Implementation of a Hypersonic Rarefied Flow Particle Simulation on the Connection Machine

Leonardo Dagum

December, 1988

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 88.46

NASA Cooperative Agreement Number NCC 2-387

RIACS

Research Institute for Advanced Computer Science

Implementation of a Hypersonic Rarefied Flow Particle Simulation on the Connection Machine

*Leonardo Dagum**

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 88.46
December, 1988

A very efficient direct particle simulation algorithm for hypersonic rarefied flows is presented and its implementation on a Connection Machine is described. The implementation simulates ideal diatomic Maxwell molecules with three translational and two rotational degrees of freedom. Results for a 2D simulation of supersonic flow over a 30^{o} wedge are presented and used for validation.

Keywords: Connection Machine, particle simulation, hypersonic, rarefied flow.

Work reported herein was supported by Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA).

*Department of Computer Science, Stanford, Stanford, California. Work performed as a summer visitor at RIACS and at Stanford University.

Introduction

Of increasing interest to NASA and the fluid mechanics community in general has been the development of accurate and efficient methods to treat hypersonic rarefied flow problems. Hypersonic flows are characterised by large Mach numbers ($M > 5$) and rarefied flows are characterised by large Knudsen numbers ($K_n > 0.1$). These conditions are encountered by flight vehicles operating in the upper atmosphere (altitude 50–150 km) and are of consequence in the design of future vehicles such as the National Aerospace Plane (NASP) and Aero-Assisted Orbital Transfer Vehicles (AOTV's). The standard method for solving hypersonic rarefied flow problems is through direct particle simulation methods[1–5],

however the huge computational capacity required to solve even a modest sized problem of practical interest has severely restricted their use.

The present paper outlines a very efficient direct particle simulation algorithm developed at Stanford University by Donald Baganoff and his students[6] and proceeds to describe a fine-grained parallelized implementation of this algorithm on a Thinking Machines Connection Machine Model 2 (CM2).

Description of Algorithm

For a small discrete time step, the molecular motion and collision terms of the Boltzmann equation may be decoupled[1]. This allows the simulated particle flow to be considered in terms of two consecutive but distinct events in one time step, specifically there is a collisionles motion of all particles followed by a motionless collision of those particles which have been identified as colliding partners. The collisionless motion of particles is strictly deterministic and reversible. However, the collision of particles is treated on a probabilistic basis. This is the characteristic feature of simulation methods which distinguish them from the methods of molecular dynamics.

The state of the system is updated on a per time step basis. A single time step is comprised of four sub-steps:

1) collisionless motion of particles
2) enforcement of boundary conditions
3) selection of collision partners
4) collision of selected collision partners

The following sections will briefly consider these sub-step in general and then consider in detail their fine grain parallel implementation on a Connection Machine. For greater general detail and a vectorized implementation see McDonald and Baganoff[6].

Particle Motion

Each particle i has a position vector $\vec{x}_i$ and a translational velocity vector $\vec{u}_i$. On each time step, every particle's position vector is updated simply by

$$\vec{x}_i \longleftarrow \vec{x}_i + \Delta t \cdot \vec{u}_i. \tag{1}$$

By using a time scale normalised by a time step, this simplifies to

$$\vec{x}_i \longleftarrow \vec{x}_i + \vec{u}_i. \tag{2}$$

Boundary Conditions

If the aim is to solve for the flow around some aerodynamic body, it is usual to set up physical space to simulate a wind tunnel. In such a set up boundaries can be of two types which here will be called "hard" boundaries and "soft" boundaries. Hard boundaries consist of solid impermeable barriers, specifically the walls of the wind tunnel and the object in the test section. These are most easily implemented as inviscid boundaries although the no slip condition with an isothermal or adiabatic wall represents the more physical situation. To simulate inviscid boundaries the particles are specularly reflected from surfaces; this sort of boundary allows the direct comparison of simulation results with 2D inviscid theoretical results thereby providing an important check in the development of new code.

Soft boundaries delimit regions where particles pass into sources or sinks. The downstream boundary of the wind tunnel is an example of a sink, all particles exiting downstream are removed from the simulation. For physical consistency this constrains the downstream boundary to be supersonic.

The upstream boundary may be implemented either as a soft or hard boundary, the choice depends on the target architecture. As a soft boundary the upstream region acts as a source of particles at the freestream conditions. The strength of this source has to be controlled to maintain a constant freestream density. This is a useful implementation on vector or serial architectures where the spawning of new particles is simple and efficient due to the global data structures these architectures best support[6]. On parallel architectures it is useful to implement a hard boundary in the upstream region. This boundary acts as a plunger, moving with the freestream until it crosses a predefined trigger point which causes the plunger to be withdrawn and enough new particles to be introduced to fill the void. In this manner the introduction of new particles can be delayed an arbitrary number of time steps.

Selection of Collision Partners

The selection of collision partners is made by considering the interactive potential of collision candidates. It is important to distinguish between candidates for collision and actual partners in a collision. To identify collision candidates in an efficient manner it is necessary to introduce a grid of cells associated with discrete regions in the simulated space. Since particles occupying the same cell are neighbouring particles in physical space, these then are considered collision candidates.

McDonald and Baganoff[6] argue for small, geometrically simple and similar cells and such are implemented here. This leads to a rectangular grid (in two dimensions) of square cells of unit normal width.

With the set of collision candidates identified, it is necessary to select suitable collision partners. The most common approach is that used in Bird's Monte Carlo method[1-4] where pairs of molecules within a cell are randomly chosen and collided until the asynchronous

cell time exceeds the global simulation time. Pryor and Burns[7] describe a vectorized implementation of this method but clearly it suffers a strong dependence on the number of cells in the simulation. At best this method can be parallelized only at the cell level and thus is strongly influenced by statistical fluctuations in the cell populations. Nanbu[8] introduces the idea of a probability of collision which he applies unconditionally to decide on a collision and then on a conditional basis to select a collision partner. This approach has a better theoretical foundation however it has the drawback of being an $O(N^2)$ calculation. Ploss[9] shows how Nanbu's scheme can be implemented as $O(N)$ and vectorized thus yielding performance comparable to Bird's scheme. However, both Ploss's and Nanbu's scheme conserve only the mean energy and momentum of a cell and their extension to reacting flows is questionable.

McDonald and Baganoff[6] introduce a selection rule based on collision probability and which allows a fine grained parallelization while also conserving energy and momentum in a collision. In this approach, a probability of collision is computed for each pair of collision candidates and collisions are carried out in accordance with this probability. The decision to perform a collision is applied on the individual candidate pairs and not on the cell as a whole. Consequently, like Ploss's scheme, the selection rule can be parallelized at a particle level.

The time counter approach uses a mean time for collision given by

$$t_c \sim (n\sigma\bar{c})^{-1}, \tag{3}$$

where n is the local number density, σ is the collision cross section and $\bar{c}$ is the mean molecular speed. McDonald and Baganoff derive a probability of collision

$$P_c = \Delta t / t_c \tag{4}$$

valid only if Δt, the time step, is at least 3 or 4 times smaller than the mean value of t_c. This constraint ensures that the probability of more than one collision in a single time step

is negligibly small. Combining (3) and (4) yields

$$P_c \sim \Delta t n \sigma \bar{c} \tag{5}$$

which is the most general form of the selection rule. For power law molecules, (5) becomes

$$P_c \sim n g^{1-4/\alpha} \tag{6}$$

where g is the relative velocity of collision and α is the exponent for the inverse power law being used. It is useful to scale P_c to the desired upstream mean free path, such that

$$P_c/P_\infty = (n/n_\infty)(g/g_\infty)^{1-4/\alpha}, \tag{7}$$

where the subscript ∞ indicates freestream conditions. Finally, for the special case of a Maxwell molecule, $\alpha = 4$ and (7) reduces to

$$P_c = P_\infty (n/n_\infty). \tag{8}$$

The Collision Algorithm

The algorithm presented here is that developed by McDonald and Baganoff[6] and considers collisions between perfect diatomic molecules. The outcome of a collision of two particles is, for each particle, a new velocity and internal energy subject to the constraints of conservation of linear momentum and energy. In this model, rotational energy is accounted for by a rotational velocity vector $\vec{r}$ such that

$$E_{rot} = \frac{1}{2} m(\vec{r} \cdot \vec{r}). \tag{9}$$

For a diatomic gas, $\vec{r}$ has two components (two degrees of freedom in rotation) and the translational velocity $\vec{u}$ has three components (three degrees of freedom in translation). Conservation of energy can then be written as

$$E_{tot} = E_{tot'} \tag{10}$$

or

$$|\vec{u}_{rel}|^2 + |\vec{r}_{rel}|^2 + |\vec{u}_{mean}|^2 + |\vec{r}_{mean}|^2 = |\vec{u}_{rel'}|^2 + |\vec{r}_{rel'}|^2 + |\vec{u}_{mean'}|^2 + |\vec{r}_{mean'}|^2, \quad (11)$$

where

$$\vec{u}_{rel} = \frac{\vec{u}_i - \vec{u}_j}{2} \quad (12)$$

$$\vec{r}_{rel} = \frac{\vec{r}_i - \vec{r}_j}{2} \quad (13)$$

$$\vec{u}_{mean} = \frac{\vec{u}_i + \vec{u}_j}{2} \quad (14)$$

$$\vec{r}_{mean} = \frac{\vec{r}_i + \vec{r}_j}{2} \quad (15)$$

and the prime indicates a post-collision value (these equations correspond to eqs. 16–21 of reference 6). Conservation of linear momentum can be written as

$$\vec{u}_{mean} = \vec{u}_{mean'}. \quad (16)$$

Then, by assuming

$$\vec{r}_{mean} = \vec{r}_{mean'} \quad (17)$$

the two conservation equations can be combined as a single equation

$$|\vec{u}_{rel}|^2 + |\vec{r}_{rel}|^2 = |\vec{u}_{rel'}|^2 + |\vec{u}_{rel'}|^2. \quad (18)$$

Equation (18) forms the basis of the collision algorithm. One begins by computing the relative and mean pre-collision velocity components for each collision partner. It is important to note that any post-collision values that satisfy (18) are valid. Computationally, the simplest way to arrive at five values that satisfy (18) is to use the same pre-collision values calculated by eqs. (12) and (13). By re-ordering these values in a random fashion and assigning each element a random, equally-probable sign, one arrives at a valid and completely new post-collision relative velocity vector. The post-collision velocity vector for the particles is now easily obtained. For the first particle the new relative velocity is

added to the mean velocity and for the second particle the relative velocity is subtracted from the mean velocity.

IMPLEMENTATION ON THE CONNECTION MACHINE

Data Structure — Processor Mapping

A key issue in the implementation of a particle simulation on the Connection Machine is the mapping of data to processors. Two approaches may be taken—one can map computational cells to individual processors or one can map individual particles to individual processors. Considering the cells-to-processors mapping first, it is clear that this mapping will suffer from inefficient communication and poor load balancing. Communication between processors will occur when particles exit one cell to enter another. In order to avoid conflicts, a cell must only communicate with a single neighbour at a time. In two dimensions this implies eight distinct communication events with only one eighth of the processors active in any single event. The situation is considerably worse in three dimensions where a cell must communicate with twenty-six neighbours.

The load balancing associated with this mapping displays both inefficient hardware utilization and wasteful memory management. Not only are computations slowed to the rate of the most populated cell, but also the memory assigned to each processor must be great enough to accomodate the highest density of particles encountered in the simulation. Consequently, throughout most of the calculation a great number of processors will be idle with large parts of their memory unused.

These inefficiencies can be completely eliminated by choosing a particles-to-processors mapping. On a heuristic basis, one need only consider that the finest grained parallelism of a particle simulation exists at the particle level whence the particles-to-processors mapping

ought to be most compatible with this architecture. Further justification for this mapping will be given as the implementation is outlined in detail. However, at this point it is important to note that the Connection Machine's support of "virtual processors"[10], i.e. creating the impression of two or more processors from a single physical processor, allows for a great deal of generality in the processor mapping. Consequently, there is no limit on problem size with the chosen mapping other than that of the total memory capacity of the machine.

In further discussing the present implementation of a particle simulation, it is useful to make the distinction between the "physical" state of a particle and the "computational" state of a particle. For the perfect diatomic gas molecules of the model, the physical state of a particle is completely defined by its position and its translational and rotational velocities, i.e. $\vec{x}_i, \vec{u}_i, \vec{r}_i$. In two dimensions this representation requires seven distinct values. The computational state of a particle includes as a subset the physical state, but adds information to this that makes the computation of a new physical state more efficient. The added information in the computational state can be completely derived from the physical state information or created independently. Specifically, the computational state of a particle adds to the physical state the cell index and a five element permutation vector (or permutation sequence). The cell index is a distinct index value that identifies the cell occupied by the particle. The permutation vector is a permutation of five numbers (0 through 4) used in the collision routine to re-order the relative velocity components.

Particle Motion and Boundary Interaction

The implementation of particle motion in the particles-to-processors mapping is very straightforward and perfectly load balanced. All particles simply add their velocity components to the appropriate position co-ordinate. All processors are active for this event.

In the present implementation the only geometry supported is an inclined flat plate. Particles requiring boundary interactions are identified by their position and this selected set of particles perform the appropriate action. Those particles exiting through the soft downstream boundary are removed from the physical space of the simulation and put in a separate reservoir. These particles are given velocities from a rectangular distribution with the same variance as the freestream, therefore after a few time steps collisions with other reservoir particles relaxes these to the correct Gaussian distributions. When new particles need to be introduced at the upstream boundary they are taken from this reservoir.

The reservoir serves several purposes in this manner. With the particles-to-processors mapping, particles which are not used directly in the simulation, as in the start up transient phase of the wind tunnel simulation, represent an inefficiency in the form of idle processing power. Putting these particles in a separate reservoir and letting them collide amongst themselves is a way to get useful work from these otherwise idle processors. Without the reservoir, new particles would have to be initialised with freestream conditions and this would involve sampling directly from a Gaussian distribution which involves either costly calls to transcendental functions or repeated calls to a random number generator. Neither of these two options is as satisfying as simply picking up particles from a reservoir.

Selection of Collision Partners

Once the particles have been moved and all the boundary conditions enforced, each particle computes its occupying cell index. In order to identify collision candidates it is necessary to access all particles occupying the same cell. Before this can be done in an efficient manner, it is necessary to sort the particles by order of cell index. It should be noted that sorts are very efficiently implemented on the Connection Machine[11] and do not incur the large computational cost usually associated with sorts on sequential machines.

The sort is a crucial step in the implementation of this particle simulation algorithm. It introduces an overhead not present in the vectorized implementation[6] but proves very rewarding for the rest of the algorithm. The primary purpose of the sort is to put all particles occupying a given cell into neighbouring addresses thus making it easy both to identify collision candidates and to sample macroscopic quantities from cells. Although this is the primary effect, the consequences of the sort are more subtle. Since each particle is assigned to a virtual processor, one can think in terms of a fixed amount of processing power per particle and in these terms the sort achieves a perfect dynamic load balance for the collision routine. In other words, the total processing power of the machine is evenly distributed amongst the computational cells of the simulation.

The collision of particles is the most computationally intense part of the calculation, and achieving a perfect load balance here is crucial to the performance of the algorithm. Being able to make full simultaneous use of all the Connection Machine processors is, for this architecture, the equivalent of being able to use vector pipelines in vector machines. Therefore one could say that the algorithm is "vectorized" on the Connection Machine.

A further requirement of the sort is to change the order of particles within a cell between time steps. This is necessary because collision candidates are identified on an "even/odd" basis, i.e. all even numbered partners within a cell are eligible for collision with their odd numbered neighbour. This, in conjunction with the use of virtual processors, proves to be a very efficient arrangement because collision candidates are now guaranteed to be in the same physical processor, hence communication time is minimized for the collision. However, it is important that candidate partners change between time steps otherwise the situation arises where the same partners collide repeatedly leading to correlated velocity distributions. To obtain this additional randomization, the cell index of a particle is scaled by some constant factor and, before sorting, a random number less than the scale factor is added to it. Now sorting the particles no longer preserves the relative ordering within a cell and there is confidence in the statistical randomness of the collision candidate pairs.

Collision partners are selected from the candidate pairs by applying the selection rule given by (8). This requires specific knowledge of the cell density which can be best obtained on the Connection Machine by making use of the scan functions[10].

Collision of Particles

The collision of particles proceeds in the manner prescribed by the collision algorithm. The essential issue that needs to be addressed in the implementation is that of re-ordering the relative velocity components to arrive at the post-collision state. On the Connection Machine this is done by using the permutation vector which is part of the computational state of the particles. Of the two available permutation vectors, which one gets used is inconsequential, however to maintain statistically random collision outcomes it is desirable for particles to have different permutation vectors in succeeding time steps. The standard algorithm for creating random permutations is given by Knuth[12] and an adaptation of this is implemented here. The approach taken is to initialise the particles with random permutations (taken from a table stored on the front end computer) and generate new permutations by performing random transpositions on the existing permutation. By a random transposition is meant the operation of arbitrarily switching the order of two randomly selected elements in the permutation sequence. Consider a permutation $\vec{p}$ with n elements. If p_j is the j^{th} element of $\vec{p}$ then transposition of the j^{th} element with the first element produces the new permutation $\vec{p}\,'$.

Aldous and Dioconis[13] prove that $n\log n$ transpositions of this type are required to generate a new, statistically uncorrelated permutation. In the present implementation, for each collision a single random transposition of a particle's permutation vector is performed. It follows that 10 collisions are required before a particle has a completely new permutation vector. However the collision algorithm is only loosely bound to the randomness of the

permutation since randomization of the outcome is enhanced by random partner selection, consequently a single transposition per collision is found sufficient to ensure unbiased outcomes.

Specific Implementation Issues

In order to obtain the best performance of this algorithm on the CM2, some very specific optimizations were implemented. Individual Connection Machine processors are bit serial and therefore most suited for integer computations. Although floating point computations are supported in hardware, a floating point implementation loses the power of bit addressability and much of the versatility afforded to integer calculations by the machine. These considerations led to an integer implementation of the simulation. In this implementation the physical state of a particle is stored in a 32 bit fixed point format with 23 bits for precision. This compares favourably with the IEEE floating point standard which employs a 23 bit mantissa, however it now becomes neccessary to be aware of the effect of truncation and to perform some rounding where required. The IEEE standard employs three extra bits (the "guard" and "sticky" bits) to correctly round off results from operations such as division. In a fixed point format the result of division of two numbers is truncated if the number of bits required to correctly represent it is greater than the number of bits allocated in the format. This becomes a problem in the collision of particles when the relative and mean velocity components are computed (eqs. (12)–(15)); the consistent truncation after division by 2 can lead to a significant loss in total energy in stagnation regions of the flow. The problem is solved by arbitrarily adding with uniform probability either 0 or 1 to the result of this division, in a statistical sense this achieves the correct rounding. Despite the extra computation required for this correction, there is a marked improvement in performance with an integer implementation.

An addiditonal advantage of this implementation is the availability of a quick but dirty random number in the low order bits of a physical state quantity. This provides a random number of limited size and unspecified ditribution but finds use in low impact situations. Specifically, it is used during the sort to enhance mixing, and in the collision routine to choose a random transposition, to choose a random sign, and to correct the truncation error in the manner described above.

Results

To verify the validity of the code, the near-continuum flow over a 2D wedge was simulated and results were compared with the 2D theoretical results. Near continuum flow is simulated by setting the mean free path in the free stream to be zero. As a consequence, all collision candidates must collide and the number of collisions in a cell is just equal to half the number of particles in the cell. Presented in figures 1–3 is the density distribution for Mach 4 flow over a 30° wedge. A total of 512k particles were employed in this solution with 460000 particles actually in the flow and another 45000 particles in the reservoir. The grid had dimensions 98X64; the wedge was place 20 cells from the upstream boundary and was 25 cells wide at the base. The simulation was run for 1200 time steps to reach steady state and then time averaged for a further 2000 timesteps to generate the solution.

Figure 1 shows the density contours in the solution. The theoretical shock angle for this flow is 45° and the solution matches this exactly. Furthermore, from the Rankine–Hugoniot relations we expect the density behind the shock to be 3.7 times the freestream value, this again is reflected in the solution. The Prandtl–Meyer expansion fan around the corner of the wedge was also compared to theory and found to be correct. The shock thickness can be measured from figure 1 and is equal to 3 cell widths.

Figure 2 shows a perspective view of the density surface. This figure clearly depicts

the fully developed wake shock created when the fluid which has expanded around the corner of the wedge meets the bottom surface of the wind tunnel. Evidence of this can also be seen in figure 1. Figure 3 presents an expanded view of this surface in the stagnation region by the wedge. This figure is useful for studying the approach that the simulation takes to the theoretical rise in density behind the shock. The jagged edge in the figure represents the wedge surface. The wedge surface is smooth however the grid is rectangular, and where cells are divided by the wedge special allowance must be made for the fractional cell volume when employing the selection rule (equation (8)) and in computing the time average cell density. The plotting package used for generating these surfaces did not allow the same special consideration for fractional cell volumes, whence the jagged edge.

To examine the ability of the method to properly simulate rarefied flows, the same simulation was run but with the mean free path adjusted to be 0.5 cell widths in the freestream. The molecular model is for perfect diatomic Maxwell molecules. The wedge is 25 cell widths in length hence the flow has Knudsen number 0.02 and Reynolds number 600. The results from this simulation are presented in figures 4–6. Figure 4 depicts the density contours using the same intervals employed in figure 1. The shock width in this solution is measured to be 5 cell widths. As expected, the shock in the rarefied flow is wider than in the near-continuum case. This is characteristic of rarefied flows and consistent with the greater mean free path and Knudsen number in this simulation. On looking at figure 5 it is at first surprising to notice there is no longer a wake shock, however this is merely another manifestation of the greater rarefaction or higher Knudsen number. The wake region is highly rarefied and the mean free path in this region is great enough that the wake shock is completely washed out. Figure 6 is an expanded view of the stagnation region by the wedge. Comparing this with figure 3 provides a more visual understanding of the effect flow rarefaction has made on the shock.

These results in addition to other results from simulations at differing Mach numbers and wedge angles indicate that this implementation is performing correctly.

Performance

The simulation results presented here employed a total of 512k particles with some 460000 particles actually in the flow and 45000 particles in the reservoir. The simulations were run for 1200 time steps to reach steady state and then time averaged for a further 2000 time steps to generate the solution. Using 32k processors a run typically takes 3.5 hours on the Connection Machine. For comparison with other particle simulation algorithms which scale linearly with the number of particles, it is useful to consider the average time to advance one particle through one time step. Excluding the reservoir particles, for this implementation that value is 7.2 μsec/particle/timestep. By comparison, the corresponding fully vectorized implementation of this algorithm on the Cray-2 takes 0.9μsec/particle/timestep[14]. It should be noted that the Cray-2 implementation was hand vectorized with 30% of the code written directly in assembler, whereas the Connection Machine implementation was written almost fully in C* Version 4.3 with 5% of the code directly using C/Paris instructions expanded in line.

The distribution of computational time within the algorithm is as follows:

1) collisionles motion of particles (including boundary conditions)—14%
2) sort—27%
3) selection of collision partners—20%
4) collision of selected partners—39%

Figure 7 shows the computational time per particle per time step as a function of the total number of particles in the simulation. The interesting feature of this plot is the decrease in the per particle computational time with larger problems. This is a manifestation of the decreased communications time for greater virtual processor ratios. The effect is most pronounced in going from a virtual processor ratio of 1 to a ratio of 2 because collision pairings are even with odd, hence for virtual processor ratios greater than one, communication in the collision routine is maintained within the physical processor. As

the ratio becomes still greater, the communications in the sorting routine become more efficient and there is again some improvement in the computational time. General communication, that is, communication between physical processors, must take place in the sorting routine when either the motion of particles in the flow or their arbitrary rearrangement for improved randomization forces them to change physical processors. For larger virtual processor ratios this becomes less common and there is a corresponding decrease in general communication.

Future Work

Future work on this project should proceed in two directions. One direction to be followed is in improving the performance of the code. For the most part this needs to await the delivery of C* Version 5.0. The newer software allows dynamic modification of the virtual processor configuration, this can be used to speed up the computational time spent to reach steady state. There is also a richer set of scan functions in the Version 5.0 software which may be used to decrease the time spent in identifying collision candidates. Furthermore, it should be possible to run simulations with 10^6 particles just because of the increased usable memory (presently, 25% of the memory is reserved for back–compatibility).

The other direction to be followed in future work is that of increasing the generality of the algorithm. Specifically, the boundary conditions should include no slip adiabatic and isothermal walls and allow bodies other than wedges to be studied. The code should also be extended to 3D and the molecular model should ge generalised to allow power law interactions and relaxation into vibrational energy.

Acknowledgements

I would like to acknowledge Donald Baganoff for his guidance throughout the course of the work and George Adams for his help with understanding the Connection Machine. I would also like to acknowledge Jeff McDonald for his enthusiasm and willingness to answer all questions at all times.

The work reported here was supported in part by Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Administration (USRA), by NASA under grant NAGW-965 and by the Air Force under grant AFOSR 88-0139.

References

[1] Bird, G.A., *Molecular Gas Dynamics*, Oxford University Press, London, 1976.

[2] Bird, G.A., "Monte Carlo Simulation of Gas Flows," *Annual Review of Fluid Mechanics,* Vol. 10, 1978, pp.11-31.

[3] Bird, G.A., "Monte Carlo Simulation in an Engineering Context," *Prog. in Astro. and Aero.,* Vol. 74, pp. 239-255, 1981.

[4] Bird, G.A., "Direct Simulation of Gas Flows at the Molecular Level," *Proceedings of the First World Congress on Computational Mechanics,* The University of Texas at Austin, September 22-26, 1986.

[5] Derzko, N.A., "Review of Monte Carlo Methods in Kinetic Theory," *UTIAS Review,* No. 35, Univ. of Toronto, 1972.

[6] McDonald, J.D., Baganoff, D. "Vectorization of a Particle Simulation Method for Hypersonic Rarefied Flow," *AIAA-88-2735* from AIAA Thermophysics, Plasmadynamics and Lasers Conference, San Antonio, June 27-29, 1988.

[7] Pryor, D.V., Burns, P.J., "Vectorized Monte Carlo Molecular Aerodynamics Simulation of the Rayleigh Problem," *Proceedings-Supercomputing '88, Nov. 14-18, 1988, Orlando FL*, pp. 384–391.

[8] Nanbu, K., "Direct Simulation Scheme Derived from the Boltzmann Equation," *J. Phys. Soc. Japan*, vol. 49, pp. 2042–2049, 1980.

[9] Ploss, H., "On Simulation Methods for Solving the Boltzmann Equation," *Computing,* Vol. 38, pp. 101–115, 1987.

[10] Thinking Machines Corp., *The Connection Machine System-Paris Reference Manual Version 5.0A Field Test*, June 1988.

[11] Hillis, W.D., Steele, G.L., "Data Parallel Algorithms," *Communications of the ACM,* Vol. 29, No.12, pp. 1170–1183, 1986.

[12] Knuth, D.E., *The Art of Computer Programming*, Vol. 2, 2^{nd} ed., pp. 139–140, Addison-Wesley, Reading MA, 1973.

[13] Aldous, D., Diaconis, P., "Shuffling Cards and Stopping Times," *American Mathematical Monthly,* Vol. 93, No. 5, pp. 333–348, 1986.

[14] McDonald, J.D., Private communication, January 1989.

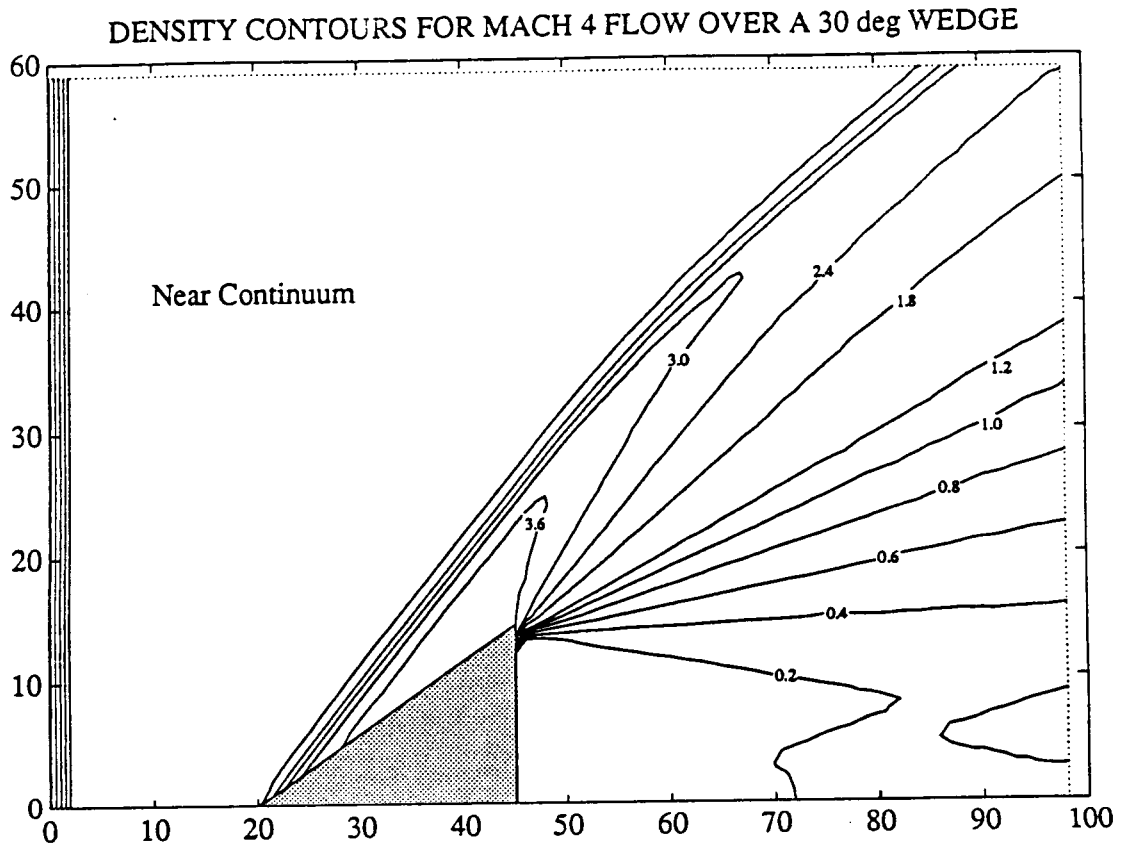

Figure 1. Density contours for near continuum Mach 4 flow over a 30° wedge.

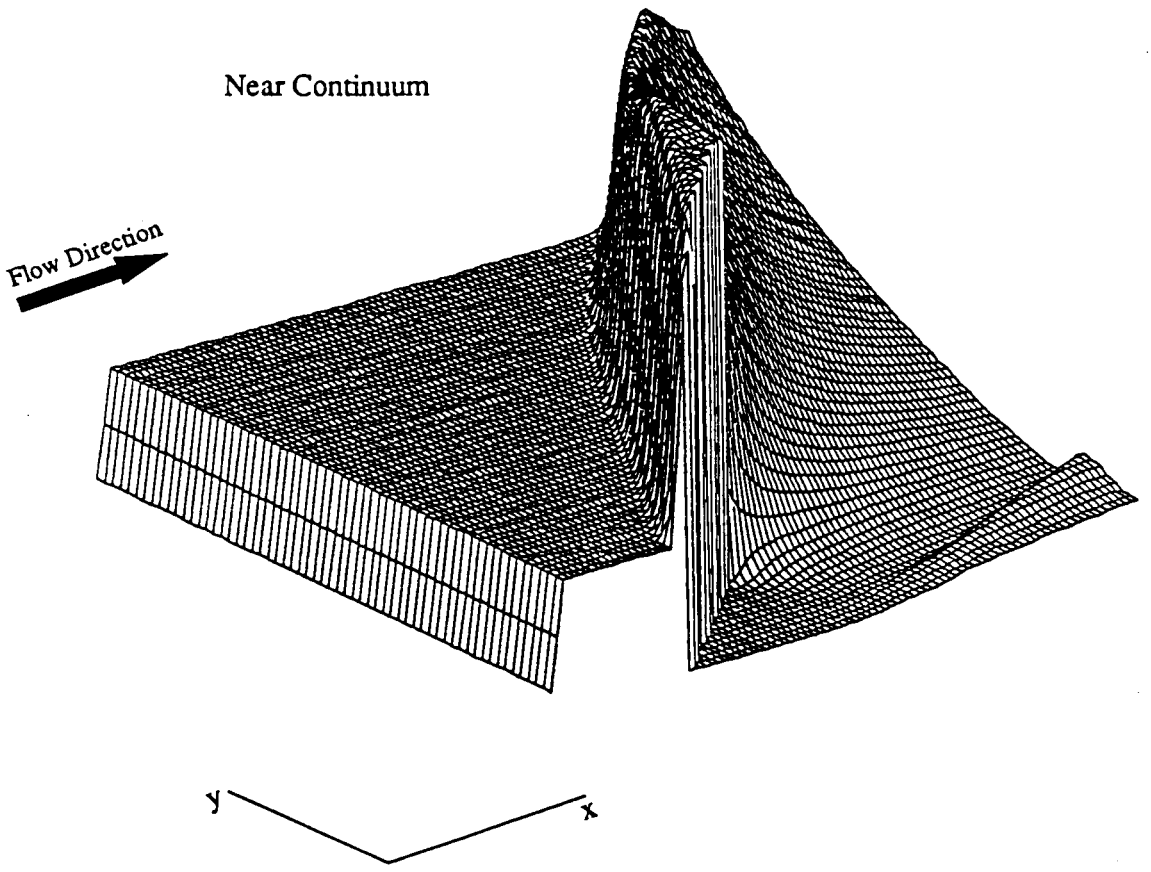

Figure 2. Density surface for near continuum Mach 4 flow over a 30° wedge.

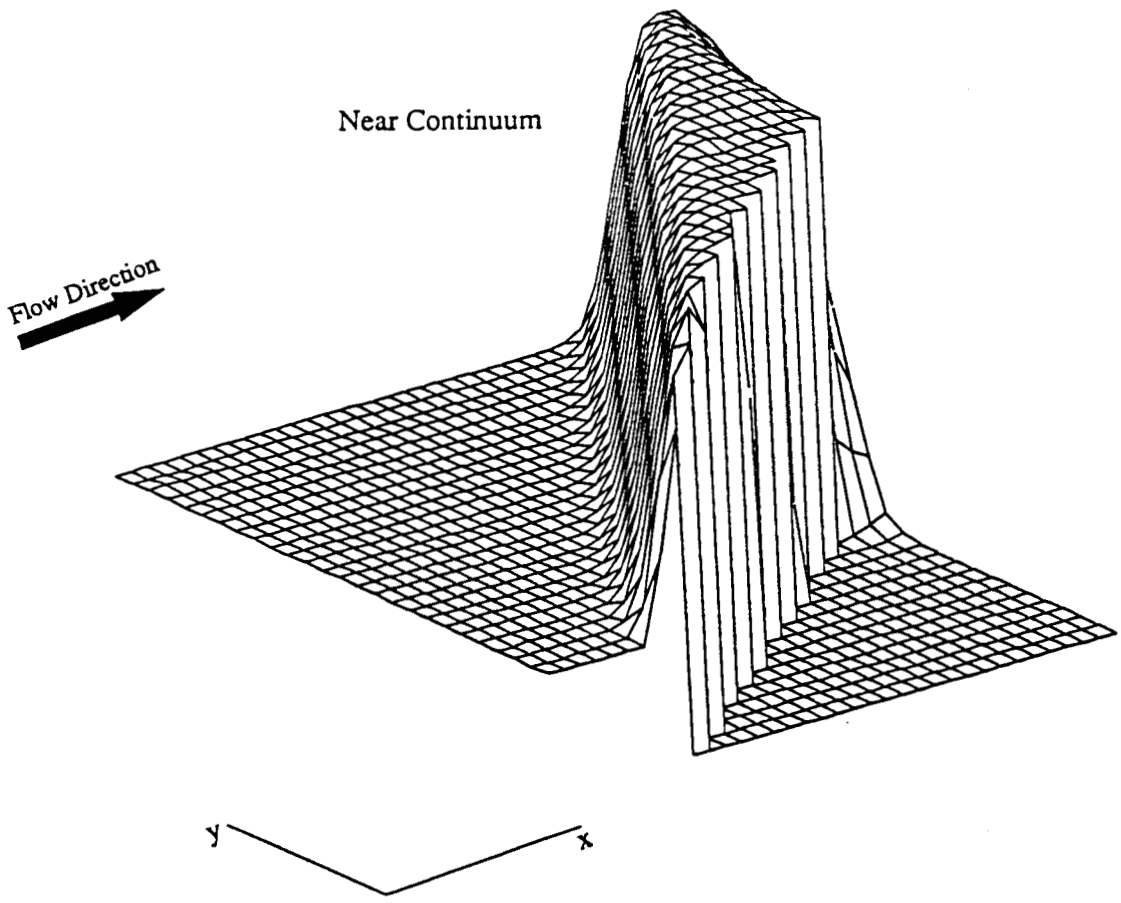

Figure 3. Density surface in the stagnation region for near continuum Mach 4 flow over a 30° wedge.

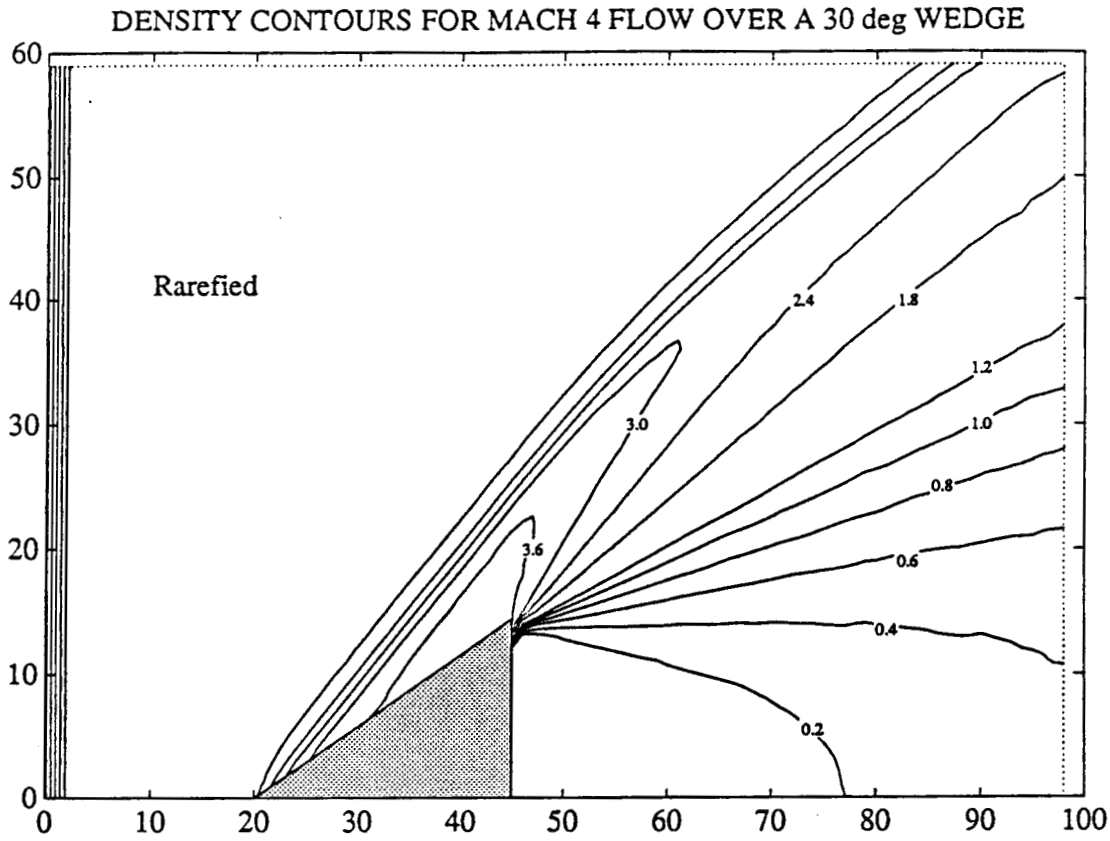

Figure 4. Density contours for rarefied Mach 4 flow over a 30° wedge. The freestream mean free path is 0.5 cell widths, corresponding Knudsen and Reynolds number are 0.02 and 600 respectively.

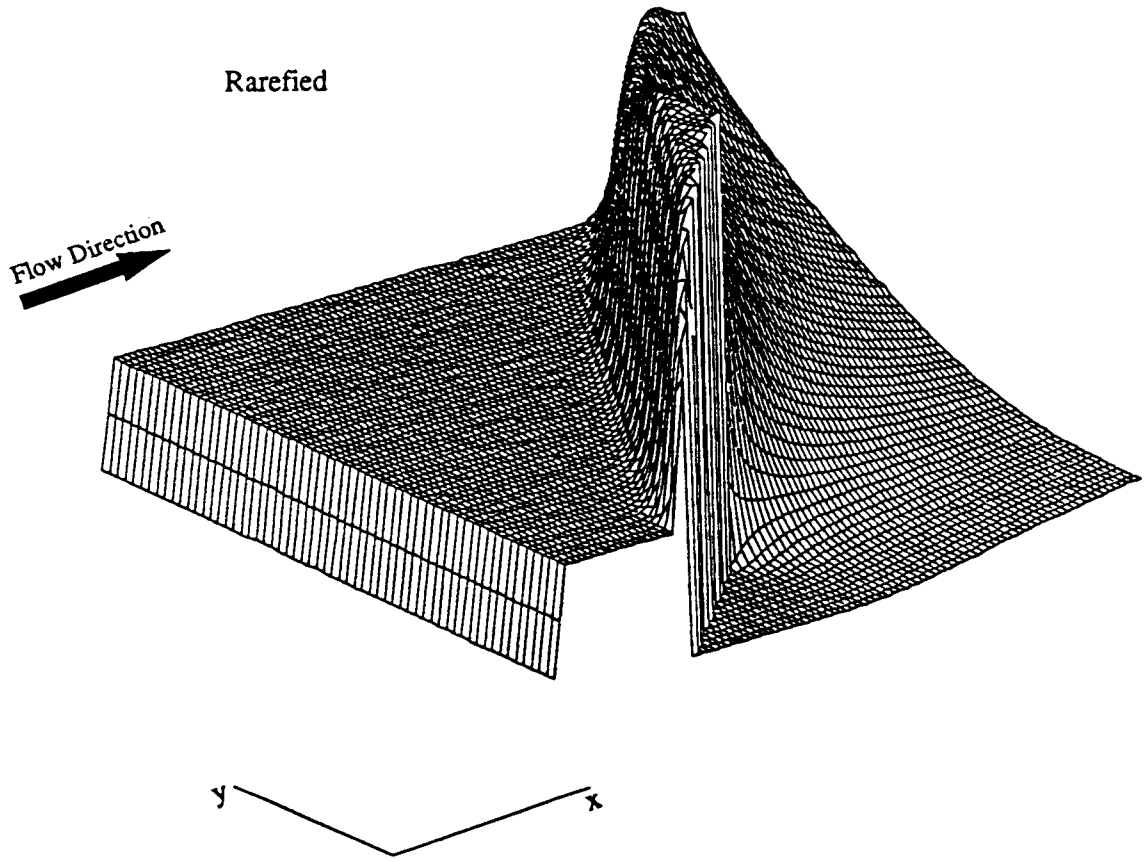

Figure 5. Density surface for rarefied Mach 4 flow over a 30° wedge. The freestream mean free path is 0.5 cell widths, corresponding Knudsen and Reynolds number are 0.02 and 600 respectively.

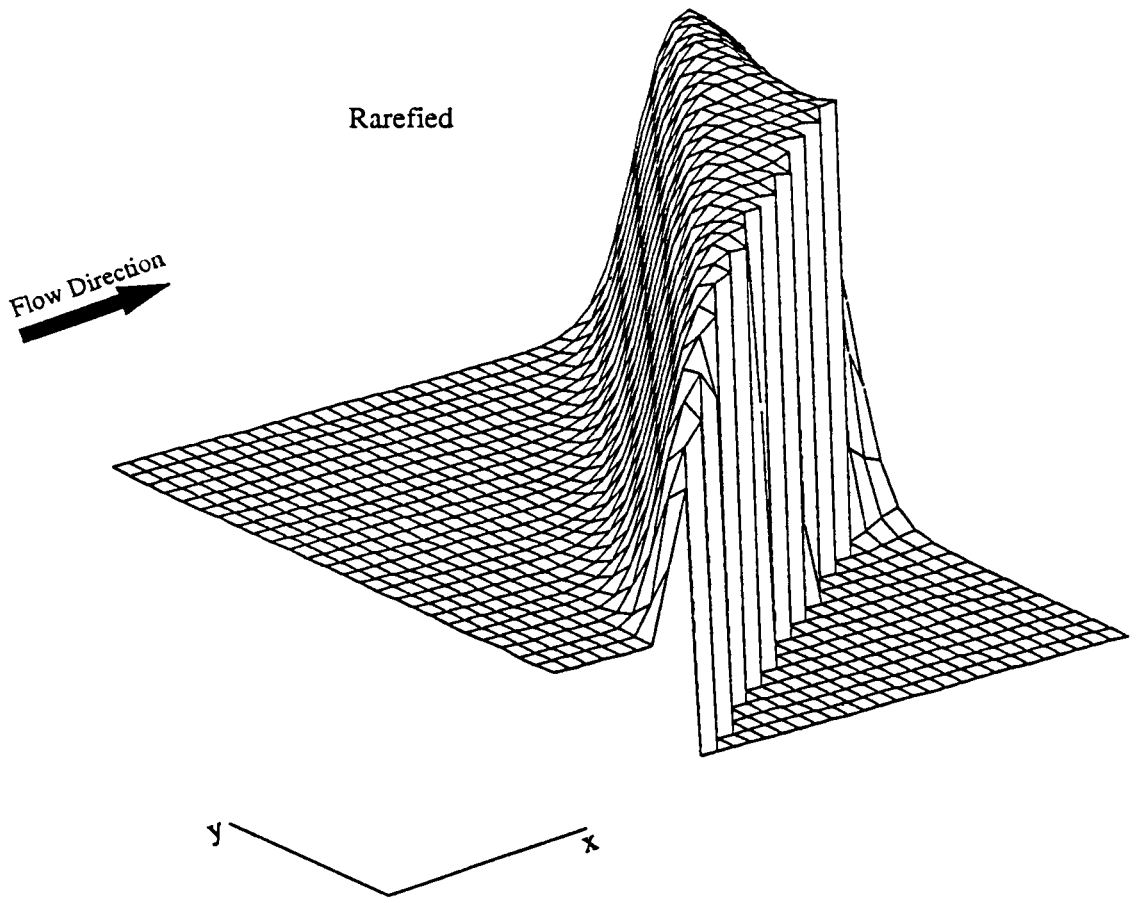

Figure 6. Density surface in the stagnation region for rarefied Mach 4 flow over a 30° wedge. The freestream mean free path is 0.5 cell widths, corresponding Knudsen and Reynolds number are 0.02 and 600 respectively.

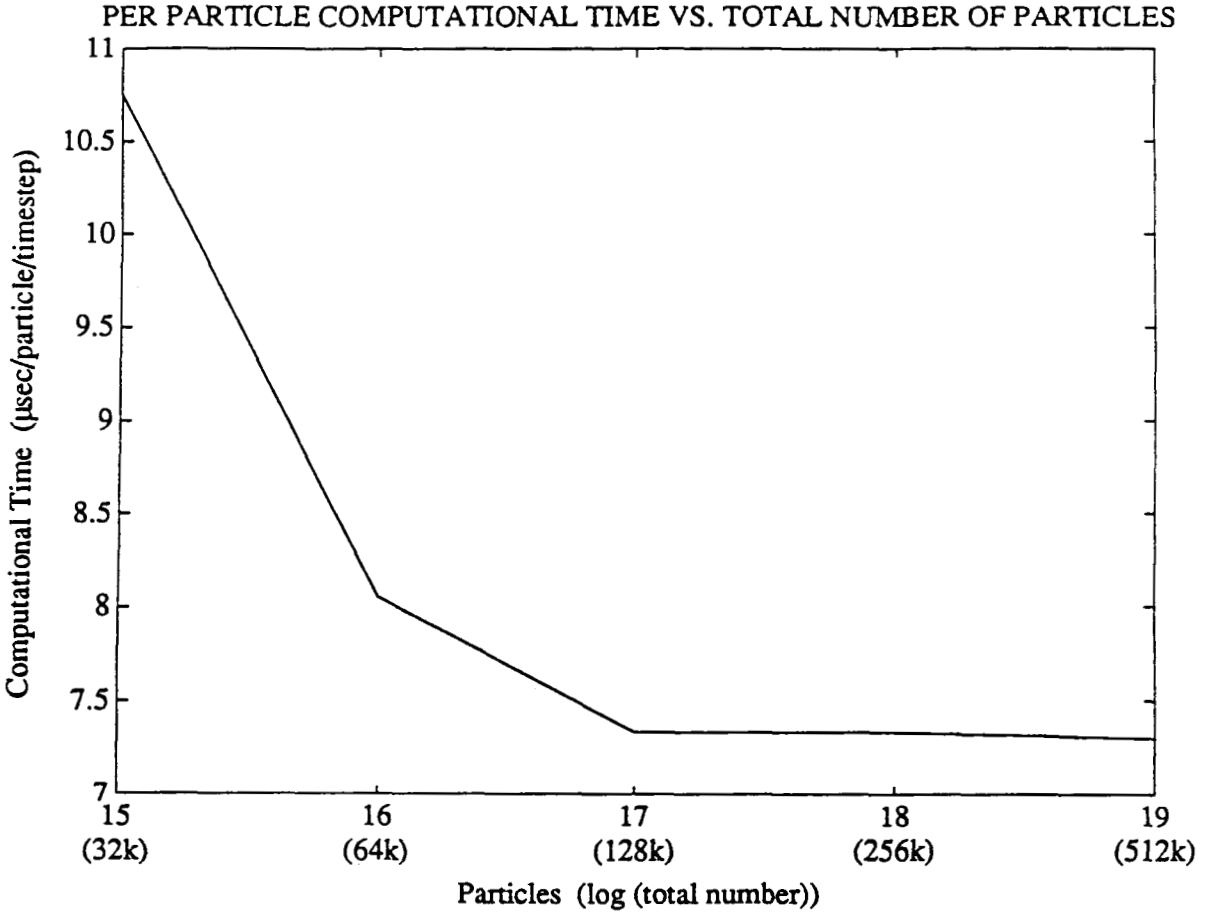

Figure 7. Computational time per particle per time step as a function of the total number of particles in the simulation. The computational time is ratioed by the number of particles actually in the flow, this number is 10% less than the total number of particles in the simulation. The size of the machine was held fixed, consequently the virtual processor ratio corresponds directly with the total number of particles in the simulation.