A Parallel Solution Adaptive Implementation of the Direct Simulation Monte Carlo Method

Stuart Jackson Wishart

A thesis submitted in fulfilment
of the requirements for the degree of
Doctor of Philosophy

Department of Aerospace, Mechatronic and Mechanical Engineering
The University of Sydney

August 2004

Declaration

This thesis is submitted to The University of Sydney in fulfilment of the requirements for the degree of Doctor of Philosophy. This thesis is entirely my own work and, except where otherwise stated, describes my own research.

Stuart Wishart

August 30, 2004

Abstract

Stuart Wishart
The University of Sydney

Doctor of Philosophy
August 2004

A Parallel Solution Adaptive Implementation of the Direct Simulation Monte Carlo Method

This thesis deals with the direct simulation Monte Carlo (DSMC) method of analysing gas flows. The DSMC method was initially proposed as a method for predicting rarefied flows where the Navier-Stokes equations are inaccurate. It has now been extended to near continuum flows. The method models gas flows using simulation molecules which represent a large number of real molecules in a probabilistic simulation to solve the Boltzmann equation. Molecules are moved through a simulation of physical space in a realistic manner that is directly coupled to physical time such that unsteady flow characteristics are modelled. Intermolecular collisions and molecule-surface collisions are calculated using probabilistic, phenomenological models. The fundamental assumption of the DSMC method is that the molecular movement and collision phases can be decoupled over time periods that are smaller than the mean collision time.

Two obstacles to the wide spread use of the DSMC method as an engineering tool are in the areas of simulation configuration, which is the configuration of the simulation parameters to provide a valid solution, and the time required to obtain a solution. For complex problems, the simulation will need to be run multiple times, with the simulation configuration being modified between runs to provide an accurate solution for the previous run's results, until the solution converges. This task is time consuming and requires the user to have a good understanding of the DSMC method. Furthermore, the computational resources required by a DSMC simulation increase rapidly as the simulation approaches the continuum regime. Similarly, the computational requirements of three-dimensional problems are generally two orders of magnitude more than two-dimensional problems. These large computational requirements significantly limit the range of problems that can be practically solved on an engineering workstation or desktop computer.

The first major contribution of this thesis is in the development of a DSMC implementation that automatically adapts the simulation. Rather than modifying the simulation configuration between solution runs, this thesis presents the formulation of algorithms that allow the simulation configuration to be automatically adapted during a single run. These adaption algorithms adjust the three main parameters that effect the accuracy of a DSMC simulation, namely the solution grid, the time step and the simulation molecule number density. The second major contribution extends the parallelisation of the DSMC method. The implementation developed in this thesis combines the capability to use a cluster of computers to increase the maximum size of problem that can be solved while simultaneously allowing excess computational resources to decrease the total solution time. Results are presented to verify the accuracy of the underlying DSMC implementation, the utility of the solution adaption algorithms and the efficiency of the parallelisation implementation.

Acknowledgments

I would like to thank all of the people who encouraged and supported me during the undertaking of this research.

Firstly, I would like to thank all the members of the Aeronautical Engineering Department. To my supervisor, Dr. Doug Auld, who set me on this path of research. To my post-graduate colleagues Jeremy Randle, Dr. Hugh Stone and Dr. Nick van Bronswijk, who helped make the experience of this thesis more rewarding. Thanks also to the members of the ACFR who encouraged my research and provided me with good coffee.

I would also like to acknowledge the support of my employer BAE Systems Australia. While they have tried to tempt me away from my research with interesting projects, they have also been supportive of my completion.

My last and greatest thanks is for my family. To my parents and siblings for their encouragement and support throughout the entirety of this project. To my in-laws who kept my strength up with good food and the inquisitive questions. Lastly, I would like to thank my wife Clare who has been steadfast in her support. But most importantly for her love and companionship.

"The end of a matter is better than its beginning, and patience is better than pride."

Ecclesiastes 7:8

Contents

List of Figures

List of Tables

Nomenclature

a	speed of sound
$\mathbf{c}$	molecule position vector, computational space
$\bar{c}'$	average molecule speed
c_f	skin friction coefficient
C_P	coefficient of pressure
d	molecular diameter
F_N	ratio real to simulation molecules
l_m	cell side length
$\bar{l}_m$	average cell side length
Ma	Mach number
N_m	number of molecules
k	Boltzmann's constant, 1.380658×10^{-23} JK^{-1}
Kn	Knudsen number
m	molecule mass
m_r	reduced molecule mass
n	number density
S_{sf}	stream species, of species group, fraction
$\bar{S}$	cell boundary surface area vector
t	time
$\bar{t}_{mc}$	average collision time
T	temperature
$\mathbf{v}$	molecule velocity vector, physical space
V_c	cell volume
$\mathbf{x}$	molecule position vector, physical space
x, y, z	physical domain coordinates
η, ξ, ζ	computational domain coordinates
λ	local mean free path
υ	collision frequency
ω	temperature exponent of the coefficient of viscosity

Subscripts

avg	average
HS	hard sphere molecule model
max	maximum
min	minimum
ref	reference value
rem	remainder
∞	freestream value

Chapter 1

Introduction

1.1 Background and Motivation

This thesis deals with the direct simulation Monte Carlo (DSMC) method of analysing gas flows. The DSMC method was initially proposed as a method for predicting rarefied flows where the Navier-Stokes equations are inaccurate and it has now been extended to near continuum flows. The method models gas flows using simulation molecules which represent a large number of real molecules in a probabilistic simulation to solve the Boltzmann equation. Molecules are moved through a simulation of physical space in a realistic manner that is directly coupled to physical time such that unsteady flow characteristics can be modelled. Intermolecular collisions and molecule-surface collisions are calculated using probabilistic, phenomenological models. The fundamental assumption of the DSMC method is that the molecular movement and collision phases can be decoupled over time periods that are smaller than the mean collision time.

Two obstacles to the wide spread use of the DSMC method as an engineering tool are in the areas of simulation configuration, which is the configuration of the simulation parameters to provide a valid solution, and the time required to obtain a solution. For complex problems[1], the simulation will need to be run multiple times, with the simulation configuration being modified between runs to provide an accurate solution for the previous run's results, until the solution converges. This task is time consuming and requires the user to have a good understanding of the DSMC method. Furthermore, the computational resources required by a DSMC simulation increase rapidly as the simulation approaches the continuum regime. Similarly, the computational requirements of three-dimensional problems are generally two orders of magnitude

[1] The expression "complex problems" refers to problems that contain mixed subsonic and supersonic flows, transition regimes, boundary layers, slip planes or chemically interacting flows.

more than two-dimensional problems. These large computational requirements significantly limit the range of problems that can be practically solved on an engineering workstation or desktop computer.

While there are less computationally expensive alternative methods available for solving near continuum flows, e.g. Navier-Stokes based programs, these methods do not work well rarefaction effects are present. One example of where rarefaction effects are significant is in the micro-electro-mechanical systems (MEMS) [53]. The design of MEMS components requires the simulation of gas flows around microscale structures. It is possible to adapt Navier-Stokes based solvers to solve these types of problems by the use of special boundary conditions. However, this is a time consuming task that requires a very high level of user knowledge, and is not applicable to a standard engineering tool.

For a program to be a useful engineering tool, it should be simple to set up and run, and provide accurate results in a timely manner. It should warn the user when results are expected to be inaccurate and ideally, it should automatically adapt the program parameters to obtain a more accurate solution. However, as Harvey and Gallis state in their review of DSMC validation studies, [25], “writing and running a DSMC code is a demanding task that requires considerable skill, care and experience.”. In the comprehensive presentation of the DSMC method [7] Bird states "The objective is to develop a code that requires only the specification of the boundaries ... and the flow conditions. The program should itself generate the grid and, ideally it should adapt the grid to its optimal form as the flow develops."

1.2 Problem Summary

The aim of this thesis is to develop a DSMC implementation that allows a non-expert user to efficiently solve arbitrary problems with the DSMC method in both serial and parallel environments.

1.3 Principle Contributions

This thesis addresses the issues related to the development of a parallel, solution adaptive direct simulation Monte Carlo implementation. The principle contributions of this thesis arise from the formulation of different solution adaption and parallelisation

algorithms. The integration of these algorithms into a single DSMC implementation results in a significant improvement in performance. The contributions made are:

- A novel molecule movement algorithm is developed which allows the efficient calculation of a molecule's trajectory through an arbitrary quadrilateral grid. This movement algorithm combines the simplicity and accuracy of calculating the molecule movement in physical space with the simple cell indexing afforded by tracking the molecules computational space position. The algorithm achieves this by performing the molecule movement in physical space and then transforming the molecule's final position into computational space. The computational space position is then used to determine the molecule's new cell index and whether any surface/boundary interactions took place over the molecule's trajectory.
- A multi-block grid adaption algorithm is developed. Using the intermediate results of the solution, the grid adaption algorithm calculates the number of cells and the distribution required to ensure that the grid meets the DSMC cell size requirements. The algorithm determines whether the distribution of cell size across a block would be more efficiently represented by splitting the block into two or more blocks and creating the required splits. Furthermore, the algorithm adjusts the ratio of real to simulation molecules to ensure that the number of molecules per cell is sufficient to ensure that the correct collision rate is maintained.
- A parallelised time step adaption algorithm is developed. This algorithm allows the time step of each cell to be set to a locally optimal value while still maintaining the ability to efficiently synchronise the solution process in a parallel implementation. Additionally, the algorithm is formulated such that the calculation of a new time step distribution is performed in parallel.
- Two different but complementary parallelisation methods are integrated. This integration of complementary parallelisation methods allows the use of the optimal parallel configuration for the available computational resources. Furthermore, the parallel implementation allows the parallel distribution to be dynamically changed to account for changes in the solution.
- An improved implementation of the stop-at-rise algorithm is developed. This modified algorithm incorporates information relating to the total solution progress

with the current solution performance to more determine whether it is efficient to perform a domain decomposition repartitioning.

1.4 Outline

Chapter 2 presents an outline of the direct simulation Monte Carlo method and summarises the implementation issues that effect the accuracy and validity of the method.

Chapter 3 presents an analysis of the different movement algorithms. A novel movement algorithm is developed which allows the trajectory calculation of a molecule through an arbitrary quadrilateral grid.

Chapter 4 details the program structure of the baseline serial implementation.

Chapter 5 develops the solution adaption procedures implemented to ensure that the simulation is configured to meet the requirements for an accurate simulation.

Chapter 6 discusses two different parallelisation methods and details their integration into the serial implementation.

Chapter 7 presents the verification and results of the direct simulation Monte Carlo implementation developed in the previous chapters.

Finally, Chapter 8 presents conclusions and directions for future research.

Chapter 2

The Direct Simulation Monte Carlo Method

2.1 Introduction

The direct simulation Monte Carlo (DSMC) method uses probabilistic simulation to solve the Boltzmann equation. Bird states in [9] "The direct simulation of the physical processes contrasts with the general philosophy of computational fluid dynamics which is to obtain solutions of the mathematical equations that model the processes.". Initially proposed as a method for predicting rarefied flows where the Naiver-Stokes (NS) equations are inaccurate, [11], it has now been shown to be accurate in the continuum regime, [2]. Currently the DSMC method has been applied to the solution of flows ranging from estimation of the Space Shuttle re-entry aerodynamics, [42], to the modelling micro-electro-mechanical systems (MEMS), [53].

2.2 DSMC Theory

The DSMC method models gas flows using simulation molecules[2] that represent a large number of real molecules. Molecules are moved through a simulation of physical space in a realistic manner that is directly coupled to physical time such that unsteady flow characteristics can be modelled. Intermolecular collisions and molecule-surface collisions are calculated using probabilistic, phenomenological models. A grid is used for associating molecules that are in close spatial proximity. Molecules in the same grid cell are used as possible collision partners as well as for sampling molecules states to calculate the macroscopic properties in the cell. The fundamental assumption of the

[2] Note for brevity, the term “molecule” shall be used synonymously for the term “simulation molecule” unless it is unclear from the context. Where a simulation molecule represents a number of real, physical molecules in the DSMC simulation.

DSMC method is that the molecular movement and collision phases can be decoupled over time periods that are smaller than the mean collision time.

The most comprehensive presentation of the DSMC method is given in Bird's monograph "Molecular Gas Dynamics and the Direct Simulations of Gas Flows" [7]. This book is a sequel to Bird's first book on the subject "Molecular Gas Dynamics" [6] that was published in 1976. This updated work details the theory and implementation of the phenomenological approach upon which the DSMC method is based. The book covers all areas of DSMC theory from the derivation of the underlying theory through to example implementations of the method for the solution of three-dimensional problems. Bird keeps a list of developments in DSMC theory that have occurred since the publication of his book on his web site [10].

There are a number of published papers that review the current status of the DSMC method, [3], [8], [25]. Results of validation studies are available in the following papers [25], [43], [61].

2.3 Method Outline

The steps involved in a scalar DSMC simulation are:

1. Load the simulation configuration: Load, or input, the simulation configuration parameters used to define the geometry, freestream conditions and initial flow state.
2. Initialise simulation: If the initial flow state is a uniform flow, then the simulation domain is populated with molecules. Alternatively, if the simulation is restarting from a previous saved state, then the saved molecule states are loaded.
3. Enter molecules through boundaries: All simulation boundaries that allow molecule influx, ie. external freestream boundaries or jet boundaries, are processed to calculate the number and state of molecules that enter during the time step.
4. Move all molecules: Each molecule is moved a distance equal to the sum of its velocity and the time step. Any surface collisions detected during the molecules movement are processed. The final cell location of each molecule is determined and stored. Some DSMC implementations break out the process of determining the final cell location into a separate step referred to as indexing.

5. Calculate molecule collisions: Each cell is processed to determine the number of possible collisions that could take place in the time step. The number of possible collision pairs is a function of the number of molecules in the cell, the average number of molecules normally in the cell, the cell volume and the time step. Collision pairs are chosen at random from the molecules in the cell. However, this selection process is biased towards minimising the separation between molecules. A collision is accepted or rejected based upon a probability proportional to the molecules relative velocity and molecular properties. If a collision is accepted, then the kinetic and internal energies of the molecules are redistributed.
6. Sample molecules: The state of all the molecules in the cell at the end of a time step are sampled for the purpose of calculating cell macroscopic properties. Sampling is generally not performed every time step to reduce the correlation between samples. In addition, different sampling strategies are used for steady and unsteady flow problems.
7. Increment simulation time: The simulation time is incremented by the time step.
8. Finished?: Repeat steps 3 through 6 until simulation has finished. This is either when sufficient samples have been accumulated for a steady state simulations, or for unsteady simulations when the required simulation time has been reached.
9. Save Results: Sampled macroscopic cell data is saved for post processing and analysis. Additionally, the state of all the molecules can be saved to allow the simulation to be resumed from its current condition.

This process is summarised in Figure 2.1.

2.4 Implementation Issues

There have been a number of papers published, [4], [21], [24], [46], that seek to quantify the errors associated with the DSMC methods stochastic implementation of the Boltzmann equation. A summary of the implementation issues that effect the accuracy and validity of a DSMC solution is presented in the following section.

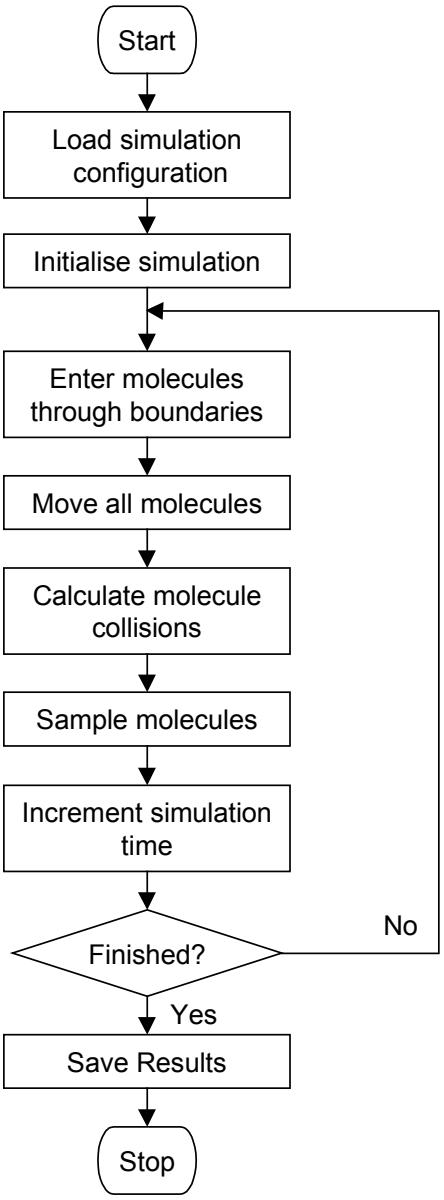


Figure 2.1 Serial DSMC flow chart

2.4.1 Grid Representation

The function of the grid in the DSMC method is to subdivide the flow field for the purpose of molecule collision partner selection and macroscopic property sampling. To guarantee a physically correct collision partner selection, the cell dimensions should be less than the local mean free path, λ. If distance between a pair of colliding molecules is larger than λ, then there would be a physically incorrect transfer of momentum and energy.

It was shown in [4] that the cell size dependence of the transport coefficients truncation error is proportional to the square of the cell size. For a hard sphere collision model, the

maximum transport coefficient error was proved to be 7.5% when the collision cells are one mean free path wide. However, if collision partners are selected via sub-cell sampling, then the condition of relating the cell size relative to the mean free path then applies to the sub-cell size.

Additionally, to ensure that the sampled cell macroscopic properties are physically representative of the underlying gas flow, the cell size should be no larger than λ. For rectangular cells, the longest dimension of a cell is between two opposite corners. Therefore, for two-dimensions the cell side should be kept less than $\lambda/\sqrt{2}$ and for three-dimensions $\lambda/\sqrt{3}$. Furthermore, if the cell is skewed then the longest dimension of a cell will be greater than a rectangle with the same cell side lengths. Therefore, if the cell side length is kept to a maximum of $\lambda/2$ and $\lambda/3$ for two- and three-dimensions respectively, then both the collision partner selection and macroscopic sampling requirements will be satisfied for any cell shape.

A more detailed review of the different griding methods available is presented at the end of this chapter in Section 2.5.

2.4.2 Time Step Selection

As discussed in Section 2.2, the simulation time step must be smaller than the local mean collision time so that the movement and collision phases of the simulation can be decoupled. The local mean collision time, $\bar{t}_{mc}$, is defined as,

$$\bar{t}_{mc} = \frac{1}{\upsilon} = \frac{\lambda}{\bar{c}'} \tag{2.1}$$

where,

υ is the collision frequency, and

λ is the local mean free path, and

$\bar{c}'$ is the average molecule speed.

It has been shown that the error in the transport coefficients are proportional to the square of the time step, [21] and [24]. When the time step is equal to the local mean collision time, the truncation error is approximately 5% [24].

Moreover, to allow the properties of a molecule to be included in the macroscopic property sampling of a cell, it should take longer than one time step for the molecule to traverse the cell. Therefore, the time step should be a small fraction of the local mean collision time and calculated separately for each cell. Generally, the cell time step is chosen to be between a quarter and a sixth of the local mean collision time, [9].

2.4.3 Number of Molecules per Cell

The number of molecules per cell, N_m, needs to be greater than 10, and preferably in the order of 20-30, to ensure that a realistic collision rate is maintained, [7] p334, [17] & [46]. Furthermore, keeping the same number of simulated molecules in each cell is important because it allows the cell sampling to accumulate at the same rate and thus have the same level of random noise [48].

The number of molecules in a cell is given by,

$$N_m = \frac{nV_c}{F_N} \tag{2.2}$$

where,

n is the cell number density

V_c is the cell volume

F_N is the ratio of real to simulation molecules (discussed in Section 2.4.4)

The minimum number of molecules required increases as the number of different molecule species, or species groups, increases. This is required to avoid the distortion of inter-species collision rates, [17]. The main reason the number of molecules per cell is not set at an arbitrarily high value is due to computational resources, system memory and storage space limitations; as well as the solution time increasing in proportion to the total number of molecules.

Therefore, the desired average number of molecules in a cell can be calculated as a function of the minimum species or species group fraction in the simulated freestream gas,

$$\overline{N_m} = \max\left(\frac{2}{(S_{sf})_{\min}}, 20\right) \tag{2.3}$$

where,

$\overline{N_m}$ is the average number of molecules per cell, and

$\left(S_{sf}\right)_{\min}$ is the minimum stream species, or species group[3], fraction.

Unless the product nV_c = constant, then N_m will vary through out the simulation grid. Similarly, if the simulated physical flow causes the local concentrations of molecule species to change, this will affect the local minimum species fraction.

2.4.4 Ratio of Real to Simulation Molecules

The ratio of the number of real molecules represented by a single simulation molecule, F_N, is used to control the total number of simulation molecules in the solution. The discussion from the previous section implies that it is advantageous to have a variable value of F_N to ensure that there is sufficient number of each species of simulation molecule. This approach is taken by a number of DSMC implementations, [19] & [46].

However, the use of variable F_N for different regions of the simulation, or species, can lead to random walks in the solution. This is because the molecular quantities are conserved only on the average, and not exactly, ([7], p213). Therefore, a constant value of F_N is generally used for a simulation and its value is chosen either using Equation 2.3 or limited by the maximum number of simulation molecules that can be accommodated.

2.5 Griding Methods

As discussed in Section 2.4.1, the function of the grid in the DSMC method is to subdivide the flow field for the purpose of molecule collision partner selection and macroscopic property sampling. Based on this purpose for the grid, the DSMC method has no requirement that cells be regular and/or orthogonal. Similarly, discontinuities between grid regions do not degrade the method's results. However, post-processing software used to analyse the simulation results often has difficulty with grid discontinuities, so it is desirable that they be kept to a minimum.

The choice of griding method has a direct affect on the utility and efficiency of the DSMC implementation. Quoting from Bird in [7] p334,

[3] Species groups are used to group gas species that have similar molecular masses. This will improve the efficiency of the collision selection process without measurably decreasing its accuracy.

"The ideal DSMC grid would:

a. have a high computational efficiency;

b. allow efficient definition of complex flow geometries;

c. be 'fitted' to any body and other significant flow boundaries;

d. have variable cell sizes related to the local gradients in the flow properties and to the mean free path;

e. allow this size to adapt to local conditions as the flow develops; and

f. employ sub-cells for effective 'nearest neighbour' collision pair selection."

Point *a* in the above list refers to the efficiency of moving the molecules through the grid. This task involves calculating the molecule's final physical position, its relative position in the grid and if a surface collisions occurred during the molecules movement. This process is discussed in detail in the following chapter.

Point *f* relates to the ability of the grid to subdivide cells for the purpose of collision partner selection. This ability is relatively independent of the grid scheme used, though the difficulty of implementation varies between schemes. This point is covered in detail in Section 4.2.1.1.

The remaining points, *b*-*e*, relate to the definition and structure of the grid scheme used. The different categories of griding methods are summarised in the following sections.

i. **Regular rectangular grid**

This is the simplest of all griding schemes where cells are a constant size and aligned with the orthogonal axes systems. While this grid allows for efficient molecule movement it has significant shortcomings with respect to modelling surfaces and adapting the grid.

Unless all external boundaries and internal objects are rectangular, surfaces will not coincide with cell boundaries which complicates the process of calculating surface interactions. More significantly, it is not possible to adapt the grid to local flow properties. This means for flows that have significant density variations, which is all but the simplest of flows, parts of the grid will be sub-optimal for the local flow properties.

The one area where regular rectangular grids are still used for non-trivial flow problems is in the solution of unsteady, oscillatory flows. For this class of problems, it is impractical to adapt the grid to the calculated flow density as it is continually changing. Therefore, a fine uniform grid is generally used.

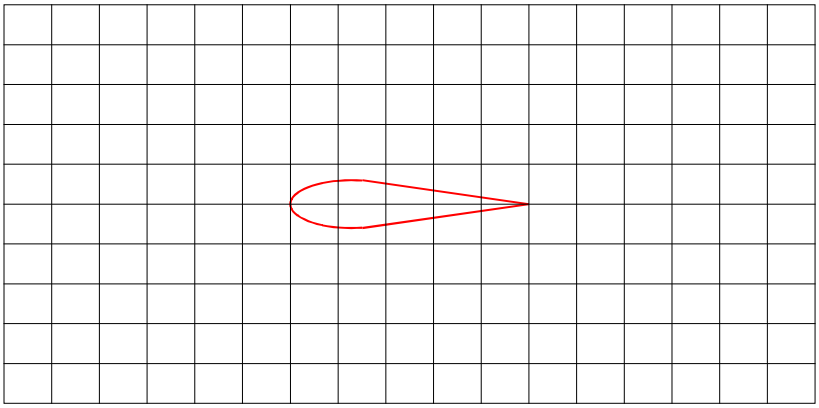


Figure 2.2 Example of a regular rectangular grid

***ii.* Algebraic grids**

Algebraic grids extend the regular rectangular scheme by allowing the grid distribution to be defined by an algebraic expression. It is a requirement that the algebraic expression be uniquely and explicitly invertible. Moreover, this scheme is readily adaptable to work in cylindrical or spherical coordinates.

While this scheme does offer some level of ability to adapt the grid, it is only applicable to flows that have linear or circular objects on the boundary of the grid.

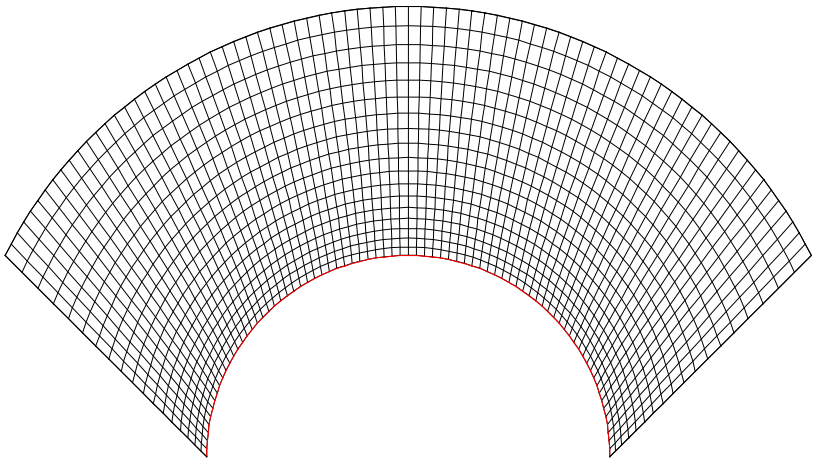


Figure 2.3 Example of an algebraic grid

iii. Variable resolution rectangular grids

Variable resolution rectangular grids attempt to resolve the shortcoming of the regular rectangular scheme with respect to the inability to adapt to the grid by subdividing individual cells. There are two methods of implementing this scheme. In both implementation methods, there is a top-level regular rectangular grid which generally has square or cubic cells.

The first method has two levels of cell resolution. At the second level, each grid cell can be subdivided into arbitrary an number of sub-cells that are equal in size. The sizing of the second level grid cells is determined by the local flow conditions.

The second method has multiple levels of cell resolution. Each cell can be subdivided into four or eight equal sized cells for two- and three-dimensions respectively. If an individual subdivided cell is still too large, it is subdivided. This continues until the required cell size is achieved. This method allows a finer, more adaptable grid than the first method but has a higher overhead in grid maintenance.

An example of a DSMC implementation that uses the first method of a variable resolution rectangular grid is presented in [37].

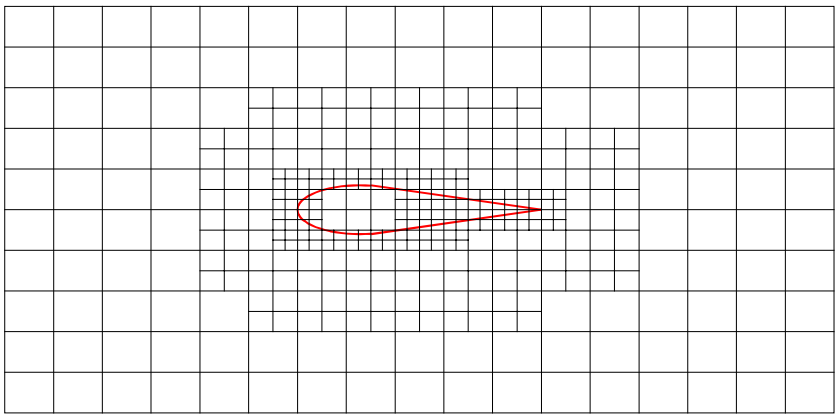


Figure 2.4 Example of a variable resolution rectangular grid

iv. Body-fitted grids

Body-fitted schemes use a non-regular grid that is fitted to the geometry of the external boundaries and internal objects. This method allows arbitrary geometries to e accommodated with no change to the underlying code. In addition, this scheme allows the grid to be adapted by redistributing the grid points and/or changing the number of cells in the grid.

Examples of DSMC implementations that use body-fitted grids are presented in [52], [59].

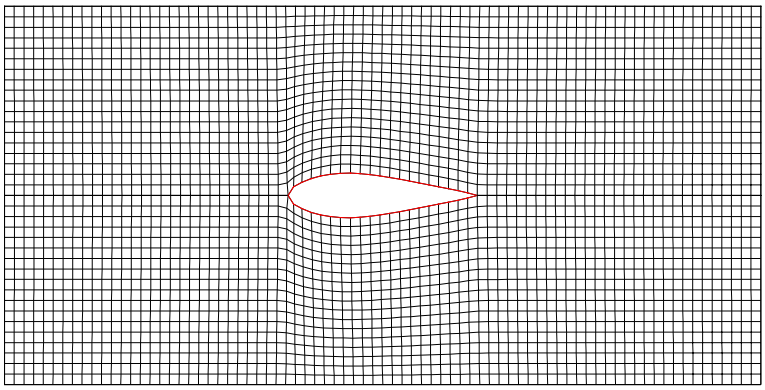


Figure 2.5 Example of a body-fitted grid

***v.* Unstructured grids**

The regular rectangular, algebraic and body-fitted grid schemes are all structured methods, ie. a two-dimensional grid is defined as a block of $n \times m$ cells. These structured methods impose differing degrees of restrictions on the ability to adapt the grid to the local flow conditions. This imposes the constraint that the resolution of the grid in one area has an affect on other areas in the grid. Variable resolution grids attempt to compensate for this by allowing adaption at the cell level.

Unstructured grids remove these constraints by only constraining the individual cells to share sides. Unstructured grids generally use triangular or tetrahedral cells for two- and three-dimensional problems respectively.

Surfaces are always on the edge/face of cells and the scheme is able to handle any physically definable geometry. Grid adaption is performed by subdividing cells to increase resolution, or fusing adjacent cells to decrease resolution.

The primary disadvantage of this scheme is the complexity of managing the grid. Each cell edge/face needs to keep a list of which other cells it is linked too. This becomes more complicated if the adaption scheme is allowed to split edges/faces on one side, ie. if two cells share a common edge and one of the cells is split, then the un-split cell will now have an edge composed of two segments. Grid adaption also causes inefficiencies with the storage of the grid in the host computer's memory. This is because the properties of cells that are physically close may be stored in different sections of memory which increases the time to access these properties.

Examples of DSMC implementations that use unstructured grids are presented in [19], [29], [34].

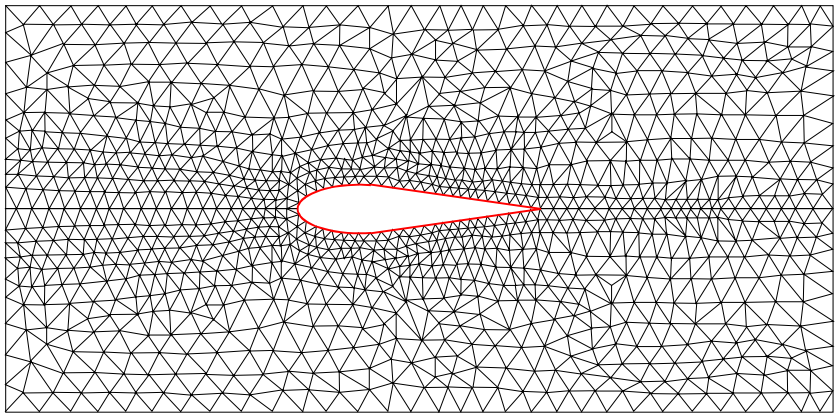


Figure 2.6 Example of an unstructured grid

vi. **Multi-block grids**

Any of the previous grid methods can be incorporated into a multi-block grid method. This method subdivides the simulation region into sub-regions, or blocks, that are grided separately. Structured grid schemes generally benefit most from being implemented in a multi-block scheme. There is a small additional overhead in the complexity of managing the grid, but this is generally outweighed by the improved matching of the grid to the local flow conditions.

An example of a DSMC implementation that uses a multi-block grid is Bird's G2 code, [7] p337.

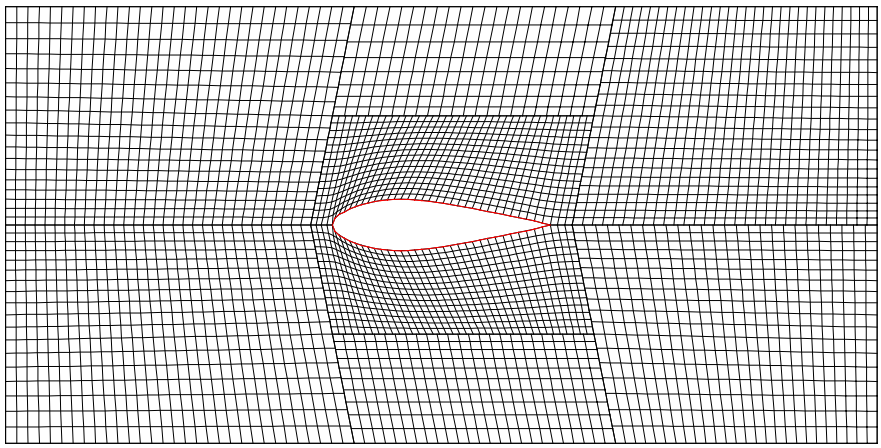


Figure 2.7 Example of a multi-block body-fitted grid

2.5.1 Discussion

The griding schemes summarised above are now analysed with respect to points *b-e* from Bird's list of ideal DSMC grid qualities.

The regular rectangular and algebraic grid schemes do not allow the grid to be adapted to the local flow conditions and therefore, do not meet the requirements of points *d* and *e*.

The variable resolution rectangular grid scheme meets the requirements of adapting to local flow conditions, points *d* and *e*. However, the requirements of accepting complex flow geometries and being fitted to bodies and flow boundaries, points *b* and *c*, is achieved in a inefficient manner. As the top-level grid is fixed to being rectangular, internal bodies and external boundaries that are not rectangular cause regions of the grid not to be used in the solution. Furthermore, additional grid management is required to track where bodies and boundaries cut through cells.

Body-fitted grid schemes are by definition fitted to bodies and flow boundaries, point *c*. However, to efficiently define complex geometries, point *b*, a multi-block scheme is generally required. Furthermore, the scheme is well suited to adapting to the local flow conditions, points *d* and *e*, especially if a multi-block scheme is used. This allows individual block grid resolutions to be adjusted.

Unstructured grid schemes are in effect similar to body-fitted grid schemes, and meet all of the requirements in Bird's list. The main difference between the two schemes is that the unstructured scheme allows finer control of the grid adaption. However, this is at the cost of increased complexity of grid management [38].

Chapter 3

Movement Algorithms

3.1 Introduction

The movement phase in a DSMC program has three primary functions:

1. Update the physical position of the molecule,
2. Calculate any molecule-surface collision, and
3. Determine the cell in which the molecule ended its movement in, and the relative position of the molecule within that cell.

The final position of a molecule is calculated as the linear sum of the velocity components multiplied by the time step. This simple calculation is complicated by the need to determine if the molecule collided with a surface over the course of it movement. For arbitrary surface geometries, the calculation of the molecule-surface impact point can be difficult to calculate as the molecule may just graze the surface or pass through the object, as depicted in Figure 3.1 below. Therefore, the molecule-surface calculation cannot only consider the start and end points of the molecule, but must analyse the entire trajectory.

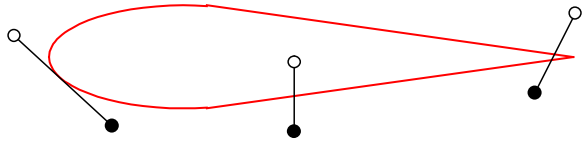


Figure 3.1 Types of molecule-surface interactions

As discussed in Section 2.4.2, the ideal molecule time step should only allow the average molecule to move a half to a third of the cell length, if the cell size is equal to half the local mean free path length. Therefore, on average only a fraction of the molecules will change cells or collided with a surface each time step. Additionally, the significant majority of the molecules that do change cells will only move into an adjacent cell. However, the movement algorithm needs to account for the infrequent

molecules that have a velocity significantly higher than the average. These molecules may move across several cells in a single time step.

If it is determined that a molecule collides with a surface, the effects of the collision needs to be calculated. This requires,

- determining the point where the molecule first intersects the surface,
- the properties of the surface at this point, ie. geometric gradient, surface type, surface temperature, etc.,
- the time at which the collision occurs,
- adjusting the molecules velocity components to account for the collision dynamics, and
- moving the molecule for the remainder of its time step after the molecule-surface collision is calculated.

Determining the cell in which the molecule ends its movement is required so that the molecule can be associated with other nearby molecules. This is required for calculating the local macroscopic properties. The relative position of the molecule in the cell is required so that it can be allocated to a sub-cell for the purpose of identifying possible collision partners. See Section 4.2.1 for discussion of this requirement.

3.2 Alternate Methods

As discussed in Section 2.5 there are a number of different griding methods available to describe a DSMC simulation. The choice of movement algorithm is significantly influenced by the griding method used. There are two main methods used for implementation of the movement phase, physical domain methods and computational domain methods.

Both methods are widely used in DSMC programs, [7], [19], [35], [52] and in the particle tracing community, [18], [26], [33].

3.2.1 Physical Domain Methods

The physical domain methods perform all the calculations of the molecule's trajectory, determination of any surface collisions, and final cell location in the physical domain.

For regular algebraic grids, all of the movement functions can be calculated using exact closed form equations. However, as discussed in Section 2.5.i this griding method substantially limits the complexity of the geometry able to be modelled. Regular grids that allow surface objects to pass through a cell can model arbitrary geometries, but require a more complex surface interaction method. The standard method is to perform a "ray tracing" analysis of the molecule trajectory to determine if and where a surface interaction occurred. Ray tracing methodologies are discussed in more detail below.

Non-regular grids, ie. body-fitted grids, offer a compromise between the above two alternatives by allowing arbitrary geometries to be modelled while ensuring that all surface boundaries are on cell edges. These methods require ray tracing calculations for every molecule movement to determine whether it crossed a cell boundary. Furthermore, there is an added computational burden associated with calculating the relative location of the molecule endpoint in the cell.

3.2.1.1 Ray Tracing

The basic ray tracing algorithm is described below,

1. The molecule is moved through its full time step via the physical linear update equation, the linear sum of the velocity components multiplied by the time step,

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_t \cdot t \tag{3.1}$$

2. Calculate the distance to the intersection point between the ray defined by the start and end points of the molecule, and each of the cell sides/faces.

3. Find the smallest intersection distance that is in the same direction as the molecule movement ray.

4. Compare the intersection distance, from step 3, relative to the length of the ray. If the intersection distance is smaller, the correct molecule end cell has been located and the process is finished. Otherwise, if greater, the molecule passed through the side/face, and is processed accordingly.

   Internal Boundary: Update the molecule end cell index and repeat the procedure from step 2.

   External Boundary: Remove the molecule from the simulation.

Plane of Symmetry: Determine the fraction of the time step used to travel to the intersection point. Update the molecule's velocity components for the symmetry interaction. Recalculate the molecule's final position using Equation 3.1, the molecule's new velocity and the remaining portion of the time step. Repeat the procedure from step 2.

Object Surface: Calculate the molecule impact point and remaining movement time using the same process as the plane of symmetry case. Record the molecule surface interaction. Update the molecule's velocity components, and possibly the remaining movement time, based upon the type of surface. Recalculate the molecule's final position using Equation 3.1, the molecules new velocity and the remaining portion of the time step. Repeat the procedure from step 2.

The computational cost of the ray tracing method can be decreased by pre-calculating and storing the cell edge gradients or surface normals.

3.2.1.2 Geometric Cell Location

Another method that has been proposed as alternative to using ray tracing to check whether a molecule is inside a cell is presented in [18] and [19]. This method is based upon vectorial analysis.

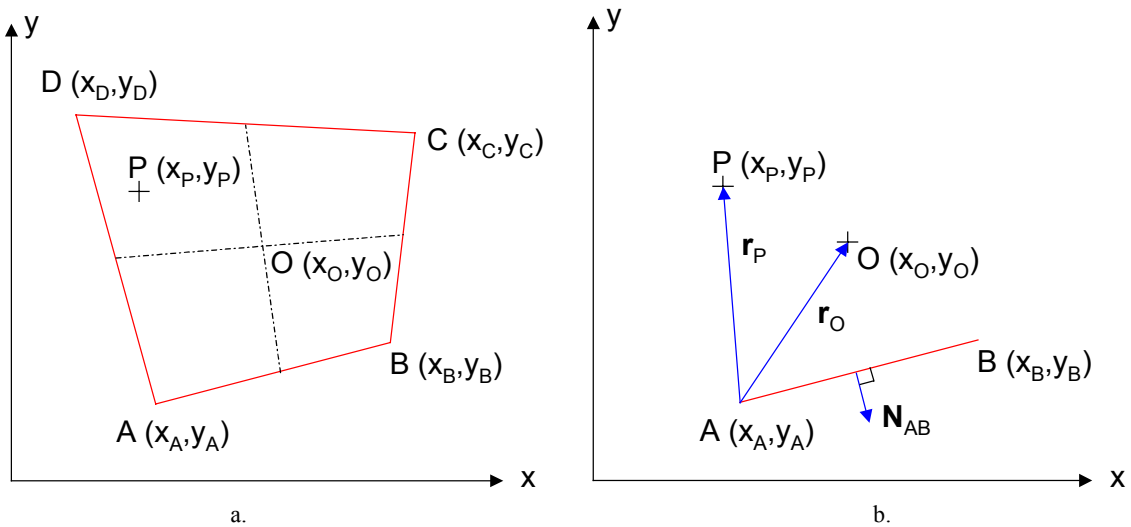


Figure 3.2 Geometric cell location vector definitions

Figure 3.2a shows an arbitrary quadrilateral cell **ABCD**, reference point **O** that is the centre of the cell and the molecule position **P**. If the point **P** is inside the cell, then it

will be on the same side of each cell edge as the point **O**. This property can be tested by the following equation,

$$\Omega_{AB} = [\mathbf{r}_0 \cdot \mathbf{N}_{AB}][\mathbf{r}_P \cdot \mathbf{N}_{AB}] \tag{3.2}$$

where Figure 3.2b shows the definition of the vectors.

If $\Omega \geq 0$, then **P** is on the same side of each cell edge as the point **O**. By calculating Equation 3.2 for all cell edges it can be determined whether the molecule position is inside cell. Furthermore, if the molecule is not inside the cell, the side which has a negative value of Ω indicates the cell to check next.

The computational cost of this method is more than the standard ray tracing method, but can be improved by pre-calculating and storing the sign of the cell edge/surface normals. If it is determined that the molecule crossed a plane of symmetry or object surface during its movement then a ray tracing calculation is still required to determine the edge/surface intersection point.

3.2.1.3 Calculation of Relative Cell Position

The relative position of the molecule in its cell is generally calculated using the iterative Newton-Raphson method,

$$\mathbf{c}_{n+1} = \mathbf{c}_n - \frac{F(\mathbf{c}_n)}{F'(\mathbf{c}_n)} \tag{3.3}$$

where,

- $\mathbf{c}_n$ is the nth estimate of the computational position,
- $F(\mathbf{c}_n)$ is the difference between molecules physical location and the transformation of $\mathbf{c}_n$ into physical space, given by bilinear or trilinear interpolation for two- and three-dimensions respectively,
- $F'(\mathbf{c}_n)$ is the derivative of $F(\mathbf{c}_n)$ which corresponds to the Jacobian matrix of the cell. The 1D division turns into the multiplication of the inverse of the 2D or 3D Jacobian matrix.

An initial guess of the location of the molecule in computational space, $\mathbf{c}_0$, is made. This guess can either be the molecules last computational position if the molecule did not change cells, or the centre of the molecule's new cell. Equation 3.3 is then iterated until the difference between successive results reaches a pre-defined threshold. For the

purpose of relative cell location, the threshold is set as the inverse of twice the maximum number of sub-cells in any direction.

Due to the large number of calculations required each iteration, this calculation is computationally expensive.

3.2.2 Computational Movement Methods

The computational movement method tracks the molecules trajectory in computational space. This allows for simple detection of molecule-surface collisions and the calculation of the relative location of the molecule endpoint in the cell. However, this comes at the expense of the calculation of the molecules trajectory which is no longer linear in computational space. This method is discussed in more detail in the following section.

3.3 Computational Movement Algorithms

The basic premise of a computational movement algorithm is that an arbitrary grid in physical space can be transformed into a regular, unit square grid in computational space. As a body-fitted grid can generally not be described in analytical terms, the transformation from physical to computational space is defined separately for each cell in the grid.

The general transformation for a two-dimensional region in physical space (x,y) to a regular rectangular region in computational space (ξ,η) is given by,

$$\begin{aligned} \xi &= \xi(x, y) \\ \eta &= \eta(x, y) \end{aligned} \tag{3.4}$$

and illustrated in Figure 3.3.

The primary reference for the use of computational movement algorithms in DSMC simulations is [52]. This method proposed by Shimada and Abe was to subdivide the simulation time step into a number of smaller time steps, small enough to resolve the molecule trajectory through a series of linear time integration steps. The time step subdivision factor used was dependent on the molecules velocity and a small constant.

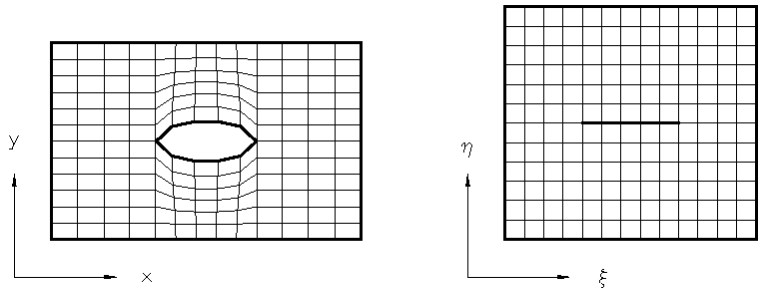


Figure 3.3 General transformation between physical body-fitted coordinates and computational rectangular coordinates

A brief overview of the theory used in the computational movement algorithm, developed by Shimada and Abe, is present below. The full derivation of the method is given in [1] and [52].

3.3.1 Constant Time Step Subdivision Algorithm

Molecular movement is performed by transforming the physical velocity vector, $\overline{q}$, into the computational domain and integrating the equations of motion,

$$\begin{aligned}\dot{\xi} &= \frac{\xi^{n+1}-\xi^{n}}{\Delta t^{n}} = \frac{\overline{S}_I^{\,n}\cdot\overline{q}}{\left|\overline{S}_I^{\,n}\times\overline{S}_J^{\,n}\right|}\\ \dot{\eta} &= \frac{\eta^{n+1}-\eta^{n}}{\Delta t^{n}} = \frac{\overline{S}_J^{\,n}\cdot\overline{q}}{\left|\overline{S}_I^{\,n}\times\overline{S}_J^{\,n}\right|}\end{aligned} \tag{3.5}$$

where, $\overline{S}_I^{\,n}$ and $\overline{S}_J^{\,n}$ are the interpolation of the cells boundary surface area vectors to the point (ξ^n, η^n). The cell boundary surface area vectors are defined in Figure 3.4 and can be evaluated from the cell vertices,

$$\begin{aligned}\overline{S}_{I_{ij}} &= \left(y_{i,j+1}-y_{i,j}\right)\overline{e}_x - \left(x_{i,j+1}-x_{i,j}\right)\overline{e}_y\\ \overline{S}_{J_{ij}} &= -\left(y_{i+1,j}-y_{i,j}\right)\overline{e}_x + \left(x_{i+1,j}-x_{i,j}\right)\overline{e}_y\end{aligned} \tag{3.6}$$

A first order interpolation scheme is used to evaluate these vectors at the point (ξ^n, η^n), assuming $\Delta\xi = \Delta\eta = 1$,

$$\begin{aligned}\overline{S}_I^{\,n} &= \left(i+1-\xi^{n}\right)\overline{S}_{I_{i,j}} + \left(\xi^{n}-i\right)\overline{S}_{I_{i+1,j}}\\ \overline{S}_J^{\,n} &= \left(j+1-\eta^{n}\right)\overline{S}_{J_{i,j}} + \left(\eta^{n}-j\right)\overline{S}_{J_{i+1,j}}\end{aligned} \tag{3.7}$$

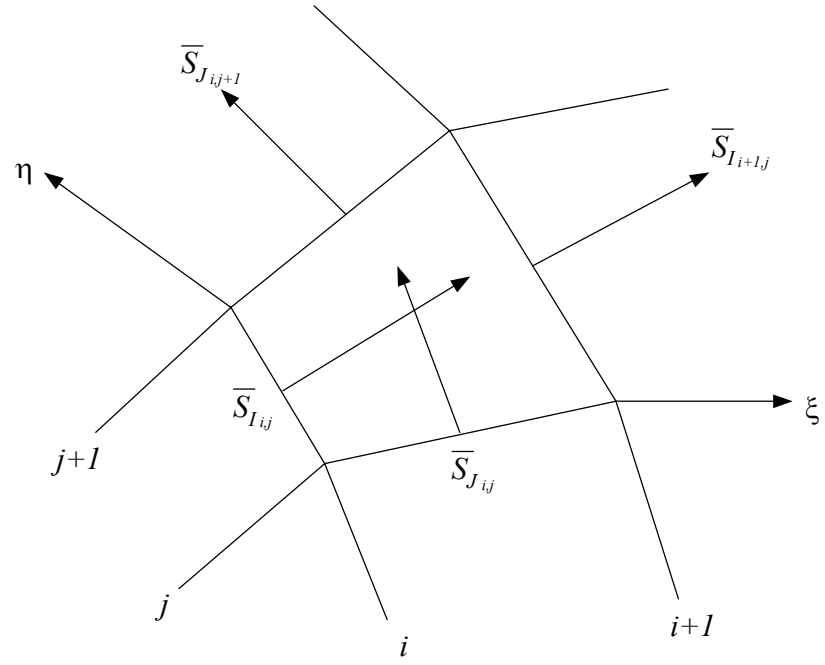


Figure 3.4 Definition of cell boundary surface area vectors

The time step used in Equation 3.5 is the local time step, that is a fraction of the simulation time step. The local time step is calculated as,

$$\begin{aligned} \Delta t^n &= \min\left[\Delta t_r, \Delta t_\xi, \Delta t_\eta\right] \\ \Delta t_\xi &= \delta \frac{\Delta \xi}{\left|\dot{\xi}\right|} \\ \Delta t_\eta &= \delta \frac{\Delta \eta}{\left|\dot{\eta}\right|} \end{aligned} \tag{3.8}$$

where,

δ = small constant

$\Delta\xi$, $\Delta\eta$ = computational grid spacing = 1

$\dot{\xi}$, $\dot{\eta}$ = computational velocity components

Δt_r = remaining time in simulation time step

The velocity integration is repeated until the full simulation time step has been completed, the molecule hits an object or leaves the simulation area. Molecule object/boundary interactions are computationally trivial to determine since both simulation boundaries and object surfaces correspond to lines of constant ξ or η. Similarly, the identification of the cell in which the molecule is contained is determined by finding the largest integers, (*i*,*j*) not larger than (ξ,η).

The small constant value, δ, used in Equation 3.8, was determined empirically to be 0.05 ([52], p264). This value yielded good trajectory tracking through the severest grid irregularities tested by the authors. However, this necessitates that all molecules are moved with a time step subdivision that assumes that they moving through the worst case cell.

3.3.2 Improved Time Step Subdivision Algorithm

The limitation of using a constant time step subdivision value can be reduced by making the value a variable calculated for each cell based on the cell's 'skew' and relative size [33]. This allows the use of the property that if the physical cell geometry is a parallelogram, then the molecule's transformed velocity vector is still a linear function. In addition, a molecule needs to take smaller steps to accurately traverse a small cell, conversely a larger step size can be used to traverse a large cell.

The skew of each cell is estimated as the product of the ratio of geometrically opposite cell side lengths. The ratio of the opposite cell sides is arranged such that the longest side of each pair is placed in the denominator. Using this formulation, a parallelogram will have a skew factor of 1 and this value will decrease as the difference between the cell sides grows.

The local time step is calculated as,

$$\begin{aligned} \Delta t^n &= \min\left[\Delta t_r, \Delta t_\xi, \Delta t_\eta\right] \\ \Delta t_\xi &= \delta'\left(\frac{l_3}{l_1}\frac{l_4}{l_2}\right)\frac{\Delta\xi}{|\dot{\xi}|} \\ \Delta t_\eta &= \delta'\left(\frac{l_3}{l_1}\frac{l_4}{l_2}\right)\frac{\Delta\eta}{|\dot{\eta}|} \end{aligned} \tag{3.9}$$

where,

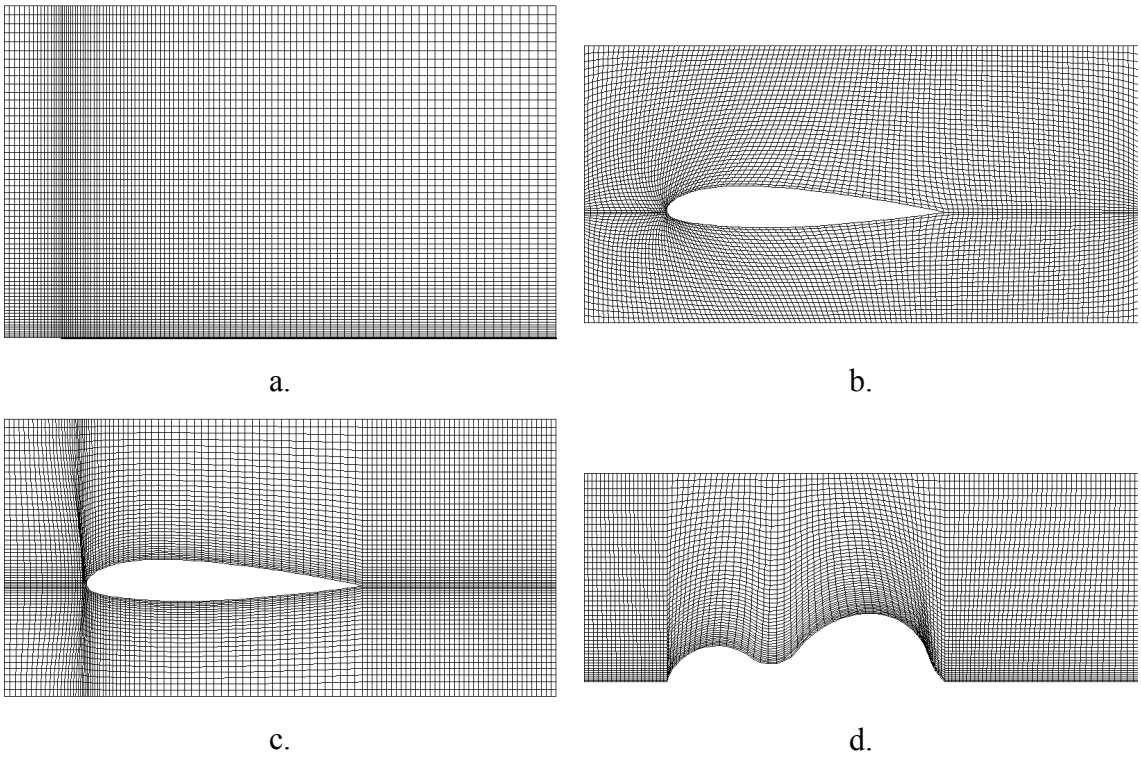
δ' = small constant

l_m = cell side lengths, numbered circularly such that $l_1 > l_3$ and $l_2 > l_4$

To improve the efficiency of the local time step calculation, the cell skew factor is pre-calculate for each cell when the grid is generated.

3.3.3 Results

Simulation test were conducted on a number of relatively simple grids, as shown in Figure 3.5, to evaluate the relative performance of the standard and modified algorithms.



a. b.

c. d.

Figure 3.5 Movement algorithm test grids

An 18% reduction in the total simulation move time was measured for the modified algorithm relative to the standard algorithm. However, neither algorithm was satisfactory for general use because the small constant value used in each algorithm had to be determined empirically for each test case to achieve the best performance. A uniformly very small value could be used, but this resulted in an increased movement time for grids with a small amount of skew.

The tuning of the small constant values was done by starting with a large value and reducing the value until there was no discontinuities in the results caused by changes in grid geometry. Figure 3.6 illustrates the effects of using too large a value for the time step subdivision constant.

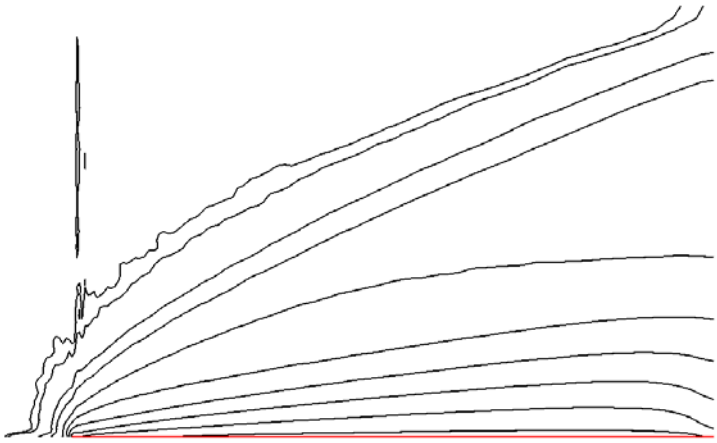
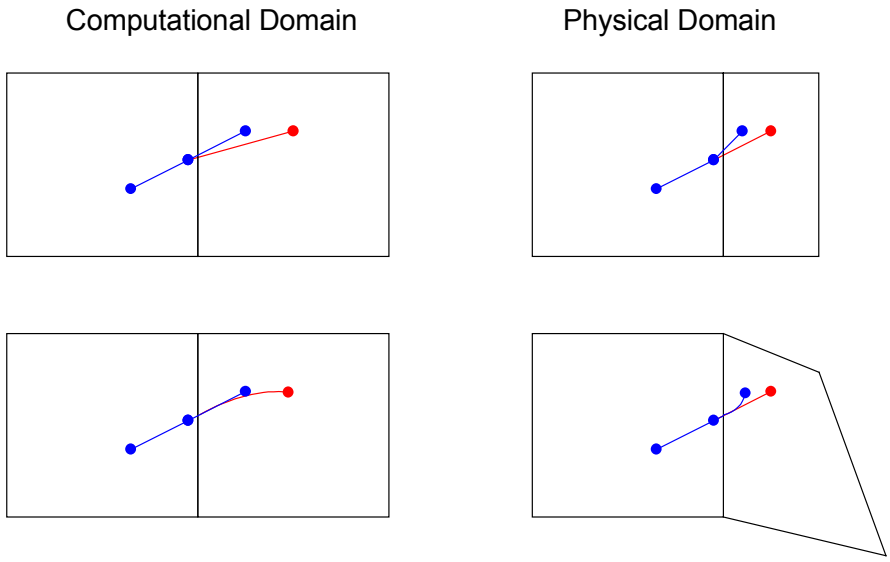


Figure 3.6 Mach No. contours of flow past a horizontal flat plate

Figure 3.6 shows the Mach number contours for flow past a horizontal flat plate using the grid defined in Figure 3.5a. The freestream conditions for this test are $Ma_\infty = 4$ and a Knudsen number of, $Kn = 0.0143$, referenced to the plate length (see Section 7.2.2). The results of this test indicate that there is a weak standing shock at the leading edge of the flat plate, which is incorrect for the test geometry. Figure 3.5a shows that there is a significant reduction in the horizontal grid spacing around the leading edge of the plate that corresponds to the location of weak standing shock. This erroneous flow feature is an artefact of the movement algorithm which implicitly assumes that when a molecule crosses a cell boundary the adjacent cell is an extrapolation of the current cell. Figure 3.7 graphically illustrates the effects of this assumption.



Red lines are the correct physical molecule trajectory and blue lines are the computational estimate.

Figure 3.7 Molecule movement error due to changes in cell aspect ratio and 'skew'

In Figure 3.7, the red lines show the molecules true trajectory and the blue lines the calculated trajectory. In the computational domain, each sub-time step moves the same distance. However, when a molecule crosses a grid edge, and the new cell has a different geometry, the portion of the molecules movement in the grid for that remaining fraction of the sub-time step will have the incorrect trajectory. In the test case shown in Figure 3.6, the majority of the molecules are moving from left to right across the flow filed. When a molecule moves from the column of large cells to the left of the leading edge of the flat plate and first enters the adjacent column of smaller cells, the x velocity component will be effectively reduced. This effect is illustrated in Figure 3.7. The reduction of the molecules x velocity component artificially increases the number density in this column of cells, which in turns changes the cell's macroscopic properties resulting in the weak standing shock.

To correct this error, the algorithm was changed to check whether a molecule entered a new cell during a sub-time step. If a molecule was determined to cross a cell boundary, the time and point at which it crossed into a new cell was calculated. The boundary surface vectors (Equation 3.6) were calculated for the new cell and a new local time step was calculated as well. The molecule was then propagated through the rest of its time step.

This correction removed the artificial weak standing shock from the horizontal flat plate test case. It also resulted in a small decrease in the total simulation movement time for all the test cases. This was due to the small constant values, δ and δ', being increased slightly. However, this reduction was partially offset by the increased computational burden of having to calculate all cell boundary crossing points.

A more significant advantage of the correction is that it allowed the small constant value used in the improved algorithm, δ', to be fixed independently of the grid. A value of $\delta' = 0.2$ was found to give good results for all test grids. However, the original algorithm still required hand tuning for each grid to achieve the best performance.

3.4 Hybrid Algorithm

An alternative movement algorithm was developed to overcome the problem of non-linear molecule movement in computational space while retaining the simple cell indexing afforded by tracking the molecules computational space position. The

algorithm achieves this by performing the molecule movement in physical space and then transforming the molecule's final position into computation space. The computation space position is then used to determine the molecules new cell index and whether any surface/boundary interactions took place over the molecule's trajectory.

3.4.1 Derivation

The physical to computation space transformation function was derived by inverting the computation to physical space transformation. For two-dimensions, bilinear interpolation is used to transform the computation coordinates of an arbitrary quadrilateral cell to physical space,

$$\mathbf{x}_P(\xi,\eta) = [\mathbf{x}_A(1-\xi) + \mathbf{x}_B\xi](1-\eta) + [\mathbf{x}_D(1-\xi) + \mathbf{x}_C\xi]\eta \tag{3.10}$$

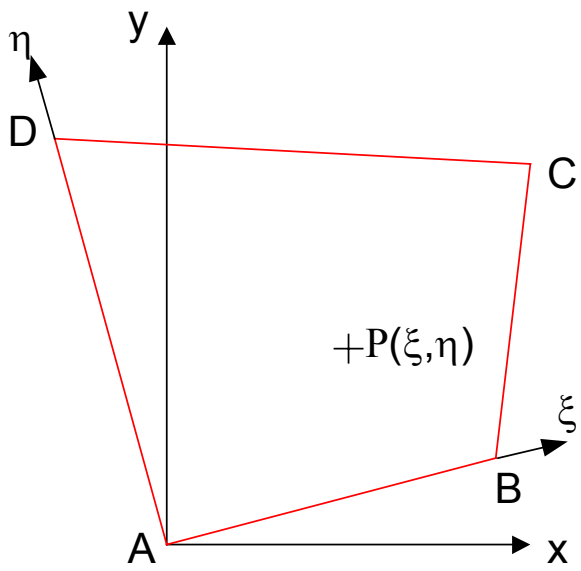


Figure 3.8 Geometry definition for an arbitrary quadrilateral cell

Since the transformation is only dependent on a single cells geometry, a local coordinate system can be used. For the purpose of this derivation, the point A is used as the cells origin. Rearranging Equation 3.10 to group the computational space coordinates and referencing the cell vertices with respect to the cell origin,

$$\begin{aligned} x_P &= x_B\xi + x_D\eta + (x_C - x_B - x_D)\xi\eta \\ y_P &= y_B\xi + y_D\eta + (y_C - y_B - y_D)\xi\eta \end{aligned} \tag{3.11}$$

where,

x_i and y_i are the distances parallel to the x and y axes respectively from the cell origin, point A, to the point *i*.

The solution can be seen to be of the form,

$$\begin{aligned} x_P &= a\xi + b\eta + c\xi\eta \\ y_P &= d\xi + e\eta + f\xi\eta \end{aligned} \tag{3.12}$$

Now, solving Equation 3.12 for ξ and η,

$$\xi = \begin{cases} \dfrac{-2C}{B + \text{sign}(B)\sqrt{B^2 - 4AC}} & ,B > 0 \\ \dfrac{B + \text{sign}(B)\sqrt{B^2 - 4AC}}{-2A} & ,B \le 0, A \neq 0 \\ \dfrac{-C}{B} & ,B < 0, A = 0 \\ \text{undefined} & ,B = 0, A = 0 \end{cases} \tag{3.13}$$

$$\eta = \frac{y_P - d\xi}{e + f\xi}$$

where,

$$\begin{aligned} A &= af - cd \\ B &= ae - bd + cy_P - fx_P \\ C &= by_P - ex_P \end{aligned}$$

Equation 3.13 is defined for all A, B, C; except for when $A = B = 0$. This exception can be countered by solving for η first, which gives the following equation,

$$\eta = \begin{cases} \dfrac{-2C'}{B' + \text{sign}(B')\sqrt{B'^2 - 4A'C'}} & , \quad B' < 0 \\ \dfrac{B' + \text{sign}(B')\sqrt{B'^2 - 4A'C'}}{-2A'} & , \quad B' \ge 0, A' \neq 0 \\ \dfrac{-C'}{B'} & , \quad B' > 0, A' = 0 \\ \textit{undefined} & , \quad B' = 0, A' = 0 \end{cases} \tag{3.14}$$

$$\xi = \frac{y_P - e\eta}{d + f\eta}$$

where,

$$\begin{aligned} A' &= bf - ce \\ B' &= bd - ae + cy_P - fx_P \\ C' &= ay_P - dx_P \end{aligned}$$

Similarly, Equation 3.14 is defined for all A', B', C'; except for when $A' = B' = 0$.

3.4.2 Robustness

3.4.2.1 Internal Points

Both Equations 3.13 and 3.14 will exactly calculate the computational coordinates of P for a valid quadrilateral cell geometry, except for the special case where $A = B = 0$ and $A' = B' = 0$. A valid quadrilateral cell geometry is one where all internal angles are less than or equal to 180° and no edge crosses another. For a valid geometry, the solution from the algorithm is unique and reversible, ie. substituting the calculated result into Equation 3.12 will yield the original point to within calculation accuracy.

This special case where $A = B = 0$ and $A' = B' = 0$ corresponds to a physical cell geometry that is a rectangle aligned with the x and y axes. This physical cell geometry can be easily detected and the solution then becomes a trivial linear interpolation.

The algorithm will yield a correct solution for the two special cases where the cell geometry is equivalent to a triangle,

i. one side has a length of zero, or

ii. two abutting sides are collinear.

3.4.2.2 External Points

If P is located outside of the cell, then the transformation will calculate the computational coordinates based on an extrapolation of the current cell geometry. This solution is still accurate, except when P is in a region where the extrapolated grid is folded. The boundary of this region is defined by the turning point of the gradient of the discriminate from either Equation 3.13 ($B^2 - 4AC$) or Equation 3.14 ($B'^2 - 4A'C'$).

It is not possible to calculate the gradient of the discriminate without first knowing the computational coordinates. However, this case is very rare and generally only occurs when a molecule is moving an order of magnitude faster than the average molecule velocity and the cell is highly skewed. The movement algorithm presented in Section 3.4.4 is robust enough to handle these cases and it generally results in only one extra iteration of the cell location calculation.

The point where opposing cell edges that are not parallel intersect is undefined in computational space. If the point being transformed is found to be at an edge

intersection point, the physical position used in the transformation equation is shifted by a small amount to move it off the intersection point and allow a solution to be found.

It was found that the best accuracy for the calculation of the computational position of a point located outside of the reference cell is achieved if,

i. the local origin used is not one of the end points of the cell's shortest edge, and

ii. if the shortest edge is parallel to the ξ-axis then Equation 3.13 is used, conversely if the shortest edge is parallel to the η-axis Equation 3.14 is used, and

iii. if the cross product of the two edges connected to the local origin is negative and (A,$A' \neq 0$), then the sign of (B,B') is in effect swapped for the purpose of selection which form of the quadratic equation to solve.

3.4.3 Simplifications

The solutions of the transformation algorithm presented in Equations 3.13 and 3.14 can be simplified if one or both of the pairs of opposing cell edges are parallel to the x or y axes. Starting with the definition of the computational to physical transformation presented in Equation 3.11, the calculation of the computational coordinates can be simplified for special cell geometries,

$$\begin{aligned}
\overline{AB} \parallel \overline{CD} \parallel x \quad &\Rightarrow \quad \eta = \frac{y_P}{y_D} \\
\overline{AD} \parallel \overline{BC} \parallel x \quad &\Rightarrow \quad \eta = \frac{y_P}{y_B} \\
\overline{AB} \parallel \overline{CD} \parallel y \quad &\Rightarrow \quad \xi = \frac{x_P}{x_D} \\
\overline{AD} \parallel \overline{BC} \parallel y \quad &\Rightarrow \quad \xi = \frac{x_P}{x_B}
\end{aligned} \tag{3.15}$$

3.4.4 DSMC Movement Implementation

3.4.4.1 Cell Initialisation

After the grid for a simulation has been generated, a number of cell geometry constants are calculated to improve the efficiency of the movement algorithm. Specifically, the constants are,

1. Local cell origin (§3.4.2.2.i)
2. Transformation equation (§3.4.2.2.ii)

3. Sign of the cross product of the two edges connected to the local origin (§3.4.2.2.iii)
4. Whether any of the pairs of opposing cell edges are parallel to the x or y axes (§3.4.3)

This set of pre-calculated constants can be represented by seven flags that are packed into a single byte. Therefore, there is only a very small penalty for the storage of the constants with each cell.

3.4.4.2 Molecule Movement

The method implemented to update the position of a molecule during its movement phase is present below.

1. The molecule is moved through its full time step via the physical linear update equation, Equation 3.1
2. Calculate the molecules position in computational space relative to the starting cell. The pre-calculated cell constants discussed in the previous section are used to determine:
   i. which cell vertice to use as the local origin,
   ii. which computational coordinate to calculate first, Equations 3.13 or 3.14, and
   iii. if a simplified form of the algorithm can be used, Equation 3.15;
3. If the calculated values of (ξ,η) are in the domain $(0 \le \xi,\eta < 1)$, then the molecule stayed in its starting cell and no further processing is required.
4. Calculate index of next cell to check,

```
if (ξ < 0)
    i = i – 1
else if (ξ ≥ 1)
    i = i + 1

if (η < 0)
    j = j – 1
else if (η ≥ 1)
    j = j + 1
```
(3.16)

5. Check whether cell edge crosses corresponds to an internal boundary, external boundary, plane of symmetry or object surface; and process accordingly.

Internal Boundary: Check whether new cell estimate is oscillating between two values. If it is not, repeat the procedure from step 1 with updated cell estimate. Otherwise, cell oscillation only occurs if both the calculated values of (ξ,η) are outside the domain $(0 \le \xi,\eta < 1)$. Therefore, update the cell index that corresponds to the estimated computational position that is furthermost from the valid domain. Repeat the procedure from step 1.

External Boundary: Remove the molecule from the simulation.

Plane of Symmetry: Calculate the intersection point of the molecule trajectory and the cell edge. Determine the fraction of the time step used to travel to the intersection point. Update the molecule's velocity components for the symmetry interaction. Recalculate the molecule's final position using Equation 3.1, the molecules new velocity and the remaining portion of the time step.

Object Surface: Calculate the molecule impact point and remaining movement time using the same process as the plane of symmetry case. Record the molecule surface interaction. Update the molecules velocity components, and possibly the remaining movement time, based upon the type of surface. Recalculate the molecule's final position using Equation 3.1, the molecules new velocity and the remaining portion of the time step.

6. Repeat the procedure from step 1.

As discussed in Section 3.1, for a correctly configured simulation time step, on average only a quarter of the molecules will move to a new cell each time step. Therefore, three-quarters of the molecules will only require a single calculation of the physical to computational space transformation. For the remaining quarter that change cells, the vast majority of these will only need two calculations of the physical to computational space transformation to determine the molecule's final cell. The remaining few molecules that require more than two transformations have a significantly higher than average velocity and/or are in a region of small, skewed cells.

3.4.4.2.1 Numerical Issues

The movement calculations are performed using double precision variables, while the molecules physical and computational space positions are stored in single precision variables. Therefore, caution needs to be exercised to ensure that rounding does not

cause the stored position to “move” the molecule into a new cell if its final position is very close to a cell edge.

The movement algorithm is structured such that all the molecules starting in the same cell are processed together. This allows some of the cell dependent values, ie. the location of the cell corners relative the local origin, to be calculate once for the cell and only updated if the molecule moves to a new cell.

3.4.4.3 General Point Location Algorithm

The implementation presented above is efficient for the case where the molecule is generally located in the same cell as it was the previous time step. However, if it is required to calculate the computational position of a point in the grid where no prior information is available, ie. after a grid adaption, then a large number of inverse transformations may need to be calculated. To reduce this computational burden, a simple search algorithm is used to first identify which cell the molecule is most likely in. Then the cell update algorithm derived in the previous section is used to correctly identify the cell and calculate the computational position.

The search algorithm used “walks” along the grids rows and columns until the closest cell corner point is found, and is as follows,

1. Specify a starting cell for the search. This could be the result of the last search if successive molecule positions are related, or more generally the centre of the grid.
2. Use a bisection followed by a hunt search algorithm to search along the current grid row to find the nearest bottom/left cell corner to the molecule.
3. Repeat step 2 but search along the grid column.
4. Repeat steps 2 and 3 until the estimated cell does not change between successive passes, or the estimated cell gets into a repetitive loop.

This algorithm will correctly estimate the molecules cell the majority of the time, and at most will be one cell incorrect for highly distorted grids.

3.4.5 Results

The performance of the Hybrid Movement Algorithm (HMA) was compared to the improved computational movement algorithm (ICMA) present in Section 3.3. This comparison was performed using the same test cases as defined in Section 3.3.3.

It was found that HMA was always faster than the ICMA. On average, the HMA achieved a 23% reduction in total simulation move time relative to the ICMA. The largest improvement in total simulation move time was achieved for case C which was 50%. This significant improvement for case C was due to the nature of the grid that contained regions of rectangular cells as well as regions of small, highly skewed cells around the aerofoil leading edge. In the regions of rectangular cells, the HMA was able to use a simplified physical to computational transformation algorithm that decreases the movement time. Conversely, in the regions of small, highly skewed cells the ICMA decreased the local cell time step, to ensure accurate tracking of the molecule through the cell, and thus increased the movement time.

Additionally, the HMA will always give the exact molecule movement trajectory independent of the time step used. This is an important result as it allows a large simulation time step to be used without compromising the molecule's trajectory. The utility of increasing the simulation time step above the local mean collision time is discussed in Section 5.5.2.2.

Therefore, for any arbitrary structured two-dimensional grid, the physical movement algorithm is faster and more accurate than the improved computational movement algorithm.

3.4.6 Extension to 3D Geometries

For three-dimensions, the transformation from computation to physical space is defined by the trilinear interpolation function.

For an arbitrary hexahedral cell, as defined in Figure 3.9, the trilinear interpolation function is,

$$\begin{aligned}\mathbf{x}_P(\xi,\eta,\zeta) = &\{[\mathbf{x}_A(1-\xi)+\mathbf{x}_B\xi](1-\eta)+[\mathbf{x}_D(1-\xi)+\mathbf{x}_C\xi]\eta\}(1-\zeta)+\\ &\{[\mathbf{x}_E(1-\xi)+\mathbf{x}_F\xi](1-\eta)+[\mathbf{x}_H(1-\xi)+\mathbf{x}_G\xi]\eta\}\zeta\end{aligned} \tag{3.17}$$

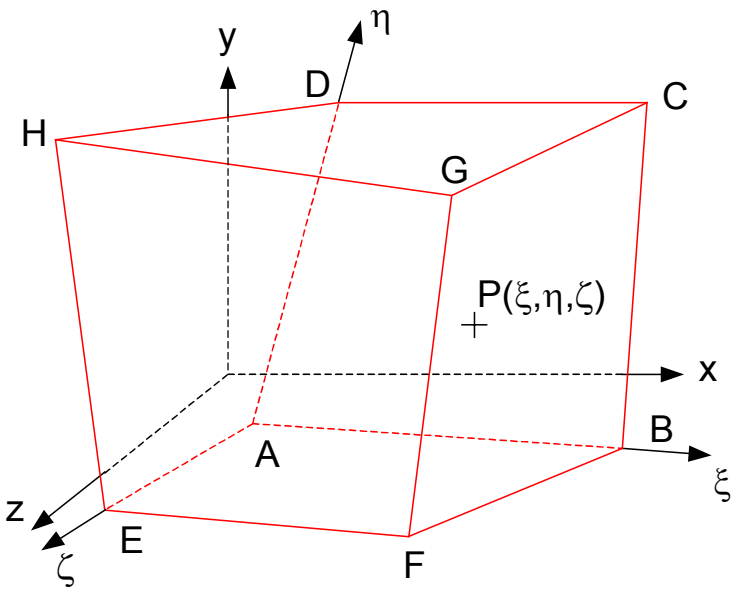


Figure 3.9 Geometry definition for an arbitrary hexahedral cell

A closed form solution to the inversion of this transformation is not possible for an arbitrary hexahedral cell.

Using the Newton-Raphson method, an iterative numerical method, it is possible to estimate the computational space coordinates that provide a solution to the inverse transformation for an arbitrary hexahedral cell. However, these solution methods are quite computationally expensive and therefore relatively slow compared to an exact solution [31].

3.4.6.1 Special Case

A close form solution is possible for the special case where two of the opposing faces of the cell are parallel to one of the coordinate frames orthogonal planes. This form of grid still allows arbitrary and complex geometries to be modelled without the need to customise the program to the problem. The grid is not constrained to ensure that the four hexahedral faces, between the two parallel faces, are planer which simplifies the grid generation.

3.4.6.1.1 Derivation

For the purpose of deriving the solution to the inverse transformation, it is assumed that the faces $\overline{ABCD}$ and $\overline{EFGH}$, as defined in Figure 3.9, are parallel to the XY plane of the coordinate frame. The solution method is the same for any two opposing faces

parallel to one of the coordinate frames orthogonal planes, with the only difference being the order of solution.

Using point A as the origin, the solution for ζ is given by,

$$\zeta = \frac{z_P - z_A}{z_E - z_A} \tag{3.18}$$

Using this result, the plane that passes through the point P and is parallel to the *XY* plane of the coordinate frame can be calculated by linear interpolation,

$$\overline{IJKL} = (1-\zeta)\overline{ABCD} + \zeta\overline{EFGH} \tag{3.19}$$

The coordinates of the plane $\overline{IJKL}$ can then be used to solve for ξ and η by using the 2D algorithm presented in Section 3.4.1.

3.4.6.1.2 Implementation

The only limitation of this grid is that the thickness of cells between adjacent planes in the grid is constant. However, the thickness of cell planes can vary and be adapted to the local flow conditions. Additionally, the majority of all local flow phenomena can be considered 2D. This form of grid is well suited to a multi-block configuration.

The DSMC movement implementation is the same as presented in Section 3.4.4 except for the extra checks required for the additional molecule dimension. The algorithm first checks whether the molecule has moved between grid planes, as it known that two of the cell faces are parallel to the *XY* plane. The algorithm then follows the 2D cell update method detailed in Section 3.4.4.2.

3.4.6.2 General Case

For a grid where the constraint of having two opposing faces parallel to one of the coordinate frames orthogonal planes is not possible, there is an alternative method available. Instead of solving the physical to computational space transformation for the hexahedral, Equation 3.17, the hexahedral cell is subdivided into tetrahedra. For an arbitrary tetrahedra it is possible to calculate the closed form solution of the physical to computational space transformation.

3.4.6.2.1 Derivation

The derivation of the physical to computational space transformation for an arbitrary tetrahedral is derived in [32] and is summarised here.

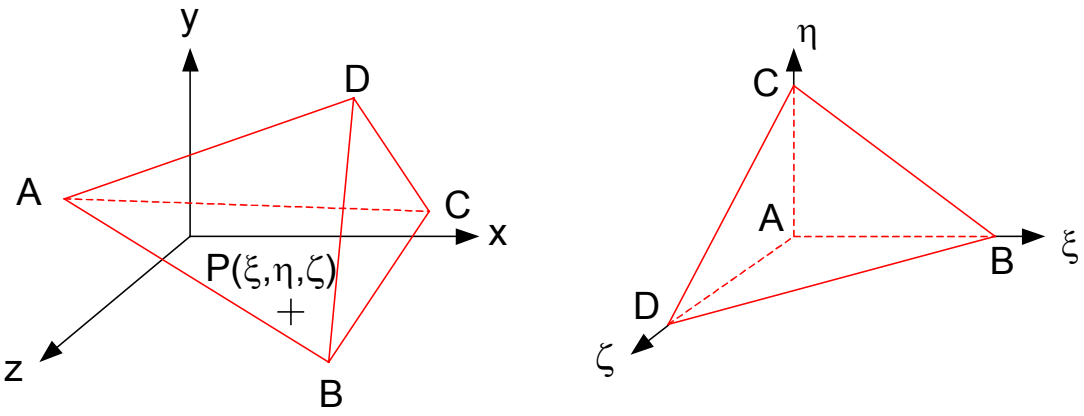


Figure 3.10 Geometry definition for an arbitrary tetrahedral cell

The transformation from computational to physical space is given by

$$\begin{aligned} x_P &= x_B\xi + x_C\eta + x_D\zeta \\ y_P &= y_B\xi + y_C\eta + y_D\zeta \\ z_P &= z_B\xi + z_C\eta + z_D\zeta \end{aligned} \tag{3.20}$$

where,

x_i, y_i and z_i are the distances parallel to the x, y and z axes respectively from the cell origin, corner A, to the point i.

As the transformation equation is a linear sum of the computational space coordinates, it can be directly inverted. Rearranging Equation 3.20 into matrix form gives,

$$\begin{pmatrix} x_P \\ y_P \\ z_P \end{pmatrix} = \begin{pmatrix} x_B & x_C & x_D \\ y_B & y_C & y_D \\ z_B & z_C & z_D \end{pmatrix} \begin{pmatrix} \xi \\ \eta \\ \zeta \end{pmatrix} \tag{3.21}$$

This matrix equation can be solved by pre-multiplying both sides of the equation by the inverse of the 3x3 matrix on the right side. This results in the following equation

$$\begin{pmatrix} \xi \\ \eta \\ \zeta \end{pmatrix} = \frac{1}{V} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_P \\ y_P \\ z_P \end{pmatrix} \tag{3.22}$$

where,

$$
\begin{aligned}
a_{11} &= (y_C - y_D)z_D - y_D(z_C - z_D) \\
a_{21} &= y_D z_B - y_B z_D \\
a_{31} &= (y_B - y_C)z_B - y_B(z_B - z_C) \\
a_{12} &= x_D(z_C - z_D) - (x_C - x_D)z_D \\
a_{22} &= x_B z_D - x_D z_B \\
a_{32} &= x_B(z_B - z_C) - (x_B - x_C)z_B \\
a_{13} &= (x_C - x_D)y_D - x_D(y_C - y_D) \\
a_{23} &= x_D y_B - x_B y_D \\
a_{33} &= (x_B - x_C)y_B - x_B(y_B - y_C) \\
V &= x_B(y_C z_D - y_D z_C) + x_C(y_D z_B - y_B z_D) + x_D(y_B z_C - y_B z_B)
\end{aligned}
$$

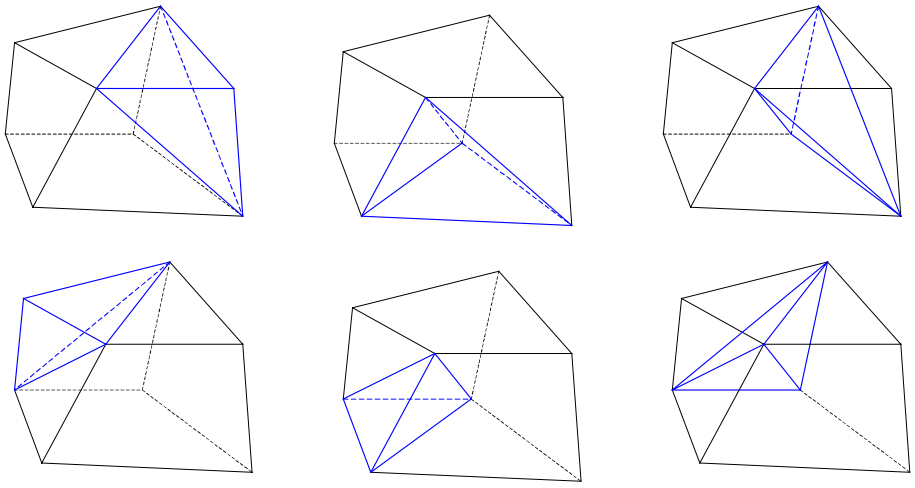
The constant V is the determinant of the 3x3 matrix from Equation 3.21 and is equivalent to 6 times the volume of the tetrahedral.

3.4.6.2.2 Cell Subdivision

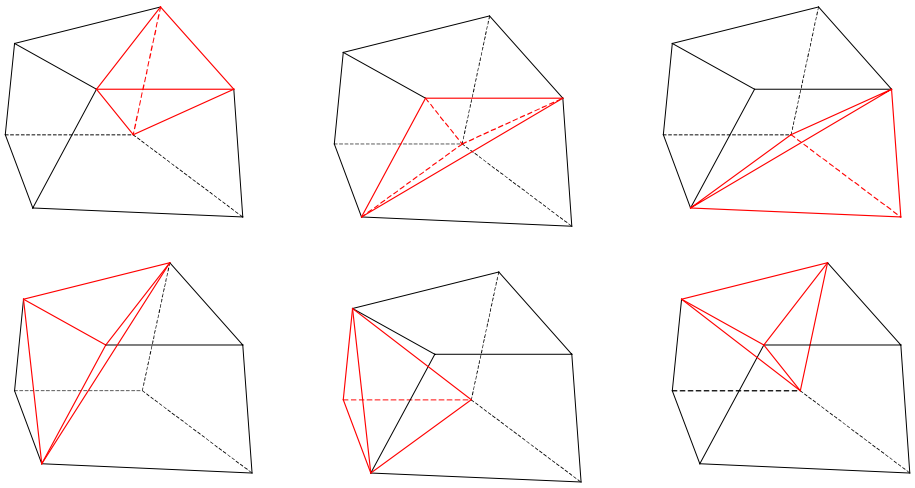
Firstly, each hexahedral cell needs to be subdivided into tetrahedra. Using only the cells corner points, it is possible to subdivide a hexahedral cell into either five or six tetrahedron. The six-tetrahedron scheme is preferred as the five-tetrahedron scheme can break down for very thin tetrahedra ([7], p. 390). The subdivision of a hexahedral cell into six tetrahedral is shown in Figure 3.11 below.

It is possible to subdivide a hexahedral cell into six tetrahedra in 12 different ways. As the faces of the hexahedral cell are usually non-planar, it is a requirement that adjacent cell faces have matching diagonals to prevent gaps. This is achieved by alternating between the two subdivision solutions that meet this requirement. Figure 3.11 shows pair of matching subdivision solutions.

The correct subdivision solution for each cell is determined by the summation of the cell's three global grid indices. When the sum is odd, solution A is used and conversely when the sum is even, solution B is used. As only the cell corner points are used, no extra information storage is required to define the hexahedral subdivision.



a. Solution A



b. Solution B

Figure 3.11 Subdivision of a hexahedra cell into six tetrahedral

3.4.6.2.3 Cell Update

Initially the computational coordinates of the molecule is calculated by using Equation 3.22, assuming the molecule did not change cells during the movement time step. This assumption is verified by using the calculated computational coordinates and he following four conditions,

$$\begin{aligned} &\xi \geq 0 \\ &\eta \geq 0 \\ &\zeta \geq 0 \\ &1-\xi-\eta-\zeta > 0 \end{aligned} \tag{3.23}$$

If any of these conditions are violated, then the molecule moved out of it's starting cell during its movement. If only one condition is violated, the molecule crossed the face associated with that condition and the adjacent tetrahedron is checked. Where the conditions $\xi,\eta,\zeta \geq 0$ correspond to the faces of the tetrahedron that are on the defining hexahedral cell, and the condition $1\text{-}\xi\text{-}\eta\text{-}\zeta > 0$ corresponds to the internal diagonal face. If more than one condition is violated, then the condition that is violated by the largest margin is used to determine which tetrahedron to check next.

The remainder of the cell update method follows the 2D method outlined in Section 3.4.4.2.

3.4.6.3 Discussion

One disadvantage of the general 3D method is that an average molecule will change tetrahedra almost every time step. Additionally, as the "thickness" of tetrahedra is small near the verticies, then a molecule that passes near a vertice is likely to cross a number of tetrahedra in a single time step. This means that generally at least two physical to computational space transformations are required per molecule each movement time step. In contrast the Special Case method, Section 3.4.6.1, should only have a molecule change cells every fourth time step.

Chapter 4

Program Structure

4.1 Introduction

Based upon the analysis and results presented in the previous two chapters it was decided that the grid structure used for this thesis would be a multi-block body-fitted grid and the hybrid movement algorithm derived in Section 3.4 would be used. The version of the program developed for this thesis was for two-dimensional problems only. However, the program has been structured such that it can be readily extended to three-dimensions.

4.2 Physical Models

The physical models implemented in this thesis primarily follows the theory presented by Bird in his monograph [7]. Details of the theory implementation are only presented when it is in variance with this reference. The reader is directed to this reference for full details of the underlying theory.

4.2.1 Collision Model

By default the Variable Hard Sphere (VHS) model is employed for the purpose of calculating molecule collision dynamics. In addition, the user may chose to use the Variable Soft Sphere (VSS) model. All gas properties are defined independently for each species and cross-collision parameters can be specified if the VSS model is used. For the purpose of collision partner selection, each gas species can be treated individually or associated with other species in a species group.

The No Time Counter (NTC) method is used to calculate the number of possible collisions per cell and the probability of these collisions being processed. The maximum value of the product of the collision cross-section, σ_T, and the relative

velocity, c_r, is stored separately for each species group collision pair and is updated after each collision if required.

4.2.1.1 Variable Sub-cells

To minimise the separation between molecule collision pairs, a variable number of sub-cells are used in each grid cell. The number of sub-cells is chosen such that the ratio between the number of sub-cells and the number of molecules is between 0.5 – 1.0. The number of sub-cells along the ξ and η directions is different by at most one sub-cell, with the larger number of sub-cells being assigned based upon the cell aspect ratio. All sub-cells are of equal size in the computational domain, and molecules are assigned to sub-cells based upon their relative position in the cell.

After a collision between two different species groups has been selected to be evaluated, the molecule from the first species group is randomly selected from the list of appropriate molecules in the cell. If a molecule of the second species group is not in the same sub-cell as the first molecule, a spiral search through the other sub-cells is conducted until the first molecule of the second species group is found. The starting step direction and spiral direction of rotation is chosen randomly and the process is illustrated in the Figure 4.1.

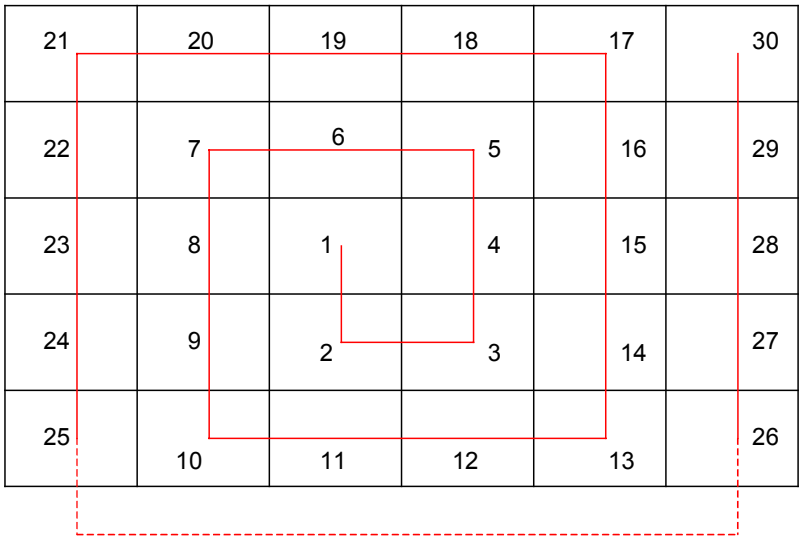


Figure 4.1 Variable sub-cell search method

4.2.1.2 Rotational Relaxation Model

The program allows the user to enable the simulation of internal rotational energy for diatomic and polyatomic molecules. The Larsen-Borgnakke phenomenological model

[13] is implemented to model the rotational relaxation in inelastic collisions. A constant value or a second order polynomial with respect to local total temperature can be used to specify the rotational relaxation collision number, Z_r.

4.2.2 Molecule-Surface Interactions

For the simulation of molecule-surface interactions, the program includes the Maxwellian models and the Cercignani-Lampis-Lord (CLL) model. The Maxwellian model implementation allows the user to select either a specular surface or a diffuse surface with complete temperature accommodation. The CLL model implementation allows the normal energy, rotational energy and tangential momentum accommodation coefficients to be specified independently.

4.2.3 Boundary Models

The user is able to assign different models to internal and external boundaries. The available boundary models are,

- *Vacuum*: No molecule inflow. Molecules exiting through the boundary are removed from the simulation. Molecule-boundary collision properties are not sampled.
- *Freestream*: Molecule inflow is freestream gas properties. Molecules exiting through the boundary are removed from the simulation. Molecule-boundary collision properties are not sampled.
- *Symmetry*: Perfectly elastic collision surface. Molecule-boundary collision properties are not sampled.
- *Object*: Uses one of the defined gas-surface interaction models from Section 4.2.2. Molecule-boundary collision properties are sampled for post-processing. Each surface is independently assigned a temperature and three velocity components. The surface velocity components are vectorially added to the molecules post-collision velocity.
- *Jet*: Molecule inflow that has independently defined gas properties. Molecules exiting through the boundary are removed from the simulation. Molecule-boundary collision properties are not sampled.

- *Internal*: Molecule crossing is uninhibited and molecule-boundary collision properties are not sampled. This model cannot be used for an external boundary.

The user is able to associate any of the boundary models with either an internal or external boundary, except the internal model. However, it is the responsibility of the user to ensure that selected configuration is physically realistic.

4.2.3.1 Molecule Injection

Freestream or jet boundaries have molecule inflow. The number of molecules entering through a boundary segment per unit length and time, the inward number flux, $\dot{N}_i$, is given by,

$$\dot{N}_{i,p} = \frac{n_p \left\{ e^{-(s\cos\theta)^2} + \sqrt{\pi} s \cos\theta \left[1 + \mathrm{erf}\left(s\cos\theta \right) \right] \right\}}{2\sqrt{\pi}\beta} \tag{4.1}$$

where,

$\dot{N}_{i,p}$ is the inward number flux of gas species p,

n_p is the inflow stream number density of gas species p,

s is the molecular speed ratio, = $c_o\beta$,

θ is the angle between the inflow stream and the edge surface normal,

β is the inverse of the most probable molecular thermal speed

c_o is the inflow stream velocity, and

erf is the error function, $\mathrm{erf}(a) = \frac{2}{\sqrt{\pi}} \int_0^a e^{-x^2} \, \mathrm{d}x$

The total number of molecules of gas species p that enter a cell side per time step is given by,

$$N_{i,p} = \dot{N}_{i,p} \cdot l \cdot \Delta t + R_{i,p} \tag{4.2}$$

where,

$N_{i,p}$ is the number of molecules of gas species p entering the cell,

l is the cell side length on the inflow boundary,

Δt is the cell time step, and

$R_{i,p}$ is the remainder from the previous inflow calculation.

As the result of Equation 4.2 is normally not an integer value and only an integer number molecules can enter a cell side per time step, the actual number of molecules that enter is the truncated result of Equation 4.2, $\lfloor N_{i,p} \rfloor$. The remainder from the truncation is given by,

$$R_{i,p} = N_{i,p} - \lfloor N_{i,p} \rfloor \tag{4.3}$$

4.2.4 Flowfield Initialisation

When a new simulation problem is started, the user has the option of starting the simulation as a vacuum state or populated with molecules that are at the freestream state. If the simulation is initiated to the freestream state, the number of molecules starting in each cell is given by,

$$N_{m,p} = \frac{n_p V_c}{F_N} \tag{4.4}$$

where,

$N_{m,p}$ is the number of molecules of gas species p in the cell,

n_p is the inflow stream number density of gas species p,

V_c is the cell volume, and

F_N is the ratio of real to simulation molecules.

4.3 Grid Generation

A multi-block, body-fitted structured grid is used to define the physical domain used in the simulation problems. Each grid block is composed of arbitrary quadrilateral cells and the cell spacing can be discontinuous at inter-block boundaries. This grid structure was chosen based upon the results of the analysis presented in Chapter 3.

4.3.1 Geometry Definition

The geometric definition of the external boundaries and internal objects are constructed from simple geometric segments. One or more segments are joined to make an edge, four edges are coupled to make a grid block and one or more grid blocks are linked to define the simulation domain. The requirements and constraints of each step of this process is detailed in the following sections.

4.3.1.1 Geometric Segments

The available geometric segments are lines, arcs, elliptic segments and cubic splines. The information required to define each of the geometric segments is,

Line Segment: Start and end points.

Arc Segment: Start, end and centre points.

Elliptic Segment: Start, end and centre points; major and minor axis lengths; and the angular orientation of the major axis with respect to the *x*-axis.

Cubic-spline Segment: An array of control points and the start and end gradient of the spline.

It is a requirement that all geometric segments are monotonic in either the *x*- or *y*-axis. This requirement is not a severe limitation as a surface that is not monotonic can be defined by two contiguous segments that are monotonic.

When a geometric segment is created, it is associated with one of the boundary models defined in Section 4.2.3. In addition, extra properties of the boundary model, i.e. temperature and velocity, are defined at the same time.

4.3.1.2 Grid Edges

A grid edge is a set of geometric segments that form a contiguous contour. When a grid edge is created, its primary orientation, either the *x*- or *y*-axis, is recorded. This information is used for the purpose of determining whether the edge forms a ξ and η grid boundary. Furthermore, it is a requirement that all the geometric segments in a grid edge are monotonic in the edge's primary orientation.

4.3.1.3 Grid Blocks

By definition, a grid block cannot have any edges inside its boundary. The boundary of a grid block is defined by four grid edges, or portions of an edge, that when linked form a contiguous boundary. If a portion of a grid edge is used as a boundary, the intersection point of the next boundary must occur at the end point of a geometric segment.

It is a requirement that a grid block boundary must be composed of two edges with an *x*-axis primary orientation and two with an *y*-axis primary orientation. In addition, the boundary edges must join to edges with a different primary orientation.

4.3.2 Grid Generation

After the simulation domain geometry has been defined, an initial grid is generated. The internal grid structure of each grid block is generated independently using transfinite interpolation (TFI) and a user defined edge point distribution. If the grid's internal structure is going to be adapted, then a uniform edge point distribution is sufficient for the initial grid. However, if the grid is not going to be adapted, ie. an unsteady flow problem, it may be preferable to use a non-uniform distribution. The user is able to chose between uniform, cosine, parabolic, linear and double-linear point distributions.

By default, the initial grid size is chosen so that the average cell size is equal to half the freestream mean free path length. As with the edge point distribution, this default value is adequate if the grid structure is going to be adapted. However, the user has the option of applying a scale factor to the grid size.

4.3.2.1 Transfinite Interpolation

The transfinite interpolation scheme developed by Gordon [23] is a second order method that maps an arbitrary grid boundary point distribution on four edges into an interior grid. The interpolation scheme is given by the following equation,

$$\begin{aligned}\mathbf{x}(\xi,\eta) = {} & (1-\bar{\xi})\mathbf{x}(1,\eta) + (\bar{\xi})\mathbf{x}(\xi_{\max},\eta) + (1-\bar{\eta})\mathbf{x}(\xi,1) + (\bar{\eta})\mathbf{x}(\xi,\eta_{\max}) \\ & - (1-\bar{\xi})(1-\bar{\eta})\mathbf{x}(1,1) - (1-\bar{\xi})(\bar{\eta})\mathbf{x}(1,\eta_{\max}) - (\bar{\xi})(1-\bar{\eta})\mathbf{x}(\xi_{\max},1) \\ & - (\bar{\xi})(\bar{\eta})\mathbf{x}(\xi_{\max},\eta_{\max})\end{aligned} \tag{4.5}$$

where,

$\xi_{\max}$, $\eta_{\max}$ are the maximum computational domain indices in the ξ and η directions respectively, and

$$\begin{aligned}\bar{\xi} &= \frac{\xi - 1}{\xi_{\max} - 1} \\ \bar{\eta} &= \frac{\eta - 1}{\eta_{\max} - 1}\end{aligned} \tag{4.6}$$

For the majority of grid block boundary contours, this scheme generates an effective internal grid. However, if one or more boundary contours are discontinuous, then the TFI generated grid can have regions of folding, ie. grid lines crossing each other or the boundary contours. Boundaries that contain convex corners which protrude inside the grid block area are particularly susceptible to folding when used with a TFI scheme. Figure 4.2 illustrates a TFI generated grid the exhibits folding, and the folded region is shown in detail in Figure 4.3.

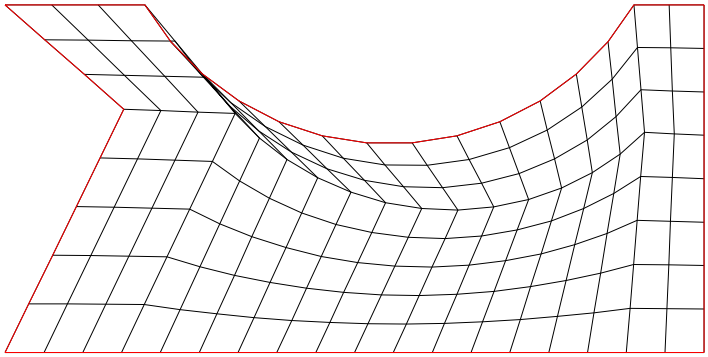


Figure 4.2 TFI grid with folding

Grid folding can be automatically detected using the following algorithm,

$$\text{if } \left[\left(\overline{AB}\times\overline{AD}<0\right)\text{or}\left(\overline{DA}\times\overline{DC}<0\right)\right] \;\Rightarrow\; \text{cell folded} \tag{4.7}$$

where,

A, B, C, D are the vertices of an arbitrary quadrilateral cell defined in a circular order.

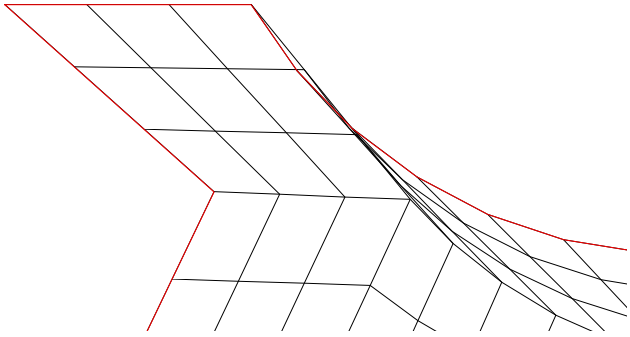


Figure 4.3 Detail of TFI grid with folding

While detection of grid folding is relatively trivial, it is not possible to use this information in the TFI algorithm to correct the error. Additionally, the elliptic grid generation technique of Thompson [55] was tested but was not sufficiently robust to handle all the possible boundary shapes. The most reliable solution to the grid folding

problem was found to be that of correcting the TFI generated grid with a smoothing technique.

4.3.2.2 Grid Smoothing

Jeng [27] present a grid smoothing method that is equivalent to adding a second-order damping to the grid structure. This method reduces the discontinuities in grid lines while preserving the general grid topology. The smoothed grid structure is calculated by,

$$\begin{aligned} x_{i,j} &= (1-\omega_1-\omega_2)x_{i,j} + \omega_1\left(\frac{\Delta s_i^+ x_{i-1,j}}{\Delta s_i^+ + \Delta s_i^-} + \frac{\Delta s_i^- x_{j+1,j}}{\Delta s_i^+ + \Delta s_i^-}\right) \\ &\quad +\omega_2\left(\frac{\Delta s_j^+ x_{i,j-1}}{\Delta s_k^+ + \Delta s_k^-} + \frac{\Delta s_j^- x_{i,j1}}{\Delta s_k^+ + \Delta s_k^-}\right) \\ y_{i,j} &= (1-\omega_1-\omega_2)y_{i,j} + \omega_1\left(\frac{\Delta s_i^+ y_{i-1,j}}{\Delta s_i^+ + \Delta s_i^-} + \frac{\Delta s_i^- y_{j+1,j}}{\Delta s_i^+ + \Delta s_i^-}\right) \\ &\quad +\omega_2\left(\frac{\Delta s_j^+ y_{i,j-1}}{\Delta s_j^+ + \Delta s_j^-} + \frac{\Delta s_j^- y_{i,j1}}{\Delta s_j^+ + \Delta s_j^-}\right) \end{aligned} \tag{4.8}$$

where,

$$\Delta s_i^{\pm} = \sqrt{\left(x_{i\pm1,\,j} - x_{i,\,j}\right)^2 + \left(y_{i\pm1,\,j} - y_{i,\,j}\right)^2}$$

$$\Delta s_j^{\pm} = \sqrt{\left(x_{i,\,j\pm1} - x_{i,\,j}\right)^2 + \left(y_{i,\,j\pm1} - y_{i,\,j}\right)^2}$$

ω_1, ω_2 are the smoothing parameters, $\omega_1 = \omega_2 = 0.45$

Figure 4.4b - f illustrates the effects of successively applying the grid smoothing algorithm to the grid shown in Figure 4.4a.

A single application of the grid smoothing algorithm adjusted the grid sufficiently to remove the folding. However, the grid was still highly skewed around the previously folded region. Subsequent applications of the smoothing algorithm reduced the overall grid skewness, except in the area adjacent to the convex corner. After five successive applications of the smoothing algorithm, the grid had moved far enough to be folded around the convex corner. It is a characteristic of the smoothing algorithm to pull grid lines towards convex corners and away from concave corners.

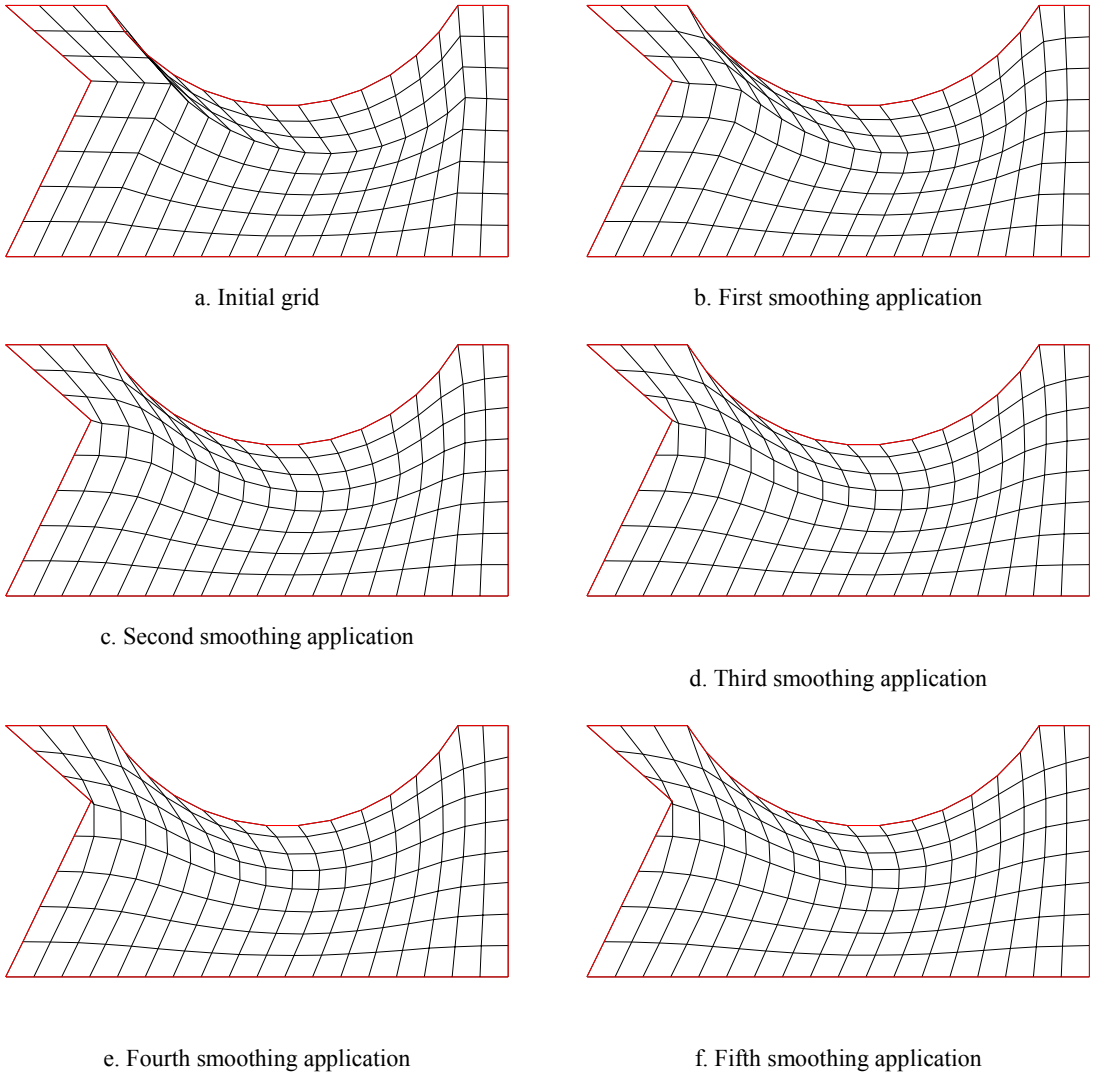


Figure 4.4 Grid smoothing results of folded TFI grid

As it was intended that in general the initial grid should be adapted to the flowfield of the problem being simulated, it was decided that the grid smoothing algorithm should be run the minimum number of times to achieve a correct non-folded grid. While this could result in some areas of the grid being highly skewed, this was deemed to be acceptable as it did not effect the accuracy of the simulation. It was not possible to implement a robust algorithm that could determine at which point the grid smoothing algorithm results become detrimental. Furthermore, if after five successive applications of the smoothing algorithm the grid was still folded, the smoothing was halted and it was assumed that the grid could not be corrected. During testing, the majority of folded grids only required a single application of the smoothing algorithm and all grids were able to be corrected in less that five applications.

4.4 Implementation

The DSMC program was implemented to run as a command line program on a computer using a Unix (Posix) compatible operating system. All Operating System (OS) specific functions were kept to a minimum and partitioned from the main code to allow for straightforward porting to a different OS.

4.4.1 Simulation Configuration Parameters

All simulation configuration parameters were passed to the program via a text file. The primary method of generating the configuration file was through the program prompting the user for each value. The program was written such that the user could not input physically inconsistent data. The format of the configuration file was,

<*data label*>: <*numeric value*> # (*optional comment string*)

Subsequent runs of the same simulation could be made using the same configuration file or by manually making changes to the data values in the file. The ordering of the data labels was not fixed and any manually edited data were checked to ensure that it was physically consistent. Appendix A provides the listing of a sample configuration file.

4.4.2 Geometry Definition

The definition of the simulation geometry and the initial grid structure was generated by the user modifying a standard geometry definition template file. This file was then compiled into a simple program that when run assembled the grid block boundary contours, checked their consistency and generated the internal grid structure. Appendix A provides the listing of an example geometry definition file.

4.4.3 Optimisations

All modern CPU designs now incorporate between one to three levels of fast data cache placed between the CPU and main system memory. When data is accessed from memory, the cache is checked to see whether it is already resident there, and if it is, it can be read from cache with a very small delay. If the data is not resident in the cache, then it must be read from the main memory, which is a slower operation. However, at the same time the requested data is read from main memory, a small block of memory surrounding the requested data is also read into cache.

Therefore, the performance of the program can be increased if,

1. once data is read from memory, as many operations as possible are performed on the data before other data is used so that the cache copy is used, and
2. all data that will be processed simultaneously should be stored close together in memory to maximise the possibility that it is read in the same main memory operation.

The DSMC implementation used in this thesis was designed using these performance objectives.

To improve the performance of the code with respect to the first performance strategy, it is beneficial to perform all the cell processing steps sequentially. However, all molecules must complete their movement before the collision processing can be started. Therefore, it is not possible to perform all of the cell processing steps sequentially. Instead, each time step is broken into two loops. One loop performs the molecule injection and movement, while the other performs the collision and sampling.

The traditional order of performing the processing steps is injection, movement, collisions and sampling, as detailed in Section 2.3. However, for distributed parallel implementations, see Section 6.2, it required that all processing nodes be synchronised after the completion of the movement processing and at the end of each simulation time step. To improve the efficiency of the parallel implementation it is beneficial if these two synchronisation points are combined. Therefore, the DSMC processing steps are reordered to collisions, sampling, injection and movement, as shown in Figure 4.5. This modified processing order can be used for serial DSMC implementations without any noticeable performance penalty.

To improve the performance of the code with respect to the second performance strategy, all molecule and cell variables are aggregated in data structures so that they are stored contiguously in memory. Furthermore, linked lists are used to associate molecules with their host cells. This negates the need to access, and keep up to date, a molecule-cell index array.

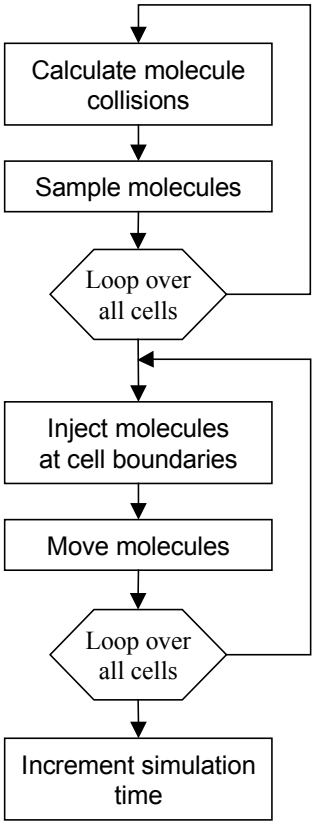


Figure 4.5 Parallelm implementation optimised time step processing

4.4.4 Programming and Storage Issues

As the number of molecules in a complex simulation is generally limited by the available system memory, anything that can be done to reduce the memory requirements of molecule data storage is beneficial. Generally, it is sufficient to store the various state properties of each molecule as single precision floating points (32bit) instead of double precision floating points (64bit). However, for accuracy of the simulation result, all calculations on molecule properties should be performed using double precision floating points and converted back to single precision floating points at the end of the calculation.

The conversion from double to single precision floating points can introduce errors in the simulation. The primary mechanism for these errors is in the storage of the molecules physical and/or computational position. A molecule that is very close to the edge of a cell may be artificially moved to the adjacent cell when the position data is converted from double to single precision. This can cause simulation errors when the new position is on the other side of an object or it may cause a program error if the physical and computational locations no longer correspond.

The smallest value greater than zero that can be represented by a floating point number is called the epsilon constant, ε. This constant can be used to test for and correct values that will change to the next integer value on conversion from double to single precision.

For the case where molecules are index by the integer component of their position, ie. regular grids or computational positions, the following algorithm can be used to ensure that the conversion to a single precision does not cause a change in the index,

$i_1 = \text{floor}(P_{64})$
$P_{32} = P_{64}$
$i_2 = \text{floor}(P_{32})$
if ($i_2 > i_1$)
    $P_{32} = i_2 - \varepsilon_{32}$
else if ($i_2 < i_1$)
    $P_{32} = i_2 + \varepsilon_{32}$

where the subscripts $_{32}$ and $_{64}$ refer to a single or double precision floating point number respectively.

4.4.5 Solution Management

The program implements a number of features to automate and simplify the solution process. These features are steady flow detection, results formatting and solution state saving. All of these features are performed at a constant user selectable rate or can be selectively performed at any time.

4.4.5.1 Steady Flow Detection

For a steady flow simulation there is an initial instationary flow phase where the flowfield structure is developing towards its steady state composition. During this flow phase the molecule sample data cannot be incorporated in the final solution average. The primary indicator that the simulation has not reached its steady state composition is a non-constant number of molecules in the simulation. The program determines whether the steady flow state has been reached by examining the variation in the total number of simulation molecules over a user defined number of samples, N_t. The stationary flow phase is deemed to have been achieved if the following equality is true,

$$\left[\left(\overline{N}_m - 2\sigma_{N_m} \geq \min\left(N_m\right)\right) \& \left(\overline{N}_m + 2\sigma_{N_m} \leq \max\left(N_m\right)\right)\right] \tag{4.9}$$

where,

N_m is the set of the number of simulation molecules at each time step since the last test,

$\overline{N}_m$ is the average number of simulation molecules at each time step since the last test, and

σ_{N_m} is the standard deviation of the number of simulation molecules at each time step since the last test.

4.4.5.2 Results Formatting

During the instationary and stationary flow phases, the current results of the cell sampling are output to a data file. This output occurs at the same interval, N_t, used for steady flow checking. Before being written to the output data file, the cell sample data is processed to calculate the equivalent cell macroscopic properties. The output data file is written in a TecPlot text file format. Similarly, the surface collision sample data is processed to determine the surface segment macroscopic properties and appended to the end of the cell data file.

4.4.5.3 Solution State Saving

At the same interval that the steady flow checking and results output occurs, the program also saves the current state of the simulation. This allows the program to be stopped and the solution restarted from its current state at later time or alternatively on a different computer. This state file is in binary format to ensure that no precision is lost when converting to a text format and to save storage space.

4.4.6 Flow Chart

Figure 4.6 shows the full simulation flow chart where the following symbols are used,

n_t number of samples since last results processing

N_t number of samples between results processing

n_s number of steady state samples

N_s number of steady state samples to accumulate for solution

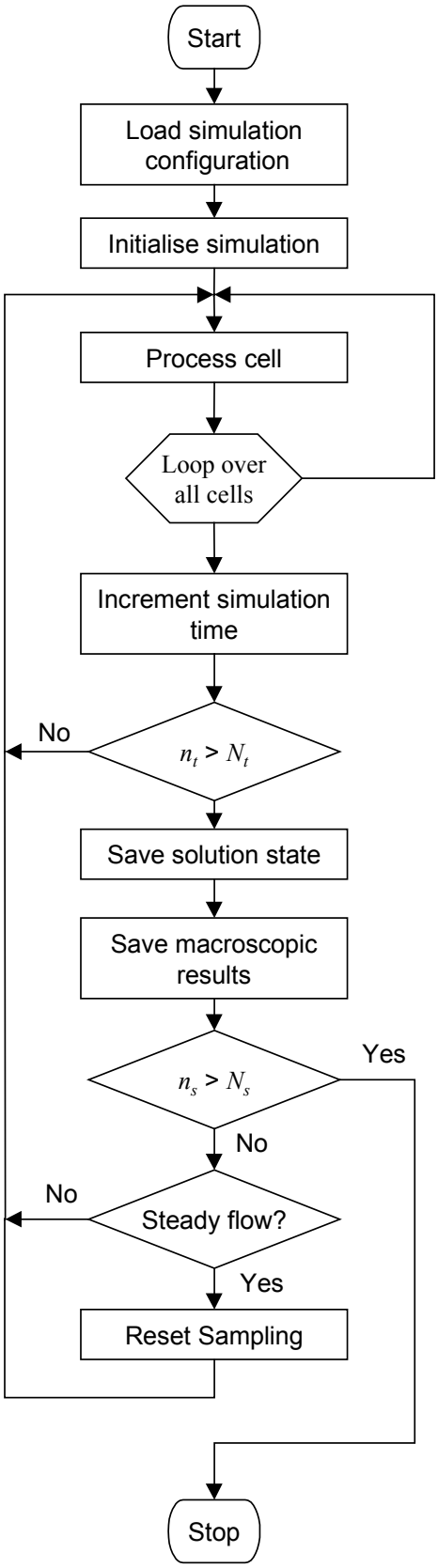


Figure 4.6 Full simulation flow chart

Chapter 5

Solution Adaption

5.1 Introduction

By its very definition, a simulation problem is something that does not have a predictable answer. It follows that it is not possible to set-up a DSMC simulation for a new flow problem with a configuration that guarantees an accurate solution. Generally, for most complex problems the simulation will have to be run multiple times, with the simulation configuration modified between runs to provide an accurate solution for the previous runs results, until there is no significant deference between successive runs. This is a time consuming task if undertaken manually. Alternatively, the DSMC program can be written to automatically adapt its configuration between runs, or during a run, to achieve an accurate solution.

The adaption procedure has the secondary benefits of decreasing the simulation time and computational resources required. This is because the non-adaptive solution would generally be configured conservatively to ensure an accurate solution [46].

This chapter discusses the procedures implemented to ensure that the simulation is configured to meet all the requirements for an accurate simulation as discussed in Section 2.4.

5.2 Grid Adaption

As discussed in Section 2.4.1, the maximum cell side length should be $\lambda/2$ and $\lambda/3$ for two- and three-dimensions respectively. Due to the complexity of the flows being simulated, the initial grid definition will not meet this requirement over the entire flow field. Each grid block is adapted independently using a two step process. First the grid cells are adapted to the flow field and then the grid size is adjusted if required.

The adaption step is performed before the final grid size is calculated because the resizing function sums the number of local mean free path lengths across the grid rows and columns. These sums are then used as the basis for calculating the new grid size. If this calculation is performed before the grid has been adapted, the result is not an accurate representation of the final grid structure.

5.2.1 Adaption Methodology

The grid points are redistributed using a one-dimensional grid adaption algorithm successively along grid lines of constant ξ and η. This method of successive one-dimensional adaptions requires less computational effort and is not susceptible to mathematical instabilities as methods that adapt rows and columns simultaneously. Additionally, the method is easily extensible to three-dimensional problems.

5.2.1.1 Base Algorithm

The base adaption algorithm utilises the model of equi-distributing weight functions such that,

$$\frac{\Delta s_k}{W(k)} = \text{constant} \tag{5.1}$$

where,

W is the adaption distribution function at the grid vertice, and

Δs_k is the desired local grid size for segment k along lines of η or ξ = constant.

Therefore, the expression for the grid adaption along $\xi = i$ is,

$$D_\eta(i,m) = \sum_{j=1}^{m} \frac{\Delta\eta_{i,j}}{W(j)} \Bigg/ \sum_{j=1}^{J} \frac{\Delta\eta_{i,j}}{W(j)} \tag{5.2}$$

where,

$D_\eta(i,m)$ is the relative cell spacing function for column i, and $\sum_{j=1}^{J} D_\eta(i,m) = 1$,

$\Delta\eta_{i,j} = \sqrt{(x_{i,j} - x_{i,j-1})^2 + (y_{i,j} - y_{i,j-1})^2}$, and

J is the maximum column ordinate.

5.2.1.2 Adaption Distribution Function

Woronowicz [60] proposes two rules relating to grid cell size to local flow properties that should be followed to ensure the results are accurate. A summary of these rules are,

1. Where the flow undergoes strong changes in macroscopic properties, ie. shock waves and shear layers, the cell spacing should not exceed λ.
2. Near the surface of an object, the cell spacing normal to the body should be less than λ.

These rules are similar in intent but less stringent to those defined in Section 2.4.1.

The authors suggest using a composite distribution function of λ and l_n, where l_n is the gradient length scale based on number density, as the adaption function. They use the term l_n to obtain a finer grid in regions of strong changes in macroscopic properties.

The equation to calculate l_n along a constant η line is,

$$\left(l_n\right)^{-1} = \left|\frac{\partial \ln n}{\partial \eta}\right| \tag{5.3}$$

Which can be seen to be of the form,

$$l_n \propto \frac{1}{n} \tag{5.4}$$

The hard-sphere (HS) definition of λ_{HS} is,

$$\lambda_{HS} = \frac{1}{\sqrt{2}\pi \bar{d}^2 n} \propto \frac{1}{n} \tag{5.5}$$

Equations 5.4 and 5.5 show that λ_{HS} and l_n are both proportional to the inverse of the number density. Additionally, Woronowicz [60] notes that the contribution of l_n to the composite distribution function is noisy. Therefore, if the more stringent grid spacing rules defined in Section 2.4.1 are followed, a single variable adaption distribution function based upon λ_{HS} will give an accurate grid for DSMC simulations.

The hard-sphere definition of λ is used as the adaption distribution function,

$$W = \lambda_{HS} \tag{5.6}$$

λ_{HS} is used instead of the variable-hard-sphere (VHS) definition as it is easily calculated and is not dependent on the temperature. As λ_{VHS} is proportional to temperature, this

increases the magnitude of λ_{VHS} in regions of relatively high temperatures, ie. shock fronts, [34]. Therefore, it is beneficial to use λ_{HS} for adaption since it is conservative relative to λ_{VHS} which maybe used in the collision process. All future references to the local mean free path, λ, will refer to the hard sphere definition unless otherwise explicitly noted.

As the λ data is generated for the cell centres and the redistribution algorithm adapts the grid cell vertices, the cell vertice data is estimated by bilinear interpolation of the surrounding cell centred data.

5.2.1.3 Grid Smoothing

As the distribution function is derived from cell averaged flow properties that are not in a state of equilibrium, there is a perceptible level of statistical scatter in the result. This statistical scatter is accentuated by areas of highly localised gradients in the adaption distribution function. If not corrected, these two factors result in a grid that is discontinuous and has excessive skewness. While, the DSMC method is relatively immune to these grid effects, programs used in post-processing to analyse the results have difficulties with these grid effects. A two step approach is implemented to smooth the adapted grid while maintaining the adapted grid distribution.

Firstly, the adaption distribution function used for each grid line is coupled to the adjacent grid lines as suggested by Jeng [27]. This coupling of adjacent grid lines reduces skewness caused by areas of localised gradients in the adaption distribution function. The modified adaption distribution function is calculated by,

$$W'(k) = \begin{cases} \beta_1 W(k-1) + (1-\beta_1-\beta_2) W(k) + \beta_2 W(k+1) & \text{, for interior lines} \\ \beta_1 W(k-1) + (1-\beta_1) W(k) & \text{, for boundary lines} \end{cases} \tag{5.7}$$

where,

$W'(k)$ is the modified adaption distribution function, and

β_1, β_2 are the weighting factors among grid lines, $\beta_1 = \beta_2 = 0.25$.

Secondly, the effects of the statistical scatter in the distribution function results in short-wave oscillations in the adaptive grid. The grid smoothing method presented in Section 4.3.2.2 significantly removes the short-wave grid oscillations while preserving the grid adaption.

5.2.1.4 Implementation

The grid points were redistributed using the following procedure,

1. Calculate the adaption distribution function, W, at the cell centres using Equation 5.6 and the macroscopic flow properties of the starting grid.
2. Use bilinear interpolation to calculate distribution function at the grid vertices using the starting grid and the result from step 1.
3. Apply Equation 5.2, using the distribution function defined in Equation 5.7, along lines of constant ξ. Adjust the grid block boundary grid lines, ξ = [1,I]; so that there is a grid vertice at all boundary control points.
4. Use bilinear interpolation to calculate distribution function at the grid vertices using the starting grid and the result from step 1.
5. Apply Equation 5.2, using the distribution function defined in Equation 5.7, along lines of constant η. Adjust the grid block boundary grid lines, η = [1,J]; so that there is a grid vertice at all boundary control points.
6. Apply the grid smoothing algorithm, Equation 4.8.
7. Optionally repeat steps 2-6 one or more times. The effect of changing the number of adaption cycle iterations is investigated in Section 5.2.3.

5.2.2 Grid Resizing and Splitting

The aim of the grid resizing and splitting functions are to ensure that the adapted grid meets the maximum cell side length requirement. If the flow field has large variations or localised strong gradients of λ across the grid block, it may not be possible for a structured grid to meet the cell side length requirements without making some cells significantly smaller than the maximum cell size. Large ratios of cell size (volume) lead to an inefficient simulation as the value of F_N is set by the smallest cell volume.

As discussed in Section 2.4.3, if the product of nV_c is constant, the number of molecules in a cell, N_m, will also be constant. Now Equation 5.5 shows that n is inversely proportional to λ, which implies that to achieve a constant number of molecules per cell the ratio V_c/λ should be constant. The ratio V_c/λ is equivalent to the grid adaption distribution function, Equation 5.1. Therefore, a well adapted grid will have a near

constant number of molecules per cell. To compensate for the case discussed in the previous paragraph where large variations in λ cause significantly smaller cells, the grid block should be split.

5.2.2.1 Implementation

The grid resizing function is implemented using the following methodology,

1. For each block, calculate the number of the local mean free paths along the lines of constant ξ and η.

$$S_{\xi j} = \sum_{i=1}^{I} \frac{\overline{\Delta\xi}_{i,j}}{\lambda(i,j)}\bigg|_j$$

$$S_{\eta i} = \sum_{j=1}^{J} \frac{\overline{\Delta\eta}_{i,j}}{\lambda(i,j)}\bigg|_i$$

where,

$$\overline{\Delta\xi}_{i,j} = 0.5\left[\sqrt{\left(x_{i,j} - x_{i-1,j}\right)^2 + \left(y_{i,j} - y_{i-1,j}\right)^2} + \sqrt{\left(x_{i,j+1} - x_{i-1,j+1}\right)^2 + \left(y_{i,j+1} - y_{i-1,j+1}\right)^2}\right]$$

$$\overline{\Delta\eta}_{i,j} = 0.5\left[\sqrt{\left(x_{i,j} - x_{i,j-1}\right)^2 + \left(y_{i,j} - y_{i,j-1}\right)^2} + \sqrt{\left(x_{i+1,j} - x_{i+1,j-1}\right)^2 + \left(y_{i+1,j} - y_{i+1,j-1}\right)^2}\right]$$

$\lambda(i,j)$ is the hard-sphere value of λ in the cell (i,j)

2. Calculate the grid block's new grid size.

The number of cells required to ensure that the adapted grid meets the maximum cell side length requirement is given by,

$$N_\xi = \frac{\max\left(S_{\xi j}\right)}{R_{\lambda,\max}}$$

$$N_\eta = \frac{\max\left(S_{\eta i}\right)}{R_{\lambda,\max}} \qquad (5.8)$$

where,

N_ξ and N_η are the number of cells in the ξ and η directions respectively, and

$R_{\lambda,\max}$ is the maximum allowable ratio of cell size to λ (see Section 5.2.3).

3. Determine if the grid block needs to be split.

   A grid block is split if the ratio of the maximum to the minimum number of local mean free paths along the lines of constant ξ and η, is greater than the ratio of maximum to minimum allowable cell size,

   $$R_\xi > R_{\text{Split}} \tag{5.9a}$$

   $$R_\eta > R_{\text{Split}} \tag{5.9b}$$

   where,

   $$R_\xi = \frac{\max\left(S_{\xi j}\right)}{\min\left(S_{\xi j}\right)}$$

   $$R_\eta = \frac{\max\left(S_{\eta i}\right)}{\min\left(S_{\eta i}\right)}$$

   $$R_{\text{Split}} = \frac{R_{\lambda,\max}}{R_{\lambda,\min}}$$

   $R_{\lambda,\,\min}$ is the minimum allowable ratio of cell size to λ (see Section 5.2.3).

4. If either or both of the inequalities of Equation 5.9 are true, then the grid block needs to be split.

   If only Equation 5.9a is true then the grid is split along lines of constant η. Conversely, if only Equation 5.9b is true then the grid is split along lines of constant ξ. Finally, if both Equation 5.9a and 5.9b are true then the direction of the split lines is chosen by the larger of R_ξ and R_η.

   Considering the case were the grid is split along lines of constant η, the number of splits is calculated by the following algorithm,

   if (R_ξ > 3×R_{Split})
   // requires more than two split lines
   // split once and re-split, if still required at next interval
   NoSplits = 1
   SplitPoint[1] = $\sqrt{R_\xi} \times \min(S_{\xi j})$
   else if (R_ξ > 2×R_{Split})
   // requires two split lines
   NoSplits = 2
   SplitPoint[1] = $\sqrt[3]{R_\xi} \times \min(S_{\xi j})$
   SplitPoint[2] = $\sqrt[3]{R_\xi}$ × SplitPoint[1]

```
else
    // requires one split line
    NoSplits = 1
    SplitPoint[1] = √R_ξ × min(S_ξj)
end
```

In the above algorithm, SplitPoint[n], corresponds to the value of $S_{\xi j}$ where the grid is to be split. The array of $S_{\xi j}$ data calculated in step 1 is searched through until a split point crossing is detected. This crossing becomes the index for the start of the next new grid block. The process is repeated until the end of the $S_{\xi j}$ vector is reached. The procedure is limited to creating three split lines, ie. four new grid blocks. This procedure is illustrated in Figure 5.1.

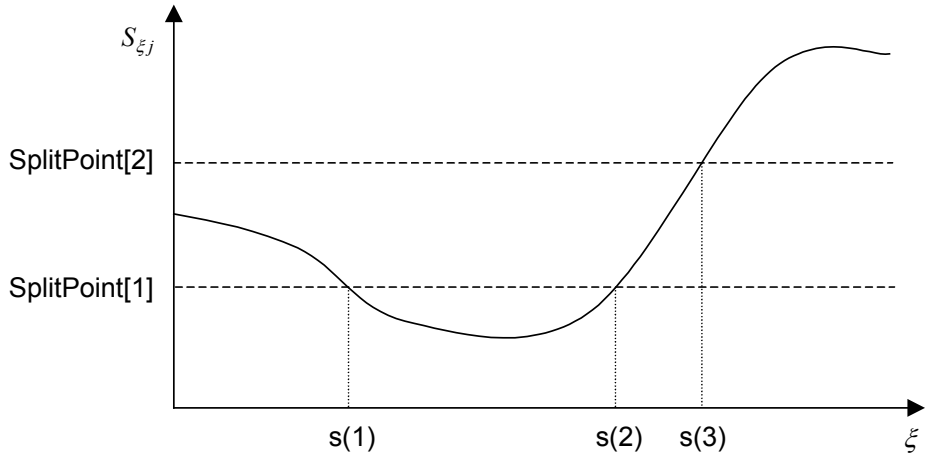


Figure 5.1 Grid split line calculation

5. To minimise the number of geometry segments used to define the grid, the current grid geometry is check to determine if there is an existing segment end points within 3 vertices of the desired split point location. If there is, the new split point is moved to be coincident with the existing end point.

6. If the grid block is to be split, the contour of the split line needs to be calculated. This contour should be a relatively simple geometry to minimise the number of grid points forced to be at the end points of geometric segments, while still allowing the shape of the current split line contour to be approximated relatively accurately.

   It was decided to limit the number of segments that could be used to describe the split contour to four segments. Testing showed that this produced a good compromise between contour accuracy and minimum complexity. The majority of

split contours only required two segments to describe them. The algorithm used to calculate the internal split points is described below and illustrated in Figure 5.2.

a. Calculate the length of the line that joins the two end points of the split.

b. Calculate the equation of the line that joins the two end points of the split.

c. Search along the current grid contour from both ends and find the first local maximum perpendicular distance to the line from step *b*.

d. Select the larger of the two local maxima from step *c*. If this distance is less than 5% of the value from step *a*, then no more internal points are required. Otherwise, change the line from step *b* to two segments using the local maxima as a new internal point.

e. If three internal points have been defined, end the process. Otherwise, repeat from step *b* with the new segment end point replacing one of the old end points

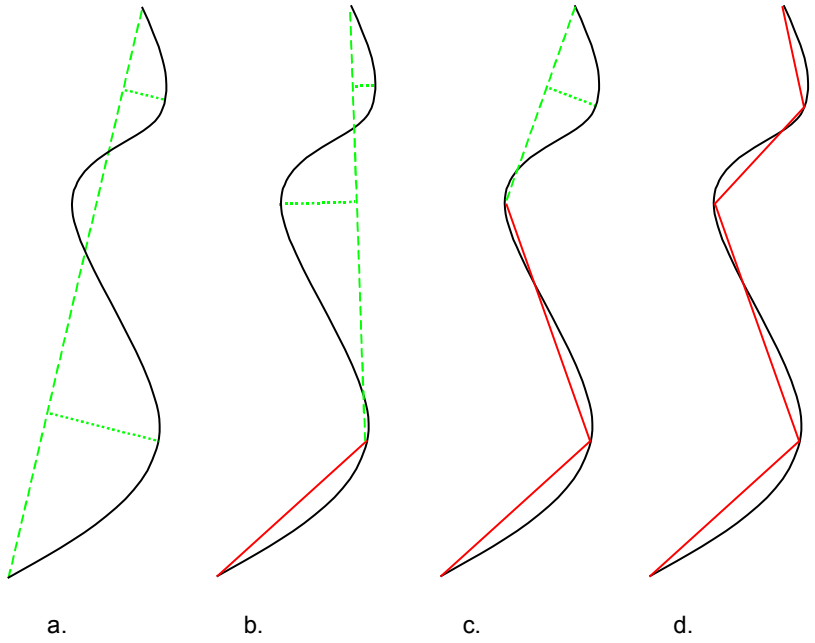


Figure 5.2 Split contour calculation

7. The number of cells in the new grid blocks are then calculated using Equation 5.8.

5.2.3 Parameter Tuning

Three tunable grid adaption parameters were identified in the previous two sections,

1. The number of adaption cycle iterations, A_n, (Section 5.2.1.4 Step 7),
2. The maximum allowable ratio of cell size to λ, $R_{\lambda,\max}$, (Section 5.2.2.1 Step 2)

3. The minimum allowable ratio of cell size to λ, $R_{\lambda,\,\min}$, (Section 5.2.2.1 Step 3).

The primary effect of each parameters is,

A_n defines how aggressively the grid is adapted to distribution function. Large values tend to produce excessive grid skew.

$R_{\lambda,\,\max}$ defines the maximum cell size and has direct correspondence to the minimum number of cells in the grid block.

$R_{\lambda,\,\min}$ defines the minimum cell size and the ratio between $R_{\lambda,\,\max}$ and $R_{\lambda,\,\min}$ affects how often grid blocks are split.

5.2.3.1 Test Series 1

Initial testing revealed that the first two parameters, A_n and $R_{\lambda,\,\max}$, are closely coupled. Therefore, a series of test were conducted to determine the optimum values for these two parameters. The simulated flow used in the test corresponded to a uniform λ distribution that has a sharp discontinuity diagonally across the simulation area, as shown in Figure 5.3. A uniform grid with a size equal to $\lambda/2$ was used as the starting condition for the test. The configuration parameters for the tests are listed in Table 5.1 and $R_{\lambda,\,\min}$ was kept constant at 0.25 for all tests.

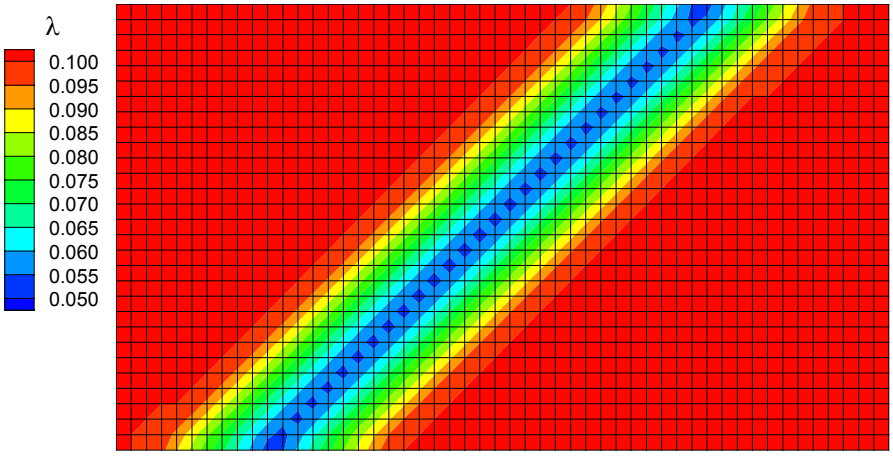


Figure 5.3 Grid adaption test case 1 flow field.

Config.	A1	A2	A3	B1	B2	B3
A_n	1	2	3	1	2	3
$R_{\lambda,\,\max}$	0.50	0.50	0.50	0.45	0.45	0.45

Table 5.1 Grid adaption test case 1.

Starting with the initial uniform grid, the adaption and resizing/splitting routines were run once using the listed configuration parameters. The resultant grid for each configuration is shown in Figure 5.4 a-f and the minimum and maximum value of the ratio of average cell side length to local mean free path, $\bar{l}_m/\lambda$, is summarised in Table 5.2.

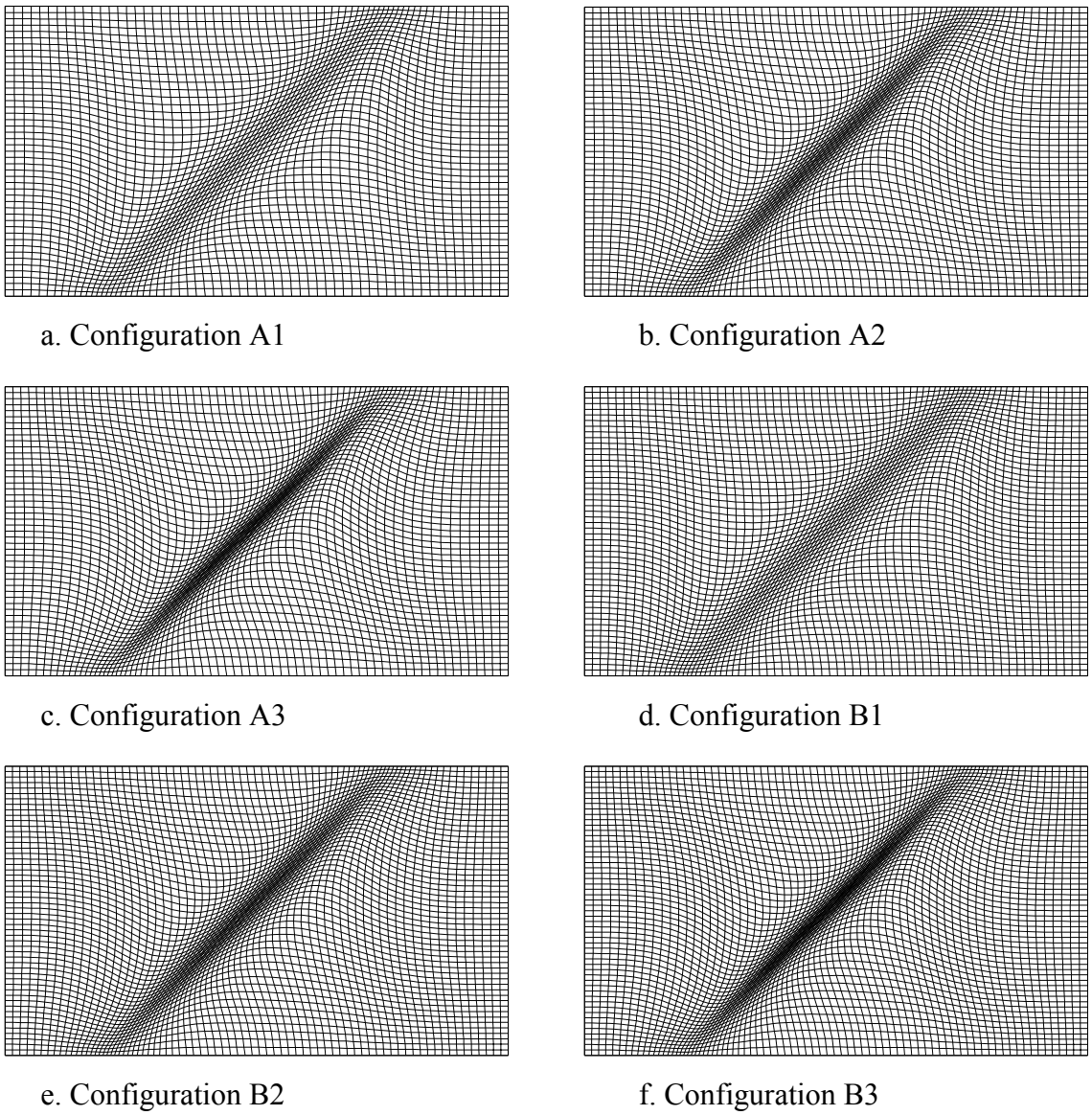
Config.	A1	A2	A3	B1	B2	B3
$\bar{l}_m/\lambda\|_{\max}$	0.63	0.56	0.55	0.57	0.51	0.49
$\bar{l}_m/\lambda\|_{\min}$	0.38	0.36	0.35	0.34	0.32	0.31

Table 5.2 Grid adaption test case 1 results.

A general trend for all of the test configurations was that with increasing A_n there was a marked increase in cell skewness along the line of minimum λ. While this effect is accentuated by the sharp discontinuity in the simulated flow field, the chosen parameters need to be able to work robustly for all flow fields. Based on the results of this test, it was found that configuration B2 offered the best combination of adaption solution and minimum cell skew. The adapted flow field for configuration B2 is shown in Figure 5.5. This configuration was used as the default value for the parameters A_n and $R_{\lambda,\max}$ for the remainder of this thesis.

5.2.3.2 Test Series 2

A second series of tests were conducted to determine the optimum value for the parameter $R_{\lambda,\min}$. Ideally, this parameter should be as close as possible to $R_{\lambda,\max}$ so that there is a minimum variation in $\bar{l}_m/\lambda$ across the solution domain. However, as $R_{\lambda,\min}$ is increased, the grid will be subdivided into more blocks to maintain the required ratio of $R_{\lambda,\max}$ / $R_{\lambda,\min}$. It is undesirable to have an excessive number of grid blocks as it constricts the solution adaption and increases grid management overhead.



a. Configuration A1 b. Configuration A2

c. Configuration A3 d. Configuration B1

e. Configuration B2 f. Configuration B3

Figure 5.4 Grid adaption test case 1 adapted grids

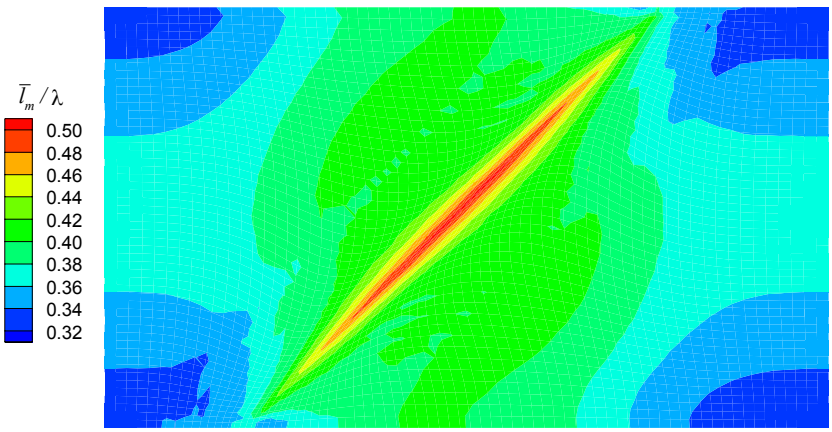


Figure 5.5 Grid adaption test case 1, configuration B2 adapted flow field

The simulated flow used in the second series of tests is shown in Figure 5.6. Again a uniform grid with a size equal to $\lambda/2$ was used as the starting condition for the test. The configuration parameters for the tests are listed in Table 5.3 and for all tests $A_n = 2$ and $R_{\lambda,\,\max} = 0.45$.

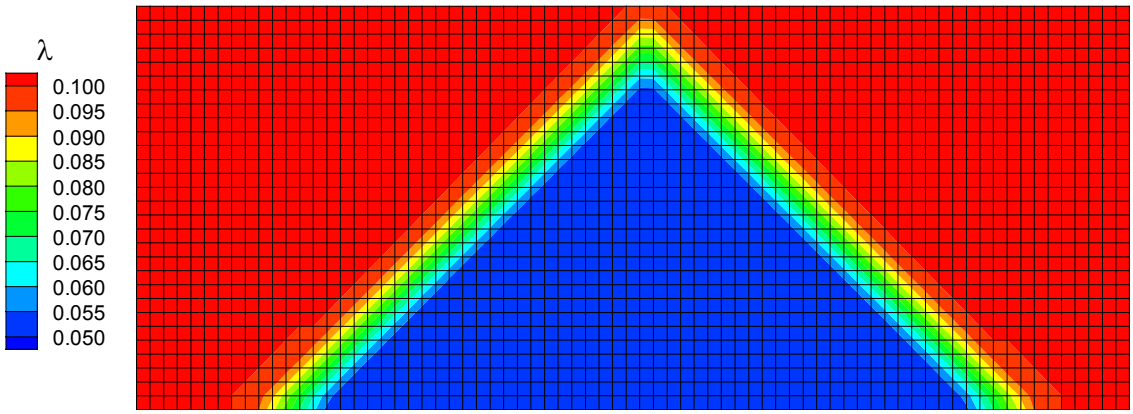


Figure 5.6 Grid adaption test case 2 flow field

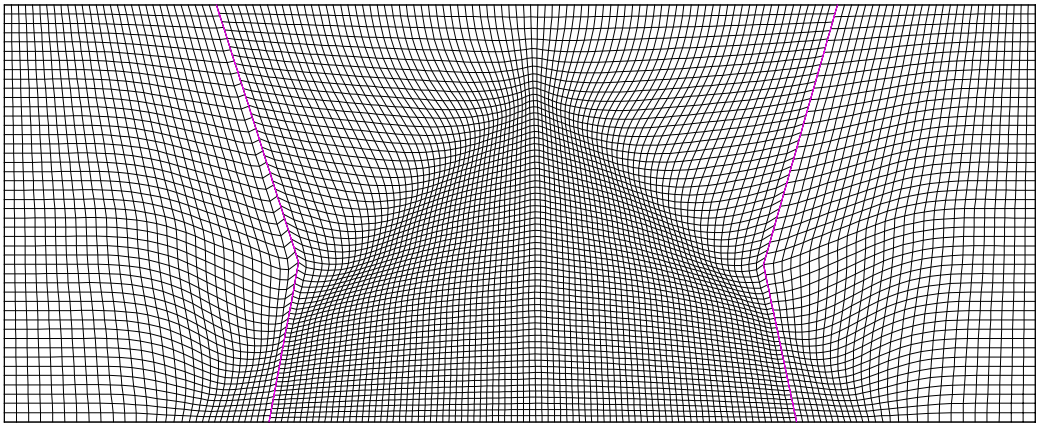
Config.	C1	C2
$R_{\lambda,\,\min}$	0.25	0.30

Table 5.3 Grid adaption test case 2

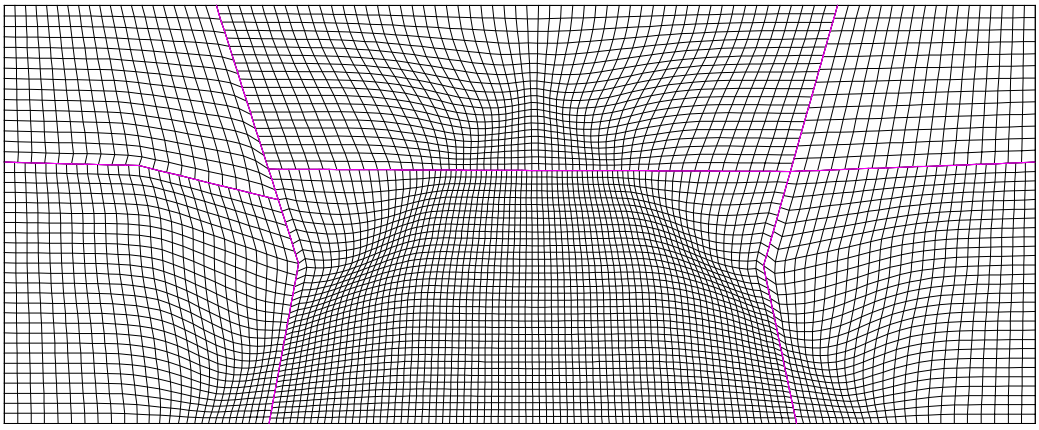
Starting with the initial uniform grid, the adaption and resizing/splitting routines were run using the listed configuration parameters until the number of grid blocks stabilised. The resultant grid for each configuration is shown in Figure 5.7 a-b and the minimum and maximum value of the ratio of average cell side length to local mean free path, $\bar{l}_m / \lambda$, is summarised in Table 5.4. In Figure 5.7 the magenta lines indicate grid block boundaries.

Config.	C1	C2
$\bar{l}_m/\lambda\big\|_{\max}$	0.48	0.49
$\bar{l}_m/\lambda\big\|_{\min}$	0.29	0.34

Table 5.4 Grid adaption test case 2 results.



a. Configuration C1



b. Configuration C2

Figure 5.7 Grid adaption test case 2 adapted grids.

Figure 5.7 shows that the principle difference between the two adapted grids is that the C2 configuration has three extra grid blocks. The addition of these grid blocks does not offer any significant improvement in the quality of the adapted grid. The adapted flow field for configuration C1 is shown in Figure 5.8. This configuration was used as the default value for the parameters $R_{\lambda,\,\min}$ for the remainder of this thesis.

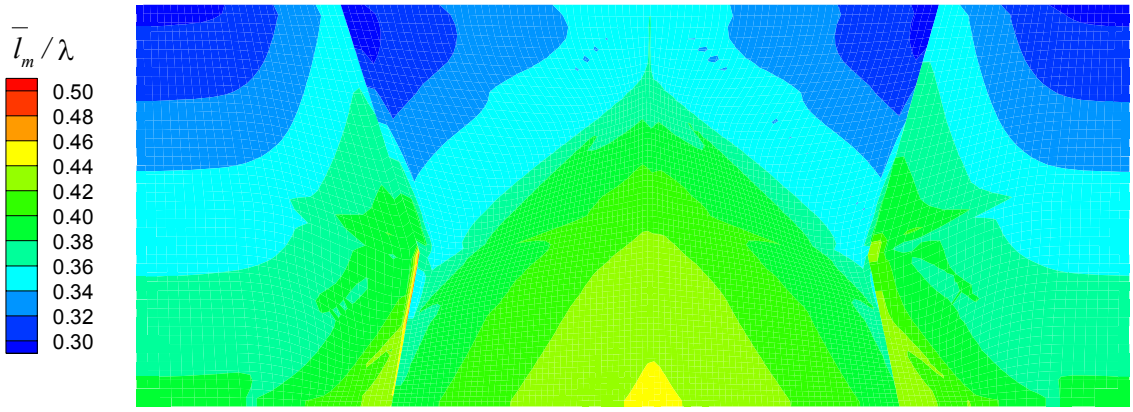


Figure 5.8 Grid adaption test case 2, configuration C1 adapted flow field.

5.2.4 Implementation

The implementation of the adaption algorithm outlined in the previous two sections is presented in Figure 5.9 below.

5.3 Ratio of Real to Simulation Molecule Adaption

As discussed in Section 2.4.3, the number of molecules in a cell, N_m, needs to be greater than 10, and preferably in the order of 20-30. Considering Equation 2.2 which defines the calculation of the number of molecules in a cell,

$$N_m = \frac{nV_c}{F_N}$$

The number density, n, is fixed by the flow field properties and the cell volume, V_c, is set by the results of the grid adaption. Therefore, the only way to adjust the number of molecules in a cell is through changing the ratio of real to simulation molecules, F_N.

After each grid adaption, the value of F_N is recalculated so that the no cell had less than $C_{Nm,\min}$ molecules and the average size cell had at least $C_{Nm,\text{ave}}$ molecules. This calculation is performed using the following algorithm,

$$F_{N,\min} = \frac{(nV_c)_{\min}}{C_{Nm,\min}}$$

$$F_{N,\text{ave}} = \frac{(nV_c)_{\text{ave}}}{C_{Nm,\text{ave}}}$$

if $F_{N,\min} < F_{N,\text{ave}}$

$$F_N = F_{N,\min} \tag{5.10a}$$

else

$$F_N = F_{N,\text{ave}} \tag{5.10b}$$

After calculating the new value of F_N, the molecule distribution needs to be recalculated. This is done using the same method as described in Section 4.2.4 for the flow field initialisation, except that the freestream number density is replaced with the local number density of each cell.

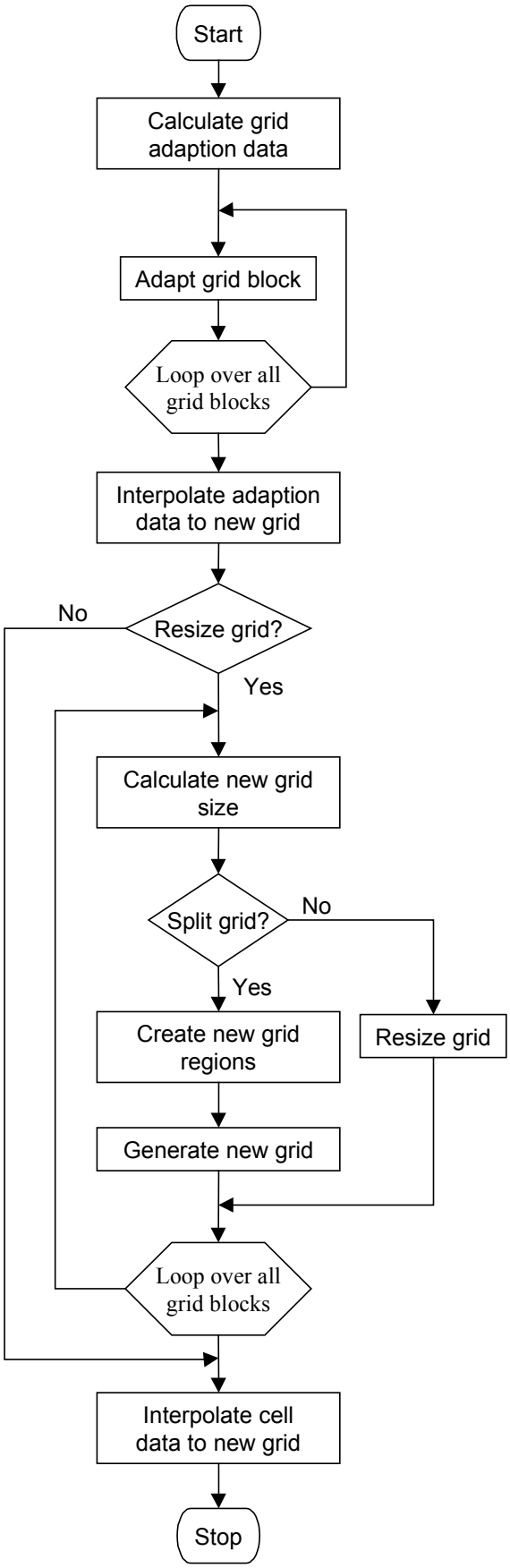


Figure 5.9 Grid adaption flow chart

5.3.1 Adjusting Grid blocks with an Excessive Number of Molecules per Cell

Results from initial grid and F_N adaption tests revealed that grid blocks with a significantly larger average λ than the rest of the simulation flow field also had a larger molecule number sum. This effect can be proved by starting with the equation that determines the number of molecules in a cell, N_m. Equation 2.2,

$$N_m = \frac{nV_c}{F_N}$$

This shows that N_m is proportional to the product n and V_c,

$$N_m \propto nV_c \tag{5.11}$$

Similarly, Equation 5.5 shows that λ is inversely proportional to n,

$$\lambda \propto \frac{1}{n}$$

Now, if the cell size is adapted to be proportional to λ, then for two-dimensions,

$$V_c \propto \lambda^2 \tag{5.12}$$

Therefore, substituting Equation 5.12 into Equation 5.11 gives,

$$N_m \propto \lambda \tag{5.13}$$

Equation 5.13 shows that if a grid block has higher average λ, the number of molecules per cell will be larger.

As the computational processing time is primarily dependent on the number of molecules and not the number of cells, it was decided to increase the cell density in grid blocks with a high average number of molecules per cell. This increased number of cells allowed a more even number of molecules per cell to be retained as well as increasing the resolution of the macroscopic flow field results.

The correction for excessive number of molecules in a cell is only performed after the grid had been adapted and the F_N calculated. This is done to ensure that grid blocks that had been corrected would not influence the calculation of F_N.

5.3.1.1 Implementation

The grid blocks new dimensions were calculated using the following algorithm,

1. Determine scale factor for grid rows and columns.

   The minimum sum of the number of molecules along the grid rows and columns was determined. This minimum value was then divided by the product of the number of cells in the row or column times $C_{Nm,\text{ave}}$. The resultant value was the desired scale factor for the grid rows or columns.

$$i_{scale} = \frac{\left(\sum_i N_m\right)_{\min}}{I \cdot C_{Nm,\text{ave}}}$$

$$j_{scale} = \frac{\left(\sum_j N_m\right)_{\min}}{J \cdot C_{Nm,\text{ave}}}$$

2. Check whether the application of the desired row and column scale factors will cause the cell with the minimum number of molecules to have less than $C_{Nm,\text{ave}}$ molecules. If this is found to be true, adjust the row and column scale factors.

$$\text{if}\left(\mathrm{i}_{\text{scale}} \cdot j_{scale} > \frac{(N_m)_{\min}}{C_{Nm,\text{ave}}}\right)$$

$$\mathrm{r}_{\text{scale}} = \frac{\mathrm{i}_{\text{scale}}}{j_{scale}}$$

$$i_{scale} = \sqrt{r_{scale} \cdot \frac{(N_m)_{\min}}{C_{Nm,\text{ave}}}}$$

$$j_{scale} = i_{scale} \cdot r_{scale}$$

3. Calculate the new number of cells in the grid block's rows and columns.

$$I' = \max(I, i_{scale} \cdot I)$$

$$J' = \max(J, j_{scale} \cdot J)$$

4. Resize the grid block using the new dimensions of (I',J') while maintaining the original grid distribution.

5.4 Time Step Adaption

As discussed in Section 2.4.2, the cell time step should be a small fraction of the local mean collision time, $\bar{t}_{mc}$. If a global time step is used for the simulation domain, then the value required to ensure an accurate result would be chosen based upon the cell with the highest collision frequency. Laux [36] showed that using this conservative approach resulted in a total simulation time that was more than six times slower than using a locally optimal time step. Where a locally optimal time step is defined as each cell having an individually defined time step that is a constant fraction of $\bar{t}_{mc}$. However, for a simulation to maintain physical accuracy, all cells, and molecules, must be kept in time synchronisation. Furthermore, the optimal time step for a cell or molecule will change while the simulation is in the instationary flow phase. Therefore, the complexity of implementing an optimal time step solution is in the selection of the time steps and efficiently keeping all cells and molecules in time synchronisation.

Two different methods of implementing a local time step adaption scheme have been proposed. Laux [36] and Olynick [46] use a scheme where the time step is determined at the cell level, while Bird [10] uses a scheme where the time step is set independently for each molecule.

The Olynick's cell based method uses a continuously variable cell time step. Where as Laux's method limits the cells time step to being integer multiples of a base time step. While the continuously variable scheme results in a more optimal time step, the method is not feasible to implement in a parallel environment. This is because synchronisation between parallel solution domains needs to occur at a fixed rate. Similarly, while Bird's molecule based scheme results in the optimal time step for each molecule, it too cannot be efficiently implemented in a parallel environment.

The time step adaption scheme implemented was based upon Laux's method and is summarised below.

5.4.1 Base Algorithm

The basic premise of the scheme is that each cell has an individual time step that is as an integer multiple, κ_i, of the base time step, Δt_b. As discussed in Section 2.4.2, a molecule should take longer than one time step to traverse a cell. It was decided that

the average molecule should take two steps to cross a cell. Therefore, as the average adapted cell side length is $\lambda/2$ for two-dimensional problems, the local time step made proportional to $\lambda/4$.

The procedure followed at each time step adaption is,

1. Calculate the desired cell time step,

$$\Delta t_{i,des} = \left. \frac{\lambda}{4\bar{c}'} \right|_i \tag{5.14}$$

2. Limit the change in cell time step to between 0.5 - 2.0 times its old value to avoid fluctuations caused by statistical scatter. This scatter is a result of the small number of samples used in the calculation of the cell macroscopic properties.

if $\Delta t_{i,des} > \left(\kappa_i \Delta t_b\right)\big|_{old}$

$$\Delta t_{i,new} = \min\left[2 \cdot \left(\kappa_i \Delta t_b\right)\big|_{old}, \Delta t_{i,des} \right] \tag{5.15}$$

else

$$\Delta t_{i,new} = \max\left[0.5 \cdot \left(\kappa_i \Delta t_b\right)\big|_{old}, \Delta t_{i,des} \right] \tag{5.16}$$

3. Calculate desired new base time step,

$$\Delta t_{b,new} = \min_i \left(\Delta t_{i,new}\right) \tag{5.17}$$

4. If desired new base time step difference relative to old base time step is less than 1%, use old base time step

$$\text{if} \quad \left| \frac{\Delta t_{b,new} - \Delta t_b}{\Delta t_b} \right| > 0.01 \tag{5.18}$$

$$\Delta t_{b,new} = \Delta t_{b,old}$$

5. Calculate new cell time step multiple,

$$\kappa_{i,new} = \text{div}\left(\Delta t_{i,new}, \Delta t_{b,new}\right) \tag{5.19}$$

if $\kappa_{i,new} < 1$

$$\kappa_{i,new} = 1$$

else if $\kappa_{i,new} > \kappa_{i,\max}$

$$\kappa_{i,new} = \kappa_{i,\max}$$

where,

$\kappa_{i,\max}$ is a user specified parameter.

6. Adjust all molecule time steps to account for changing cell time step,

$$\Delta t_{im} = \mathrm{mod}\left(N_{BTS}, \kappa_{i,old}\right) \cdot \Delta t_{b,old} + \left[\kappa_{i,new} - \mathrm{mod}\left(N_{BTS}, \kappa_{i,new}\right)\right] \cdot \Delta t_{b,new} \quad (5.20)$$

where,

Δt_{im} is *i*'ths molecules time step for the next movement iteration, and

N_{BTS} is the total number base time steps (main loop iterations).

5.4.2 Implementation

In Laux's implementation, the base time step is a user specified parameter. This puts the onus on the user to "guess" this critical parameter and will require the simulation to be repeated if too large a value is used or increase the solution time if too small a value is chosen. The scheme implemented in this thesis adapts the base time step along with the time step multiple of each cell. This allows the optimal base time step to be used for the simulation.

The maximum value of the time step multiple, $\kappa_{i,\max}$, is a user specified parameter. The value of $\kappa_{i,\max}$ used is trade-off between the reduction in solution time due to an optimal cell time step versus an increase in solution time due to an increased number of base time steps required to allow all cells to be sampled a minimum time. Laux's results showed a logarithmic speedup in solution time. By default $\kappa_{i,\max}$ was set to 20 which was the roll-off point in Laux's results.

Each iteration of the DSMC main loop incremented the global simulation base time time by Δt_b. The program then determined which cells were scheduled to be processed during the main loop iteration by pre-calculating which time step multiples to process. A time step multiple is processed if the following equality was true,

$$\mathrm{remainder}\left(\frac{N_{BTS}}{\kappa_i}\right) = 0 \quad (5.21)$$

When a cell is processed the sampling, inflow, collision and movement routines are advanced with a time step of $\kappa_i \Delta t_b$. The sampling routine was only called at a user definable rate of N_s iterations and the inflow routine was only called if the cell had an edge defined as an external free stream boundary.

If a molecule crossed a cell boundary during its movement phase, the time increment that the molecule was moved the next time in its new cell was adjusted to,

$$\Delta t_{im} = \left[\kappa_{i,new} - \mathrm{mod}\left(N_{BTS}, \kappa_{i,new}\right)\right] \cdot \Delta t_b \tag{5.22}$$

where,

$\kappa_{i,new}$ is the base time step multiple of the molecules new cell.

Therefore, Δt_{im} is equivalent to the time remaining before the molecule's new cell is next processed.

5.4.2.1 Initialisation

At the start of the simulation, the base time step is specified as being equal to the time required for a molecule with the maximum expected velocity to traverse a quarter of the freestream mean free path. The maximum expected velocity is either the average thermal velocity or freestream velocity.

$$\Delta t_b = \frac{\lambda_\infty}{4 \cdot \max\left(\bar{c}'_\infty, \left|V_\infty\right|\right)} \tag{5.23}$$

and all cells are assigned a time step multiple of, $\kappa_i = 1$.

5.5 Solution Adaption Implementation

The solution adaption implementation is dependent upon whether a steady or unsteady flow is being simulated. One implementation procedure common to both flow types is the processing of all flow boundary cells prior to adaption.

5.5.1 Molecule Injection

As the adaption processes will change the cell size and/or time step, all flow boundary cells need to be processed so that the molecule injection is correctly accounted for. The standard molecule injection procedure is modified in the following manner.

1. The injection time period, $\Delta t_m = \kappa_i \_ \Delta t_b$, for the cell is changed to use the time since the cell was last processed for molecule injection,

$$\Delta t'_m = \mathrm{remainder}\left(\frac{N_{BTS}}{\kappa_i}\right) \cdot \Delta t_b \tag{5.24}$$

2. After the adaption process, the remainder component from injection process, $R_{i,p}$ from Equation 4.2, will be reset to zero. Therefore, to account for this component of

molecule influx, $R_{i,p}$ is used as the cut off point in random sampling to determine whether an extra simulation molecule is injected,

if $R_f \leq R_{i,p}$

$$\dot{N}_{i,p} = \dot{N}_{i,p} + 1$$

where,

R_f is random value between [0,1].

5.5.2 Steady Flow Problems

For steady flow problems, the solution adaption is only performed during the instationary phase of the simulation. It was found during testing that the grid adaption process was more susceptible to statistical noise caused by low sample count, whereas the time step adaption process was more resilient to statistical noise. Furthermore, the accumulated sample data can be retained after a time step adaption, whereas it needs to be reset after a grid adaption. Therefore, a number of time step adaptions can occur between grid adaptions without affecting the grid adaption rate.

Time step adaption is performed every N_t base time steps, while grid adaption is performed every N_g time step adaptions. In addition, after every grid adaption process, the ratio of real to simulation molecules adaption process is performed. When both time step and grid adaption processes are performed during the same interval, the grid adaption process is performed first. This is implemented by saving a copy of the old grid cell centre coordinates, and using this information to interpolate the sampled cell temperature and λ onto the new grid.

5.5.2.1 Simulation Molecule Regeneration

Whenever the grid is changed, ie. when the number of cells in a grid block was adjusted because of an excessive number of molecules per cell, the simulation molecules need to be redistributed. This can be simply done using the general point location algorithm discussed in Section 3.4.4.3. However, for the majority of grid adaption processes F_N was also changed. This change in F_N requires the simulation molecules to be regenerated.

This regeneration is performed using a modified version of the process detailed in Section 4.2.4. Specifically, Equation 4.4 which defines the equilibrium number of molecules of each species in a cell, is changed to use the local cell macroscopic properties instead of the free stream flow properties.

5.5.2.2 Accelerated Flow Field Initialisation

One common problem with steady state flow problems is the time taken for the simulation flow to develop to its steady state condition. The time taken for the solution to reach its steady state is proportional to how fast a molecule can move through the simulation domain. By relaxing the cell size to λ adaption criteria and maintaining the same ratio of molecule time steps per cell, the number of time steps required for a molecule to traverse the simulation domain will be decreased. In addition, if the criteria on the number of molecules per cell is also relaxed, then the clock time required to process a time step will be reduced. While relaxing these criteria will result in the solution not being physically accurate enough to be used for the generation of results, it will still have macroscopic flow properties that are close to correct values. These macroscopic flow properties can then be used to initialise a solution that has the correct cell size and number of molecules.

The accelerated flow field procedure was implemented by allowing the user to optionally specify the ratio by which the cell size to λ, $R_{\lambda,\mathrm{accl}}$, and number of molecules per cell, $R_{Nm,\mathrm{accl}}$, criteria could be relaxed. The simulation was then run in the accelerated configuration until steady state flow conditions were detected. At this point the cell size and number of molecules per cell relaxation parameters were reset and the steady flow flag is cleared. The simulation then proceeds in instationary mode until steady state flow conditions were detected again. Initial testing showed that a value of 2 for both $R_{\lambda,\mathrm{accl}}$ and $R_{Nm,\mathrm{accl}}$ resulted in a good compromise between solution acceleration and the accuracy of accelerated flow field.

5.5.2.2.1 Grid Adaption Modification

For the grid adaption function, the parameter $R_{\lambda,\mathrm{accl}}$, was inserted into Equation 5.8 as follows,

$$N_\xi = \frac{\max\left(S_{\xi j}\right)}{R_{\lambda,\max} \cdot R_{\lambda,\text{accl}}}$$
$$N_\eta = \frac{\max\left(S_{\eta i}\right)}{R_{\lambda,\max} \cdot R_{\lambda,\text{accl}}} \tag{5.25}$$

Therefore, values of $R_{\lambda,\text{accl}}$ greater than 1 result in a reduced number of cells.

5.5.2.2.2 Ratio of Real to Simulation Molecules Adaption Modification

For the ratio of real to simulation molecules adaption, the parameter $R_{Nm,\text{accl}}$, was inserted into Equation 5.10 as follows,

$$F_N = F_{N,\min} \cdot R_{Nm,\text{accl}} \tag{5.26a}$$

and,

$$F_N = F_{N,\text{ave}} \cdot R_{Nm,\text{accl}} \tag{5.26b}$$

5.5.2.3 Implementation Flow Chart

For steady flow problems, the time step adaption is optionally performed every N_t base time steps. Generally, N_t was set to be an integer fraction of the ensemble sampling period.

Referring to Figure 4.6 the modified flow chart including the solution adaption processes is shown in Figure 5.10.

5.5.3 Unsteady Flow Problems

While it is possible to adapt the grid used for unsteady flow problems this is generally not done because the grid adaption function creates the optimum grid for the average flow field that was present before the adaption was calculated. By their very nature, unsteady flow problems are in a continual state of flux and therefore, the grid would generally be sub-optimal for the current flow conditions. Furthermore, for unsteady flow problems it is generally desirable to compare the simulation results using the same background grid.

For unsteady flow problems time step adaption is optionally performed every N_t base time steps. Generally, N_t was set to be an integer fraction of the ensemble sampling period.

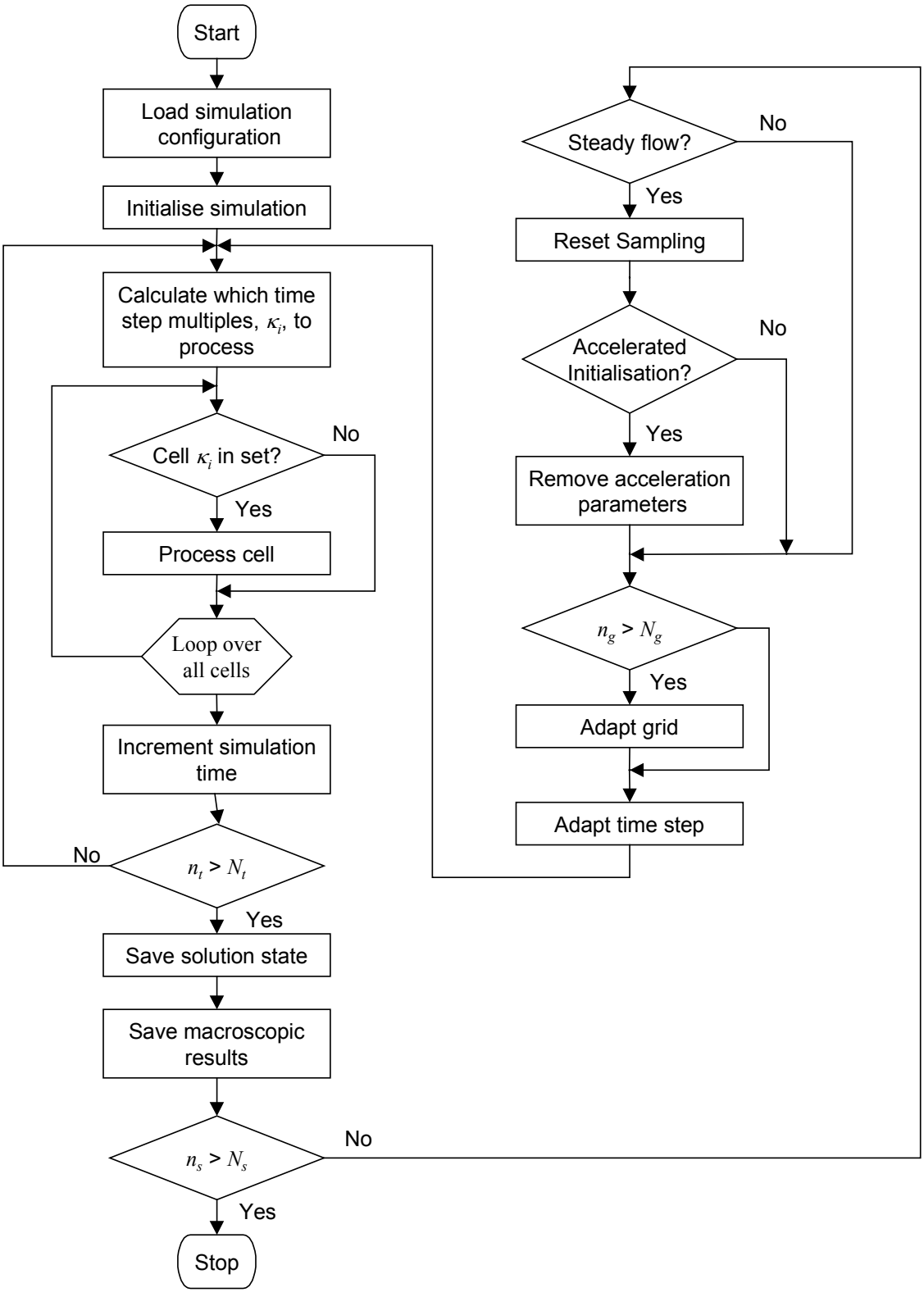


Figure 5.10 Steady flow adaption solution procedure flow chart

Chapter 6

Parallelisation

6.1 Introduction

The computational resources required by a DSMC simulation increase rapidly as the simulation Knudsen number decreases and approaches the continuum regime. Similarly, the computational requirements of three-dimensional problems are generally two orders of magnitude more than two-dimensional problems. These large computational requirements significantly limit the range of problems that can be practically solved on an engineering workstation or desktop computer.

Initial attempts by the DSMC community to solve this problem focused on utilising the most powerful computers available at the time, vector supercomputers. While the DSMC method was successfully vectorised, [5], [14] & [47], it was found that the method was not efficient for this processing implementation. This is primarily due to the random nature of the DSMC methodology. Specifically, only the movement phase of each time step is common to all molecules, while the molecules involved in a collision or boundary surfaces interaction is different at each time step and cannot be determined apriori.

From the start of the 1990s the computing power of engineering workstations and desktop computers began to increase significantly. At the same time, the availability of relatively fast inter-computer networking capabilities became available. This lead to the rise of distributed parallel processing as a viable alternative to vector supercomputers, [56].

The DSMC method is well suited to the distributed parallel processing because, during a time step, each cell can be processed independently and the only inter-cell communication required is when a molecule changes cells. Exploiting this property, the physical simulation domain can be decomposed into sub-domains, where each sub-

domain is assigned to a processing node[4] and runs a separate DSMC simulation for the cells in its sub-domain. All processing nodes are synchronised after each simulation time step and molecules that have crossed a sub-domain (node) boundary are transferred to the node that hosts the molecule's destination cell. This is the predominate method for parallelising DSMC and there is a large body of literature detailing different physical domain decomposition (PDD) implementations, [20], [28], [35], [37], [44], [51], [57], [58].

In [12] Bogdanov et. al. describes an alternative distributed parallelisation method where multiple processing nodes solve the same DSMC simulation, ie. identical grids. There is no molecules exchange between nodes and synchronisation only occurs for the purpose of sample data aggregation after the simulation has finished. This implementation makes uses of the DSMC property that different runs of the same simulation are statistically independent. Bogdanov refers to this implementation by the name Parallel Statistically Independent Runs (PSIR).

6.2 Physical Domain Decomposition Parallelisation

As mentioned in the introduction, physical domain decomposition (PDD) is the most common parallel implementation of the DSMC method. Figure 6.1 illustrates the general program flow chart.

There are a number of factors that effect the efficiency of a PDD solution. These factors can be grouped together into four areas,

1. Domain decomposition algorithm,
2. Dynamic load balancing,
3. Total node computational load, and
4. Node grid structure.

These efficiency factors are addressed in the following sections.

[4] The term node is used to refer to an instance of a DSMC simulation running in a processor. Generally, there would only be a single node per physical processor, but this is not guaranteed. Furthermore, some high end processors have multiple processing units in a single physical package. Therefore, the term node is used to maintain generality.

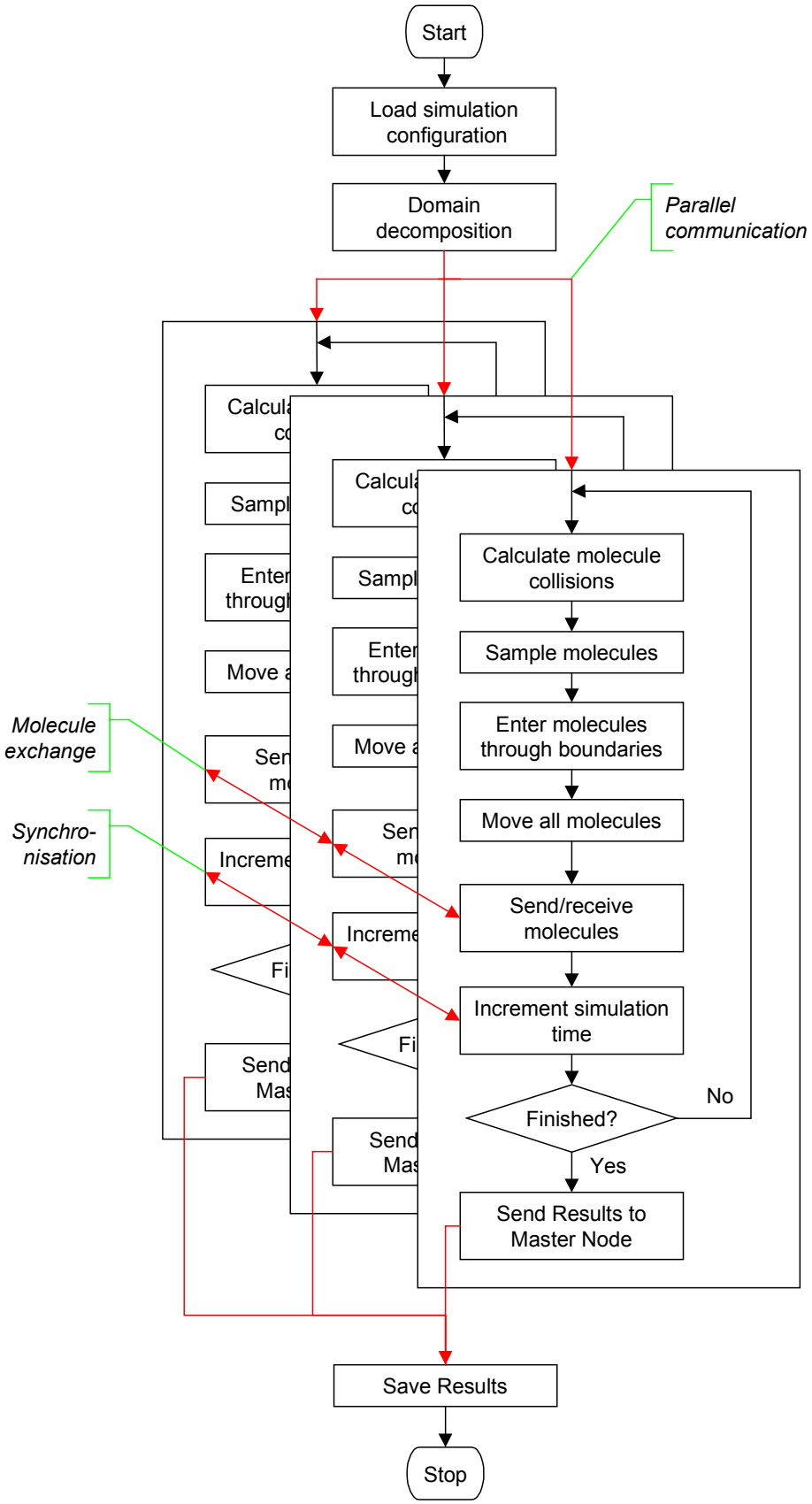


Figure 6.1 Physical domain decomposition flow chart

6.2.1 Domain Decomposition Algorithm

The primary aim of the domain decomposition algorithm is to equally distribute the load between the processing nodes. There is also a secondary aim of the minimising sub-domain molecule movement.

Nance et. al., [44] found that the quality of the domain decomposition results obtained from using two different measures of computational load, compute time per cell and the number of molecules per cell, were almost indistinguishable. Generally, the number of molecules per cells is used as the primary measure of computational load for a cell as it is already calculated as part of the sampling process whereas the compute time per cell requires extra data sampling. Other factors that effect the relative computational load of a cell are whether the cell has a side that is an inflow boundary or a non-internal boundary. The computational load associated with an inflow boundary is proportional to the molecule influx of the boundary. While for a non-internal boundary, the computational load is a function of the boundary type, ie. diffuse object, specular object or symmetry surface, and the average number of molecules that interact with the surface per unit time.

The second major factor that effects the efficiency of a decomposed domain is the amount of inter-domain molecule movement. Generally, it is preferable to split domains along streamlines to minimise inter-domain molecule movement. However, the local stream direction is only the average of the local molecules velocities. For cells with a local Mach number less than 2 there can be significant variations between molecules velocities in the same cell. The average number of molecules that cross a cell side per unit time can be calculate from Equation 4.2.

There are two main domain decomposition partition methods in general use, grid based methods and graph based methods.

6.2.1.1 Grid Based Partitioners

As the name implies, grid based partitioners decompose the simulation domain by using the grid as a template. These methods are most often used with structured grids but can also be used on unstructured grids. The two most commonly used grid based partioners are recursive bisection [57] and chain partitioning [41].

The recursive bisection method, recursively halves the domain until the number of sub-domains equals the number of processing nodes. The choice in which direction to split the domain can be made either to align the split with the free stream flow direction or to minimise the aspect ratio of the new domains [57]. In general, splitting the domain parallel to the free stream flow direction should result in less inter-domain molecule movement. However, this benefit will be reversed when the sub-domain aspect ratio reaches a threshold value. The main disadvantage of recursive bisection implementations is that it requires the number of sub-domains to be equal to a power of two.

The chain partitioning method divides the domain into a set of contiguous strips, or chains, that have the same amount of computational work. This method creates sub-domains that generally have high aspect ratios and is therefore most effective when the problem being solved is highly directional. An advantage of this method relative to the recursive bisection method is that it can generate a balanced decomposition for any number of processing nodes.

6.2.1.2 Graph Based Partitioners

Graph based partitioners are commonly used to partition finite element method problems. The basic structure of these methods is to create a graph of the domain being partitioned, then by first coarsening and subsequently refining this graph generate a decomposition of the domain. The advantages of these types of partitioners is that they produce a domain decomposition that both balances the domain work load while minimising the amount of inter-domain communication. These partitioners work well with finite element problems because the inter-cell communication load is well defined. However, for DSMC simulations the inter-cell communications is variable and expensive to calculate. Another disadvantage of graph based methods is that they do not guarantee the creation of singly connected sub-domains, which has a significant effect on the efficiency of the decomposition [35].

6.2.2 Dynamic Domain Decomposition

During the instationary phase of a steady flow simulation or throughout an unsteady flow simulation, the relative computational loading of different regions within the simulation domain can change significantly. This results in the initial domain

decomposition becoming unbalanced. As all processing nodes need to be synchronised after every movement phase, the physical time taken to process each time step is equal to the slowest node. Therefore, any imbalance between processing nodes has a significant effect on the efficiency of a PDD solution. To improve the solution efficiency most parallel DSMC programs implement dynamic domain decomposition to keep the load balanced across all nodes.

Nance [44] states that repartitioning the domain decomposition at fixed intervals can lead to poor performance. Therefore, it is preferable to us a variable interval strategy that determines at what interval to repartition the domain based upon the current solution performance. The Stop-at-Rise (SAR) algorithm of Nicol and Saltz [45] is the most commonly used method by DSMC programs [37], [44], [49] & [50].

6.2.2.1 Stop-at-Rise Algorithm

Nance [44] applies the SAR method to the problem of when to repartition the domain decomposition results in the form of a system degradation function, W, which is given by,

$$W(n) = \frac{\sum_{j=1}^{n} \left[t_{\max}(j) - t_{\text{avg}}(j) \right] + C}{n} \tag{6.1}$$

where,

- n is the number of time steps since the last remapping,
- $t_{\max}$ is the maximum time required by any one processing node during the j^{th} time step,
- t_{avg} is the average time required by a processing node during the j^{th} time step, and
- C is the time required to repartition the domain decomposition.

The quantity W is equivalent to the average node idle time per time step if the domain decomposition is repartitioned this time step. This quantity is monitored during the solution process and the domain decomposition is repartitioned at the first local minimum is detected, ie. $W(n) > W(n\text{-}1)$. This method works by estimating ratio of the time cost to perform a domain decomposition versus using the current decomposition. Using this method allows the decision of when to perform a domain decomposition to be calculated automatically based upon the current state of the simulation.

Nance states that SAR method is advantageous because “no prior knowledge of the problem is necessary for the determination of the remapping interval”. This is the case when the solution is in the instationary phase and the number of time steps until the steady flow state is achieved is unknown. However, in the stationary phase of a steady flow solution or throughout an unsteady flow simulation, the number of time steps until the end of solution is known. This information can be used to calculate an improved estimate of the optimal time to repartition the domain decomposition.

6.2.2.2 Constrained Stop-at-Rise Algorithm

During the instationary phase of the solution process, grid adaption occurs at a predefined interval and a new domain decomposition is calculate at the completion of each grid adaption. It would therefore be inefficient to recalculate the domain decomposition just prior to a grid adaption calculation. Similarly, during the stationary phase of the solution process it is possible to estimate the number of time steps until the completion of the simulation.

Therefore, by using this information it is possible to calculate whether time will be saved by recalculating the domain decomposition this time step. This is done by comparing the ratio of the estimated time until the next scheduled domain decomposition if the current domain decomposition is retained relative to recalculating the domain decomposition this time step. This ratio is given by,

$$G(n) = \frac{\frac{n_{rem}}{n}\sum_{j=1}^{n}\left[t_{\max}(j)\right]}{\frac{n_{rem}}{n}\sum_{j=1}^{n}\left[t_{\text{avg}}(j)\right] + C} = \frac{n_{rem}\sum_{j=1}^{n}\left[t_{\max}(j)\right]}{n_{rem}\sum_{j=1}^{n}\left[t_{\text{avg}}(j)\right] + nC} \tag{6.2}$$

where,

G is the estimated gain if the domain decomposition was repartitioned this time step, and

n_{rem} is the number of remaining time steps until the next scheduled domain decomposition or the completion of the simulation.

If G is greater than unity, then time will be saved by recalculating the domain decomposition this time step.

6.2.2.3 Repartitioning Strategy

After the domain decomposition has been repartitioned, the cell data and any associated molecules need to be sent between nodes for cells that changed sub-domains. In order to reduce the amount of inter-domain communication, and thus reduce the recalculation time, it is advantageous to minimise the difference between the old and new domain decompositions.

There are two methods for repartitioning the domain, recalculating the decomposition and adapting the current decomposition. Recalculating the decomposition will result in the optimal domain decomposition, but can take longer to calculate. Additionally, recalculating the domain does not guarantee any correspondence between the old and new domain decompositions.

Adapting the old decomposition, should result in the maximum correspondence between the old and new domain decompositions. However, it can be difficult to generate an optimal decomposition especially when there are a large number of sub-domains. Robinson [51] found that when the number of sub-domains exceeded a critical threshold, the graph based adaptive repartitioning scheme used by DSMC program generated an oscillatory decomposition that had a load imbalance greater than 20%.

6.2.3 Total Node Computational Load

Results from a number of different studies, Wilmoth [57], Robinson [51] and Dietrich [19], have shown that for a fixed size problem, the parallel speedup reaches a maximum at some number of processing nodes, and increasing the number of nodes beyond this number will lead to reduced efficiency. Similarly, Wilmoth [57] showed the relative speedup of parallelising a problem increased as the number of molecules per node increased. Furthermore, Robinson [51] found that as the number of particles per sub-domain decreases the statistical scatter in the load balancing becomes more significant, leading to a higher level of unbalance. Therefore, it is better to have each node as fully loaded as possible so that the ratio between computation time and communication time is large.

Dietrich [19] demonstrated this theory by measuring the parallel efficiency of a simulation using two different parallelisation loading criteria. The first loading criteria kept the total number of molecules in the simulation constant, while for the second criteria kept the number of molecules per node constant. The results of this demonstration are shown in Figure 6.2.

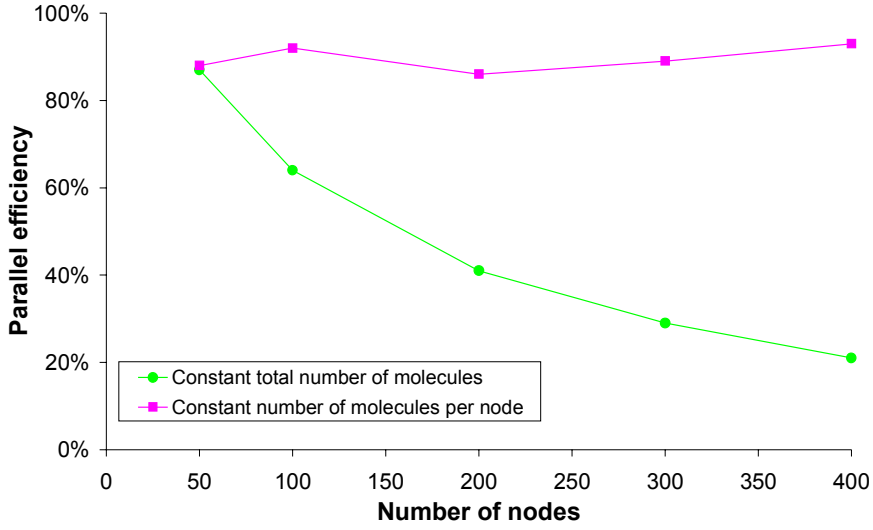


Figure 6.2 Measured parallel efficiency of the PDD method ([19], Figure 17)

6.2.4 Node Grid Structure

One area where different implementations of PDD method differ is in the storage of the grid at the processing nodes. Some implementations only store the grid used by the sub-domain [37], others add a "halo" of a single layer of cells around the sub-domain [51] and finally, other implementations give a complete copy of the grid to each node [35].

Storing only the grid used by the sub-domain at the node results in the minimum memory requirement. However, it requires a custom grid definition to be generated for each node. More significantly, when a molecule crosses a sub-domain boundary the final cell location of the molecule cannot be determined. Instead, the location and time where the molecule intersects the sub-domain boundary is calculated and this information is sent to the node on the opposite side of the boundary. If the molecule passes close to the confluence point of multiple sub-domain boundaries, it is possible for the molecule to cross more than one boundary in its time step. This would require

the molecule to be passed between multiple nodes and more significantly result in multiple node synchronisation points. This process is illustrated in Figure 6.3.

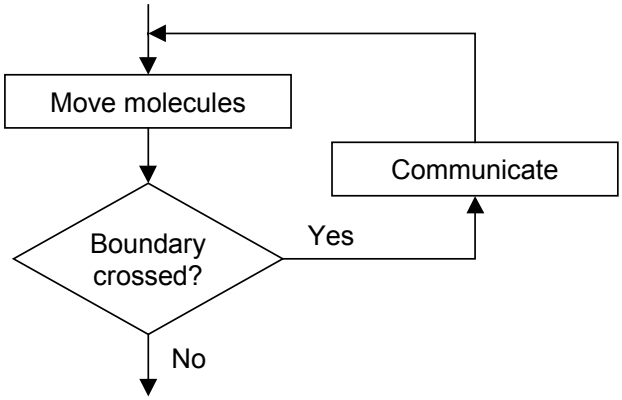


Figure 6.3 Extract of sub-domain grid movement flow chart

Using a halo layer of cells around the sub-domains grid significantly reduces the number of times that a molecule is communicated more than once. However, this is partially offset by the requirement of generating a more complex custom grid definition for each node. This can become a significant burden during the instationary solution phase when the domain may be adapted or repartitioned multiple times.

Having the entire grid available at each node negates the requirement to generate a custom grid for each node allows the molecules final cell location to be calculated when it crosses a sub-domain boundary. The main disadvantage of this method is that the memory required for storing the grid on each node is constant. For the majority of problems the storage requirement of the grid relative to the molecule information is small. However, for large three-dimensional simulations it can be significant.

6.3 Parallel Statistically Independent Runs

As discussed in Section 6.2.3 the efficiency of a domain decomposed parallel DSMC implementation increases as the ratio between computation and communication time increases. The parallel statically independent runs (PSIR) method extends this concept to the limit by removing the requirement to synchronise nodes at every time step. This is achieved by giving each node a full copy of the simulation domain and only combining the sample data at the end of the simulation for the purpose of calculating the results. Figure 6.4 illustrates the general PSIR method flow chart.

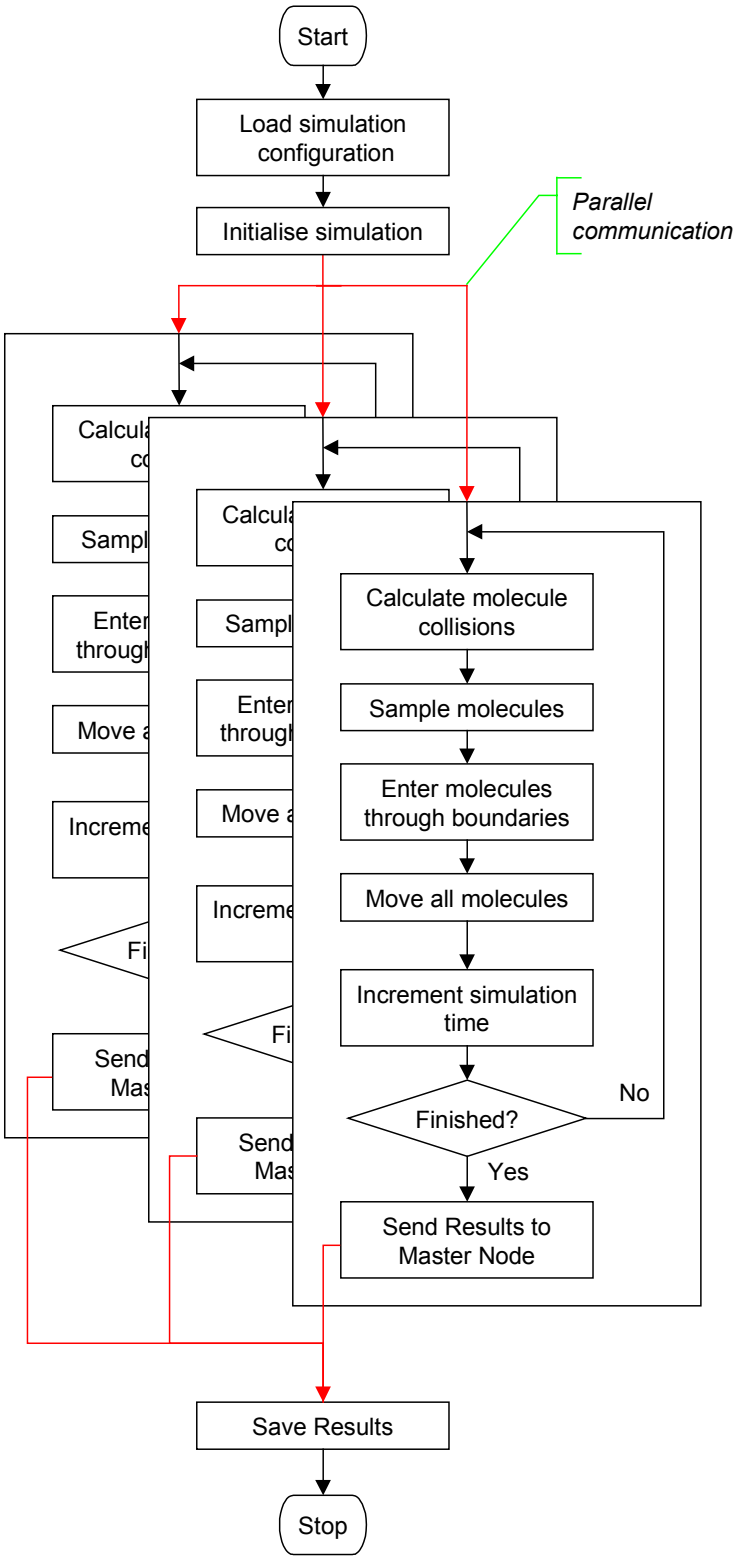


Figure 6.4 Parallel statically independent runs flow chart

By giving each node a full copy of the simulation domain, the computational burden and associated inefficiencies of keeping the load balanced across all nodes is removed. However, this method has the major limitation of only being able to solve problems that can be completely contained in an individual nodes (CPUs) memory. This is offset by

the significant increase in amount of memory available on engineering workstations and desktop computers relative to mid 1990s when distributed parallel processing solutions of the DSMC method were initially developed. It is now common to find that for most two-dimensional and some three-dimensional problems the CPU time required to solve the simulation becomes the limiting constraint as opposed to the available memory. Therefore, the PSIR method is becoming an option for more simulation problems.

6.4 Implementation

The parallelisation method implemented in this thesis is a combination of both the PDD and PSIR methods. Where possible, problems are solved using the PSIR method. However, if the problem is too large to fit into a single nodes memory, then the minimum number of nodes required to contain the solution is calculated, N_{min}. If the total number of nodes available to the program, N_{max}, is greater than twice N_{min} then the problem is solved using a combination of the PDD and PSIR methods.

For solutions that use the PDD parallelisation method, the user can optionally specify that after the steady flow state has been reached the ratio of real to simulation molecules, F_N, be recalculated. The value of F_N can be set so that each node has the maximum possible number of molecules or set to some user defined number of molecules.

For solutions that use the PSIR method, the independent node solutions are synchronised during the instationary phase. This is done so that the aggregate sample data can be used to calculate the solution adaption results and to ensure that a common grid is used by all nodes.

The parallel version of the program uses a Master/Slave programming model. The Master program preforms all of the solution adaption routines, calculates the sub-domains and assigns them to each slave. When the Slave programs are processing the sub-domain simulations, the Master monitors the progress and keeps the Slaves in time-synchronisation.

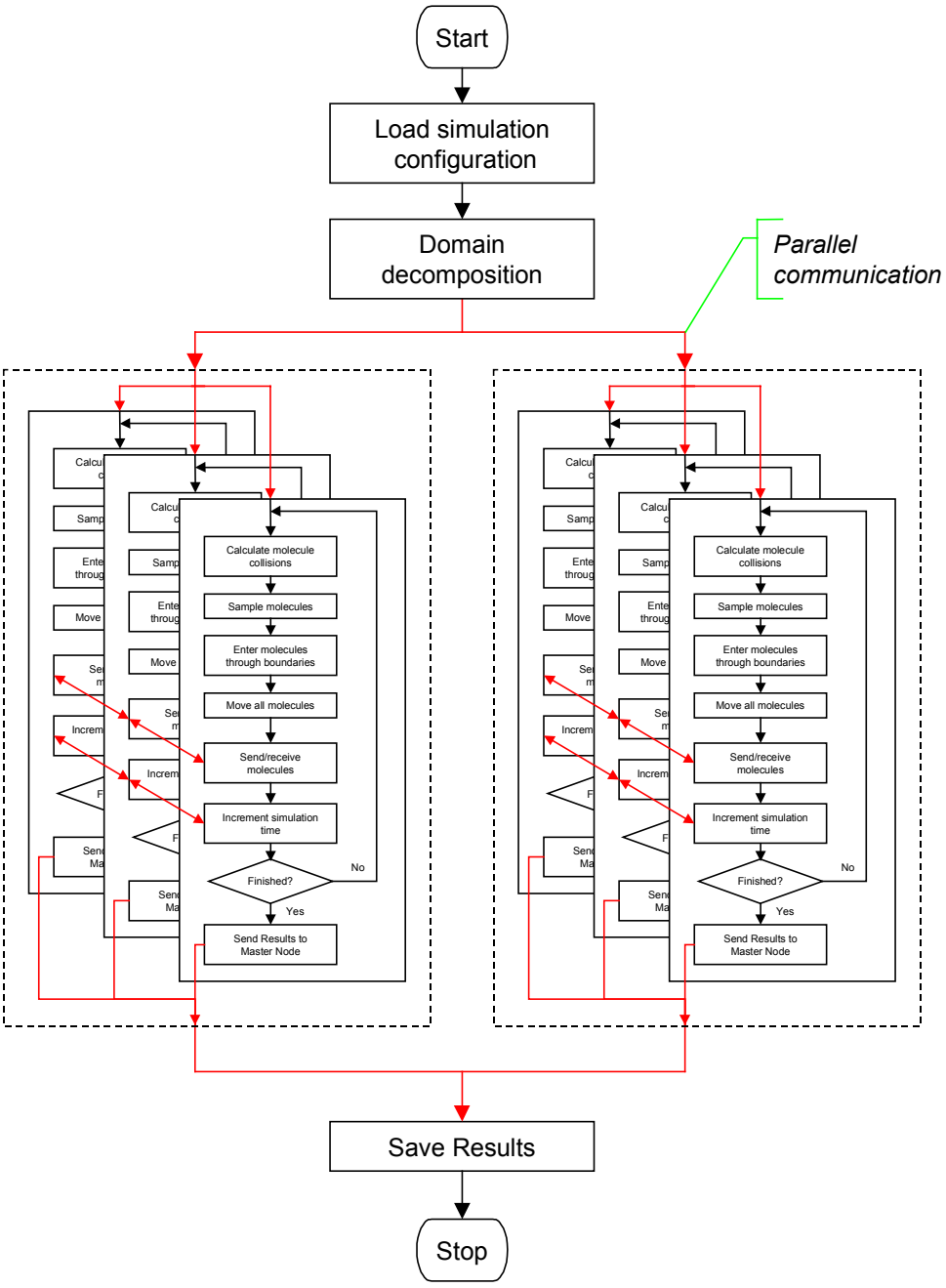


Figure 6.5 Hybrid parallel flow chart

6.4.1 Sub-domains

The global simulation is decomposed into sub-domains after each grid adaption step. The Master program keeps statistics on the relative solution time of each sub-domain from the previous period and uses this information, combined with the number of simulation molecules contained in each cell, to partition the global domain into approximately equal computational sub-domains. To minimise inter-domain communication, preference is given to subdividing the global domain into regions with

high aspect ratios aligned with the freestream direction. In addition, where feasible sub-domain boundaries are made coincident with object boundaries. As the parallelisation method implemented in this thesis uses the PSIR method where possible, this will result in a large ratio of molecule storage to grid storage. Therefore, each of the slave nodes was given a complete copy of the simulation grid. This minimises the need to generate a custom grid for each node and should have a minimal effect on the amount of memory available for the molecule storage.

6.4.2 Domain Decomposition

The domain decomposition algorithm implemented in this thesis used a variation of the chain partitioning method. The base algorithm was modified to take advantage of the multi-block nature of the grid to minimise the edge cut, and was intentionally keep simple to reduce the computation time and provide robust performance.

The computational cost of calculating the domain decomposition was small compared to the total simulation time, of the same order of magnitude as a single time step. Therefore, a new domain decomposition was calculated when the constrained SAR algorithm, Equation 6.2, estimated a net time saving and after each grid adaption. This resulted in an optimal domain decomposition for the current solution conditions. Additionally, the decomposition algorithm was formulated to minimise the change in cell allocation between successive iterations. This minimised the number of cells exchanged after each decomposition.

The domain decomposition algorithm is implemented using the following methodology,

1. Calculate the computational load to be assigned to each node.

   The computational load of the simulation is measured using the sum of the number of molecules. For the first domain decomposition, each node is assigned an equal number of molecules. For subsequent domain decompositions, the number of molecules assigned to each node is adjusted based upon the node execution time relative to the average node execution time for the last period. The number of molecules to be allocated to each node is given by the following equation,

$$N_{mi} = \frac{(1+\beta)\overline{T}_i - \beta T_i}{n\overline{T}_i} N_m \tag{6.3}$$

where,

T_i is the total execution time for node i for the last period,

$\overline{T}_i$ is the average total execution time for all nodes,

β is a relaxation constant

n number of solution nodes, and

N_m is the total number of simulation molecules.

The relaxation constant, β, was added to the equation to stop oscillations in decomposition distribution. A value of $\beta = 0.67$ was found to give stable results, and was used for all simulations presented in this thesis.

2. Subdivide the simulation domain.

   To minimise the change in decomposition between iterations, the nodes and grid blocks were processed in the same order at each iteration. If a grid block was split since the last iteration, its place in the processing was arranged to maintain the relative order of the grid blocks.

   Using the number of molecules to be allocated to a node that was calculated in the previous step, the algorithm processes each grid block in order. If the total number of molecules in the block was less than the remaining molecule allocation for the node, the complete block was assigned to the node. If the number of molecules in the block was greater than the node's remaining molecule allocation, then each cell was stepped through until the remaining molecule allocation was fulfilled. The cells in the grid block were processed by row or column, depending on which was more aligned with the primary flow direction. This was done to minimise the edge cut of the decomposition.

3. Exchange of molecules and cell data.

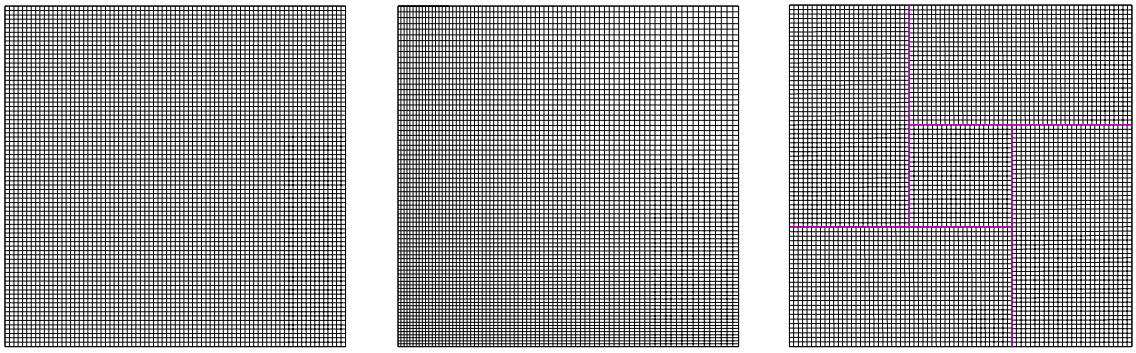
   Each node is given a full copy of the domain decomposition. By comparing the new and the old decompositions, the node can calculate which cells, and associated molecules, are no longer in its sub-domain and which node to send them to. This allows the exchange of data to occur in a distributed manner.

If it is found that the calculated decomposition results in an exchange of less than 0.5% of the total number of cells, then the next decomposition calculation is delayed to minimise oscillations.

6.4.2.1 Efficiency Measurements

Several test cases were run to determine the efficiency of the domain decomposition algorithm. These tests used three different grids, Figure 6.6, and two different flow configurations.

The gas flow was molecular Nitrogen at a Mach number of 4, a temperature of 300K and a number density of 10^{20} m^{-3}. These conditions correspond to a mean free path of 0.01294m and a Knudsen number of 0.0143. For the first flow condition the stream direction was parallel to the *x*-axis. For the second condition the stream direction was inclined at 30° to the *x*-axis. For each test case the simulation was run with 4 solution nodes, for a total of 5000 time steps with all solution adaption functionality disabled.



a. Uniform b. Linear Varying c. Multi-block uniform

Figure 6.6 Domain decomposition test grids

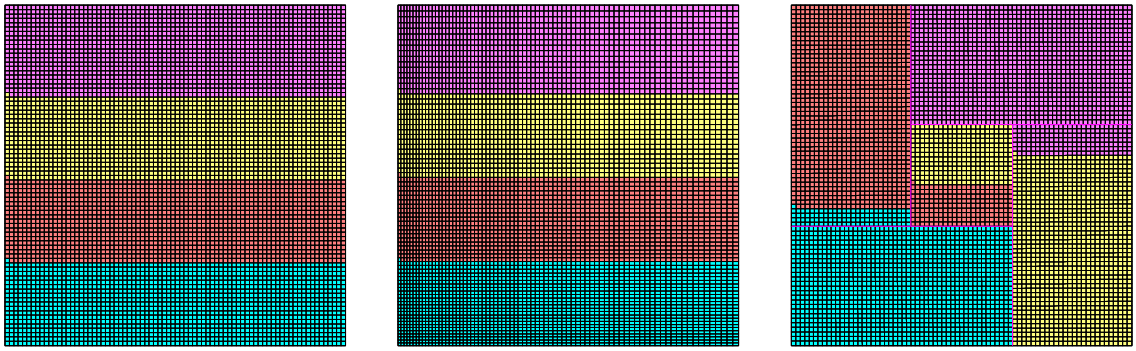
The results presented in Table 6.1 show that there was only a variation of 3.8% in the solution time between all six test cases. Figure 6.7 shows the final grid decomposition for the aligned flow direction tests. As there was no significant change in the domain decomposition between the aligned and inclined flow direction tests the final grid decompositions are not presented. Changing to a multi-block resulted in a 1.5 – 2% increase in solution time. The majority of this time increase was due to initial load imbalance which is shown by the increased number of domain decompositions. However, once the stationary flow phase has been reached there should be no significant difference between a single block and multi-block solution time. It was interesting to note that both of the linear grid tests had a shorter solution time than the

corresponding uniform grid tests. This was primarily due to the linear grid tests performing their domain decomposition earlier than the uniform grid tests. Additionally, the linear grid test with the aligned flow direction produced a marginally better domain decomposition than the aligned uniform grid test.

Overall, these test results show that the domain decomposition algorithm is able to produce an efficient decomposition for both non-uniform and multi-block grids.

Grid	Flow direction	Relative solution time	No. of domain decompositions
Uniform	0°	100.0%	1
Uniform	30°	100.8%	1
Linear	0°	99.0%	1
Linear	30°	100.6%	1
Multi-block	0°	101.5%	3
Multi-block	30°	102.8%	4

Table 6.1 Domain decomposition test results



a. Uniform b. Linear Varying c. Multi-block uniform

Figure 6.7 Domain decomposition test results

6.4.3 Solution Adaption

The solution adaption algorithms developed in the previous chapter were incorporated into the parallel solution implementation. As with the other parallel functionality, the Master node was responsible for coordinating the solution adaption algorithms. The method of integrating the grid and time step algorithms was different and is described in detail below.

6.4.3.1 Grid Adaption

The grid adaption algorithm was implemented in a serial manner with the Master node performing all of the adaption calculations. This method of implementation was chosen as the Master node already had a copy of each nodes' sample results and therefore could generate the adaption distribution function. In addition, the adaption algorithm needs to know the adapted state of the other grid blocks during the adaption process. This is required to maintain consistent internal boundaries and to allow the joining internal nodes that are closely located. The Master node aggregates the Slave nodes sample data and uses this data with the serial adaption algorithm. After the adapted grid has been generated, a new domain decomposition was calculated. This domain decomposition and the new grid were then sent to all nodes. Each node then recalculated the computational position of all molecules that it held from the previous solution period. Any molecule that was not located in the node's new domain decomposition was transferred to its new host node.

6.4.3.2 Time Step Adaption

The time step adaption algorithm was implemented in both a parallel and serial manner. When the PSIR parallelisation method is used in the solution process, the Master node calculated the time step adaption in serial. This was done as only the Master node had a copy of all PSIRs sample data required to calculate the adaption. While it would have been possible for each PSIR to calculate its own time step adaption distribution, this would have resulted in each PSIR having a different distribution and would not have used all of the available sample data.

When only the PDD parallelisation method is used in the solution process, the time step adaption algorithm uses a parallel implementation. This implementation follows the serial algorithm developed in Section 5.4.1, and adds two synchronisation points.

The procedure followed at each time step adaption is:

1. Each node calculates its desired new base time step using Equations 5.14 - 5.17.
2. All nodes send their desired base time step to the Master node. The Master node selects the smallest time step to be the new base time step for the solution and broadcasts this value to all nodes.

3. Each node uses the new base time step to calculate the new time step multiple for each cell in its domain using Equation 5.19. This information is then used to adjust all molecule time steps to account for changing cell time step using Equation 5.20.
4. All nodes then send back a flag indicating whether their time step adaption process had stabilised. The Master node aggregates these flags and broadcasts the result to all nodes.

This procedure was found to work efficiently, required minimal data transfer and only two synchronisation points.

6.4.4 Parallel Libraries

The most widely used method for implementing distributed computing is the message-passing paradigm. This paradigm allows the user to setup a virtual machine across a distributed set of computers. Communication of control signals and data is passed between nodes via messages over the connecting network infrastructure. This network infrastructure is most often a standard ethernet link but can be an inter-processor bus in the case of multi-processor computer. The message-passing library handles all the low-level communications for the user and provides a set of interface independent routines.

Two standard message-passing software libraries are currently in wide use, Parallel Virtual Machine (PVM) [22] and Message Passing Interface (MPI) [40]. There are a number of papers that publish results of comparisons between these two libraries, [16], [39] and [54]. All these comparisons agree that the relative performance of the two libraries is very similar. Furthermore, there are examples of DSMC implementations for both the libraries, PVM: [62], and MPI: [37] and [51].

The PVM library was used to provide the message-passing functionality in this thesis as it is the preferred parallel library of the University of Sydney Engineering Department. However, the program has been written so that the message-passing library function calls are wrapped in a non-specific interface to simplify the translation to a different library, if required in the future.

Chapter 7

Program Verification and Results

7.1 Introduction

This chapter presents the verification and results of the program implemented using the theory presented in the previous chapters. Section 7.2 presents the results for two simulations that were run to verify the correct physical modelling of the program. Section 7.3 presents results with the program running serially and demonstrates the utility of the solution adaption procedures. Finally, Section 7.4 presents results for the program running different parallelisation models.

7.2 Program Verification

This section presents the results of two simulations that were run to verify the correct physical modelling of the program. The first test was designed to verify the accuracy of the simulation collision rate. The second test demonstrated the overall accuracy of the program.

7.2.1 Collision Rate Test

The purpose of this test is to verify that the program correctly simulates the inter-molecule collision rates and total energy conservation. The test configuration is based on the collision test configuration presented by Bird [7] §11.2, with the main difference being the results presented below are for a two-dimensional simulation as opposed to a one-dimensional simulation used in [7].

The problem uses a homogeneous gas mixture composed of five different species. The molecular collisions were simulated using the variable hard sphere model with each gas assigned to different species group for the purpose of collision partner selection. Energy exchange between translational and rotational modes was calculated using the Larsen-Borgnakke model. A constant rotational relaxation collision number of 5 was

used for all species. All gas species had a reference temperature of 273K, a coefficient of viscosity of 0.75 and 5 degrees of freedom. The species fraction, molecular diameter and mass are listed in Table 7.1. The gas had a number density of 10^{20} m^{-3} at a temperature of 300K. These conditions correspond to a mean free path of 0.01817m for the gas mixture. The simulation region was a square with a side length equal to 10 mean free paths and a fixed cell size equal to half a mean free path. The ratio of real to simulated molecules was set so that there were on average 20 molecules per cell. All boundaries were modelled as planes of symmetry.

Species	Species fraction %	Molecule diameter d_{ref} (m)	Molecule mass m (kg)
1	0.60	3.5×10^{-10}	5.0×10^{-26}
2	0.20	4.0×10^{-10}	4.5×10^{-26}
3	0.10	3.0×10^{-10}	2.5×10^{-26}
4	0.08	3.0×10^{-10}	2.0×10^{-26}
5	0.02	4.0×10^{-10}	4.0×10^{-26}

Table 7.1 Collision rate test gas species molecular properties.

7.2.1.1 Theoretical Collision Rate

The theoretical collision rate between molecule species p and q, $\left(N_{pq}\right)_0$, is defined in [7] Equation 4.78, and is given by,

$$\left(N_{pq}\right)_0 = 2\pi^{\frac{1}{2}}\left(d_{ref}\right)_{pq}^2 n_p n_q \left[\frac{T}{\left(T_{ref}\right)_{pq}}\right]^{1-\omega_{pq}} \left[\frac{2k\left(T_{ref}\right)_{pq}}{m_r}\right]^{\frac{1}{2}} \tag{7.1}$$

where,

$\left(d_{ref}\right)_{pq}$ is the reference diameter for collisions between molecule species p and q, and is equal to average of each species reference diameters,

$\left(T_{ref}\right)_{pq}$ is the reference temperature for collisions between molecule species p and q, and is equal to average of each species reference temperature,

ω_{pq} is the temperature exponent of the coefficient of viscosity between molecule species p and q, and is equal to average of each species value,

m_r is the reduced mass of the collision pair and is given by, $m_r = \dfrac{m_p m_q}{m_p + m_q}$

7.2.1.2 Results

The results of the test are summarised in Table 7.2 as the ratio between the simulated collision rate and the theoretical rate. The average difference between the simulated collision rate and the theoretical rate was 0.14% and the maximum difference was 0.42%. The final total temperature of the simulation region was 300.423K, with the translational temperature being 299.815K and the rotational temperature 301.325K.

Species	1	2	3	4	5
1	1.00105	1.00065	1.00070	1.00084	1.00126
2	1.00065	1.00034	0.99943	0.99947	0.99868
3	1.00070	0.99943	1.00108	0.99949	0.99657
4	1.00084	0.99947	0.99949	0.99808	0.99653
5	1.00126	0.99868	0.99657	0.99653	0.99578

Table 7.2 Ratio of simulated collision rate to theoretical value.

These values are equivalent to those published in [7] and show that the DSMC implementation correctly simulates the collision rate and maintains the correct total system energy balance.

7.2.2 Supersonic Leading-edge Test

The supersonic leading-edge problem was chosen as the second validation test case as there are published results available, Bird [7] §14.3. Additionally, Bird notes that the supersonic leading-edge makes a good test case because "the shear stress is the same order as the pressure and … is particularly sensitive to any failure to meet the computational requirements of the DSMC method".

7.2.2.1 Test Configuration

The configuration of the problem is identical to [7] and is detailed below. The flow is comprised of Nitrogen at a Mach number of 4, a temperature of 300K and a number density of 10^{20} m^{-3}. These conditions correspond to a mean free path of 0.01294m and a Knudsen number of 0.0143. The plate is located on the lower *x*-boundary and is parallel to the free stream flow direction. The leading edge of the plate starts 5.4 mean free paths from the upstream boundary, x = 0.0699m, and extends to the downstream boundary at x = 1.0m. The plate has a fixed temperature of 500K and is modelled as a

diffuse surface with full temperature accommodation. The small segment of the *x*-boundary in front of the plate leading edge was modelled as a plane of symmetry, while all other boundaries were modelled as stream boundaries. The upper *y*-boundary was located at y = 0.6m.

7.2.2.2 Results

The results of the simulation are presented in the Figure 7.1 - Figure 7.6. Figure 7.1shows the results generated by this author overlaid upon the results presented in [7] Figure 14.7. This figure shows a good correspondence between the two sets of results and is indicative of the good correspondence between the other figures published in [7] §14.3.

The main difference evident in Figure 7.1 is the velocity slip at the surface of the plate is more pronounced in the current results. This is due to the smaller cell size at the plate surface and the use of an optimal time step for each cell. The smaller cell size stops the flow properties close to the surface being diffused by the flow properties above the thin slip region. Whereas the optimal cell time step reduces the probability of more energetic molecules moving through the cells near the surface without colliding. The difference in the velocity slip is less pronounced at the trailing edge of the plate due to the influence of the downstream boundary condition. There is also a minor difference in the thickness of the boundary layer which is again attributable to the use of an optimal cell time step as well as being influenced by the change in velocity slip profile.

Figure 7.4 shows that the velocity slip at the surface of the plate is confined to a thin region and has a Mach number less than 0.5. Figure 7.6 shows the short, sharp increase in skin friction coefficient near the leading edge and the decreasing value after the maximum. Also evident is the slight increase in skin friction coefficient near the trailing edge that is caused by the acceleration of the flow that is induced by the downstream boundary conditions.

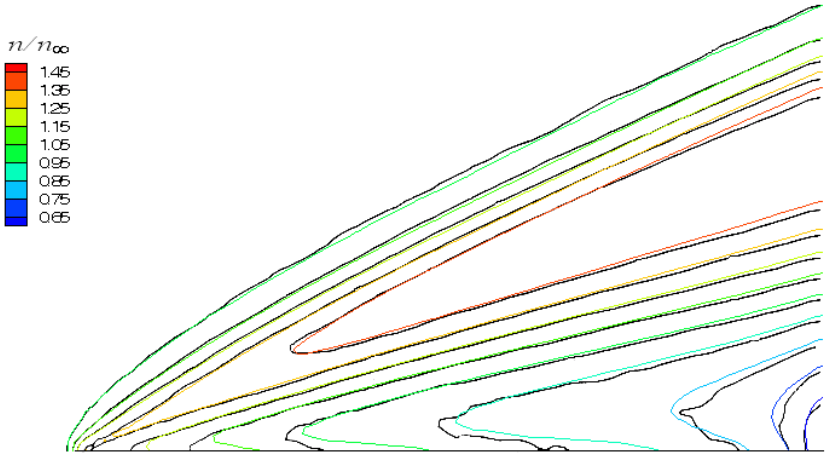


Figure 7.1 Supersonic leading-edge number density contours overlaying the results from [7] Figure 14.7, $Ma_\infty = 4.0$, $T_\infty = 300\text{K}$, $Kn = 0.0143$, $n = 10^{-20}\ \text{m}^3$.

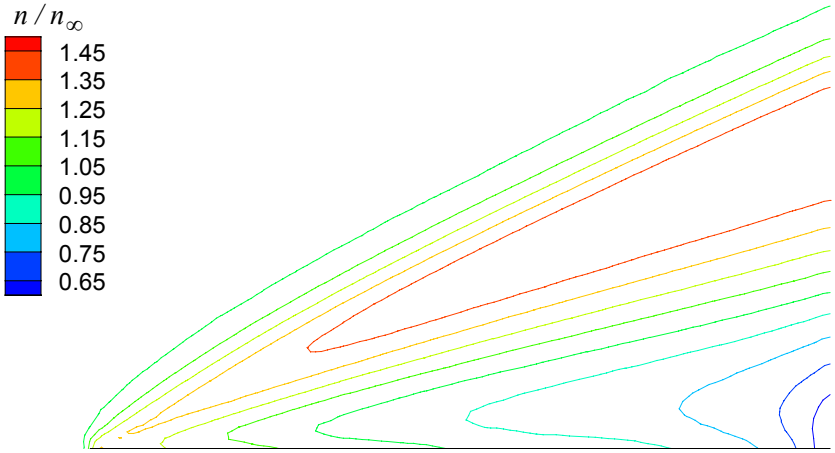


Figure 7.2 Supersonic leading-edge number density contours, $Ma_\infty = 4.0$, $T_\infty = 300\text{K}$, $Kn = 0.0143$, $n = 10^{-20}\ \text{m}^3$.

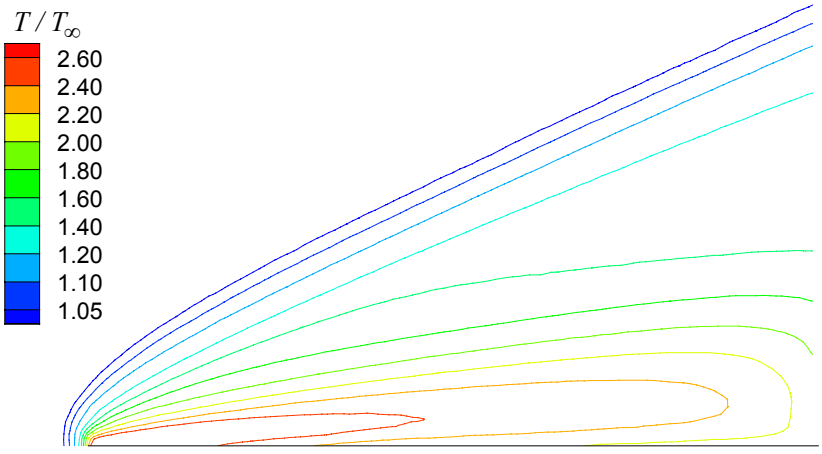


Figure 7.3 Supersonic leading-edge temperature number contours, $Ma_\infty = 4.0$, $T_\infty = 300\text{K}$, $Kn = 0.0143$, $n = 10^{-20}\ \text{m}^3$.

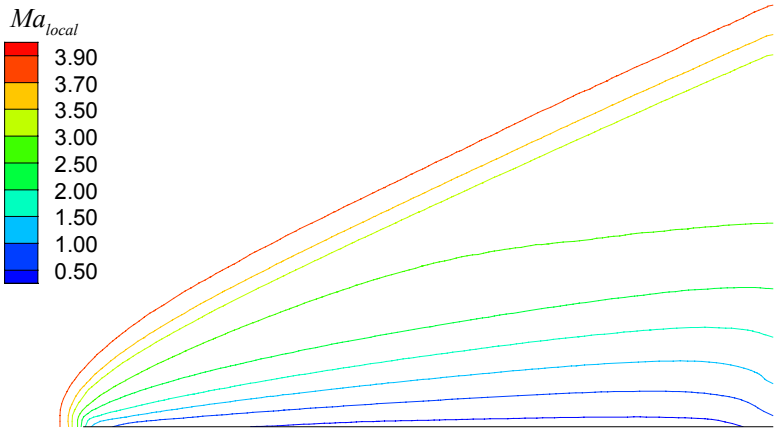


Figure 7.4 Supersonic leading-edge local Mach contours, $Ma_\infty = 4.0$, $T_\infty = 300\text{K}$, $Kn = 0.0143$, $n = 10^{-20}\ \text{m}^3$.

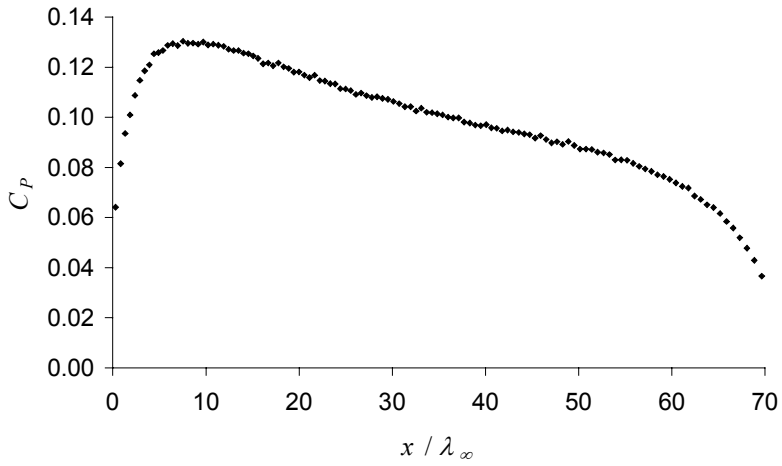


Figure 7.5 Supersonic leading-edge pressure coefficient, $Ma_\infty = 4.0$, $T_\infty = 300\text{K}$, $Kn = 0.0143$, $n = 10^{-20}\ \text{m}^3$.

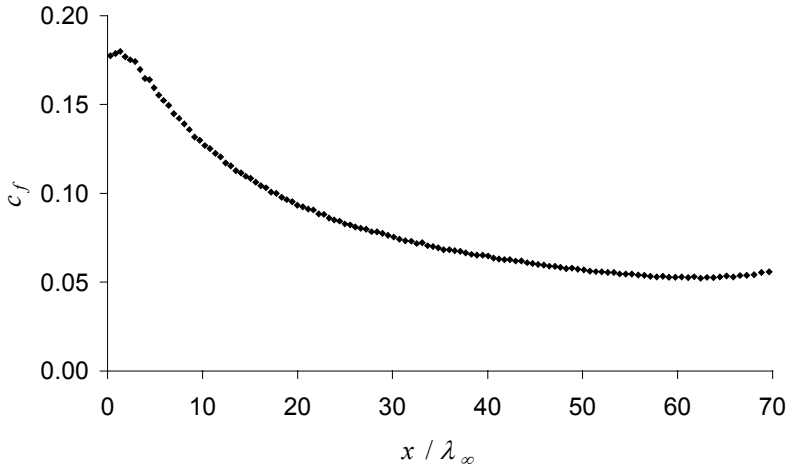


Figure 7.6 Supersonic leading-edge skin friction coefficient, $Ma_\infty = 4.0$, $T_\infty = 300\text{K}$, $Kn = 0.0143$, $n = 10^{-20}\ \text{m}^3$.

7.3 Serial Results

This section presents results of the DSMC implementation running serially and demonstrates the utility of the solution adaption procedures. The aim of the solution adaption procedures is to allow the user to enter the problem geometry and freestream flow conditions and have the DSMC implementation adapt the grid, time step and simulation number density to obtain an accurate result. To demonstrate the effectiveness of the solution adaption procedures the results of two simulations that use the same geometry and freestream flow properties, except the freestream Mach number and temperature, are presented. The solution adapted results of these two tests are compared against an unadapted solution.

7.3.1 Test Configuration

The two test configurations used are equivalent to the problems in [7] §14.5 and §14.6 respectively except that the location of the simulation boundaries relative to the plate have been extended. The primary aim of the test was to show that the solution adapted results kept the two main solution accuracy parameters, the ratio of average cell side length to local mean free path, $\bar{l}_m / \lambda$, and the ratio of maximum collision separation to local mean free path, $d_{c,\max}/\lambda$, to less than $1/\sqrt{2}$ and 1.0, respectively.

The problem geometry used for both tests was a vertical flat plate with a height of 0.15m. The plate was located 0.6m from the front boundary and bottom end was positioned on the lower boundary. The rear boundary was set at 1.2m from the front boundary and the upper boundary was set at 0.75m from the lower boundary. The lower boundary was modelled as a plane of symmetry, while all other boundaries were modelled as stream boundaries. The freestream flow was Argon with a number density of 10^{20} m^{-3}. These conditions correspond to a mean free path of 0.01294m and a Knudsen number of 0.043 (using twice the plate height due to the plane of symmetry).

For the subsonic tests, the freestream temperature was 300K and speed was $u_\infty = 172$ ms^{-1}, which corresponds to a Mach number of 0.53. Both surfaces of the plate were specularly reflecting. For the supersonic tests, the freestream temperature was 200K and speed was $u_\infty = 1317.3$ ms^{-1}, which corresponds to a Mach number of 5. The plate surfaces were diffusely reflecting with the front at the stream stagnation temperature of 1866.7K and the rear at the stream temperature of 200K.

For both of the test configurations a reference simulation was run using a static, unadapted grid with a uniform cell size equal to $\lambda_\infty/2$, Figure 7.7.

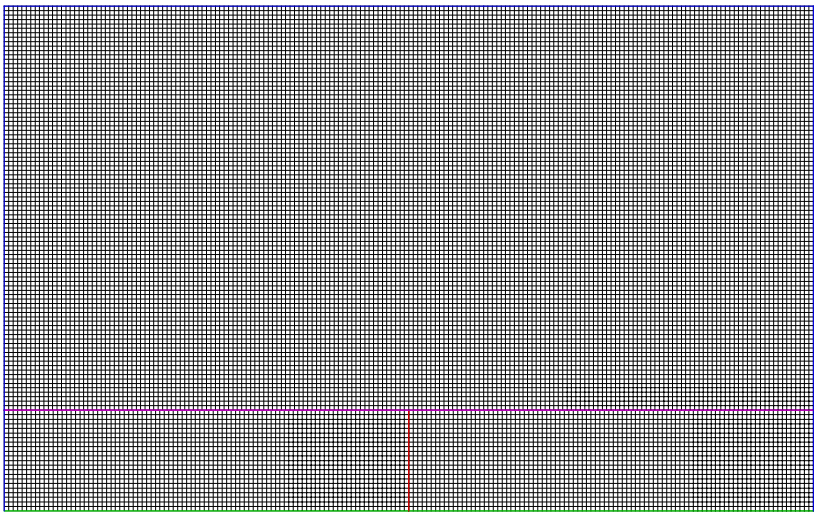


Figure 7.7 Vertical flat plate static, unadapted grid

7.3.2 Subsonic Test Results

The results of the solution adapted DSMC implementation are presented in Figure 7.8 through Figure 7.14. These results show good correspondence with streamline and Mach number results published in [7] §14.5. The main difference between the two sets of results is that the flow features downstream and above the vertical plate are elongated. This is due to the enlarged simulation domain used to generate the results presented here. The static, unadapted solution results for the accuracy parameters $\bar{l}_m/\lambda$ and $d_{c,\max}/\lambda$ are presented in Figure 7.15 and Figure 7.16 respectively.

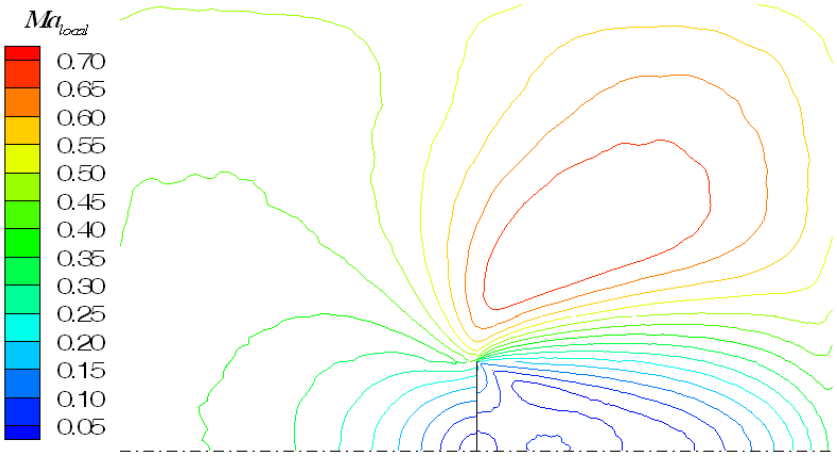


Figure 7.8 Local Mach number contours past a vertical flat plate, $Ma_\infty = 0.53$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\ \text{m}^3$.

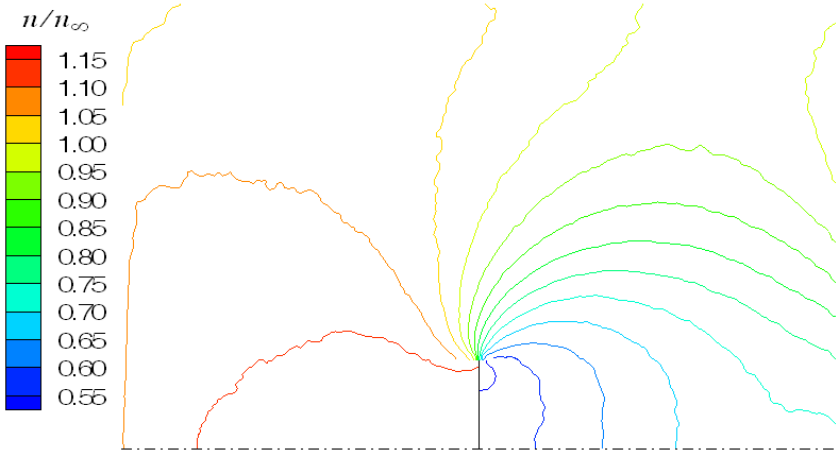


Figure 7.9 Number density contours past a vertical flat plate, $Ma_\infty = 0.53$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\,\text{m}^3$.

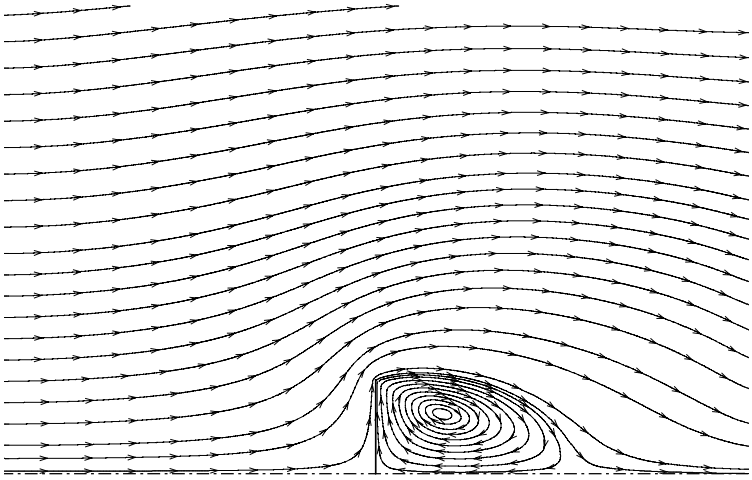


Figure 7.10 Streamlines past a vertical flat plate, $Ma_\infty = 0.53$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\,\text{m}^3$.

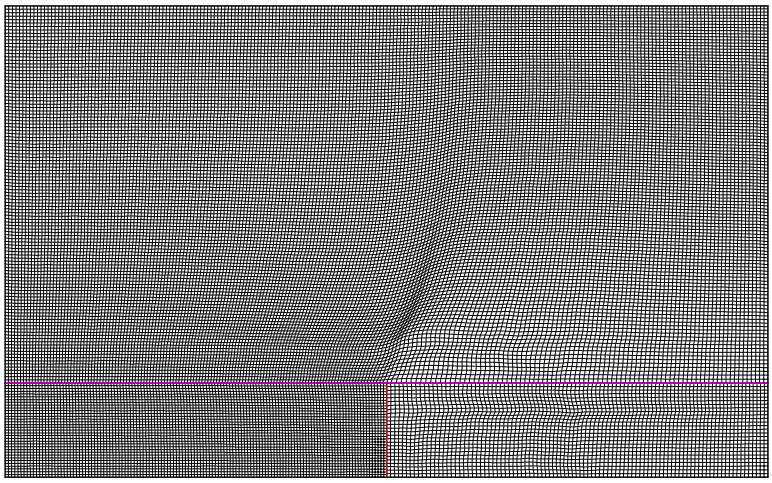


Figure 7.11 Adapted grid, $Ma_\infty = 0.53$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\,\text{m}^3$.

Figure 7.12 Adapted time step multiple, $Ma_\infty = 0.53$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\,\text{m}^3$.

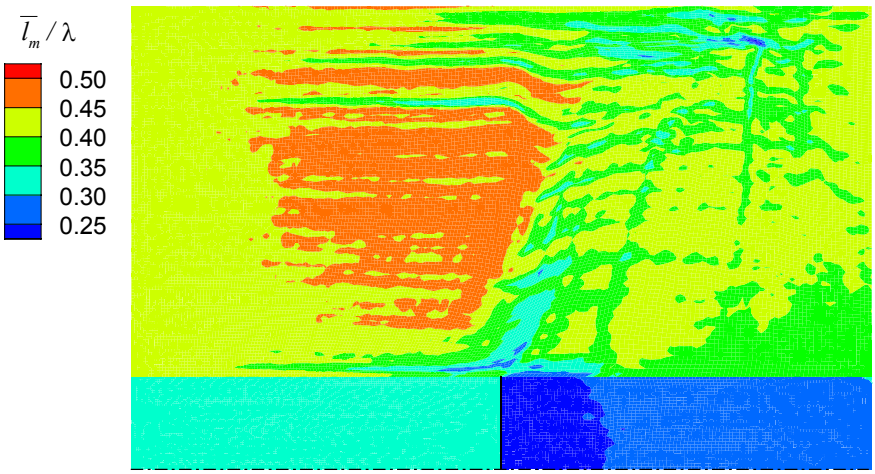


Figure 7.13 Ratio of average cell side length to local mean free path, $Ma_\infty = 0.53$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\,\text{m}^3$.

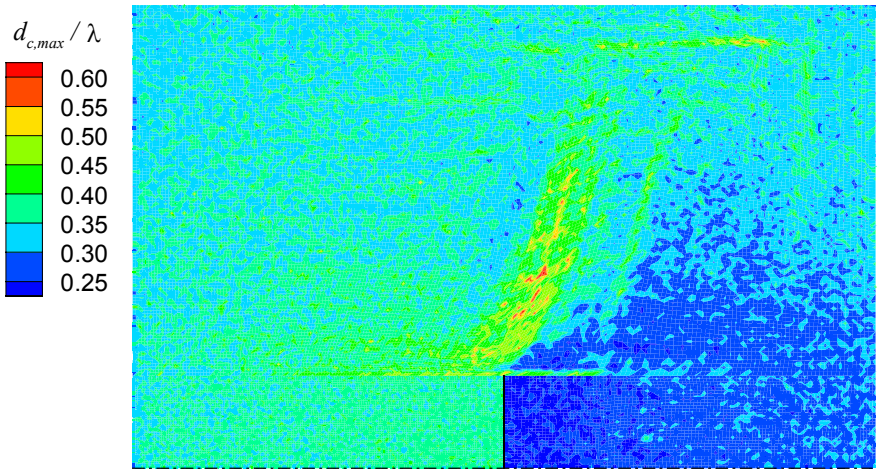


Figure 7.14 Ratio of maximum collision separation to local mean free path, $Ma_\infty = 0.53$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\,\text{m}^3$.

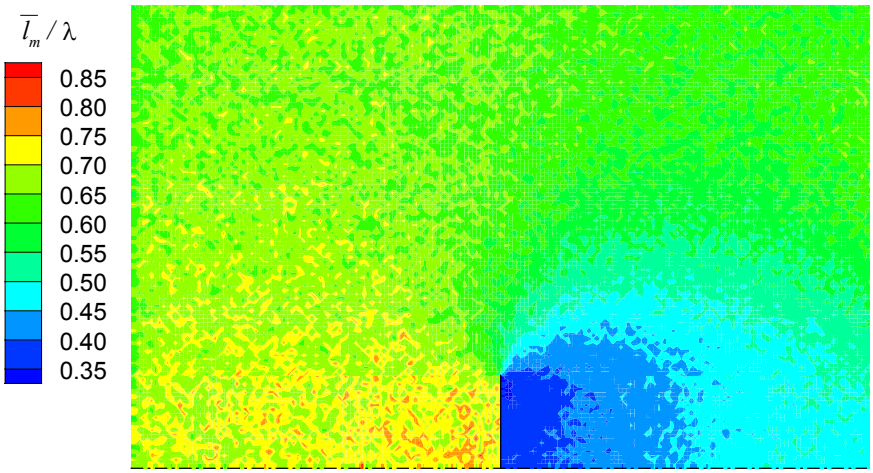


Figure 7.15 Static grid, ratio of average cell side length to local mean free path, $Ma_\infty = 0.53$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\ \text{m}^3$.

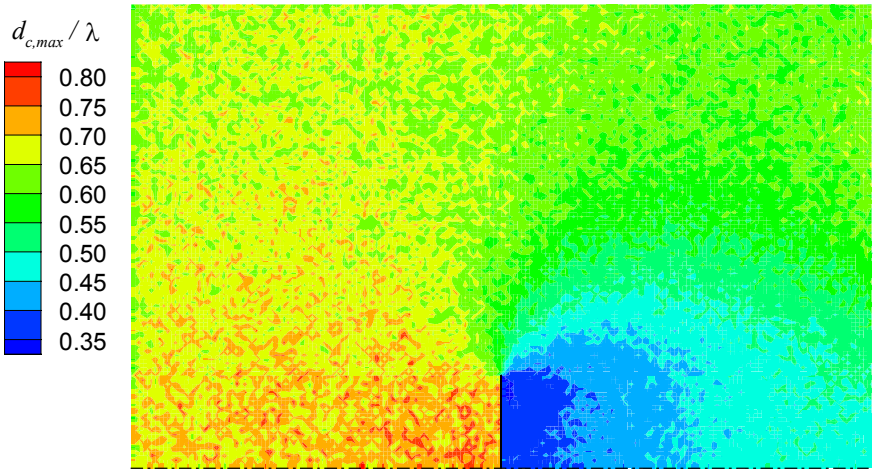


Figure 7.16 Static grid, ratio of maximum collision separation to local mean free path, $Ma_\infty = 0.53$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\ \text{m}^3$.

The final adapted grid, shown in Figure 7.11, was divided into 27350 cells and 3 blocks, which is the same number of blocks used to define the solution geometry. The final grid is relatively uniform and can be divided into two regions. The smallest of the regions is located downstream of the plate and the second region encompasses the remainder of the simulation domain. The main difference between the two regions is the relative cell size. The cells located in the first zone are approximately 1.5 times smaller than the cells located in the second zone.

Starting above the tip of the plate is a small region of relatively moderate grid distortion. This region corresponds to the interface between the two zones and is a common artefact of the grid adaption algorithm when there is a rapid change in grid density. Figure 7.8 and Figure 7.9 do not show any adverse effects on the solution

result caused by this region of gird distortion. As expected, Figure 7.13 shows that this region corresponds to an area of smaller than average $\bar{l}_m/\lambda$. Figure 7.14 shows that there are small localised areas of high $d_{c,\max}/\lambda$ located within the distortion region. The cause of these areas was investigated and it was found to be a result of a combination of small cell volume, cell skew and relatively low number sum. It was found that in very rare cases there would be a small number of molecules in the cell, and the distribution of molecules in the cell was such that they were grouped around opposite corners of the cell. This resulted in a collision partner being selected from across the diagonal length of the cell and thus an unusually large ratio of $d_{c,\max}/\lambda$. These occurrences were very rare, and did not have a measurable effect on the simulation results.

Figure 7.13 shows that the grid adaption algorithm was able to maintain the cell size closely to the defined limits of $0.25 \leq \bar{l}_m/\lambda \leq 0.45$. The maximum and minimum values of $\bar{l}_m/\lambda$ in the grid were 0.48 and 0.19 respectively. The region behind the plate had the smallest value of $\bar{l}_m/\lambda$. This area corresponded to a region of higher than average molecule number sum and therefore had its cell size reduced using the procedure detailed in Section 5.3.1. Figure 7.12 shows that this area also corresponded to the only region with a time step multiple greater than 1. This is due to the relatively low average speed of the molecules in the region.

Comparing Figure 7.13 with Figure 7.15 and Figure 7.14 with Figure 7.16 shows that the grid adaption algorithm has reduced the magnitude of accuracy parameters $\bar{l}_m/\lambda$ and $d_{c,\max}/\lambda$ relative to the static, unadapted solution.

7.3.3 Supersonic Test Results

The results of the solution adapted DSMC implementation are presented in Figure 7.17 through Figure 7.24. These results show good correspondence with those published in [7] §14.6, except as noted for the subsonic test where the enlarged simulation domain has extended the flow features. The static, unadapted solution results for the accuracy parameters $\bar{l}_m/\lambda$ and $d_{c,\max}/\lambda$ are presented in Figure 7.25 and Figure 7.26 respectively.

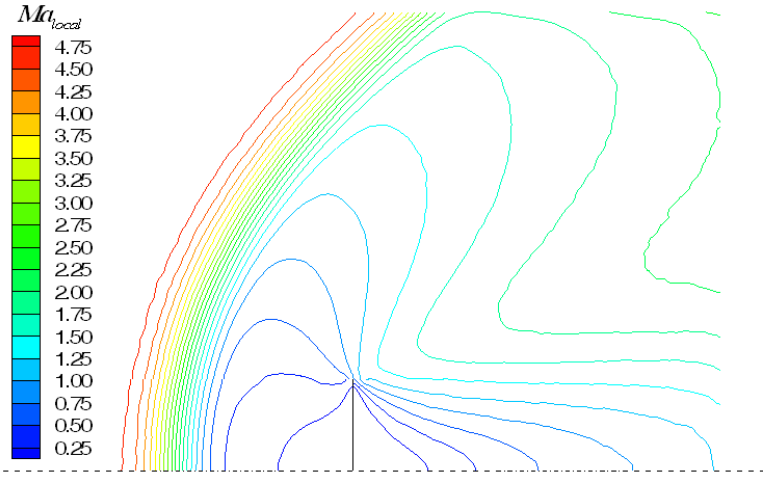


Figure 7.17 Local Mach number contours past a vertical flat plate, $Ma_\infty = 5.0$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\ \text{m}^3$.

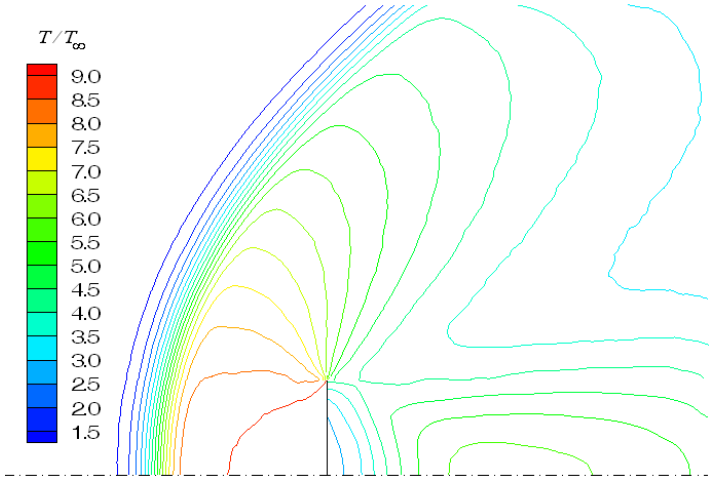


Figure 7.18 Temperature contours past a vertical flat plate, $Ma_\infty = 5.0$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\ \text{m}^3$.

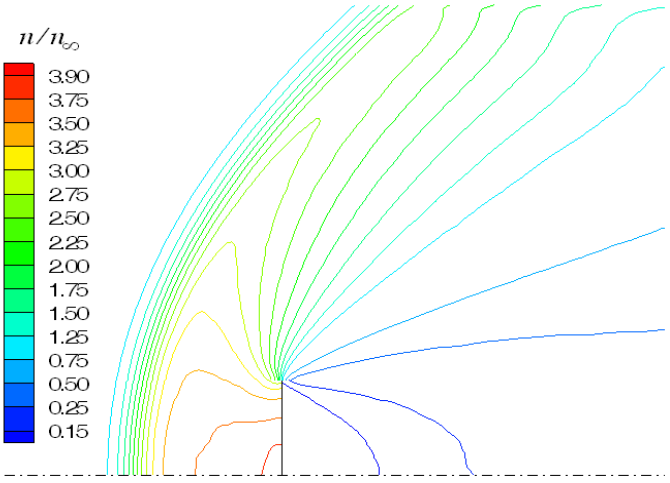


Figure 7.19 Number density contours past a vertical flat plate, $Ma_\infty = 5.0$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\ \text{m}^3$.

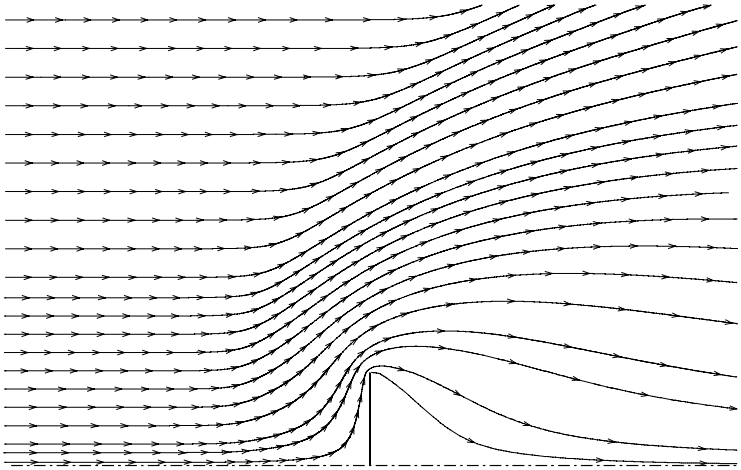


Figure 7.20 Streamlines past a vertical flat plate, $Ma_\infty = 5.0$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\,\text{m}^3$.

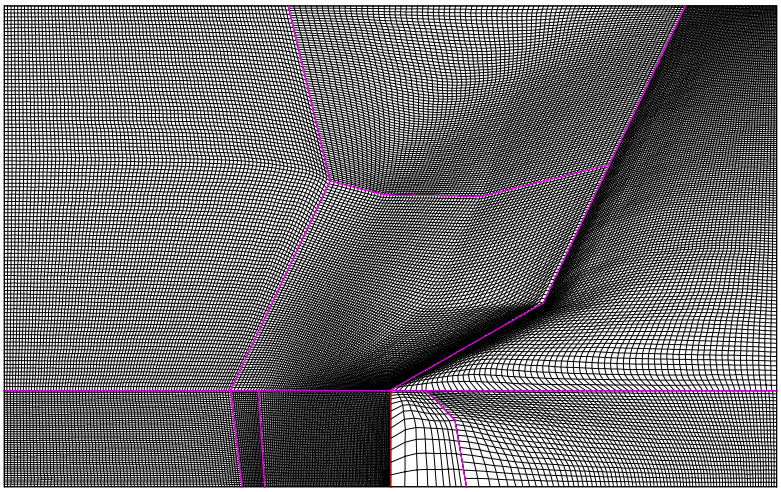


Figure 7.21 Adapted grid, $Ma_\infty = 5.0$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\,\text{m}^3$.

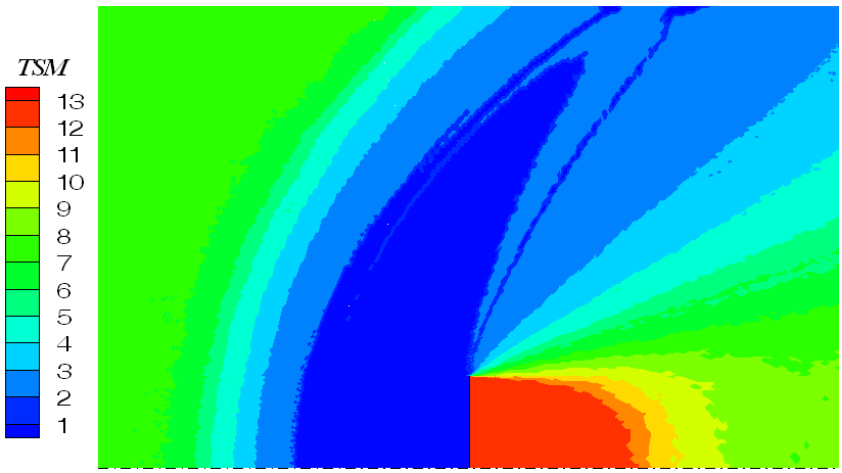


Figure 7.22 Adapted time step multiple, $Ma_\infty = 5.0$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\,\text{m}^3$.

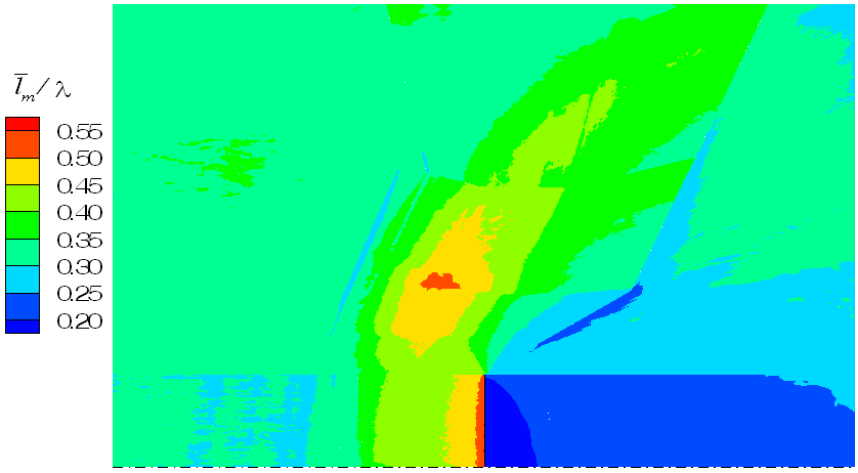


Figure 7.23 Ratio of average cell side length to local mean free path, $Ma_\infty = 5.0$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\ \text{m}^3$.

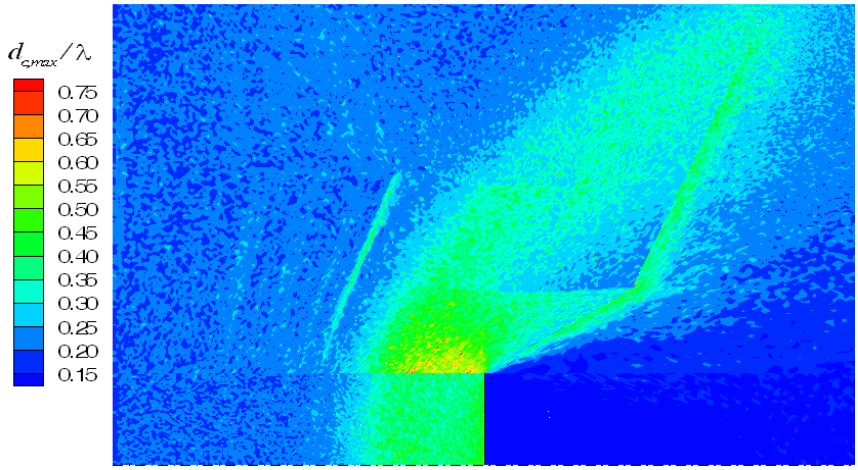


Figure 7.24 Ratio of maximum collision separation to local mean free path, $Ma_\infty = 5.0$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\ \text{m}^3$.

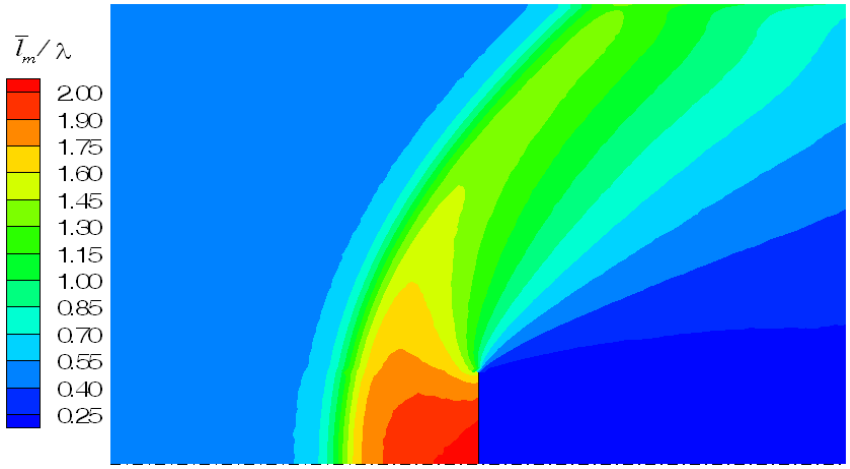


Figure 7.25 Static grid, ratio of average cell side length to local mean free path, $Ma_\infty = 5.0$, $T_\infty = 300\text{K}$, $Kn = 0.043$, $n = 10^{-20}\ \text{m}^3$.

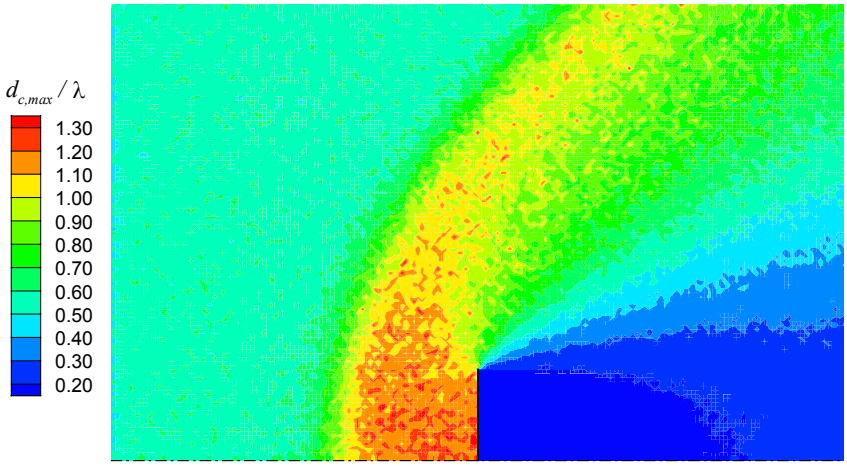


Figure 7.26 Static grid, ratio of maximum collision separation to local mean free path, $Ma_\infty = 5.0$, $T_\infty = 300$K, $Kn = 0.043$, $n = 10^{-20}$ m^3.

Figure 7.19 shows that there is a number density ratio of 26 across the simulation domain for this test configuration. This large number density ratio directly correlates to a large λ ratio through Equation 5.5. Figure 7.23 shows that the grid adaption algorithm was able to keep the $\bar{l}_m/\lambda$ ratio close to the defined limits of $0.25 \leq \bar{l}_m/\lambda \leq 0.45$. There are two small areas where the final grid is greater the desired value of $\bar{l}_m/\lambda \leq 0.5$, and the maximum value of $\bar{l}_m/\lambda$ within the adapted grid was 0.56. Figure 7.24 shows that these areas do not have an adverse effect on the $d_{c,\max}/\lambda$ values and overall $d_{c,\max}/\lambda$ was less 0.5 for the majority of the simulation domain. Similar to the subsonic test case there are small localised areas of high $d_{c,\max}/\lambda$ which are again due to a combination of small cell volume, cell skew and relatively low number sum.

The effectiveness of the grid adaption is evident by comparing the plots of $\bar{l}_m/\lambda$, in Figure 7.23 and Figure 7.25. The maximum value of $\bar{l}_m/\lambda$ for the adapted and unadapted solutions is 0.56 and 2.0 respectively. For the unadapted solution the distribution of $\bar{l}_m/\lambda$ follows the number density contours, whereas, for the adapted solution the distribution of $\bar{l}_m/\lambda$ is more uniform. Figure 7.24 and Figure 7.26 show similar results for the distribution of $d_{c,\max}/\lambda$. The maximum value of the cell mean collision separation, $[d_{c,\text{ave}}/\lambda]_{\max}$, for the adapted and unadapted solutions is 0.78 and 1.34 respectively. This shows that both solutions are on average achieving physically accurate collision partner selection. However, the adapted solution produces a measurable reduction in the collision separation.

The final grid generated by the adaption algorithm was divided into 9 blocks with a total of 43950 cells, as shown in Figure 7.21. It can been seen that the cell density is concentrated in the region in front of the vertical plate and behind the shock. Similar to the subsonic test case there is a region of grid distortion starting at the tip of the plate and following the local density contours. There is also a thin region of smaller cells that is on the downwind side of the upper-right block boundary. This region can be seen Figure 7.23 as being on the lower limit of $\bar{l}_m/\lambda$ adaption criteria. This is an artefact of the grid adaption algorithm as the number cells in each grid block is calculated independently.

Figure 7.22 shows that 13 different time step multiples were used in the solution and the distribution follows the number density contours. If time step adaption was not used, then solution time would have been significantly increased as over half the simulation domain had a time step multiple greater than 5. Thus, the time step adaption functionality allows the grid adapted solution to be used efficiently.

7.3.4 Conclusions

The serial results have shown that the grid adaption algorithm is able to generate a grid that ensures that the majority of cells have a value of $\bar{l}_m/\lambda \leq 0.5$. Similarly, the adapted grid ensured that the maximum value of $d_{c,\max}/\lambda$ was less than unity over the entire simulation domain and that the mean collision separation was measurably reduced. Finally, the time step adaption functionality allowed the efficient use of the grid adapted solution by enabling the optimum time step to be used for different regions of the solution domain.

7.4 Parallel Results

This section presents results for the DSMC implementation running different parallelisation models. The aim of the tests is to determine the relative parallel efficiency of the different models and their range of applicability.

7.4.1 Cluster Description

The computers used to perform the parallel testing consisted of 16 Pentium4 1.6Ghz machines with 512Mb of memory. All of the computers were connected via 100Mbit

ethernet through a switching hub. For all of the tests, the supervisory *Master* program was run on a separate computer so that it did not influence the relative loading of the solution nodes. For tests involving 16 computers, another Pentium4 computer on the same network, but not part of the cluster, was used to host the *Master* program.

All of the computers were running Linux with kernel 2.4.18-3 and version 3.4.3 of the PVM library was used. The timing of the tests was performed internal to the *Master* program by making calls to the system real-time clock. No other programs, apart from the operating system, were active on any of the machines during the testing.

7.4.2 Test Configuration

The simulation configuration used for tests was the same as the configuration used in the subsonic vertical flat plate test (Section 7.3.2). This configuration was chosen as it had a moderate number density and streamline direction variation across the flow field.

To ensure that an accurate relative measure of parallel performance was obtained, the solution adaption routines were disabled and the simulation was started from a pre-generated state of equilibrium. At the start of each test, a fixed number of iterations were run to ensure that the simulation was in the stationary phase and then the solution process was timed until a set number of molecule samples were obtained.

7.4.3 Results

When run on a single node the solution of the test configuration took 10 hours and 55 minutes and used an average of 345000 molecules. This solution time was used as the reference point to calculate the parallel efficiency and speedup results. Parallel efficiency is defined as the ratio of the product of the solution time by the number of nodes (computers) used relative to the serial solution time.

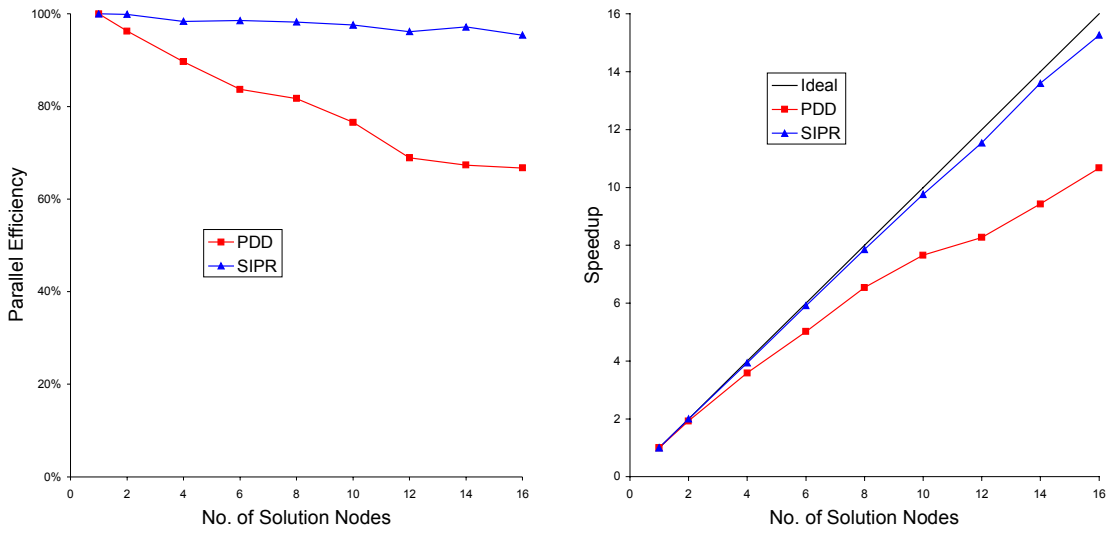
7.4.3.1 Relative performance of PDD and PSIR parallelisation methods

The first series of tests measured the relative performance of the physical domain decomposition (PDD) and parallel statically independent runs (PSIR) parallelisation methods and the results of these tests are presented in Table 7.3 and Figure 7.27. These results show that the efficiency of the PDD method decreases as the number of solution nodes increases. This is in line with other published results of this method, [19], [49],

[51], [57]. The parallel efficiency of this implementation is equivalent to the results in [49] but slightly less than those in [51].

	PDD		PSIR	
No. of Nodes	Efficiency	Speedup	Efficiency	Speedup
2	96.3%	1.93	99.9%	2.00
4	89.7%	3.59	98.4%	3.94
6	83.7%	5.02	98.6%	5.91
8	81.7%	6.54	98.2%	7.86
10	76.6%	7.66	97.6%	9.76
12	68.9%	8.27	96.2%	11.54
14	67.3%	9.43	97.2%	13.60
16	66.7%	10.67	95.4%	15.27

Table 7.3 PDD and PSIR parallelisation results



a. Parallel efficiency b. Parallel speedup

Figure 7.27 PDD and PSIR parallelisation results

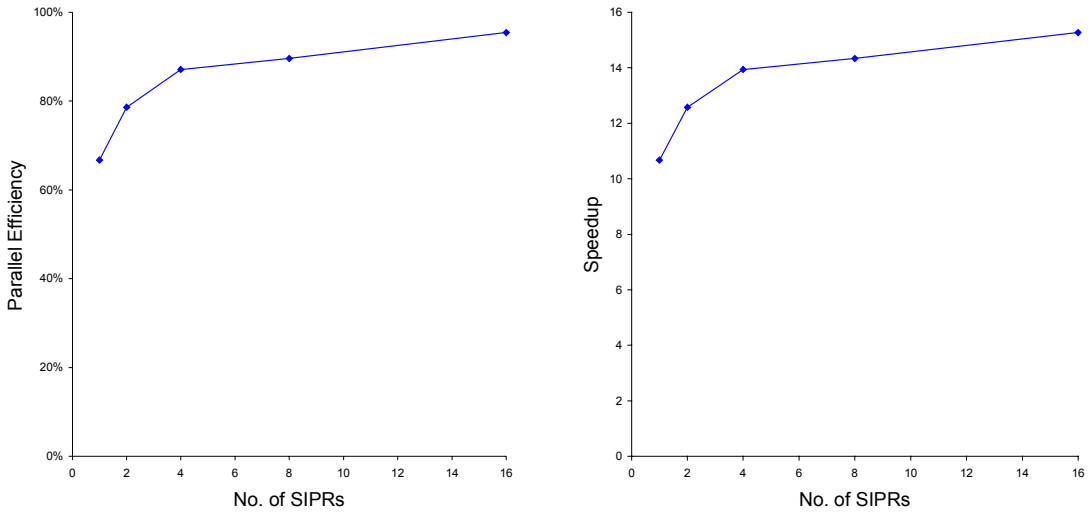
The PSIR results show that the decrease in performance as the number of solution nodes increases is minimal. This shows the advantage of the PSIR parallelisation method in that the number of synchronisation points is significantly reduced. Furthermore, as each node is solving exactly the same problem the solution is inherently load balanced. This implies that the time taken for each node to reach the synchronisation point should be equal and thus only a very small amount of the time will be spent waiting for other nodes.

7.4.3.2 Combined PDD and PSIR performance

A series of tests were run to evaluate the efficiency of using a combined PDD and PSIR parallelisation method. For these tests the number of nodes over which the domain was distributed and the number of PSIRs were varied to keep the total number of solution nodes equal to 16. The results presented in Table 7.4 and Figure 7.28 show that the parallel efficiency increases sharply as the number of PSIRs increase. Furthermore, Figure 7.29 shows that there is only a small efficiency penalty, approximately 4%, of converting a PDD only solution into a combined PDD and PSIR solution.

No. of Distributed Domains	No. of PSIRs	Efficiency	Speedup
16	1	66.7%	10.67
8	2	78.6%	12.57
4	4	87.1%	13.93
2	8	89.6%	14.33
1	16	95.4%	15.27

Table 7.4 Combined PDD and PSIR results



a. Parallel efficiency b. Parallel speedup

Figure 7.28 Combined PDD and PSIR results

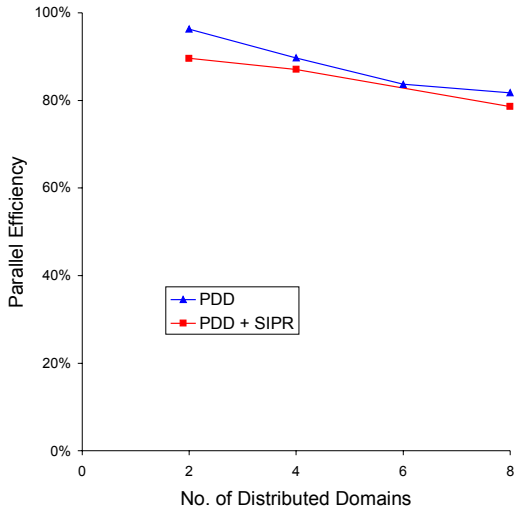


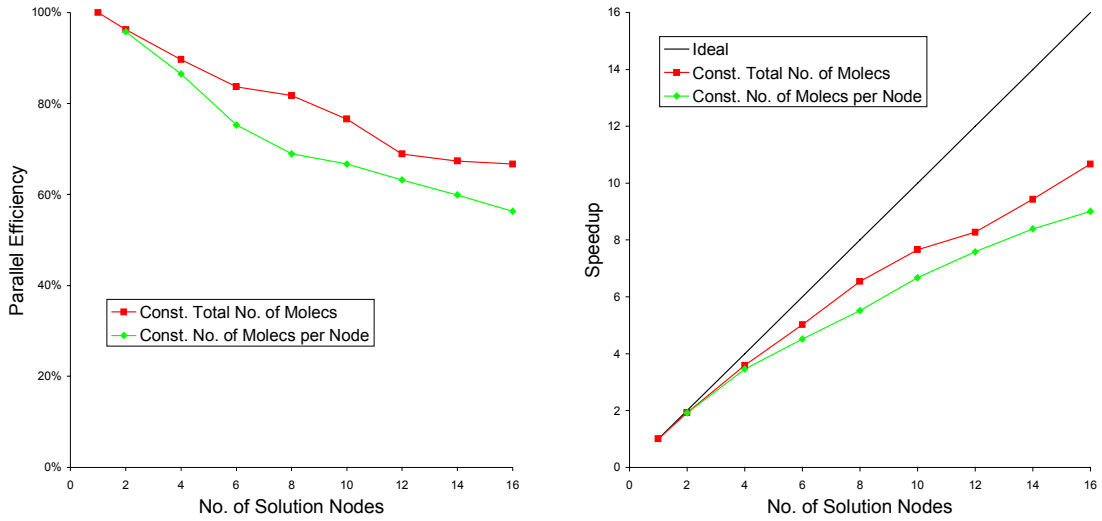
Figure 7.29 Performance of comdined PDD and PSIR method relative to PDD method

7.4.3.3 PDD performance with a constant number of molecules per node

The PDD test runs were repeated with the F_N of the solution adjusted so that a constant number of molecules per node was maintained. These results are summarised in Table 7.5 and are plotted in Figure 7.30. Comparing these results to the PDD results for a constant total number of molecules it can be seen that for more than two solution nodes, the constant number of molecules per node solution has a reduced efficiency.

No. of Nodes	Efficiency	Speedup
2	95.8%	1.92
4	86.5%	3.46
6	75.3%	4.52
8	69.0%	5.52
10	66.7%	6.67
12	63.2%	7.58
14	59.9%	8.39
16	56.3%	9.01

Table 7.5 PDD results with a constant number of molecules per node



a. Parallel efficiency b. Parallel speedup

Figure 7.30 PDD results with a constant number of molecules per node

This result is at variance with other published results for simular tests, [19] and [57]. Examination of the run logs revealed that the primary factor contributing to the decrease in parallel efficiency was load imbalance. One of the limitations of the current domain decomposition algorithm is that it measures cell work load by the number of molecules only and does not take into consideration variations in work load caused by differing collision rates. Furthermore, with the large number of molecules per cell, small imbalances in cell assignment are amplified. The effect of the domain decomposition algorithm limitation was demonstrated by the algorithm being run more often in the constant number of molecules per node simulations, but still resulting in a moderate load imbalance after decomposition.

Secondary effects that decreased the parallel efficiency were the increased amount of data passed between nodes each time step and communication delays. For the 16 node test each node was exchanging on average approximately 1% of its molecules to each of the nodes that it was connected too. As each node was connected to an average of two other nodes, each node was sending and receiving 2% of its molecules, ≈8000, each time step. This large amount of molecule exchange contributed to intermittent communication delays between nodes. These delays were a result of the inter-node communication hardware used by the computer cluster. The single hub that connects all nodes together can become a choke point under heavy inter-node communications. This problem is compounded by the fact that nodes that have a similar load balance would be

trying to communicate at the same time. The computers used in [19] and [57] were an Intel iPSC/860 hypercube and an IBM SP2 respectively. Both which are purpose built parallel processing machines with specialised high speed inter-node communication buses that reduce the effects of inter-node communication delays.

7.4.4 Summary

The parallel performance tests have demonstrated the effectiveness of the two parallelisation methods. The results for the PDD method using a constant total number of molecules were equivalent to previously published results. However, the results for the PDD method using a constant number of molecules per node showed that the current implementation does not perform as efficiently as other implementations when the number of molecules per cell is significantly increased. This lack of performance was attributed to a deficiency in the domain decomposition algorithm.

The results of the PSIR method showed that this method scaled well with an increasing number of solution nodes and maintained a parallel efficiency above 95% for all tests. Furthermore, the results showed that using a combined PDD and PSIR method was always more efficient than using a PDD with the same number of solution nodes. These results show that for the solution of large problems, maximum parallel efficiency will be obtained if the combined PDD and PSIR method is used. The solution nodes should be subdivided so that the minimum number of nodes are used in each PDD and the number of PSIRs is maximised.

Chapter 8

Conclusion

8.1 Introduction

The objective of this thesis was to develop a DSMC implementation that allows a non-expert user to efficiently solve arbitrary problems with the DSMC method in both serial and parallel environments. This chapter summarises the contributions of this thesis. Section 8.2 highlights the major theoretical and practical solutions it has offered. Section 8.3 suggests areas of future work in this field of research. Finally, Section 8.4 provides a brief summary of the thesis and concluding remarks.

8.2 Principle Contributions

The major contributions of this thesis arise from the formulation of different solution adaption and parallelisation algorithms. The integration of these algorithms into a single DSMC implementation results in a significant improvement in performance.

8.2.1 Efficient Molecule Movement

A novel molecule movement algorithm was developed which allows the efficient calculation of a molecule's trajectory through an arbitrary quadrilateral grid. This movement algorithm combined the simplicity and accuracy of calculating the molecule's movement in physical space with the simple cell indexing afforded by tracking the molecule's computational space position. The algorithm achieved this by performing the molecule movement in physical space and then transforming the molecule's final position into computational space. The computational space position was then used to determine the molecule's new cell index and whether any surface/boundary interactions took place over the molecule's trajectory.

8.2.2 Multi-block Grid Adaption

A multi-block grid adaption algorithm was developed that used the intermediate results of the solution, to calculate the number of cells and the distribution required ensuring that the grid meets the DSMC cell size requirements. The algorithm determined whether the distribution of cell size across a block would be more efficiently represented by splitting the block into two or more blocks and it creates the splits if required. Furthermore, the algorithm adjusted the ratio of real to simulation molecules to ensure that the number of molecules per cell is sufficient to ensure that the correct collision rate is maintained.

8.2.3 Time Step Adaption

A parallelised time step adaption algorithm was developed which allowed the time step of each cell to be set to a locally optimal value while still maintaining the ability to efficiently synchronise the solution process in a parallel implementation. Additionally, the algorithm was formulated such that the calculation of a new grid time step distribution was performed in parallel.

8.2.4 Parallelisation

Two different but complementary parallelisation methods were integrated. This integration of complementary parallelisation methods allows the use of the optimal parallel configuration for the available computational resources. Furthermore, the parallel implementation allows the parallel distribution to be dynamically changed to account for changes in the solution.

An improved implementation of the stop-at-rise algorithm was developed. This modified algorithm incorporates information relating to the total solution progress with the current solution performance to determine whether it is efficient to perform a domain decomposition repartitioning.

The results of the parallel tests showed that for the solution of large problems, maximum parallel efficiency would be obtained if the combined PDD and PSIR method is used. The solution nodes should be subdivided so that the minimum number of nodes are used in each PDD and the number of PSIRs is maximised.

8.3 Future Research

The following section outlines two areas where further work could make improvements to the implementation of the DSMC method.

8.3.1 Domain Boundary Adaption

The results presented in Section 7.3 show the placement of the domain boundaries can affect the results and efficiency of a simulation. For both the subsonic and supersonic vertical flat plate solutions, repositioning of the domain boundaries would have improved the solution. In the supersonic solution, there was an area in front of the domain where the flow properties were equivalent to the freestream conditions. The total solution time could have been reduced if the upwind boundary was moved closer to the leading edge of the shock. Similarly, the results for both the subsonic and supersonic solutions show that the location of the downstream and top boundaries had an effect on the final flowfield. Therefore, a solution adaption procedure that compared the flowfield properties around the domain boundary with the boundary conditions and adjusted the boundary, if required, would make an improvement to the solution process. Research needs to be undertaken into the optimum mix of boundary movement and boundary property adaption.

8.3.2 Domain Decomposition

As the size of DSMC problems increase and the number of nodes used in the parallel solution increases, the quality of the domain decomposition will become more critical. As discussed in Section 7.4, the domain decomposition algorithm could benefit from further research. Ideally the measurement of cell computational load should include the collision rate and where applicable the chemical reaction rate and surface interaction rate. Research would need to be conducted into developing a consistent measurement of cell load that can combine all of these factors but is relatively inexpensive to calculate.

Furthermore, research into the application of graph based partitioners to the DSMC method would be beneficial. This partitioning method offers the potential to decrease the inter-domain communication while maintaining an even distribution of computational load. However, a reliable and inexpensive method of calculating the

inter-cell communication load will need to be developed. Further research also needs to be conducted into mitigating the problem of non- singly connected sub-domains. Finally, parallel implementations of this method have been developed, [30], and would be advantageous for use in large parallel DSMC simulations.

8.3.3 Unsteady Flow Problems

The results presented in this thesis only deal with steady flow problems. As discussed in Section 5.5.3, unsteady flow problems are generally not suited to grid adaption and a method of implementing the time step adaption was developed. With respect to parallelisation of the DSMC solution process, the PSIR method would be well suited to unsteady flow problems. However, for problems that require a combined PDD and PSIR parallelisation, research needs to be performed into the optimal implementation of the domain decomposition strategy.

8.4 Summary

This thesis has made a significant contribution to the realisation of the DSMC implementation that can be used as an engineering tool. It is through the combination of solution adaptive and parallelisation techniques the DSMC method will be able to transition from a research tool to an engineering tool.

Appendix A

Simulation Configuration Files

This Appendix contains the listing of the parameter configuration file and geometry definition file used to run the solution presented in Section 7.3.2.

A.1 Parameter Configuration File

```
ProgVersion :                   3.5042

SteadyFlowProb :                1       # 0 - unsteady flow, 1 - steady flow
BTS_Factor :                    1.0     # Factor of estimated base time step to use for starting BTS,
                                          make smaller for unsteady problems with localised high
                                          density flow
SimMaxBTSM :                    20      # maximum base time step multiple

NoTimeSamp :                    2       # No time steps between samples
NoSampCMP :                     10      # No samples between updates of Cell macro properties
MinSampleSize :                 10      # min No. of samples between time step adaptions & result
                                          file saves

UseGridAdaption :               1       #0 - No, 1 - Yes
NoGridSweep :                   2       # No. of i/j sweeps each grid adaption
UseGridSplit :                  1       # 0 - No, 1 - Yes

UseTSAdaption :                 1       # 0 - No, 1 - Yes
NoTSAdapt :                     250     # No samples between time step adaptions
NoGridAdapt :                   3       # No time step adaptions between grid adaptions
UseMaxMem :                     0       # 0 - No, 1 – Yes
NoMolecSampleStop :             75000 # Min No of steady state molecule samples before program
                                          termination

AcclStart :                     1       #  0 - No, 1 - Yes
GridScaleFact :                 2.0     # Scale factor to apply to starting grid size
MolecScaleFact :                2.0     # Scale factor to apply to starting number of molecules

RatioCellLen2LMFP_Min :         0.25    # minimum number of cells per LMFP
RatioCellLen2LMFP_Max :         0.45    # maximum number of cells per LMFP

RatioMolecs2Cells_Min :         10.0    # minimum number of molecs per cell
RatioMolecs2Cells_Max :         30.0    # desired maximum number of molecs per cell

MaxNoMolecs :                   1500000 # maximum allowed number of molecs in simulation
```

```
StreamVelX :        172.0   # Uniform stream X velocity component (m/s)
StreamVelY :        0.0     # Uniform stream Y velocity component (m/s)
StreamTemp :        300.0   # Stream temperature (K)
StreamNoDens :      1.0e20  # Stream number density

InitVolState :      1   #  0 - vacuum or 1 - uniform stream

UseRotEng :         0   #  0 - No, 1 - Yes; Calculate transfer of rotational energy?
UseVSS :            0   #  0 - No, 1 - Yes

NoSpecGroups :      1
NoSpecs :           1
GasType1 :          Argon
SpecGroup1 :        1
StreamSpecFrac1 :   1
SurfReflnModel :    1    # specular
RotEngAccomCoef :   -1
NormEngAccomCoef :  -1
TangMomAccomCoef :  -1
RDoFRelax1 :        0
RDoFSpec1 :         0
RotRelaxConst1 :    0.0
RotRelaxTemp1 :     0
RotRelaxTemp21 :    0

NoJets :            0
```

A.2 Geometry Definition File

```
#include <math.h>
#include <iostream>
#include <fstream>
#include <strstream>
#include <sstream>
#include <string>
#include <cmath>

#include "mathdefs.hpp"
#include "geomgridgen.hpp"
#include "geom_cgmwrite.hpp"
#include "dsmc-inc.hpp"
#include "initdata.hpp"

string ProgVersion;

int main(void) {
  int face, edge, node, cnt;
  char *str;
  string PV;
  GRIDEDGE* pGE;
```

```
  Err_os_control Err_os_ctrl;

// ###################################
  string BaseFileName = "vfp_Ar";
  double LMFP = 0.01294;       // 1/(No. cells per unit length)
  float SurfTempFront = -1.0;  // stream temp
  float SurfTempBack = -1.0;   // stream temp
  float L = 0.15;              // characteristic length
// ###################################

  PLANEORIENT PO(ZDIRN, 0.0);
  GEOMGRID_2D GG2D, GG2DC;
  GeomCGMWrite GCW;

  float EdgeFact[S2D_NOSIDES];
  EDGETYPES EdgeType[S2D_NOSIDES];

  GG2D.Init(PO, XDIRN);

  ofstream OutFile;
  ifstream InFile;
  string FileName;

  UNION_16BYTES MiscData;
  DefaultValue(MiscData);

  ProgVersion = CodeVersion + '.' + SaveVersion + GridVersion + CfgVersion;

  cout << "\nDSMC2G Ver. " << ProgVersion << " : Setting up geometry definition file for \""
       << BaseFileName << "\""  << "\nCreating nodes ... " << flush;

  // create nodes used in defition of problem
  Vector<POINT2> vP(8);

  vP(0).X = 0.0;
  vP(0).Y = 0.0;

  vP(1).X = 4.0*L;
  vP(1).Y = 0.0;

  vP(2).X = 8.0*L;
  vP(2).Y = 0.0;

  vP(3).X = 8.0*L;
  vP(3).Y = 1.0*L;

  vP(4).X = 8.0*L;
  vP(4).Y = 5.0*L;

  vP(5).X = 0.0;
  vP(5).Y = 5.0*L;

  vP(6).X = 0.0;
  vP(6).Y = 1.0*L;

  vP(7).X = 4.0*L;
  vP(7).Y = 1.0*L;

  // add nodes to geometry model
```

```
node = GG2D.GD_2D.AddNode(vP);

cout << "Finished\nAssembling Geometry ... " << flush;

// join nodes with geometry segments to define edges

// bottom external edge
pGE = GG2D.GD_2D.AddEdge(XDIRN, edge); // edge = 0
pGE->SetOrigin(node);
pGE->AppendLineSeg(node+1, BT_SYMMETRY);
pGE->AppendLineSeg(node+2, BT_SYMMETRY);

// right external edge
pGE = GG2D.GD_2D.AddEdge(YDIRN, edge); // edge = 1
pGE->SetOrigin(node+2);
pGE->AppendLineSeg(node+3, BT_STREAM);
pGE->AppendLineSeg(node+4, BT_STREAM);

// top external edge
pGE = GG2D.GD_2D.AddEdge(XDIRN, edge); // edge = 2
pGE->SetOrigin(node+4);
pGE->AppendLineSeg(node+5, BT_STREAM);

// left external edge
pGE = GG2D.GD_2D.AddEdge(YDIRN, edge); // edge = 3
pGE->SetOrigin(node+5);
pGE->AppendLineSeg(node+6, BT_STREAM);
pGE->AppendLineSeg(node, BT_STREAM);

// define a vertical flat plate that intersects with bottom outer edge
// left side
pGE = GG2D.GD_2D.AddEdge(YDIRN, edge); // edge = 4
MiscData.f32[0] = SurfTempFront;      // surface temp
MiscData.f32[1] = 0.0;                // surface X velocity
MiscData.f32[2] = 0.0;                // surface Y velocity
MiscData.f32[3] = 0.0;                // surface Z velocity

pGE->SetOrigin(node+1);
pGE->AppendLineSeg(node+7, BT_OBJECT, &MiscData);

// right side
pGE = GG2D.GD_2D.AddEdge(YDIRN, edge); // edge = 5
MiscData.f32[0] = SurfTempBack;       // surface temp
MiscData.f32[1] = 0.0;                // surface X velocity
MiscData.f32[2] = 0.0;                // surface Y velocity
MiscData.f32[3] = 0.0;                // surface Z velocity

pGE->SetOrigin(node+7);
pGE->AppendLineSeg(node+1, BT_OBJECT, &MiscData);

// Add faces to define simulation area and any internal objects, see <geom-def.hpp>
// for definition
face = GG2D.GD_2D.AddFace(FT_EXTBDY, XDIRN, 4);

// NOTE: edges must be defined in a circular order
GG2D.GD_2D.Face(GD_External)->LinkEdge(S2D_BOTTOM, 0);
GG2D.GD_2D.Face(GD_External)->LinkEdge(S2D_RIGHT, 1);
GG2D.GD_2D.Face(GD_External)->LinkEdge(S2D_TOP, 2);
GG2D.GD_2D.Face(GD_External)->LinkEdge(S2D_LEFT, 3);
```

```
// define a vertical flat plate that intersects with bottom outer edge
face = GG2D.GD_2D.AddFace(FT_EDGEOBJ, YDIRN, 2);
GG2D.GD_2D.Face(face)->LinkEdge(S2D_BOTLEFT, 4);
GG2D.GD_2D.Face(face)->LinkEdge(S2D_TOPRIGHT, 5);

cout << "Finished\nChecking Continuity ... " << flush;

if (GG2D.GD_2D.CheckContinuity()) {
  cerr << "\nGEOM_2D is not conituous!" << endl;
  return(2);
}

cout << "Finished\nDecomposing GeomGrid ... " << flush;

// create internal edges
// left internal edge
pGE = GG2D.GD_2D.AddEdge(XDIRN, edge);    // edge = 6
pGE->SetOrigin(node+7);
pGE->AppendLineSeg(node+6, BT_INTERNAL);

// right internal edge
pGE = GG2D.GD_2D.AddEdge(XDIRN, edge);    // edge = 7
pGE->SetOrigin(node+7);
pGE->AppendLineSeg(node+3, BT_INTERNAL);

// initialise HardRegions (global grid sub-divisions)
GG2D.CreateHardRegions(3);

// Bottom/left HardRegion
GG2D.LinkEdge(0, S2D_BOTTOM, 0, 0, 1);
GG2D.LinkEdge(0, S2D_RIGHT, 4);
GG2D.LinkEdge(0, S2D_TOP, 6);
GG2D.LinkEdge(0, S2D_LEFT, 3, 0, 6);

// Bottom/right HardRegion
GG2D.LinkEdge(1, S2D_BOTTOM, 0, 1, 2);
GG2D.LinkEdge(1, S2D_RIGHT, 1, 2, 3);
GG2D.LinkEdge(1, S2D_TOP, 7);
GG2D.LinkEdge(1, S2D_LEFT, 5);

// Top HardRegion
GG2D.LinkEdge(2, S2D_BOTTOM, 6);
GG2D.LinkEdge(2, S2D_BOTTOM, 7);
GG2D.LinkEdge(2, S2D_RIGHT, 1, 3, 4);
GG2D.LinkEdge(2, S2D_TOP, 2);
GG2D.LinkEdge(2, S2D_LEFT, 3, 6, 5);

cout << "Finished\nChecking GEOMGRID Continuity ... " << flush;

if (GG2D.CheckContinuity()) {
  cerr << "FAILED\nGEOM_2D is not conituous!" << endl;
  return(3);
}

cout << "Finished\nCreating grid ... " << flush;

// Generate starting grid
GG2D.HardRegion(0).GenGrid(LMFP*0.5);
GG2D.HardRegion(1).GenGrid(LMFP*0.5);
GG2D.HardRegion(2).GenGrid(LMFP*0.5);
```

```
  GG2D.InitGrid();

  cout << "Finished\nSaving GeomGrid to file ... " << flush;

  FileName = BaseFileName+ OP_FileExt[CGM_FILE];
  GCW.Draw(&GG2D, BaseFileName.c_str());

  FileName = BaseFileName+ IP_FileExt[GEO_FILE];
  #ifdef BINARY_DATA_FILES
    OutFile.open(FileName.c_str(), ios::out | ios::trunc | ios::binary);
    OutFile.iword(BinaryFileIndex) = 1;  // turn binary file flag on
  #else
    OutFile.open(FileName.c_str(), ios::out | ios::trunc);
    OutFile.iword(BinaryFileIndex) = 0;  // turn binary file flag off
  #endif
  if (!OutFile) {
    cerr << "\nCould not open \"" << FileName.c_str() << "\" for writing!" << endl;
    return(1);
  }

  if (OutFile.iword(BinaryFileIndex) == 0) // default, text stream
    OutFile << ProgVersion << "\n";
  else {
    cnt = ProgVersion.length();
    OutFile.write((char*)&cnt, sizeof(cnt));
    OutFile.write((char*)ProgVersion.c_str(), ProgVersion.length());
  }

  GG2D.Write(OutFile);
  OutFile.close();

  GG2D.Delete();

  cout << "Finished\nExiting ... " << endl;

  return(0);
}
```

Bibliography

[1] Abe, T.; *Rarefied Gas Flow Analysis by Direct Simulation Monte Carlo in Body-Fitted Coordinate System*, Journal of Computational Physics, No. 83, 1989

[2] Alexander, F.J., Garcia, A.L., Alder, B.J., *A Consistent Boltzmann Algorithm*, Physical Review Letters, Vol. 74, No. 26, pp. 5212, 1995

[3] Alexander, F.J., Garcia, A.L., *The Direct Simulation Monte Carlo Method*, Computational Physics 11, pp. 588, 1997

[4] Alexander, F.J., Garcia, A.L., Alder, B.J., *Cell size dependence of transport coefficients in stochastic particle algorithms*, Physics of Fluids 10, pp. 1540, 1998

[5] Baganoff, D., *Vectorization of a particle code used in the simulation of rarefied hypersonic flow*, Computer Systems Engineering, 1 (2-4), 1990

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

[7] Bird, G.A., *Molecular Gas Dynamics and the Direct Simulations of Gas Flows*, Clarendon Press, Oxford, 1994

[8] Bird, G.A., *Recent Advances and Current Challenges for DSMC*, Computers Math. Applic., Vol. 35, No. 1, pp 1-14, 1998

[9] Bird, G.A., *The DSWT Implementation of the DSMC Method*, Program Notes, http://www.gab.com.au/DSWT.html, 2002

[10] Bird, G.A., *Recent DSMC Developments*, http://www.gab.com.au/, 2002

[11] Bird, G.A., *Direct Simulation of the Boltzmann Equation*, Physics of Fluids, Vol. 13, No. 11, 1970

[12] Bogdanov, A.V., Bykov, N.Yu., Grishin, I.A, et. al., *Algorithms of Two-Level Parallelization for DSMC of Unsteady Flows in Molecular Gas Dynamics*, Preprint HPCN 99, 1998

[13] Borgnakke, C., Larsen, P., *Statistical Collision Model for Monte Carlo Simulation Statistical Collision Model for Monte Carlo Simulation of Polyatomic Gas Mixture*, Journal of Computational Physics, Vol. 18, No. 4, pp. 405-420, 1975

[14] Boyd, I.D., *Vectorization of a Monte Carlo Scheme for Nonequilibrium Gas Dynamics*, Journal of Computational Physics, 94, pp. 411-427, 1991

[15] Bruno, D., Capitelli, M., Esposito, F., et. al., *Direct Simulation of Non-Equilibrium Kinetics Under Shock Conditions in Nitrogen*, Chemical Physics Letters 360, pp. 31-37, 2002

[16] Bucchignani, E., Mella, R., Schiano, P., et. al., *Comparison of the MPI and PVM performances by using structured and unstructured CFD codes*, Parallel Computational Fluid Dynamics: Recent Developments and Advances Using Parallel Computers, Elsevier Science, Amsterdam, 1998

[17] Carlson, A.B., Wilmoth, R.G., *Shock Interference Prediction Using Direct Simulation Monte Carlo*, Journal of Spacecraft and Rockets, Vol. 29, No. 6, 1992

[18] Chen, X.Q., *Efficient Particle Tracking Algorithm for Two-Phase Flows in Geometries Using Curvilinear Coordinates, Numerical Heat Transfer A, Vol. 32, No. 4, 1997*

[19] Dietrich, S., Boyd, I.D., *Scalar and Parallel Optimized Implementation of the Direct Simulation Monte Carlo Method*, Journal of Computational Physics, Vol 126, 1996

[20] Furlani, T.R., Lordi, J.A., *Comparison of Parallel Algorithms for the Direct Simulation Monte Carlo Method: Application to Exhaust Plume Flowfields*, Proceedings of the Sixteenth Symposium on Rarefied Gas Dynamics, P. Muntz, Ed., (AIAA publications, New York), 1988

[21] Garcia, A.L., Wagner, W., *Time step truncation error in direct simulation Monte Carlo*, Physics of Fluids, 12, pp. 2621-2633, 2000

[22] Geist, A., Beguelin, A., Dongarra, J., et. al., *PVM: Parallel Virtual Machine: A Users's Guide and Tutorial for Networked parallel Computing*, MIT Press, Cambridge, 1994

[23] Gordon, W.J., *Blending Function Methods of Bivariate and Multivariate Interpolation*, SIAM Journal of Numerical Analysis, Vol. 8, pp. 158-177, 1971

[24] Hadjiconstantinou, N.G., *Analysis of discretization*, Physics of Fluids, 12, pp. 2634-2638, 2000

[25] Harvey, J.K., Gallis, M.A., *Review of Code validation Studies in High-Speed Low-Density Flows*, Journal of Spacecraft and Rockets, Vol. 37, No. 1, 2000

[26] Hultquist, J.P.M., *Improving the Performance of Particle Tracing in Curvilinear Grids*, AIAA Aerospace Sciences Meeting, AIAA Paper 94-0324, 1994

[27] Jeng, Y.N., Lee, Z.-S., *Revisit to the Modified Multiple One-Dimensional Adaptive Grid Method*, Numerical Heat Transfer, Part B, Vol. 29, 1996

[28] Kannenberg, K.C., Boyd, I.D., Dietrich, S., *Development of an Object-Oriented Parallel DSMC Code for Plume Impingement Studies*, 30th AIAA Thermophysics Conference, AIAA Paper 95-2052, 1995

[29] Kannenberg, K.C., Boyd, I.D., *Development of a 3D Parallel Code for Plume Impingement Studies*, 31st AIAA Thermophysics Conference, AIAA Paper 98-1848, 1998

[30] Karypis, G., Kumar, V., *Parallel Multilevel k-way Partitioning Scheme for Irregular Graphs*, Technical Report #96-036, Department of Computer Science, University of Minnesota, 1996

[31] Kenwright, D.N., Lane, D.A., *Interactive Time-Dependent Particle Tracing Using Tetrahedral Decomposition*, IEEE Transactions on Visualization and Computer Graphics 2(2), pp. 120-129, 1996

[32] Kenwright, D.N., Lane, D.A., *Optimization of Time-Dependent Particle Tracing Using Tetrahedral Decomposition*, Proceedings of IEEE Visualization 1995, pp. 321-328, 1995

[33] Lane, D.A., *Scientific Visualization of Large-Scale Unsteady Fluid Flows*, Chapter 5, Scientific Visualization: Overviews, Methodologies, and Techniques; IEEE Computer Society; Los Alamos, 1999

[34] Laux, M., Fasoulas, S., Messerschmid, E.W., *Development of a DSMC Code on Planar Unstructured Grids with Automatic Grid Adaptation*, 30th AIAA Thermophysics Conference, AIAA Paper 95-2053, 1995

[35] Laux, M., *Optimization and Parallelization of the DSMC Method on Unstructured Grids*, 32nd AIAA Thermophysics Conference, AIAA Paper 97-2512, 1997

[36] Laux, M., *Local Time Stepping with Automatic Adaptation for the DSMC Method*, 7th AIAA/ASME Joint Thermophysics and Heat Transfer Conference, AIAA Paper 98-2670, 1998

[37] LeBeau, G.J., *A parallel implementation of the direct simulation Monte Carlo method*, Computer Methods in Applied Mechanics and Engineering, No. 174, 1999

[38] Lee, D., Tsuei, M., *A Modified Adaptive Grid Method for Recirculating Flows*, International Journal for Numerical Methods in Fluids, Vol. 14, 1992

[39] McMillan, W., Woodgate, M., Richards, B.E., et. al., *Demonstration of Cluster Computing for Three-dimensional CFD Simulations*, The Aeronautical Journal, Vol. 103, No. 1027, 1999

[40] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, International Journal of Supercomputing Applications 8(3/4), 1994

[41] Moon, B., Saltz, J., *Adaptive runtime support for direct simulation Monte Carlo methods on distributed memory architectures*, Proceedings of the Scalable High Performance Computing Conference (SHPCC94), IEEE Computer Society Press, pp. 176183, 1994

[42] Moss, J.N., Bird, G.A., *Direct Simulation of transitional flow for hypersonic re-entry conditions*, Progress is Astronautics and Aeronautics, Vol. 96, pp. 113-139, 1985

[43] Moss, J.N., Price, J.M., Dogra, V.K., et. al., *Comparison of DSMC and experimental results for hypersonic external flow*, AIAA Paper 95-2028, 1995

[44] Nance, R.P., Wilmoth, R.G., Moon, B., et. al., *Parallel DSMC Solution of Three-Dimensional Flow Over a Finite Flat Plate*, AIAA/ASME Sixth Joint Thermophysics and Heat Transfer Conference, AIAA Paper 94-0219, 1994

[45] Nicol, D., Saltz, J., *Dynamic remapping of parallel computations with varying resource demands*, IEEE Transactions on Computers, 37(9), pp. 1073--1087, 1988

[46] Olynick, D.P., Moss, J.N., Hassan, H.A., *Grid Generation and Adaptation for the Direct Simulation Monte Carlo Method*, Journal of Thermophysics, Vol. 3, No. 4., AIAA Paper 88-2734, 1989

[47] Prisco, G., *Optimization of direct simulation Monte Carlo (DSMC) codes for vector processing*, Journal of Computational Physics, 96, 1991

[48] Rault, D.F.G, *Efficient Three-Dimensional Direct Simulation Monte Carlo for Complex Geometry Problems*, Rarefied Gas Dynamics: Theory and Simulation; Progress in Astronautics and Aeronautics, Vol. 159, AIAA, 1994

[49] Robinson, C.D., Harvey, J.K., *A Parallel DSMC Implementation on Unstructured Meshes with Adaptive Domain Decomposition*, Proceedings of the Twentieth International Symposium on Rarefied Gas Dynamics, 1996

[50] Robinson, C.D., Harvey, J.K., *Adaptive Domain Decomposition for Unstructured Meshes Applied to the Direct Simulation Monte Carlo Method*, Proceedings of Parallel CFD 96, 1996

[51] Robinson, C.D., Harvey, J.K., *A Fully concurrent DSMC Implementation with Adaptive Domain Decomposition*, Parallel Computational Fluid Dynamics: Recent Developments and Advances Using Parallel Computers; Elsevier Science, Amsterdam, 1998

[52] Shimada, T., Abe, T., *Applicability of the Direct Simulation Monte Carlo Method in a Body-Fitted Coordinate System*, Rarefied Gas Dynamics: Theory and Computational Techniques; Progress in Astronautics and Aeronautics, Vol. 118, AIAA, 1989

[53] Sun, Q., Boyd, I.D., *A Direct Simulation Method for Subsonic, Mircoscale Gas Flows*, Journal of Computational Physics, 179, 2002

[54] Syms, G.F., *Parallel Performance of Aerodynamic CFD Codes*, Canadian Aeronautics and Space Journal, Vol. 47, No. 1, March, 2001

[55] Thompson, J.F. (Ed.), *Numerical Grid Generation*, Proceedings of a Symposium on the Numerical Generation of Curvilinear Coordinate Systems and their Us, 1982

[56] University Space Research Association, *Beowulf History*, http://www.beowulf.org/overview/history.html, 2002

[57] Wilmoth, R.G, *Application of a Parallel Direct Simulation Monte Carlo Method to Hypersonic Rarefied Flows*, AIAA Journal, Vol. 30, No. 10, AIAA Paper 91-0772, 1992

[58] Wilmoth, R.G., Carlson, A.B., Bird, G.A., *DSMC Analysis in a Heterogeneous Parallel Computing Environment*, AIAA Paper 92-2861, 1992

[59] Wishart, S., Auld, D., *Investigation of Improved Movement Algorithms for Body-fitted Coordinate Systems used in Direct Simulation Monte Carlo Flow Analysis*, Proceedings IAC, Vol. 3, 1997

[60] Woronowicz, M.S., Wilmoth, R.G., Carlson, A.B., et. al., *Procedure for Adapting Direct Simulation Monte Carlo Meshes*, Rarefied Gas Dynamics: Theory and Simulation; Progress in Astronautics and Aeronautics, Vol. 159, edited Shizgal, B.D. and Weaver, D.P., AIAA, Washington, DC, 1994

[61] Woronowicz, M.S., Rault, D.F.G., *Cecignani-Lampis-Lord gas-surface interaction model: Comparisons between theory and simulation*, Journal of Spacecraft and Rockets, No. 31, 1994

[62] Yokokawa, M., Schneider, D., Watanabe, T., et. al., *Parallel Simulation on Rayleigh-Bénard Convection in 2D by the Direct Simulation Monte Carlo Method*, Parallel Computational Fluid Dynamics: Implementations and Results Using Parallel Computers, Elsevier Science, 1995