DEVELOPMENT OF AN AMR OCTREE DSMC APPROACH FOR SHOCK DOMINATED FLOWS

BY

SAURABH S. SAWANT

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Aerospace Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Professor Deborah Levin

Abstract

Key strategies used in the development of a scalable, three-dimensional direct simulation Monte Carlo (DSMC) program are described. The code employs an Octree based adaptive mesh refinement (AMR) that gives flexibility in capturing multi-scale physics. It is coupled with a robust cut-cell algorithm to incorporate complex triangulated geometries. With the use of distributed memory systems and Message-Passing-Interface (MPI) for communication, the code is potentially scalable. However, to simulate continuum-like conditions involving multi-scale physics, better scalability that has yet been achieved is desirable. The thesis identifies two main performance bottlenecks in simulating at continuum-like conditions, first, improving the scalability of the code for more than 128 processors by reducing the communication and evenly balancing the computational load, and second, improve the algorithmic performance of the code by eliminating the expensive recursive tree traversal inherent in Octree based mesh structure. In order to resolve the first issue sophisticated graph-partitioners have been used, however, without success. The thesis also explains the special considerations required for embedded geometries in a parallel computational environment. An efficient algorithm is discussed that allows for the checking of particle-surface interaction only if they are close enough to the geometry. The code calculates various surface coefficients and employs the Borgnakke-Larsen continuous relaxation model to simulate inelastic collisions of diatomic molecules. Finally, these strategies and models are validated by simulating hypersonic flows of argon and nitrogen over a hemisphere and double-wedge configuration and the solutions are compared with the results obtained from an older DSMC code known as SMILE.

"When you have eliminated the impossible, whatever remains, however improbable, must be the truth."

-Sir Arthur Conan Doyle

Acknowledgments

I have had the distinct privilege to work with my adviser Professor Deborah Levin and would like to express my sincere gratitude for giving me this opportunity, her wise guidance, and valuable lessons she taught me during the course of this work. I am greatly indebted to my colleague Burak Korkut for his mentorship, encouragement, and motivation since my first day in the lab. The material of this work has benefited from immense help and valuable comments from my colleagues Ozgur, Burak, and Dr. Zheng Li. I would like to express my gratitude towards my friends and lab mates Revathi, Neil, and Tong for all the intellectual and fun discussions.

I will be forever grateful for having a mother who always believed in me, a father who supported me through thick and thin, a sister and aunt who motivated me in tough times, and my loving late grandmother who was very proud of my achievements.

Finally, I would like to acknowledge the sponsor, Air Force Office of Scientific Research, and Blue Waters sustained-petascale computing facility, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois, for providing us with valuable resources.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

SUGAR	Scalable Unstructured Gasdynamic Adaptive mesh Refinement.
DSMC	Direct Simulation Monte Carlo.
AMR	Adaptive Mesh Refinement.
MPI	Message-Passing-Interface.
OOP	Object Oriented Programming.
C-Mesh	Collision Mesh.
V-Mesh	Visualization Mesh.

List of Symbols

$FNUM$	Number of molecules represented by one simulated molecule
ρ	Mass density
ε_r	Rotational energy of a particle
ε_v	Vibrational energy of a particle
m	Molecule mass
m_r	Reduced mass
ω	Temperature exponent of the coefficient of viscosity
α	$\omega - 0.5$, accommodation coefficient
ζ	Number of internal degrees of freedom
ζ	total number of modes (translational and rotational) that are participating in the distribution.
C_r	Relative speed of two particles selected for collision
Z_r^C	Rotational number to used in continuum methods
Z_r	Rotational number to used in DSMC method
$Z_{r,\infty}$	Constant used in Parker's formula
T^*	Constant used in Parker's formula
R_f	Uniformly distributed random number between zero and one
N_{hit}	Number of collisions of particles with the surface
S_p	Surface area of the panel
$\vec{n_p}$	Surface normal of the panel
Θ	Angle between the velocity vector and the normal to the surface
$\vec{V}$	Velocity of particle incident onto the surface
α_t	Tangential accommodation coefficient
α_e	Energy accommodation coefficient
ΔT	Time used to collect the information about collisions normal to the surface
δT	Time step

Indices used in collision models

i	Parameter referring to the pre-collisional state of the particle
p	Parameter referring to the post-collisional state of the particle

Indices used in surface coefficient calculation

i	Parameters of the particle incident onto the panel
r	Parameters of the particle reflected from the panel
xyz	Projection of the corresponding particle axis
∞	Free-stream parameters

Chapter 1

Introduction

1.1 Introduction

The steep gradients of macroscopic variables that characterize the hypersonic flows question the validity of the assumptions made in the formulation of the Navier-Stokes equations. On the other hand, the Boltzmann Equation of Transport provides a generic mathematical model which can be applicable to the entire range of Knudsen number. The Direct Simulation Monte Carlo (DSMC) method is a widely used particle based probabilistic approach that is derived based on the same physical reasoning as the Boltzmann transport equations [1]. Applications of this approach range from flows in the space environment to vacuum processes where the characteristic length of the flow is of the same order as the mean free path of the molecules. In this method, deliberate attention must be paid to key factors such as the ratio of the number of simulated molecules to real molecules which should be sufficient to reduce statistical scatter, cell size which should be approximately one third of the mean free path, and time step which should be approximately one fifth of the mean collision time so as to justify the decoupling of molecular movement and collision processes. Following all of these criteria everywhere in the domain is the key to obtaining an accurate as well as computationally efficient solution. However, many rarefied flows of practical interest exhibit orders-of-magnitude variation in density as a result of the high flow velocities involved, which necessitates the need for efficient computational grids that can resolve such multi-scale physics efficiently.

Existing DSMC codes employ a variety of meshing concepts. The SMILE [2] (Statistical Modeling In Low-Density Environment) system uses a two level Cartesian grid, meaning that each level one cell can be refined into any number of level two Cartesian cells in each direction, given the number of level of refinement before starting the simulation. It employs a cut-cell method for simulating flows over geometries. NASA's DSMC Analysis Code (DAC) [3] similarly uses a two-level rectilinear grid which can be adapted based on the solution at a previous time step. It also employs a cut-cell method that enables it to employ arbitrary triangulated surface mesh. The Molecular Gas Dynamics Simulator [4] (MGDS) code uses an adaptive mesh refinement (AMR) to refine up to three levels of Cartesian grid and employs the same cut-cell method imple-

mented in DAC. The MONACO code [5] uses unstructured body-fitted quadrilateral or tetrahedral meshes. A very recent open-source implementation of the DSMC method known as dsmcFOAM in OpenFOAM system can handle unstructured polyhedral meshes. The present work describes the algorithms and strategies used in creation of a new DSMC implementation called Scalable Unstructured Gas-dynamics Adaptive Refinement (SUGAR) [6], which builds upon the aforementioned implementations by inclusion of multiple Octree based AMR meshes and a new generation of hybrid parallelization strategies applicable to heterogeneous computer systems. The Cartesian grid has been chosen over tetrahedral meshes because it takes substantially less memory to store the cell structure information. Fully automated AMR allows flexibility for capturing high gradients in hypersonic flows without excess grid refinement of the entire domain. It employs a robust cut-cell method that allows simulation over complex geometries. The SUGAR code makes efficient use of distributed systems and will be shown in this thesis to scale linearly up to 128 processors. For 512 processors, a maximum speed up of 335 was achieved for a case of hypersonic flow over a double-wedge and 358 for hemisphere. When geometries are embedded in a MPI parallelized code, the DSMC methods such as particle reflection over the surface and heat flux calculation need careful consideration. The thesis describes efficient ways to deal with these problems. In addition to that, the code also employs Borgnakke-Larsen (BL) model for simulating inelastic collisions of diatomic species such as nitrogen.

The SUGAR code has been applied to the study of three dimensional simulation of ion thruster plumes by Korkut et al. [6] In this work, we modify it to simulate hypersonic flows involving argon and nitrogen over a hemisphere and double-wedge configurations for a series of Knudsen numbers and compare with the results obtained from the SMILE code by Mr. Ozgur Tumuklu. After successful validation of thermal non-equilibrium models for flow over a sphere, the goal was to apply it to a case involving the study of laminar, shock-shock interactions from hypersonic flows about a double-wedge configuration in a continuum-like environment.

The latter has been a challenging problem because of the multiple shock-shock and shock-boundary layer interaction, separated flows near the hinge, sheer layer, and three-dimensional effects. Experiments done by Swantek et al. [7] have been simulated and compared using the SMILE code by Patil et al. [8] on a two dimensional wedge geometry and efforts to simulate three dimensional results using the SMILE code have been reported by Tumuklu et al. [9]. Yet the problem is extremely difficult and multi-scale when the Knudsen number is 0.0002. In order to resolve the circulation region near the hinge and capture the boundary layer, billions of particles are needed in the simulation. Therefore, driven by the purpose of having a highly scalable and efficient DSMC code the development of the SUGAR code was undertaken and this work describes various strategies that have been implemented so far by careful verification with the existing

SMILE code. The final ability to numerically simulate these complexities would greatly reduce the efforts in analyzing the aerothermodynamics of such flows by accurately predicting the pressure loads, heat transfer rate, and skin friction in the design of future hypersonic vehicles. The work of this thesis has been presented as an AIAA conference paper. [10]

Although the code uses sophisticated techniques and gives a speed up of approximately 350 for 512 processors, it needs to be made more scalable and computationally efficient in order to tackle the main problem. Therefore, the code was profiled using the Cray Performance Measurement and Analysis Tools (CPMAT) [11] on the BlueWaters peta-scale facility. It was revealed that one of the main bottlenecks while using thousands of processors is communication between the processors and imbalance in the amount of computation performed by each processor. The use of graph partitioners such as PARMETIS or SCOTCH to get a better distribution of load while minimizing the communication was attempted, but, there was no improvement in performance. The algorithmic performance of the SUGAR code was also compared on a single processor and based on these results the time required to run the actual case is predicted.

1.2 Thesis Outline

The thesis is organized as follows: Chapter II describes the AMR strategy, the parallelization strategies and an extensive discussion of performance capabilities of the SUGAR code, preliminary scalability tests, a profiling study, the results of using graph partitioners, algorithmic assessment of the SUGAR code, a robust cut-cell algorithm, re-consideration of the reflection subroutine in an MPI parallelized domain and its optimization, surface flux computation, and the continuous rotational relaxation model. In Chapter III, the results for a nitrogen heat bath case and 3-D simulations of hypersonic flows of argon and nitrogen over a hemisphere, and the double-wedge configuration are presented and compared with the SMILE DSMC code. Chapter IV reports the conclusions drawn from the simulations and explains the future efforts in brief.

Chapter 2

Development Strategies

2.1 Adaptive Mesh Refinement

The SUGAR code uses a grid approach known as Adapive Mesh Refinement (AMR) combined with the Octree method which has unique capabilities for obtaining efficient solutions for problems that have multi-scale properties and large gradients in their computational domain. This approach is being widely used in many diverse scientific branches from astrophysics [12] to biology [13] following the pioneer by Berger [14]. In addition, there is a recent interest in the use of AMR within DSMC context for aerospace applications. [15, 16]

The motivation of using the Octree storage in hexahedral (i.e. polyhedron with six faces) meshes for cell connectivity lies in its efficient data storing as well as straight-forward adaptive refinement capabilities. In tree structures, of which Octrees are a subset, each node has 2^d children where d represents the number of dimensions present in the problem and for this reason binary, quad, and Octree algorithms are analogous to each other. For three dimensional cases, a node is divided into eight children nodes. Describing the terminology briefly, every node that has children is called a parent. It is apparent that in this tree structure a node can simultaneously be a parent and a child. A node without any parent is called a root and a node without any child is called a leaf. A schematic description for a level two AMR grid is summarized in Fig. 2.1. Initially, a computational cell is at a root position. If it violates the given refinement criterion which could be that the maximum number of particles cannot exceed more than eighty, then it gets divided in eight parts along its center point. If any of its children violate the criterion then they are subdivided further. In this figure, the fifth child has been subdivided. When this recursive process stops, we get the final mesh which is composed of cells of different sizes. These cells are the leaves of the Octree in which the DSMC procedure is performed. Although shown in three separate stages, the parent and its children occupy the same space in the computational domain, so flags are introduced for each node to distinguish if it is active or inactive. This guarantees that the computational domain is taken up by unique nodes which only share boundaries or surfaces with each other.

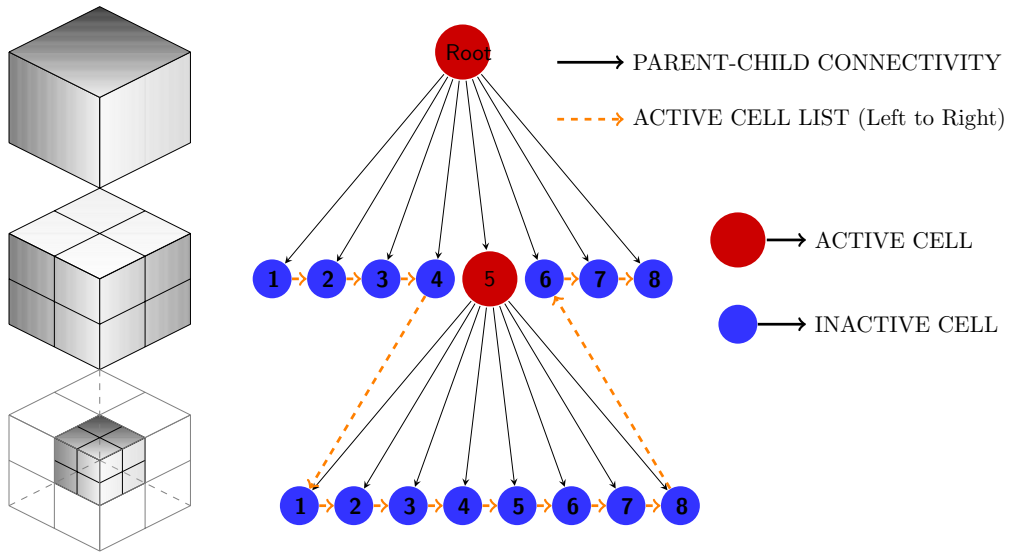


Figure 2.1: Computational mesh and corresponding Octree structure.

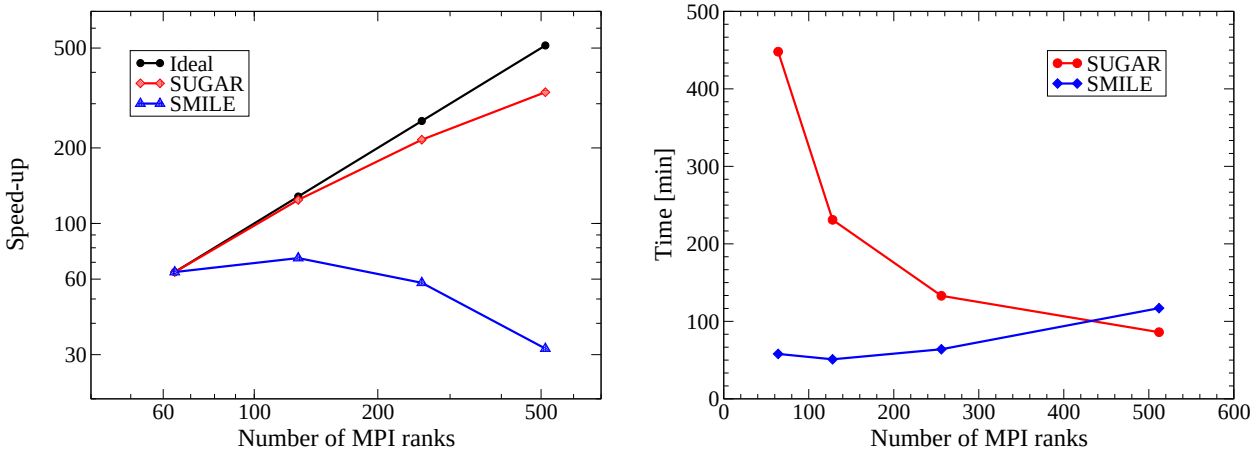
Previous work focusing on the modeling of ion thruster plumes [6] included the usage of a single AMR mesh to carry out the computations related to collisions, electric field, and visualization. Now, the SUGAR code employs a more realistic approach that separates this single AMR mesh into three individual meshes. The current work, only required two of these meshes since the species are all neutral. The use of object oriented approach makes it easy to switch on or off any of these meshes and each mesh is allowed to be adapted individually based on the given refinement criteria. The first of these meshes is named the collision mesh (C-Mesh) where computational particles are mapped to cells to model the collisions occurring in the flow using the DSMC method. The second mesh, called the visualization mesh (V-Mesh), is used to sample the flow field parameters such as density, velocity, etc. over time and visualize it. Both of these meshes are independent of each other.

The criteria of refinement is important for these meshes. For accurate simulation using the DSMC method, it is necessary that the cells in which collisions are performed be smaller than the local mean-free-path. Thus, the refinement criterion for the C-Mesh is that the local Knudsen number cannot exceed one and the number of particles cannot be less than four to simulate a binary collision. If chemical reactions are present then the number of particles per collision cells should be significantly higher. For the V-Mesh there is no physically enforced criteria as such but, the only criterion is to have enough particles per cell to correctly represent the macroparameters with small statistical scatter. For this reason, the V-Mesh cells are coarser than C-Mesh cells. [15, 2] In this work, the criterion for subdivision was such that each cell would have 30-40 computational particles in each V-Mesh cell which gave a smooth profile for flow field variables.

2.2 Parallelization and Scalability

The SUGAR code uses Message Passing Interface (MPI) to harness the computational power of modern computing architectures for parallel computing. It is developed in C++ to fully make use object oriented programming (OOP) paradigms in order to minimize the effort needed to develop, maintain and extend the capabilities for a large scale effort. The parallelization is achieved by using domain decomposition for the computational domain and using MPI for communication between the processors. The domain is decomposed across the underlying coarser Cartesian grid. These Cartesian cells are the roots of the Octree and are referred to as the Root-cells. For the current work 2-D blocking algorithm has been used to partition the domain, although, testing was performed using graph partitioners, as will be discussed in Sec. 2.2.4.

2.2.1 Scalability study I



(a) Speed-up comparison.

(b) Time comparison.

Figure 2.2: Performance study of the SUGAR code using the double wedge case.

To test the scalability of the SUGAR code, the case of argon flow over a wedge was chosen which will be described further in Chap. IV as Case-II. The flow properties are the same as detailed in Table 3.2 except for the FNUM which is taken as $5.0 \text{x} 10^{10}$ to give around 25 million particles in the domain. Figure 2.2(a) shows the comparison of speed-up for the SUGAR code using a 2-D blocking algorithm. It can be seen that for up to 128 processors the SUGAR code generates linear scalability. For 256 and 512 processors, a deficiency is observed due to the fact that the computational load for each processor becomes uneven in terms of the number of computational particles and AMR cells. It can be seen that the maximum speed up obtained for 512 processors is 335. On the other hand, the SMILE code does not scale beyond 64 processors which is a major disadvantage when the task is computationally expensive. Because of the lack of scalability

of the SMILE code, the time taken by 256 and 512 processors is in fact higher than that was taken by 64 processors. The main conclusion from this study is that the SUGAR code is potentially scalable which would be helpful for problems that demand high computational resources.

Figure 2.2(b) shows the time taken using different number of processors. The main reason that the SUGAR code to takes more time is the high level of refinement of the C-Mesh. The way particles are located on the mesh is by a method called mapping where a particle is sorted recursively into a leaf cell starting from its root. This procedure has to be performed at each time step for each particle in the domain. Thus, higher level of refinement entails more computational time per time step. Figure 3.6 shows the C-Mesh mesh where it is observed that the Octree cells after the shock are very refined and the depth of refinement increases in the vicinity of the surface where the number density is high. Thus high number of particles are mapped on the finer mesh.

2.2.2 Scalability study II

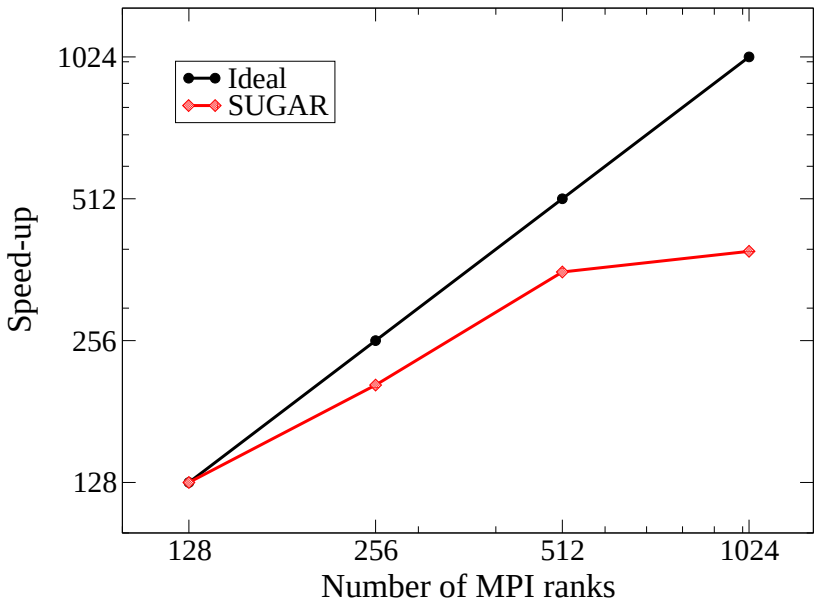


Figure 2.3: Speed-up plot for the flow over Hemisphere (Case-IV).

Another speed-up study was performed after the implementation of computations that includes surface flux coefficients and rotational relaxation. This study was performed for a hypersonic flow of nitrogen over a hemisphere as will be described in Chap. IV as Case-IV by incrementing the number of processors from 128 to 1024. It was assumed that the code gives linear speed-up up to 128 processors which is a reasonable observation based on the previous study. Examinations of Fig. 2.3 reveals that for more than 128 processors,

the speed-up is again less than ideal because of the load imbalance, giving the maximum speed-up of 358 for 512 processors. Speed up for this case is a bit better because a relatively smaller portion of the domain is highly refined as compared to the double-wedge case thus causing less load imbalance. Beyond 512 processors, the speed-up profile flattens and it is clear that the 2-D blocking algorithm is not suitable for high number of processors. In order to understand this behavior and locate the cause of poor performance, a preliminary profiling study was performed over a double wedge case using 512 processors with the Cray Performance Measurement and Analysis Tools (CPMAT) [11] profiler.

2.2.3 Profiling using CPMAT

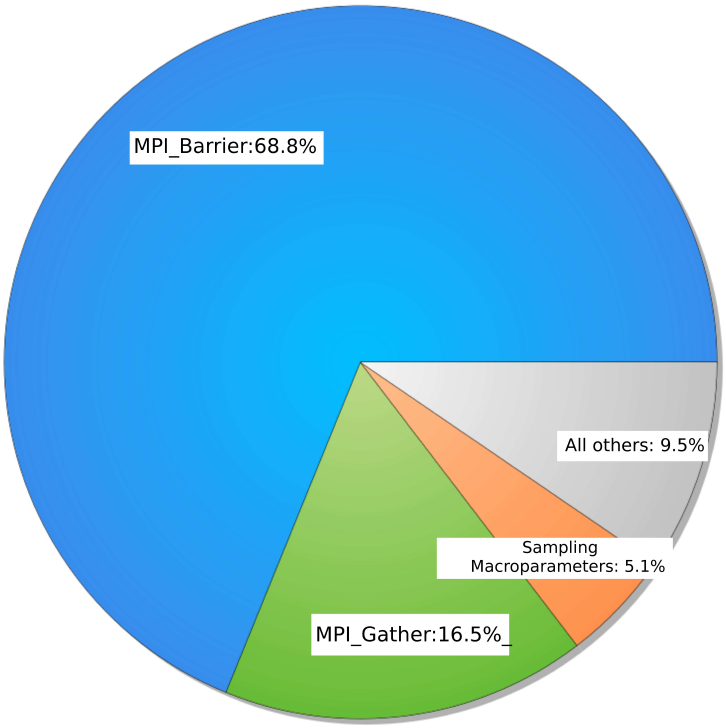


Figure 2.4: Percent time spent in various processes.

The profiling was done using Automatic Performance Analysis (APA) [11] strategy which traces all the subroutines in the code and gives a primary estimate of the time consuming functions in the code that take the most amount of time. Then the code is run again by specifically tracing those subroutines for an accurate estimation of the overall time consumed. Figure 2.4 shows the percent time taken by the time consuming routines. It was found that the calls to MPI routines are computationally more expensive. The MPI_Barrier function takes 68.8% of the time which is used to synchronize all the MPI processes after putting the particles

in their respective cells. Next to that is 'MPI_Gather' which is used by the root processor to gather the information from other processors during communication stage. Among user defined functions sampling of macroparameters takes the most time as expected.

Figure 2.5 shows percent time (Y-axis) spent by each processor, denoted on the X-axis, in performing various activities. Yellow color denotes percent time spent in performing user defined functions which varies a lot from processor to processor. The processors that show a spike in the yellow color are sitting idle while other processors perform their task. In summary, this plot shows the wide disparity between the time spent by each processor which ideally should be equal. The main reason for these differences is multi-scale nature of the problem where the density in different regions of the domain varies by orders of magnitude. This problem is worsened by the inclusion of the surface geometry where the processors occupying regions inside the geometry have very few or no particles and they sit idle until those having the most number of particles finish their task. Therefore, equal division of workload is one of the important tasks in order to make the code computationally efficient. Furthermore, for particle based methods this division should be such that each processor should get the Octree cells that are connected to each other at boundaries in order to reduce the inter-processor communication. A manual way of assigning all the processors that lie inside the geometry to a single processor was tested but it has many limitations and did not guarantee good load balance. Therefore, sophisticated graph partioning tools were investigated as a possible approach to solve this problem.

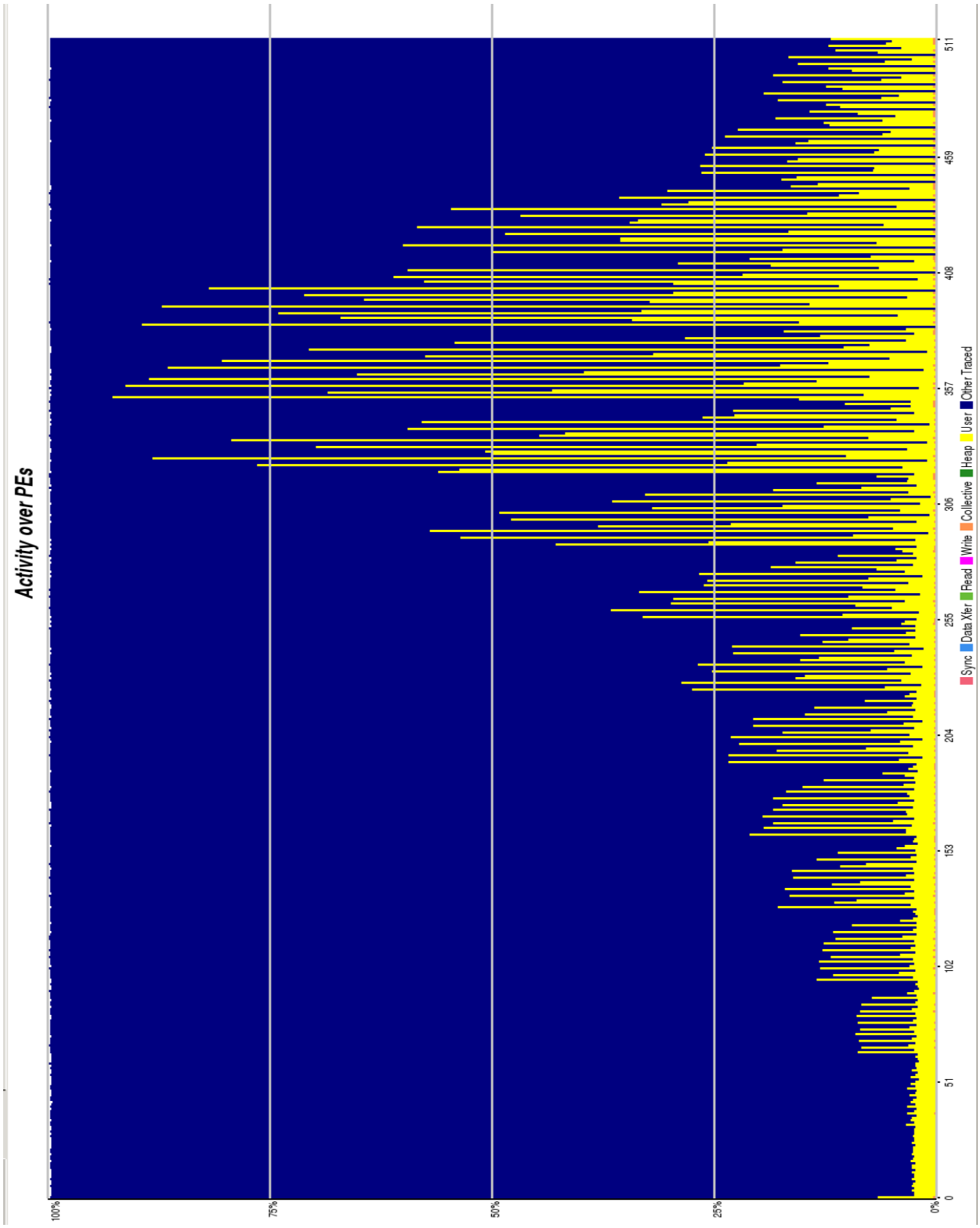


Figure 2.5: Activity plot for 512 processors.

2.3 Graph Partitioning for Potentially Improved Load Balancing

2.3.1 Brief Introduction of the Load Balancing Problem

A parallel program is composed of many processes, each of which performs a number of tasks defined in the program. A task is the smallest unit of concurrency that a parallel program can exploit. A process is an abstract entity that executes its assigned tasks on a processor. [17] First, the tasks are divided among processes and processes execute the tasks on processors. By definition, this step of decomposing the tasks and assigning it to processes is called partitioning and the step of assigning processes to processors is called mapping. The objective of partioning is to divide the tasks equally among processes and reduce the interprocesses communication whereas that of mapping is to balance the workload of processors and minimize inter-processor communication. The number of processes may not be equal to the processors. Therefore, a processor can sit idle or be loaded with processes. For many problems including the one in hand, the load balancing problem is viewed by the process-processor model. [17]

The load balancing algorithms are categorized into static and dynamic. In static load balancing, processes are mapped to processors at compile time, therefore, they need to be provided with a priori estimated knowledge of execution time of processes and interprocess communication. However, dynamic load balancing aims at distributing the workload among processors and minimizing inter-processor communication costs at run time. Therefore, they are also called remapping algorithms.

To formulate our DSMC method in terms of a load balancing problem, consider each portion of the domain divided among a certain number of processes which can be thought of as root level Octree cells. Since, particles move during the simulation, the workload of these processes changes. Therefore, the program behavior is dynamic and cannot be predicted a priori. Furthermore, these processes have to be synchronized at many instances in a time step, and an imbalance in their workload increases the idle time for some processes and consequently, the processors that occupy them. In order to improve the efficiency of the code, the workload of all processors have to be redistributed during the run-time. This can be done by using a dynamic load balancing algorithm that distributes the Octree cells among processors in such a way that each processor has nearly equal amount of computation to perform and each processor contains contiguous blocks of cells such that the inter-processor communication is minimal. To assess the amount of workload of each processor, there should be a load imbalance index that quantifies the workload of each processor relative to others. It is computed using the weighting factor for each Octree cell which essentially represents its computational load. Weighting factor could be a function of the number of particles or the number of children cells in an Octree. The Octree cells having the highest number of children cells would have

higher weighting factors which suggest that these processes are computationally expensive. In addition to the weight assigned to an Octree cell, information such as the number of neighbors of that Octree, the IDs of those neighbors, and coordinate location would be provided to the load balancing algorithm which would then output a better distribution of these processes among available processors.

Devising a load balancing algorithm is a separate area of research altogether, however, there are well established libraries such as Parmetis [18] and Scotch [19] that make use of state-of-the-art load balancing algorithms such as diffusion algorithm or multilevel Kernighan-Lin partitioning. The Zoltan library [20], developed at Sandia National Laboratory, is an extensive toolkit of such algorithms to solve problems such as dynamic partitioning, graph coloring and ordering. The specialty of Zoltan is that it links with Parmetis and Scotch and with minimal modifications in the Zoltan parameters, one can test a variety of algorithms and use the one that gives better results. Furthermore, it is not only limited to graph problems but also includes some geometric and hypergraph partitioning methods. Among these, the geometric partitioning methods could be particularly useful which are based on geometric locality such that objects that are physically close to each other are assigned to the same processor. These methods are faster than graph partioning methods but the quality of the outputted graph is not as good as the latter. Therefore, it could be used where frequent load balancing is necessary and a high level of accuracy is not an issue such as the cases where we have unsteady simulations. Table 2.1 summarizes the methods in Zoltan.

Table 2.1: Graph Partitioning Methods Available in Zoltan.

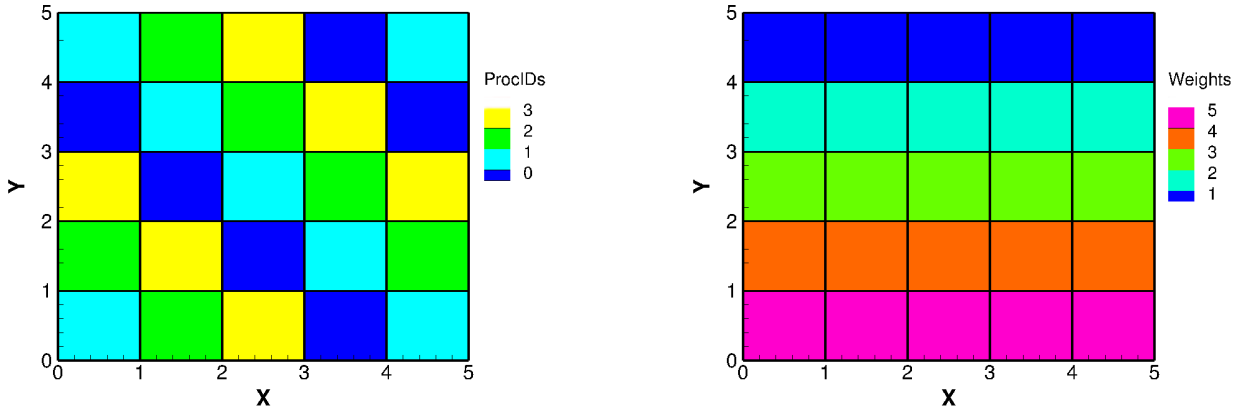
Geometric (Coordinate-based)	**Graph Partitioners**	**Hypergraph Partitioners**
Recursive Coordinate Bisection (RCB)	PHG*	PHG*
Recursive Intertial Bisection (RIB)	Scotch (RBISECT)	PaToH
Space Filling Curves (Hilbert**)	ParMETIS (5 Various Methods)	
Refinement Tree based partitioning		

* PHG is a Parallel Hypergraph Partitioner available in Zoltan which can also be used for graph partitioning.

** By some modification in the source code it can be applied to any Space filling curve, eg. Morton-Z. [21]

2.3.2 Preliminary Tests Using the Zoltan Framework

Zoltan distribution comes with some sample problems. [21, 22] In order to understand the functionality of Zoltan, one of such problems was modified by changing various Zoltan functions and parameters and



(a) Initial Processor Distribution.

(b) Weight Distribution.

Figure 2.6: 2D decomposition problem.

applied to a problem involving a square domain divided into five cells in each direction, respectively. In an Octree DSMC simulation, these cells would represent the Octree cells. Initially these cells are randomly assigned to the given number of processors as shown in the Fig. 2.6(a). In all, 25 nodes are divided into four processors having processor IDs 0, 1, 2, and 3. The cells having the same color belong to the same processor. Since the cells are randomly assigned to processors, a processor contains cells from different regions. This is unfavorable in the DSMC simulation since when particles move from one cell to other, that particle data has to be communicated to the processor which contains the destination cell. Theoretically, a favorable distribution is the one where the same colored nodes are closer to each other. Furthermore, in multi-scale problems, the distribution of number of particles varies across the domain and the amount of computation is not uniform across the Octree cells. Therefore, the complexity of this toy problem is increased by assigning to each cell a specific weight which can be considered as a relative representation of the amount of computation. In the current problem, weights are assigned using the following formula,

$$weight = int(6 - Y_i) \tag{2.1}$$

Y_i is the Y-coordinate of the center of the i^{th} node and the int function gives the integer part of the argument. The weight distribution is shown in the Fig. 2.6(b). In the DSMC method, the weight for each Octree would be a function of the number of children cells in it. Now in addition to the aforementioned favorable distribution, the partitioner must make sure that each processor gets equal amount of weight.

In order to reassign the cells to processors, each processor has to provide the IDs of its cells, the IDs

of the neighbors of those cells, and the weights on those cells to Zoltan functions and specify parameters listed in Table 2.2 to specify the conditions of partitioning. The rest all parameters are kept to its default value and one should refer to the Zoltan manual [21] for more information. The best output from the graph partitioning algorithm would be the one where the cells are reassigned to the given processors in such a way that all processors get approximately equal amount of weight and the inter-processor communication is minimum. When two adjacent cells belong to different processors, they form a cut edge, which represents a communication link between the two processors. Thus, the job of a partitioner is to reduce the number of these cut edges in order to reduce the communication. In this work, three methods have been applied to the 5x5 problem described above: 1. PartKWay method based on multilevel Kernighan-Lin Partitioning, 2. PartGeomKWay method which fuses PartKWay and PartGeom method, which is a geometric partitioning method that prioritizes the locality of nodes, and 3. Scotch using recursive bisection to compute k-way partitioning. The final output from these methods is shown in the Fig. 2.7.

Table 2.2: Zoltan Parameters.

Zoltan Parameters	**Value**
LB_METHOD	GRAPH
LB_PARTITION	PARTITION
NUM_GID_ENTRIES	1
NUM_LID_ENTRIES	1
GRAPH_PACKAGE	SCOTCH or PARMETIS
PARMETIS_METHOD (only for Parmetis)	PARTKWAY or PARTGEOMKWAY
OBJ_WEIGHT_DIM*	1
EDGE_WEIGHT_DIM	1
ADD_OBJ_WEIGHT†	PINS
GRAPH_SYM_WEIGHT	ADD

* If the weight dimension is one but weights are not specified then equal weight is assumed for each object.

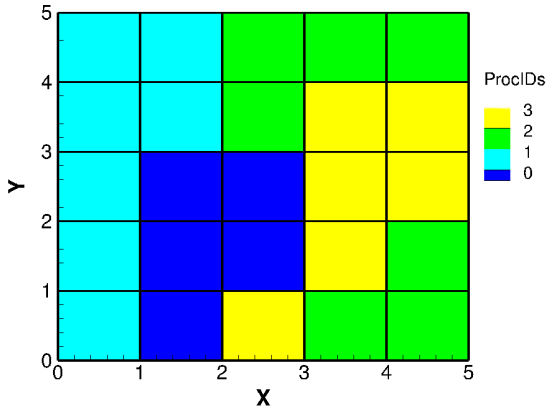
† For Scotch ADD_OBJ_WEIGHT=NONE.

It is clearly seen in Fig. 2.7(c) that the output from the Recursive Bisection method in Scotch very well satisfies our first criterion where each processor has contiguous blocks of cells thus having minimum communication links. The output of PartKWay as well as PartGeomKWay methods in Parmetis do have

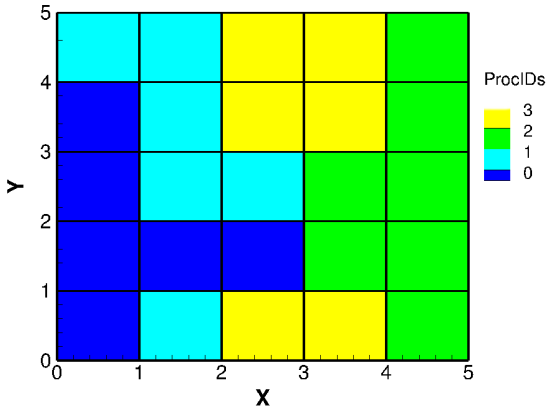
separated regions of domain occupied by the same processor as shown in Fig. 2.7(a) and Fig. 2.7(b), respectively, which does not guarantee geometric locality. These results are quantified in terms of the number of cut edges in Table 2.3. Initial random distribution gives 40 number of edges which is reduced to 11 by using Scotch. Moreover, to quantify the measure of equal weight distribution, a load imbalance index is used to calculate the imbalance of each processor using the following formula,

$$Load\ Imbalance\ for\ i^{th}\ Processor = \frac{Total\ Number\ of\ Processors * Weight\ of\ ith\ Processor}{Total\ Weight} \tag{2.2}$$

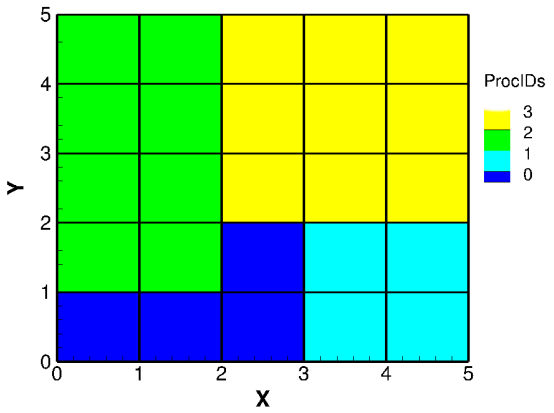
An imbalance index of one means that there is no imbalance, ie., the load is perfectly balanced. An index higher than one would mean that the load distribution is non-uniform and some of the processors would have to sit idle until those having more computation finish their work. For the 5x5 graph problem, the total number of nodes are 25, and total weight count is 75. Using these values, the load imbalance index is calculated for each processor, and the worst index is listed in Table 2.3. Nearly equal values show that each method assigns nearly equal amount of work load to each processor.



(a) Parmetis PartKWay Method.



(b) Parmetis PartGeomKWay Method.



(c) Scotch Recursive Bisection method.

Figure 2.7: Output of 2D decomposition problem.

Table 2.3: Results of the 5x5 graph problem.

Method	**Worst load imbalance index**	**Cut edges**
Parmetis (PartKWay)	1.01	18
Parmetis (PartGeomKWay)	1.03	19
Scotch	1.06	11

Next the problem is extended to three dimensions and the procedure is repeated for a cube of 16 cells using 256 processors having the same weight distribution formula 2.1. For this problem, the total number

of nodes are 4096, the total weight count is 34,816 and before partitioning the number of cut edges are 7680. The output is quantified in the Table 2.4 which shows that the load imbalance index is nearly equal meaning that all three methods distribute the work load nearly equally among 256 processors. To make an estimation of benefits of these algorithms over the current 2-D blocking algorithm, this graph problem was also solved using 2-D blocking strategy. Note that this algorithm does not take into account the weights of the cells and hence, performs poorly if the amount of work load of each Octree cell differs significantly. A 2-D blocking algorithm gave the number of cut edges equal to 7680 and minimum and maximum load imbalance indexes were 0.11 and 1.88. First, looking at the number of cut edges and comparing with the values listed in Table 2.4, it is observed that each method gives output that would require less amount of computation. Especially, Scotch would require 36.4% less communication. Also as compared to Parmetis, Scotch gives approximately 32% reduction in communication links. Second, comparing the maximum load imbalance index of graph partitioning algorithms and a 2-D blocking algorithm, the former methods assign the work load that is close to idea. By these observation, one can make an estimate of the potential reduction in computational time if a DSMC problem is constructed with the same 16 cube Octree domain and run with 256 processors. Assume that the MPI ranks would have to synchronize just once which contributes to 68% of a time step and there is just one instance of communication which contributes to 16% of a time step. Also for this simple problem using Scotch would result in just $(68/0.88) * 0.06 = 4.63\%$ of a time step spent in synchronization and $(16/7680) * 4883 = 10.17\%$ of a time step spent in communication. In all, at least 70% reduction in overall time is expected. Thus, the use of graph partitioners for the SUGAR code might seem promising. In fact, the user has access to any of the aforementioned and some more partitioning methods accessible through the Zoltan framework.

Table 2.4: Results of the 16x16x16 graph problem.

Method	**Worst load imbalance index**	**Cut edges**
Parmetis (PartKWay)	1.04	5782
Parmetis (PartGeomKWay)	1.03	5890
Scotch	1.05	4883

An important observation made during these tests is that if the weight is assigned such that,

$$weight = int(Y_i) \tag{2.3}$$

where Y_i is the Y-coordinate index of the i^{th} node, the quality of the load balance degrades. Note that

this new formula does not change the relative weights of different nodes but it does assign zero weight to some of the nodes. It was found that these partitioners do not handle such cases well. The quality of the weight distribution degrades significantly in the case of Scotch and the number of cut edges increases using Parmetis. The minimum weight should therefore be kept as 1.0. Table 2.5 compares the quality of imbalance for the five nodal cube problem for which the total number of nodes are 125, total weight count is 250 and before partitioning the number of cut edges are 300. Also, note that if no weights are assigned to any of the Octree cells, the number of cut edges is very close for Scotch and Parmetis, which suggests that for some problems, the PartKWay method in Parmetis does give as good results as Recursive Bisection method in Scotch. However, this does not cause a problem if no weight is assigned to any of the nodes and the parameter OBJ_WEIGHT_DIM is kept at zero, in which case the partitioner solves a problem to satisfy only the communication constraint.

Table 2.5: Comparison study using weight formula A. $weight = int(6 - Y_i)$ and B. $weight = int(5 - Y_i)$

Method	**WLII* (A)**	**WLII* (B)**	**Cut edges (A)**	**Cut edges (B)**
Parmetis (PartKWay)	0.98	0.97	59	75
Scotch	0.96	0.12	58	62

* WLII refers to Worst load imbalance index.

2.4 Scalability Study Using Zoltan

2.4.1 Flow Over a Double Wedge Using Scotch

The SUGAR code was coupled with the Zoltan framework to use graph partitioners and a speed-up study was repeated for the hypersonic flow of nitrogen over a double-wedge as described in Sec. 2.2.1. Since Scotch showed better performance than Parmetis, Scotch was used for domain decomposition. The domain size was 0.08x0.12x0.08 m^3 which was occupied by 32, 48, and 32 Octree cells in the X, Y, and Z directions, respectively. The overall number of particles in the domain was approximately 25.6 million. For this study the refinement criterion for the C-Mesh was the same as the old study where it was primarily guided by the number of particles per cell. Technically, the criterion for refinement of the C-mesh should be based on the local mean-free-path, however, for the current study it was based on the number of particles so as to compare the effect of using Scotch with the older results described in Sec. 2.2.1. The code starts with domain decomposition using a 2-D blocking algorithm, and before the start of the sampling step and after

the final adaptive mesh refinement, the topology of the processors is given to Scotch through the Zoltan framework which then outputs a new topology which is used to partition the domain. Topology refers to the information of Octree cell IDs occupied by each processor. In DSMC, the amount of computation in Octree cells is a function of the number of particles in each Octree cell. [23] For specifying the weights on each Octree cell, a criterion based on the number of particles is used as shown in the following listing. 2.1 Table 2.6 shows the comparison between the two algorithms. The Sampling time is shown in minutes for 12,000 samples. Figure 2.8 shows the weighting factor on each Octree cell in the XZ plane passing through the center of the double wedge.

Listing 2.1: Weighting Factor Based on the Number of Computational Particles

```
/* NPar = Number of particles in an Octree */
if (Npar > 1200) { Weight = 5 }
else if ( Npar > 1000) {Weight = 4}
else if ( Npar > 500) {Weight = 3}
else if ( Npar > 100) {Weight = 2}
else {Weight = 1}
```

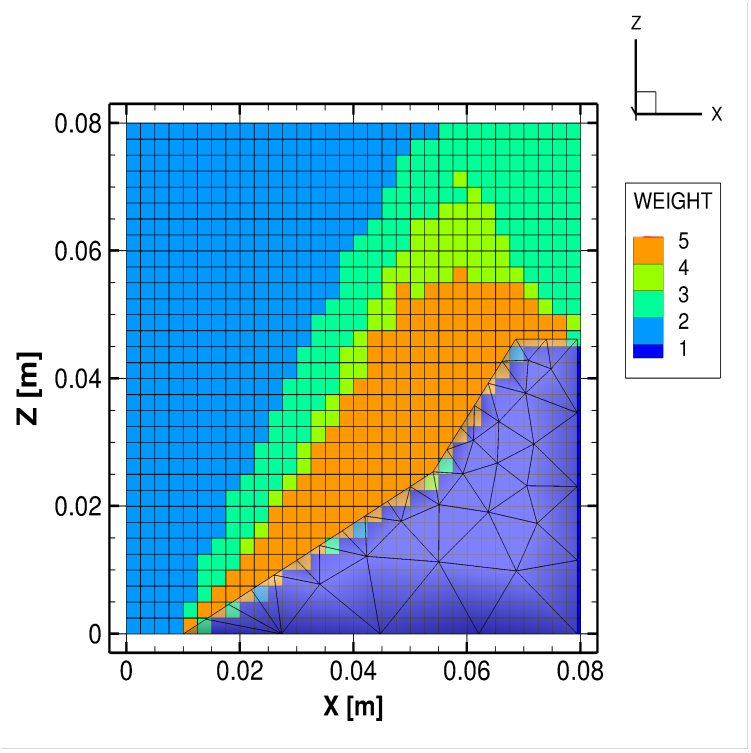


Figure 2.8: Weighting factor distribution based on the number of computational particles.

Table 2.6: Scalability comparison using 2D blocking and Scotch for a flow over a double wedge

Processors	**Cut edges (2-D Blocking)**	**Cut edges (Scotch)**	**WLII* (2-D Blocking)**	**WLII* (Scotch)**	**Time in min (2-D Blocking)**	**Time in min (Scotch)**
64	50124	79996	1.24	1.07	481	843
128	52355	85201	1.28	1.05	275	627
256	56758	91246	1.29	1.06	160	375
512	65601	97135	1.28	1.07	93	252
1024	83250	104628	1.42	1.10	78	162

* WLII refers to Worst load imbalance index.

Table 2.6 shows the effect of implementing Scotch. A uniform weight of one was provided to each edge in the domain. Comparing the number of cut edges, it is observed that their number increases with the use of Scotch. It means that the amount of communication has increased with the use of new topology. The Zoltan parameter IMBALANCE_TOL was set to 1.1, i.e., the new topology was such that the imbalance should not exceed this value. Comparing the load imbalance index, it is seen that the index was not far from one for this problem, therefore, the computational time is dominated by the number of communication links. Nevertheless, Scotch does try to bring the imbalance index very close to one. Relaxing the imbalance tolerance to 1.2 did not improve the results. Scotch always try to keep the imbalance index as close to one as possible.

The time taken for sampling for the two algorithms was plotted and overlaid on the results obtained in the earlier parallelization study as shown in Fig. 2.9. The time for the 2-D blocking algorithm has increased from the earlier results since the new code also computes the surface flux properties and writes a restart file, periodically. However, the increase in time with the use of Scotch is unacceptable.

In order to understand why it gives such unfavourable results for this particular case, it was suspected that the criterion by which the weight on Octree cells is determined could be inaccurate and different criteria were implemented. At first, the leaf cell distribution in each Octree cell was studied. Figure 2.10 shows the leaf cell distribution in each Octree cell in two orthogonal planes (a YZ plane and the centered XZ plane) for better understanding. The number of leaf cells increase after the shock. To understand the reason, one can see Fig. 2.11 which shows the velocity contour and the basic shock structure. The velocity decreases and the number density increases after the shock, which causes a reduction in mean free path and consequently

high refinement of the Octree cells. Therefore, a weighting criterion was selected to imitate such a weight distribution which was a function of number of leaf cells. Listing 2.2 shows the C++ code fragment for the new weighting criterion. Fig. 2.12 shows the weighting factor distribution in the centered XZ plane which closely imitates the leaf cell distribution. However, implementation of this new weighting factor did not make any significant reduction in the available time as shown in Table. 2.7. The previous refinement criterion based on the number of particles appears to have been equally suitable for this study.

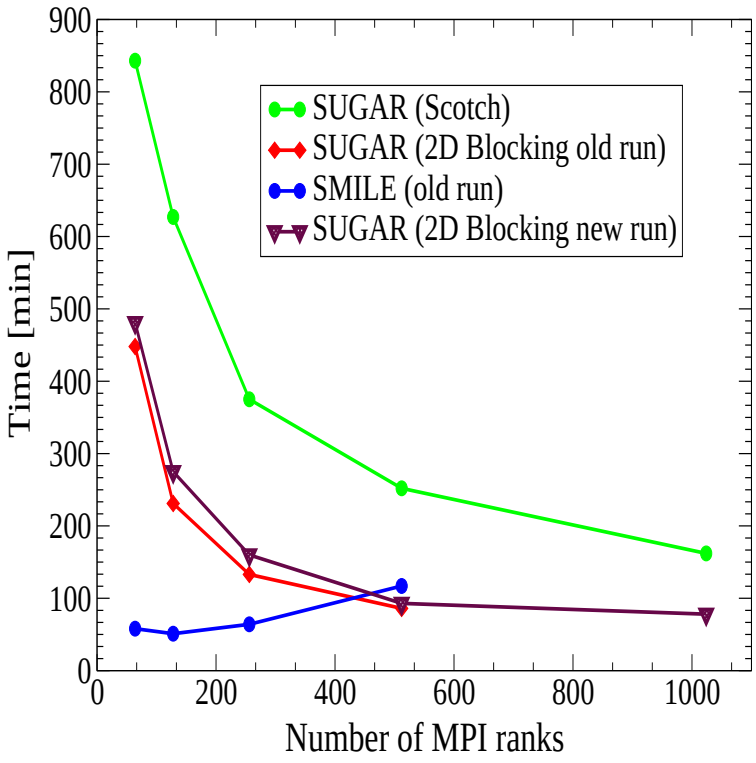
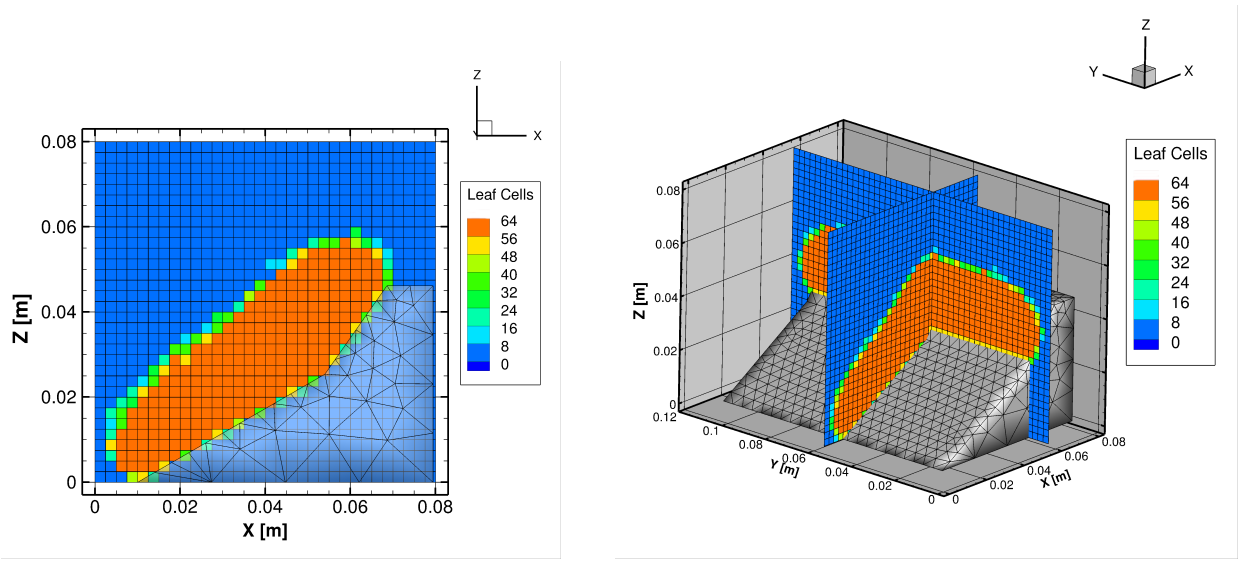


Figure 2.9: Speed-up comparison using Scotch for the double wedge.

Listing 2.2: Weighting Factor Based on Number of leaves

```
if (Nleaves > 4) {
   Weight = (int) log2(Nleaves/2) }
else { Weight = 1 }
```



(a) Leaf Cell distribution in the centered XZ plane.

(b) 3-D representation of the leaf cell distribution.

Figure 2.10: Leaf cell distribution on Octree cells.

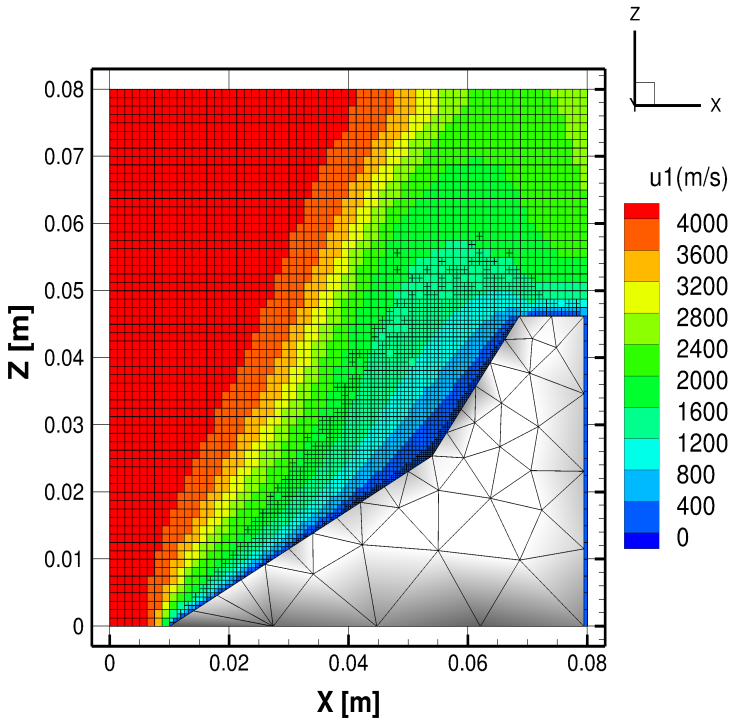


Figure 2.11: Velocity contour.

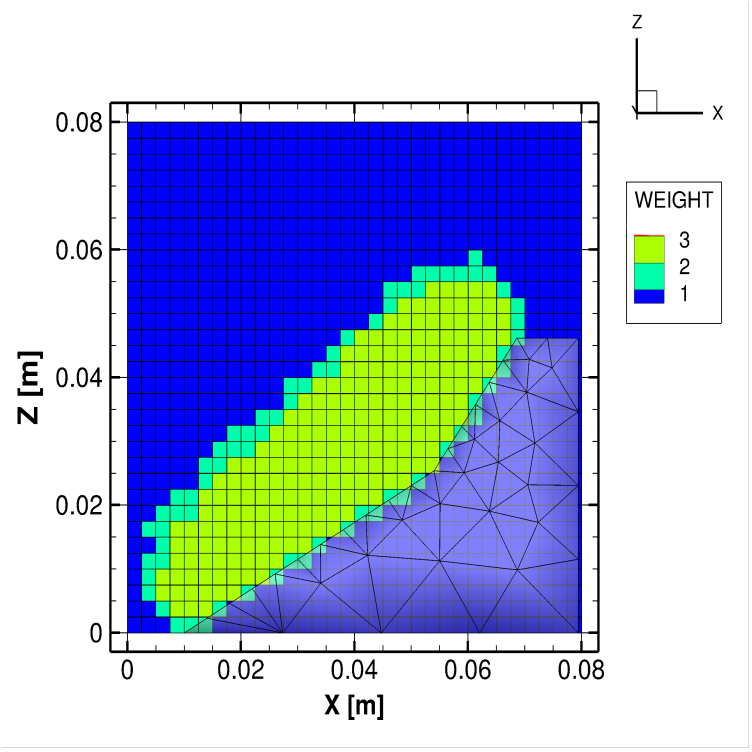


Figure 2.12: Weighting factor distribution on Octree cells based on the number of leaves.

Another weighting factor that was implemented is shown in the listing 2.3 which is also a function of the local number density. However, this factor increased the amount of time even more. Figure 2.13 shows the weighting factor distribution in the centered XZ plane. Table 2.7 lists the timing taken by the use of all three formulas for weighting factor for the double wedge case.

Listing 2.3: Weighting Factor Based on Number Density

```
if (Npar > 1 && Nleaves > 4) {
   Weight = (int) log10(Nden * log2(Nleaves/2)) }
          /* NLeaves = Number of leaves in an Octree
             Nden = Local number density of an Octree */
else { Weight = 1 }
```

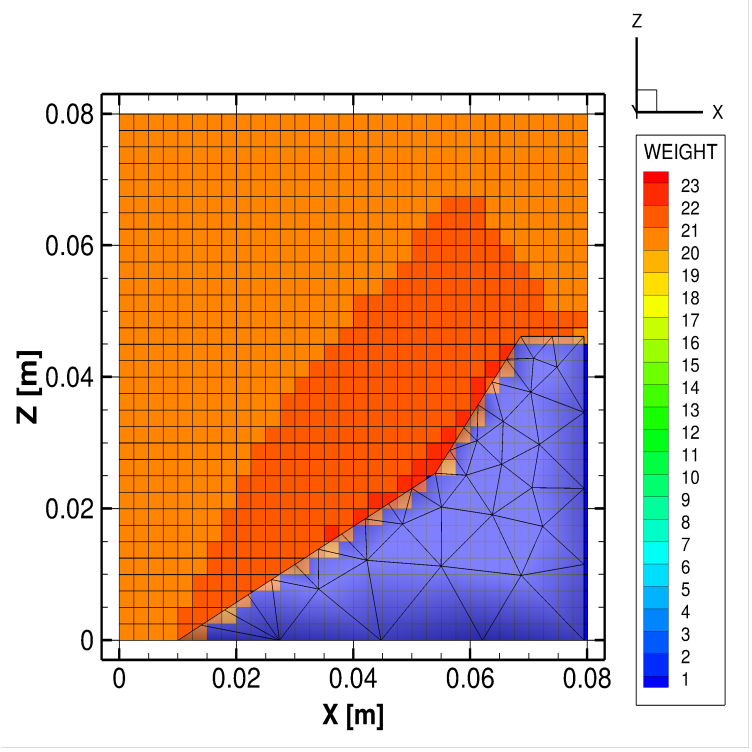


Figure 2.13: Weighting factor on Octree cells based on the number density.

Table 2.7: Effect of different weighting factors on the sampling time.

Weighting Factor Formula	**Percentage* Increase in Sampling Time (Using 512 Processors)**
Based on Number of Particles	171%
Based on Number of Leaves	170%
Based on Number Density and Number of Leaves	415%

* The percentage is calculated with respect to the time taken by a 2-D blocking algorithm for a case with 512 processors as shown in Table 2.6.

The reason for the increase in time could be explained by the work of Abou-Rjeili et al. [24], which shows that having high sum of weights on the two vertices of the edge decreases the effectiveness of the

bisection algorithms since it prevents them from cutting across such edges because such a move leads to a highly unbalanced bisection. Therefore, the domain is partitioned by cutting those edges which have smaller sum of vertex weights. The use of number density criterion increases the sum of vertex weights in stream wise direction and the range of weights varies from one to 23. By many trials, it was observed that the graph partitioners work well if the weight index variation across the domain is less than 10 and is gradual. In fact, the reason for dividing the number of leaves by a factor of two in the formula shown in the listing 2.3 is that it reduces the range of weighting factor in the domain and causes a reduction in time. If instead of division, it is multiplied by two, then the time of the simulation increases. The conclusion of all these tests related to changing the weighting criteria was that the criterion of number of particles and the number of leaves both gave equal results, however, the determination of the optimum weighting criterion is very difficult and it may vary from case to case.

However, the aforementioned results did not explain the reason for the bad performance of graph partitioners for the case of the double wedge. It was suspected that it could be due to the fraction of the domain that has leaf nodes higher than eight which is the highly refined region after the shock. It was computed for the case having the entire double wedge and a case using a symmetry plane. In both these cases, only 24% of the volume is occupied by the Octree cells where the refinement is more than one which shows that the load balancing problem is highly non-uniform. Therefore, it was expected that the increase in the number of processors would make the task of graph partitioning more difficult. As the number of processors increase, there would be less flexibility to partition the 24% volume of the domain having Octree cells with higher weights. Consequently, as the number of processors increase, one would see an increase in the percentage increase in time taken for the sampling stage by the use of Scotch. Table 2.8 shows the percentage increase in time by the use of Scotch with the increase in the number of processors. However, this number reduces for 1024 processors because even a 2-D blocking algorithm is very inefficient at those many number of processors.

Table 2.8: Percentage increase in time with the use of Scotch over a 2-D blocking algorithm.

Processors	**Percentage Increase in Time**
64	75
128	128
256	134
512	171
1024	106

2.4.2 Flow Over a Hemisphere Using Scotch

A similar study was repeated for the hypersonic flow of nitrogen over a hemisphere at a Knudsen number 0.20 as described in Sec. 2.2.2. For the former study the adaptive mesh refinement criterion for the collision mesh was primarily guided by the number of particles and if that criterion was satisfied then the mesh was further refined based on the local Knudsen number. However, for the current study the criterion was changed to strictly depend on the local Knudsen number in such a way that the cells keep refining unless this number is greater than or equal to one. Therefore, the level of refinement in the current run is very high. The domain size is 0.09x0.09x0.09 m^3 which is occupied by 32 Octree cells in each direction. The overall number of particles in the domain is approximately 22.3 million. Using the results of the previous section, for this case a weighting factor based on the number of particles was chosen for each Octree cell as shown in the following listing 2.4. The criterion based on the number of leaves was also tried, however, the following formula based on the number of particles gave the best possible results and hence, is documented here. Figure 2.14 shows the weights assigned to each Octree cell and Fig. 2.15 presents the leaf cells in each Octree cell. Since, the maximum level of refinement in the domain is five, the number of leaves is very high.

Listing 2.4: Weighting Factor Based on the Number of Computational Particles

```
/* NPar = Number of particles in an Octree */
else if ( Npar > 1500) {Weight = 9}
else if ( Npar > 1000) {Weight = 6}
else if ( Npar > 100) {Weight = 3}
else {Weight = 1}
```

The study was done for 128, 256, 512, 1024, and 2048 processors however, for 64 processors, the memory available per processor on Bluewaters node was exceeded and the job could not be run. Table 2.9 shows the

comparison using both the algorithms. Sampling time was noted for 4000 samples and shown in minutes.

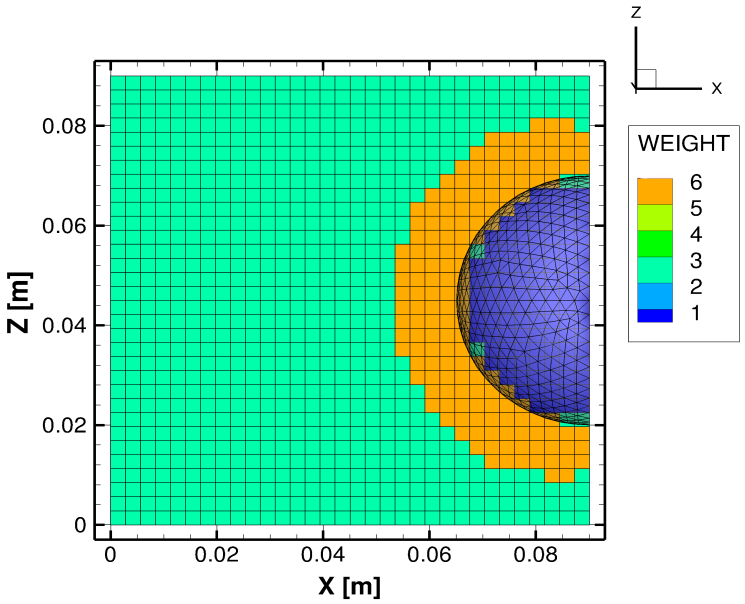


Figure 2.14: Weighting scheme for a flow over a hemisphere.

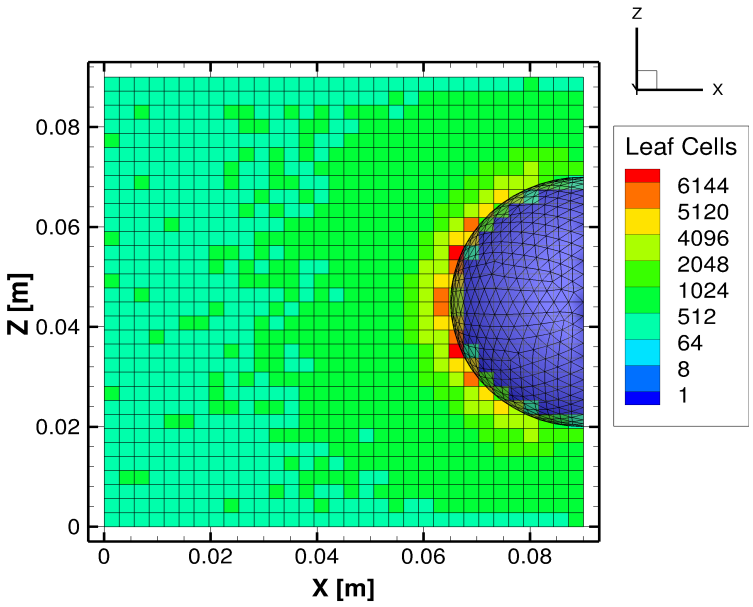


Figure 2.15: Leaf cell distribution for a flow over a hemisphere.

Table 2.9: Scalability comparison using a 2D blocking algorithm and Scotch for a flow over a hemisphere

Processors	**Cut edges (2-D Blocking)**	**Cut edges (Scotch)**	**WLII* (2-D Blocking)**	**WLII* (Scotch)**	**Time in min (2-D Blocking)**	**Time in min (Scotch)**
128	34816	20835	1.44	1.01	200	245
256	38912	25125	1.49	1.03	145	155
512	47104	30051	1.49	1.05	85	104
1024	63488	36002	1.52	1.06	58	71
2048	64512	43656	1.55	1.09	48	71

* WLII refers to Worst load imbalance index.

Table 2.9 shows a reduction in the number of cut edges which should translate to reduction in the communication links. Only 0.4% of the domain volume lies inside the geometry. From Fig. 2.16, which shows the size of cells in the C-Mesh in X-direction, it is clear that even though the mesh in the vicinity of the geometry has five to six levels of refinement, most of the free stream region in the domain has four levels of refinement. Therefore, the change in the weighting factor of Octree cells is gradual and the graph partitioner has more flexibility than the double wedge case in partitioning the domain which results in a reduced number of edges. The load imbalance index based on the given weighing scheme is reduced and brought very close to one as seen in column five. Based on these facts, it is expected that the overall sampling time should be reduced, however, contrary to these expectations, the performance using Scotch remained nearly the same for all number of processors. The speed-up plot comparison shown in the Fig. 2.17 does not seem to improve as compared to the 2D-blocking algorithm for more than 512 processors.

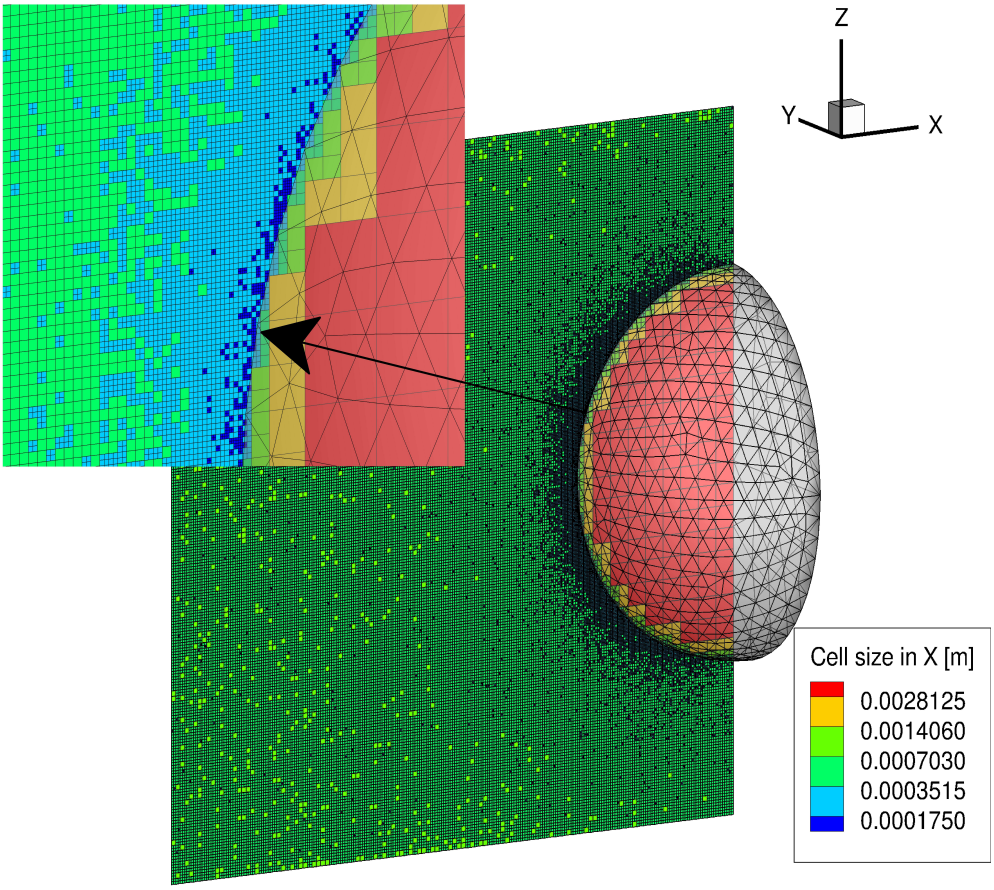


Figure 2.16: C-Mesh for a hypersonic flow over a hemisphere.

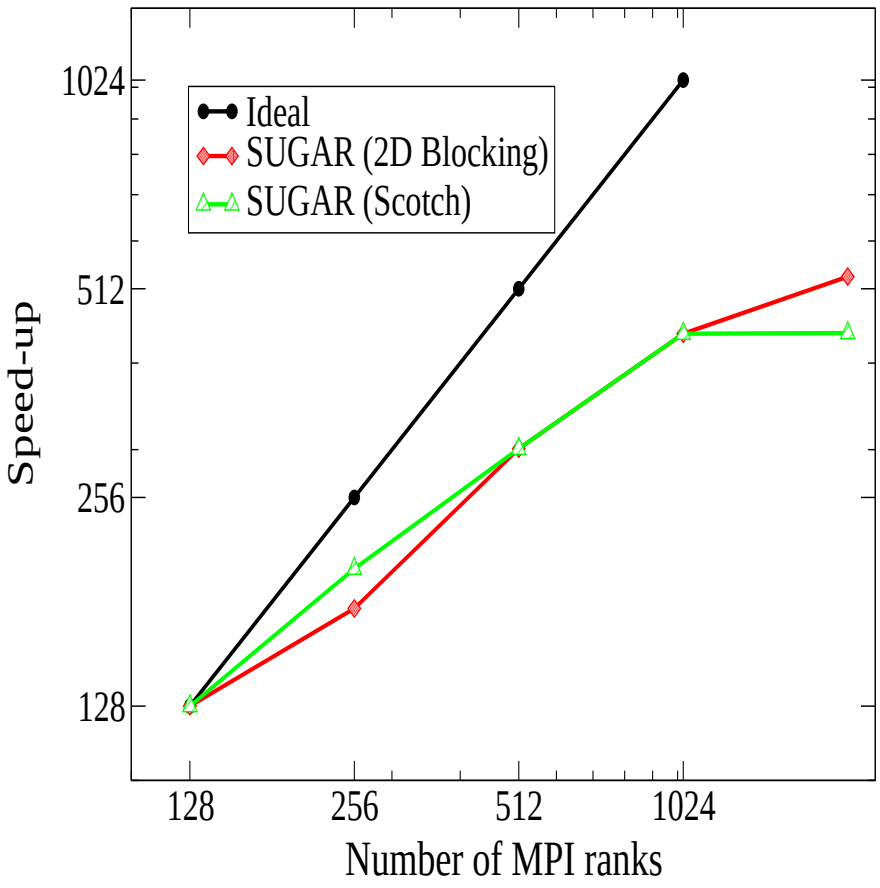


Figure 2.17: Speed-up comparison using Scotch for hemisphere.

2.5 Algorithmic Assessment of SUGAR Code

As explained in the section 2.1 the SUGAR code employs an Octree grid on an underlying coarser Cartesian grid. Each Octree cell is refined based on a refinement criterion and hence any two Octree cells could have different levels of refinement. On this unstructured mesh used to obtain the smallest leaf cells for performing DSMC procedure, the SUGAR code employs a pointer based technique which requires traversal of the entire Octree from the root to its leaf cells. In each time step such traversal has to be performed six times-once to perform collisions, once to collect macroparameter values, and four times during mapping. Four times traversal is required in every step of mapping for deallocating old particle arrays per Octree cell, determination of the new particle array size, then dynamic allocation of new particle arrays and finally, putting particles in their destination cells. The number of traversals also increases when the results are outputted, however, the values are written once in 4,000 time steps therefore this extra traversal time can be neglected. If the mesh size is very fine, however, then this cost could be a bottleneck.

This could be avoided using a sophisticated technique to store the leaf cells in a linear fashion in an array using the Morton-Z encoding method which essentially maps multidimensional data to a single dimension while preserving the locality of leaf cells. This method would avoid the unnecessary traversal of the Octree cells and give significant time improvement. In order to understand whether the performance of the SUGAR code would improve by replacing the present pointer based system with Morton-Z encoding, it was compared with the SMILE code both ran on a single processor and a serial Octree code, CHAOS, written by a colleague, Ms. Revathi Jambhunathan, which incorporates Morton Z-encoding. The SUGAR code was profiled using CPMAT and the other two codes were profiled using the TAU profiler for 2000 samples. CPMAT slows down the code significantly since it uses accurate subroutine tracing method whereas TAU does not cause significant overhead. However, although the absolute times of the code obtained during profiling study could be unreliable for comparison, the main goal was to determine the percentage of time taken by the major DSMC methods such as moving and mapping of particles, sampling, and collisions.

Table 2.10: Comparison of the percentage time taken by major DSMC procedures*.

DSMC Procedure	**SUGAR**	**SMILE**	**CHAOS Serial**
Move and Map	70	54	73
Sampling	25	17	2
Collision	5	25	23

* Absolute time taken by the SUGAR code without profiling is 16000 s, SMILE is 6000 s, and CHAOS is 5076 s for 2000 samples.

Table 2.10 shows the percentage time taken by all three codes. The SUGAR code takes a comparable fraction of the time taken by CHAOS during particle moving and mapping. However, considering the absolute time without profiling the absolute time taken during mapping and moving combined is more than three times higher in SUGAR. In the current implementation of the SUGAR code, sampling does involve traversal of the Octree to access the leaf cells. Approximately 16% of the sampling time is spent in accessing the required leaf cells. The time taken to access collision cells is included in mapping and hence, the collision time shows just the time spent in performing binary inelastic and elastic collisions. Based on these numbers it can be concluded that the main reason for the SUGAR code to be slower is the pointer based tracking of the leaf nodes. In addition, the code could be significantly improved by the implementation of a space

filling curve by saving all the leaf nodes in an array and accessing them by their binary IDs as is done using the Morton-Z curve. [25, 26] The Zoltan framework has an option of geometric partitioning using a Hilbert space filling curve. However, major changes would need to be made in the current data structure of the code to implement such strategies.

2.6 Prediction of the time required to run the experimental case

Based on the above study a simple prediction can be made of the time required to run at the desired continuum-like experimental conditions. Table 2.11 shows the basic input conditions for the main case. For this assessment, FNUM was taken to be $4.0\text{x}10^{13}$, therefore, the number of particles are 1000 times lower than the actual case. The case is physically not correct since the Knudsen number is not satisfied everywhere, however, the main focus is to predict the approximate time taken by the actual case based on the number of particles. The time taken by the SMILE and SUGAR code is shown in the Table 2.12. Based on these numbers the projected time for the actual case with 1000 times smaller FNUM is computed for both the codes by multiplying the sampling time by 1000 as shown in Table 2.12.

Table 2.11: Basic experimental input conditions.

Parameters	**SUGAR**
Number Density [#/m^3]	7.9E22
FNUM	4.0E10
Freestream Temperature [K]	200
Freestream Velocity [m/s]	3812
Time Step [s]	1.0E-09
α_t & α_e	1
Surface Temperature [K]	298.5
Elastic Collision Model	VHS
Viscosity Index	0.81
Number of Samples	200,000

Table 2.12: Time comparison.

Parameters	**SUGAR**	**SMILE**
Number of Samples	200,000	200,000
Sampling Time [min]	180	20
Projected Sampling Time for the actual Case	5 months	2 weeks
Projected Sampling Time per Time Step for the actual Case [min/time step]	0.9	0.1

As shown in the previous section the SUGAR code is dominated by the recursive pointer based traversal of the Octree cells. The computational time complexity of this algorithm can be assessed in terms of inputs using asymptotic analysis. It is a measure of how slow a program will be if the inputs are increased. For example, the asymptotic behavior of a function $F(n)$ such as $F(n) = c*n^2+d*n$, where c and d are constants, is $O(n^2)$. The slower the asymptotic growth rate, the better the algorithm. The asymptotic complexity of the current DSMC method per time step per Octree cell can be written as $O(N \log_8 M)+O(5 \log_8 M)+O(N)$ where N is the number of particles, and M is the number of leaf cells in that Octree. The first term refers to the cost of the mapping procedure where each particle is put into its destination leaf cell. The second term refers to the complexity involved in traversing the leaves for five times one by one before the collision step, the sampling step, and during some part of the mapping procedure. The last term denotes the complexity during other DSMC subroutines such as moving particles and performing collisions. The first term dominates in the asymptotic limit as the number of particles in a root cell increases. On the other hand the complexity of a lineared Octree using Morton-Z space filling curve would be $O(N)$ because the leaf cells are arranged and accessed in a linear fashion based on their binary IDs and the code time only depends on the number of particles. That means even if the mesh is refined, for a steady flow, the run time would simply be a function of the number of particles.

By ignoring the first two terms in the complexity expression of the SUGAR code and assuming that the time taken by the SUGAR code does not depend on the level of mesh refinement, it could be predicted that for 1000 times more number particles it would take about five months to run this case where as SMILE just takes two weeks. Based on this estimate one can calculate the time taken per time step for the actual case and looking at the numbers this time per time step requires significant reduction.

Referring to Table 2.10 in the previous section it is noted that the time taken during collisions by the SUGAR code is 5% of the overall time. Table 2.12 shows that the projected sampling time per time step for

the main case would be 0.9 min. Assuming the same 5% of the time spent in collisions, the main case would require 0.045 min/time step for collisions. However, this absolute time corresponds to 22% of the overall time for a code with Morton-Z implementation provided the same mesh. Based on this number the overall absolute time taken by the code with the Morton-Z encoding would require approximately 0.19 min/time step. This number is comparable to the one required by the SMILE code as shown in Table 2.10. Therefore, it could be safely assumed that the performance of the SUGAR code would be significantly improved by shifting to linear Octree storing techniques.

2.7 Robust Cutcell Algorithm

The SUGAR code has a provision to simulates complex embedded geometries using robust cut-cell algorithm. The geometry information should be provided in the form of triangular elements with its normal pointed outward using the STL format. This format is chosen since it is available on any CAD software and one can create any type of geometry using triangular elements. At first, the code reads in the geometry in the STL format using STLA_IO library [27] and sends the information of the triangular panels to each processor. Although the domain is divided among the given processors, the geometry is not divided but broadcasted to each processor. This approach avoids the need for re-triangulation of the triangles that might be cut at the boundary of a processor's domain. The robust cut-cell algorithm involves two main functions-1. geometric sorting, and 2. volume calculation of cut-cells.

2.7.1 Geometric Sorting

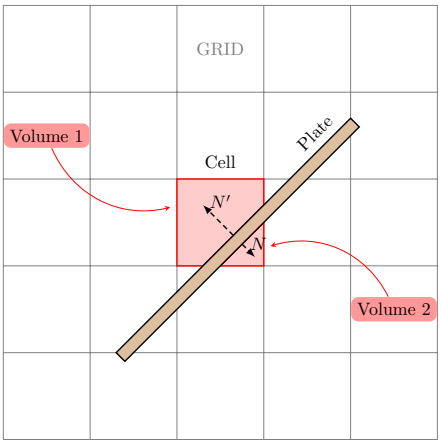
Geometric sorting refers to organizing the list of triangular panels of the surface into the computational cells (leaves of Octree) such that a cell possesses the IDs of the triangles that are intersected by the cell or those that fall inside the cell. Checking each cell with each triangle for possible intersections is computationally very expensive and the cost is O(M*N) where, M is the number of cells and N is the number of triangular panels. [28] However, since we are only interested in the leaves of the Octree, we can take advantage of the inherent recursion involved in accessing these leaves. Initially, the intersection check is performed on all Root-cells, using the signed tetrahedral volume [28] approach, and two linked-lists are formed. One contains the IDs of the triangles intersecting the cell and the other contains the IDs of the triangles lying inside the cell. Then the communication step is performed, so that each processor knows the occupied triangles by each Root-cell. This step is not needed for the volume calculation, but it plays a large role in increasing the efficiency by modifying the reflection subroutine, as explained in next section. Now, in a given Root-cell,

when the code checks for the possible intersection of its children with the panels, it loops over only those panels that belong to its parent, thus reducing the computational cost.

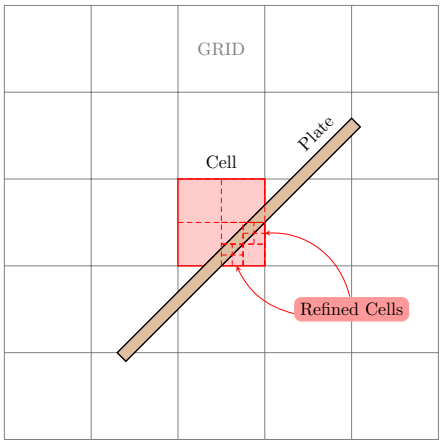
2.7.2 Volume Computation

Once all triangles are sorted into the appropriate leaves of the Octree, the volume of each of these cut-cells is computed using the direct method implemented in the DAC and MGDS codes. [29] Accurate volume computation of the cut-cells is a critical part of AMR/Octree meshing because the calculation of local number density depends on the local cell volume which may affect the local mean free path calculation and thus the refinement criteria for the C-Mesh. Furthermore, the cell volume appears in the calculation of collision frequency which makes it an essential variable that may change the gas-surface interaction. The idea behind this algorithm is to form a polyhedron from the part of the cut-cell that lies outside in the flow domain. This polyhedron has its faces comprised of the face-polygons and the base formed by the triangle-polygons. Face-polygons are the outer lying parts of the six faces of the original Cartesian cut-cell and the triangle-polygons are composed of the triangular panels that are in the two linked-lists of the cell. The intersecting triangles only contribute their part of the triangles that lie in the cell and the remainder is cut off using the Sutherland-Hodgman polygon clipping algorithm [30]. The inner lying part needs to be re-triangulated if it does not form a triangle which is done using the Ear Clipping Method [31]. Once this polyhedron is formed, its accurate volume can be computed. This approach is particularly chosen because of its capability to calculate the volume of the split cell, i.e., the cell where the Cartesian cell is split into many different flow volumes with substantially different properties. 2-dimensional representation of such a case is shown in Fig. 2.18(a). The split cell can have multiple polyhedrons (in the 2-D case, polygons) and thus multiple volumes. Accurate representation of split cells on the V-Mesh is difficult since the data is written as cell-centered data. However, this is possible using the VTK-polygon class [32] in ParaView. Another easier way is just to use one more refinement criterion for the V-Mesh and divide the cell until each cell has only one polyhedron, as shown in Fig. 2.18(b).

Figure 2.19 represents an embedded double-wedge triangulated geometry in an AMR grid with a hypersonic flow in the X direction. The flow is stagnated as it approaches the wedge which results in high number of computational particles near the front surface. As a result, a fourth level of refinement is observed near the surface as shown in the zoomed portion near the hinge. The green cells in the picture represent the cut-cells that are completely divided between two portions, one lying in the flow domain and the other inside the geometry. The red cells are the cut-cells that have one or more of the triangular edges passing through their faces. The blue cells are the cells that are not cut-cells or cells that are co-planer with the



(a) Case describing a split cell.



(b) Refinement in case of a split cell.

Figure 2.18: Split cell and its visualization.

triangular panels without having any of the triangular edge passing through their faces. The volume of the red and the blue cells is the complete cell volume and the volume of the green cells is the volume of the portion of those cells lying outside in the flow domain.

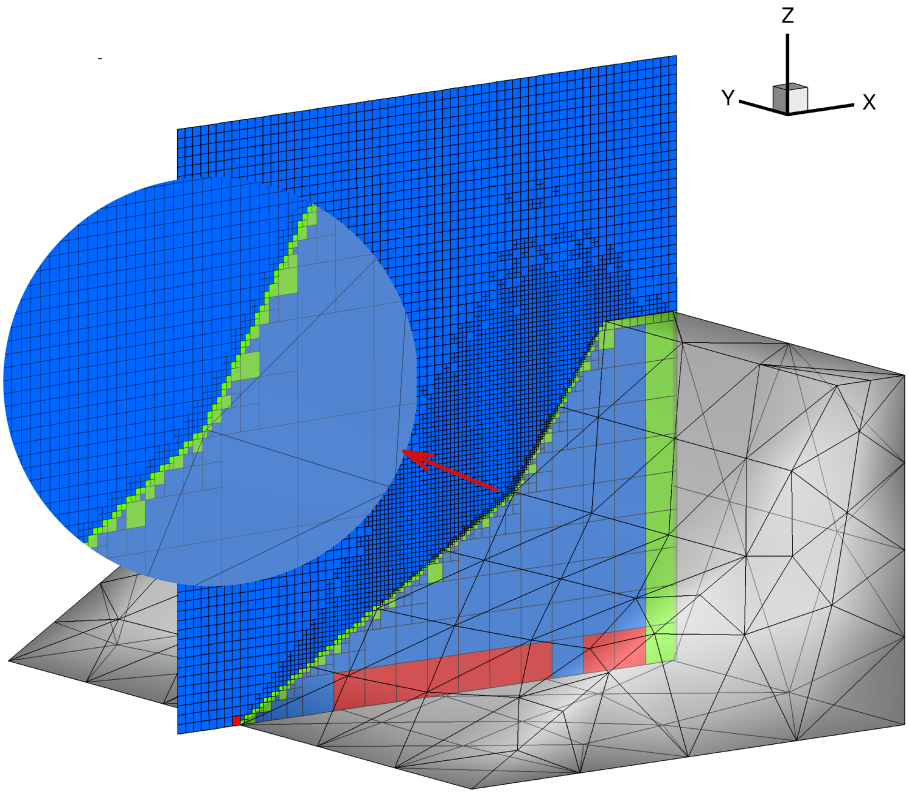


Figure 2.19: Cut-cell and AMR representation for an embedded double-wedge.

2.8 Reflection in MPI Parallelized Domain and Its Optimization

The particles are reflected from the geometry based on the momentum accommodation coefficient. [33] The reflection is specular, if the coefficient is 0, diffuse, if coefficient is one, and partially diffuse if it lies between zero and one. The SUGAR code employs Maxwell's model for diffuse reflection [33].

The particles are moved without keeping track of the cells they cross during the movement. This strategy saves computational effort which would otherwise have to be spent in ray-tracing in which a particle detects which cell it is moving through by checking the intersection of its direction vector with six faces of the Cartesian cells. Note, however, that as opposed to the ray-tracing method, the detection of the geometry has to be checked separately after the movement of the particle. In a very crude algorithm, one would have to check each particle with each triangle of the geometry and the computational cost would be O(N*P) where, N is a number of particles and P is the number of triangular panels. This is extremely inefficient.

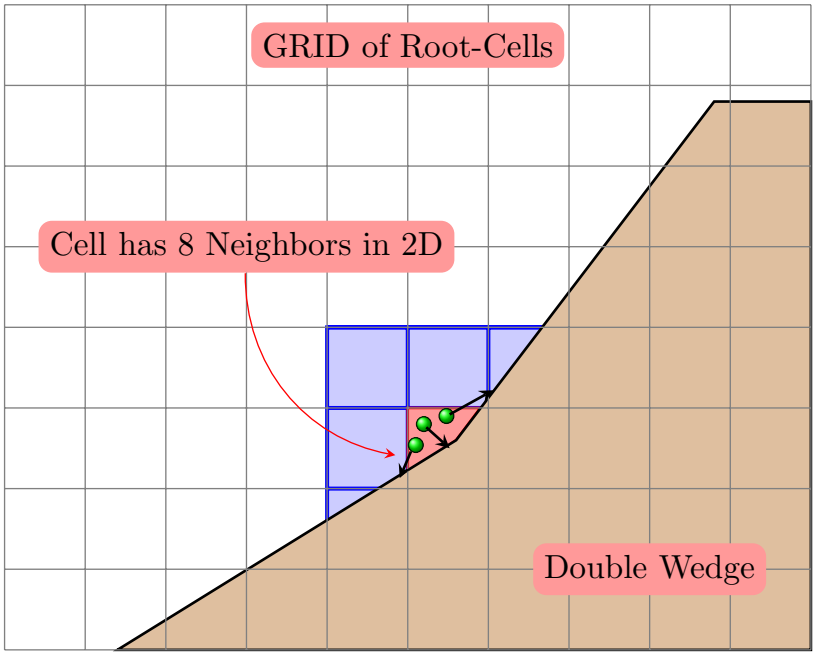


Figure 2.20: Neighbors of a cut Root-cell.

The SUGAR code employs an efficient algorithm to deal with this problem. In a usual DSMC simulation the Root-cells are refined near the geometry due to the decrease in the local mean free path and increase in the number of particles. Hence, the leaves are much smaller than the Root-cells. If the time step is sufficiently small (i.e., the DSMC criteria is accurately followed) then it is impossible for the particle to cross more than two Root-cells in any direction. Hence, it makes sense to check for reflection of those particles that are located in Root-cells that are cut by the geometry and their immediate neighbors. As explained in

the previous section, the Root-cells that are cut by the geometry are already known. Let us create a new flag named NearTheGeometry which is one (true) for these cells. Furthermore, each of their 26 neighbors are flagged as one. The particles have the information of which Root-cell they belong to at a particular time step, since assigned to them when they are sorted into the processors' Root-cells. Thus, each particle can be detected if it is close enough to the geometry by checking if its Root-cell has NearTheGeometry flag equal to one or zero.

The efficiency can be further increased if the particles are checked for intersection with only those panels that are in the linked-lists of the Root-cell and its neighbors. Usually, the number of such triangles is orders of magnitude less than the total number of all the panels of an intricate geometry. One important consideration to be noted is that when the domain is divided into different processors, it may happen that the neighboring Root-cell would belong to another processor. In this case, the information of its corresponding panel intersection list is at the neighboring processor. Hence, it is necessary to communicate the panel intersection lists to each processor using the function MPI_AllGatherv so that each processor gets the information of each Root-cell. This does not create any overhead since it is performed only once in the entire run. The strategy is summarized in Fig. 2.20.

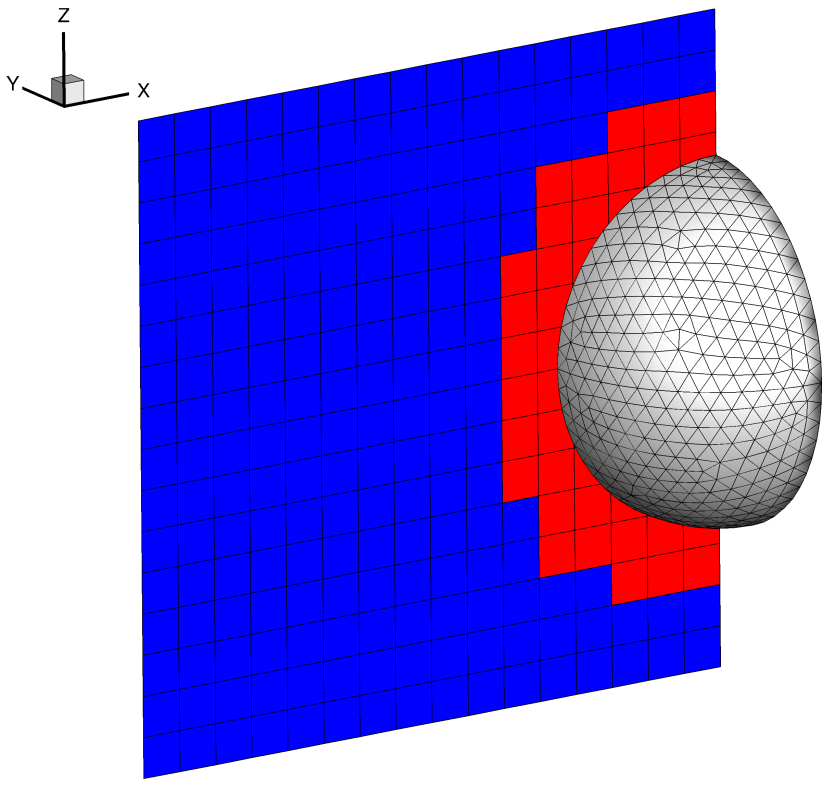


Figure 2.21: Region of cells in which the particle-surface interaction procedure is performed.

Figure 2.21 shows a triangulated hemisphere geometry embedded in an unrefined Octree grid. The flow is in the X direction and the X-Z plane at the centered Y location is shown. The Root-cells marked in red show the region of interest in which if the particle lies, it goes through the gas-surface interaction routine. These cells are composed of cut-cells and their neighboring cells. Moreover, a closer look reveals that each Octree cut-cell occupies not more than ten surface panels. Hence, instead of checking a particle for each of the 1400 surface triangles of the geometry, only less than 50 triangles are checked for a possible intersection. For a case of 1.3 million particles in a domain containing the hemisphere this algorithm gave considerable reduction in overall wall-clock time. Moreover, the SUGAR code took 134 s for 5000 samples as opposed to SMILE which took 256 s. Note that the case is very simple where the particle-surface interactions dominate. However, for high number of particles the trend may not be the same since major bottleneck would be mapping of particles to the highly refined grid.

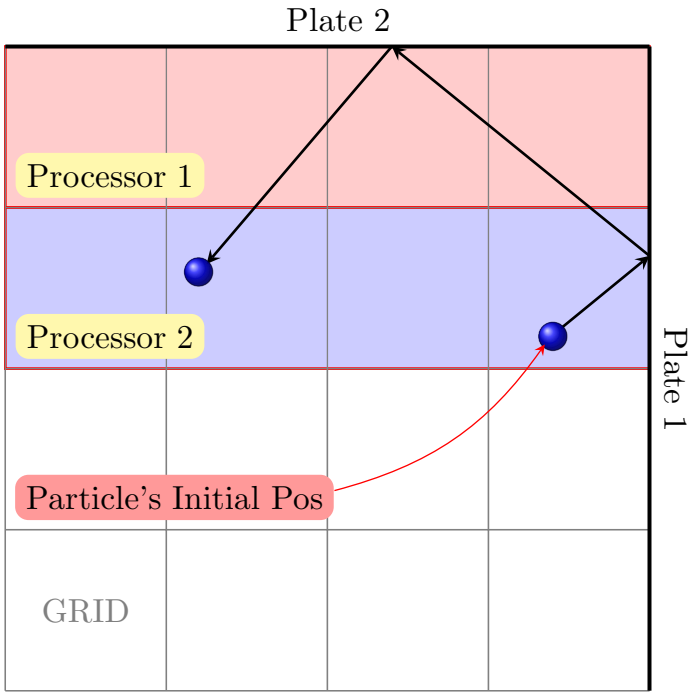


Figure 2.22: Multiple Reflection in a single time step across different processors.

The same idea can be extended to the implementation of a symmetry plane in the domain. Only the Root-cells that are in contact with the symmetry plane and its neighbors can be tagged for performing reflection checks. In this case, a cell may have 15, 11, or 7 neighbors based on whether the cell is in contact with only one domain boundary, two boundaries, or three boundaries (a corner cell), respectively.

In case of intricate geometries, a particle may go through multiple reflections during single time step. It is also possible that after first reflection it might bounce off to a region occupied by another processor. To clarify, Fig. 2.22, shows an exaggerated scenario where the particle starts in processor two, goes to the domain that belongs to processor one, and again reflects back to processor two, all in single time step. If the geometry were also divided among the processors then the particle would need to be communicated two times during the reflection step itself. However, in SUGAR, since each processor has all the geometry information and the particle moves for the full time step regardless of how many cells are crossed, the particle's final position and velocity can be computed for the entire time step and only then is it communicated if its final position is in the region of another processor. In the case shown in Fig. 2.22, no communication is necessary during or after the reflection since the particle lands up on the same processor.

2.9 Computation of Surface Coefficients in MPI Parallelized Domain

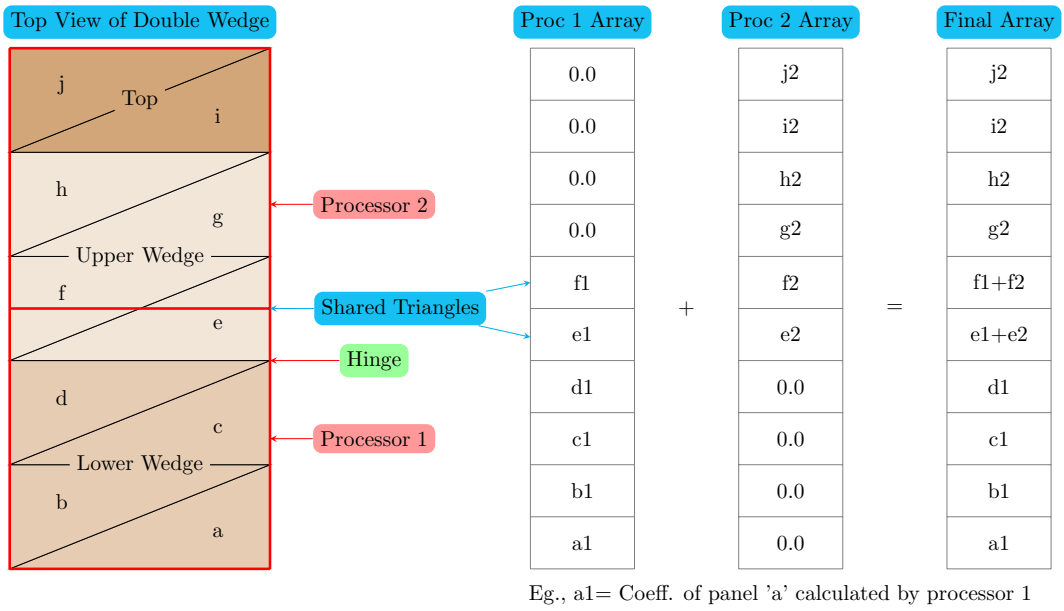


Figure 2.23: Calculation of Surface Coefficients.

The SUGAR code has the facility to calculate surface coefficients such as heat flux, friction, pressure, etc. and fluxes of mass, kinetic energy, and rotational energy over the triangular panels of the geometry. The formula to calculate these in the code is given in the Appendix. Calculation of these parameters in a parallel code is simplified because each processor holds the information of the entire geometry. However, the Cartesian domain is divided among different processors and hence, a processor can only update the portion of the geometry that belongs to it. Since, the edges of the triangular panels of the geometry are not necessarily lying on the processor boundaries, it is possible that a triangular panel is shared by multiple processors and all of those processors would then try to update the heat flux of that panel.

Such a case and its remedy is explained in Fig. 2.23 which shows the top view of the double wedge on the left hand side. Triangles f and e, that are the part of the upper wedge, are shared by two processors. Initially each processor obtains a 1-D array of size equal to the number of surface panels to store the values of different surface coefficients. At the beginning of the simulation its values are initialized to zero. During the calculation of the coefficient each processor loops over all the panels, but only the value of those panels that fall into the boundaries of that particular processor are updated. As shown, both the processors update the value of the f and g triangle in their respective arrays. For the shared triangle, the surface coefficient

should be the sum of the values calculated by each processor. Hence, a final array is created by summing the corresponding indexes of all the arrays. The actual surface panels for a double-wedge configuration were shown in Fig. 2.19. The panels that are intersected with the green cells are used for computing the heat flux data over the surface and compared with the experiments at the locations of the thermocouples.

2.10 Collision Models

Modeling the interactions of molecules is the most important aspect of the DSMC method. The phenomenological collision models are being implemented in the SUGAR code, but efforts are on going to improve and implement new models that are more accurate.

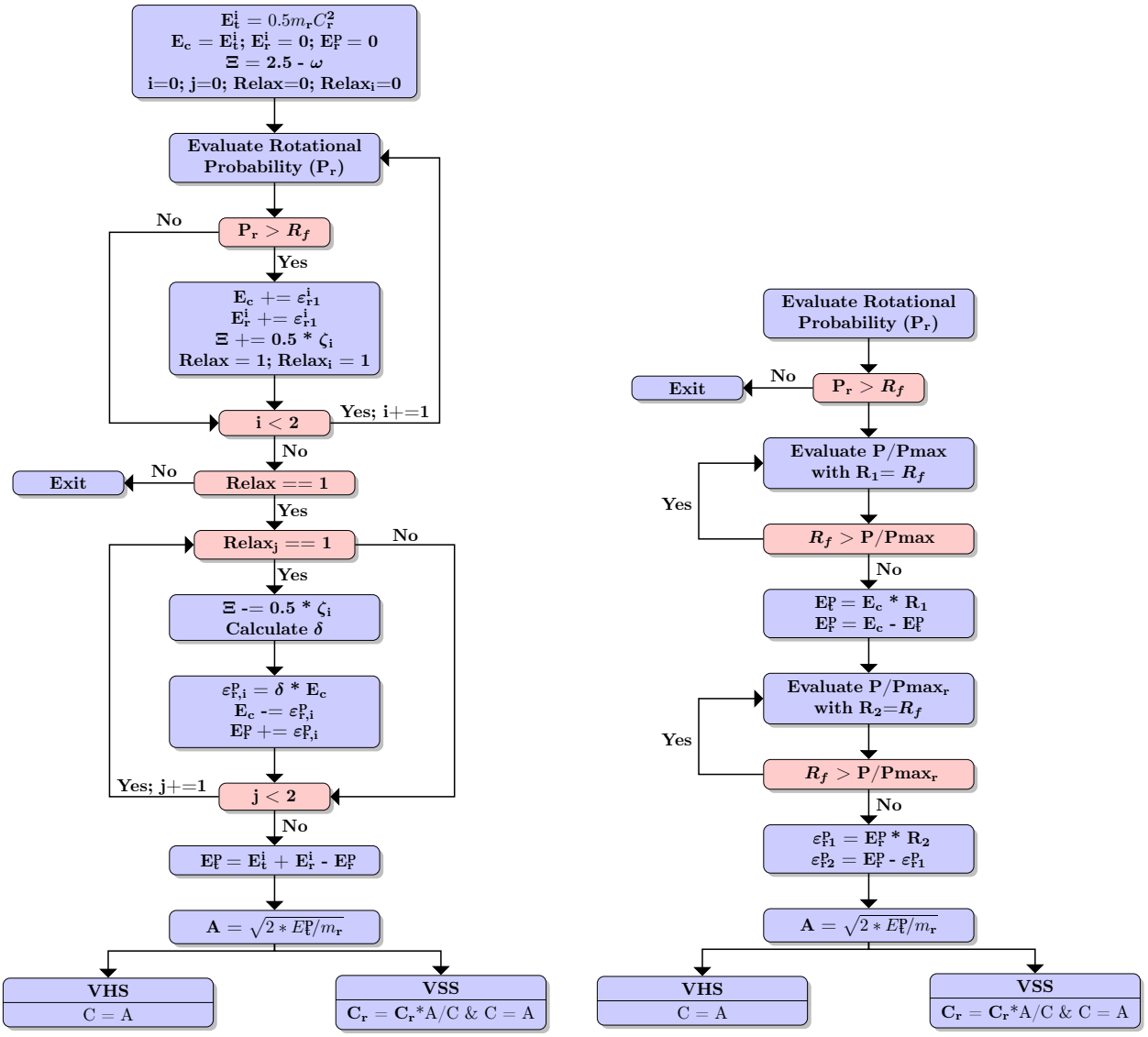
To describe elastic collisions, the Variable Hard Sphere (VHS) [1], and Variable Soft Sphere [34] (VSS) model has been implemented. For rotational relaxation, the Borgnakke-Larsen (BL) model [35] has been implemented with translational-rotational (TR) energy transfer applicable for continuous energy levels. Vibrational model has not been implemented. SUGAR code employs two implementations of the BL models. One algorithm is based on the hierarchical application of the BL model as explained by Bird [1] which has been proved to follow the equipartition theorem and is summarized in Fig. 2.24(a). In a collision step, one step decides whether the rotational energy of either or both of the molecules is to be adjusted. The probability with which the rotation energy of a molecule is to be changed is determined by taking the inverse of the rotational collision number which can be taken as a constant or as temperature dependent. The temperature dependence is based on Parker's model, [36]

$$Z_r^C(T) = \frac{Z_{r,\infty}}{1 + \frac{\pi^{3/2}}{2}\left(\frac{T^*}{T_e}\right)^2 + \left(\frac{\pi^2}{4} + \pi\right)\frac{T^*}{T_e}} \tag{2.4}$$

where T_e is the effective temperature computed from the relative translational energy of the colliding molecules sampled over the collisions in each cell at each time step. The values of constants $Z_{r,\infty}$ and T^* for nitrogen is 18.5 and 91, respectively. As described by Lumpkin et al. [37], Z_r is corrected by the following factor before use in DSMC,

$$Z_r = \frac{\zeta_t}{\zeta_t + \zeta_R} Z_r^C \tag{2.5}$$

where, $\zeta_t = 4 - 2\alpha$, is the number of relative translational degrees of freedom and ζ_R is the number of rotational degrees of freedom of the participating molecules. The energy to be distributed, which is referred to as the available energy E_c, is the summation of the relative translational and rotational energies of the participating molecules. In the Hierarchical model shown in Fig. 2.24(a), the post-collisional rotational



(a) Hierarchical BL Model.

(b) Serial BL Model.

Figure 2.24: Demonstration of continuous rotational relaxation algorithm.

energy of the participating molecule is of the fraction δ of the available energy where δ is the ratio of the new rotational energy to available energy and is a function of the total number of modes (translational and rotational) that are participating in the relaxation. For two internal degrees of freedom it is given as, $\delta = 1 - R_f^{1/\Xi}$. For three internal degrees of freedom one should refer to the book by Bird [1]. The particles are selected one by one for energy redistribution and should both are subject to selection, the available energy for molecule two is reduced by the new rotational energy of molecule one.

According to Bird, the end result is the same as if the total available energy is first distributed between the translational and total internal energy, and the total internal energy is then distributed between the internal energies of two molecules. The latter approach was originally described by Borgnakke and Larsen [35] and has been implemented in SMILE [2]. For verification even the second algorithm is implemented in SUGAR and no difference was observed between the results by two algorithms. This approach is summarized in Fig. 2.24(b). One important point to note is that for the latter algorithm the value of ζ_R in the formula 2.5 for the binary collision of nitrogen would be four whereas for the former it should be taken as two since the probability of inelastic collision for each particle is evaluated one after other. This model assumes local thermodynamic equilibrium and a fraction of E_t/E_c is sampled with the acceptance-rejection method using Eq. 2.6 that calculates the probability of total energy redistribution into translational and internal modes.

$$\frac{P}{Pmax_{tot}} = \left\{\frac{\zeta + 1/2 - \omega}{3/2 - \omega}\left(\frac{E_t^p}{E_c}\right)\right\}^{3/2-\omega} \left\{\frac{\zeta + 1/2 - \omega}{\zeta - 1}\left(1 - \frac{E_t^p}{E_c}\right)\right\}^{\zeta-1} \tag{2.6}$$

Once the post-collision translational energy of the system of colliding molecules is assigned, the remaining energy is the new total rotational energy. This total rotational energy is further distributed between the two particles with the acceptance-rejection method using Eq. 2.7 which calculates the probability of internal energy redistribution normalized with its maximum value.

$$\frac{P}{Pmax_r} = 2^{\zeta-2}\left(\frac{\varepsilon_{r,1}^p}{E_r^p}\right)^{\zeta/2-1}\left(1 - \frac{\varepsilon_{r,1}^p}{E_r^p}\right)^{\zeta/2-1} \tag{2.7}$$

Once, the exchange of energies is completed, the relative speed is calculated using the post-collision relative translation energy as described in Fig. 2.24.

Chapter 3

Verification and Validation

The research problem in hand is to apply the SUGAR code to study shock-shock and shock-boundary interactions from hypersonic flows about a double-wedge configuration. These phenomena have a significant effect on the flow features resulting in a separation zone near the hinge of the wedge and shear layer. In the design system process, the accurate prediction of these effects on the macroparameters is crucial to avoid extreme heating and pressure loads in the vicinity of the interaction points. Because of the complexity of the problem, for the present work we have chosen relatively simpler cases of the flows over a hemisphere and double-wedge configuration at a larger Knudsen number than the experimental conditions for the validation of the code and the implementation of relaxation model. The results are compared with the 3-D version of the SMILE code written in Fortran-77 [2].

3.1 Hypersonic Flow Simulation Using Argon

Initially a nonreactive gas is chosen so that the relaxation model can be switched off. These cases are chosen for the validation of the DSMC procedure, elastic collision models, and the applicability of the cut-cell algorithm to different geometries.

3.2 CASE I: Flow of Argon at Knudsen Number 0.0277 Over a Hemisphere

The hemisphere, as shown in Fig. 2.21, is of radius 0.025 m with its center located at the Cartesian coordinate (0.09,0.045,0.045) m in the domain of dimension 0.09x0.09x0.09 m^3. For line plots, values of the macroparameters along the stagnation line (0.0-0.065, 0.045, 0.045) m have been extracted from the V-Mesh. The simulation conditions are shown in Table 3.1.

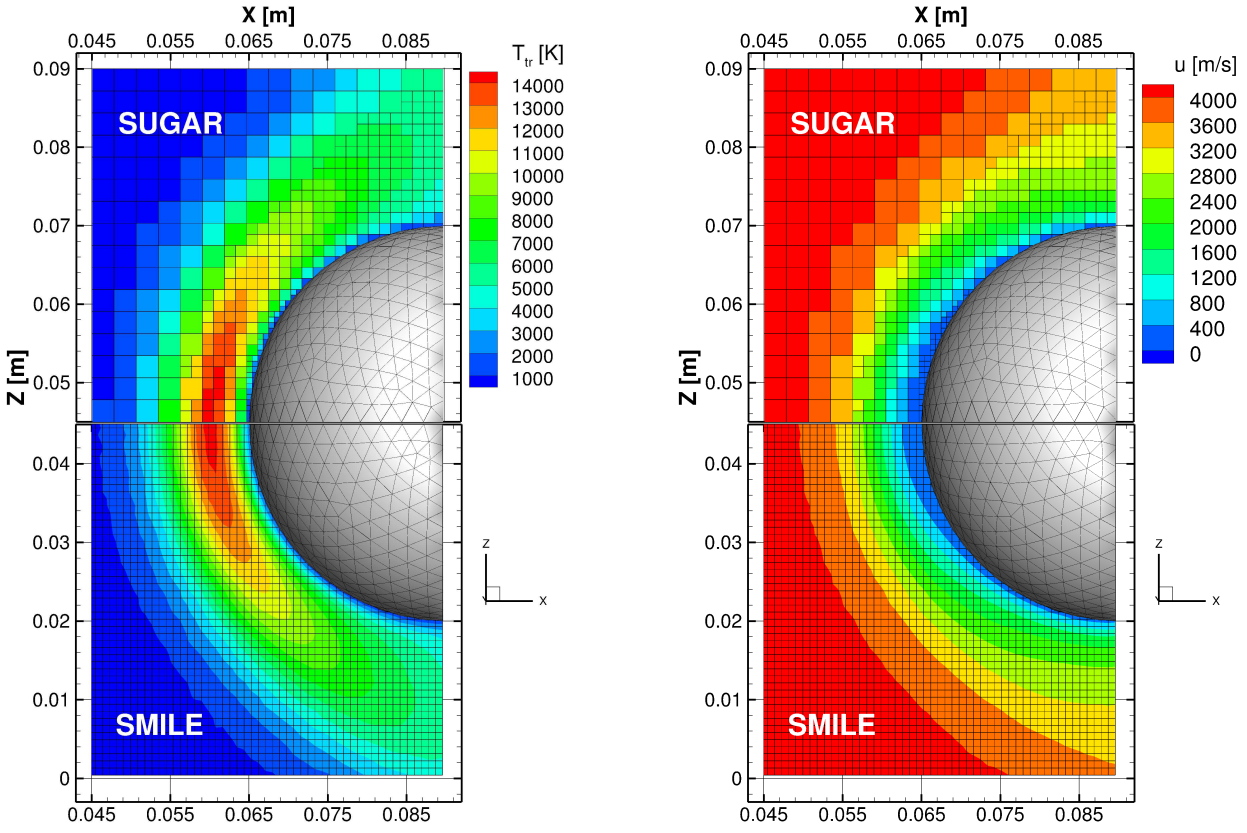
Table 3.1: Numerical parameters for the flow of argon over a hemisphere.

Parameters	**SUGAR**
Number Density [#/m^3]	9.33E20
FNUM	0.5E11
Freestream Temperature [K]	200
Freestream Velocity [m/s]	4200
Time Step [s]	5.0E-08
α_t & α_e	1
Surface Temperature [K]	200
Elastic Collision Model	VHS
Viscosity Index	0.81
Number of Samples	12000
Smallest Cell Size on the V-Mesh	7.0132E-04
Number of Particles	12,709,000
Number of Processors Used	256
Time Required for sampling [min]	262

For the SMILE run the number of particles are 17,608,704, the cell size on the V-Mesh is 9.0E-04, and the time required for sampling is 66 min.

The simulation conditions are kept the same in both codes except for the time step at which the simulation is started. In the SUGAR code, the particles fill up the domain by entering through the inlet boundary. After a sufficient amount of time, the adaptation and collision routines start. This time is generally taken to be three times the time required by the particles to traverse the domain in the stream-wise direction. Adaptation takes place at a predefined interval until the steady state is reached since we want to obtain a mesh that is well adapted to the steady flow conditions. After this step the sampling stage begins. The SMILE code initializes the particles in the domain at the start of the simulation, thus allowing the collisions to start relatively early. Because of this difference, the SUGAR code takes more time to reach to steady state and starts sampling at a later time step. The manner of initialization is not so important for the current work as the majority of computational effort is taken during the sampling stage.

The Mach 14 flow encounters a bow shock ahead of the hemisphere. At such a high Mach number there is a sharp increase in the number density and temperature. Since argon is a monoatomic gas which has only three translational degrees of freedom, the rise in translational temperature is very high. Figure 3.1 shows the comparison of the temperature and velocity contours over the V-Mesh using the SUGAR and SMILE code. Since the flow is symmetric, the upper half of the plot shows contours obtained using SUGAR and lower half by SMILE. The contour plane is passing through the peak of the hemisphere. As observed from Fig. 3.1, the shock stand-off distance is 0.015 m. In front of the shock, sudden increase in the number density results in more computational particles thus refining the V-Mesh. A third level of refinement is enough to resolve a sudden change in macroproperties in front of the shock. Very close to the geometry even a fourth level of refinement is observed where the velocity in x-direction comes to rest because of the geometry and the flow deflects away.



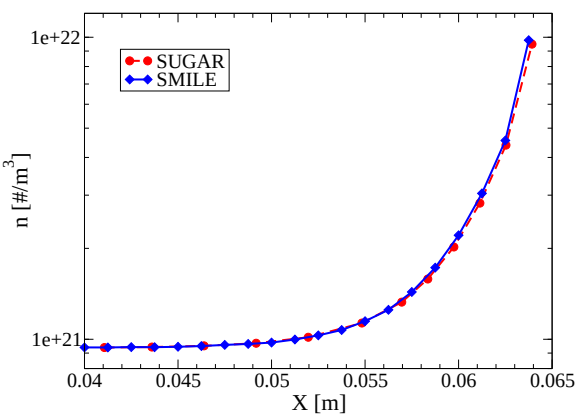
(a) Translational temperature.

(b) Velocity in X-direction.

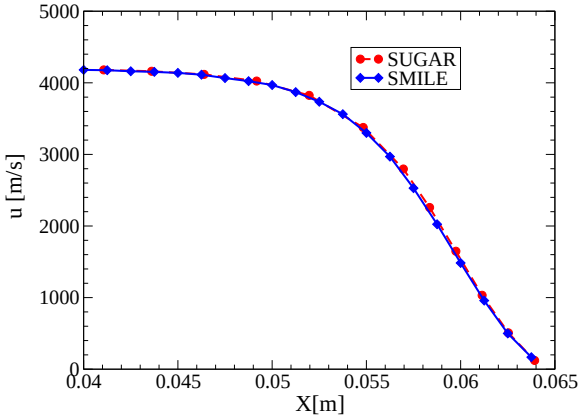
Figure 3.1: Comparison of contours for the flow of argon over a hemisphere.

Note that the SMILE contours look smooth because they are interpolated over the uniform grid in the visualization software. AMR grid can also be interpolated during the post-processing phase, however,

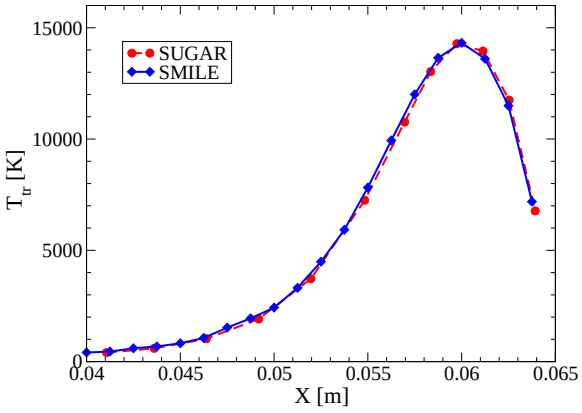
it is of no importance as long as the contours correctly represent the cell-centered data outputted by the DSMC method. For the contour plots in the next sections, the interpolation for the SMILE results have been switched off. It can be observed that the cell centered data points before the shock are fewer in number for SUGAR which is a major advantage over a uniform grid implemented in SMILE since it reduces the number of cells over which macroparameters are calculated. Better refinement can be obtained for SUGAR by reducing the threshold of number of particles exceeding which cells in V-Mesh are refined. Since the geometry is simple, this case does not heighten the advantages of AMR, however, in case of complex geometries the adapted grid would save significant computational efforts.



(a) Number density.



(b) Velocity in X-direction.



(c) Translational temperature.

Figure 3.2: Comparison of macroparameters for the flow of argon over a hemisphere.

Figure 3.2 shows the comparison of the macroparameters obtained along the stagnation line. The number density increases to $1.0x10^{22}$ m^{-3} after encountered by the bow shock, resulting in slowing down of the velocity in X-direction and rise in translational temperature to 14200 K. The results are in exact agreement with SMILE for velocity, temperature, and number density, thus validating the DSMC procedure

and elastic collision model.

3.3 CASE II: Flow of Argon at Knudsen Number 0.02 Over a Double-Wedge

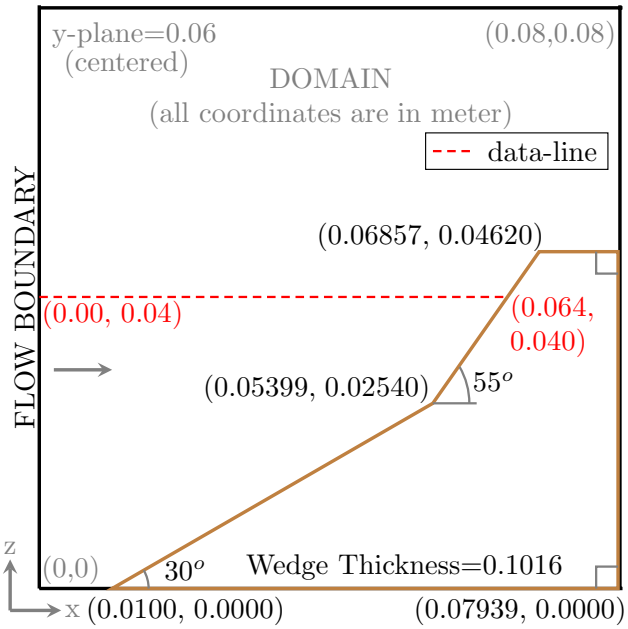


Figure 3.3: Schematic of the double-wedge.

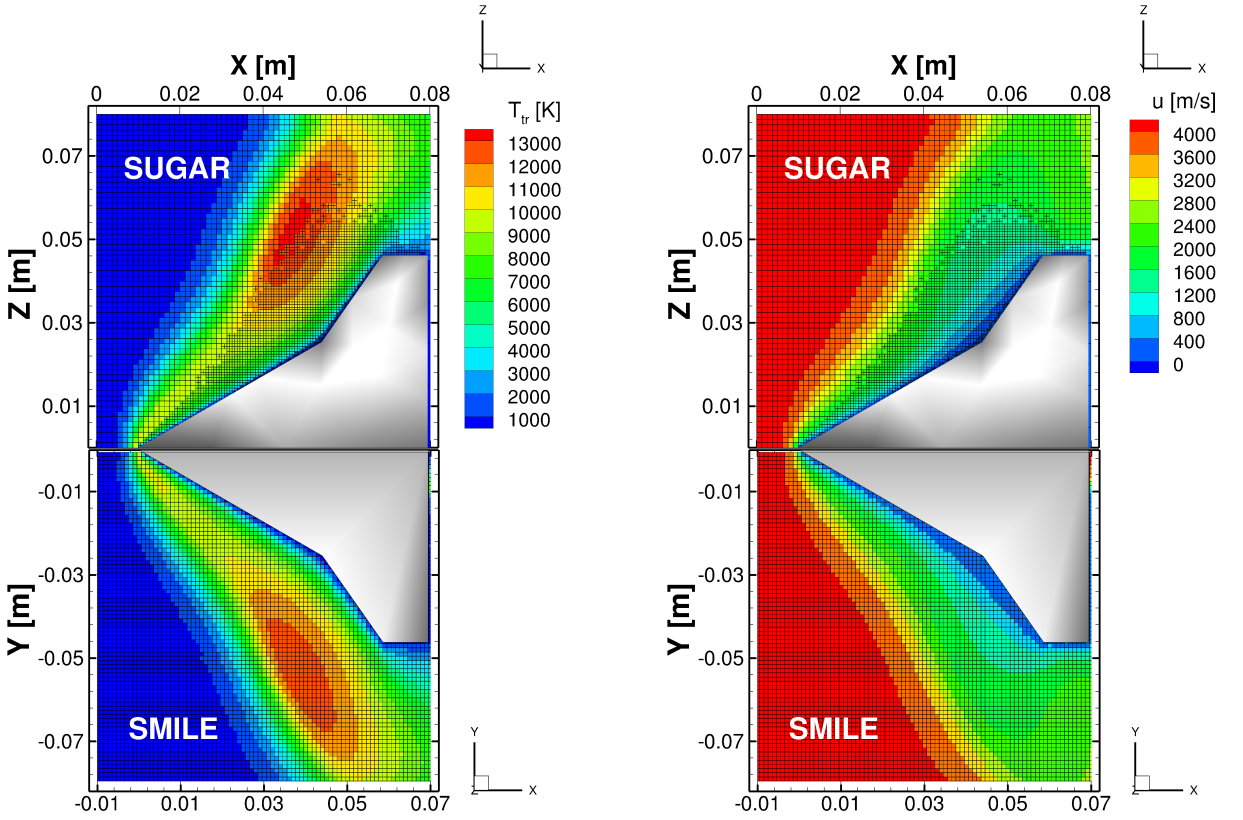
After achieving a good agreement for the hemisphere geometry, the flow over a double-wedge configuration is presented to show the capability of the SUGAR for simulating complex three-dimensional flows. This geometry, shown in Fig. 3.3, has been used in the experiments conducted by Swantek and Austin [7] to analyze the impact of the thermochemical effects on shock wave boundary layer interactions by changing the chemical composition nitrogen to air at different stagnation enthalpies. The ultimate goal is to be able to simulate the complex flow features caused by the hypersonic flow over this configuration. At continuum-like conditions Edney type of shock-shock interactions are observed which include the attached oblique shock formed by the first wedge, detached bow shock caused by the upper wedge, separation and reattachment shocks resulting from their interactions at the triple point, separation of the boundary layer near the hinge, and three-dimensional effects [9]. However, for this preliminary case the chosen number density for the simulation is too low to observe the Edney type of shock interactions. Table 3.2 shows the Numerical parameters used in this study.

Table 3.2: Numerical parameters for the flow of argon over a double-wedge.

Parameters	**SUGAR**
Number Density [#/m^3]	9.33E20
FNUM	0.25E11
Freestream Temperature [K]	200
Freestream Velocity [m/s]	4200
Mach Number	14
Time Step [s]	5.0E-08
α_t & α_e	1
Surface Temperature [K]	200
Elastic Collision Model	VHS
Viscosity Index	0.81
Number of Samples	12000
Smallest Cell Size on the V-Mesh [m]	6.25E-04
Number of Particles	51,990,000
Number of Processors Used	256
Time Required for sampling	330

For the SMILE run the number of particles are 59,850,000, smallest cell size on the V-Mesh is 8.0E-04, and time required for sampling is 93 min.

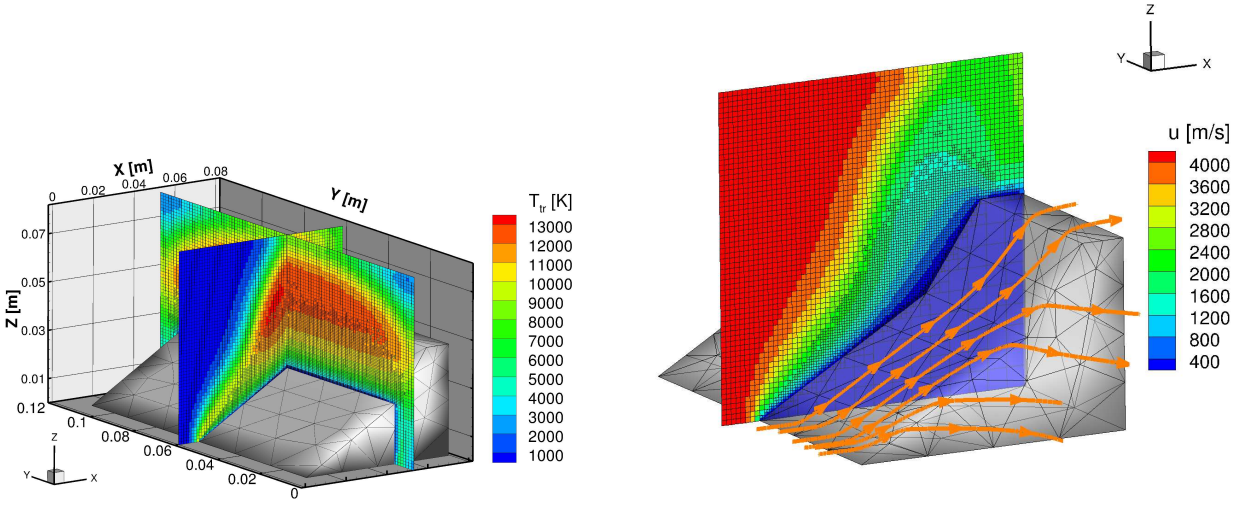
Figure 3.4 shows the spatial distribution of translational temperature and the velocity field in the plane passing through the center of the wedge on the V-Mesh for the SUGAR and SMILE codes. Because of the differences in the orientation of the axes used for both codes, the center plane of the wedge in the SUGAR code is at y=0.060 m as shown and for the SMILE code it is at z=0.0 m. Looking at these contour plots and the streamlines plotted in Fig. 3.5(b), only the primary features of the oblique and bow shocks can be recognized. Highest temperature is obtained after the bow shock. 3-D effects play major role in flow over a double-wedge. Since the streamlines are directed in the span-wise direction, the shock thickness and hence, the highest temperature obtained after the bow shock decreases along the span as seen in Fig. 3.5(a).



(a) Translational temperature.

(b) Velocity in X-direction.

Figure 3.4: Comparison of contours for the flow of argon over a double-wedge.



(a) Temperature reduction in span-wise direction.

(b) Streamlines over the double-wedge.

Figure 3.5: 3-D effects for the flow of Argon over a double-wedge in SUGAR.

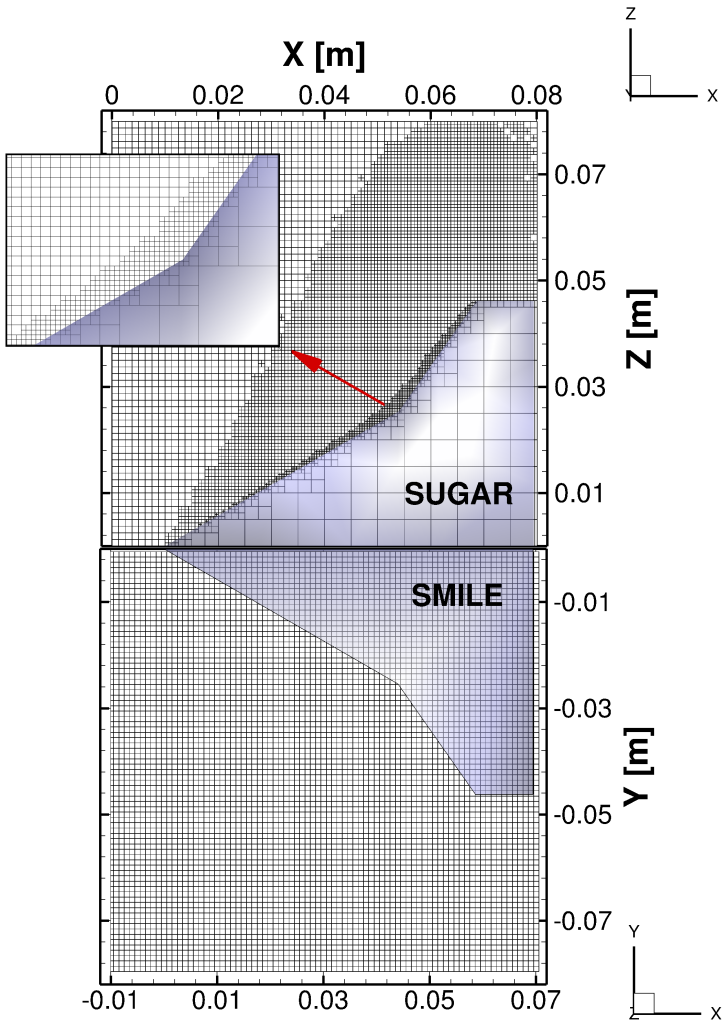
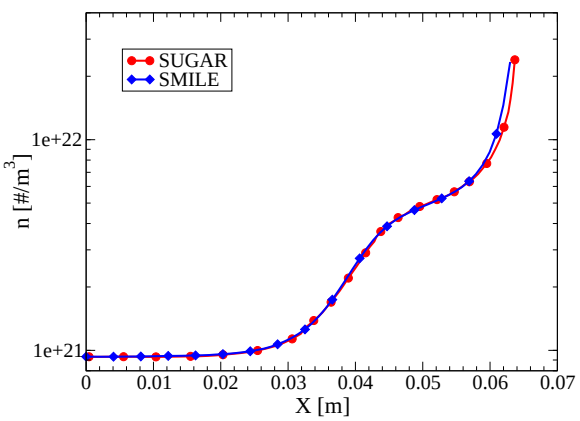


Figure 3.6: Collision mesh comparison for SUGAR and SMILE.

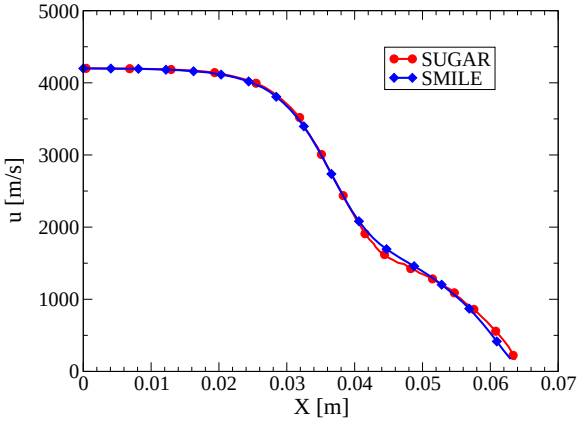
Figure 3.6 shows the collision mesh formed in this simulation at a steady state. For SUGAR mesh, the zoomed section near the surface reveals a fourth level of refinement. Such a high level of refinement would help capture the separation near the hinge, shear layer, and shock-boundary layer interactions for the continuum-like Knudsen number which is our ultimate goal. Note that the SMILE code can achieve two levels of refinement. Therefore, to obtain a similar level of refinement using the SMILE code in the vicinity of the surface, the uniform grid must be created with a cell size equal to that of the third level cell in the SUGAR code. This approach is computationally disadvantages because of the additional efforts

spent in refining the domain that does not need such a high level of refinement. Also, the grid needs to be set considering the lowest resolution, which is hard to predict in such simulations. Thus, the SUGAR code stands out in such extreme cases where multi-scale phenomena need to be captured efficiently.

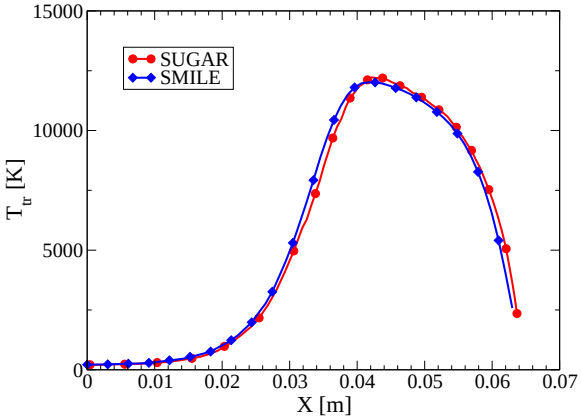
To compare the results more closely, the data is extracted along the dashed red line shown in Fig. 3.3. Figure 3.7 shows the number density, velocity in the x-direction, and translational temperatures for both codes. The number density and velocity profiles are in a good agreement. Small discrepancies are observed in the translational temperature profile which is likely due to the small differences in the sampling cell sizes and the number of particles per cell. Moreover, both the codes predict the highest temperature value of around 13,400 K achieved after the bow shock in the center plane of the wedge.



(a) Number density.



(b) Velocity in X-direction.



(c) Translational temperature.

Figure 3.7: Comparison of macroparameters for the flow of argon over a double-wedge.

In future, the SUGAR code will be used for simulating the experimental conditions and compared with the SMILE code. The recent study has been reported by Tumuklu et al. [9] that discusses the application

of the SMILE code for such continuum-like cases using argon, nitrogen, and the air.

3.4 Hypersonic Flow Simulation Using Diatomic Gases

After the aforementioned validation, rotational relaxation is modeled to simulate diatomic gases (having two rotational degrees of freedom). The temperature is kept low so that the vibrational modes are inactive for the cases considered here.

3.5 CASE III: Rotational Relaxation Using Heat Bath of a Simple Gas

To test the inelastic collision model based on the BL implementation, a simple case of a homogeneous gas [1] was simulated using the majorant frequency scheme. If the initial translational and rotational temperature are 500 and 0 K, respectively, then the gas should relax to an overall temperature of 300 K. The analytical expressions are,

$$T_{tr} = 300 + 200 \exp\Big\{ - \frac{\nu t}{Z_r} \Big\}$$

$$T_{rot} = 300 \Big\{ 1 - \exp\Big(- \frac{\nu t}{Z_r} \Big) \Big\}$$

The simulation conditions are described in Table 3.3. The results obtained using DSMC for the relaxation process are compared with the predictions of analytical expressions in Fig. 3.8. It shows that the relaxation rates obtained with the BL implementation match with the rates predicted by the analytical expressions. The collision frequency ν in the expression is calculated for each cell at each time step using the majorant frequency scheme.

Table 3.3: Numerical parameters for the heat bath of a simple diatomic gas.

Parameters	**SUGAR**
Number Density [#/m^3]	1.0E20
FNUM	1.0E15
Time Step [s]	2.0E-05
Mass [Kg]	5.0E-26
Species Diameter [m]	3.5E-10
Rotational Degrees of Freedom	2
Viscosity Index (for VHS)	0.75
Rotational Number (Constant)	5
Time Steps for the Relaxation	100
Sampling Start	100
Sampling End	1000
Simulation Domain [m]	1 x 1 x 1

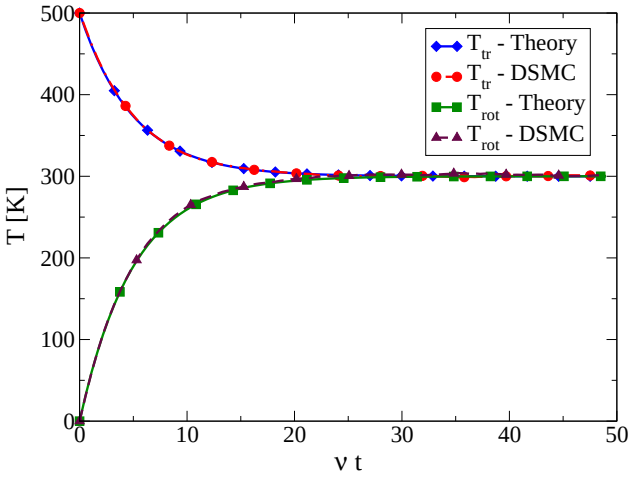


Figure 3.8: Rotational relaxation in a heat bath of a simple diatomic gas.

3.6 CASE IV: Flow of Nitrogen at Knudsen Number 0.277 Over a Hemisphere

A case with a high Knudsen number is chosen because the degree of non-equilibrium is high, thus, giving a clear indication of any possible differences in the relaxation model. The number density is ten times smaller than the previous cases. The hemisphere geometry used for the this case is the same as used earlier. The input conditions are tabulated in Table 3.4.

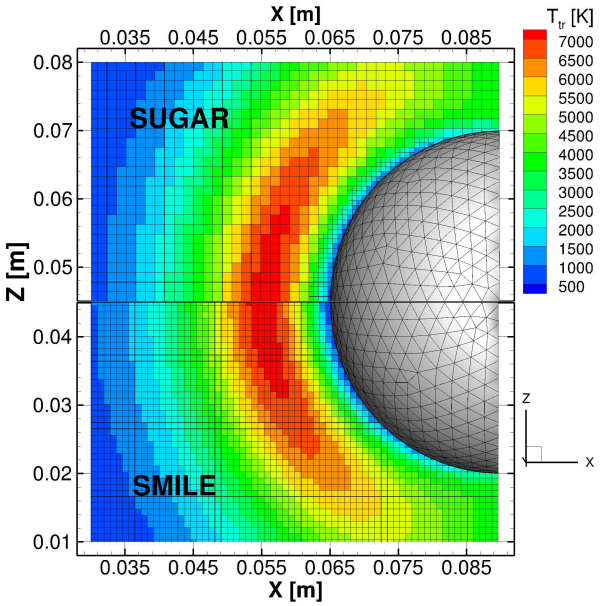
Table 3.4: Numerical parameters for the flow of nitrogen over a hemisphere.

Parameters	**SUGAR**
Number Density [#/m^3]	9.33E19
FNUM	4.0E09
Freestream Temperature [K]	200
Freestream Velocity [m/s]	4200
Time step [s]	1.0E-07
Surface Temperature [K]	200
α_t & α_e	1
Elastic Collision Model	VHS
Viscosity Index	0.74
Number of Samples	22000
Smallest Cell Size on the V-Mesh	7.0132E-04
Number of Particles	22,066,000
Number of Processors Used	256
Time Required for sampling [min]	493

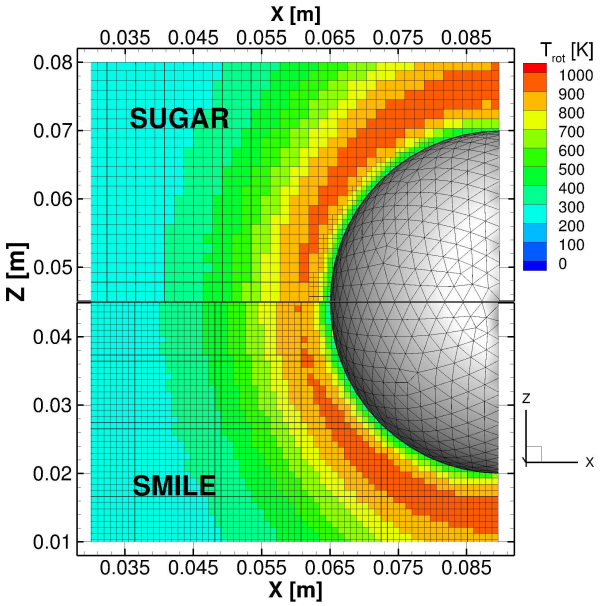
For the SMILE run the number of particles are 23,754,496, smallest cell size on the V-Mesh is 9.0E-04, and time required for sampling is 56 min with 256 processors.

Figure 3.9 shows the comparison of contour plots obtained using the SUGAR and SMILE code for this case. The physical phenomenon is same as explained for the argon case. However, since nitrogen also has two rotational degrees of freedom, the reduced kinetic energy is shared between translational and rotational modes. The flow exhibits translational as well as rotational non-equilibrium after encountering the bow shock. The highest translational temperature achieved is lower than the previous case of argon flow. Since, the number density is low, the computational particles are fewer in number after the shock as compared to previous cases. This results in fewer collisions that causes increased shock stand-off distance of 0.022 m. In this case, particle-surface interactions dominate over particle-particle interactions because the flow is rarefied. A fourth level of refinement is observed as a result of the increase in the number density near the

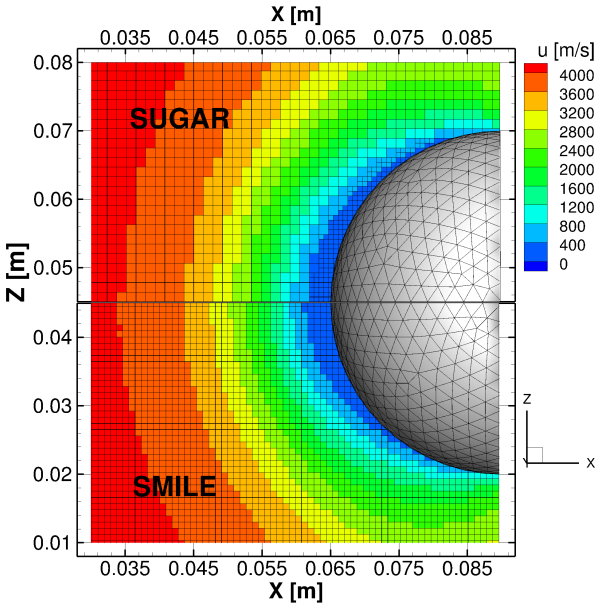
surface.



(a) Translational temperature.

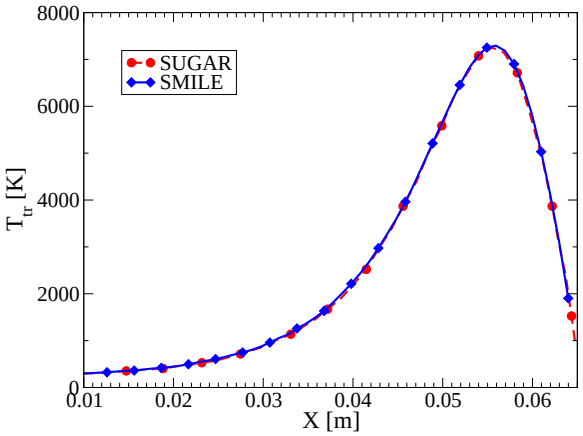


(b) Rotational temperature.

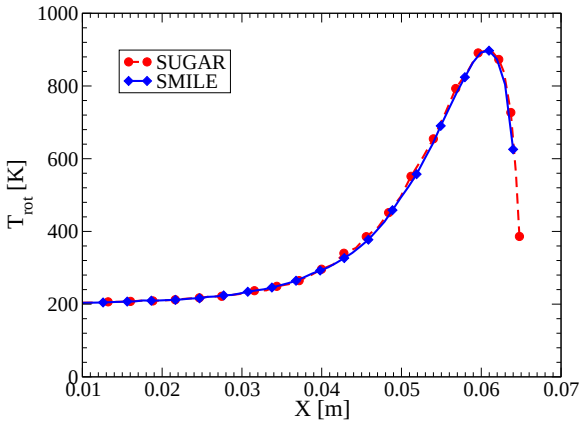


(c) Velocity in X-direction.

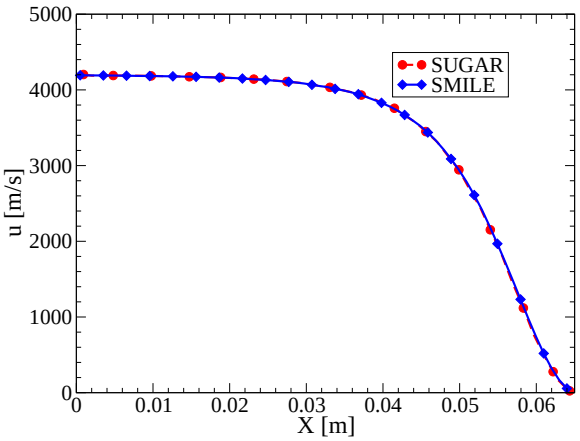
Figure 3.9: Comparison of contours for the flow of nitrogen over a hemisphere.



(a) Translational temperature.



(b) Rotational temperature.



(c) Velocity in X-direction.

Figure 3.10: Comparison of macroparameters for the flow of nitrogen over a hemisphere.

Line plots obtained along the stagnation line are shown in Fig. 3.10. The change in macroparameters is relatively gradual as compared to previous cases. The results are in good agreement for translational temperature and velocity, however, the temperature slip predicted by the SUGAR code at the wall is lower than the SMILE code by 44.44% lower in case of translational temperature and 48% lower in case of rotational temperature. This is because the AMR allows higher levels of refinement thus capturing the temperature near the wall more accurately.

3.7 Case-V: Flow of Nitrogen at Knudsen Number 0.02 Over a Double Wedge

After the successful verification of the flow of nitrogen over a hemisphere at high non-equilibrium conditions, the code was applied to the double-wedge configuration shown in Fig. 3.3 at a lower Knudsen number of 0.02. Note that the Knudsen number for the actual experimental case is 100 times lower than this. Therefore, before jumping directly to a case where the shock-shock interactions and separations are involved, the Knudsen number was lowered gradually and results were compared. The input conditions are listed in Table 3.5. Figure 3.11 shows the comparison of rotational temperature and velocity in a stream wise direction for the SUGAR and SMILE code which is in exact agreement. The detailed comparisons were made by extracting the values of macroparameters along the dashed red line that passes through the plane of symmetry as shown in Fig. 3.12. Observations show that the number density, velocity, rotational temperature are in exact agreement with the results obtained from SMILE. Furthermore, this case was run after the implementation of various surface coefficients such as heat transfer and friction coefficient which were then plotted in Fig. 3.13 by extracting the values along the data line that passes through the plane $Y = 0.025\ m$ over the lower and upper wedge surface. These plots are in fairly good agreement with small discrepancies on the upper wedge which can be attributed to the accurate cut-cell volume calculation in the SUGAR code.

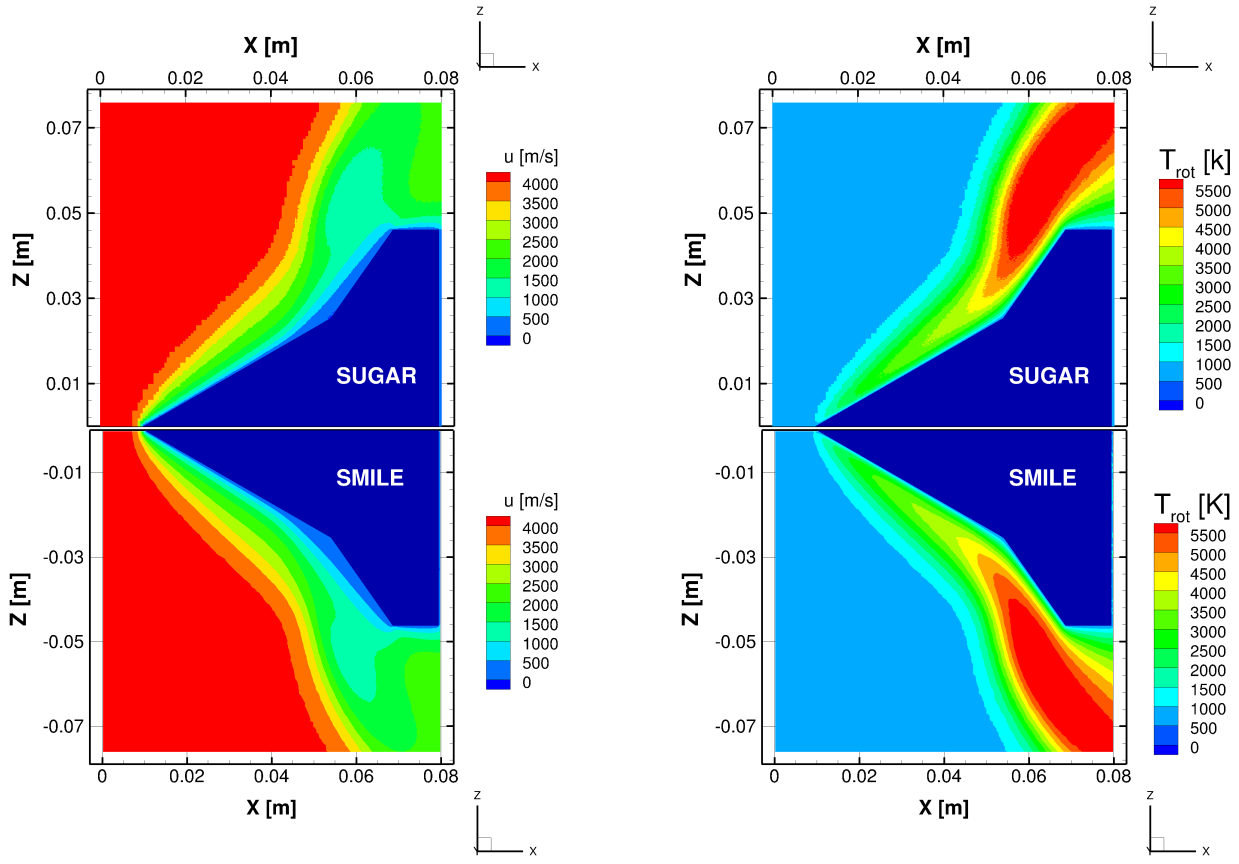
As mentioned in Table 3.5, the level of refinement for this case is seven for the C-Mesh. This is because the refinement criterion for the C-Mesh is kept such that the collision cells stop refining only when the cell size is less than half of the mean-free-path. However, it may cause the region very close to the geometry to be refined to an extent that the cells end up with no particles. In such a case, one has no choice but to restart the simulation by increasing the number of particles by reducing the FNUM. However, once an Octree cell refines, it forms eight children. Therefore, in order to put twice the number of particles in the smallest cell, we need to put eight times more particles in the simulation domain everywhere. Apart from that, since the cell size is reduced, particles need to be moved slowly, i.e., the time step has to be halved, in order to keep the local mean collision time higher. Therefore, theoretically, the computational time increases by a factor of 2^4 with each additional level of refinement. This factor can be reduced by noting that the gradient does not vary much in the span-wise direction for the wedge case. Therefore, if the cells in this direction are elongated by twice the original length, then the above factor would be reduced by two. This strategy has been implemented in the current run and instead of keeping FNUM at $2.0\text{x}10^{10}$, it is kept at $4.0\text{x}10^{10}$ and it is ensured that the simulation satisfies all the DSMC criteria. Note that there are some limitations to the

amount by which the cells can be elongated. First, because of the Octree structure, the maximum possible elongation obtained is given by, $log_2 S$, where S is the number of Octree cells in the span-wise direction such that the Octree cells are cubic. For example, if originally there are 12 Octree cells in the span-wise direction, then theoretically it can be elongated just 3.5 times the original length. However, for MPI communication functions to work, it is necessary to have cells in the span-wise direction to be divisible by two. Therefore, in the above example, one can only reduce the number of Octree cells in the span-wise direction to six, achieving the elongation by a factor of two.

Table 3.5: Numerical parameters for the flow of nitrogen over a double-wedge.

Parameters	**SUGAR**
Number Density [#/m^3]	9.33E20
FNUM	4.0E10
Freestream Temperature [K]	710
Freestream Velocity [m/s]	4200
Time step [s]	2.0E-07
Rotational Number	15
α_t&α_e	1
Surface Temperature [K]	298.5
Elastic Collision Model	VHS
Viscosity Index	0.74
Sampling Start	4000
Sampling End	22000
Smallest Cell Size [m]	7.8125E-05 (7th level)
Number of Particles	15,851,960
Number of Processors Used	512
Time Required for sampling [min]	165

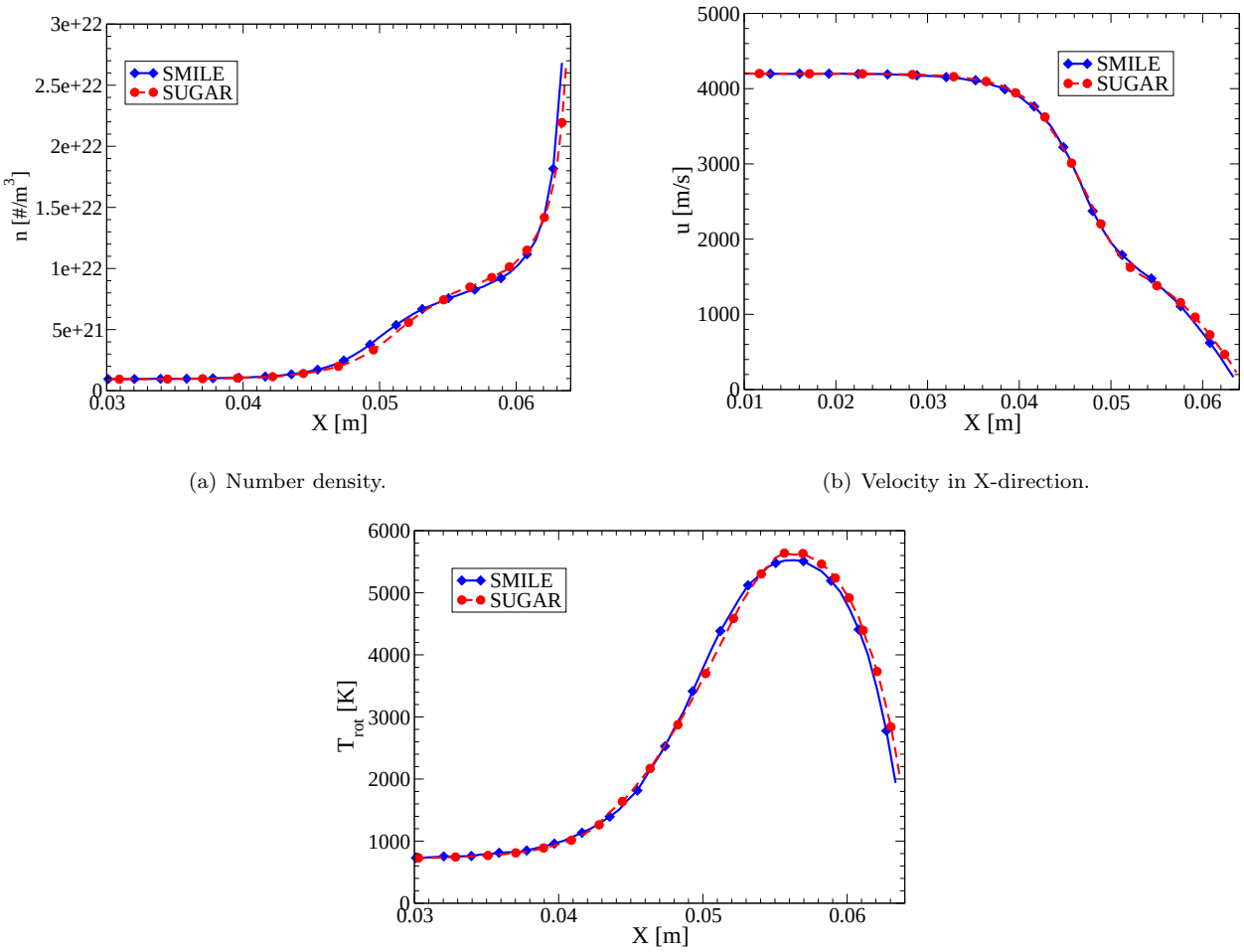
For the SMILE run the number of cells in X, Y, and Z direction are 180, 135, and 180, respectively, the number of particles are 16,370,000 and time taken for sampling is 40 min using 640 processors.



(a) Velocity in X-direction.

(b) Rotational temperature.

Figure 3.11: Comparison of macroparameters for the flow of nitrogen over a double-wedge.

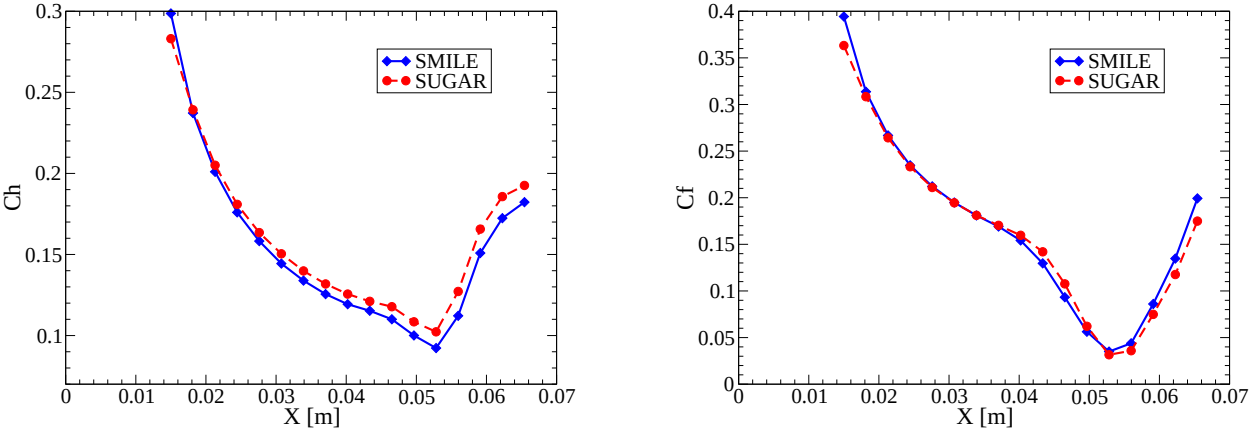


(a) Number density.

(b) Velocity in X-direction.

(c) Rotational temperature.

Figure 3.12: Comparison of macroparameters for the flow of nitrogen over a double-wedge.



(a) Heat transfer coefficient.

(b) Friction coefficient

Figure 3.13: Comparison of surface coefficients for the flow of nitrogen over a double-wedge.

Chapter 4

Conclusion and Future Work

The presented work describes in detail the development strategies used for an Octree based DSMC approach such as the AMR implementation emphasizing octal structure for capturing multi-scale physics, scalability and performance study without and with graph partitioners (through the Zoltan framework), a cut-cell algorithm, an algorithm for improving particle-surface interaction, heat flux computation, and the BL rotational relaxation model for simulating diatomic molecules. In the preliminary performance study described in Sec. 2.2, the major reasons for the SUGAR code being slower than SMILE were the high level of refinement obtained over a major portion of the domain and expensive pointer based tree traversal. Maximum scalability of 335 and 350 was obtained using a 2-D blocking algorithm for a flow over a double-wedge and a hemisphere, respectively. These studies showed that the SUGAR code is potentially scalable, however, in order to simulate continuum-like conditions, it needs to scale to thousands of processors for which a 2-D blocking algorithm does not give improvement in speed-up. The primary obstacle in achieving more scalability was the multi-scale nature of the problem, where a high degree of load imbalance is observed. Further, the dynamic movement of particles in the DSMC method demands frequent communication between processors. Therefore, state-of-the-art graph partitioners such as Parmetis and Scotch were tested to improve the scalability of the SUGAR for more than 256 processors. Preliminary study using these partitioners was performed on a 2-D and 3-D sample problems. The results revealed that Parmetis and Scotch both distribute the computational load among available processors equally well, however, Scotch gave approximately 32% reduction in communication links as compared to Parmetis algorithms for a 16 nodal cubic Cartesian mesh with varying weights on cells using 256 processors. Based on these studies Scotch was implemented in the SUGAR code through Zoltan library. However, it gave no improved performance for a double wedge problem with a Knudsen number of 0.020. While using these partitioners, it is necessary to correctly represent the computational load of each Octree cell in terms of some weight that could be a function of the number of particles and leaf cells. Three different formulas were tested for assigning weights to the Octree cells, however, all of them increased the computational time as compared to a 2-D blocking algorithm. It was observed that a fraction of 0.24 of the volume of the domain contains high levels of refinement which gives less flexibility to

a graph partitioner to effectively partition the domain. Furthermore, in order to improve the performance of the code, the SUGAR code was compared to the performance of the SMILE and in-house CHAOS code by profiling it on a single processor. It was observed that the main drawback of the SUGAR code being slower than SMILE is the way the Octree is being traversed in a pointer based manner. An equivalent Octree DSMC code with a linearized representation of Octree cells using the Morton-Z space filling curve, implemented by a colleague, Revathi Jambunathan, gave better performance than SMILE. Based on these comparisons, Sec. 2.6 detailed the comparison of projected time taken by the SUGAR code for the experimental case based on a simplified assumption that the computational time is just a function of the number of particles. It also described that if the Morton-Z based linearized Octree structure is implemented, the projected time for the target case would be comparable to the SMILE code, whereas the improved SUGAR code would have an additional advantage of being more scalable than the SMILE code. Therefore, based on Sec. 2.5 and 2.6, it was concluded that the Morton-Z based Octree structure would give significant reduction in the computational time.

As described in Sec. 2.7, the SUGAR code is equipped with a sophisticated cut-cell algorithm that can handle any intricate geometry. It accurately calculates the volume of the cells cut by the geometry which is essential for correctly capturing the physics near the geometry. Moreover, Sec. 2.8 described the efforts made to improve the efficiency of the gas-surface interaction routine. Particles make use of the geometric locality to find the target surface panel thus, checking only a limited amount of surface panels for possible intersection. It was also noted that there are advantages of broadcasting the geometry information to each processor such as reduction in the communication during multiple reflections and ease in surface coefficient computation. Section 2.9 detailed the computation of surface coefficients in an MPI parallelized domain and Sec. 2.10 explained the correct implementation of BL model for simulating rotational relaxation in inelastic collisions.

The SUGAR code has been verified for the important elements of the DSMC procedure such as, an elastic collision model, and a cut-cell algorithm by comparing the results for a case of argon flow at a Knudsen number of 0.0277 over a hemisphere with the 3-D version of the SMILE code. Followed by this study, a relatively easy case with a Knudsen number of 0.02 was simulated over a double-wedge. Although the flow physics did not involve separation, sheer layer, or the Edney type of shock interactions, the basic flow features such as oblique and bow shock interaction can be seen in the solutions. Successful results from this study is the first step towards the complex problem of simulating flow over a double-wedge at continuum-like conditions. This case demonstrated the ability of the SUGAR code to refine the grid in the region closer to the surface, which would be advantageous in simulating difficult cases that involve complex shock-shock

interactions and flow separation. Furthermore, the code was extended to allow for the simulation of diatomic species. The preliminary case of heat bath showed exact agreement with the analytical expressions, thus verifying the majorant frequency implementation and the hierarchical BL model. The relaxation algorithms were further verified by simulating the case for a flow over a hemisphere for a Knudsen number of 0.277 by using nitrogen which gave a good overall agreement for macroparameters. The SUGAR results predicted more temperature slip in the region near the surface which was attributed to a high level of refinement of an Octree mesh. Finally, a high density case having a Knudsen number of 0.02 was simulated using a flow of nitrogen over a double-wedge and plots for comparing macroparameters as well as distribution of surface coefficients were shown to be in exact agreement. For this case, a rigorous refinement criterion of mean-free-path was used over the C-Mesh which resulted in a seventh level of refinement near the geometry.

To simulate the experimental case having 100 times lower Knudsen number, very high levels of refinement are expected in the vicinity of the wedge, and to satisfy the DSMC criteria in the finest cells, particles in the entire domain are to be increased which renders the DSMC simulation computationally impossible with current pointer based tree traversal. Based on aforementioned conclusions of the studies in Sec. 2.4, 2.5 and 2.6, Morton-Z based linear Octree structure is currently being implemented in the SUGAR code. To further increase the performance, a spatially varying time step will be employed in the future. This would allow the time step to not be limited by the mean collision time of the smallest leaf cell, but be based on each individual cell. Therefore, particles would be able to move faster in larger cells, achieving an accelerated simulation time. After these changes, the current MPI parallelized code structure will be extended using CUDA to make use of heterogeneous GPU architectures.

Appendix

Surface Parameters

Heat transfer coefficient:

$$Q_i = \frac{FNUM}{\Delta T}\Big(\frac{m}{2}\sum_{j=1}^{N_{hit}}\vec{V_j}^2 + \sum_{j=1}^{N_{hit}}(\varepsilon_{rj}+\varepsilon_{vj})\Big)_i$$

$$Q_r = \frac{FNUM}{\Delta T}\Big(\frac{m}{2}\sum_{j=1}^{N_{hit}}\vec{V_j}^2 + \sum_{j=1}^{N_{hit}}(\varepsilon_{rj}+\varepsilon_{vj})\Big)_r$$

$$Q = Q_i - Q_r$$

$$c_h = \frac{Q}{\frac{1}{2}\rho_\infty V_\infty^3 S_p}$$

Pressure coefficient:

$$p_i = \frac{FNUM\, m}{S_p \Delta T}\Big(\sum_{j=1}^{N_{hit}}(\vec{V_j} cos\Theta_j)_i$$

$$p_r = \frac{FNUM\, m}{S_p \Delta T}\Big(\sum_{j=1}^{N_{hit}}(\vec{V_j} cos\Theta_j)_r$$

$$p = p_i - p_r$$

$$c_p = \frac{p}{\frac{1}{2}\rho_\infty V_\infty^2}$$

Skin friction coefficient in X-direction:

$$f_{xi} = \frac{FNUM\, m}{S_p \Delta T}\Big(\sum_{j=1}^{N_{hit}}\overrightarrow{V_{ji}} - \overrightarrow{n_p}\Big(\Big(\sum_{j=1}^{N_{hit}}\overrightarrow{V_{ji}}\Big)\cdot\overrightarrow{n_p}\Big)\Big)_x$$

$$f_{xr} = \frac{FNUM\, m}{S_p \Delta T}\Big(\sum_{j=1}^{N_{hit}}\overrightarrow{V_{jr}} - \overrightarrow{n_p}\Big(\Big(\sum_{j=1}^{N_{hit}}\overrightarrow{V_{ji}}\Big)\cdot\overrightarrow{n_p}\Big)\Big)_x$$

$$f_x = f_{xi} + f_{xr}$$

$$c_{fx} = \frac{f_x}{\frac{1}{2}\rho_\infty V_\infty^2}$$

Skin friction coefficient in Y-direction:

$$f_{yi} = \frac{FNUM\, m}{S_p \Delta T}\Big(\sum_{j=1}^{N_{hit}} \overrightarrow{V_{ji}} - \overrightarrow{n_p}\Big(\Big(\sum_{j=1}^{N_{hit}} \overrightarrow{V_{ji}}\Big)\cdot \overrightarrow{n_p}\Big)\Big)_y$$

$$f_{yr} = \frac{FNUM\, m}{S_p \Delta T}\Big(\sum_{j=1}^{N_{hit}} \overrightarrow{V_{jr}} - \overrightarrow{n_p}\Big(\Big(\sum_{j=1}^{N_{hit}} \overrightarrow{V_{ji}}\Big)\cdot \overrightarrow{n_p}\Big)\Big)_y$$

$$f_y = f_{yi} + f_{yr}$$

$$c_{fy} = \frac{f_y}{\frac{1}{2}\rho_\infty V_\infty^2}$$

Skin friction coefficient in Z-direction:

$$f_{zi} = \frac{FNUM\, m}{S_p \Delta T}\Big(\sum_{j=1}^{N_{hit}} \overrightarrow{V_{ji}} - \overrightarrow{n_p}\Big(\Big(\sum_{j=1}^{N_{hit}} \overrightarrow{V_{ji}}\Big)\cdot \overrightarrow{n_p}\Big)\Big)_z$$

$$f_{zr} = \frac{FNUM\, m}{S_p \Delta T}\Big(\sum_{j=1}^{N_{hit}} \overrightarrow{V_{jr}} - \overrightarrow{n_p}\Big(\Big(\sum_{j=1}^{N_{hit}} \overrightarrow{V_{ji}}\Big)\cdot \overrightarrow{n_p}\Big)\Big)_z$$

$$f_z = f_{zi} + f_{zr}$$

$$c_{fz} = \frac{f_z}{\frac{1}{2}\rho_\infty V_\infty^2}$$

Friction coefficient:

$$c_p = (c_{fx}^2 + c_{fy}^2 + c_{fz}^2)^{1/2}$$

Mass flux:

$$J = \frac{FNUM\, N_{hit} m}{S_p \Delta T}$$

Kinetic energy flux:

$$J_{ti} = \frac{FNUM\, m}{2 S_p \Delta T} \sum_{j=1}^{N_{hit}} \overrightarrow{V_{ji}}^2$$

$$J_{tr} = \frac{FNUM\, m}{2 S_p \Delta T} \sum_{j=1}^{N_{hit}} \overrightarrow{V_{jr}}^2$$

Internal energy flux:

$$J_{ti} = \frac{FNUM}{2 S_p \Delta T} \sum_{j=1}^{N_{hit}} (\varepsilon_{rj} + \varepsilon_{vj})_i$$

$$J_{tr} = \frac{FNUM}{2 S_p \Delta T} \sum_{j=1}^{N_{hit}} (\varepsilon_{rj} + \varepsilon_{vj})_r$$

References

[1] G. A. Bird, *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*. Oxford University Press, Oxford, 2nd ed., June 1994.

[2] M. S. Ivanov, A. V. Kashkovsky, S. F. Gimelshein, G. N. Markelov, A. A. Alexeenko, Y. A. Bondar, G. A. Zhukova, S. B. Nikiforov, and P. V. Vaschenkov, "Smile system for 2d/3d dsmc computations," in *Proceedings of the 25th International Symposium on Rarefied Gas Dynamics, St. Petersburg, Russia, edited by M. S. Ivanov and A. K. Rebrov*, pp. 539–544, Publishing House of Siberian Branch of the Russian Academy of Science, Novosibirsk, 2007, July 2006.

[3] G. J. Lebeau and I. F. E. Lumpkin, "Application highlights of the dsmc analysis code (dac) software for simulating rarefied flows," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, pp. 595–609, December 2001.

[4] D. Gao, C. Zhang, and T. E. Schwartzentruber, "Particle simulations of planetary probe flows employing automated mesh refinement," *Journal of Spacecraft and Rockets*, vol. 48, pp. 397–405, May-June 2011.

[5] S. Dietrich and I. D. Boyd, "Scalar and parallel optimized implementation of the direct simulation monte carlo method," *Journal of Computational Physics*, vol. 126, pp. 328–342, July 1996.

[6] B. Korkut, P. Wang, Z. Li, and D. A. Levin, "Three dimensional simulation of ion thruster plumes with amr and parallelization strategies," tech. rep., San Jose, CA, July 2013.

[7] A. B. Swantek and J. M. Austin, "Heat transfer on a double wedge geometry in hypervelocity air and nitrogen flows," tech. rep., Nashville, Tennessee, January 2012.

[8] V. N. Patil, D. A. Levin, S. G. Gimelshein, and J. M. Austin, "Study of shock-shock interactions for the het facility double wedge configuration using the dsmc approach," tech. rep., Red Hook, NY, June 2013.

[9] O. Tumuklu, D. A. Levin, S. G. Gimelshein, and J. M. Austin, "Shock-shock interactions for double wedge configuration in different gases," tech. rep., Kissimmee, Florida, January 2015. presented in current conference.

[10] S. S. Sawant, B. Korkut, O. Tumuklu, and D. A. Levin, "Development of an amr octree dsmc approach for shock dominated flows," tech. rep., Kissimmiee, FL, January 2015.

[11] "Cray Performance Measurement and Analysis Tools (CPMAT)." `https://bluewaters.ncsa.illinois.edu/cpmat`.

[12] R. Klein, "Star formation with 3-D adaptive mesh refinement: the collapse and fragmentation of molecular clouds," *Journal of Computational and Applied Mathematics*, vol. 109, pp. 123–152, 1999.

[13] L. Botti, M. Piccinelli, B. Iordache, A. Remuzzi, and L. Antiga, "An adaptive mesh refinement solver for large-scale simulation of biological flows," *International Journal for Numerical Methods in Biomedical Engineering*, vol. 26, pp. 86–100, 2009.

[14] M. Berger and P. Colella, "Local adaptive mesh refinement for shock hydrodynamics," *Journal of Computational Physics*, vol. 82, pp. 64–84, 1989.

[15] I. Nompelis and T. Schwartzentruber, "A parallel implementation strategy for multi-level cartesian grid based DSMC codes," *AIAA Paper 2013-1204*, 2013.

[16] R. Arslanbekov, V. Kolobov, J. Burt, and E. Josyula, "Direct simulation monte carlo with octree cartesian mesh," *AIAA Paper 2012-2990*, 2012.

[17] C. Xu and F. C. Lau, *Load balancing in parallel computers: theory and practice*. Springer Science & Business Media, 1997.

[18] G. Karypis and V. Kumar, "MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0." `http://www.cs.umn.edu/~metis`, 2009.

[19] F. Pellegrini, "Scotch and libscotch 5.1 user's guide." `http://www.labri.fr/perso/pelegrin/scotch/`, 2008.

[20] E. Boman, K. Devine, L. A. Fisk, R. Heaphy, B. Hendrickson, V. Leung, C. Vaughan, U. Catalyurek, D. Bozdag, and W. Mitchell, "Zoltan home page." `http://www.cs.sandia.gov/Zoltan`, 1999.

[21] E. Boman, K. Devine, L. A. Fisk, R. Heaphy, B. Hendrickson, C. Vaughan, U. Catalyurek, D. Bozdag, W. Mitchell, and J. Teresco, *Zoltan 3.0: Parallel Partitioning, Load-balancing, and Data Management Services; User's Guide*. Sandia National Laboratories, Albuquerque, NM, 2007. Tech. Report SAND2007-4748W `http://www.cs.sandia.gov/Zoltan/ug_html/ug.html`.

[22] K. Devine, E. Boman, L. Riesen, U. Catalyurek, and C. Chevalier, "Getting started with zoltan: A short tutorial," in *Proc. of 2009 Dagstuhl Seminar on Combinatorial Scientific Computing*, 2009. Also available as Sandia National Labs Tech Report SAND2009-0578C.

[23] J. Wu, K. Tseng, *et al.*, "Concurrent dsmc method using dynamic domain decomposition," in *AIP CONFERENCE PROCEEDINGS*, pp. 406–416, IOP INSTITUTE OF PHYSICS PUBLISHING LTD, 2003.

[24] A. Abou-Rjeili and G. Karypis, "Multilevel algorithms for partitioning power-law graphs," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pp. 10–pp, IEEE, 2006.

[25] R. Jambunathan and D. A. Levin, "A hybrid cpu-gpu parallel octree direct simulation monte carlo approach," in *22nd AIAA Computational Fluid Dynamics Conference*, p. 3057, 2015.

[26] S. Zabelok, R. Arslanbekov, and V. Kolobov, "Adaptive kinetic-fluid solvers for heterogeneous computing architectures," *arXiv preprint arXiv:1503.00707*, 2015.

[27] J. Burkardt, "Stereolithography Interface Specification." `http://people.sc.fsu.edu/~jburkardt/cpp_src/stla_io`, 1989.

[28] M. J. Aftosmis, M. J. Berger, and J. E. Melton, "Robust and efficient cartesian mesh generation for component-based geometry," *AIAA Journal*, vol. 36, pp. 952–960, June 1998.

[29] C. Zhang and T. E. Schwartzentruber, "Robust cut-cell algorithms for dsmc implementations employing multi-level cartesian grids," *Journal of Computers and Fluids*, vol. 69, pp. 122–135, October 2012.

[30] I. E. Sutherland and G. W. Hodgman, "Reentrant polygon clipping," *Communications of the ACM*, vol. 17, pp. 32–42, January 1974.

[31] J. O'Rourke, *Computational Geometry in C*. Cambridge University Press, 1993.

[32] W. Schroeder, "The Visualization Toolkit." `http://www.vtk.org`, 2003.

[33] C. Cercignani, *Rarefied gas dynamics: from basic concepts to actual calculations*, vol. 21. Cambridge University Press, 2000.

[34] K. Koura and H. Matsumoto, "Variable soft sphere molecular model for inverse-power-law or lennard-jones potential," *Physics of Fluids A: Fluid Dynamics (1989-1993)*, vol. 3, no. 10, pp. 2459–2465, 1991.

[35] C. Borgnakke and P. S. Larsen, "Statistical collision model for monte carlo simulation of polyatomic gas mixture," *Journal of computational Physics*, vol. 18, no. 4, pp. 405–420, 1975.

[36] J. Parker, "Rotational and vibrational relaxation in diatomic gases," *Physics of Fluids (1958-1988)*, vol. 2, no. 4, pp. 449–462, 1959.

[37] F. E. Lumpkin III, B. L. Haas, and I. D. Boyd, "Resolution of differences between collision number definitions in particle and continuum simulations," *Physics of Fluids A: Fluid Dynamics (1989-1993)*, vol. 3, no. 9, pp. 2282–2284, 1991.