6

Three-Dimensional Direct Particle Simulation On the Connection Machine

Leonardo Dagum[1]
Computer Sciences Corporation
NASA Ames Research Center
Moffett Field, CA 94035

Abstract

This paper presents the algorithms necessary for an efficient data parallel implementation of a three dimensional particle simulation. In particular, a general master/slave algorithm and a fast sorting algorithm are described and the use of these algorithms in a particle simulation is outlined. A particle simulation using these algorithms has been implemented on a 32768 processor Connection Machine that is capable of simulating over 30 million particles at an average rate of 2.4 μs/particle/step. Results are presented from the simulation of flow over an Aeroassisted Flight Experiment (AFE) geometry at 100 km altitude.

1 Introduction

Interest in analysing hypersonic flows has led to great activity in the field of direct particle simulation. Much of the current work has focused on designing algorithms targeted specifically to vector architectures in an attempt to improve the performance of this method on vector machines [2, 6, 14, 15]. The results from this effort have been rewarding, and it can be said that the algorithms for implementing a fully vectorized particle simulation are well established. Not as well established, however, are the algorithms necessary for an efficient data parallel implementation of a particle simulation. The current trend towards massively parallel architectures makes the investigation of parallel algorithms for direct particle simulation both appropriate and timely.

This paper describes the data parallel algorithms necessary for the efficient implementation of a three dimensional particle simulation on the Connection Machine. Results from a flow simulation for the Aeroassisted Flight Experiment (AFE) geometry are presented.

[1] Research Scientist, Member AIAA

2 Connection Machine Architecture

The Thinking Machines Connection Machine Model CM-2 is a massively parallel single-instruction multiple-data (SIMD) computer consisting of many thousands of bit serial data processors under the direction of a front end computer. The system at NASA Ames consists of 32768 bit serial processors each with 1 Mbit of memory and operating at 7 Mhz. The processors and memory are packaged as 16 to a chip. Each chip also contains the routing circuitry which allows any processor to send and receive messages from any other processor in the system. In addition, there are 1024 64-bit Weitek floating point processors which are fed from the bit serial processors through a special purpose "Sprint" chip. There is one Sprint chip connecting every two CM chips to a Weitek. Each Weitek processor can execute an add and a multiply each clock cycle thus performing at 14 MFLOPS and yielding a peak aggregate performance of 14 GFLOPS for the system.

The Connection Machine can be viewed two ways, either as an 11-dimensional hypercube connecting the 2048 CM chips or a 10-dimensional hypercube connecting the 1024 Weitek processing elements. The first view is the "fieldwise" model of the machine which has existed since its introduction. This view admits to the existence of at least 32768 physical processors (when using the whole machine) each storing data in fields within its local memory. The second is the more recent "slicewise" model of the machine which admits to only 1024 processing elements (when using the whole machine) each storing data in slices of 32 bits distributed across the 32 physical processors in the processing element. Both models allow for "virtual processing", where the resources of a single processor or processing element may be divided to allow a greater number of virtual processors (VP's).

Regardless of the machine model, the architecture allows interprocessor communication to proceed in three manners. For very general communication with no regular pattern, the router determines the destination of messages at run time and directs the messages accordingly. This is referred to as general router communication. For communication with an irregular but static pattern, the message paths may be precompiled and the router will direct messages according to the pre-compiled paths. This is referred to as compiled communication and is faster than general router communication. Finally, for communication which is perfectly regular and involves only shifts along grid axes, the system software optimizes the data layout by ensuring strictly nearest neighbor communication and uses its own pre-compiled paths. This is referred to as NEWS (for "NorthEastWestSouth") communication. Despite the name, NEWS communication is not restricted to 2-dimensional grids, and up to 31-dimensional NEWS grids may be specified. When processors are prescribed by a NEWS grid it is possible to efficiently perform *scan* operations [16] across a grid axis. A scan operation with binary operator $\oplus$ across an ordered set $[a_0, a_1, \ldots, a_{n-1}]$ returns the ordered set $[a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \cdots \oplus a_{n-1})]$.

3 Implementation Details

Because of the SIMD nature of the Connection Machine, it is difficult to efficiently implement either the "time counter" [3] or the "no time counter" [5] collision selection rules of Bird's direct simulation Monte Carlo (DSMC) method. For this reason Baganoff and McDonald's [2] collision selection rule has been applied, and the algorithms of the Stanford particle simulation method have been implemented.

A single time step in the particle simulation can be considered comprised of six events:

1. Collisionless motion of particles.
2. Enforcement of boundary conditions.
3. Sorting of particles into cells.
4. Pairing of collision partners.
5. Collision of selected collision partners.
6. Sampling for macroscopic flow quantities.

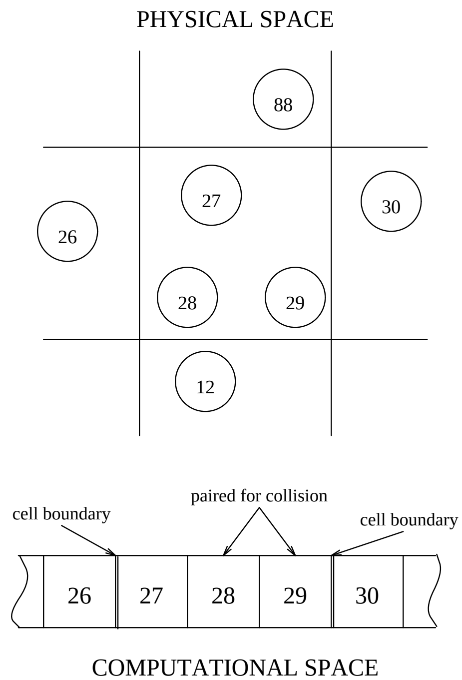

Figure 1: Mapping of particle data to virtual processors in the Connection Machine. Each particle in physical space is represented by one virtual processor in computational space.

Detailed description of these algorithms may be found in [13] and [8].

In the implementation, one virtual processor is assigned to each particle, therefore each physical processor is simulating a fixed and identical number of particles. Figure 1 illustrates the parallel decomposition of the problem. The set of virtual processors (VP-set) storing the particle data is referred to as the *main* VP-set. Several other VP-sets are used in the simulation and all the VP-sets are one-dimensional NEWS grids. Note in figure 1 that particles occupying the same cell in physical space are represented by neighboring processors in computational space. When the particles are ordered in this manner, cell boundaries may be quickly determined by having every processor compare its particle's cell index with that of its neighboring processor. Cell boundaries are often used to delimit segments in scan operations. This convenient ordering is lost when particles move into different cells and the order must be restored by sorting. With the particle data sorted it becomes a simple matter to match pairs for collision as well as to sample for macroscopic flow quantities. The algorithms used to implement the six events comprising a time step are described below.

3.1 Enforcing Boundary Conditions

The Stanford particle simulation method currently employs a grid of cubic cells for deciding on collisions and sampling macroscopic flow quantities. Arbitrary three dimensional geometries are defined in the grid of cells in the manner described in [13]. The geometry information is stored in a distinct VP-set, referred to as the *geometry* VP-set, and is accessed through a master/slave scheme similar to that described in [8].

The master/slave scheme is a way of allowing a good load balance to persist in parts of the calculation which do not involve a large fraction of the particles. The enforcement of boundary conditions is such an example. Typically only a small number of particles interact with the body on any given time step (for example, in the application discussed below, less than 4% of the total number of particles need to be considered for possible surface interaction). Since the work for each particle is being carried out by an individual virtual processor, a straightforward application of the boundary conditions would result in a large number of VP's remaining idle with only a small number of VP's actually involved in the calculation. However, assume for now that without accessing the geometry VP-set one knows which particles must must be considered for possible surface interaction. In such a case it is profitable to carry out the surface interaction with a master/slave algorithm. In its general form, the master/slave algorithm is as follows:

Master/Slave Algorithm:

1. In the *master* VP-set:
   (a) Create a *slave* VP-set large enough to accommodate all the particles to undergo the action.
   (b) Send the address of these particles to the slave VP-set.
2. In the slave VP-set:
   (a) Get the information necessary for the calculation from the master VP-set and any other VP-sets.
   (b) Perform the necessary calculation.
   (c) Send the results back to the master VP-set.
3. In the master VP-set:
   (a) Destroy the slave VP-set and free its memory.

Note that the master performs the minimum allowable work and the slave initiates communication wherever possible. The latter is especially important because the execution time for communication across VP-sets is not commutative when the VP-sets are of different size.

In order to effectively use the master/slave algorithm for enforcing boundary conditions it is necessary to somehow identify which particles have entered so-called "geometry" cells. These are cells for which a boundary is defined in the geometry VP-set. It is not practical to let the processors in the main VP-set directly access the data in the geometry VP-set because of the enormous communication cost it would entail. Instead, for every cell in the simulation there is stored a 28-bit descriptor of the region local to the cell. The first 27 bits of the descriptor map the cell itself and its 26 immediate neighbors, the last bit maps the region comprised by the 98 cells that are two away. If a bit in the descriptor has value 1, then the region it maps (either a neighboring cell or group of cells) includes a geometry cell. Conversely, if a bit has value 0 then the region it maps does not include a geometry cell.

The local region descriptors are stored in a separate VP-set which will be referred to as the *space* VP-set. The space VP-set also stores, for every a cell, a pointer to the appropriate address in the geometry VP-set. At the beginning of every time step, descriptors are broadcast to every processor in the main VP-set as follows:

Broadcast Algorithm:

1. In the main VP-set, identify one processor for every occupied cell.
2. Send the processor's self-address to the appropriate processor in the space VP-set.
3. Processors in the space VP-set which receive a message from the main VP-set return the local region descriptor.
4. The local region descriptors received by the processors active in Step 1 are copied with a scan operation to the other processors representing particles in the same cell.

Note that the space VP-set is much smaller than the main VP-set, therefore the principle behind the work allocation in the master/slave algorithm applies. Furthermore, note that at the beginning of the time step the particle data in the main VP-set is sorted, therefore steps 1 and 4 can be carried out using NEWS communication.

Processors in the main VP-set can now update their particle's position. At the end of the free motion, each particle's cell position is computed and its local region descriptor is consulted. Those particles which may need to interact with a body surface are identified and the master/slave algorithm is applied. The slave processors access the space VP-set to get pointers into the the geometry VP-set from which they can then retrieve the geometry data. Because it is extremely unlikely that a particle move more than two cell widths in one time step (see [13, 8]), the local region descriptor maps neighboring cells no further than two away.

3.2 Sorting Particles and Pairing Collision Partners

When the particle data is sorted, neighboring virtual processors in the main VP-set will store data for particles occupying the same cell (except, of course, at cell boundaries). Collision partners then may be paired on an even-with-odd basis, that is, all even numbered particles are considered for collision with their odd numbered neighbor (see figure 1). Naturally if the odd numbered neighbor is occupying a different cell then the two particles are not allowed to collide.

At the beginning of a time step the particle data is in an ordered state allowing easy access to the information for particles occupying the same cell. However, diffusion and convection of the particles through a time step destroys the order, and it is necessary to sort the particles on every time step. Sorting is a communication intensive process and can be very costly on a parallel architecture. However, knowledge of the mechanism behind the disordering can be used to greatly reduce this cost, and [9] describes a very efficient sorting algorithm for two dimensional particle simulations. The extension of this algorithm to three dimensions is described below. The changes made to the algorithm should be carried back to the two dimensional case to improve the performance there as well.

Three fundamental observations on the mechanism behind the disordering may be used to reduce the problem of sorting to one of merging. In particular, one may observe:

1. At the beginning of a time step the particle data is ordered and becomes disordered only through the motion of the particles.

2. The range of motion of a particle over one time step is, to a very high probability, limited to less than two cell widths.

3. Most particles do not change cells over one time step. Furthermore, of those particles which do change cells, the majority make the same cell change.

The first observation indicates that the data is never very far out of order. Therefore it is reasonable to expect that a specially tailored sorting algorithm may be substantially faster than a general one. This observation implies that disorder occurs when particles change cells. Trivially, if particles did not change cells then they would remain sorted, but also *if all particles made the identical cell change then they would still remain sorted.* All particles which undergo the same cell change represent an ordered set, and one can divide the particles into mutually exclusive sets based on their cell change. The problem of sorting then becomes one of identifying and merging these ordered sets. The next two observations aid in that respect.

The second observation indicates that there is a limited number of ordered sets to be identified. If particles move no more than two cell widths in a time step, then there can only be 125 possible cell changes and no more than 125 ordered sets to be found. In practice many of these will be empty sets and typically only about 25 non-empty ordered sets will be found in the data. For the two dimensional algorithm described in [9] one proceeded simply to merge the non-empty sets. Since in two dimensions there usually are only 9 sets to be merged this is sufficiently efficient. However with 25 non-empty sets the merging becomes too costly, therefore one can make use of the third observation to reduce the number of sets to be merged. This observation indicates that most of the particles will belong to just two sets. The three-dimensional sorting algorithm then proceeds as follows:

Flux-sort

1. Identify the two largest ordered sets of particles and enumerate them.

2. Sort the remaining particles using a master/slave algorithm.

3. Merge the three ordered sets.

The enumeration in step 1 restarts at every cell boundary. The sorting of step 2 is very fast because only a small number of keys need to be sorted. The merging used for step 3 is described in detail in [9]. Since only three sets need to be merged, the memory and communication requirements for the merging are less than for the two dimensional algorithm described in [9]. Flux-sort will have about the same performance in either a two dimensional or a three dimensional simulation and in both cases will outperform the algorithm in [9]. Since the scan operations used in flux-sort are vectorizable (see [7]), flux-sort should also be considered for vector architectures where it may outperform the partially vectorized bucket sort described in [6]. Finally, it is important to note that flux-sort scales linearly with the number of particles in the simulation.

3.3 Statistical Independence

For the simulation to accurately reflect the expected collision frequency in a gas, it is necessary that the set of pairings considered for collision be statistically independent between time steps. A straightforward approach toward ensuring such independence would be to select pairs randomly from a cell. Unfortunately, such a scheme does not readily lend itself to implementation on a SIMD architecture like the Connection Machine. The alternative approach is to employ a regular pairing (for example, the even-with-odd pairing used here) but apply a shuffling of the in-cell ordering prior to the pairing. The regular pairing allows a regular communication pattern which leads to a simple and efficient implementation. The difficulty then is to find efficient algorithms for shuffling the in-cell ordering. Two different shuffling algorithms are employed every time step and these are described below.

Shuffling occurs before the particle data has been moved into its ordered arrangement. The following assumes that each processor in the main VP-set is storing its particle's in-cell number, IC_i. The numbering for cell C_i has range $[0, n_{C_i})$ where n_{C_i} is the cell density. The object is to shuffle this numbering such that the order of particles within a cell is randomized. Since it does not matter if neighboring cells are randomized in the same fashion, it is feasible to apply a single random permutation to the in-cell ordering of all the cells in the simulation. Because permutation of fixed length, P, must be applied to lists of varying length, the permutation is repeated within lists of length greater than $2P$ and not applied to lists of length less than P. The algorithm proceeds as follows:

Repeated Random Shuffle

1. On the front end, generate a random permutation p_j where j has range $[0, P)$. Applying $P \log P$ random transpositions to an existing permutation guarantees independence between the two permutations [1].
2. Select all processors representing particles occupying cells with density $n_{C_i} \geq P$.
3. Compute a group number as $g_i = \lfloor IC_i/P \rfloor$.
4. In processors with $g_i < \lfloor n_{C_i}/P \rfloor$, reassign in-cell numbers as:

$$IC_{i \bmod P} \longleftarrow g_i P + p_j \text{ for } j = 0, \ldots, P-1$$

Note that the scale over which randomization has occurred is fixed by P. If P is too small, then any large scale mixing is inhibited. If P is too large, then too many cells may be neglected. In practice the algorithm is applied twice, once with a small value of P and once with a large value of P (where 32 is considered large). For very large scale mixing the algorithm suffers from poor load balance. In particular step 4 requires P reassignments each with no more than N/P processors active (where N is the total number of particles).

To introduce mixing at the scale of the cell density a regular shuffling algorithm is employed. This algorithm proceeds as follows:

Regular Shuffle

1. Divide the particles in a cell into k groups of size $G = \lfloor n_{C_i}/k \rfloor$ and number the particles in each group as $g_i = IC_i \bmod G$.
2. Within each group, reverse the even group numbers as:

$$g_i \longleftarrow 2\lfloor G/2 \rfloor - g_i \text{ for } i = 0, 2, \ldots, 2\lfloor G/2 \rfloor$$

3. Reassign in-cell numbers as:

$$IC_i \longleftarrow g_i + G\lfloor IC_i/G \rfloor \text{ for } i = 0, \ldots, n_{C_i}$$

The mixing scale for this algorithm depends on the selected value for k. In practice k alternately has value 1, 2, or 3 depending on the time step. Large scale mixing is necessary because the merging used in the flux-sort algorithm effectively ensures that particles with the same characteristic bulk motion get grouped together in the main VP-set (see [9]). This is a natural outcome of making sets from particles which undergo the same cell change, and without the regular shuffle the even-with-odd pairing would result in collisions primarily amongst particles belonging to the same set.

3.4 Colliding Particles

Collision mechanics are implemented as described in [8]. Currently only single specie monatomic or diatomic gases may be simulated. Thermal nonequilibrium between translational and rotational modes is handled according to the Borgnakke-Larsen model described in [13]. Vibrational relaxation also employs the discrete model also described in [13]. Collision numbers are fixed at 5 for rotation and 50 for vibration. Eventually multiple reacting species will be allowed and it is expected that the vibration and chemistry models described in [12] will be implemented.

3.5 Sampling Macroscopic Flow Quantities

For diatomic gases 7 macroscopic quantities are sampled. These include density, velocity, and three temperatures. These quantities are computed in the main VP-set for every occupied cell. Because the particles are sorted, scan operations may be used to compute the necessary averages. The results are stored in a separate *sampling* VP-set. A master/slave algorithm (with the sampling VP-set acting as the slave) is employed to transfer the sampled quantities from the main VP-set. Samples are collected every time step once steady state has been reached.

4 Validation

Validation of the implemented algorithms was undertaken through test calculations for thermal relaxation of a gas, shock wave profiles, and shear stress profiles in Couette flow. Figure 2 presents sample results from a shock wave simulation. The normalized density and temperature profiles for a Mach 10 shock wave in a perfect diatomic gas as computed by McDonald [13] are given by the solid curves. Position along the profile is normalized by the mean free path before the shock. The symbols represent results obtained with the Connection Machine implementation described here. The shock tube was three dimensional with size $60 \times 4 \times 4$.

The object in making this and other shock wave calculations was to validate the shuffling algorithms for the three dimensional simulation. Any statistical dependence between pairings made on subsequent time steps results in an incorrect collision frequency. Typically this will widen the shock profiles and prevent thermal equilibrium from being reached behind the shock. These effects were not observed and the shock wave results compared favorably to accepted solutions. Results from thermal relaxation simulations are given in [8]; results from Couette flow simulations are given in [10].

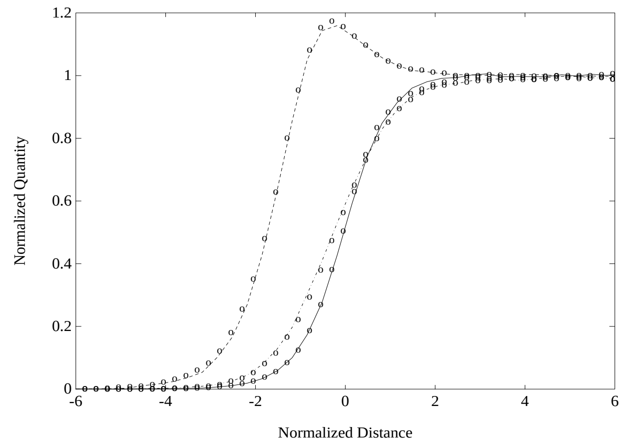

Figure 2: Normalized density and temperature profiles across a Mach 10 shock wave in a perfect diatomic gas. Symbols represent results from three dimensional simulation on the Connection Machine and lines represent calculations made by McDonald [13]: — ρ; - - - T_{tr}; -.-.-. T_{rot}.

5 Calculations

To demonstrate the capabilities of the method the results from a large scale three dimensional simulation for the hypersonic flow about the Aeroassisted Flight Experiment (AFE) vehicle are presented. The geometry is identical to that described in [11]. The free stream conditions corresponded to flight at 100 km altitude where the mean free path is 10 cm and the temperature is 194K. The Knudsen number based on the 4.25 m aeroshell diameter is 0.0235.

The simulation geometry had a diameter of 44 cell widths and the hard sphere free stream mean free path was set at 1.035 cell widths in order to match Knudsen numbers. For the assumed inverse ninth power law potential, this required setting the simulation mean free path to 0.854 as described in [4]. The free stream Mach number was 35.42 corresponding to a vehicle velocity of 9.9 km/s. The surface temperature was fixed at 1500K. The simulated particles corresponded to molecular nitrogen with a characteristic temperature for vibration of 3390K.

The geometry was placed in a wind tunnel of dimensions $55 \times 120 \times 60$ and the simulation was started with just 10^6 particles. As steady state was approached particle cloning was used to bring the particle count up to about 32 million. Averaging was then carried out for 150 steps with samples collected on every step.

Figure 3 presents the temperatures along the stag-

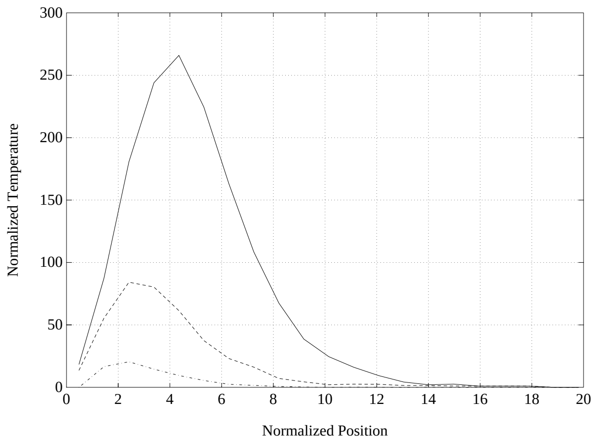

Figure 3: Normalized temperatures along the stagnation streamline: — T_{tr}; - - - T_{rot}; –.–.–T_{vib}. Position is normalized by the free stream mean free path.

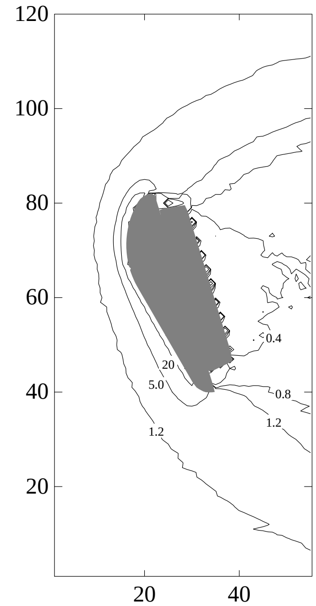

Figure 4: Density contours in symmetry plane for flow over AFE. Contours shown at 0.4, 0.8, 1.2, 5.0 and 20 times the free stream density.

nation streamline normalized by the free stream temperature. Unfortunately these results cannot be compared to the the Cray implementation results presented in [11] because of a known error in boundary conditions used in [11]. The peak normalized translational temperature of 266 corresponds to a 9% overshoot on the Rankine-Hugoniot jump value of 245 for this Mach number in a perfect diatomic gas. The rotational and vibrational temperatures lag behind the translational temperature with peak values of 84 and 20 respectively. With a collision number of 50, vibrational modes are comparatively slower to activate and greater vibrational temperatures are found downstream of the stagnation region.

Figures 4-7 present density and temperature contours in the symmetry plane of the vehicle. The figures were generated from a single plane of the solution. (Note that several planes could have been averaged to produce smoother contours, but part of the purpose in presenting these results is to demonstrate the quality of solution which can be obtained with 32 million particles in 150 time steps.) Comparing the density and translational temperature contours (figures 4 and 5), one can see the greater standoff distance for the latter. The peak translational temperature occurs before the body and there is a sharp drop in temperature on approaching the body owing to the cold surface temperature. The rotational temperature (see figure 6) reaches its peak downstream from the stagnation region. This is a result of the greater number of collisions required to thermalize rotational modes in the gas. Vibrational modes are slower yet to thermalize and the peak vibrational temperature is not reached until well downstream of the stagnation region (see figure 7). The subsequent expansion of the flow around the shoulder of the vehicle lowers the collision rate and the activated vibrational modes are frozen into the wake. Figure 8 presents the velocity field in the symmetry plane about the vehicle. The turning of the flow about the body and the stagnant region directly behind the afterbody are clearly evident.

6 Performance

The code used for the calculation described in the previous section was written completely in C/Paris and run on all 32k processors of the Connection Machine at NASA Ames Research Center using a Sun 4/90 as the front end. The average time to advance one particle over one time step at steady state was 2.4 μsec and the entire calculation was completed in less than 6 hours. This performance is comparable to that obtained from a fully vectorized implementation running on a single CPU of the Cray 2. However, the greater amount of memory available on the CM allows a greater number of particles to be simulated (specifically, 32 million particles on a 32k processor CM compared to 20 million particles on a Cray 2 with 2 GB of memory). It is expected that a fully config-

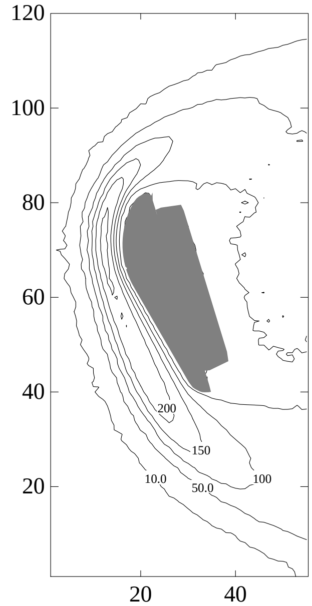

Figure 5: Translational temperature contours in symmetry plane for flow over AFE. Contours shown at 10, 50, 100, 150, 200 and 250 times the free stream temperature.

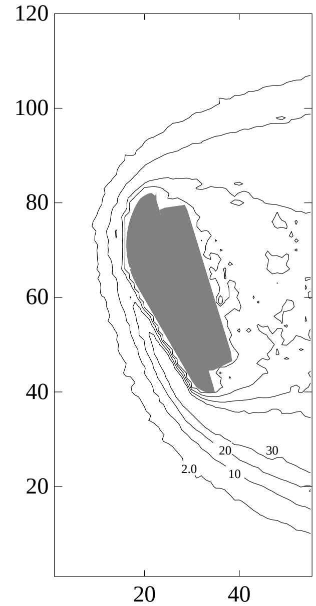

Figure 7: Vibrational temperature contours in symmetry plane for flow over AFE. Contours shown at 2.0, 10, 20 and 30 times the free stream temperature.

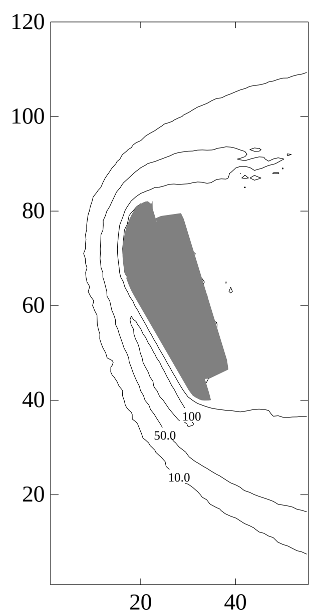

Figure 6: Rotational temperature contours in symmetry plane for flow over AFE. Contours shown at 10, 50, and 100 times the free stream temperature.

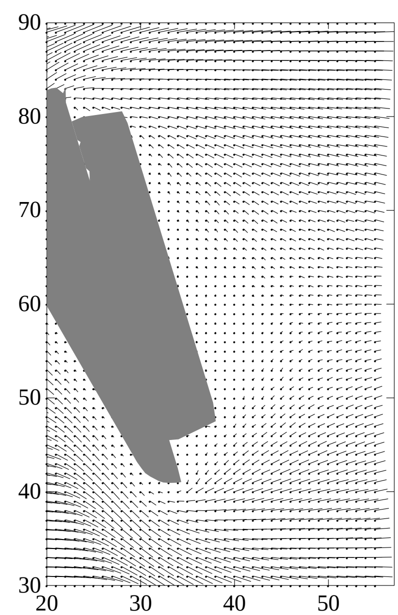

Figure 8: Velocity field in symmetry plane around the AFE.

move	20%
sort	38%
re-order	26%
collide	10%
sample	6%

Table 1: Distribution of computational time for particle simulation.

ured 64k processor CM would be capable of simulating 64 million particles at an average rate of 1.2 μsec per particle per time step.

The distribution of computational time in the code is given in table 1. The fraction of time listed for sorting includes the time to arrive at the rank for each particle and the time to shuffle the in-cell ordering. The "re-order" time is simply the time to permute the particle data into its ordered arrangement. It is this re-order time which makes the "collide" and "sample" times so low, since once the data is re-ordered the memory references for these two events are primarily local or nearest-neighbor. In a sequential implementation the re-ordering time would not appear explicitly in this manner but would essentially be accounted for in terms of memory references in the collide and sample events.

7 Discussion and Conclusions

The applicability of the current work to hypervelocity flow is somewhat limited since chemistry has not yet been incorporated. However, although the sample calculation of the previous section is physically meaningless, the aim in presenting those results is not to provide an accurate description of the real flow about the AFE but rather to demonstrate the capacity of the Connection Machine to execute large scale particle simulations. It is expected that extending this work to include multiple reacting species will not require defining any new data parallel algorithms but may be accomplished by applying the algorithms outlined in the first part of this paper.

The results presented in this paper give some indication of the possibilities open to particle simulation through distributed memory architectures. The low computational cost per simulated particle of the Stanford method makes memory the bounding resource as opposed to CPU time. In this respect, distributed memory architectures are ideally suited to particle simulation since they are most capable of providing extremely large memory capacities.

An important result of this investigation is a quantitative measure of the performance of the Connection Machine for direct particle simulation. From this result it is possible to conclude that the massively parallel architecture of the Connection Machine is quite suitable for this type of calculation with performance comparable to that of a single processor Cray 2. The main advantage of the Connection Machine is a large memory which allows the simulation of over 30 million particles. However, there are difficulties in taking full advantage of this architecture because of the lack of a broad based tradition of data parallel programming. An important outcome of this work is new data parallel algorithms specifically of use for direct particle simulation but which also expand the data parallel diction.

Acknowledgements

The author would like to thank Jeff McDonald for supplying shock profiles for comparison and Bill Feiereisen for supplying the AFE geometry. This work was sponsored under NASA contract NAS 2-12961.

References

[1] Aldous, D., Diaconis, P., *Shuffling Cards and Stopping Times*, American Mathematical Monthly, **93**, 5, pps. 333-348, 1986.

[2] Baganoff, D., McDonald, J.D., *A Collision-Selection Rule for a Particle Simulation Method Suited to Vector Computers*, Phys Fluids A, **2**, 7, pps. 1248–1259, 1990.

[3] Bird, G.A., *Molecular Gas Dynamics,* Clarendon Press, Oxford, 1976.

[4] Bird, G.A., *Definition of Mean Free Path for Real Gases,* Phys Fluids, **26**, 11, pp. 3222–3223, 1983.

[5] Bird, G.A., *Perception of Numerical Methods in Rarefied Gas Dynamics,* Rarefied Gas Dynamics, eds. E.D. Muntz, D.P. Weaver and D.H. Campbell, Progress in Astro and Aero, **118**, pps. 211-226, 1989.

[6] Boyd, I.D., *Vectorization of a Monte Carlo Scheme for Nonequilibrium Gas Dynamics,* Journal of Computational Physics, (to appear) 1991.

[7] Chatterjee, S., Blelloch, G.E., Zagha, M., *Scan Primitives for Vector Computers,* Proceedings Supercomputing '90, November 12-16, New York, NY, 1990.

[8] Dagum, L., *On the Suitability of the Connection Machine for Direct Particle Simulation,* Ph.D. Thesis, Dept Aero and Astro, Stanford Univ, Stanford, CA, 1990.

[9] Dagum, L., *Sorting for Particle Flow Simulation on the Connection Machine,* In Horst D. Simon, editor, *Research Directions in Parallel CFD*, MIT Press, Cambridge (to appear), 1991.

[10] Dagum, L., *Lip Leakage Flow Simulation for the Gravity Probe B Gas Spinup Using PSiCM* NAS Applied Research Branch Report RNR-91-010, 1991.

[11] Feiereisen, W., McDonald, J.D., Fallavollita, M.A., *Three Dimensional Discrete Particle Simulation about the AFE Geometry,* AIAA-90-1778, 1990.

[12] Haas, B.L., *Thermochemistry Models Applicable to a Vectorized Particle Simulation,* Ph.D. Thesis, Dept Aero and Astro, Stanford Univ, Stanford, CA, 1990.

[13] McDonald, J.D., *A Computationally Efficient Particle Simulation Method Suited to Vector Computer Architectures*, Ph.D. Thesis, Dept Aero and Astro, Stanford Univ, Stanford, CA, 1989.

[14] Ploss, H., *On Simulation Methods for Solving the Boltzmann Equation,* Computing, **38**, pps. 101–115, 1987.

[15] Pryor, D.V., Burns, P.J., *Vectorized Monte Carlo Molecular Aerodynamics Simulation of the Rayleigh Problem,* Journal of Supercomputing, **3**, 4, pp. 305–330, 1989.

[16] Thinking Machines Corp., *The Connection Machine System: Paris Reference Manual Version 5.0,* Cambridge, MA, 1989.