Object-Oriented/Data-Oriented Design of a Direct Simulation Monte Carlo Algorithm

Derek S. Liechty[1]
NASA Langley Research Center, Hampton, VA 23681-2199

Over the past decade, there has been much progress towards improved phenomenological modeling and algorithmic updates for the direct simulation Monte Carlo (DSMC) method, which provides a probabilistic physical simulation of gas flows. These improvements have largely been based on the work of the originator of the DSMC method, Graeme Bird. Of primary importance are improved chemistry, internal energy, and physics modeling and a reduction in time to solution. These allow for an expanded range of possible solutions in altitude and velocity space. NASA's current production code, the DSMC Analysis Code (DAC), is well-established and based on Bird's 1994 algorithms written in Fortran 77 and has proven difficult to upgrade. A new DSMC code is being developed in the C++ programming language using object-oriented and data-oriented design paradigms to facilitate the inclusion of the recent improvements and future development activities. The development efforts on the new code, the Multiphysics Algorithm with Particles (MAP), are described, and performance comparisons are made with DAC.

I. Introduction

As originally implemented, the direct simulation Monte Carlo (DSMC) method, which provides a probabilistic physical simulation of gas flows, had strict limitations because of grid size, time step, number of simulated particles per cell, etc., causing simulations to have long run times even on many CPU's, especially for near-continuum conditions (100's of hours on thousands of CPU's). However, with the introduction of modified DSMC algorithms[1], simulations can be made more quickly while using fewer numbers of simulated particles, for some cases. Also, since fewer numbers of particles are required for a given flow, the range of Knudsen numbers that can readily be simulated has been extended further into the continuum regime. The new algorithms improve the DSMC method by focusing on physical accuracy and computational efficiency. Some improvements in the past have included additions such as conservative species weighting[2] and the use of virtual sub-cells[3]. The new algorithms include features such as nearest neighbor collisions excluding the previous collision partner, separate collision and sampling cells, automatically adaptive grid and variable time steps, a modified no-time counter procedure for collisions, and discontinuous and event-driven physical processes. Further algorithm development has included improvements to axisymmetric simulations[4] and extensions to the recently proposed Quantum-Kinetic (Q-K) reaction model[5] to include internal energy levels[6-8] and charged species[8, 9].

In addition to the recent algorithmic improvements, the capability to solve highly energetic flows, including ionization and radiation, is needed. These flows are generally characterized by high velocity entry into a planetary atmosphere or the atmosphere of a natural satellite, but can also include simulations of ion propulsion systems for satellites. Although solution methods for these flows have been developed for DSMC simulations[10-17], none are currently available to NASA for large-scale problems involving many particles in flows approaching continuum conditions. In addition, there is a need in the future for an efficient, two-way coupled solution of flows containing both continuum and rarefied/transitional regions. There have been several recent advances in this area both for coupled CFD/DSMC simulations[18] and hybrid particle schemes[19, 20].

The current production DSMC algorithm employed by NASA and many other organizations is the DSMC Analysis Code (DAC)[21, 22]. DAC was developed jointly by teams at the NASA Langley Research Center and Johnson Space Center and was recognized in 2002 with the NASA Software of the Year Award. It has a long history of support of both manned missions[23, 24] and planetary missions[25]. The DSMC algorithms used in DAC are based on those developed by Bird[26], which have mostly been either revised or superseded by the newer

[1] Aerospace Engineer, Aerothermodynamics Branch, 16 Victory St./MS 408A, Senior Member.

algorithms. In addition, DAC was written in Fortran 77, and has proven challenging to implement the new algorithms, let alone including additional physics modules.

In order to take advantage of the new algorithms and physics modules, and because of the difficulties listed above regarding upgrading DAC, the development of a new DSMC code has begun using the C++ programming language. Extensive use of Object-Oriented Design (OOD) practices is being built into the proposed new code. OOD provides a clear modular structure for programs and makes it easy to maintain and modify existing code as new objects can be created with small differences to existing ones. In addition to employing OOD, which only focuses on the interaction of objects, Data-Oriented Design (DOD) shifts the perspective from the objects to the data. When considering DOD, the data type used, how it is laid out in memory, and how it will be read and processed must be taken into account when designing the program. The remainder of the current paper outlines the organization of the proposed new DSMC code and how OOD and DOD are incorporated into its design. The performance of the proposed new code is then compared to that of DAC for flow over a sphere at various conditions, the Mars Reconnaissance Orbiter in air, and the Orion Crew Module on a single processor/single core. In addition to efficiency comparisons, additional features of the proposed code will be outlined.

II. General Concepts of Object-Oriented and Data-Oriented Design with Application to DSMC

A. Object-Oriented Design

Object-Oriented Design has many advantages over conventional, procedural programming approaches. OOD provides a clear modular structure for programs, which is ideal for defining abstract data types where implementation details are hidden within the object and the object has a clearly defined interface to the rest of the program. OOD also makes it easy to maintain and modify existing code as new objects can be created with relatively small differences to existing objects.

Objects are the basic run-time entities in an object-oriented system. Programming is analyzed in terms of objects and the nature of communication between them. When a program is executed, objects interact with each other by sending messages. Different objects can also interact with each other without knowing the details of their data or code. An object is created from a class definition and any number of objects can be created from a single class definition. Of particular interest to the current application are the ideas of inheritance and polymorphism. Inheritance is the process by which objects can acquire the properties of objects of other classes. In OOD, inheritance provides reusability, like adding additional features to an existing class without modifying the original class. This is achieved by deriving a new class from the existing one. The new class will have the combined features of both classes. Polymorphism means the ability to take more than one form. An operation may exhibit different behaviors in different instances. The behavior depends on the data types used in the operation. Polymorphism is extensively used in implementing inheritance.

When considering OOD with regards to a DSMC algorithm, base classes for various objects of interest, such as particles, cells, species, etc., must be designed to be general enough to take advantage of inheritance. For example, derived data types for particles have the potential to streamline memory and communication overheads by not including particle information on a species-by-species basis that is not required for that species (e.g. argon does not need rotational or vibrational information while molecular nitrogen does). In addition, since derived data types potentially have different required information, each derived class must be able to perform its own input/output and communications operations. A simple example of what a base particle class might look like is presented in Figure 1.

```
class bParticle
{
      short int pNum;          /* Particle number */
      short int lastColl;      /* Particle number of last
                                  collision partner */
      float *vars;             /* Variables for base particle
                                0-2: x,y,z
                                3-5: u,v,w
                                6,7: dt,time */
public:
      bParticle();             /* Class constructor */
      ~bParticle();            /* Class destructor */
      /* Class I/O */
      void writeFile();
      void readFile();
      /* Public access to private variables */
      float getX(int &i);
      void setX(int &i, float val);
      float getU(int &i);
      void setU(int &i, float val);
      float getDt();
      void setDt(float val);
      float getTime();
      void setTime(float val);

};
```

Figure 1. Example of OOD particle base class.

B. Data-Oriented Design

Simply using OOD practices, however, can lead to random memory access patterns and constant cache misses. Data-Oriented Design is a different way to approach program design that addresses these issues. Procedural programming focuses on procedure calls as its main element, and OOD deals primarily with objects. The main focus of these approaches is code: simply procedures (or functions) in one case, and grouped code associated with some internal state in the other. DOD shifts the perspective of programming from objects to the data itself: the type of data, how it is laid out in memory, and how it will be read and processed during program execution.

A good example of the use of DOD is the definition of a particle container class and is presented in Figure 2. Notice that, compared to the OOD class in Figure 1, the new particle class now contains an extra dimension to the previous definition of particle variables/arrays. This arrangement allows for rapid traversal through the particle variables because the data are located in contiguous memory locations.

```
class bParticle
{
      unsigned short int npart;   /* Number of particles */
      unsigned short int arrayLen;/* Array length */
      short int *pNum;            /* Particle number */
      short int *lastColl;        /* Particle number of last
                                     collision partner */
      float **vars;               /* Variables for base particle
                                   0-2: x,y,z
                                   3-5: u,v,w
                                   6,7: dt,time */
public:
      bParticle();                /* Class constructor */
      ~bParticle();               /* Class destructor */
      void resize(int len);       /* Resize particle arrays */
      int newParticle();          /* Add particle */
      /* Remove particle from simulation */
      void removeParticle(int &prtNum);
      /* Class I/O */
      void writeFile();
      void readFile();
      /* Public access to private variables */
      float getX(int &prtNum, int &i);
      void setX(int &prtNum, int &i, float val);
      float getU(int &prtNum, int &i);
      void setU(int &prtNum, int &i, float val);
      float getDt(int &prtNum);
      void setDt(int &prtNum, float val);
      float getTime(int &prtNum);
      void setTime(int &prtNum, float val);

};
```
Figure 2. Example of DOD particle base class.

III. Organization of Code

The design of the new DSMC code must be flexible, allowing for the addition of various physical models in the future with minimal changes to the original code. To achieve this, a base class has been created for each major portion of the code, as described below.

A. Manager

The manager class is the main interface for the new DSMC code. It sets up and creates all other classes and their derived types. It also organizes all input/output operations. Its implementation allows for future development and interaction of various simulation types.

B. Simulation

Currently, the only type of simulation is strictly a DSMC simulation. However, one can imagine future derived simulation classes such as CFD, PIC, or radiation capabilities that can be easily included. Of particular importance in the simulation class is the advanceTimeStep() function. This sequences the move/collide/sample routines typical of DSMC codes. Again, this function can be redefined for derived classes and instructions on how to solve a system of equations can be included in the sequence of instructions.

C. Time Scheme

A separate class to handle simulation timing variables was created to take advantage of different time stepping schemes. There are currently three derived time schemes. The first is that of DAC where all particles are moved and collide at each time step and the time step can vary from cell to cell. The second time scheme follows the scheme that Bird's DS(n)V codes implement[1]. In this scheme, each particle has its own time and time step, as do the cells, and the simulation time is advanced by the minimum flow field time step. The particles are moved such that, on average, the particles in a flow cell are at the same time as the flow cell. Collisions are performed in a flow cell when the flow cell time falls half of the flow cell time step behind the simulation time. The last time scheme is that of Laux[27]. It is similar to Bird's scheme, however each flow cell is assigned a "time zone" that is a multiple of the minimum flow field time step. The time steps are then advanced in the individual flow cells when their "time zone" is processed.

D. Gas Properties

The gas properties class contains all data pertaining to the simulation gas. This includes bulk flow properties, particle weighting information, species information, interaction parameters, reaction information, etc. The base species class stores the species number, charge, diameter, mass, and free stream number fraction. Derived species classes then store rotational, vibrational, and/or electronic data as well as recombination energy and ionization energy where applicable. The interaction class describes how the particles interact with each other (hard sphere, variable hard sphere[28], variable soft sphere[29], etc.). Parameters that depend on the interaction model are calculated within the class and include the mean free path, mean collision rate, deflection angles, collision cross section, and particle relaxation parameters. The reaction class defines reaction parameters and then calculates the probability of a reaction occurring. Current reaction models include the total collision energy (TCE) model of Bird[30] and the quantum kinetic (Q-K) model of Bird[1, 5], but other models can be added in the future.

E. Grid and Geometry

The current grid and geometry classes are structured similarly to that of DAC. The major difference is that they are now organized in terms of OOD. The geometry is represented as a watertight triangulated surface where each surface element is an object containing all pertinent local information. The grid is a two- or three-level Cartesian grid made up of grid cell objects that can be divided into IxJxK children grid cells. The grid cells contain information about flow type, node number (for MPI implementation), limiting coordinates, pointers to children cells (if Level 1 or 2), pointers to flow cells (if Level 3), and any surfaces contained in the grid cell. The grid is also sent particles to be moved from the flow cells so that the flow field does not have to know what type of grid is being used. Additionally, future implementations may include unstructured grids so that the current structured multi-level Cartesian system may be replaced.

F. Flow Field

The flow field keeps track of overall sampled quantities and a pointer to the flow cells. The flow cells organize the particles and buffered particles across the grid cells and retain sampled information. The flow cells have been separated from the grid cells because of the possibility of multiple cut cells in a grid cell. The base flow cell class only allocates memory for the basic geometric information and samples for velocity. Derived flow cell classes then add sampling for rotation, vibration, and electronic energies and then also for electron energy when ionization is being considered. In addition to OOD, the flow cells are designed such that they conform to DOD as well so that the sampling procedures are streamlined.

G. Particles

The base particle class stores the position, velocity, time, time step, species, particle number and last collision partner. Derived particle classes are then defined for combinations of rotational, vibrational, and electronic energy. Charged particles included linked electron velocity. Again, as presented in Figures 1 and 2, the particle class adheres to both OOD and DOD practices.

IV. Comparisons with DAC

Since DAC is the current standard DSMC algorithm used at NASA and many other government and industry locations, it will be used as the benchmark for comparison to the new DSMC code. Great care was taken to make sure that the simulations being run by each code were as similar as possible.

A. Sphere

The first test case was a 0.2 m diameter sphere in either argon, nitrogen (N_2) or air (O_2, N_2, O, N, and NO with 23.7% O_2 and 76.3% N_2) with various simulation flags turned on (see Table 1). For all test cases, the number density was 1.2×10^{20} /m^3, the velocity was 5100 m/s, the free stream temperature was 190 K, and the wall temperature was constant at 500 K. Full accommodation was assumed for collisions with the wall and a fully catalytic wall boundary condition was employed for the single case with chemistry turned on. For all simulations, the bounding box for the flow field was 1.4 m in each direction with 50 cells in each direction. The simulation time step was 2.543×10^{-6} s with a global particle weighting factor of 2.634×10^{14}. Generally, for the simulations of interest discussed above, multiple processors are utilized for faster computation time. However, to focus on

Table 1. List of cases for comparison.

Case	Gas	Collisions	Rotation	Vibration	Chemistry
1	Ar				
2	Ar	X			
3	N_2	X	X	X	
4	Air				
5	Air	X	X	X	
6	Air	X	X	X	X

Table 2. Ratios of time to number of steps and average time step.

	Case S1 (MAP/DAC)	Case S2 (MAP/DAC)	Case S3 (MAP/DAC)	Case S4 (MAP/DAC)	Case S5 (MAP/DAC)	Case S6 (MAP/DAC)
1,000 Steps	0.851	0.818	0.891	0.632	0.772	0.779
5,000 Steps	0.935	0.916	0.968	0.727	0.879	0.881
10,000 Steps	0.992	0.972	1.004	0.768	0.933	0.920
Avg. over last 100 Steps	1.076	1.044	1.058	0.803	1.001	0.965

algorithmic speed comparisons, the comparisons made in this study are on a single processor/single core of a MacBook Pro with a 2.7 GHz Intel Core i7 processor. A single unadapted run of 10,000 time steps (5,000 steady-state samples) was run for each condition and code.

The results of the comparisons are compiled in Table 2. The first three rows of the table list the ratios of time required to get to the specified number of time steps between MAP and DAC. The fourth row lists the ratio of average time step taken over the last one hundred time steps between MAP and DAC. For the single species simulations (Cases 1-3), the performance of the proposed code began around 15% faster than DAC and then, towards the end of the 10,000 time steps, plateaued to around 6-7% slower than DAC. The increased efficiency at the beginning of the simulation is because MAP is structured around the flow cells (Level II cells in DAC) and logic is in place to exclude flow cells without any particles from being processed in the collision and sampling subroutines. However, for the single species cases, the speed comparison of 6-7% slower than the benchmark code is quite favorable considering that an object-oriented code is being compared to a procedural code. Without the DOD considerations, the proposed code was around three times slower than DAC. This clearly demonstrates that the inclusion of DOD is very important when designing a computational code with OOD and the current design of the new code is nearly as efficient as DAC for single species flows.

The real advantage of the proposed code's structure around the flow cell and other DOD considerations is clearly demonstrated when multi-species simulations (Cases S4-S6) are considered. The collisionless air simulation (Case S4) was around 44% faster than the corresponding simulation using DAC. The comparison to DAC becomes somewhat less favorable when internal energy (Case S5) and chemistry (Case S6) are included in the simulation, but the proposed code is still around 15% faster than DAC for these cases. As mentioned above, the increased speed is primarily due to the layout of the particle data and sampling arrays. For each flow cell, all the particle data for the cell is located in contiguous memory for reduced cache misses. The same goes for the flow cell sampling arrays in the sampling subroutine. Again, an OOD code can be quite efficient if DOD considerations are observed while the flexibility and upgradability of the OOD code are still maintained. A comparison of surface pressue and total heat transfer distributions are presented in Figure 3 and free stream

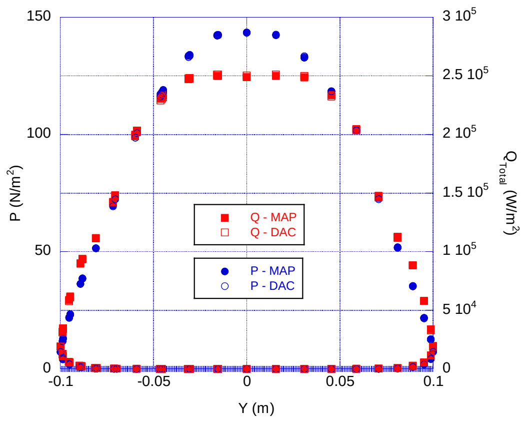

Figure 3. Comparison of surface pressure and total heating for case S6.

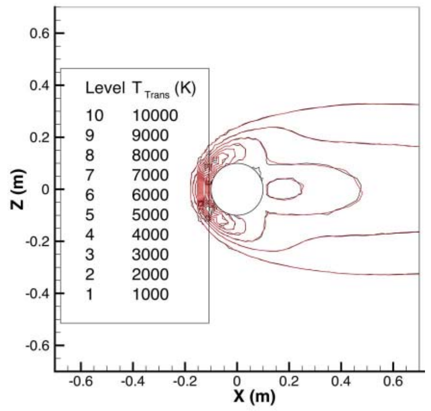

Figure 4. Comparison of flow field translational temperatures for case S6 between DAC (black) and MAP (red).

translational temperatures are presented in Figure 4 for case S6. Considering the coarse nature of the flow field grid, the comparisons are quite favorable.

B. Mars Reconnaissance Orbiter

The Mars Reconnaissance Orbiter (MRO) represents a typical satellite in an aerobraking configuration (Figure 5). This vehicle design is typical of many of the programs that NASA supports. For the current comparison, free stream conditions typical of those at 250 km in Earth atmosphere ($n = 1.96 \times 10^{15}$, $T = 1059$ K, $X_{O2} = 0.0106$, $X_{N2} = 0.2028$, $X_O = 0.7866$) at 6 km/s were assumed with a surface temperature of 300 K. Diffuse surface collisions were assumed with no catalytic recombination.

The results of this comparison once again showcase the DOD nature of MAP. The ratio of the average time step over the last 100 timesteps between MAP and DAC for this simulation was 0.578. The global distribution of the ratio of total heating between MAP and DAC is presented in Figure 6 where most of the variation was under 1%. The differences in heating were above ±1% where the number of samples were limited by either shadowing or small surface triangles. These differences would be expected to decrease as the surface sample size increased. The ratio of drag and pitch coefficients was 1.0004 and 1.0007, respectively.

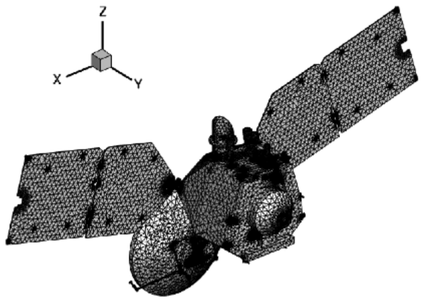

Figure 5. Typical satellite aerobraking configuration represented by the Mars Reconnaissance Orbiter.

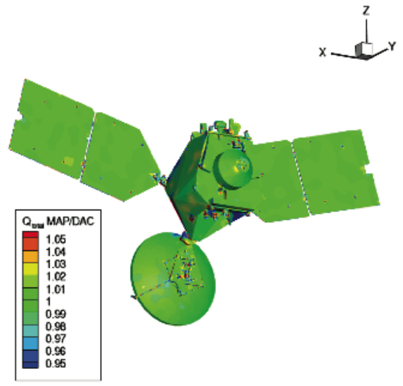

Figure 6. Ratio of total heating rate between MAP and DAC for the Mars Reconnaissance Orbiter.

C. Orion

Another important class of vehicle is the blunt reentry body, exemplified by the Orion Crew Module (see Figure 7). This vehicle class is particularly interesting for demonstrating the advantages of MAP due to the additional physics included in the proposed code. These new features will be discussed in the following sections. For the current comparisons to DAC, the three cases examined can be examined in Table 3 and were selected from Reference [31].

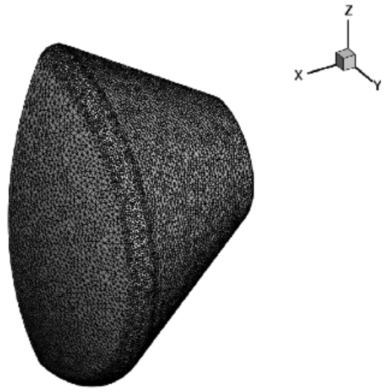

Figure 7. Computational geometry of the Orion Crew Module.

Table 3. Orion Crew Module Simulations.

Case	O1	O2	O3
Altitude (km)	120	105	105
n_∞ (1/m^3)	$5.21\text{x}10^{17}$	$4.98\text{x}10^{18}$	$4.98\text{x}10^{18}$
T_∞ (K)	368.0	211.0	211.0
U_∞ (km/s)	7.6	7.6	12.3
X_{O2}	0.08451	0.15280	0.15280
X_{N2}	0.73271	0.78187	0.78187
X_O	0.18278	0.06533	0.06533
$Kn_{\infty,D,HS}$	0.601	0.0629	0.0629
T_w (K)	567.0	760.0	877.0

Table 4. Ratios of time to number of steps and average time step for Orion Crew Module.

Case	O1	O2	O3
5000 Steps	0.790	0.853	0.824
10,000 Steps	0.794	0.862	0.840
20,000 Steps	0.801	0.874	0.849
30,000 Steps	0.803	0.881	0.852
Avg. over last 100 Steps	0.802	0.904	0.881

Table 5. Ratios of force in the X-direction and pitching moment for Orion Crew Module.

Case	O1	O2	O3
C_X	1.000	1.000	1.000
C_m	1.000	0.999	0.999

Comparisons between the MAP and DAC results are presented in Tables 4 and 5 for simulation timing and forces and moments, respectively. Once again, as presented in Table 4, MAP performs better for the three simulations examined here, ranging from 10-20% faster than the corresponding DAC simulation. Table 5 presents the ratio of the force along the X-axis and the pitching moment between MAP and DAC. The results agree to within ±0.1%.

V. Other Code Features

In addition to efficiency, there are other features that have been included in MAP that are currently unavailable in DAC. The addition of these features was greatly facilitated by the OOD nature of MAP, as it will the implementation of additional physics features in the future. As mentioned previously, the blunt nature of the Orion Crew Module is ideal for exemplifying the additional features in MAP and will be used throughout this section at the O3 free stream conditions listed in Table 3.

A. Internal Energy

It is standard to include both rotational and vibrational internal energy modes in both CFD and DSMC simulations. They are necessary for the accurate computation of fluid dynamics and the resulting forces and heating on vehicle surfaces. However, it has been shown[32] that the inclusion of electronic energy levels[8] is also necessary for flows of sufficient energy, as is already done with CFD codes. Currently only DS1 and MAP include electronic energy levels among DSMC codes.

The timing ratios presented in Table 6 compare the MAP simulation with electronic energy levels included to the MAP solution without electronic energy levels

Table 6. Ratios of time to number of steps and average time step for Orion Crew Module with electronic energy included

Case	MAP/MAP$_{O3}$
5000 Steps	1.120
10,000 Steps	1.116
20,000 Steps	1.117
30,000 Steps	1.115
Avg. over last 100 Steps	1.082

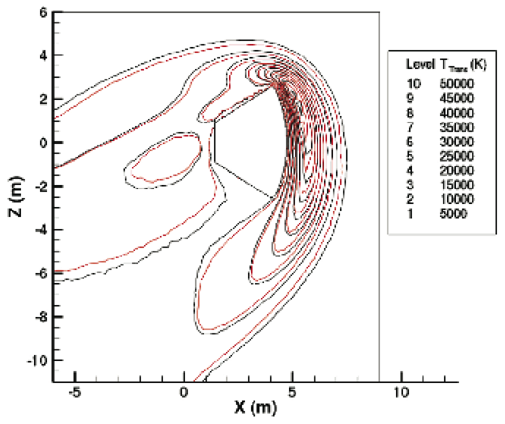

Figure 8. Comparison of translational temperature with (red) and without (black) electronic energy levels included.

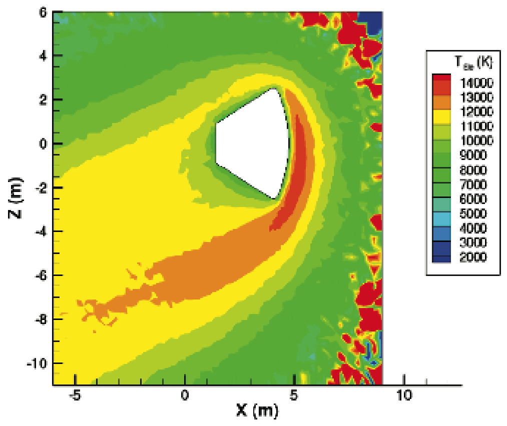

Figure 9. Resultant electronic temperature.

included (Case O3 above) at 105km altitude and 12.3 km/s. The timing results indicate that there is about a 10% speed penalty when electronic energy levels are considered. Figure 8 depicts what happens to the flow field translational temperature when the electronic energy levels are considered. As expected, the addition of the new energy mode has reduced the free stream translational temperature and shock stand-off distance, as shown in Figure 8. The resultant electronic temperature is then as shown in Figure 9.

B. Chemistry Modeling

The most widely used chemistry model used in DSMC is the Total Collision Energy (TCE) model[30] of Bird. This is currently the only model available in DAC and is also included in MAP. More recently, the Quantum-Kinetic model[5] was introduced by Bird and has since been extended by Liechty[8] to include reactions including charged species. The Q-K model is an additional model available in MAP.

C. Electric Field Modeling

The electric field model implemented in the current study is a hybrid of Bird's model[10] and that of Boyd[33]. In this model, the electron particles are moved along with their associated ion during the movement portion of the DMSC algorithm. However, the electrons retain their individual velocity components, which are used in the collision portion of the DSMC algorithm. Neither the electrons nor ions actually experience an electric field, in that their velocity components are not adjusted to account for a field. The resultant flow field electron temperature and molar fraction are presented in Figures 10 and 11, respectively.

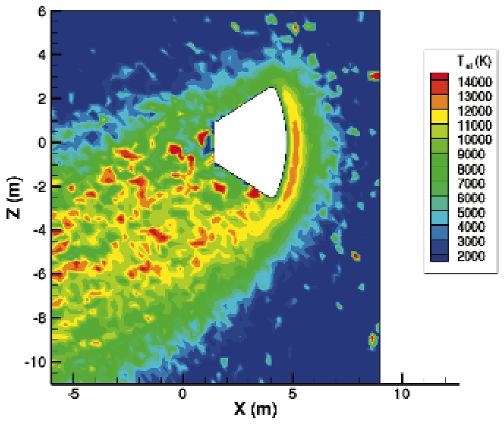

Figure 10. Resultant electron temperature.

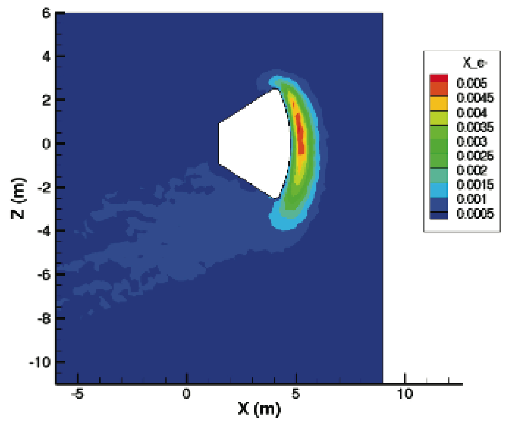

Figure 11. Resultant electron molar fraction.

D. Collision Procedure

The current collision procedure[1] in DS(n)V and DAC does not take into account separate species collision rates. While this should, on average, reproduce the correct species collision rates, a more rigorous treatment of the Boltzmann equation can be obtained by separating the species as described by Nanbu[34]. This collision procedure has been included in MAP and can be chosen as a pre-processing option, however there is a performance penalty since the algorithms must now loop over species as well. However, this penalty for neutral species is negated when ionized flows are considered because of the inclusion of electrons into the flow. The time spent in the collision algorithm decreases significantly in this case because the number of pair selections decreases.

E. Spatial Discretization

As is done in DAC, the initial grid of the proposed code is a uniform Cartesian grid with one or more Level II cells per Level I cell (depending on the prescribed number of Level I cells and the free stream mean free path). The procedure for adapting the grid in DAC is to first obtain a solution on the initial grid, perform an adaptation as a post-processing step, and begin a new solution on the adapted grid. This procedure is repeated until the flowfield is resolved down to the local mean free path (the number of real particles per simulated particle is allowed to vary by Level I cell). The proposed code, however, adapts the grid dynamically. The proposed code currently does not vary the number of real particles per simulated particle, but this is a feature that will be included as it is required for some applications. An example adapted grid can be seen in Figure 12.

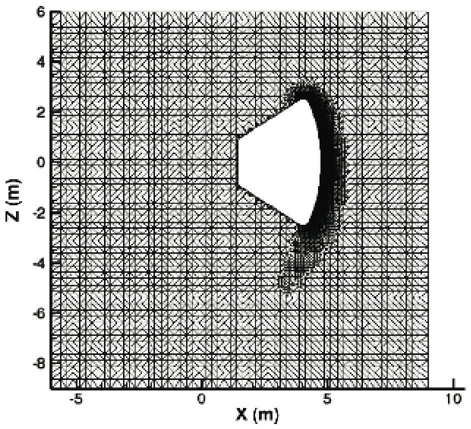

Figure 12. Grid adapted to the local hard sphere mean free path.

F. Temporal Discretization

As mentioned previously, several time stepping schemes have been implemented in the proposed code. These fall into two classes: time accurate and non-time accurate schemes. The time accurate schemes are those of Bird[1] and Laux[27], however the primary development effort has focused on the implementation of Bird's scheme. As in Bird's DS(n)V codes, the local time step is assigned to each individual flow cell and is dynamically adapted as the solution progresses. There is logic included in the algorithm then that determines when the particles are to be moved (based upon the global and local times) and when the particles are to be collided within the collision cells (which are also the flow cells in the proposed code currently).

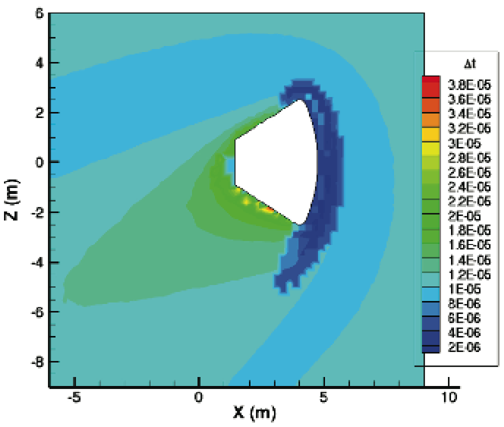

Figure 13. Local time step adapted to the local mean collision time or transit time.

The original time stepping scheme of DAC is time accurate before the first grid adaptation (which occurs upon the completion of the first complete solution), and then is not time accurate as each Level I cell has its own time step and is processed at each global time step increment. The proposed code includes this non-time accurate simulation capability, however the time step is dynamically adapted as the solution progresses and is not limited to the Level I cell but each flow cell can have its own time step (flow cells are assigned to Level II cells). The local time step is based on the minimum of the mean collision time or mean transit time. If time accuracy is not a requirement, the non-time accurate scheme can be significantly faster than the time accurate schemes. An example adapted grid can be seen in Figure 13. The quality of time adaptation, especially near the body, is currently degraded slightly due to the lack of ability to dynamically adapt the number of particles per cell (there are too few particles). As mentioned previously, this will be added in future versions of the code.

G. Parallel Implementation

While MAP was written with parallel implementation in mind, the current study has focused on the serial implementation of MAP and DAC in an attempt to minimize the computation time for each separate algorithm (move, collide, sample, etc.). Future studies will examine the performance of the parallel implementations of the codes.

VI. Conclusion

With the introduction of new DSMC algorithms and the perceived need for a new suite of physics modules, it has been determined that a new production DSMC code for NASA may need to be introduced that has the flexibility to incorporate all of the algorithms and physics currently available and also to expand the code's capabilities in the future. The C++ programming language was chosen for the present study because of its wide acceptance as a scientific, object-oriented programming language. Fundamentals of object-oriented design and data-oriented design were considered in the creation of the new DSMC code to take advantage of both flexibility and speed/memory requirements.

Comparisons have been made between DAC, the current production DSMC code at NASA, and the proposed code, MAP. When comparing single species simulations for a sphere, MAP is between 6-7% slower than DAC. However, the real advantage of MAP's design around the flow cell and other data-oriented design considerations is clearly demonstrated when multi-species simulations are considered. For rarefied (mostly collisionless) flows, MAP is as much as 44% faster than DAC. For simulations where collisions, internal energy and chemistry are considered for the multi-species simulations, the efficiency decreases slightly, but MAP is still 10-20% faster than DAC. Comparisons of free stream and surface properties have shown that the two codes produce the same results.

In addition to the speed comparisons, a list of additional features has been presented for the proposed code. In order to decrease the total time to the final solution, MAP has the ability to dynamically adapt the flow field grid and local time step. MAP also has the option to include electronic energy levels in addition to rotational and vibrational energy levels as well as a charge-neutral electric field modeling scheme for weakly ionized flows. In addition, the recently proposed Quantum-Kinetic chemistry model is also included as an option, as well as the ability to more accurately represent inter-species collisions by separating the species and computing collisions on a species-to-species basis. Although not discussed in this paper, MAP also has the ability to run in parallel computing environments.

With challenging missions on the horizon, such as high-mass payload delivery to Mars and high velocity entry into planetary and satellite atmospheres, the need for expanded capabilities in a rarefied gas dynamics solver is evident. A code that is efficient and easily updated is going to be a requirement. While current, well established DSMC codes still perform the jobs they were originally designed to do well, they have proven difficult to update to meet future challenges. The proposed code MAP has been shown to meet these requirements by making use of both object-oriented and data-oriented design paradigms.

References

[1]Bird, G. A. "The DSMC Method." CreateSpace, Charleston, 2013.

[2]Boyd, I. D. "Conservative Species Weighting Scheme for the Direct Simulation Monte Carlo Method," Journal of Thermophysics and Heat Transfer **10**, (1996).

[3]LeBeau, G. J., Boyles, K. A., and Lumpkin III, F. E., "Virtual Sub-Cells for the Direct Simulation Monte Carlo Method," AIAA 2003-1031, 2003.

[4]Liechty, D. S., "Modifications to Axially Symmetric Simulations Using New DSMC (2007) Algorithms," in *Proceedings of the 26th International Symposium on Rarefied Gas Dynamics*, edited by Abe, T.), pp.

[5]Bird, G. A. "The Q-K Model for Gas-Phase Chemical Reaction Rates," Physics of Fluids **23**, 106101 (2011).

[6]Liechty, D. S., and Lewis, M. J. "Electronic Energy Level Transition and Ionization Following the Quantum-Kinetic Chemistry Model," Journal of Spacecraft and Rockets **48**, 283 (2011).

[7]Liechty, D. S., "State-to-State Internal Energy Relaxation Following the Quantum-Kinetic Model in DSMC," AIAA 2013-2901, 2013.

[8]Liechty, D. S., and Lewis, M. J. "Extension of the quantum-kinetic model to lunar and Mars return physics," Physics of Fluids **26**, 027106 (2014).

[9]Liechty, D. S., and Lewis, M. J., "Extension of a Kinetic-Theory Approach for Computing Chemical-Reaction Rates to Reactions with Charged Particles," in *Proceedings of the 27th International Symposium on Rarefied Gas Dynamics*, edited by Levin, D. (AIP), pp.

[10]Bird, G. A., "Low Density Aerothermodynamics," AIAA Paper No. 1985-0994, 1985.

[11]Carlson, A. B., and Hassan, H. A. "Direct Simulation of Reentry Flows with Ionization," Journal of Thermophysics and Heat Transfer **6**, 400 (1992).

[12]Bartel, T. J., Plimpton, S. J., and Justiz, C. R. "Direct Monte Carlo Simulation of Rarefied Flows on Large MIMD Parallel Supercomputers," *Rarefied Gas Dynamics: Theory and Simulations*. Vol. 159, AIAA, Washington, D.C., 1994, pp. 155-165.

[13]Gallis, M., Prasad, R., and Harvey, J. K., "The Effect of Plasmas on the Aerodynamic Performance of Vehicles," AIAA 1998-2666, 1998.

[14]Bird, G. A., "Nonequilibrium Radiation During Re-Entry at 10 km/s," AIAA Paper No. 1987-1543, 1987.

[15]Carlson, A. B., and Hassan, H. A. "Radiation Modeling with Direct Simulation Monte Carlo," Journal of Thermophysics and Heat Transfer **6**, 631 (1992).

[16]Sohn, I., Ozawa, T., and Levin, D., "DSMC Hypersonic Reentry Flow Simulation with Photon Monte Carlo Radiation," AIAA Paper No. 2009-1566, 2009.

[17]Sohn, I., Levin, D., and Modest, M., "Closely Coupled DSMC Hypersonic Reentry Flow Simulations with Photon Monte Carlo Radiation," AIAA 2010-819, 2010.

[18]Schwartzentruber, T. E., Scalabrin, L. C., and Boyd, I. D., "Modular Implementation of a Hybrid DSMC-NS Algorithm for Hypersonic Non-Equilibrium Flows," AIAA Paper No. 2007-613, 2007.

[19]Burt, J. M., and Boyd, I. D., "A Hybrid Particle Scheme for Simulating Multiscale Gas Flows with Internal Energy Nonequilibrium," AIAA Paper No. 2010-820, 2010.

[20]Jun, E., Boyd, I. D., and Burt, J. M., "Assessment of an All-Particle Hybrid Method for Hypersonic Rarefied Flow," AIAA 2013-1203, 2013.

[21]Wilmoth, R. G., LeBeau, G. J., and Carlson, A. B., "DSMC Grid Methodologies for Computing Low-Density, Hypersonic Flows About Reusable Launch Vehicles," AIAA 1996-1812, 1996.

[22]LeBeau, G. J., and Lumpkin III, F. E. "Application Highlights of the DSMC Analysis Code (DAC) Software for Simulating Rarefied Flows," Computer Methods in Applied Mechanics and Engineering **191**, (2001).

[23]Lumpkin III, F. E., Stuart, P. C., and LeBeau, G. J., "Enhanced Analysis of Plume Impingement During Shuttle-MIR docking using a combined CFD and DSMC Methodology," AIAA 1996-1877, 1996.

[24]Lumpkin III, F. E., Stuart, P. C., and LeBeau, G. J., "The Airlock Depressurization Plume Anomaly on the STS-82 Hubble Servicing Mission," AIAA 2000-462, 2000.

[25]Liechty, D. S. "Aeroheating Analysis for the Mars Reconnaissance Orbiter with Comparison to Flight Data," Journal of Spacecraft and Rockets **44**, 1226 (2007).

[26]Bird, G. A., *Molecular Dynamics and the Direct Simulation of Gas Flows* (Oxford University Press, Oxford, UK, 1994).

[27]Laux, M., "Local Time Stepping with Automatic Adaptation for the DSMC Method," AIAA 1998-2670, 1998.

[28]Bird, G. A. "Monte Carlo simulation in an engineering context," Progess in Astronautics and Aeronautics **74**, 239 (1981).

[29]Koura, K., and Matsumoto, H. "Variable soft sphere molecular model for inverse-power-law or Lennard-Jones potential," Physics of Fluids A **3**, 2459 (1991).

[30]Bird, G. A., "Simulation of Multi-Dimensional and Chemically Reacting Flows," in *Proceedings of the 11th International Symposium on Rarefied Gas Dynamics*, edited by Campargue, R. (Commissariat a Lenergie Atomique, Paris, 1979), pp. 365-388.

[31]Moss, J. N., Boyles, K. A., and Greene, F. A., "Orion Aerodynamics for Hypersonic Free Molecular to Continuum Conditions," AIAA 2006-8081, 2006.

[32]Liechty, D. S., and Lewis, M. J., "Comparison of DSMC and CFD Solutions of Fire II Including Radiative Heating," AIAA Paper 2011-3494, 2011.

[33]Boyd, I. D. "Monte Carlo Simulation of Nonequilibrium Flow in a Low-power Hydrogen Arcjet," Physcis of Fluids **9**, 4575 (1997).

[34]Nanbu, K. "Probability Theory of Electron-Molecule, Ion-Molecule, Molecule-Molecule, and Coulomb Collisions for Particle Modeling of Materials processing Plasmas and Gases," IEEE Transactions of Plasma Science **28**, (2000).