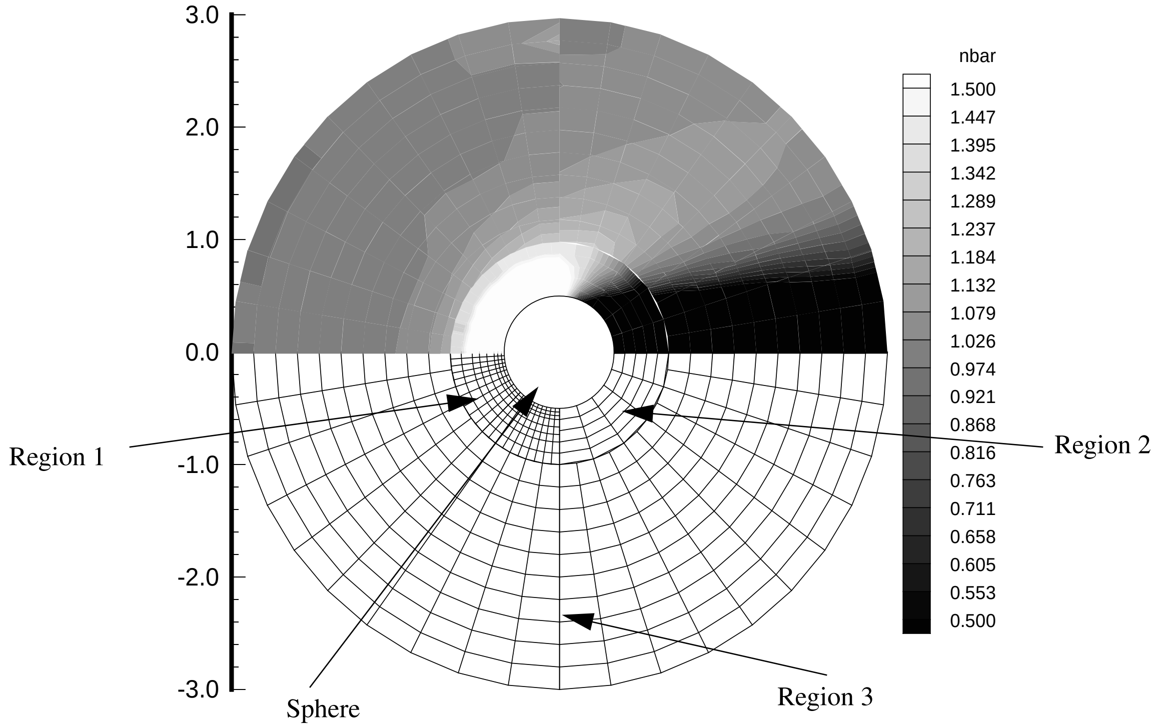

Figure 1: Gridding and flow pattern around a sphere (simulated as a 2-D axisymmetric problem) for 2 species (O_2 and N_2) at a freestream velocity of 6.7 km/sec and freestream number density of 1.42×10^{18} #/m^3. The flow enters from the left. The lower half of the figure is the computational grid used by the DSMC computation, blocked into 3 different regions. The upper half is the steady-state flow field (contour plot of local number density normalized to the freestream) around the sphere. While the gray-scaling obscures the details of the simulation data, the wake behind the sphere and density build-up in front of it are visible.

2: DSMC with Body-Fitted Grids

Our flavor of DSMC employs body-fitted, multi-region grids such as that illustrated in the bottom half of Figure 1. The grid is composed of adjoined regions. Each region is constructed of curved and/or straight lines of variable spacing which may be body-fitted if the edge of the region is the surface of an object (such as the sphere in the figure). There is no continuity constraint for cells going from one region into an adjacent region (e.g. regions 1 and 3 in figure). The goal with a more general grid such as this is to put properly sized cells at computational “hot” spots (e.g. to the left of the sphere in the figure) and to use appropriate weighting factors to minimize the number of particles and cells that must be computed on. Also, body-fitted grids allow complex physical geometries to be accurately modeled. The trade-off is the additional computational complexity of finding particle positions on the grid in order to do collisions, interact with surfaces, etc.

Additionally, we apply region weighting factors so that particles can be created and destroyed in appropriate areas where more or fewer particles are needed to provide adequate collision statistics. An example of the general kind of particle move that must be tracked during a timestep is shown in Figure 2. A particle might travel this far in a single timestep if a Monte Carlo collision imparted a high velocity to it (i.e. on the tail of a distribution function). Such complex motion along with the coding for multi-species chemistry reactions (recombinations and dissociations) within cells creates computational difficulties for a vectorized or SIMD implementation.

By contrast, these events are handled straightforwardly on a MIMD parallel machine where each processor owns the data for a few cells and all the particles in them. This is also a decomposition which naturally scales to large numbers of grid cells (and particles) and processors. If each processor can store the geometric definition of the entire grid (slopes of lines, coefficients of curves, etc. --- a relatively small amount of information) then it can track the collisionless motion of its particles independent of the other processors. Additionally, the chemistry and collision calculations within each cell are completely parallelized.

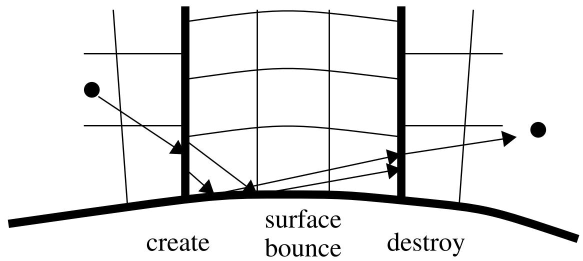

Figure 2: An unlikely, but possible, one-timestep event in a DSMC simulation. The particle's trajectory includes motion across a semi-irregular grid, crossing of region boundaries where the particle may be cloned or annihilated according to weighting rules, and a computed bounce off a reflective surface.

The communication that needs to take place at each timestep is to pass particles that have moved to new cells to the processors that own them. This is necessarily irregular communication because of the nature of the grid, but is also handled straightforwardly within the typical non-blocking MIMD message-passing model.

3: Results

We have implemented a 2-D (with axisymmetry) DSMC model that has the generality outlined in the previous section on the nCUBE 2 parallel computer and benchmarked it on several problems. The timing results for two of the calculations are listed in Table 1. Problem A is the small flow-around-a-sphere problem shown in Figure 1 (with a denser grid) for which an analytic solution is also known. Problem B is a medium-sized problem (80,000 particles) for flow around a re-entry vehicle.

Timings for the CRAY-YMP (single processor) as compared to i860 or workstation timings show how poorly this kind of DMSC vectorizes. (It is worth noting that this YMP timing is for our new "single-node" parallel code which is 2x faster than the original YMP version).

The nCUBE results show that even for a modest number of cells good performance vs. the CRAY is achieved ----- 5-10x the YMP (single processor) for small and medium-sized problems. The timings show a loss of parallel efficiency for large numbers of processors because we have too few cells per processor for these small problems. For the larger problems we plan to run (10^4 cells, 10^6 particles) we expect good parallel scaling and peak performances on the full 1024-node nCUBE of ~20x the YMP (single processor).

	Problem A	Problem B
cells	1040	2797
particles	30,000	80,000
CRAY-YMP	50	102
i860	179	367
SPARC 2	170	347
nCUBE 4	221.	---
8	116.	---
16	71.4	138.
32	42.3	83.5
64	23.1	46.9
128	13.1	26.7
256	9.42	15.5
512	8.89	10.8

Table 1: Timings (sec) for 100 timesteps of the DMSC simulation on 2 benchmark problems. Timings are given for 3 single-processors (CRAY-YMP, i860, and workstation) and for a parallel implementation on varying numbers of nCUBE 2 processors.

4: Load-Balancing

The end goal of a DSMC simulation is typically a steady-state flow which may have density variations of many orders of magnitude. (The sphere problem in Figure 1 has a 3x density variation). Additionally, in the transient phase of the simulation, density fluctuations of particles on the grid can be very large. Also, it is often difficult to predict *a priori* which cells will have a high computational load even for the steady-state regime. These features combine to produce a serious load-balancing problem for a parallel implementation if high efficiency is desired.

For our cell-wise decomposition, there are several choices to address this load-balancing issue. We have had the best success thus far with a scattered decomposition scheme, where each processor is assigned an equal number cells (at the beginning of the simulation) which are scattered across the simulation domain. On average, each processor will own some cells with many particles and some with few, so that a rough load balancing is achieved, even for a dynamically changing balance of work among the cells.

In particular, a scattered decomposition has proven to work better (by a factor of 1.5-2x) than schemes that allocate a 2-d "block" of cells to a processor. This is because a "block" to the right (wake region) of the sphere in Figure 1 will have much less work to do than a "block" to the left. The drawback of the scattered decomposition method is that it maximizes communication since the processor will not typically own any of the cells adjacent to one of its owned cells and hence many particles will have to be exchanged at every timestep.

By contrast, the advantage of a load-balancing scheme that assigns "blocks" of cells to each processor is that it does, in fact, minimize the communication for this simulation (by minimizing the surface/volume ratio). Examples of such load-balancers that we have tried include the heuristic Kernighan/Lin graph bisectioner [6] and the newly developed spectral techniques [7,8,9]. The spectral methods, which partition the cells by finding an eigenvalue of a matrix associated with the mesh topology, are particularly attractive for this problem because they can be readily parallelized (because the computational kernel in the Lanczos eigenvalue calculation is a sparse matrix-vector multiply). However, we have not yet been able to use the spectral technique for the DSMC problem because it does not provide for a "weighted" decomposition (by particle numbers or computational load in a cell) so that a smaller "block" of computationally-intense cells can be given to some processors and a larger "block" of few-particle cells to others. Research on such an enhancement to the spectral technique is being worked on at Sandia [10] and we plan to eventually use it in our DSMC simulation to achieve both a computational load-balance and to minimize the communication of particles between processors.

One final note worth mentioning is that even with a properly weighted decomposition, it will be necessary to periodically update it as particle densities change and the computational load varies from region to region within the simulation domain. The cell-wise decomposition outlined in section 2 provides a means to do this dynamic load-balancing. This is because the natural unit of exchange between processors (during a load-balancing adjustment of cell assignments to processors) will be a cell and its associated particles.

5: Summary

We have described a DSMC model which uses general body-fitted irregular grids and particle weighting factors to minimize the numbers of cells and particles that must be used to simulate a particular physical effect. MIMD parallel computers provide the flexibility needed to efficiently parallelize such a code and allow for each processor to independently compute trajectories and randomized collisions and chemistry interactions for its particles. The MIMD paradigm is critical because the computations can be completely different for each particle.

Load-balancing of DSMC models is a serious issue. We have had the best success thus far by using a scattered decomposition which balances the computational load adequately, but does not minimize the inter-processor communication. We hope that enhanced spectral domain decomposition methods which allow for weighted decompositions will also reduce communication costs.

Acknowledgments

We thank Bruce Hendrickson and Rob Leland at Sandia for many useful discussions concerning load-balancing algorithms and strategies.

References

[1] G. A. Bird, *Molecular Gas Dynamics*, Clarendon Press, Oxford (1976).

[2] D. Baganoff and J. D. McDonald, "A Collision-Selection Rule for a Particle Simulation Method Suited to Vector Computers", Phys Fluids A, 2, (1990) 1248.

[3] G. Prisco, "Optimization of DMSC Codes for Vector Processing", J of Comp Phys, 94, (1991) 454.

[4] T. R. Furlani and J. A. Lordi, "Implementation of the DSMC Method for an Exhaust Plume Flowfield in a Parallel Computing Environment", Computers & Fluids, 18, (1990) 217.

[5] J. D. McDonald, "Particle Simulation in a Mulitprocessor Environment", proceedings of AIAA 26th Thermophysics Conference, published by American Institute of Aeronautics and Astronautics, AIAA 91-1366 (1991).

[6] B. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", Bell System Technical Journal, 29, (1970) 291.

[7] R. Bopana, "Eigenvalues and Graph Bisection: An Average Case Analysis", proceedings of 28th Annual Symposium on Foundations of Computer Science", published by IEEE, (1987), p. 280.

[8] A. Pothen and H. Simon and K. Liou, "Partitioning Sparse Matrices with Eigenvectors of Graphs", SIAM J. Matrix Anal, 11, (July 1980) 430.

[9] H. Simon, "Partitioning of Unstructured Problems for Parallel Processing", proceedings of Conference on Parallel Methods on Large Scale Structural Analysis and Physics Applications, published by Pergammon Press, 1991.

[10] B. Hendrickson and R. Leland, "Spectral Load Balancing for Hypercubes", in preparation.