Performance Modeling for Concurrent Particle Simulations

Thesis by

Marc A. Rieffel

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1998

(Submitted May 21, 1998)

Acknowledgements

I would like to thank my advisor, Dr. Stephen Taylor, for all of the support and guidance that he has given me over the past four years. Under his direction, I was able to progress rapidly in the academic environment, while simultaneously enjoying a host of other activities. When the going got tough, he showed me what it means to stand up for one's self and one's friends, no matter the size or strength of the opposition.

The work presented in this thesis is closely tied to that of Jerrell Watts in his thesis, *Dynamic Load Balancing and Granularity Control on Heterogeneous and Hybrid Architectures.* Without the continuous collaboration with Jerrell, this work would have taken much longer, and the results would not have been nearly as satisfactory. I have learned more from discussions with him than from any of the classes I have ever taken.

The application of the techniques presented here to problems of industrial relevance is primarily due to the guidance and assistance of Sadasivan Shankar of Intel Corporation. His advice and encouragement have been very helpful during my graduate study. Discussions with Mikhail Ivanov and Sergey Gimelshein from the Russian Academy of Sciences have been essential for understanding the DSMC method and its associated implementation techniques. The numerical validation studies in Chapter 2 and a number of the optimizations presented in Chapter 3 are direct results of this collaboration.

I would also like to thank Tiranee Achalakul, Maura Benton, Gillian Borgnak, Andrew Brown, Wayne Christopher, Evan Cohn, Viren Dogra, Xavier Fan, Peter Haaland, Bob Haimes, Ross Harvey, Alexander Kashkovsky, Gennady Markelov, John Maweu, Daniel Maskit, Jeremy D. Monin, Bradley Nelson, Sirikunya Nilpanich, Michael Palmer, Diane Poirier, Karie Smart, David Weaver, and Armin Wulf, for their assistance at different stages of this work. Thanks are due to the other members of

my thesis defense committee, Jim Arvo, K. Mani Chandy, and Vincent McKoy, for their insightful comments and suggestions, particularly on Chapter 4. Finally, I would like to thank all of those who joined me on numerous sailing, flying, diving, hiking, camping, and surfing trips.

Access to the Intel Paragon was provided by Caltech/CACR. Access to the Cray T3D was provided by the Jet Propulsion Laboratory. Infrastructure support and computing resources for this research were provided by BMDO under contract DAAH04-96-1-0319, and by Avalon Computer Systems, Intel Corporation, and Silicon Graphics. The research described in this report is sponsored by Intel Corporation and the Advanced Research Projects Agency under contract number DABT63-95-C-0116. This project includes Russian participation that is supported by the U.S. Civilian Research and Development Foundation under Award No. RE1241. The information contained herein does not necessarily reflect the position or policy of the government of the United States, and no official endorsement should be inferred.

This document was prepared using LaTeXand the thesis template written by Michael Kelsey.

Abstract

This thesis develops an application- and architecture-independent framework for predicting the runtime and memory requirements of particle simulations in complex three-dimensional geometries. Both sequential and concurrent simulations are addressed, on a variety of homogeneous and heterogeneous architectures. The models are considered in the context of neutral flow Direct Simulation Monte Carlo (DSMC) simulations for semiconductor manufacturing and aerospace applications.

Complex physical and chemical processes render algorithmic analysis alone insufficient for understanding the performance characteristics of particle simulations. For this reason, detailed knowledge of the interaction between the physics and chemistry of a problem and the numerical method used to solve it is required.

Prediction of runtime and storage requirements of sequential and concurrent particle simulations is possible with the use of these models. The feasibility of simulations for given physical systems can also be determined. While the present work focuses on the concurrent DSMC method, the same modeling techniques can be applied to other numerical methods, such as Particle-In-Cell (PIC) and Navier-Stokes (NS).

Contents

List of Figures

List of Tables

List of Programs

List of Symbols

Symbol	Description
A	sampling time, inflow area
$\vec{a}$	acceleration
b	largest prime factor of number of partitions
C	number of cells in a simulation
C_e	exposed cells
c	performance model parameter
$\vec{c}$	inflow stream velocity
d	distance, dimension of bounding box, HS collision diameter
D	number of dimensions, mesh diameter
E	parallel efficiency
e	statistical scatter or error
eff	efficiency of load balance
f^i	ratio of avg. compute time to i-th computer's compute time
g	collision relative velocity
h	free dimension of simulated volume
J	number of samples for error estimation
j	particle influx per unit area
K	convergence time, or ratio of mean free path to cell size
k	Boltzmann constant
L	characteristic dimension of simulated region
l	length, cell size
M	memory
m	particle mass
N	number of particles in a simulation
n	particle number density
n_p	number of particles communicated

Symbol	Description
P	number of simulations for unsteady flow
p	partitions, processors
P_t	target pressure
P_m	measured pressure
p	number of grid partitions
p_m	minimum number of particles per cell
$\vec{p}$	particle, probe position
q	dimensionality of the partitioning or the mesh
R	random variable uniformly distributed in, $0 \leq R < 1$
r	radius, reflectivity, samples per cell
S	total simulation timesteps, DSMC speed
s	molecular speed ratio, connections per partition, steps, processor speed
T	time or temperature
T_{ex}	particle exchange communication time
T_g	global communication time
T_l	communication overhead time
T_m	communication time per processor
t	time
Δt	timestep
U	ratio of avg. to max. compute time, load balance
u	measure of heat or work, dimensionless speed
$\hat{u}$	tangential unit vector
V	volume
v	speed
v_{mp}	most probable velocity
v_t	tangential or thermal velocity
$\vec{v}$	velocity

Symbol	Description
$\vec{v_n}$	normal component of velocity
$\overline{v}$	average particle speed
$\hat{v}$	tangential unit vector
w_p	particle weight, ratio of real to simulated particles
w_l	local particle weight
w_s	species weight
α	Variable Hard Sphere (VHS) model parameter, adaptive boundary condition parameter
β	Variable Soft Sphere (VSS) model parameter, inverse of most probable velocity
ϵ	azimuthal impact angle
λ	mean free path between collisions
Φ	particle flux
ϕ	tangential thermal velocity angle
σ	collision cross section
θ	inflow angle
θ_t	tangential inflow angle
τ	unsteady oscillation period
χ	collision scattering angle
ξ	any macroscopic parameter
$\Delta\xi$	error in macroscopic parameter ξ

Chapter 1 Introduction

This thesis presents a framework for understanding the runtime and memory requirements of particle simulations involving complex three-dimensional geometries. Application- and architecture-independent models for sequential and concurrent simulations are developed. Extensions of these models address the performance characteristics of concurrent simulations on a variety of homogeneous and heterogeneous architectures. Evaluation of the models is achieved with a variety of neutral flow Direct Simulation Monte Carlo (DSMC) simulations for semiconductor manufacturing and aerospace applications.

Due to the complex physical and chemical processes involved, traditional algorithmic analysis alone is insufficient for understanding the performance characteristics of particle simulations. In fact, these characteristics are primarily determined by the interaction between the physics and chemistry of a problem and the simulation technique.

The models presented here can be used to predict the runtime and storage requirements of sequential and concurrent particle simulations, and thereby determine the feasibility of simulations for given physical systems. While the present work focuses on the concurrent DSMC method, the same modeling techniques can be applied to other numerical methods, such as Particle-In-Cell (PIC) and Navier-Stokes (NS).

1.1 Contributions

The contributions of this research are:

1. Novel *sequential and concurrent algorithms* for three-dimensional DSMC simulations involving complex geometries. These include techniques for grid adaption, static and dynamic partitioning, static and dynamic load balancing [99,

100, 101], automatic granularity control [85], adaptive boundary conditions [80], and adaptive collection of results. Validation of these algorithms is performed for a variety of simulation conditions [32, 79], and their application to problems of industrial relevance is presented [90, 92]. These algorithms provide the infrastructure for validation and evaluation of the sequential and concurrent performance models presented in subsequent chapters [81, 83, 84, 93].

2. A *sequential model* for predicting runtime and storage requirements of DSMC simulations in a variety of physical configurations[83].

3. A *concurrent model* for predicting runtime and storage requirements of concurrent DSMC simulations on a variety of parallel, distributed, and heterogeneous architectures [86]

4. Evaluations of the *predictive power* of these models when applied to large-scale simulations[83, 86].

5. A study and quantification of the benefits of *automatic granularity control* in the context of realistic large-scale concurrent DSMC simulations[84].

1.2 Motivation

Recent advances in microprocessor performance have been driven primarily by improvements in manufacturing technology. New processes and equipment have paved the way for smaller feature sizes and larger wafer sizes. These have, in turn, enabled the production of microprocessors with more transistors, operating at lower voltages and with higher clock rates. One of the key pieces of equipment in microelectronics manufacturing is the *plasma reactor*, used in 30 to 40 percent of processing steps. Plasma processing equipment also accounts for approximately 20 percent of the cost of configuring a new semiconductor manufacturing plant. The cost of these production facilities is escalating, as are the research and development costs of introducing each new generation of processing technology. It is widely recognized that the devel-

opment of computational tools for modeling plasma reactors is essential to reduce the costs of validating new reactor designs and of improving manufacturing processes.

Significant advances have been made in recent years in the modeling of low-pressure neutral flow [15, 20]. While it is now possible to address a wide variety of engineering problems associated with neutral, ion, and electron transport in plasma reactors. However, the utility of these modeling techniques has so far been limited, as they do not address the entire design cycle of a simulation.

Process engineers seek simulation methods that are increasingly realistic, pushing toward three-dimensional models of complex geometries and reacting chemistry. Even two-dimensional calculations have large memory and performance requirements, and the conceptual leap to three-dimensional calculations has proved elusive. Acceptable simulation results require not only appropriate chemical and physical models, but also substantial investments in geometric modeling, gridding, post-processing, and analysis techniques. These steps in the design cycle are interrelated, and the process of obtaining a meaningful result is frequently a time-consuming, iterative process. For example, after some initial calculations it is frequently necessary to regenerate the computational grid or simplify the geometry. Simulation techniques will continue to have limited engineering applicability until each of these steps is addressed in a comprehensive simulation methodology.

The reactors of interest involve weakly ionized plasmas (with degree of ionization $\approx 10^{-4}$) with electron-impact gas-phase reactions and ion-enhanced surface reactions. Neutral flow is of importance in studying these systems since it determines the reactor pressure, the center-of-mass motion, and the production of chemically active species. In addition, the lateral location and possible agglomeration of particulates are determined by the characteristics of neutral flow. Since pressure drops are more significant in low pressure reactors, the study of neutral flow is critical to optimal design. The flow can be supersonic close to gas inlets, and good design is necessary to avoid large pressure drops and highly asymmetric flows.

Figure 1.1 shows a schematic representation of the process of semiconductor etching. Gas enters the reactor, reacts with the surface of the wafer, and the products

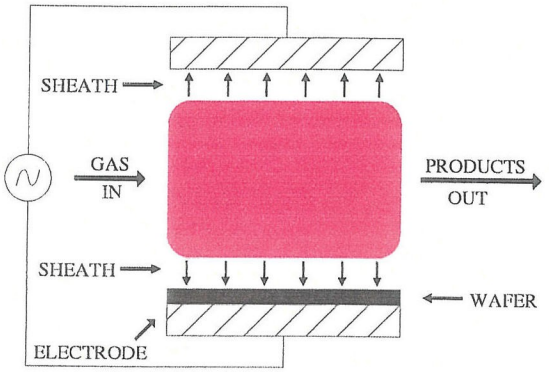

Figure 1.1: Schematic representation of semiconductor etching

of these reactions are pumped out of the reactor. Energy is supplied to the system through direct or alternating currents applied to electrodes in the reactor.

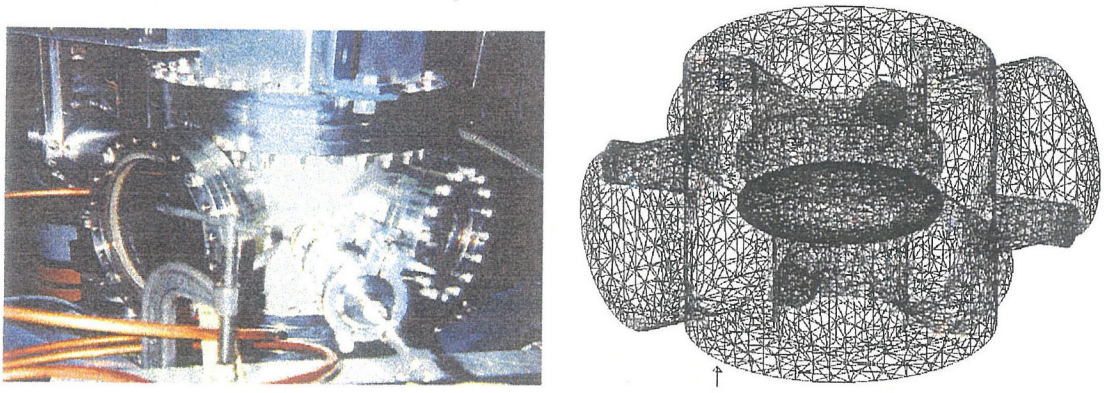

Figure 1.2: The Gaseous Electronics Conference (GEC) Reference Cell Reactor (left) and tetrahedral grid used to represent it (right)

The DSMC technique can be applied to the simulation of neutral flow inside three-dimensional plasma reactors with complex geometries. A typical plasma reactor, the Gaseous Electronics Conference (GEC) reference cell, is shown in Figure 1.2 (left). At low pressures (less than 200 Pa (1.5 Torr)), the mean free path is on the order of the characteristic length of the reactor (approximately 30 cm). The DSMC technique considered in this thesis leverages and integrates a variety of ideas taken from computational fluid dynamics and finite-element methods. A central aim is to ex-

ploit existing industrial tools that are already in use by process engineers in order to shorten the design cycle to acceptable engineering timescales. Application of the performance models developed in this thesis allows for the prediction of computational requirements of such simulations. A better understanding of the performance characteristics of particle methods, such as the DSMC technique, significantly increases the applicability of these methods to problems of industrial relevance.

1.3 Overview

The simulation of rarefied gas flow and the DSMC method as an example of particle simulation techniques are introduced in Chapter 2, and validation of a DSMC implementation is presented. Chapter 3 presents novel optimizations and extensions to the basic DSMC method. Chapter 4 presents a model of the prediction of runtime and storage requirements of sequential simulations, based on flow configurations and physical parameters. The model is then evaluated in the context of realistic large-scale simulations.

The performance model in Chapter 4 is extended in Chapter 5 to concurrent DSMC, and novel concurrent algorithms for partitioning, mapping, communication, load balancing, and granularity control are presented. Chapter 6 presents a model of the performance characteristics of concurrent DSMC simulations, and evaluates this model in the context of realistic simulations on several concurrent architectures. Chapter 7 explores the performance implications of dynamic load balancing and automatic granularity control, and Chapter 8 demonstrates the application of the concurrent DSMC method to simulations for semiconductor manufacturing and aerospace applications.

Chapter 2 Physical and Chemical Models

This chapter provides background information on the simulation of rarefied gas flow and presents the Direct Simulation Monte Carlo (DSMC) method. The infrastructure described in this chapter is used to assess the performance models in subsequent chapters. The DSMC method was chosen in order to illustrate the full complexity of particle simulations. In addition to the particle transport phase common to all particle simulations, the collision phase of the DSMC method provides additional constraints on performance analyses. By addressing the most complex of particle simulation techniques, the analyses presented in subsequent chapters are also applicable to a wide range of other numerical methods.

2.1 DSMC Overview

A variety of simulation techniques are used for the simulation of fluid flow, or gas dynamics. The characteristic parameter that determines gas flow properties is the Knudsen number, $Kn = \lambda/L$ where λ is the mean free path in a gas and L is the reference flow scale. In the *continuum regime*, where the Knudsen number tends toward zero, microscopic structure can be ignored, and a system can be completely described in terms of macroscopic parameters such as density, temperature, and velocity. In the *free-molecular regime*, where the Knudsen number tends toward infinity, collisions between molecules can be neglected, and the flow behavior is controlled by interactions between molecules and boundary surfaces. The region between the continuum and free-molecular regimes, where the Knudsen number is close to unity, is called the *transition regime*.

In the transition regime, viscosity, heat conduction, relaxation, diffusion, and

chemical processes are important, and it is also possible for velocity distribution functions to be non-Maxwellian, resulting in strong thermal nonequilibrium. As thermal and chemical relaxation lengths may be comparable to the reference flow scale, differences between translational, rotational, and vibrational temperatures may be important.

Several numerical techniques for simulating transitional gas flow have been developed in the past 20 years. Navier-Stokes and viscous shock layer equations can typically be used for the simulation of near-continuum flows, with appropriate extensions for modeling slip velocity and temperature jumps at surfaces. Because the Navier-Stokes equations assume only small deviations from thermal equilibrium, however, they are not suitable for studying rarefied flows with flow disturbances, such as shock waves, in which the velocity distribution functions are strongly nonequilibrium.

The governing equation in the transition regime is the Boltzmann Equation, a detailed treatment of which can be found in [23], [24], or [56]. It is a nonlinear integral-differential equation, closed with respect to the one-particle distribution function, which in turn determines the density of particles in a six-dimensional phase space of particle coordinates and velocities.

Some approaches for solving the Boltzmann equation include direct integration, molecular dynamics methods, the Direct Simulation Monte Carlo (DSMC) method, techniques coupling both DSMC and continuum methods [18], model equation approaches [88], and the test particle method [36]. The DSMC method is the approach of choice for the study of complex multidimensional flows of rarefied hypersonic aerothermodynamics, as well as for the simulation of neutral flow in plasma reactors. Reasons for this include the simple transition from one-dimensional to two- and three-dimensional problems, and the ease with which complex models of particle interaction can be incorporated without substantial increase in computational costs [52]. It is also well suited for use on modern concurrent architectures [80].

The DSMC method was pioneered by Graeme A. Bird [12, 13, 15]. It can be used to model chemical reactions [14], and has been extended to address translational and rotational effects in gaseous expansions [11], and to include the maximum entropy

(ME) and Borgnakke-Larsen (BL) models for internal energy exchange [63]. Sophisticated models have been developed for energy transfer between vibrational and translational modes, such as those used in simulating flow over a two-dimensional wedge [19]. DSMC has also been combined with fluid electron models and self-consistent electric fields to simulate plasma systems [9, 71]. A detailed description of the VSS collision model and its relation to the Lennard-Jones and inverse-power-law potentials can be found in [57, 58].

In principle, the DSMC technique can account for all of the physics needed for any problem [68], and is therefore a pure form of computational fluid dynamics. In practice, however, the technique can be substantially more computationally intensive than continuum approaches for high-pressure (low altitude) flows.

Systems that the DSMC method can be used to simulate include space vehicles in the upper atmosphere [14, 48, 49], plasma reactors for semiconductor manufacturing [9, 82, 96], lava flow from volcanos [2], and many others.

2.2 DSMC Algorithm

The Direct Simulation Monte Carlo (DSMC) method is an approach for solving the Boltzmann equation by simulating the behavior of individual particles. Since it is impossible to simulate the actual number of particles in a realistic system, a smaller number of simulation particles is used, each representing a large number of real particles. A computational grid is used to represent the simulated region of interest, and statistical techniques are employed to reproduce the correct macroscopic behavior.

At the start of a simulation, computational grid cells are filled with simulated particles according to initial freestream conditions. The simulation takes discrete steps in time, timesteps, during which a *transport model* is used to move particles through the grid, a *collision model* is used for particle-particle (including chemical) interactions, and a *boundary model* is used for interactions between particles and boundary surfaces. Macroscopic properties, such as density and temperature, are computed by appropriate averaging of particle masses and thermal velocities.

The DSMC algorithm is shown in Program 2.1. Particle motion and interactions are decoupled over the duration of a timestep. Each timestep is composed of two phases, the *transport phase*, during which particles move between grid cells, and the *collision phase*, during which particles interact within a cell. Global information, such as the total number of particles in the domain, may also be computed at every timestep. These phases are described in detail in the following sections.

```
DSMC()
{
  InitializeAllCells();
  for(step = 0; step < NUM_STEPS; step++)
  {
    MoveParticles( dt);         /* Transport Phase */
    CollideParticles(dt);       /* Collision Phase */
    ComputeGlobalInformation();
  }
  ComputeMacroscopicParameters();
}
```

Program 2.1: DSMC Algorithm

2.3 Transport Model

The transport model of the DSMC technique is concerned with moving particles through the simulated region for a specified period of time in order to determine their final positions both in space and within the computational grid. The transport phase of the algorithm is shown in Program 2.2. Each particle in each cell must be moved for a duration specified by the simulation timestep.

The requirements of the transport model are strongly influenced by the type of computational grid that is used. Unlike other types of grids, unstructured tetrahedral grids can be easily generated automatically, using existing CAD/CAM packages. They can also represent complex geometries, and can be locally adapted. For these reasons, the present work is based on, but not limited to, the use of unstructured tetrahedral grids. Particle transport through tetrahedral grid cells can be accomplished

```
MoveParticles(dt)
{
  for(each cell in domain)
  {
    for(particle = FirstParticle(cell); particle;
        particle = NextParticle(particle))
    {
      MoveParticle(cell, particle, dt);
    }
  }
}
```

Program 2.2: DSMC Transport Algorithm

using standard ray-tracing techniques [34]. This is accomplished by computing intersections between a particle's trajectory and the faces of the cell in which it is located, thereby determining the face through which the particle will exit and the adjacent cell into which it will travel. This process is repeated until the final position is found, both in physical space and in the computational grid.

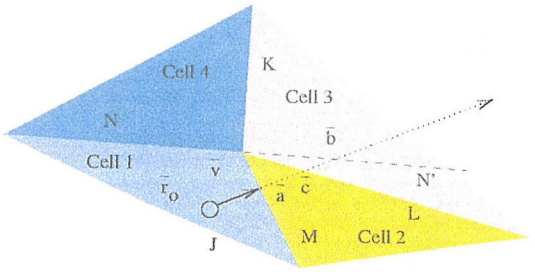

Figure 2.1: Particle trajectory through tetrahedral cells

Figure 2.1 illustrates this process in a two-dimensional representation of a sample geometry. Cells are represented as triangles and cell faces as line segments. Consider a particle initially at position $\vec{r_0}$ with velocity $\vec{v_0}$, initially located in Cell 1. The particle's vector position and cell location at the end of the timestep Δt are determined by its initial position, velocity, starting cell, and the timestep duration Δt.

Program 2.3 shows the algorithm for moving a single particle within a cell. To determine whether a particle will move into another cell, it is necessary to calculate which cell face the particle's trajectory will first intersect. In Figure 2.1, for example,

```
MoveParticle(cell, particle, dt)
{
  RayTrace(cell, particle, &closest_time, &closest_face);
            /* Recursion base case:  particle ends in this cell */
  if(closest_time > dt)
    MoveParticleWithinCell(cell, particle, dt);
  else
  {         /* Move particle to the edge of the cell */
    MoveParticleWithinCell(cell, particle, closest_time);
    if(BoundaryFace(closest_face))
    {
      PerformBoundaryInteraction(face, particle);
            /* Recursively move particle within same cell */
      MoveParticle(cell, particle, dt - closest_time);
    }
    else
    {
      nextcell = NeighborCell(face);
            /* Recursively move particle in adjacent cell */
      MoveParticle(nextcell, particle, dt - closest_time);
    }
  }
}
```

Program 2.3: Single-Particle Transport

the particle's trajectory will first intersect face M and then face L.

Given the closest face along the particle's trajectory, the intersection time for that face is compared to the timestep length, Δt. If the intersection time is greater than Δt, the particle will not hit any faces in the current timestep and can therefore be moved directly to its new position. If, however, the intersection time is less than the timestep, the particle will hit the closest face. Most cell faces are internal to the grid, and particles move through them to adjacent cells. If the cell face corresponds to a physical surface, the boundary model is employed to determine the interaction between the particle and the surface. Surface reactions may also occur at this point.

A particle that experiences no forces will travel in a straight line. In the presence of a uniform acceleration field, $\vec{a}$, particles experience acceleration, and follow parabolic

trajectories defined by,

$$\vec{r}(t) = \vec{r_0} + \vec{v_0}t + \frac{1}{2}\vec{a}t^2, \tag{2.1}$$

where $\vec{r}(t)$ is the particle's position at time t. Acceleration of this form is typically due to gravity or electric fields. The intersection time between the particle's curved trajectory and a cell face, t_i, is given by the equation,

$$\hat{n} \cdot (\vec{r}(t_i) - \vec{p}) = \hat{n} \cdot \left(\frac{1}{2}\vec{a}t_i^2 + \vec{v_0}t_i + \vec{r_0} - \vec{p}\right) = 0, \tag{2.2}$$

where $\hat{n}$ is a vector normal to the plane, and $\vec{p}$ is any point on the plane.

Since this equation is quadratic in t_i, it may have zero, one, or two real roots. The solution of iterest, corresponding to the first intersection with the closest cell face, corresponds to the smallest positive root. Complex roots have no physical meaning, while negative roots correspond to intersections *behind* the particle's initial position. Figure 2.2 (left) shows several possible intersection scenarios. Here, the particle's initial position is represented as a blue circle, the trajectory as a black curve, and the plane as a red line. In each case, the desired intersection, if any, is indicated by a green square.

This method, however, is inapplicable whenever $|\hat{n} \cdot \vec{a}| = 0$, which occurs whenever the acceleration is zero or parallel to the plane. The different possibilities for this case are shown in Figure 2.2 (right). If the particle's trajectory is not parallel to the plane, there is exactly one intersection time, and this is given by,

$$t_i = \frac{\hat{n} \cdot (\vec{r_0} - \vec{p})}{\hat{n} \cdot \vec{v}}. \tag{2.3}$$

Once the intersection time for the cell face is found, the intersection position, $\vec{p_i}$, is given by,

$$\vec{p_i} = \vec{r_0} + \vec{v_0}t_i + \frac{1}{2}\vec{a}t_i^2. \tag{2.4}$$

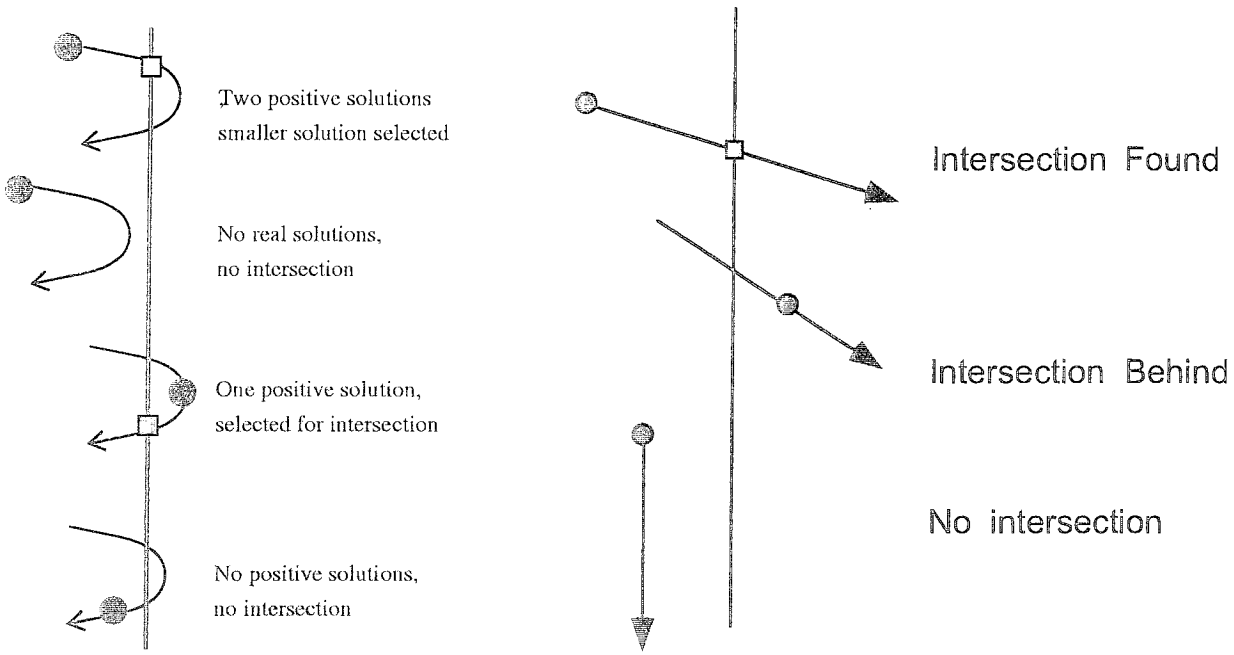

Figure 2.2: Possible intersections between a particle trajectory and a cell face for parabolic (left) and straight (right) trajectories

2.4 Collision Models

The ability of a DSMC implementation to correctly model a physical system is primarily determined by the complexity of particle-particle interactions, as specified in its collision model. Simplistic collision models can lead to erroneous results, while unnecessarily complex models can dramatically increase simulation time. Binary collisions are fundamentally an N-squared interaction for N particles. For simulations involving millions of particles, it is impractical to consider all possible interactions. If the mean free path is larger than the cell size, however, there can be no statistically significant spatial gradients within a cell. Taking advantage of this, the DSMC method needs only consider collisions between particles in the same cell. Statistical techniques are used to determine the correct number of collisions in time proportional to the number of particles in the cell.

The DSMC collision model uses an acceptance-rejection method based on collision

tests. The maximum frequency of test collisions in a cell, ν, is computed using,

$$\nu = \frac{N\,(N-1)\,w_p\,[\sigma(g_{ab})g_{ab}]_{max}}{2V}, \tag{2.5}$$

where g_{ab} is the magnitude of the relative velocity between a pair of particles a and b, $\sigma(g_{ab})$ is the collision cross section evaluated at that velocity, N is the number of particles in the cell, $[\sigma(g_{ab})g_{ab}]_{max}$ is an estimate of maximum possible value of $\sigma\,(g_{ab})\,g_{ab}$, w_p is the ratio of real to simulated particles, and V is the volume of the cell. The collision algorithm is shown in Program 2.4.

```
CollideParticles()
{
  for(each cell in domain)
  {
    nu = N * (N - 1) * w_p * sigijmax/(2 V);
    t = 0;
    while( t < dt)
    {
      t = t + 1/nu;
      if(t < dt)
      {
        SelectParticles(a,b);
        p = sigma(g_ab) * g_ab/sigijmax} ;
        with(probability p)
          PerformCollision(a,b);
      }
    }
  }
}
```

Program 2.4: DSMC Collision Algorithm

A time counter, t, is initialized to zero and incremented by the collision time, $\tau = \frac{1}{\nu}$, for every collision test. Collision tests are performed as long as t is less than the simulation timestep, Δt. Two particles, a and b, are selected randomly from the N particles in the cell, and these particles collide with probability $\frac{\sigma(g_{ab})g_{ab}}{[\sigma(g_{ab})g_{ab}]_{max}}$. The collision updates particle velocities appropriately .

In a collision, two scattering angles, χ and ϵ, are defined. As shown in Figure 2.3,

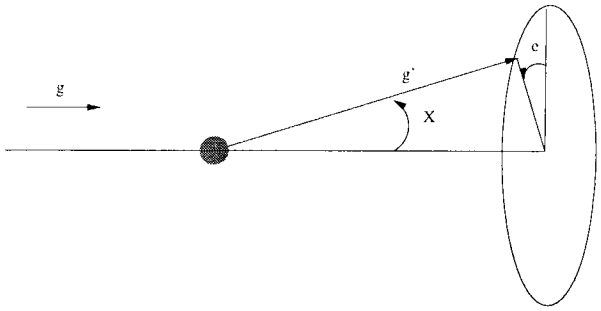

Figure 2.3: Collision scattering angles

χ is the angle between the pre-collision relative velocity and the post-collision relative velocity and ϵ is the azimuthal impact angle measured between the collision plane and some reference plane.

Three different collision models are used for monatomic gas interactions, Hard Sphere (HS), Variable Hard Sphere (VHS), and Variable Soft Sphere (VSS). The Hard Sphere model uses a constant collision cross section and isotropic scattering angles. In order to more accurately reproduce viscosity effects, the Variable Soft Sphere uses a cross section that is a function of temperature, or relative velocity. The Variable Soft Sphere model can also match diffusion coefficients by using non-isotropic scattering. In order to model polyatomic molecules with internal degrees of freedom, the Larsen-Borgnakke energy exchange model is employed [16].

The following sections describe these models in more detail, and explain the tests used for their validation. For certain tests, comparisons were made with another DSMC implementation, SMILE. SMILE is a computational tool for solving problems of rarefied gas aerothermodynamics [47], created at the Institute for Theoretical and Applied Mechanics (Novosibirsk, Russia). It is based on the majorant principle of construction and substantiation of numerical schemes for the DSMC method. The coupling of "cell" and "free cell" schemes [48] provides the required spatial resolution throughout the whole flow field, including regions with strong gradients. Comparison of two implementations is particularly useful when modeling physical systems that do not have simple analytical solutions.

2.4.1 Hard Sphere

In the Hard Sphere model, the collision cross section, $\sigma(g)$, is a constant of the model,

$$\sigma(g) = \sigma = \pi d^2. \tag{2.6}$$

For a given particle diameter, d, and impact parameter, b, the scattering angle, χ, is specified by,

$$b = d\cos(\chi/2). \tag{2.7}$$

Validation of the Hard Sphere model was conducted with two series of tests, one involving thermal relaxation in a box geometry, the second involving hypersonic flow past a cylinder.

The box tests were designed to verify that all of the basic DSMC operations were correctly implemented, including transport, inflow, accommodation, and HS collisions. Two identical species, Ar and Ar* were simulated, and the collision frequencies were found to agree with theory. Simulations were then conducted with Ar and He in equal concentrations, with density $n = 1.83 \times 10^{19}$ particles per cubic meter, and both species at equilibrium temperature 300K. The collision frequencies were equal to theoretical values, and the system remained in equilibrium. The next test considered a mixture of Ar and He at different temperatures. The system reached equilibrium at the correct temperature, the collision frequencies were correct, and equilibrium was reached at the same rate as SMILE simulations.

The next test considered different concentrations of Ar and He, (90% and 10%, respectively), with Ar at 100K and He at 500K. The final temperature of the system was verified to be $0.9 \times 100 + 0.1 \times 500 = 140K$. The last box case considered the opposite concentrations (10% and 90%), with argon at 100K and He at 500K. The final temperature of the system was verified to be $0.1 \times 100 + 0.9 \times 500 = 460K$.

The final validation test for the HS collision model considered Mach-4 flow past a cylinder of radius 0.05m, with a grid extending 0.15m in front of the cylinder, 0.35m behind, and 0.2m to the side. A freestream number density of 1.83×10^{21} particles per

cubic meter was used, with freestream speed 526.86 m/s and freestream temperature 50K. The cylinder itself was fully accommodating at 300K, and the gas was 50% Ar and 50% He. Because of the symmetry of this problem, only the upper half of the cylinder was simulated. A thin (0.01 m) 3-D grid was used for the *Hawk*, while the SMILE simulation was conducted in two dimensions. The results from the two simulations are shown in Figures 2.4 and 2.5, and they are in excellent agreement.

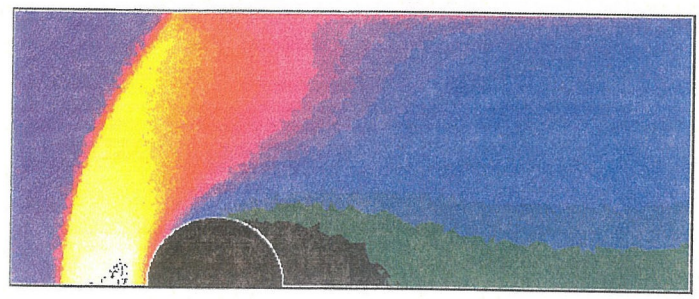

Figure 2.4: Hawk HS Argon Density

Figure 2.5: SMILE HS Argon Density

2.4.2 Variable Hard Sphere

The Variable Hard Sphere (VHS) model, developed by Bird in 1981, uses cross sections that are functions of relative velocity, but with hard sphere (HS) scattering angles. Collision, viscosity, and diffusion cross sections therefore take the form,

$$\sigma_T = \pi d^2 = c\left(m_R g^2/2\right)^{\alpha} \tag{2.8}$$

$$\sigma_\mu = \frac{2}{3}\pi d^2 \tag{2.9}$$

$$\sigma_D = \pi d^2 \tag{2.10}$$

where c and α are parameters of the model. As with the Hard Sphere model, the VHS model uses isotropic scattering angles,

$$b = d\cos(\chi/2). \tag{2.11}$$

For a gas at equilibrium, the total collision cross section may be written

$$\sigma = \sigma_T = \sigma_{ref}\left(\frac{T_{ref}}{T}\right)^{\alpha}, \tag{2.12}$$

where σ_{ref} is the reference value of the cross-section at the reference temperature T_{ref}, and α is a model parameter used to match viscosity. For a coefficient of viscosity μ, one can write $\mu \propto T^{\omega}$, where $\omega = \alpha + 0.5$. It is computationally inconvenient, however, for the cross section, a microscopic property, to depend on temperature, a macroscopic property. Integration of the velocity distribution function, however, yields the relation,

$$T = \frac{m_r g^2}{2k}\left(\Gamma(2-\alpha)\right)^{1/\alpha}, \tag{2.13}$$

where m_r is the reduced mass of the collision, g is the relative velocity of the collision, and k is the Boltzmann constant. The collision cross section can the be written,

$$\sigma_t = \sigma_{ref}\left(\frac{2kT_{ref}}{m_r g^2}\right)^{\alpha}\frac{1}{\Gamma(2-\alpha)}. \tag{2.14}$$

Initial validation of the VHS model consisted of running a uniform specular box and comparing the number of collisions between *Hawk* and SMILE. The results were in excellent agreement. The next series of tests studied the temperature relaxation rate in the specular box, with mixtures of argon and helium at different temperatures and in different concentrations.

The first of these tests was a 50%-50% mixture of argon and helium, with argon at 100 K and helium at 500K. As would be expected, the final temperature of the mixture was 300K. Figure 2.6 (left) shows that SMILE and *Hawk* both reached the

correct final temperature at the same rate. The second test was with 90% argon at 100K and 10% helium at 500K. The number of collisions, and convergence rates, as shown in Figure 2.6 (right), both agreed.

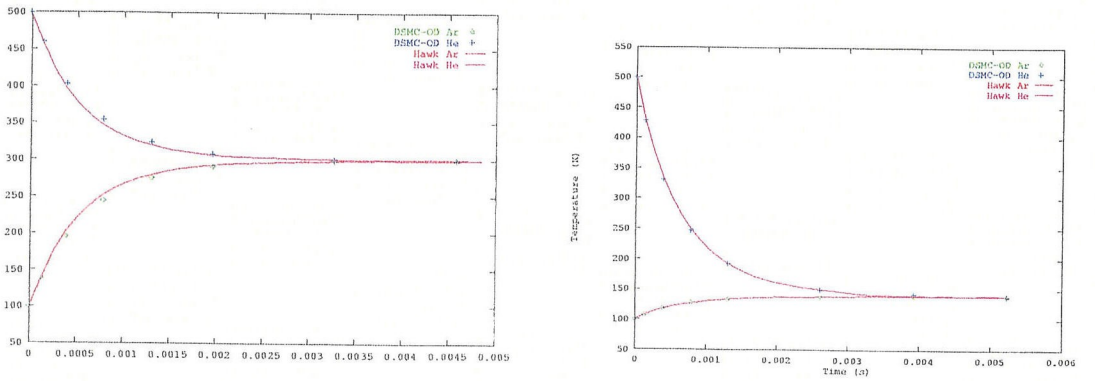

Figure 2.6: Temperature vs. time for VHS simulations, using 50% Ar and 50% He (left) and 90% Ar and 10% He (right)

The final test of the VHS model was the cylinder problem, as described in Section 2.4.1. For this, a mixture of 50% argon and 50% helium was used, with $\alpha = 0.5$. Density plots for the two codes are shown in Figures 2.7 and 2.8, and these agree very well. The density along the streamline was also compared and found to be the same for the two codes.

Figure 2.7: Hawk VHS Argon Density

2.4.3 Variable Soft Sphere

The main disadvantage of the VHS model arises when matching viscosity coefficients σ_μ. The diffusion cross-sections for the two models, and hence, the diffusion coef-

Figure 2.8: SMILE VHS Argon Density

ficients, coincide only for the hard sphere model. The variable soft sphere (VSS) model developed by Koura [57, 58] has no such drawback. The VSS model uses the same velocity-dependent cross section as the VHS model, but introduces anisotropic scattering angles. In terms of the molecular diameter, d, and the impact parameter, b, the VSS model can be written as,

$$\left(\frac{b}{d}\right)^{\beta} = \cos\left(\frac{\chi}{2}\right). \tag{2.15}$$

The model parameter β is used to characterize the anisotropy of the scattering angle, and is used to match the diffusion coefficient. (This β corresponds to $1/\alpha$ in [15].) These parameters must be used for computing the post-collision relative velocity $\vec{g'}$ as a function of the pre-collision relative velocity $\vec{g_0}$. The azimuthal impact angle, ϵ, is computed using a random number R_1,

$$\epsilon = 2\pi R_1, \tag{2.16}$$

where R_1 is a random variable, uniformly distributed $0 \leq R_1 < 1$. The two components of the scattering angle, χ, are computed using another random number, R_2,

$$\cos\chi = 2R_2^{\beta} - 1 \tag{2.17}$$

$$\sin\chi = \sqrt{1 - \cos^2\chi}. \tag{2.18}$$

The components of $\vec{g'}$ can then be computed as

$$g'_x = g_x \cos\chi + \sqrt{g_y^2 + g_z^2} \sin\chi \sin\epsilon \tag{2.19}$$

$$g'_y = g_y \cos\chi + \sin\chi \left(g' g_z \cos\epsilon - g_x g_y \sin\epsilon\right) \left(g_y^2 + g_z^2\right)^{-1/2} \tag{2.20}$$

$$g'_z = g_z \cos\chi - \sin\chi \left(g' g_y \cos\epsilon + g_x g_z \sin\epsilon\right) \left(g_y^2 + g_z^2\right)^{-1/2}. \tag{2.21}$$

The derivation of these equations may be found in [15].

Validation of the VSS model was conducted using a series of box tests and a cylinder flow test. The first box test used the parameters shown in Table 2.1. The system was shown to stay in equilibrium at 100K, and the collision frequencies agreed with those of SMILE simulations.

Table 2.1: Parameters for Isothermal VSS Box Test

Species	Ar-Ar	Ar-He	He-He
α	0.31	0.235	0.16
β	0.714	0.754	0.794
Concentration	0.5		0.5
Temperature (K)	100		100

The next test for the VSS model was with Ar and He at different initial temperatures, as shown in Table 2.2. The convergence rates of the temperatures were shown to agree with the results of SMILE, as shown in Figure 2.9 (left).

Table 2.2: Parameters for Thermal-Gradient VSS Box Test

Species	Ar-Ar	Ar-He	He-He
α	0.31	0.235	0.16
β	0.714	0.754	0.794
Concentration	0.5		0.5
Temperature (K)	100		500

Table 2.3: Parameters for Thermal-Gradient VSS Box Test

Species	Ar-Ar	Ar-He	He-He
α	0.31	0.235	0.16
β	0.714	0.754	0.794
Concentration	0.9		0.1
Temperature (K)	100		500

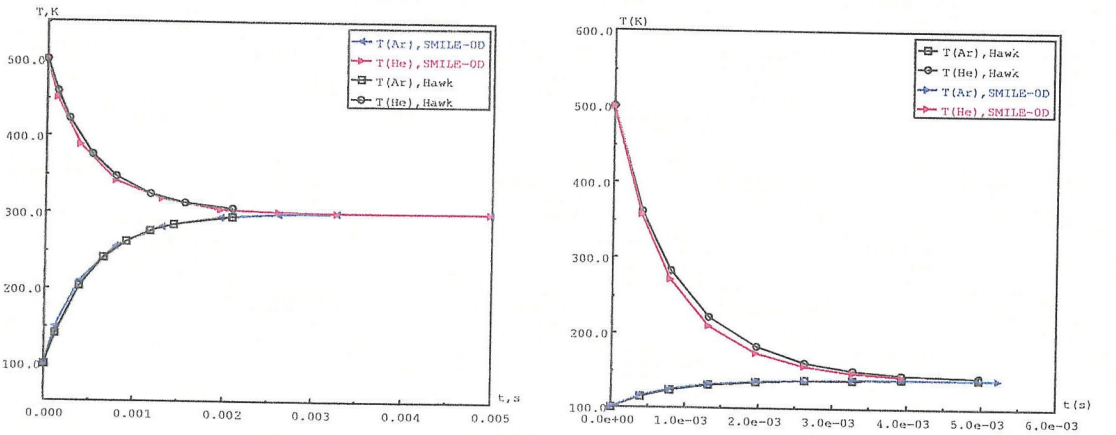

Figure 2.9: Temperature vs. time for VSS simulations, using 50% Ar and 50% He (left) and 90% Ar and 10% He (right)

The final box test for the VSS model used argon and helium in different concentrations, as shown in Table 2.3. Results were in excellent agreement with SMILE, as shown in Figure 2.9 (right). These same parameters were used for a simulation of the hypersonic cylinder flow. Results for *Hawk* and SMILE simulations are shown in Figures 2.10 and 2.11.

2.4.4 Larsen-Borgnakke

The Larsen-Borgnakke model of energy exchange is used to describe internal energy modes for rotation and vibration [16]. Relative translational and internal post-collision energies of colliding particles are assumed to be distributed according to equilibrium distribution functions.

The model associates with each species a number of atoms n_a and a charac-

Figure 2.10: *Hawk* VSS Argon Density

Figure 2.11: SMILE VSS Argon Density

teristic vibrational temperature Θ_v , and with each particle, rotational energy E_r and vibrational energy E_v. Rota- tional and vibrational energies are assumed to be continuous. The current work considers two types of energy exchange, translational-rotational (TR) and translational-rotational-vibrational (TRV). Each collision has some probability ϕ_R of a TR exchange and some (smaller)probability ϕ_V of a TRV exchange.

It is first necessary to characterize the number of degrees of freedom in each of the energy modes. Relative translational energy has 3 degrees of freedom. For rotational energy, the number of degrees of freedom ξ_r is a function of the number of atoms, n_a, given by,

$$\xi_r = \begin{cases} 0 & n_a = 1 \\ 2 & n_a = 2 \\ 3 & n_a \geq 3 \end{cases} \tag{2.22}$$

The number of effective vibrational degrees of freedom can be derived from the simple harmonic oscillator (SHO) approximation. Vibrational degrees of freedom are therefore a function of the local temperature, T, and the species' characteristic

vibrational temperature Θ_v, given by,

$$\xi_v = \frac{2\Theta_v/T}{e^{\Theta_v/T} - 1} \times \frac{n_a(n_a - 1)}{2}. \tag{2.23}$$

When a particle is first injected into the domain, whether by initial conditions, inflow, or surface emission, its initial rotational and vibrational energies must be computed. These values are sampled from the equilibrium distribution function for the specified temperature T. Internal energies, both rotational, E_r and vibrational, E_v, as functions of degrees of freedom, ξ_r and ξ_v, are computed as follows. If the number of degrees of freedom is less than or equal to two, the internal energy can be computed using a single random number R_1,

$$E = -\ln(R_1)\frac{\xi kT}{2} \quad \xi \leq 2. \tag{2.24}$$

If the number of degrees of freedom is greater than two, internal energy is sampled from the distribution function,

$$f(E) = \frac{1}{\Gamma(\xi/2)}\left(\frac{E}{kT}\right)^{\xi/2-1} e^{-E/kT} \quad \xi > 2, \tag{2.25}$$

using the acceptance-rejection method.

The two mechanisms for exchange of internal energy are between translational and rotational modes (TR), and between translational, rotational, and vibrational modes (TRV). These redistributions take place according to the equilibrium energy distribution function for a specified number of degrees of freedom, given by,

$$f(E) = \frac{1}{\Gamma(\xi/2)}\left(\frac{E}{kT}\right)^{\xi/2-1} e^{-E/kT}, \tag{2.26}$$

where $f(E)$ is the probability of the occurrence of energy E, ξ is the number of degrees of freedom, k is the Boltzmann constant, and T is the temperature. For an exchange between two modes A and B, with respective degrees of freedom ξ_A and ξ_B, the joint

distribution function is,

$$f(E_A, E_B) = \frac{e^{-(E_A+E_B)/kT}}{\Gamma(\xi_A/2)\Gamma(\xi_B/2)} \left(\frac{E_A}{kT}\right)^{\xi_A/2-1} \left(\frac{E_B}{kT}\right)^{\xi_B/2-1}. \tag{2.27}$$

This can be written in terms of the total energy, $E_t = E_A + E_B$,

$$f(E_t, E_B) = \frac{e^{-E_t/kT}}{\Gamma(\xi_A/2)\Gamma(\xi_B/2)} \left(\frac{E_t - E_B}{kT}\right)^{\xi_A/2-1} \left(\frac{E_B}{kT}\right)^{\xi_B/2-1}. \tag{2.28}$$

Acceptance-rejection sampling of this distribution yields E_B, from which $E_A = E_t - E_B$ can be computed.

Once a collision is selected to occur, exchanges may take place between translational, rotational, and vibrational energy modes. This is typically achieved by specifying a probability that TR exchange occurs, ϕ_R, and a probability that TRV exchange occurs, ϕ_V. These probabilities can also be computed as functions of species parameters and the local translational temperature [61, 76].

A TR exchange is performed as follows. The total collision energy to be redistributed is the sum of translational and rotational energies, $E_c = E_t + E_r$, where $E_r = E_r^A + E_r^B$ is the sum of the rotational energies of the two colliding particles. The first step in the exchange is to distribute the collision energy between translational and rotational energy modes. After checking the relative velocity in a collision, the distribution function is biased, so the number of relative translational degrees of freedom must be taken as $\xi_t = 4 - 2\alpha$. The number of rotational degrees of freedom is the sum of rotational degrees of freedom for the two colliding particles, $\xi_r = \xi_r^A + \xi_r^B$. The total collision energy is therefore first redistributed between ξ_t and ξ_r, respectively.

The rotational energy is then redistributed between the two particles with rotational degrees of freedom ξ_r^A and ξ_r^B, as described above, yielding E_r^A and E_r^B, respectively. The post- collision relative velocity, g', is then computed from the post-collision translational energy, E_t, using the reduced mass of the collision, m_r,

$$g' = \sqrt{\frac{2E_t}{m_r}}. \tag{2.29}$$

In a TRV exchange, the total collision energy is $E_c = E_t + E_i$, where E_t is the translational energy and E_i is the internal energy. The internal energy is in the sum of the internal energies of the two particles, $E_i = E_i^A + E_i^B$. The internal energy of a particle is the sum of rotational and vibrational energies, $E_i^A = E_r^A + E_v^A$. Similarly, the number of degrees of freedom in the various energy modes are, $\xi_t, \xi_i, \xi_i^A, \xi_i^B, \xi_r^A, \xi_r^B, \xi_v^A$, and ξ_v^B.

The total collision energy, E_c, is first distributed between translational and internal modes, using ξ_t and ξ_i, to obtain E_t and E_i. The translational energy E_t is used to compute the post-collision relative velocity, as described above. The internal energy, E_i, is distributed between the two particles, using degrees of freedom ξ_i^A and ξ_i^B, to obtain E_i^A and E_i^B, respectively. The internal energy of particle A, E_i^A, is then distributed between rotational and vibrational modes, with degrees of freedom ξ_r^A and ξ_v^A, to obtain rotational and vibrational energies, E_r^A and E_v^A, respectively. The internal energy for particle B is similarly distributed between E_r^B and E_v^B.

The translational temperature of a species, T_t, is calculated with,

$$T_t = \frac{m(\overline{v^2} - \overline{v}^2)}{3k}. \tag{2.30}$$

The rotational temperature, T_r, si similarly computed from the average rotational energy, $\overline{E_r}$, using,

$$T_r = \frac{2\overline{E_r}}{k\xi_r}. \tag{2.31}$$

The vibrational temperature, T_b, is computed using the characteristic vibrational temperature, Θ_v, and the average vibrational energy, $\overline{E_v}$,

$$T_v = \frac{\Theta_v}{\ln\left(1 + k\frac{\Theta_v}{E_v}\right)}. \tag{2.32}$$

The total temperature, T, can then be computed using,

$$T = \frac{\xi_t T_t + \xi_r T_r + \xi_v T_v}{\xi_t + \xi_r + \xi_v}, \tag{2.33}$$

where $\xi_t = 3$ is the number of translational degrees of freedom.

A series of tests was performed to validate this implementation of the Larsen-Borgnakke energy exchange model. These were designed to verify correct equilibrium conditions, as well as translational-rotational (TR) and translational-rotational-vibrational (TRV) relaxation rates. These tests were performed for a uniform box with specular walls.

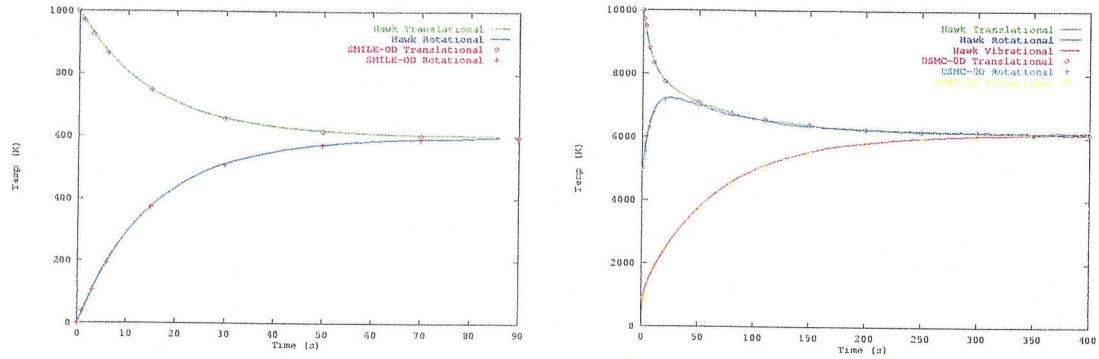

Figure 2.12: Temperature vs. time for simulations with TR relaxation (left) and TRV relaxation (right)

The first test was for N2 at equilibrium in a box, with translational, rotational, and vibrational temperatures equal to 10,000K, density 1.83×10^{19} particles per cubic meter, constant TR exchange probability $\phi_R = 0.2$, and constant TRV exchange probability $\phi_V = 0.02$. The system stayed in equilibrium for $\alpha = 0, 0.5$ and 0.24.

The next case considered relaxation between translational and rotational modes for $\alpha = 0.24$. Initially, the translational temperature was 1000K, the rotational temperature 0K, and the vibra- tional temperature 0K. The TR exchange probability was constant at $\phi_R = 0.2$, and no TRV exchanges took place ($\phi_V = 0$). The results for *Hawk* and SMILE agree as shown in Figure 2.12.

The next case considered relaxation between translational, rotational, and vibrational modes, with parameters as shown in Table 2.4.

The next case was a mixture of nitrogen and oxygen, each with internal degrees of freedom, and with TR and TRV exchange probabilities as specified in Table 2.5.

Table 2.4: Larsen-Borgnakke TR Validation Parameters

Species	N2
α	0.24
β	0.735
ϕ_R	0.2
ϕ_V	0.02
Θ_v	3390 K
T_{ref}	273 K
σ_{ref}	$5.3068 \times 10^{-19}\text{m}^2$
T_t	10000 K
T_r	5000 K
T_v	0 K

As shown in Figure 2.12 (right), the agreement between results was excellent.

Table 2.5: Larsen-Borgnakke TRV Validation Parameters

Species	N2-N2	N2-O2	O2-O2
α	0.24	0.255	0.27
β	0.735	0.725	0.7143
hline ϕ_R	0.2	0.2	0.2
ϕ_V	0.02	0.02	0.02
θ_v	3390 K		2256 K
T_{ref}	273 K	273 K	273 K
σ_{ref}	5.3068×10^{-19}	5.3069×10^{-19}	5.307×10^{-19}
Fraction	0.5		0.5
T_t	10000 K		15000 K
T_r	5000 K		7500 K
T_v	0 K		0 K

The final Larsen-Borgnakke test was for supersonic flow around the cylinder. A mixture of oxygen and nitrogen was used, with the same collision parameters as in Table 2.5. Figures 2.13 and 2.14 show trans- lational temperature flow fields for *Hawk* and SMILE, respectively. These figures show excellent agreement between the two simulations.

In order to quantitatively evaluate the agreement between the two results, temperatures in each of the three modes (translational, rotational, and vibrational) were

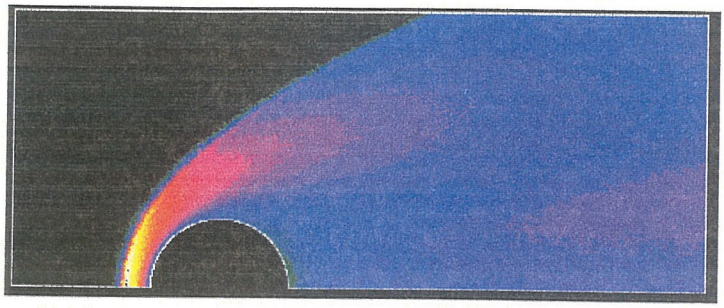

Figure 2.13: *Hawk* Larsen-Borgnakke Argon Density

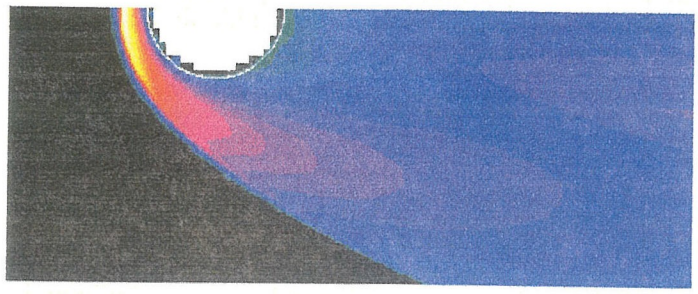

Figure 2.14: SMILE Larsen-Borgnakke Argon Density

sampled along the centerline upstream of the cylinder. Figure 2.15 shows these values for both *Hawk* and SMILE. The two simulation results agree to within statistical scatter.

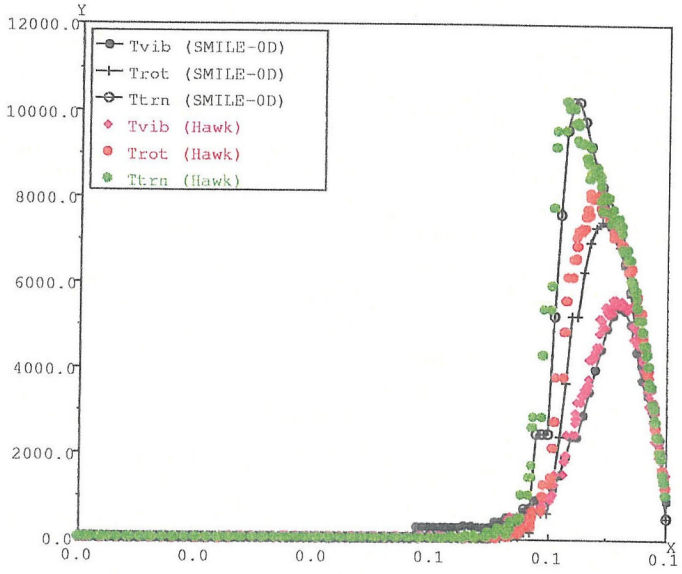

Figure 2.15: Temperature profiles along the centerline for SMILE (lines) and *Hawk* (points)

2.5 Boundary Models

The process of configuring a simulation involves specifying physical boundary conditions for each of the boundaries. These conditions determine the nature of interactions between simulated particles and the boundary surfaces. The three fundamental surface types are *inflow, outflow,* and *solid.* Inflow surfaces are used for inlet and upstream conditions, outflow surfaces are used for exhaust or downstream conditions, and solid surfaces are used for physical objects in the domain, such as reactor walls or aircraft wings.

2.5.1 Inflow Surfaces

The inflow surface model is the primary mechanism for introducing mass, momentum and energy into a neutral flow simulation. Consider a cell face on this surface, with area A, unit normal $\hat{n}$, and unit tangential vectors $\hat{u}$ and $\hat{v}$, as shown in Figure 2.16. Assume that this face is designated to have inflow of density n, at temperature T, with stream velocity $\vec{c}$. Let θ be the angle between the face normal and the stream velocity and β the inverse of the most probable velocity, $\beta = \sqrt{\frac{m}{2kT}}$, where m is the particle mass and k is the Boltzmann constant.

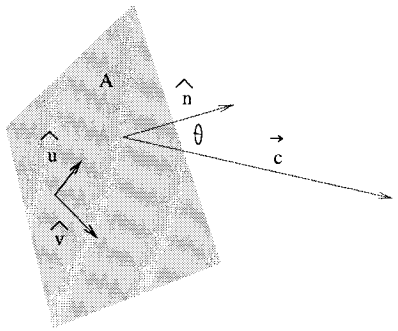

Figure 2.16: Particle inflow vectors

The molecular speed ratio, s, is given by $s = |\vec{c}|\beta$. The particle flux Φ can be computed, using,

$$\Phi = e^{-(s\cos\theta)^2} + s\sqrt{\pi}\cos\theta(1 + \mathrm{erf}(s\cos\theta)). \tag{2.34}$$

The number of simulated particles crossing the face, per unit area, per unit time, j, is then,

$$j = \frac{\Phi n}{2w_p\beta\sqrt{\pi}}, \tag{2.35}$$

where w_p is the ratio of real to simulated particles. The average number of simulated particles, N, to be injected during timestep Δt is given by,

$$N = jA\Delta t, \tag{2.36}$$

The thermal velocity of an injected particle, $\vec{v}$, is computed as the sum of normal and tangential thermal velocities, $\vec{v_n}$ and $\vec{v_t}$, respectively. Let $c_n = \vec{c}\cdot\hat{n}$ be the normal component of the stream velocity, and let $s_n = c_n\beta$. A dimensionless speed, u, is sampled, using the acceptance-rejection method, from the Maxwellian distribution function,

$$f(u) = ue^{-(u-s_n)^2}. \tag{2.37}$$

The dimensionless speed is then converted to the normal thermal speed, $v_n = \frac{u}{\beta}$, and the normal component of the thermal velocity is calculated as, $\vec{v_n} = v_n\hat{n}$. In order to determine the tangential component of the thermal speed, $\vec{v_t}$, a random thermal speed, v_t, is computed for the specified temperature, T, using,

$$v_t = \frac{3kT}{m}\sqrt{-lnR_l}, \tag{2.38}$$

where R_1 is a random number. Let ϕ represent the angle that the tangential thermal velocity will make with respect to the tangential vector $\hat{v}$. This is computed using $\phi = 2\pi R_2$, where R_2 is another random number. The thermal components of the tangential thermal velocity, $\vec{v_u}$ and $\vec{v_v}$, are given by

$$\vec{v_u} = \hat{u}v_t\sin\phi \tag{2.39}$$

$$\vec{v_v} = \hat{v}v_t\cos\phi. \tag{2.40}$$

The thermal tangential velocity, $\vec{v_t}$, is the sum of these two components,

$$\vec{v_t} = \vec{v_u} + \vec{v_v}. \tag{2.41}$$

The total thermal velocity, $\vec{v}$, is then the sum of its normal and tangential components, $\vec{v} = \vec{v_n} + \vec{v_t}$, and the total inflow velocity, $\vec{w}$, is the sum of stream and thermal components, $\vec{w} = \vec{c} + \vec{v}$.

Once the velocity of an inflowing particle has been determined, its initial position is selected as a random point on the boundary face, and it is moved for a random fraction of the global timestep Δt. Counts of outflowing particles may be kept in order to compute surface flux properties.

2.5.2 Outflow Surfaces

Outflow surfaces can be used to model the downstream conditions for an aerospace simulation, or the exhaust port of a plasma reactor. Particles that hit outflow surfaces are simply removed from the simulation.

2.5.3 Solid Surfaces

A solid surface is one through which particles cannot pass. These are typically used to model rigid objects, such as aircraft wings or walls of a reactor. A particle that hits a solid surface may leave the surface with a velocity determined by the surface temperature via *diffuse reflection*, or may bounce elastically via *specular reflection*. Surfaces may also be specified as in between specular and diffuse, with a probability of each. Surface reactions may also take place on solid surfaces.

Accommodating Surfaces

A particles colliding with an accommodating surface acquires the surface temperature. Consider a particle of mass m, colliding with a accommodating surface with temperature T, surface normal $\hat{n}$, and tangential unit vectors, $\hat{u}$ and $\hat{v}$. The most

probable velocity, v_{mp}, of a particle of mass m at temperature T is given by,

$$v_{mp} = \sqrt{\frac{2kT}{m}}. \tag{2.42}$$

The normal component of the post-collision diffuse velocity, $\vec{v_n}$, is obtained using this and a random number R_1,

$$\vec{v_n} = v_{mp}\sqrt{-\ln R_1}\hat{n}. \tag{2.43}$$

The tangential speed, v_t, is computed similarly, using a second random number R_2,

$$v_t = v_{mp}\sqrt{-\ln R_2}. \tag{2.44}$$

In order to distribute the velocity uniformly in the tangential direction, a third random number, R_3, is used to compute the tangential angle, θ_t,

$$\theta_t = 2\pi R_3. \tag{2.45}$$

The tangential direction $\hat{t}$ is computed from the tangential angle and the two tangential vectors $\hat{u}$ and $\hat{v}$,

$$\hat{t} = \hat{u}\sin\theta_t + \hat{v}\cos\theta_t. \tag{2.46}$$

The tangential velocity of the particle,$\vec{v_t}$, is obtained from the tangential speed and direction using

$$\vec{v_t} = v_t\hat{t}. \tag{2.47}$$

Finally, the post-reflection velocity of the particle is determined by adding normal and tangential components,

$$\vec{v_a} = \vec{v_t} + \vec{v_n}. \tag{2.48}$$

Specular Surfaces

A particle arriving at a specular surface will leave with angle of reflection equal to the angle of incidence. The component of the velocity in the direction of the surface

normal $\hat{n}$ is completely reversed, while the other two components are unchanged. For an incoming velocity $\vec{v_0}$, the exit velocity, $\vec{v_s}$, is given by,

$$\vec{v_s} = \vec{v_0} - 2(\vec{v_0} \cdot \hat{n})\hat{n}. \tag{2.49}$$

2.6 Convergence Considerations

As with any numerical simulation, it is important to ensure convergence of the algorithm. The total amount of simulation time required for a steady-state simulation to converge is governed by the acoustic time: the amount of time that it takes for thermal disturbances to traverse the entire width of the simulated region. For simple simulations, several acoustic periods may be sufficient for a simulation's macroscopic parameters to converge to their steady-state values. A predictive model for this process is discussed in Section 4.1.1.

Increasing the level of complexity of the simulation may increase the amount of time required for convergence. For example, the use of adaptive boundary conditions, as described in Section 3.2.1, requires the simulation to repeatedly converge after small changes in surface properties. For this type of simulation, tens or hundreds of acoustic periods may be required for final simulation convergence.

2.7 Accuracy Considerations

One advantage of the DSMC method is that it allows for straightforward estimation of the error of a solution. Once steady state has been achieved, several statistically independent solutions can be obtained by averaging macroscopic parameters over several thousand steps, resetting statistics, and then averaging again. The average of several different solutions gives a good estimate at the "true" solution, and the standard deviation of the solutions provides an estimate of the accuracy of the solution. Consider a simulation for which a macroscopic parameter ξ_i is sampled at J different

times. The best estimate of ξ is the average value,

$$\overline{\xi} = \frac{1}{J}\sum_{i=1}^{J}\xi_i. \tag{2.50}$$

Similarly, the estimated error in ξ, $\Delta\xi$, is given by the standard deviation of these values,

$$\Delta\xi = \frac{1}{J-1}\sum_{i=1}^{J}\left(\xi_i - \overline{\xi}\right)^2. \tag{2.51}$$

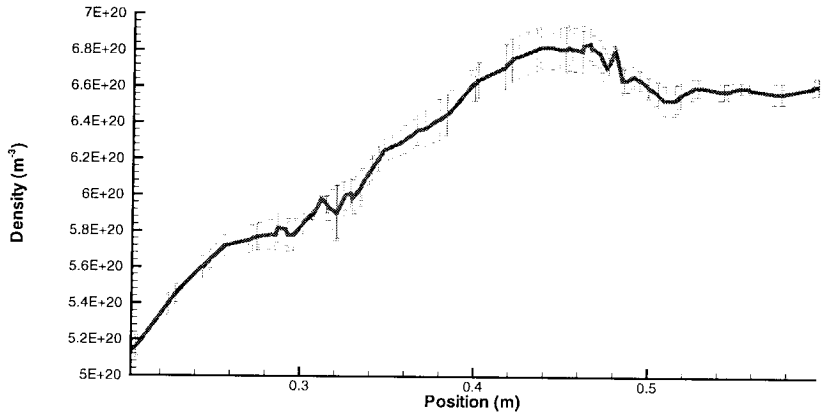

Figure 2.17: Particle density along a horizontal line above the wafer, showing estimated errors

The result of applying this procedure to a large-scale GEC simulation is shown in Figure 2.17. In this case, the macroscopic parameter ξ is particle density, n, and $J = 5$ samples were used. The mean value of density, $\overline{n}$, is plotted as a function of position. (Here, position is distance measured horizontally across the reactor, above the wafer.) At each point, error bars are drawn with sizes given by Δn. The larger errors in this line correspond to regions of small grid cells, few particles, and high statistical scatter.

2.8 Related Work

Several different gridding techniques have been used for DSMC computations. While all methods use grid cells for collision calculations, methods differ in the relations

between grid cells and boundary surfaces. For simple box-like geometries, a Cartesian, or axially aligned grid, can be used, with its surfaces representing the boundary surfaces [15, 104]. Using this type of grid, particle transport can be performed as two separate steps, movement and indexing. Particle destinations in space are quickly computed from their initial positions and velocities, and then particles are indexed into cells according to their final positions. Axially aligned grids, however, are unable to represent curved geometries directly.

For simulations involving trivial geometries, straight trajectories, and Cartesian grids, particle transport and indexing are simple and fast [15, 66, 73, 87]. A particle's final position is quickly computed from its initial position and velocity, and the final position can be indexed into the grid with a small number of floating point operations. Boundary interactions are discovered by comparing the particle's final position with the boundaries of the domain, another fast and simple operation. The computational cost of this operation is not a function of the length of the particle's trajectory.

For straight trajectories in simple geometries [106], it is possible to determine whether a particle's trajectory intersects with a boundary surface purely by considering its starting and ending positions. For curved trajectories, however, this is not possible. For parabolic trajectories, such as would result from a constant electric or gravitational field, the particle acceleration must also be considered [80]. Boundary intersections for trajectories that are even more complex may have to be considered in short segments. In this case, the cost of the algorithm depends on the length and curvature of the trajectory. If acceleration is a local property, and computed separately for each grid cell (as would be the case for a DSMC simulation coupled with an electric field solver), a particle's trajectory must be recomputed for each cell through which it passes. In this case, again, the cost depends on the length and complexity of the trajectory.

For completely arbitrary geometries, it is necessary to check whether any of the faces that the trajectory crosses represents a boundary surface. If a boundary surface is crossed during the timestep, the trajectory must be updated accordingly. The computational cost of this approach therefore depends on the number of faces (both

internal and boundary) that the particle crosses, and therefore the length of the particle's trajectory.

For non-Cartesian grids, including hexahedral [96] and tetrahedral grids [26, 80], additional complexity results from the fact that a position in space cannot be trivially indexed into the computational grid. Information relating spatial positions to grid locations is only stored locally. Given a starting cell, position, and velocity, it is only possible to determine the cell face through which the particle will exit the cell. If this face represents a boundary surface, the boundary model can be employed. If not, the particle can then pass through to the neighboring cell on the opposite side of the face. Information about this next cell can then be retrieved, and this process repeated until the final destination cell is found. The cost of this operation is also determined by the number of faces through which the particle passes.

Structured hexahedral grid surfaces can be matched to boundary surfaces for moderately complex geometries [74, 96]. For these grids, more sophisticated techniques, such as coordinate-transformation[79] or ray tracing [96], must be used to determine particle destinations within the grid. Unfortunately, hexahedral grids cannot be easily generated automatically, nor can they be locally refined.

Only two techniques allow the use of automatically-generated grids for complex geometries and local adaption of grid cells: ray-tracing on tetrahedral grids[26, 80] and the decoupled-grid approach[46]. While mature techniques exist for automatic generation, smoothing, and local refinement of unstructured tetrahedral grids for extremely complex geometries, transport operations within these grids must be performed using ray-tracing and are therefore computationally intensive. The decoupled-grid approach, on the other hand, uses a Cartesian grid for collision computations, and a triangular surface mesh for geometry representation. While the transport computations in the decoupled-grid approach are more complex than those for simple axially aligned geometries, they are generally faster than ray tracing on tetrahedral grids.

For both of these approaches, more sophisticated techniques are required to correctly capture particle-surface interactions. While a particle's destination may be

simply computed, it is necessary to ensure that the particle's trajectory does not cross any boundary surfaces during a timestep.

In the decoupled-grid approach [46, 47, 51, 52], an axially aligned grid is used for collision computations, and boundary surfaces are represented with separate data structures. A set of triangular surface patches, for example, can be automatically generated and used in place of the grid for boundary interactions. A hierarchical octree structure can then be used for local, and even directional, refinement. The computational cost of moving particles through an octree Cartesian grid, with reference to triangular surface element, is greater than that of moving particles in a simple Cartesian grid, but potentially less than that of ray-tracing through a tetrahedral grid. The performance modeling techniques presented in subsequent chapters apply equally well to this method of particle transport.

2.9 Summary

This chapter has introduced the DSMC method for the simulation of rarefied gas flows. The algorithms associated with the numerical technique have been presented. A series of validation tests for each of the collision models has been described. Issues of accuracy, convergence, and validation have also been addressed. The following chapters elaborate on the sequential algorithms, the concurrent DSMC technique, and performance considerations for DSMC simulations.

Chapter 3 Sequential Algorithms

This chapter discusses a variety of novel optimizations for DSMC implementations and extensions to the basic DSMC model. Optimizations are used to reduce the amount of computational time required for sequential simulations, while extensions to the technique are designed to improve the quality and accuracy of results for a given amount of simulation time, or to enable the simulation of more complex physical and chemical processes.

3.1 Model Optimizations

This section presents several optimizations to the basic DSMC algorithm as described in Section 2.2. Each component of the model presents opportunities for sequential optimization and tuning. Transport model optimizations revolve around fast determination of particle trajectories. Optimizations to the collision model reduce the amount of overhead computation. Boundary model optimizations allow for fast interactions between particles and surfaces. As with any Monte Carlo technique, many pseudo-random numbers must be used, and the choice of pseudo-random number generator can have a significant impact on the accuracy and speed of simulations.

3.1.1 Transport Model Optimizations

Particle transport in unstructured tetrahedral grids can be accelerated through the use of inscribed spheres. For each grid cell, the cell center and radius of an inscribed sphere can be computed. It can be quickly determined whether a particle's final position lies within the inscribed sphere of the starting cell. If so, the particle will stay in the cell and no ray tracing computations are required. If not, the full ray tracing algorithm must be used.

The effectiveness of this optimization was measured by conducting a series of simulations of a box geometry, both with and without the use of inscribed spheres. This grid was primarily composed of right tetrahedra (with 3 right angles), which are common in automatically generated grids. Measurements showed that an average of 9.4% of the volume of the cell is contained in the inscribed sphere. Since the cost of moving particles to points within the spheres is negligible compared to the cost of the full ray-tracing algorithm, the speedup of the transport algorithm should be almost as much as the fraction of the volume contained in the inscribed spheres. Timing tests confirmed this, with measurements showing a 9% speedup of the transport algorithm. The optimization was also extended to check the inscribed spheres of the neighboring cells, but experiments showed that the cost of the additional tests was greater than the time savings for the small fraction of particles found in the inscribed spheres of neighboring cells.

The effect of cell shape on the sphere optimization was also observed in a simulation with grid adaption activated. With all cells adapting, the typical cell shape changed with each level of adaption, and the fraction of particles that moved inside the spheres was seen to alternate between 7% and 10%.

3.1.2 Collision Model Optimizations

In the DSMC algorithm presented in Program 2.1, the transport and collision phases are distinct. First, all of the particles in a cell are moved, and then all of the collisions in the cell are computed. Because particles continuously move into and out of cells, it is most convenient to use a linked-list data structure to associate cells with the particles that they contain. The transport phase of a timestep is then a direct linked-list traversal. The collision phase, however, requires random access to particles in the cell. For this reason, collisions cannot be performed during a single traversal of the linked-list of particles. The technique used for this study is to traverse the list of particles, building an array of pointers to the particles and thereby allowing the collision phase to access the particles directly.

In initial experiments involving non-reacting flows, it was possible to perform the collision phase during the same traversal as the transport phase. All of the test-collision pairs were computed and stored in a table. During the linked list traversal, this table was consulted in order to determine if a particle must take part in a collision.

A one-pass timestep implemented in this manner has the advantages of speed and efficiency, particularly when the fraction of colliding particles is small. Unfortunately, this technique cannot be used for simulations involving reacting flows. When chemical reactions take place in a cell, the number of particles of each species changes during the collision process. Because the collision frequency is a function of the number of particles present from each species, the number of test collisions and collision pairs cannot be predicted *a priori*. It is therefore impossible to precompute a table of colliding particles. For reacting flows, the implementation considered in this thesis requires one traversal of the particle list per timestep, plus one additional traversal for each reaction that takes place.

Profiling experiments showed that the amount of computation required for collisions in flows involving multiple species, internal degrees of freedom, and chemical reactions is much greater than that required for single-species, monatomic, non-reacting simulations. An important optimization was therefore to use a simple streamlined version of the collision algorithm for simulations that did no not involve multiple species or chemical reactions. The result was to reduce the time required for the collision phase of the algorithm by about 20%.

3.1.3 Boundary Model Optimization

Intersections between particle trajectories and boundary surfaces are detected during the transport phase of a DSMC simulation, therefore the boundary model was implemented as part of the transport phase. Because the fraction of particles that interact with boundary surfaces during any timestep is small (typically less than 10%), and the operations required for boundary interaction are simple, the boundary model does not contribute significantly to the total timestep time.

3.1.4 Pseudo-Random Number Generation

Pseudo-random numbers are used in many parts of the DSMC method. The most common instance is for collision calculations, for example when determining whether a test collision is a real collision (Section 2.4) and for computing post-collision relative velocities (Section 2.4.3). Pseudo-random numbers are also used extensively in the boundary models, for particle inflow (Section 2.5.1) and accommodating surfaces (Sect 2.5.3).

As with any Monte Carlo method, the selection of a pseudo-random number generator can have significant effects on the speed and accuracy of the simulation. The use of a generator with too short a period or poor equidistribution properties can result in non-physical results. On the other hand, the use of a computationally-expensive generator can dramatically increase simulation time.

For the purposes of this study, five generators were considered: `ran2`, from *Numerical Recipes in C* [78]; `tt800` and `MT19937B` by Matsumoto and Nishimura [64]; `random2` by Park and Miller [78]; and `slaran` from LAPACK [28]. These generators were compared in terms of period and equidistribution properties (where available), and execution times. Execution times were measured by generating ten million pseudo-random numbers on an SGI Power Challenge. The results were normalized by the fastest time. The results are shown in Table 3.1. The fastest time was for `tt800`, followed closely by `MT19937B`, which also had the longest period and best equidistribution properties. These results show that there can be as much as a four-fold difference in execution times between generators.

Table 3.1: Pseudo-Random Number Generator Performance

Algorithm	Period	Equdistribution	Time
ran2	2^{18}	N/A	3.91
tt800	2^{800}	25	1.00
MT19937B	2^{19937}	623	1.03
random2	N/A	N/A	1.90
slaran	N/A	N/A	1.84

The generator with the longest period, `MT19937B`, works as follows. It uses a simple initialization routine to fill a table with intermediate random values. One random number is computed from each entry in the table, and then a new table is recomputed from the old table. While the recalculation of this table is computationally intensive, it must only be done infrequently. Computing random numbers from table entries, however, is a very fast operation. `MT19937B` was therefore modified to perform the table extraction inline, requiring a function call only for recomputing the table. This approach dramatically reduced the overhead time required for pseudo-random number generation, increasing the speed of `MT19937B` by a further 20%. The size of the table can also be adjusted to obtain a suitable balance between memory usage and the cost of recomputing the table. In practice, `MT19937B` has proven to be a fast and reliable generator.

With only minor modifications, it was converted to a thread-safe implementation.

3.2 Model Extensions

Extensions to the basic DSMC technique described in Chapter 2 improve the accuracy or efficiency of simulations and allow for the simulation of more sophisticated physical and chemical processes. Adaptive boundary conditions are used in situations where actual boundary conditions cannot be accurately measured or predicted. Adjustable particle weights facilitate the simulation of systems with large gradients in particle densities or species concentrations. Adaptive gridding and result collection allow for accurate and efficient simulation of complex geometries.

3.2.1 Adaptive Boundary Conditions

Adaptive boundary conditions are used in situations where actual boundary conditions cannot be accurately measured or predicted. The primary application of this technique is the simulation of plasma reactors, as described in Section 1.2. In these cases, the pressure at one position in the reactor is known, but the exact nature of the exhaust boundary condition cannot be accurately measured. The properties of the

outflow surface are related to the pumping rate, or exhaust pressure, of the reactor. Changing the reflectivity on a surface affects the pressure throughout a reactor. It is therefore useful to adaptively adjust the outflow boundary condition in order to maintain the desired pressure in the reactor. This approach assumes that the target pressure can be achieved at some reflectivity and that measured pressure is a monotonic function of the surface reflectivity.

Outflow surfaces are implemented as partially-reflective. The reflectivity, r, specifies the probability that a particle hitting the surface will be specularly reflected back into the domain. Particles that are not reflected exit the simulation. Consider a given a probe position $\vec{p}$, an initial surface reflectivity, r_0, a target pressure, P_t, and an estimate of $alpha = \frac{dR}{dP_m}$, the rate at which changes in reflectivity R affect the measured pressure P_m. After s steps, the pressure at $\vec{p}$, P_m is measured and the reflectivity is updated using,

$$R_{t+s} = R_t + \alpha \left(P_m - P_t \right). \tag{3.1}$$

If α and the number of steps between updating the reflectivity, s, are held constant, poor choices of α render this algorithm unsuitable. Too small a value of $|\alpha|$ will cause the system to take a long time to converge on the target value; too large a value of $|\alpha|$ will make the system unstable, causing large fluctuations in the reflectivity.

One approach is to adaptively change α, increasing its magnitude when the system is far from the target pressure, and decreasing its magnitude when near the target pressure. This method was studied in the context of reactor simulations, but it was found to be extremely sensitive to the statistical noise inherent in intermediate DSMC solutions. Using a large value of s yielded a long convergence time, while using a small value of s limited the accuracy of the solution.

Experimental study showed that a more effective approach is to adapt the number of timesteps between updating the reflectivity, s. Initially, a small value of s was used, so that the reflectivity was updated frequently and the system can rapidly approached the target pressure. When the measured pressure passed the target pressure and the

error changed sign, s is increased by a factor β_1, thereby increasing the number of steps over which the pressure was averaged, and decreasing the noise in the measured pressure. If the error retained the same sign, s was decreased by a factor β_2, thereby accelerating the convergence rate. Empirical studies have found the values $\alpha = -0.02Pa^{-1}$, $s_0 = 5$, $\beta_1 = 2$, and $\beta_2 = 0.9$ to be very effective for several geometries and target pressures.

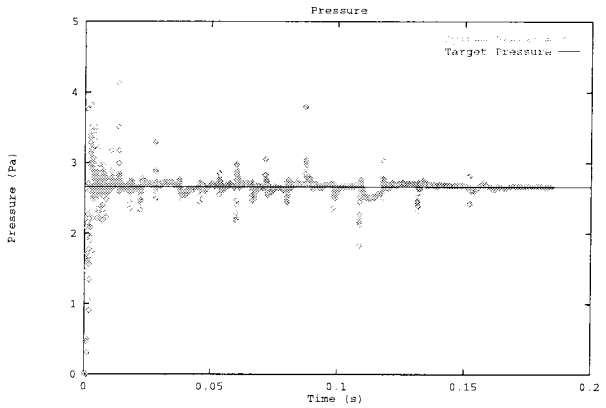

Figure 3.1: Probe Pressure as a function of simulation time

The results of applying this technique to the simulation of a plasma reactor are shown in Figure 3.1, which shows probe pressure and target pressure as functions of simulated time. During the first 0.05 seconds, oscillations well above and below the target pressure can be observed, converging to near the target pressure by 0.05 seconds. Spikes in probe pressure after this time correspond to timesteps during which macroscopic parameters were reset: after resetting parameters, the number of samples used for computing pressure is very small, and the pressure measurements thus suffer from statistical scatter. As the number of steps between changing the reflectivity changes, the statistical scatter is dampened out. By 0.15 seconds into the simulation, the pressure has converged to the target pressure.

While this approach has been developed for adapting boundary reflectivity for obtaining a desired pressure, the same approach could be used for other purposes as well. Some possibilities include adapting inflow temperature to obtain a desired wafer temperature, and adapting freestream flow speed past a spacecraft in order to obtain

a desired shock wave thickness.

3.2.2 Adjustable Particle Weights

The basic DSMC algorithm, as described in Chapter 2, assumes that each particle in the domain represents the same number of real particles. For a number of applications, however, it can be useful for the weight of a particle, or the number of real particles that it represents, to vary as a function of position, time, and chemical species.

The accuracy and quality of a solution in any portion of the grid is determined by the number of particles per cell. In some situations, smoothness and accuracy are more important in some parts of the domain than in others. In a spacecraft reentry calculation, for example, it is important to have good smoothness in the region of the bow shock, but less important in regions far away from the body. Similarly, in a plasma reactor simulation, smooth results may be more important near the wafer than in the exhaust or inflow regions. This problem is particularly important in axisymmetric calculations where the volume of cells near the axis of rotation becomes very small, while the behavior in this region is critical to the entire simulation.

An effective approach to addressing these constraints is to use different particle weights in different parts of the grid, or spatially-varying particle weights. In these cases, two weights are used, a global weight, w_g, and a local weight, w_i, for each region i. The particle weight used for collision and boundary condition calculations is the product of these two, $w_p = w_g w_l$. A complication arises when particles move between adjacent cells with different local weights. When a particle moves to a cell with half the weight of its starting cell, it must be replicated, or *cloned*: a new particle is introduced with the same position and velocity as the original particle. When a particle moves to a cell with twice the weight of its starting cell, it must be removed from the domain with a 50% probability. In general, a particle moving from a region with local weight w_1 to a region with local weight w_2 must become, on average, w_1/w_2 particles.

The disadvantage of this procedure is that the relative velocity between two cloned

particles is zero, which biases the collision process. In the extreme case, if all particles in a cell are clones of a single incoming particle, the relative velocity between any pair of particles will be zero, and no collisions will occur. It is therefore more accurate to use a number of transitions with small changes in local weights than to use one transition with a large local weight change. When it is necessary to include large weight changes in short distances, it may be effective to perturb the velocities of cloned particles by a random thermal component determined by the local cell temperature.

For steady-state simulations, these complexities can be avoided by using different timesteps in regions with different particle weights. No particle cloning or deletion is necessary when the particle flux is conserved. The particle flux is given by the ratio of the local particle weight to the local timestep. The constraint can therefore be written, $w_1/\Delta t_1 = w_2/\Delta t_2$. This approach, however is inconsistent with the concept of a global system time, and cannot be used for unsteady calculations or simulations using adaptive boundary conditions such as those described in Section 3.2.1.

The other situation in which adjustable particle weights may be beneficial is the simulation of systems involving large differences in concentration of different gas species, or when *trace species* are present. In some simulations, the flow of trace species through the system is much more important than the flow of the background species. This is the case for some reentry calculations, for which the production of small quantities nitric oxide (NO) determines the radiative emissions that may be the desired simulation results. It is also true for ions in plasma simulations. If trace species are given the same weights as background species, simulations with a sufficient number of trace particles to produce reasonably smooth results will require a prohibitively large number of background particles.

One approach for efficiently providing accurate results for trace species is to use species-dependent particle weights[13]. In this case, a particle's weight for boundary and collision processes is given by the global weight, w_g, local weight, w_l, and species weight, w_s, $w_p = w_g w_l w_s$. The number of test collisions between two species is then computed as if both species had the greater weight, and the post-collision relative velocity of the particle with lower weight is only updated with a probability given

by the ratio of the lesser weight to the greater weight. Because this technique only conserves energy on average, it can introduce random walk-type errors [21]. A more sophisticated technique that conserves energy more closely has been presented in [21].

A final use for varying particle weights is for the purpose of rapid convergence. At the beginning of a simulation, a small number of particles may be used in order to rapidly find a coarse solution. Each particle can then be replicated, and with the greater number particles, a more accurate solution can be found. This procedure can be repeated several times, until a further increase in the number of particles has no effect on macroscopic properties. During the averaging phase of a steady-state simulation, the number of particles can be further increased in order to obtain smooth results with many particles per cell.

3.2.3 Grid Adaption

In the DSMC method, error is related to the ratio of the mean free path to the cell size. If the cell size is too large, steep gradients cannot be reproduced. If the cell size is too small, computational resources are wasted, and statistical scatter is increased. The mean free path is a macroscopic parameter computed during a simulation. Since it cannot be determined *a priori*, it is impossible to generate an optimal grid before simulations have been conducted.

The implementation considered for this study uses an adaptive gridding technique to ensure simulation accuracy without wasting cells. The initial computational grid is generated so as to accurately define the geometry with the largest cells possible. Periodically during the simulation, the ratio of mean free path to cell size, $K = \lambda/l$ is computed in each cell. Cells with K too small are marked for adaption so long as they contain a sufficient number of particles.

The adaption of tetrahedral grid cells is performed using standard methods drawn from finite element analysis [53], as shown schematically in Figure 3.2. When a cell is marked for adaption, an edge in that cell is selected. A new point is introduced along this edge, each cell that shares the edge is split into two new cells, and each

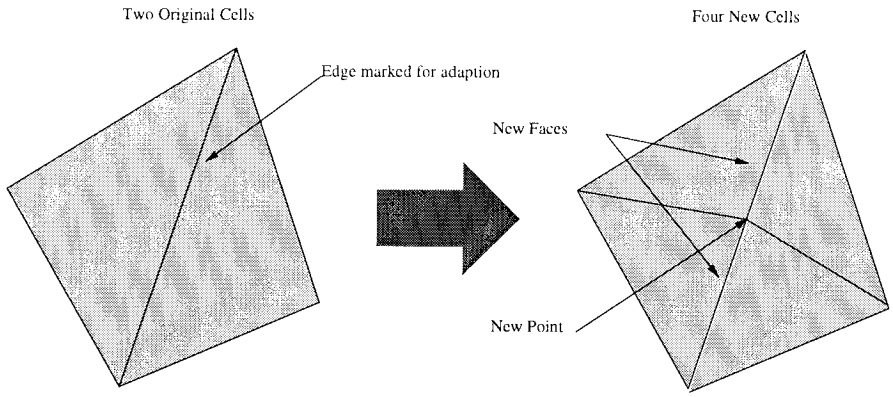

Figure 3.2: Local adaption of tetrahedral grid cells, shown in two dimensions

face that shares the edge is split into two new faces. Repeated adaption of the same edges would result in increasingly skewed grid cells. By always selecting the longest edge in a cell for adaption, however, repeated adaption is not likely to increase grid skew.

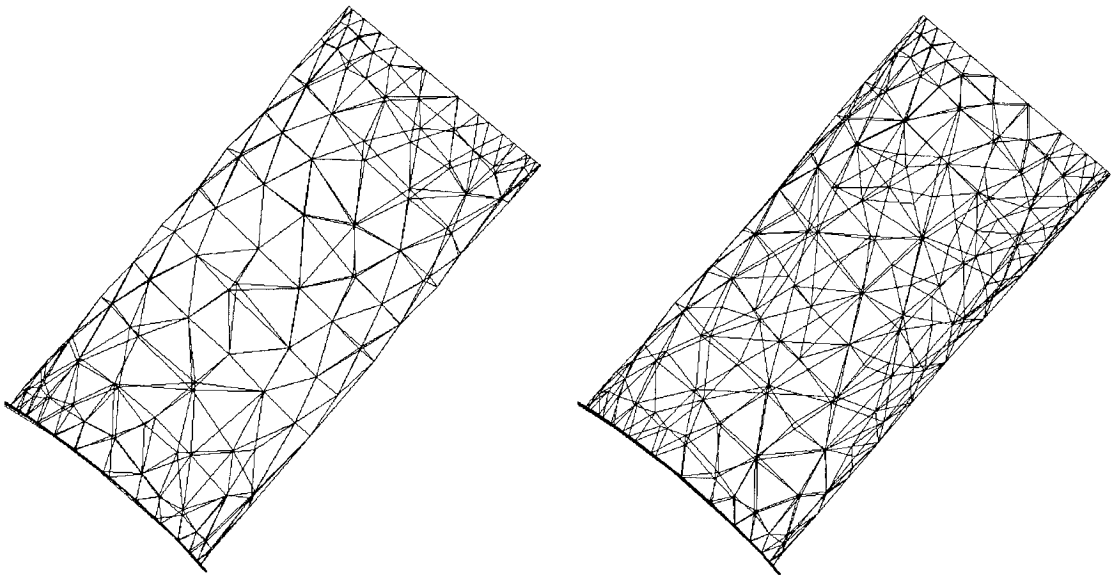

Figure 3.3: Grid for the inflow port of the GEC grid before (left) and after (right) grid adaption

Figure 3.3 (left) shows the grid for a small port in the GEC reactor (See Section 1.2) that is configured with an inflow surface at its end (top right). The particle density in these cells is relatively high, therefore the local mean free path is shorter

than the typical cell size and adaption is necessary. Figure 3.3 (right) shows the same section of the grid after several iterations of the adaption process. Note a decrease in the average cell size, and therefore an increase in the number of cells after adaption has taken place.

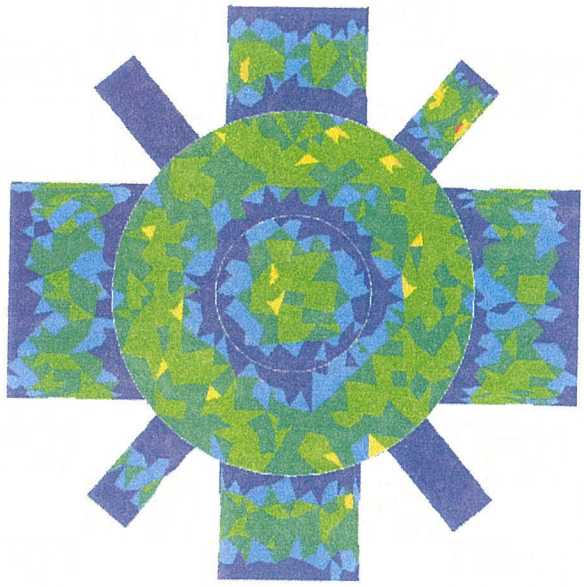

Figure 3.4: Grid adaption level in the GEC Reference Cell Reactor

Figure 3.4 shows a horizontal cut through the center of the reactor, with cells are colored according to their adaption history. Regions that have not been adapted are shown in dark blue, while brighter colors indicate increased levels of adaption. Cells shown in red are the result of 5 levels of adaption, and are therefore 32 times smaller than the original grid cells from which they were derived. The port in the upper right is the inflow port pictured in the previous figures. Note that the inflow port has been extensively adapted, while the other similar ports have only been adapted through one or two levels. Elsewhere in the reactor, the density is fairly uniform, so grid cells have adapted to be of roughly uniform size. It is clear from these pictures that such a complicated adaption pattern could not be easily generated by hand. Automatic grid generation and adaption therefore appreciably reduce the overall time to complete these simulations.

3.2.4 Adaptive Result Collection

In tetrahedral simulations involving complex geometries, it is sometimes possible to have regions of the grid with cell sizes that are much smaller than required by the collision algorithm. In these areas, the number of particles per grid cell is small and results suffer from a high degree of statistical scatter. This can also have an interesting effect on computed average particle speed. If the average, or stream, particle speed in one of these areas is small compared to the thermal speed, any computed speed will be purely due to statistical fluctuations. Since the fluctuations may be higher in regions of unnecessarily small grid cells, speed results in these areas may be incorrectly appear to be higher than speeds in other areas.

This problem can be addressed with an adaptive technique for gathering statistics. The principle behind this technique is that if two adjacent cells both have too few particles for good statistics, there can be no statistically significant difference between their macroscopic parameters. If the two cells combine their particles for the purpose of statistics collection, and both use the same resulting average, smoother results can be obtained without loss of accuracy. Note that cells are only combined for the purpose of results calculation; collisions and transport for the cells can take place as before.

The algorithm works as follows. Each cell initially belongs to a results group that contains just one cell. Any cell with a results group that has fewer than a specified number of particles (10, for example) is marked for merging. A marked cell is merged with the closest of its neighboring cells that is also marked. A hierarchy of results groups is formed. Using the hierarchy, it is possible to reverse the merges. For example, if particle doubling takes place and some cells then have enough particles on their own, they may decide to break off from their neighbors.

The effectiveness of this technique was demonstrated using simulations of the GEC reactor. Figures 3.5 and 3.6 each show scalar values along a line above the wafer in the GEC cell. Each plot shows the results using the original statistics method as well as the results using adaptive statistics. In Figure 3.5, the original results show two dips

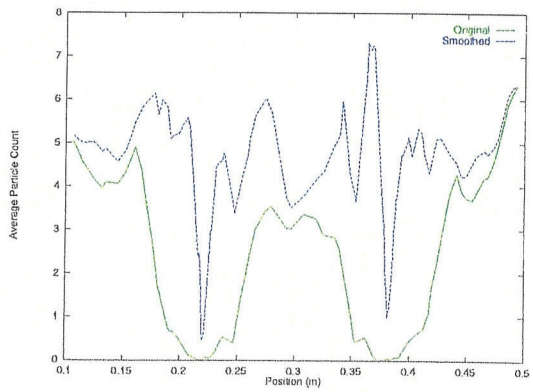

Figure 3.5: Number of particles used for calculating macroscopic parameters with and without adaptive results collection

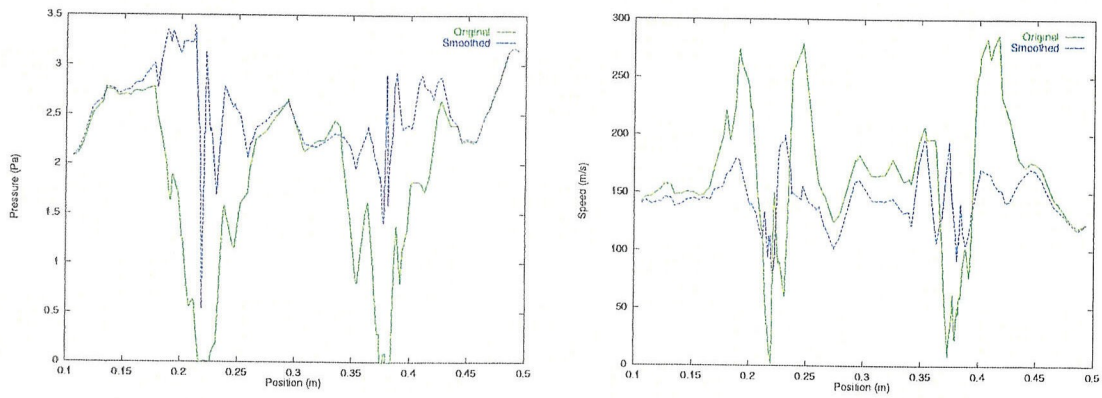

Figure 3.6: Pressure (left) and speed (right) along lines through a reactor simulation, showing results both with and without adaptive results collection

in particle count, corresponding to opposite edges of the wafer. At two points, the particle count goes all the way to zero. In the adapted results, however, particle count is always greater than zero, always greater than the original results, and sometimes by a significant amount. The effect of increased particle count is shown in Figures 3.6 (left and right). These show much smoother results after adaption, with about 50% less fluctuation in most cases. In Figure 3.6 (right) speed was close to zero where the average particle count was zero, and was very high in the regions of low particle count. After adaption, the speed was fairly uniform across the wafer, without large spikes or drops.

3.3 Software Engineering Considerations

As with any large software projects, software engineering considerations are important for DSMC implementations. This section presents some specific techniques for the engineering of DSMC implementations designed for large-scale simulations of complex geometries.

The most important consideration is software *modularity*. Components of an implementation must be distinct and their interfaces clearly defined. Transport, collision, and boundary models must also be isolated. With the use of modular components of the collision phase of the algorithm, it is easy to implement new collision models without impact on the other phases of the algorithm. A modular transport phase makes it possible to quickly switch between different gridding systems where necessary. The addition of new boundary models can also be achieved without modification to the collision and transport models.

The data structures in a DSMC implementation lend themselves to *object orientation*. The key objects are particles, grid cells, cell faces, and partitions. Particle properties include mass, position, velocity, and internal energy, and particle methods include transport and collisions. Cell structures include lists of particles and a set of macroscopic parameters. Operations on cells include transport and collision of contained particles, as well as grid adaption and the calculation of macroscopic properties. Cell face properties include temperature and boundary type, an cell faces can interact with particles or be split during grid adaption.

Figure 3.7 shows the layers in the software hierarchy used for the present work. The two lowest-level modules are the Structures Library and the Scalable Concurrent Programming (SCP) Library. The Structures Library is used for the manipulation of hash tables and for vector algebra. The SCP Library is used for file I/O. The Grid Library maintains geometrical and grid information using the hash tables from the Structures Library, and performs file I/O using the SCP Library. The application, in turn, uses the grid manipulation routines from the Grid Library, file I/O routines from the SCP Library, and the vector and hash operations from the Structures Library.

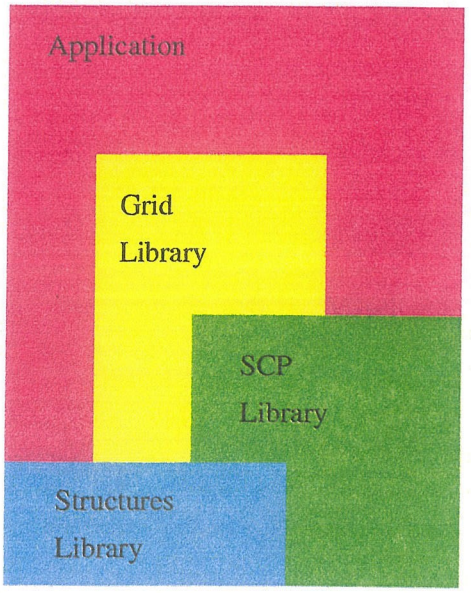

Figure 3.7: Layering of Software Components

3.4 Related Work

The optimization of DSMC simulations for workstation architectures is presented in [26]. Efficient processor utilization is demonstrated using cell-based particle lists, similar to those in the current work, in contrast to performance obtained using traditional global particle arrays. While global particle arrays are appropriate for vector computers, the resulting random access of array elements yields poor cache utilization. In this implementation, spatially-varying particle weights and particle cloning are used for axisymmetric simulations. Quasi-interactive control of simulations is achieved by rereading the configuration files periodically during a simulation. An optimized technique for selecting collision pairs is also presented. Two-dimensional grids containing mixtures of structured and unstructured cells can also be used with this implementation.

Some efforts have addressed grid generation and adaption with an iterative manual approach[95]. An initial grid is used to obtain estimates of the mean free path throughout the grid. Based on visual inspection of these estimates, the user identifies problem areas and generates a new grid, with increased or decreased resolution where necessary. This process is repeated until an acceptable solution is found. This

approach has two significant drawbacks. First, it requires time-consuming manual intervention for non-trivial geometries, and second, the use of repeated simulations dramatically increases the total amount of computational time required for a final solution. Another approach is to run a simplified simulation on a Cartesian grid, and then use the simulation results to guide the generation of a grid used for a full simulation [26]. While this approach requires less manual intervention, it still requires substantially more time for a complete simulation.

DSMC grid adaption methods for hexahedral grids are presented in [106]. This approach shifts cell vertices based on calculation of local flow gradients. Adaption constraints driven by local mean free path are presented, similar to those used in this thesis. This work also discusses the effects of statistical scatter on the grid adaption process. While it is a purely sequential implementation, the technique could be extended for concurrent execution with only nearest-neighbor communication. The disadvantages, however, are that the number of grid cells remains constant and adaption only takes place along one axis of the grid. In regions of extensive adaption or complex geometry, this approach is inadequate.

Another approach to grid adaption is to use a hierarchical Cartesian grid, with mismatched cell faces. This approach increases the complexity required for the transport phase, but can be performed locally. It can also be used for directional adaption, intentionally increasing the aspect ratio of cells in response to local gradients in the macroscopic properties of the system. In any case, solution-driven run-time grid adaption is essential for accurate and efficient simulation of complex geometries.

With the use of a hierarchical octree-type grid [47], adaptive result collection can be implemented by computing macroscopic parameters at higher levels of the grid hierarchy than the cells used for collision computations. If a cell has too few particles for smooth results, it can merge its results with the other cells at the same level of the hierarchy, producing a single common value one level higher in the hierarchy. This approach is equivalent to the tetrahedral results-merging approach, except that a single hierarchy is maintained for both grid adaption and for results merging.

The software engineering of DSMC implementations has been considered by Par-

sons [77]. His approach uses object-oriented and multi-agent systems paradigms, and he presents results for two-dimensional object-oriented implementations. The structure of this implementation has many features in common with that of the present work. A more complete software engineering approach is presented by Dietrich and Boyd [26]. This approach stresses software maintainability and the interchangeability of different physical and chemical models. Separate libraries are developed for the DSMC kernel, a geometry model for grid operations and particle transport, and a physics library for collisions. This approach is similar to that used in the present work, though its applicability to simulation techniques other than DSMC has not been demonstrated.

3.5 Summary

For large-scale simulations to be feasible on industrial timescales, attention must be focused on minimizing required simulation time. Optimizations to each model of the DSMC method have been presented in this chapter. This chapter has also presented a number of extensions to the basic DSMC technique. These extensions are necessary for ensuring efficient and accurate simulations for a variety of physical situations. Optimizations and extensions to the physical simulations such as the DSMC method revolve around the interaction between physical, physical, and computational properties of the method. Each of these properties must be studied in detail in order to provide a solution appropriate for problems of industrial relevance.

Chapter 4 Sequential Performance Model and Analysis

This chapter considers the computational requirements of DSMC simulations in terms of the physical parameters of the systems that are being simulated. Predictive models for simulation time and storage requirements are presented. These models are independent of the simulation architecture, gridding and implementation techniques, and transport, collision, and boundary models. The effects of the dimensionality and flow-configuration on performance and memory usage requirements of simulations are shown. The use of the full model defines the range of physical simulations that are feasible with existing computational resources.

There are several applications of these models. For existing DSMC applications, they facilitate the understanding of how changes in simulation parameters affect changes in computational requirements. They may also be used to compare DSMC to other techniques for a given problem, and they provide a straightforward method for determining whether a desired simulation is feasible, given available time, processing, and memory constraints. Finally, they can be used for automatic selection of simulation parameters including timestep, adaption criteria, and the ratio of real to simulated particles.

These models are developed for the basic DSMC algorithm, making no *a priori* assumptions about the nature of the solution. As such, they represent upper bounds or conservative estimates of the computational requirements for solutions. These models show that certain classes of simulations may require extremely long runtimes and large amounts of memory. In practice, simulations may be possible in shorter times and in less memory. This is only possible, however, by violating the model assumptions, which are the basic assumptions of the DSMC method. In any case, these models present a framework for understanding the computational complexity of

the DSMC method. As specific optimizations are developed for reducing the computational requirements of simulations, the basic models may be extended accordingly.

4.1 Computational Complexity Analysis

In order to understand the computational complexity of the DSMC algorithms presented in Chapter 2, it is essential to consider the interactions between physical, chemical, and computational aspects of a simulation. Superficial algorithmic analysis alone is not only inadequate, but is in fact misleading. Because the transport phase of a timestep is essentially a loop over all particles, one might expect that the time required for this phase is proportional to the number of particles, which is proportional to the particle density of a simulation. Similarly, since N particles can have N^2 possible interactions, one might expect the cost of the collision phase to be proportional to the square of the number of particles and therefore the square of the particle density. As convincing as these arguments may sound, the following sections demonstrate that the actual dependence of time requirements is *completely different.*

Internal-flow systems, such as plasma reactors, are typically characterized by the particle density, the size of the simulated domain, and the collision cross section. For certain external-flow systems, however, the domain size is not necessarily specified, and may be adjusted according to density and cross section parameters. For simulations of unsteady flows, it is also important to consider the duration of physical time for which the system must be simulated, as well as the characteristic oscillation time of the system.

4.1.1 Simulation Parameters

In order to predict the performance cost of the collision and transport phases of a DSMC computation, it is useful to calculate some general system parameters, such as the required number of cells, C, the simulation timestep, Δt, and the total number of required timesteps, S. The particle density and collision cross section determine the mean free path and therefore the required size of computational grid cells, while size

of the simulated domain determines the number of cells that are required and the time required for information to travel across the system. The number of cells required for a simulation can be determined from the DSMC constraint that the typical cell size, or characteristic cell length, $\bar{l}$, be proportional to the mean free path, λ,

$$\bar{l} = c_\lambda \lambda = \frac{c_\lambda}{n\sigma}, \tag{4.1}$$

where c_λ is a proportionality constant, n is the particle number density (particles per unit volume) and σ is the collision cross section. The number of cells along each axis of the grid is proportional to the size of the domain along that axis. Given a characteristic domain size, L, the total number of cells required for a D-dimensional simulation, C, is,

$$C = \left(c_v \frac{L}{\bar{l}}\right)^D = \left(\frac{c_v}{c_\lambda} n\sigma L\right)^D, \tag{4.2}$$

where c_v is a constant that reflects the type of grid and the skewedness of grid cells.

The number of particles required for a simulation, N, is chosen to be proportional to the number of cells required, $N = c_p C$. The number of particles per cell, c_p, must be large enough to allow for a sufficient number of collisions per cell, and to provide adequate samples for statistics, as discussed below. Using the value of C from (4.2) gives,

$$N = c_p C = c_p \left(\frac{c_v}{c_\lambda} n\sigma L\right)^D. \tag{4.3}$$

A typical three-dimensional system may contain a number of particles comparable to Avogadro's number. As it is computationally infeasible to simulate this number of particles, it is necessary for each simulated particle to represent a large number of real particles. For one- and two-dimensional simulations, however, the area or thickness of the simulated domain can be chosen such that each simulated particle represents one real particle. Let h^{3-D} represent the "free" dimensions of the grid, so that the total simulated volume is $V = L^D h^{3-D}$. The *weight* of each simulated particle, or the number of real particles represented by each simulated particle, w_p, can be written in

terms of the number of cells, using (4.2) and (4.3),

$$w_p = \frac{nV}{N} = \frac{nL^D h^{3-D}}{c_p C} = \frac{c_\lambda^D}{c_p c_v^D}\left(\frac{h^{3-D}}{n^{D-1}\sigma^D}\right). \tag{4.4}$$

The total amount of simulation time required for a steady-state simulation to converge is governed by the acoustic time: the amount of time that it takes for thermal information to traverse the entire width of the simulated region, L. This is determined by the thermal speed, $v_t = \sqrt{\frac{8kT}{\pi m}}$, where k is the Boltzmann constant, T is the gas temperature, and m is the particle mass. Assuming that c_a acoustic periods are required for convergence, the acoustic, or convergence time T_{conv}, is given by,

$$T_{conv} = c_a \frac{L}{v_t}. \tag{4.5}$$

The simulation timestep, Δt, should be chosen so that the average distance traveled by a particle in a timestep, d, is some fraction, c_t, of the average cell length, $\bar{l}$. If particles traverse too many cells in one timestep, the collision process will not be correctly captured. On the other hand, too small a timestep will result in inefficient computation. The accuracy of DSMC solutions improves as the timestep and cell size approach zero. The average distance traveled by a particle in a timestep is simply the product of the average total velocity $\overline{v_{total}}$ and the timestep Δt. The total velocity is composed of a thermal component, v_t, and a stream component, $\overline{v}$, and can be approximated as their sum, $\overline{v_{total}} \approx \overline{v} + v_t$. Using these approximations, with (4.5) and (4.2), yields,

$$\Delta t = c_t \frac{\bar{l}}{v_{total}} = c_t c_V \left(\frac{1}{\overline{v} + v_t}\right)\left(\frac{L}{C^D}\right). \tag{4.6}$$

Using the value of C from (4.2), this can be rewritten,

$$\Delta t = \frac{c_t c_\lambda}{\left(\overline{v} + v_t\right) n\sigma}. \tag{4.7}$$

It is important to note that an increase in any of the parameters, n, σ, T, or $\overline{v}$,

results in a decrease in the timestep duration. This in turn results in an increase in the number of steps required for a simulation and therefore the simulation time. The number of steps required for convergence, S_{conv}, is the ratio of the acoustic time, T_{conv}, to the timestep, Δt. Using (4.7),

$$S_{conv} = \frac{T_{conv}}{\Delta t} = \frac{c_a}{c_t c_\lambda}\left(1 + \frac{\overline{v}}{v_t}\right) n\sigma L. \tag{4.8}$$

In addition to considering the time required for a simulation to converge, it is also important to examine the tradeoff between execution time, memory usage, and solution quality. One measure of the quality of a solution is determined by the noise or statistical scatter. The statistical scatter is determined by the number of *samples*, or the product of the average number of particles in a cell, c_p, and the number of steps over which macroscopic properties are averaged, S_s. Assuming that the scatter follows a Poisson distribution, the fractional error, e, is inversely proportional to the square root of the number of samples, N_s [15],

$$e = \frac{1}{\sqrt{N_s}} = \frac{1}{\sqrt{c_p S_s}}. \tag{4.9}$$

In order to obtain r samples per cell, it is necessary to average results over $\frac{r}{c_p}$ steps. In the following sections, these parameters are used to compute the amount of time required for the transport and collision phases of a DSMC timestep.

4.1.2 Transport Phase

In order to move a particle for one timestep, each component of the particle's position must be updated, as must each component of the particle's velocity. Since the number of components is equal to the dimensionality of a simulation, the number of operations required to move a particle should be roughly proportional to the number of dimensions in which it is moved. In this case, the time required for the transport phase of a timestep, T_{trans}, is given by the product of time required to move one particle in one dimension, T_t, the number of dimensions, D, and the number of particles,

N,

$$T_{trans} = T_t DN = T_t D c_p C. \tag{4.10}$$

The parameter T_t is dependent upon both the machine speed and the implementation of the transport model. Using the value for N computed in (4.3), this can be written,

$$T_{trans} = T_t DN = T_t D \frac{c_p c_v^D}{c_\lambda^D} (n\sigma L)^D . \tag{4.11}$$

4.1.3 Collision Phase

The time required to compute collisions in a DSMC timestep is proportional to the number of collisions. Consider a computational cell of volume V_{cell} that contains c_p particles. Using the Hard Sphere (HS) collision model, the number of collisions in that cell during a given timestep, Δt, is given by,

$$N_c = \frac{c_p(c_p - 1)\sigma v_t w_p}{V_{cell}\sqrt{2}} \Delta t, \tag{4.12}$$

where σ is the collision cross section and v_r is the relative velocity between particles. Using the timestep duration from (4.7) and the particle weight computed in (4.4), as well as the average cell volume, $V_{cell} = \frac{L^D h^{3-D}}{C}$, yields,

$$N_c = \frac{(c_p - 1)c_t c_\lambda}{\sqrt{2}} \left(\frac{v_t}{\overline{v} + v_t} \right). \tag{4.13}$$

Since each component of a particle's velocity must be updated in a collision, the process of computing post-collision relative velocities is roughly proportional to the number of dimensions. Assuming that the total collision time, T_{col}, is then also proportional to the number of dimensions, it can be written as the product of the time spent on each dimension of a collision, T_c, the number of dimensions, D, the number of collisions per cell, N_c, and the number of cells, C. Combining (4.13) and (4.2) yields,

$$T_{col} = T_c D N_c C = T_c D \frac{(c_p - 1)c_t c_v^D}{\sqrt{2} c_\lambda^{D-1}} \left(\frac{v_t}{\overline{v} + v_t} \right) (n\sigma L)^D . \tag{4.14}$$

As with transport time, collision time is proportional to the D-th power of density, cross section, and the size of the domain. This analysis was developed for the HS model, where σ is a constant, while for the Variable Hard Sphere (VHS) model, σ is a function of relative velocity. Thorough analysis of the cost of the VHS model could be completed with integration over relative velocities, and the result, would have roughly the same dependence on σ^D. The extension to the Variable Soft Sphere (VSS) model does not affect the number of collisions, only the post-collision scattering angle. Because more computation is required with each collision, this would have the effect of increasing T_c, but would not change the dependence on the other parameters, n, σ, L, and D.

4.1.4 Timestep Duration

The preceding sections enable the prediction of the time required for a single timestep of a simulation. Combining Eqs. (4.11) and (4.14) yields the total time required for one timestep is given by the sum of transport and collision times,

$$T_{one} = T_{trans} + T_{col} = \frac{c_v^D}{c_\lambda^D}\left[c_p T_t + \frac{c_t(c_p-1)c_\lambda}{\sqrt{2}}\left(\frac{v_t}{\overline{v}+v_t}\right)T_c\right] D n^D \sigma^D L^D. \qquad (4.15)$$

Note that this analysis does not depend on the type of grid used or the implementation of the transport or collision phases. All particle transport algorithms must execute in time proportional to the number of timesteps and the number of particles, and all collision algorithms must execute in time proportional to the number of collisions, thus yielding the same dependence on the physical parameters n, σ, D, and L, and the constants c_p, c_a, c_v, c_t, and c_λ. Only the parameters T_t and T_c will vary between implementations and architectures.

4.1.5 Memory Requirements

In addition to modeling the required execution time, it is also important to predict the storage requirements for a simulation. The two primary uses of memory are

particles and cells. For small simulations, it is also important to consider the amount of *overhead* memory, M_0, consumed by the application code and any fixed-size data structures. As a rough approximation, the amount of memory required to store a particle or a grid cell is proportional to the number of dimensions of the simulation, D. If the memory required for each dimension of a single particle is M_p and the memory required for each dimension of a single cell is M_c, the total memory required for a simulation, M, can be written,

$$M = M_0 + M_p DN + M_c DC = M_0 + M_p D c_p C + M_c DC = M_0 + (M_p c_p + M_c) DC. \quad (4.16)$$

4.2 Flow Configurations

This section considers the four possible flow configurations: steady-state internal, steady-state external, unsteady internal and unsteady external. The computational requirements of each of these configurations have fundamentally different dependences on the physical parameters. In order to provide a complete understanding of the complexity of the DSMC method, it is essential to consider each configuration separately.

4.2.1 Steady-State Internal Flows

In order to compute a steady-state flow, a simulation is first run to convergence and then run for additional steps in order to sample and average macroscopic parameters. The total simulation time is therefore the sum of convergence and averaging times. The time required to converge an internal-flow, steady-state simulation, $K_{steady}^{internal}$, is the product of the time for each timestep and the number of steps required. Combining (4.15) and (4.8) yields,

$$K_{steady}^{internal} = T_{one} S_{conv} = \frac{c_v^D c_a}{c_\lambda^D} \left(\frac{(c_p - 1) T_c}{\sqrt{2}} + \frac{c_p T_t}{c_t c_\lambda} \right) D \left(n \sigma L \right)^{D+1}. \quad (4.17)$$

The duration of the averaging portion of a steady computation, $A_{steady}^{internal}$, is governed by the desired accuracy or number of samples. Averaging time, for a desired

number of samples, r, is the product of the time required for each timestep, T_{one}, and the number of steps required, S_s,

$$A_{steady}^{internal} = T_{one} S_s = T_{one} \frac{r}{c_p}. \tag{4.18}$$

The averaging time can then be written,

$$A_{steady}^{internal} = \frac{c_v^D}{c_\lambda^D} \left[T_t + T_c \frac{c_t c_\lambda}{\sqrt{2}} \left(1 - \frac{1}{c_p} \right) \left(\frac{v_t}{\overline{v} + v_t} \right) \right] D \left(n \sigma L \right)^D r. \tag{4.19}$$

The time required to obtain smooth results is thus proportional to the D-th power of the density, cross section, and domain size. It is also proportional to the the desired number of samples, or the inverse square of the acceptable statistical scatter, e.

For a steady-state, fixed-volume simulation, the memory requirements from (4.16) reduce to,

$$M_{steady}^{internal} = M_0 + (M_p c_p + M_c) \frac{c_v^D}{c_\lambda^D} D n^D \sigma^D L^D. \tag{4.20}$$

The storage requirements for a simulation are therefore proportional to the cube of the density, the cube of the cross section, and the size of the simulated domain.

4.2.2 Steady-State External Flows

For certain classes of external flow problems, it is appropriate to adjust the size of the computational domain according to the other physical parameters. As a first approximation, it is sometimes possible to use a computational domain with length proportional to the mean free path, λ,

$$L = c_d \lambda = \frac{c_d}{n \sigma}, \tag{4.21}$$

where c_d is the number of mean free paths to be simulated. The convergence time for an external flow can then be written,

$$K_{steady}^{external} = \frac{c_d^D c_v^D c_a}{c_\lambda^D} \left(\frac{(c_p - 1) T_c}{\sqrt{2}} + \frac{c_p T_t}{c_t c_\lambda} \right) D. \tag{4.22}$$

The convergence time is therefore not a function of cross section or density. Similarly, the averaging time for an external flow, $A_{steady}^{external}$, for a desired number of samples, r, can then be computed as,

$$A_{steady}^{external} = \left(\frac{c_v c_d}{c_\lambda}\right)^D \left[T_t + T_c \frac{c_t c_\lambda}{\sqrt{2}} \left(1 - \frac{1}{c_p}\right) \left(\frac{v_t}{\overline{v} + v_t}\right)\right] Dr. \tag{4.23}$$

The memory requirements of a steady external simulation can be written,

$$M_{steady}^{external} = M_0 + \frac{c_d^D c_v^D}{c_\lambda^D} (M_p c_p + M_c) D. \tag{4.24}$$

Just as simulation time is not a function of cross section or density for this class of problems, storage requirements are not functions of cross section or density. For some important systems, it is possible to adjust the domain size with the mean free path, but not in a directly proportional manner. For these cases, the computational requirements may be estimated by using the analysis of this section together with the analysis of the previous section.

4.2.3 Unsteady Internal Flows

For an unsteady problem, the total simulated time is a specified parameter, not determined by convergence time. Consider an unsteady simulation of a time interval T_u, with a characteristic oscillation time τ. The ratio T_u/τ is thus the number of periods to be simulated. Because the flow is changing, it is not possible to average results over a large number of steps. There are two ways to obtain smooth results for unsteady flows: the first is to choose a number of particles such that results averaged over a small number of steps will be sufficiently smooth; the second is to run a number of simulations with a small number of particles, using a different random seed for each. The results from the different simulations can then be averaged together to obtain smooth results. While both methods require approximately the same amount of simulation time, the first method requires significantly more memory. The following analysis assumes that P separate simulations are used, where $P = 1$ corresponds to

the first approach, and $P > 1$ corresponds to the second approach.

For unsteady flows, results can only be averaged over a short period of time during which the flow remains approximately unchanged. The number of particles per cell, c_p, must be chosen so that the desired number of samples can be obtained while the flow is unchanged. It must be assumed that the flow is unchanged over some (small) fraction of τ. Sampling can then take place during a time $c_\tau \tau$. The number of steps over which it is possible to average is given by,

$$S_u = \frac{c_\tau \tau}{\Delta t}. \tag{4.25}$$

The number of samples, r, obtained with P separate simulations, is the product of particles and steps,

$$r = c_p S_u = c_p P \frac{c_\tau \tau}{\Delta t}. \tag{4.26}$$

Equation (4.26) can be solved to determine the minimum number of particles required per cell, p_m,

$$p_m = \frac{r \Delta t}{c_\tau \tau P}, \tag{4.27}$$

which can be rewritten using (4.7),

$$p_m = \frac{c_t}{c_\tau c_\lambda} \left(\frac{1}{\overline{v} + v_r} \right) \frac{r}{P n \sigma \tau}. \tag{4.28}$$

The computational time required for P unsteady simulations is given by the product of the time taken for each timestep, T_{one}, and the number of steps that must be simulated, S_u. This, in turn, is the ratio of the unsteady time, T_u, to the timestep, Δt,

$$A^{internal}_{unsteady} = T_{one} \frac{T_u}{\Delta t}. \tag{4.29}$$

Using previously computed values, and the approximation $c_p - 1 \approx c_p$, this yields,

$$A^{internal}_{unsteady} = \frac{c_p c_v^D}{c_t c_\lambda^{D-1}} \left[T_t + T_c D \frac{c_t}{c_\lambda} \left(\frac{v_r}{\overline{v} + v_t} \right) \right] (\overline{v} + v_t) D n^{D+1} \sigma^{D+1} L^D T_u. \tag{4.30}$$

Substituting the minimum number of particles p_m for c_p in (4.30) yields,

$$\begin{aligned} A_{unsteady}^{internal} &= \frac{p_m c_v^D}{c_t c_\lambda^{D-1}} \left[T_t + T_c \frac{c_t}{c_\lambda}\left(\frac{v_r}{\overline{v}+v_t}\right)\right](\overline{v}+v_t) D n^{D+1} \sigma^{D+1} L^D T_u \\ &= \frac{c_v^D}{c_t c_\lambda^D c_\tau}\left[T_t + T_c \frac{c_t}{c_\lambda}\left(\frac{v_r}{\overline{v}+v_t}\right)\right] \frac{D(n\sigma L)^D r T_u}{\tau} \end{aligned} \tag{4.31}$$

This shows that for unsteady simulations with a given oscillation period, τ, the computation time is proportional to the D-th power of density, cross section, and domain size, and proportional to the number of desired samples and the simulated time, but inversely proportional to the oscillation time. The cost of unsteady simulations therefore does not grow as fast as the cost of fixed-size steady simulations, primarily for the reason that unsteady simulations are not required to converge.

For typical values of n, σ, L, and τ, however, the number of particles required for an unsteady simulation is very much greater than the number required for a steady simulation. The time and memory requirements of unsteady calculations are therefore substantially greater than for steady calculations. In some cases, the initial conditions may be sufficiently uncertain or complicated that a steady simulation must be converged to determine those conditions before an unsteady computation can begin, further increasing the cost of unsteady simulations.

In order to estimate the memory requirements for each unsteady internal flow simulation, the value of p_m from (4.28) can be used in (4.20) to obtain,

$$M_{unsteady}^{internal} = M_0 + \frac{c_v^D}{c_\lambda^D}\left[M_p \frac{c_t}{c_\tau c_\lambda}\left(\frac{1}{\overline{v}+v_r}\right)\frac{n^{D-1}\sigma^{D-1} r}{\tau P} + M_c n^D \sigma^D\right] D L^D. \tag{4.32}$$

4.2.4 Unsteady External Flows

The simulation time for an unsteady external flow computation can be obtained by reducing Eq. (4.31) to,

$$A_{unsteady}^{internal} = \frac{c_v^D c_d^D}{c_t c_\lambda^D c_\tau}\left[T_t + T_c \frac{c_t}{c_\lambda}\left(\frac{v_r}{\overline{v}+v_t}\right)\right]\frac{r D T_u}{\tau}. \tag{4.33}$$

Similarly, memory requirements can be obtained by reducing Eq. (4.32) to,

$$M_{unsteady}^{external} = M_0 + \frac{c_d^D c_v^D}{c_\lambda^D}\left[M_p \frac{c_t}{c_\tau c_\lambda}\left(\frac{1}{\overline{v}+v_r}\right)\frac{r}{n\sigma\tau} + M_c\right] D. \tag{4.34}$$

4.3 Parameter Estimation

This section considers the parameters required for predicting runtime and storage requirements for DSMC simulations. These parameters can be grouped in two classes: those that are implementation-dependent or architecture-dependent, and those that are not. The former can only be discussed in the context of a specific implementation, while the latter should be common among all DSMC implementations.

4.3.1 General DSMC Parameters

The implementation-independent parameters are summarized in Table 4.1. In general, particles should not traverse more than about one cell per timestep, so c_t should be less than one. In fact, c_t can be thought of as the inverse of the local Knudsen number, with typical values in the range, $0.3 < c_t < 1$. The ratio of the cell size to the mean free path, c_λ, should be less than one. Adaptive gridding systems, such as the one described in 3.2.3 ensure that this constraint is met. For such approaches, values for c_λ are typically between 0.3 and 1.

In order to understand the convergence time of a simulation, it is important to consider the grid shape and boundary conditions. For a spherical grid with a uniform external boundary, information at the boundaries will quickly propagate throughout the domain. On the other hand, the simulation of a long curved tube with different boundary conditions at opposite ends will require a long time to converge. In order to find typical values for the number of acoustic periods required for convergence, c_a, a series of simulations was conducted. Simulations were conducted with different geometries, numbers of cells, and initial conditions. For each simulation, the time required for macroscopic parameters to reach steady state was measured and divided by the predicted acoustic time. Typical values of c_a were found to be in the range

Table 4.1: General DSMC Parameters

Parameter	Description	Typical Values
c_t	Fraction of typical cell length traveled by typical particle in one timestep	0.3 - 1
c_λ	Ratio of cell length to local mean free path, or minimum local Knudsen number	0.3 - 1
c_a	Acoustic periods required for convergence	3 - 10
c_p	Ratio of particles to cells	3 - 10
c_v	Measure of grid skewedness	1 - 5
c_d	Number of mean free paths to be simulated for external flow	10 - 1,000
c_τ	Fraction of the oscillation period over which samples can be averaged	0.01 - 0.3

from 3 to 10.

The number of particles per cell, c_p, must be large enough that a reasonable number of collisions will take place in each cell. Using larger values of c_p also reduces statistical scatter. On the other hand, both runtime and memory usage are proportional to c_p. For steady-state simulations, c_p is typically chosen between 3 and 10. Some advanced statistical techniques have been used to produce reasonably accurate results with only about one particle per cell [46].

The parameter c_v represents the ratio of the typical cell dimension to the cube root of the cell volume, and is primarily grid-dependent. For typical tetrahedral grid cells, $c_v \approx 2$, while hexahedral cells have slightly smaller values of c_v. For skewed grid cells, c_v can be arbitrarily large.

The choice of the number of mean free paths that must be simulated, c_d, is largely problem-specific, but representative simulations use values between 10 and 1000. Similarly, the choice of c_τ, the fraction of the oscillation period over which the flow is considered unchanging, is problem-specific. For a sinusoidal oscillation, however, $c_\tau = 0.1$ is a reasonable value, providing 10 separate results for each oscillation period.

4.3.2 Implementation-Specific Parameters

The parameters T_t, T_c, M_p, and M_c, are both implementation- and architecture-specific. For illustrative purposes, typical values were obtained for a three-dimensional DSMC implementation, *Hawk*, designed for plasma reactor and spacecraft reentry calculations[79]. Other DSMC implementations have different associated constants, but must obey the same dependences on the physical parameters.

Table 4.2: Implementation-Specific Parameters

Parameter	Description	Typical Values
T_t	Time required to move one particle for one timestep in one dimension	22 μs
T_c	Time required to perform one dimension of one collision	13 μs
M_0	Memory required for overhead	2.96 MB
M_p	Memory required per particle, per dimension	18 B
M_c	Memory required per cell, per dimension	494 B

Table 4.2 summarizes the implementation-dependent parameters obtained for three-dimensional simulations on a Silicon Graphics Power Challenge with 75-MHz R8000 processors. In order to measure transport time, simulations were conducted with the collision phase disabled. Similarly, collision time was measured on simulations with particle transport disabled. A value of $c_p = 10$ was used, and the measured values were $T_t = 11\mu s$ per dimension and $T_c = 13\mu s$ per dimension. Note that a larger value of c_p will increase the number of particles being simulated, for the same amount of per-cell overhead, and result in smaller values for T_t and T_c.

The overhead memory, M_0, was estimated for *Hawk* by running a simulation with only 12 cells, and with no particles. On the SGI Power Challenge, this value was approximately $M_0 = 2.97$ MB. The majority of this is consumed by the program image alone; only a fraction of a Megabyte is used for the actual data structures. The memory usage per particle has a lower bound of 2 floating point values per dimension, one for position and one for velocity. Most implementations, however, use

additional storage space for storing additional per-particle data structures that help to reduce runtime. The particle memory usage in *Hawk* was estimated by running neutral flow simulations with varying numbers of particles and recording the memory usage reported by the operating system, then subtracting the overhead memory M_0 and dividing by the number of particles. This yielded an an approximate value, $M_p = 18$ bytes per dimension. It must also be noted that simulations using more sophisticated chemistry models may require additional memory to store, for example, internal energy or species information.

Per-cell memory requirements are likely to vary more between DSMC implementations. In *Hawk*, each cell stores several values for unstructured grid information, a pointer to a linked list of particles, local information used for collisions, and macroscopic parameter information. Many of these data structures are irregular and dynamic in nature, with sizes that depend on the nature of the problem, and may even change during a computation. For the computations discussed here, cell sizes were typically around $M_c = 494$ bytes per dimension.

4.4 Predictive Modeling

In order to illustrate the application of the model to the prediction of actual simulation requirements, several internal flow simulations were considered. The same techniques, however, are applicable to external flow simulations. In order to assess the accuracy of the performance prediction model, *Hawk* simulations were conducted on five three-dimensional box grids, each with a different number of cells. Execution time per timestep and memory usage were measured. The execution time per timestep was predicted using Equation (4.15) and the parameters in Section 4.3.

Figure 4.1 plots predicted and measured step times as functions of the quantity $n^D \sigma^D L^D$. For each simulation, the predicted step time is within 8% of the measured step time. The differences are largest for the small grids (low values of $(n\sigma L)^D$), which can be attributed to the effects of computational overhead and set-up time. The linear dependence of step time on $(n\sigma L)^D$, where $D = 3$ in this case, is clearly

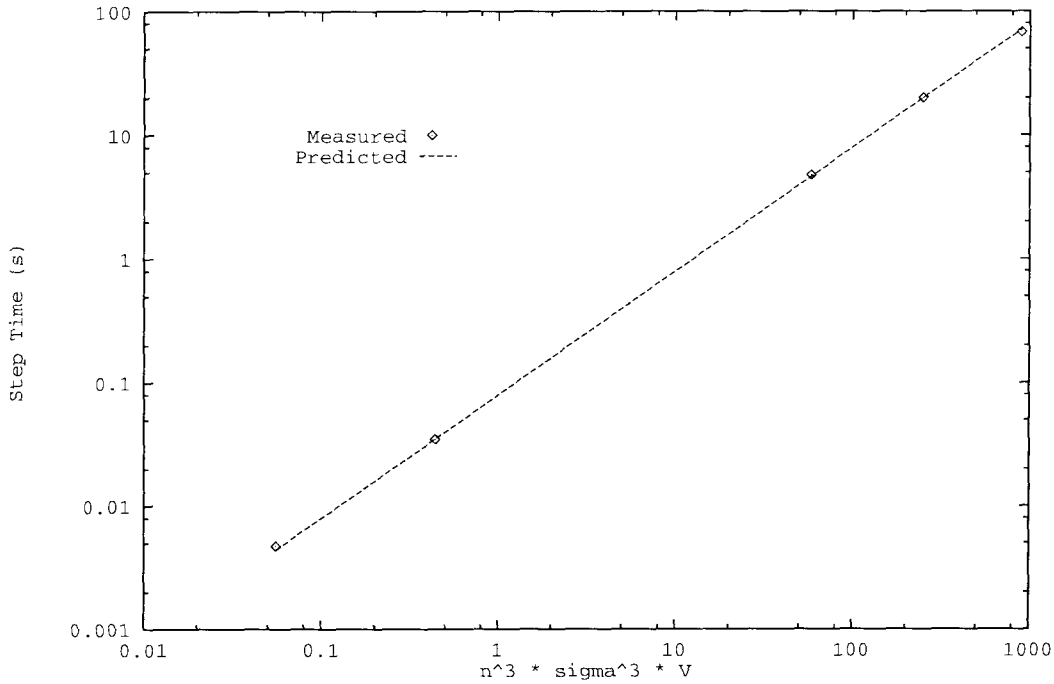

Figure 4.1: Predicted and measured step time as a function of physical parameters

demonstrated by this experiment.

Figure 4.2 shows the predicted and measured memory requirements for the same simulations. For the larger simulations, memory usage is proportional to the quantity $(n\sigma L)^D$, while for small simulations, the overhead memory, M_0, is the dominant term. These results show excellent agreement between predictions and measurements. The difference between predictions and measurements is consistently less than 4%.

4.5 Large-Scale Simulations

While the preceding experiments were performed on simple box grids, the analysis still holds for complex three-dimensional geometries. As an example of realistic simulations of industrial relevance, simulations of a plasma reactor, the Gaseous Electronics Conference (GEC) Reference Cell, were considered. This reactor has a complex three-dimensional geometry, with a volume of $0.013m^3$, and typically operates at a

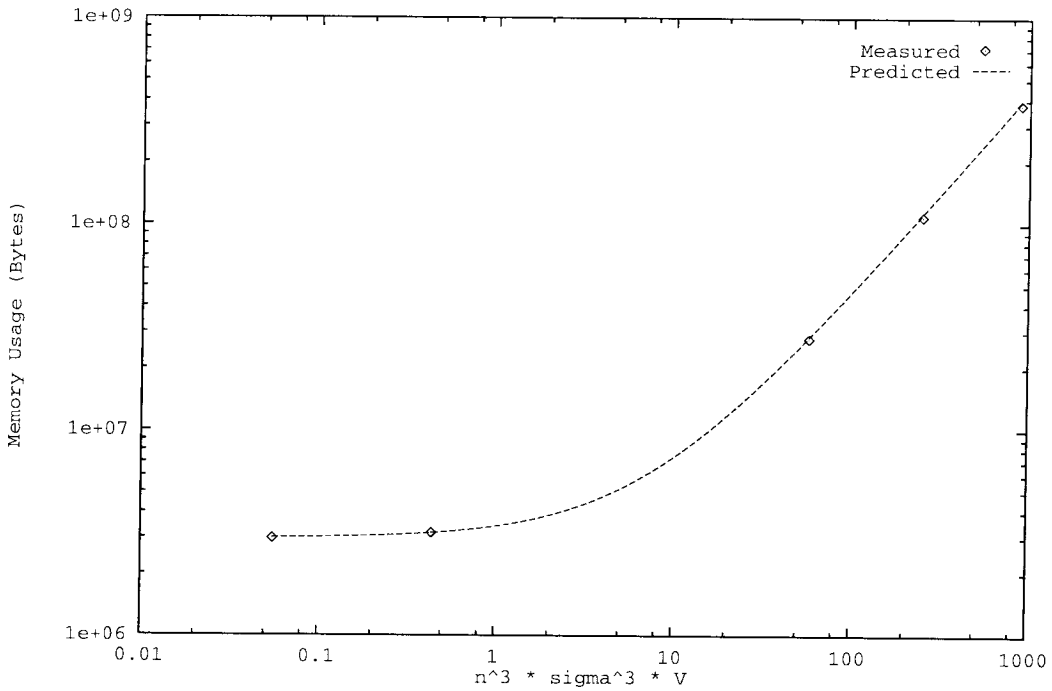

Figure 4.2: Memory usage as a function of physical parameters

temperature of $300K$. For the purpose of these experiments, the reactor was simulated with argon gas at constant temperature and pressure throughout, using the Variable Soft Sphere (VSS) collision model with $\alpha = 0.31$, $\beta = 0.714$, $T_{ref} = 273K$, and $\sigma_{ref} = 5.307 \times 10^{-19} m^2$. The reactor walls and the wafer were all thermally accommodating at 300 K.

Using the model and constants developed above, runtime and storage requirements were predicted for simulations of the GEC cell at several densities, or pressures. Table 4.5 lists predictions for the number of cells, number of particles, timestep duration, convergence time, and memory usage, for three-dimensional simulations at four different operating pressures. These values were obtained using the machine-specific parameters for the 75-MHz R8000 SGI Power Challenge. For all but the 0.291 Pa configuration, the convergence time, T_{conv}, is much greater than the averaging time, A, and is therefore a close lower bound on the total time required for a simulation.

In order to assess the applicability of the model to realistic three-dimensional

Table 4.3: GEC Simulation Predictions

Pressure (Pa)	.291	2.66	6.65	13.3
Pressure (mTorr)	2.19	20	50	100
Density (m^{-3})	7.0×10^{19}	6.4×10^{20}	1.61×10^{21}	3.21×10^{21}
Cells	1.4×10^{5}	1.1×10^{8}	1.68×10^{9}	1.3×10^{10}
Particles	1.4×10^{6}	1.1×10^{9}	1.68×10^{10}	1.3×10^{11}
T_{one} (sec.)	51.7	3.9×10^{4}	6.2×10^{5}	4.9×10^{6}
T_{conv} (sec.)	4.0×10^{3}	2.8×10^{7}	1.1×10^{9}	1.8×10^{10}
Mem (Bytes)	2.88×10^{8}	2.17×10^{11}	3.42×10^{12}	2.71×10^{13}

geometries, the first case, at 0.291 Pa, was configured and simulated on an SGI Power Challenge. Using the model and parameters above, memory usage for this simulation was predicted to within 3%. Because the model does not take into account the (implementation-specific) additional cost of moving particles in the high grid-density regions, the model underpredicted the timestep time by about 25%. In general, the model can be expected to provide an accurate estimate of memory usage, and a reasonable lower bound for simulation time, for realistic simulations.

For the other simulations listed in Table 4.5, the higher operating pressure results in larger computational costs, both in terms of simulation time and storage requirements. For three dimensional simulations, $D = 3$ so the time complexity of the algorithm is proportional to the *fourth* power of pressure, and the memory requirements are proportional to the *cube* of pressure. For a simulation at 2.66 Pa to be conducted to the same accuracy, 22 GB of RAM would be required, and the convergence portion of the simulation would take 327 days.

A simulation at 6.65 Pa would require 35 years on a single-processor machine with 3 TB RAM. For the 13.3 Pa case, a single SGI Power Challenge would require 27 TB of RAM, and convergence would take 570 years. Note, however, that these are conservative estimates, or upper bounds. They apply when no *a priori* knowledge of the flow is available. In other words, they represent the minimum amount of time required to guarantee a correct solution.

In practice, accurate simulations at these high pressures have been possible in

much shorter timescales. These are achieved by violating the assumptions of the preceding performance models. The high complexity of the algorithm is primarily a result of the large numbers of cells required in order to maintain a cell size proportional to the local mean free path. This constraint is required to ensure that all gradients of macroscopic parameters can be correctly captured. In regions of the domain where no significant gradients exist, it is often possible to use much larger cells than dictated by mean free path constraints. When fewer cells are required, fewer particles are required, and a larger timestep may be used. The net result can be a substantial reduction in computational requirements for a simulation.

Because a small sacrifice in the accuracy of a simulation may result in a substantial decrease in computational requirements, simulations may be possible in much less time and memory than predicted by the models in this chapter. Recall from Chapter 3 that the number of particles per cell can be adjusted as a function of both position in the grid and species. These techniques are used to increase accuracy in regions of interest and for species of interest, while obtaining faster solutions in regions of less interest. Such optimizations, however, are consistent with the model. Changing the ratio of cell size to local mean free path simply changes a constant in the model. The fundamental dependence on the physical parameters of the system is unchanged. Further, the development of this model reveals opportunities for additional optimizations to the basic DSMC technique.

4.6 Related Work

Early work on computational complexity was presented in [1, 6, 17]. Knuth's books [55] are excellent references on the subject. As an example of analysis of the computational complexity of another numerical technique, the computational complexity of diffusion-limited aggregation and fluid invasion in porous media is presented in [62].

While runtimes for various simulations have been presented, no model for the computational complexity of the DSMC method has previously been presented.

4.7 Summary

The simulation execution time for the different configurations is summarized in Table 4.4, while the storage requirements for the different configurations are summarized in Table 4.5.

Table 4.4: Summary of Simulation Times for Different Configurations

Simulation Type	Simulation Time
Internal steady	$\frac{c_v^D c_a}{c_\lambda^D}\left(\frac{(c_p-1)T_c}{\sqrt{2}}+\frac{c_p T_t}{c_t c_\lambda}\right) D\left(n\sigma L\right)^{D+1}$ $+\frac{c_v^D}{c_\lambda^D}\left[T_t+T_c\frac{c_t c_\lambda}{\sqrt{2}}\left(1-\frac{1}{c_p}\right)\left(\frac{v_t}{\overline{v}+v_t}\right)\right] D\left(n\sigma L\right)^D r$
External steady	$\frac{c_d^D c_v^D c_a}{c_\lambda^D}\left(\frac{(c_p-1)T_c}{\sqrt{2}}+\frac{c_p T_t}{c_t c_\lambda}\right) D$ $+\frac{c_v^D c_d^D}{c_\lambda^D}\left[T_t+T_c\frac{c_t c_\lambda}{\sqrt{2}}\left(1-\frac{1}{c_p}\right)\left(\frac{v_t}{\overline{v}+v_t}\right)\right] Dr$
Internal unsteady	$\frac{c_v^D}{c_t c_\lambda^D c_\tau}\left[T_t+T_c\frac{c_t}{c_\lambda}\left(\frac{v_r}{\overline{v}+v_t}\right)\right]\frac{D(n\sigma)L^D r T_u}{\tau}$
External unsteady	$\frac{c_v^D c_d^D}{c_t c_\lambda^D c_\tau}\left[T_t+T_c\frac{c_t}{c_\lambda}\left(\frac{v_r}{\overline{v}+v_t}\right)\right]\frac{DrT_u}{\tau}$

Table 4.5: Summary of Memory Requirements for Different Configurations

Simulation Type	Memory Requirements
Internal Steady	$M_0+(M_p c_p+M_c)\frac{c_v^D}{c_\lambda^D} D n^D \sigma^D L^D$
External Steady	$M_0+\frac{c_d^D c_v^D}{c_\lambda^D}(M_p c_p+M_c)D$
Internal Unsteady	$M_0+\frac{c_v^D}{c_\lambda^D}\left[M_p\frac{c_t}{c_\tau c_\lambda}\left(\frac{1}{\overline{v}+v_r}\right)\frac{n^{D-1}\sigma^{D-1}r}{\tau}+M_c n^D\sigma^D\right] DL^D$
External Unsteady	$M_0+\frac{c_d^D c_v^D}{c_\lambda^D}\left[M_p\frac{c_t}{c_\tau c_\lambda}\left(\frac{1}{\overline{v}+v_r}\right)\frac{r}{n\sigma\tau}+M_c\right] D$

The results of this chapter show that the runtime and memory requirements of DSMC simulations can be accurately predicted on the basis of physical properties and machine-specific parameters. The DSMC algorithm is fundamentally polynomial in the physical parameters, and the degree of the polynomial is primarily determined by the number of dimensions being simulated. For this reason, the cost of a simulation increases dramatically with the number of dimensions used. Three-dimensional simulations require both substantially more simulation time and substantially more memory. Computational requirements are most sensitive to density, or pressure. For

a three-dimensional steady-state simulation, doubling the simulation pressure results in an *eight-fold* increase in the amount of memory required, and a *sixteen-fold* increase in the time to convergence. This places a serious constraint on the class of problems that can be approached with the DSMC method.

When considering the applicability of the DSMC method to a specific problem, it is essential to consider the runtime and storage requirements for the simulation. For certain high-density or large-volume problems, these requirements may be prohibitive. By comparing the requirements of the DSMC method with the requirements of other methods, it is possible to determine the best approach for each specific problem. It is also possible to predict how changes in physical parameters will affect runtime and storage requirements, and thereby to determine bounds on the class of problems that can be solved with the DSMC technique, given finite computational resources.

Chapter 5 Concurrent Algorithms

Realistic computational grids can often be too large to fit in the memory of any single processor, and realistic simulations may require extremely long execution times on sequential computers. In order to be able to provide accurate solutions in realistic time frames, it is important for a DSMC implementation to be able to make use of all available computational resources, including shared-memory multiprocessors, distributed-memory multicomputers, and heterogeneous networks of workstations.

Fortunately, the DSMC method is well suited to implementation on concurrent architectures. This chapter discusses the implementation of the DSMC method on concurrent architectures, yielding the Concurrent DSMC method. Techniques are presented for domain decomposition, task mapping, and inter-processor communication. Novel methods of dynamic load balancing and automatic granularity control are also presented.

5.1 Concurrent DSMC

The concurrent DSMC method considered for the present study is based on the spatial decomposition of the computational grid.

The concurrent simulation technique is outlined in Figure 5.1. A three-dimensional geometry definition is taken directly from CAD/CAM descriptions already available to process engineers. An unstructured tetrahedral grid is then constructed using automatic grid generation techniques[43]. This grid is subsequently partitioned for execution on any of a variety of concurrent architectures including heterogeneous networks workstations and PC's, shared-memory multiprocessors, and distributed-memory multicomputers. Scalable concurrent algorithms are then used to reduce the numerical simulation time. Adaptive gridding is used to automatically maintain the accuracy of the simulation. Dynamic load balancing is used to maximize processor

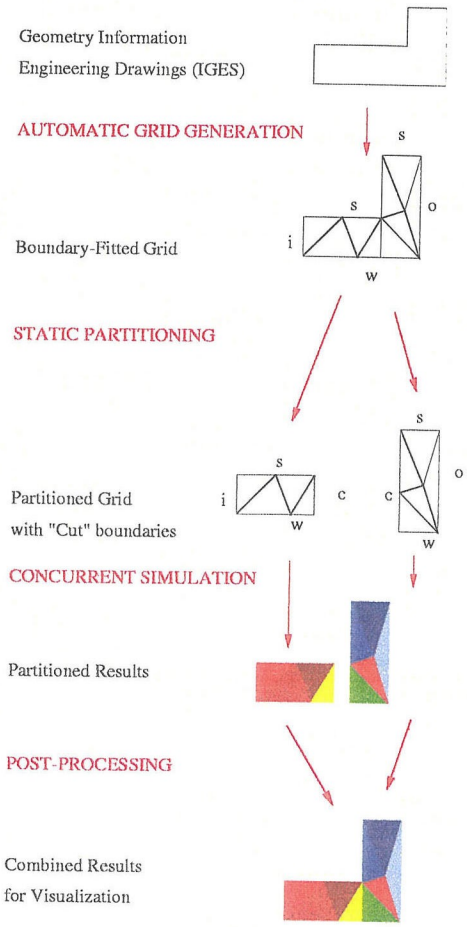

Figure 5.1: Concurrent DSMC methodology

utilization in the presence of both grid adaption and dynamic flow variations. Finally, simulation results are analyzed using standard CFD visualization tools [37].

The concurrent simulation technique for the distributed-grid approach involves four primary stages:

1. **Grid Generation.** Tetrahedral grids are automatically generated using a three-dimensional geometry definition taken directly from CAD/CAM descriptions.

2. **Static Partitioning.** The grid is partitioned for execution on concurrent architectures.

3. **Concurrent DSMC.** Scalable concurrent algorithms are then used to reduce the numerical simulation time. Adaptive gridding automatically maintains the

accuracy of the simulation, and dynamic load balancing maximizes processor utilization.

4. **Post-Processing.** After a simulation is complete, results from the partitions are combined for visualization using industry- standard tools.

Concurrent DSMC requires three modifications to the sequential DSMC algorithm in Program 2.1. First, the transport phase of the algorithm must be adapted to support communication between partitions of the grid. Second, global information, such as the total number of particles in the domain, must be computed using gather-scatter style communication. The time required to execute one timestep for a single partition is primarily dependent on the number of particles that it contains. As the number of particles varies between partitions, and changes during the course of a simulation, dynamic load balancing is required for obtaining good parallel efficiency. Dynamic granularity control is also used to adjust the number of partitions per processor in order to provide sufficient granularity for the purpose of load balancing while minimizing the overhead of multiple partitions.

Program 5.1 outlines the algorithm executed concurrently the grid partitions. For the most part, particle transport is local within a partition. A particle may, however, move across a cell face on the boundary between two partitions. In this case, it is communicated to the appropriate neighboring partition. Once all of the particles have been placed in their new cell locations, the collision phase is executed independently by each partition. At the end of a timestep, global information is computed via global communication operations. The computational efficiency is maintained through concurrent load balancing and granularity control algorithms that attempt to keep all processors equally busy.

5.1.1 Partitioning

A computational grid must be decomposed into multiple partitions in order to provide sufficient granularity for concurrent execution. This involves three phases of the simulation cycle. First, a grid is *initially partitioned* into one or more partitions for

```
ConcurrentDSMC()
{
  for(partition = 0; partition < NUM_PARTITIONS; partition++)
                                        /* Concurrently */
  {
    Initialize(partition);
    for(step = 0; step < NUM_STEPS; step++)
    {
      PartitionTransport(partition);          /* Locally */
      ExchangeParticles(partition);           /* Local communication */
      PartitionCollisions(partition);
      ComputeGlobalProperties(partition);     /* Global communication */
      if(LoadImbalance() )
        BalanceLoad(partition);
    }
    ComputeMacroscopicParameters(partition);
  }
}
```

Program 5.1: Concurrent DSMC Algorithm

each computer on which the computation will be conducted. Depending on the quality of the initial partitioning, it may be necessary to perform *static partition balancing* in order to provide balanced partitions for initial execution. During a simulation, an executing partition may be *dynamically repartitioned* if load balancing requires additional granularity.

Initial partitioning is typically controlled by the user, who specifies the size and shape of the computational network on which the simulation will be executed. On one-dimensional networks, such as a network of workstations, the user only needs to specify the number of computers and the axis along which the grid will be partitioned. On two-dimensional networks, such as that of the Intel Paragon, the user must specify the two partitioning axes and the dimensions of the network. On three-dimensional networks, such as that of the Cray T3D, the user must specify the three dimensions of the network to be used.

Initial partitioning is based on a volume decomposition that preserves *locality*. Cells are sorted in each dimension and grouped according to their sorted positions.

Adjacent cells are therefore likely to be mapped to the same partition. This procedure attempts to minimize the surface area of the partitions, and hence the total amount of communication required during a simulation. It also attempts to minimize the number of neighbors for each partition and therefore the number of separate communications required. Figure 5.2 shows the GEC grid divided into 24 partitions, with one cut in each horizontal direction, and two in the vertical direction. Some partitions cannot be seen because of the large number of cells in the center of the reactor. While the middle partitions contains less volume than the top and bottom partitions, each contains roughly the same number of cells.

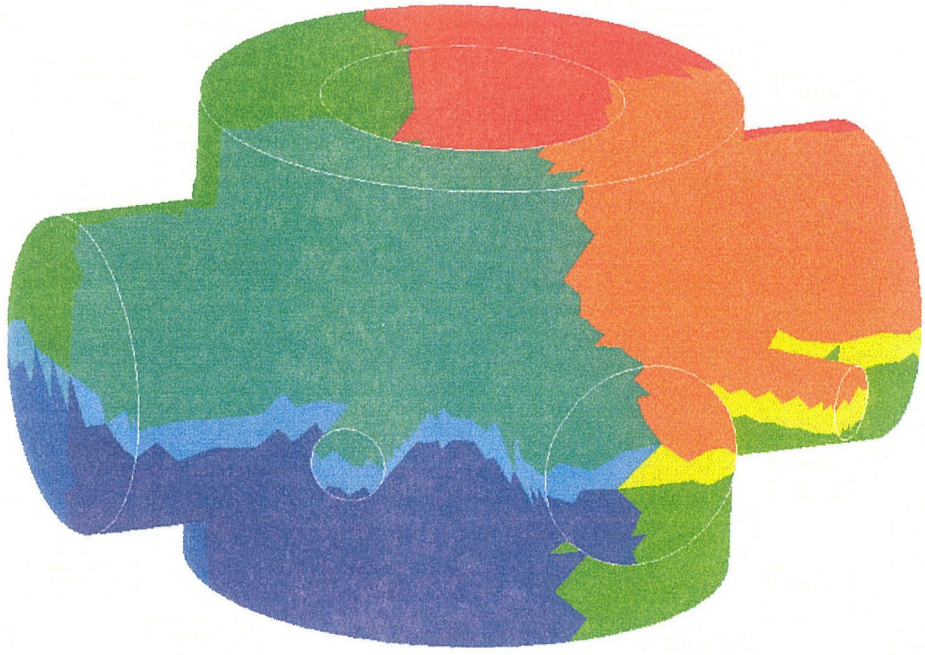

Figure 5.2: The grid for the GEC reference cell after initial partitioning

5.1.2 Mapping

The mapping strategy is guided by the principle of *over-partitioning*, whereby several partitions are mapped to each processor. This is shown schematically in Figure 5.3. Over-partitioning allows partitions of differing sizes to be mapped to the same computer in order to balance load and memory. Specifying the number of computers, and therefore the total number of cells per computer, provides control over the *granularity*

of the computation, i.e., the ratio of computation to communication. Generally speaking, this corresponds to the ratio of the volume (L^3 for a characteristic dimension L) to the surface area of all partitions in a computer (L^2). Adjusting the granularity allows the simulation to be matched to a wide variety of concurrent architectures. On platforms with high communication costs, such as networks of workstations, a small number of large partitions may be used. By contrast, a larger number of smaller partitions may be used on distributed-memory multiprocessors that employ fast communication technology.

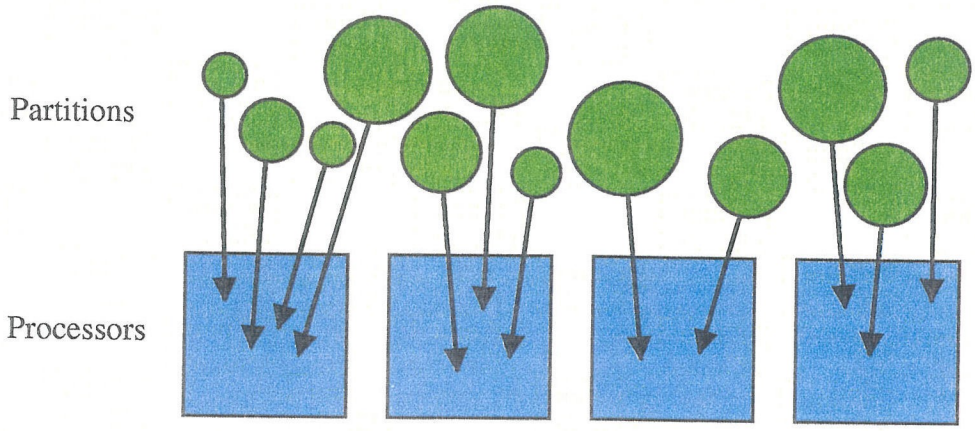

Figure 5.3: Over-partitioning for mapping multiple partitions to each processor

Partitions are mapped to computers in such a way as to place neighboring partitions in the same or adjacent computers, again maximizing locality. Information is exchanged between partitions on neighboring computers using message-passing, while information is directly exchanged between partitions on the same computer or between partitions on separate computers that share memory by pointer copying. The mapping of multiple partitions to each computer allows the overlap of communication and computation: if one partition is blocked, awaiting communication, another partition on the same computer may still proceed.

5.1.3 Communication

Communication in concurrent DSMC is required when particles move between adjacent partitions. A particle arriving at a partition boundary is sent to the appropriate

neighboring partition. The time taken for the particle to reach the surface is subtracted from the current timestep to obtain the remaining time during which the particle must be moved upon arrival at the neighboring partition.

In order for a concurrent DSMC implementation to produce the same results as a sequential DSMC implementation, it must satisfy several constraints. Because the basic algorithm is unchanged, these constraints can be readily determined. The only operations that are different between concurrent and sequential implementations are those that occur on partition boundaries. Collisions, boundary surface interactions, and transport local to a partition are performed in the same manner for both approaches.

In order to guarantee that the transport algorithm is unchanged, particle communication must obey the following rules:

1. All particles must reach their final destinations before collisions can be performed.

2. A particle's final destination must be computed in the same manner as in the sequential algorithm.

3. Particle properties, including velocity, mass, and chemical species, must be preserved during transport between partitions.

5.2 Communication Optimizations

One of the most important reasons for loss of efficiency in parallel computations is the cost of communication. Optimizations to the communication methods and functions can therefore significantly reduce the time required for concurrent computations.

The most important communication optimization is to combine particles traveling between a given pair of partitions into a small number of messages. If each particle is sent separately, communication costs can become prohibitive. The best approach is to combine a given number of particles into a single message, and then send the message. The optimal number of particles to combine into a single message is an

architecture-specific parameter determined by the latency, and bandwidth, and buffer sizes of the communication system. Distributed-memory computers typically have low-latency communications with small buffers, and are most efficient with smaller numbers of particles per message than on networks of workstations with high-latency communications and large buffers.

5.3 Static Load Balancing

If a large number of partitions is required for a computation, simple volume partitioning may be inadequate. It allows the number of cells per partition to vary widely, and it allows a partition to have an undesirably large surface area. These problems can result in imbalances in processor load and memory usage, and can require more communication than is necessary. Static load balancing, based on the concept of heat diffusion, can be used to overcome these problems. Computational cells are treated as heat to be diffused and exchanged between partitions. It is achieved using a parallel algorithm to compute the solution to the heat equation [38, 80, 99],

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u, \tag{5.1}$$

where u is an estimate of the memory and computational cost of a partition and $\nabla^2 u$ is a measure of local imbalance. This technique has a variety of useful properties. It provably converges and its rate of convergence can be determined analytically [38]. It involves only nearest-neighbor communication and is therefore scalable. The algorithm is application-independent, and has been applied in the context of Particle-In-Cell (PIC) calculations by Samanta-Roy, *et. al* [87]. It has been independently applied by Ivanov, *et. al.*, to produce substantial performance improvements in another DSMC implementation [50]. The technique has also been extended to support multiphase computations [100] and to support heterogeneous networks of PC's and workstations [101]. The technique was developed by Watts [99]; this section considers its application to the partitioning of tetrahedral grids for particle simulations.

The parameter u is computed as the sum over all cells in a partition,

$$u = \sum_{cells} (c_{cell} + c_{vol} V_{cell}), \tag{5.2}$$

where c_{cell} is an estimate of the fixed cost per cell, representing the overhead cost for computations in cells with no particles. The constant c_{vol} is an estimate of the cell cost proportional to the cell volume, V_i, and represents the cost of moving the particles in the cell. Each cell is treated as a discrete quantity of heat that can be transferred between adjacent partitions in order to evenly distribute the cost. Cells are selected for transfer in such a way as to preserve locality and to minimize the area of surfaces shared between partitions. This process terminates when all partitions have roughly the same memory and computational costs.

Figure 5.4: The grid for the GEC reference cell after initial partitioning (left) and static partition balancing (right)

In order to evaluate the effectiveness of this technique, it was performed for a simulation of the GEC Reference Cell reactor. The initial grid was partitioned and then static load balancing was performed, resulting in a new partitioning. The load balance was compared for the two partitionings. Figure 5.4 shows the result of initial partitioning (left), where it is clear that the partitions are imbalanced, as well as the same grid after static volume balancing, using $c_{cell} = 0$, $c_{vol} = 1$ (right). Note that the volume of the middle partitions has grown close to that of the outer partitions.

In this case, the ratio of maximum to average partition volume was improved from 1.4 to 1.01.

5.4 Dynamic Load Balancing

Load balancing and work distribution considerations are important for any application on concurrent architectures. The computational cost of a single DSMC timestep for a grid partition is a function of the number of grid cells and particles that it contains. These quantities both change dynamically in the presence of particle transport and grid adaption. It is therefore impossible to initially partition a complex three-dimensional grid so that load will be balanced among processors throughout the computation. For this reason, no static partitioning and mapping will yield an optimal load balance for the entire duration of a computation.

Realistic large-scale simulations of complex geometries require static and dynamic load balancing, as well as automatic granularity control. This section summarizes the load balancing framework that was developed by Watts [98], and describes its application in the context of particle simulation. The framework is based on the concept of heat diffusion, which provides a scalable, correct mechanism for determining how much work should be migrated between computers, including computers with different processing capabilities or external workloads. Heat diffusion only gives the ideal work transfer, however; to meet that ideal, neighboring computers must exchange partitions. The selection of which partitions to exchange is guided by both the sizes of the partitions involved and the effect a partition's movement would have on its communication with other partitions. If there are too few or too many partitions in the system, granularity management routines are used to increase or decrease the number of partitions. The end-result is a five-step methodology for load balancing a computation [99, 100]:

1) **Load measurement:** The load of each computer is determined, either by having the programmer provide an estimate of resource needs of the partitions or by measuring their resource usage with system timing calls.

2) **Load imbalance detection and profitability calculation:** Based on the total load measured at each computer, the efficiency of the computation is calculated. Load balancing is undertaken if its estimated cost is exceeded by the estimated reduction in run time that would result from load balancing.
3) **Ideal load transfer calculation:** Using the load quantities measured in the first step, computers calculate the ideal degree to which they should transfer load to or from their neighbors.
4) **Partition selection:** Using the load transfer quantities calculated previously, partitions are selected for transfer or exchange between neighboring computers. This phase may be repeated several times until the transfer quantities have been adequately met, and it may be guided by cost functions that encourage communications locality or discourage the movement of partitions with large data structures.
5) **Partition migration:** Once the partitions have migrated to their final locations, any data structures associated with those partitions are transferred from their old locations to their new locations, and the computation resumes.

The following sections consider these steps in further detail.

Load Measurement

The usefulness of any load balancing scheme is directly dependent on the quality of load measurement and prediction. Accurate load evaluation is necessary to determine that a load imbalance exists, to calculate how much load should be transferred to alleviate that imbalance, and to determine which partitions best fit the ideal load transfer quantities. Load evaluation can be performed either completely by the application, completely by the load balancing system or with a mixture of application and system facilities.

For the purposes of practical DSMC calculations, it has proven most effective to measure load empirically. The times required for execution and communication for a single timestep for each partition are measured using system timing functions, and

the sum of these values is used for load balancing. Balancing this quantity yields the shortest timestep time after load balancing. Balancing of execution time alone, or number of particles, is almost as effective. Balancing the number of cells per computer can be effective for uniform or fully adapted simulations, but is not effective for highly non-uniform or transient flow simulations.

Load Imbalance Detection and Profitability Calculation

For load balancing to be useful, one must first determine *when* to load balance. Doing so is comprised of two phases: detecting that a load imbalance exists and determining if the cost of load balancing exceeds its possible benefits.

Even if a load imbalance exists, it may be better not to load balance, simply because the cost of load balancing would exceed the benefits of a better load distribution. The time required to load balance can be measured directly using available facilities. The expected reduction in run time due to load balancing can be estimated loosely by assuming efficiency will be increased to $eff_{\min}$ or more precisely by maintaining a history of the improvement in past load balancing steps. If the expected improvement exceeds the cost of load balancing, the next stage in the load balancing process should begin [102]. More precisely, load balancing should be undertaken when the following holds

$$(eff_{\text{cur}} < eff_{\min}) \bigwedge \left(\left(1 - \frac{eff_{\text{cur}}}{eff_{\text{new}}} \right) T_{\text{step}} > T_{\text{bal}} \right) \tag{5.3}$$

where eff_{cur}, $eff_{\min}$, eff_{new} are the current efficiency, desired minimum efficiency and expected efficiency after load balancing, respectively, T_{step} is the time until the next load balancing opportunity, and T_{bal} is the estimated time required for load balancing.

Ideal Transfer Calculation

After determining that it is advantageous to load balance, one must calculate how much load should *ideally* be transferred from one computer to another. In the interest

of preserving communication locality, these transfers should be undertaken between neighboring computers.

As with the static load balancing technique described above, ideal transfer calculation is achieved based on the solution of the diffusion equation, $\frac{\partial L}{\partial t} = \nabla^2 L$. Diffusion was also explored in [99] and was found to be superior to other load balancing strategies in terms of its performance, robustness and scalability. A more general diffusive strategy was presented in [100]; unlike previous work, this method uses an implicit differencing scheme to solve the heat equation on a multi-dimensional torus to a specified accuracy. The advantage of an implicit scheme is that the timestep size in the diffusion iteration is not limited by the number of neighbors. For explicit schemes, the timestep size is limited to $\frac{1}{2d}$ on a d-dimensional mesh or torus. In [98], an improved, second-order diffusion scheme was derived.

Partition Selection

Once load transfers between computers have been calculated, it is necessary to determine which partitions should be moved to meet those quantities. The quality of partition selection directly impacts the ultimate quality of the load balancing.

The problem of selecting which partitions to exchange to achieve a particular load transfer is weakly **NP**-complete, since it is simply the subset sum problem. Fortunately, approximation algorithms exist which allow the subset sum problem to be solved to a specified non-zero accuracy in polynomial time.

Since the selection algorithm cannot, in general, satisfy a particular load transfer in a single attempt, it is necessary to make multiple attempts. For example, in the worst-case scenario where all of the partitions are on one computer, only those computers that are neighbors of the overloaded computer can hope to have their incoming load transfers satisfied in the first round of exchanges. In such a case, one would expect that at least $\mathbf{O}(D)$ exchange rounds would be necessary, where D is the diameter of the mesh. The algorithm for partition selection is thus as follows.

The load transfers are colored in the same manner as described for the GDE

algorithm. For each color, every computer attempts to satisfy its transfer of that color, adjusting ϵ_{max} to account for the degree to which its transfers have thus far been fulfilled. The algorithm is repeated when the colors have been exhausted. Termination occurs when no more progress toward further load transfer is possible. Termination can occur earlier if all of the computers have satisfied the minimum requirement of their outgoing load transfer quantities.

A partition may move multiple hops in the process of achieving load transfers. Since the data structures for a partition may be large, this store-and-forward style of remapping may prove costly. A better method is to instead transfer a *token*, which contains information about a partition such as its load and the current location of its data structures. Once partition selection is complete and these tokens have arrived at their final destinations, the computers can send the partitions' states directly to their final locations.

Partition Migration

In addition to selecting which partitions to move, the load balancing framework also provides mechanisms for moving those partitions from one computer to another. For the purposes of this study, partition migration is achieved by transferring all partition data structures from one computer to another computer. Because of the large amount of data that must be transferred, this is typically the longest part of the load balancing operation.

5.5 Automatic Granularity Control

Just as it is impossible to partition a grid in order to provide an optimal work distribution *a priori*, it is also impossible for a static partitioning of a grid to have the optimal number of partitions per computer, or *granularity*. With too few partitions per computer, it may be impossible to accurately satisfy the work transfer quantities required for a good load balance. Similarly, a computer with too many partitions per

computer will spend an inordinate amount of time on overhead required for switching between tasks. Simulation properties, such as the number of particles and cells, change during a simulation. For this reason, even a partitioning that is initially adequate may not remain so as the simulation progresses. This section describes the application of Watts' automatic granularity control framework [99] applied to DSMC simulations.

In the transfer quantity satisfaction phase of load balancing, the partitions may be so large, or coarse-grained, that it is impossible to balance the load. It is then necessary to split the partitions into smaller partitions, resulting in a finer granularity. A partition is split if its corresponding load is greater than a certain fraction of the average load. If the division of partitions results in a better, but still inadequate load balance, the threshold is lowered so that more partitions are divided. This continues until an adequate load balance is achieved, until no benefit results from finer granularity, or until partitions can no longer be split.

When a partition must be split, the grid cells in the partition are first traversed in order to compute a bounding box around the partition. The bounding box is then divided into the desired number of new partitions, so as to minimize the surface area of the new bounding boxes. New connections are created between the newly-created partitions, and connections to neighboring partitions are updated. This process can be completed in time proportional to the number of grid cells, though it does not necessarily guarantee minimum communication or even division.

During splitting, simulation data structures must be updated accordingly. For example, counts of the numbers of particles and cells in each partition must be recomputed. Grid cells and particles are unaffected by partition splitting. Cell faces that lie on the border between the two new partitions must be replicated, and face-level data structures updated appropriately.

Dynamic repartitioning routines are invoked when additional granularity is required for a computation. Typically, the load balancing algorithm will determine which partitions need to be repartitioned, and for each one, the desired number of new partitions.

The routines for splitting a partition at runtime are the same as those for initial partitioning. The only exception is that whereas in initial partitioning the user can specify the desired partitioning parameters along each dimension, this information can not be provided at runtime. It is therefore necessary to use a *heuristic* to determine how to split a partition.

One heuristic that has proved effective is based on the assumption that the partition is shaped similar to its bounding box, and that its cells are roughly uniform in size. In this case, partitioning parameters can be easily selected in order to minimize shared surfaces and thus communication. The following recursive algorithm is used to divide a partition into p smaller partitions.

```
BoundingBoxPartition(partition, bounding_box, p)
{
  b = LargestPrimeFactor(p);
  d = LongestDimension(bounding_box);
  DivideAlongAxis(partition, bounding_box, d, b,
                  &new_partitions, &new_boxes);
  for(i = 0; i < b; i++)
  {
    BoundingBoxPartition(new_partitions[i], new_boxes[i], p/b);
  }
}
```

Program 5.2: Bounding Box Partitioning

Program 5.2 is used to recursively divide a grid partition into a desired number, p, of new partitions. The first step is to compute b, the largest prime factor of p. The longest dimension of the bounding box, d, is then determined. The bounding box and partition can then be divided into b new partitions, along the d axis. Each of the resulting partitions is recursively split into p/b new partitions

Using the assumption that the bounding boxes of child (post-division) partitions are simply related to the bounding boxes of the parent (dividing) partition, this algorithm can determine the partitioning parameters before actually performing any partitionings, and is thus executed very quickly.

It is important that the bulk of the granularity control operations are local. Com-

munication is only required for updating connections between the new partitions and their neighbors. This allows for rapid granularity adjustment even on large concurrent computers. For the simulations considered in this study, the process of load balancing and granularity control was completed in same amount of time as several simulation steps. As typical simulations require tens or hundreds of thousands of timesteps, and load balancing only takes place about once every thousand steps, this cost is negligible.

5.6 Concurrent Grid Adaption

Grid refinement, as described in Section 3.2.3, is a local process, and can therefore usually be conducted within a single partition. In cases where an adapting edge lies on a partition boundary, only the partitions sharing that boundary need to participate in the adaption. Therefore, the communication and synchronization costs of an iteration of grid adaption are minimal. Furthermore, grid adaption can be performed infrequently during a calculation without substantial loss of accuracy.

5.7 Concurrent Adaptive Result Collection

Adaptive result collection, as described in Section 3.2.4 is also a local process. Cells that combine their particles for the calculation of macroscopic parameters are likely to be in the same partition. The combination of results from cells in adjacent partitions could be implemented with only local communication, but the resulting complexity and communication costs would outweigh the advantages of combining such cells. In practice, it is sufficient to only combine cells in the same partition.

5.8 Software Engineering Considerations

The primary new object that is introduced for concurrent DSMC is the partition. Partitions are composed of collections of grid cells and cell faces, and can execute

timesteps, split for granularity control, and move between computers for load balancing.

Concurrent simulations employ the same set of application-independent libraries as sequential simulations. As described in Section 3.3, the Structures Library is used for hash tables and vectors, the Grid Library is used for grid manipulations, and the SCP Library is used for file I/O. For concurrent simulations, the Scalable Concurrent Programming Library (SCPlib), also provides basic programming technology to support irregular applications on scalable concurrent hardware. The library has been applied to a variety of large-scale industrial simulations and is portable to a wide range of platforms. On each platform, the library provides a optimized, portable set of low-level functionality, including message-passing, thread management, synchronization, file I/O, and performance monitoring.

SCPlib also provides a higher level of functionality, which includes heterogeneous communication and file I/O, load balancing, and dynamic granularity control. Communication and file I/O occur through objects called ports. These ports are similar to Unix descriptors in that the same routines can be used to write to a channel port or to a file port. This allows considerable reuse of application code, since the same routines used to read and write a data structure can be used for both communication and file I/O. Furthermore, communication through ports is typed; when a port is created, the system inputs a header describing the writer's data type sizes, byte ordering, etc. The reader can then automatically apply the appropriate transformations, if necessary. This allows communication between partitions running on heterogeneous architectures, as well as the ability to read checkpoints written on different platforms.

An SCPlib application is constructed as a concurrent graph of communicating partitions, called nodes, as shown in Figure 5.5. Each node is comprised of a thread of execution, state information, and a set of channel ports for communication with other nodes. The mapping of nodes to computers is transparent to the user, since the channels effectively hide the locations of a node's neighbors in the communication graph. The library can thus move a node dynamically, so long as the programmer provides routines for communicating a node's state. The programmer can reuse

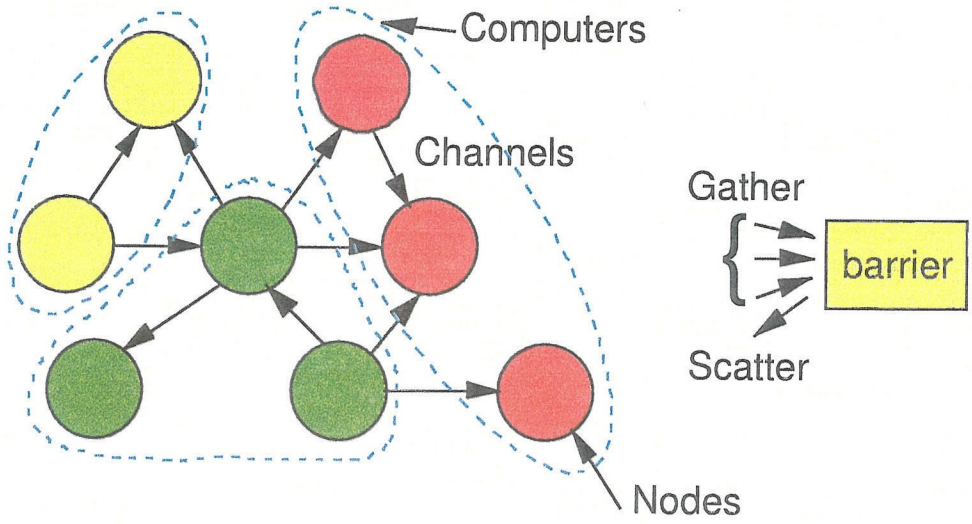

Figure 5.5: The Concurrent Graph abstraction

checkpointing routines for this purpose; node movement is achieved by checkpointing a node through the network rather than to a file. Furthermore, the library provides functionality to dynamically divide or merge nodes, given that the user has provided the necessary support routines. Node movement and dynamic granularity control are used in conjunction to provide portable load balancing.

5.9 Related Work

There are three basic techniques for concurrent DSMC, categorized by the way in which the computational grid is stored: *shared-grid*, *replicated-grid*, and *distributed-grid*.

Using the shared-grid approach, each processor manages a fraction of the particles during the transport phase, and a fraction of the cells during the collision phase. While this approach can yield reasonable speedup on shared-memory systems, it cannot be applied to distributed-memory systems.

The replicated-grid approach [48] stores a copy of the computational grid on each processor, and distributes particles among the processors. While the particle transport phase can then be performed locally, communication is required to bring together all of the particles in a cell, so that the collision phase can be executed. Computations

in the collision phase can also be completed locally. The disadvantage of this technique is that the memory required for storing the grid on each processor is constant. The use of complex geometries may result in large grid structures that cannot be stored on each processor. This approach is therefore inappropriate for large three-dimensional grid structures or large numbers of processors.

The distributed-grid approach [26, 80], distributes both the particle data structures and the grid data structures, using a spatial decomposition. While the transport phase requires communication as particles cross the boundaries between partitions of the grid, the collision phase is a purely local operation. Because the per-computer memory requirements for this technique are inversely proportional to the number of computers, it can be used for simulations involving large grids on large numbers of processors.

A number of other researchers have presented concurrent DSMC implementations with a variety of features. Wilmoth, Carlson, and Bird [105] present a portable implementation and demonstrate good scalability on the Intel iPSC/860, a Cray-YMP, and a group of Sun workstations. They present both a synchronous communication system, which is similar to that described in this thesis, and an asynchronous communication system that uses disk storage to buffer particles being communicated between processors (or between interleaved processes on a single processor machine). The disadvantage of the asynchronous system is that it does not preserve a consistent time and therefore cannot be used for unsteady simulations. The use of disk-based communication also lacks efficiency and impedes scalability. The most novel part of this work, however, is the presentation of results on a heterogeneous collection of workstations. Simulations were conducted on a network of Sun Sparc workstations with varying processing speeds. A fairly low utilization was reported, and the need for more sophisticated load balancing techniques was emphasized.

The MONACO system, developed at Cornell University, implements concurrent DSMC for two-dimensional triangular and three-dimensional tetrahedral grids [26]. This approach uses ray tracing for particle transport. The concurrent implementation based on domain decomposition is similar to that in the present work except for the

use of a global transpose for communicating the number of particles to be exchanged between processors. The high cost of this operation is the main constraint on the parallel performance of this implementation. One-dimensional domain decomposition is described for two-dimensional simulations. A global redistribution of cells is used for load balancing. Through the use of asynchronous communications, computation and communication can be overlapped.

The CHAOS library, developed at the University of Maryland [66] has been developed for communication and data structures required for particle-based applications. The Parallel Automated Runtime Toolkit at ICASE (PARTI) library developed at the Institute for Computer Applications in Science and Engineering (ICASE) [73] is designed to facilitate the implementation of parallel static irregular algorithms. Here, indirection arrays are used when data access patterns are static, but are only known at runtime. A concurrent DSMC implementation based on these two libraries is presented in [73], which also presents recursive coordinate bisection (RCB), recursive inertial bisection (RIB), and one-dimensional chain techniques for dynamic repartitioning. A heuristic for deciding when to rebalance, based on a *work degradation function* is presented.

Load balancing based on recursive spectral bisection (RSB) is described in [94, 97, 103]. Load balancing for DSMC computations has been discussed in [50] and [99]. Methods have been developed for multi-phase [100] and heterogeneous [101] load balancing, as well as for automatic granularity control [84]. For implementations that use only one partition per processor, the transfer of work between processors can also be achieved by exchanging individual cells between partitions [50].

5.10 Summary

This chapter has presented the concurrent Direct Simulation Monte Carlo technique. Extensions from the sequential algorithm have been described. Techniques for improving performance on a variety of computer architectures have been introduced, including partitioning, static and dynamic load balancing, and automatic granularity

control. The approach demonstrates the synergy between physical, chemical, numerical, and computational techniques that is required to efficiently address large-scale problems of industrial relevance.

Chapter 6 Concurrent Performance Model and Analysis

This chapter considers the parallel scalability of the DSMC method, using analysis techniques that are independent of the simulation architecture, gridding and implementation techniques, and transport, collision, and boundary models. A model is presented for predicting computational and memory requirements for concurrent simulations on homogeneous and heterogeneous architectures. Using this model, it is possible to predict what parallel speedup can be obtained on a given machine for a given physical system, and to determine how changes in physical parameters affect scalability. The effects of architectural and computational parameters, such as processing and communication speeds, are also described. Finally, it is possible to estimate the number of processors that are required to complete a specified simulation within a reasonable amount of time.

In order to answer these questions, this chapter presents a model of the parallel performance of a DSMC simulation. The performance implications of different connection topologies are considered. The model is validated and constants of the model determined for simple test cases. The model is then used to predict the scalability of a realistic three-dimensional simulation on both shared- and distributed-memory architectures, and the results shown to agree with experiments. Additional predictions define the boundaries of simulations that are feasible with existing computational resources.

6.1 Concurrent Performance Modeling

The study of the parallel performance of the DSMC method is an extension of the computational complexity study presented in Chapter 4. Consider a D-dimensional

simulation of an internal flow, with domain size L, gas number density n, collision cross section σ, stream speed $\overline{v}$, and average thermal speed v_t. Recall from Section 4.1.4 that the amount of time required for a single DSMC timestep on a sequential machine, T_{one}, is given by,

$$T_{one} = \frac{c_v^3}{c_\lambda^3}\left[c_p T_t + \frac{c_t(c_p-1)c_\lambda}{\sqrt{2}}\left(\frac{v_t}{\overline{v}+v_t}\right)T_c\right] D\left(n\sigma L\right)^D, \tag{6.1}$$

where c_λ is the ratio of the typical cell size to the particle mean free path, λ; c_v is the ratio of the average cell volume to the average cell length; c_t is the fraction of the typical cell length traversed by a particle during a single timestep; and c_p is the number of particles per cell. The implementation- and architecture-specific constants, T_t and T_c, represent the time per dimension required to move a particle for one timestep and to perform one collision, respectively. Using this equation, it is possible to predict the sequential runtime of a single timestep in a DSMC simulation. For the purposes of predictive performance modeling, the most important factors are the physical parameters, n, σ, and L, the number of dimensions, D, and the number of particles per cell, c_p.

For communication modeling in the following sections, it is also important to determine the number of computational grid cells that will be required for a simulation. Recall that the required number of cells is guided that the constraint that the typical cell size should be no more than a fraction, c_λ, of the mean free path. The number of cells required for a simulation, C, was obtained in (4.2),

$$C = \left(\frac{c_v^D}{c_\lambda^D}\right)\left(n\sigma L\right)^D. \tag{6.2}$$

The results of Equations 6.1 and 6.2 are used to model concurrent execution time and communication costs. The case of internal flows is considered because it is more general. External flows obey the same dependencies, with the appropriate substitutions for n, σ, and L. With these substitutions, however, performance requirements are essentially constant, and the analysis of this case is less interesting.

The time required for a concurrent DSMC timestep on the i-th processor, T^i_{conc}, can be written as the sum of compute time, T^i_{comp}, communication time, T^i_{comm}, and idle time, T^i_{idle},

$$T^i_{conc} = T^i_{comp} + T^i_{comm} + T^i_{idle}. \tag{6.3}$$

For a homogeneous collection of processors, the average amount of computation required per processor is the sequential DSMC time, T_{one}, divided by the number of processors, p,

$$T^{ave}_{comp} = \frac{T_{one}}{p}. \tag{6.4}$$

Consider a partitioning of the grid that gives each computer a different fraction of the total work, so that

$$T^i_{comp} = \frac{T^{ave}_{comp}}{f^i}, \tag{6.5}$$

where f^i is the ratio of the the average computation time to the i-th computer's computation time. The minimum value of f^i, U, is then the ratio of average to maximum computation time, and therefore a measure of the computational utilization, or load balance. The time taken by the computer with the most computation time, or the computer for which $f^i = U$, is given by,

$$T^{max}_{comp} = \frac{T^{ave}_{comp}}{U} = \frac{T_{one}}{Up}. \tag{6.6}$$

The fundamental scaling properties of the DSMC method are distinct from load balancing concerns. This section therefore assumes a uniform distribution of particles and load, for which the load balance can be predicted by the initial partitioning of the computational grid.

The concurrent algorithm requires that all partitions have completed a timestep before starting the next timestep. There is therefore a synchronization point at the end of the timestep. The total time required for a timestep, T_{conc}, is thus equal to the maximum sum of computation and communication times,

$$T_{conc} = T^{max}_{comp} + T^{max}_{comm}. \tag{6.7}$$

For uniform grids and simulations at low Mach numbers, it can be assumed that communication costs are proportional to computation costs. The load balancing factor, U, therefore affects both computation and communication terms in the same manner. The computer with the maximum computation time will also have the maximum communication time and therefore no idle time, Equations 6.3 - 6.7 can be combined to yield,

$$T_{conc} = \frac{1}{U}\left(T_{comp}^{ave} + T_{comm}^{ave}\right). \tag{6.8}$$

The time required for communication, T_{comm}, is strongly dependent on the manner in which the grid is partitioned, as discussed in the following section.

The parallel efficiency, E, is defined as the ratio of useful work to total work, or the ratio of the sequential time to the product of the parallel time and number of processors,

$$E = \frac{T_{comp}^{ave}}{T_{conc}} = \frac{T_{one}}{T_{conc}p}. \tag{6.9}$$

Combining (6.8) and (6.9),

$$E = \frac{U}{1 + T_{comm}/T_{comp}}. \tag{6.10}$$

In the following sections, the terms in these equations are estimated for different connection architectures, resulting in predictive models. The effects of initial partitioning on computational efficiency are also discussed.

6.2 Partitioning Issues

Computational grids must be initially partitioned for execution on multiple processors, and the choice of partitioning technique can impact the concurrent performance. Partitioning affects concurrent performance in two ways. First, the uniformity of partition sizes correlates with the uniformity of computer loads, and therefore load balance. Poor partitionings can yield poor load balances and thus inefficient concurrent execution. Second, the communication requirements are affected by the parti-

tioning. Since the area of surfaces exposed by partitioning translates directly into inter-processor communication, it is desirable to minimize the area of exposed surfaces.

Many different techniques can be used for grid partitioning, including bounding box division[79], chain partitioning [67], cell sorting[80], recursive coordinate bisection[108, 67], and recursive spectral bisection[10]. The present work considers only bouding-box division, because it is the simplest approach and can yield reasonable results for simple geometries and uniform flows. The analysis, however, can be extended to address more sophisticated partitioning strategies. The bounding box method works as follows: A first pass through the entire domain is used to determine the extent of the grid in each of three axes. The resulting bounding box is divided into equal-volume partitions. For box-like geometries with uniform grids, this method yields uniform partition sizes and minimal exposed surface area, which results in good load balance and minimum communication costs. For complex geometries and non-uniform grids, however, the bounding box method may yield poorly balanced partitions.

Three parameters are required for the bounding box method in three dimensions: the number of partitions in the x-dimension, P_x, the number of partitions in the y-dimension, P_y, and the number of partitions in the z-dimension, P_z. The total number of partitions that are generated, p, is the product of these, $p = P_x P_y P_z$. The number of cuts, or exposed surfaces, in each dimension, is one less than the number of partitions. The total number of cuts through the entire domain, N_{cut}, for a three-dimensional partitioning, is given by,

$$N_{cut} = (P_x - 1) + (P_y - 1) + (P_z - 1) = P_x + P_y + P_z - 3. \tag{6.11}$$

Assuming that for 3-dimensional partitioning $P_x = P_y = P_z = p^{1/3}$, and that for 2-dimensional partitioning $P_x = P_y = p^{1/2}$, the number of cuts in a q-dimensional

partitioning can be generalized to be,

$$N_{cut} = 2q\left(p^{1/q} - 1\right). \tag{6.12}$$

For a D-dimensional simulation, the number of cells along each axis of a grid is proportional to the D-th root of the total number of cells. The $D-1$ power of this value, or the $\frac{D-1}{D}$ power of the total number of cells, gives the number of cells exposed by each cut, E,

$$E \approx C^{\frac{D-1}{D}}. \tag{6.13}$$

Note that the dimensionality of the partitioning, q, can not exceed the dimensionality of the simulation, D:

$$q \leq D. \tag{6.14}$$

The total number of exposed cells for the entire partitioning, C_e, can be obtained by combining (6.11) and (6.13),

$$C_e = N_{cut}E = (P_x + P_y + P_z - 3)C^{\frac{D-1}{D}} = 2q\left(p^{1/q} - 1\right)C^{\frac{D-1}{D}}. \tag{6.15}$$

Assuming a uniform grid density and appropriate timestep and cell sizes, the number of particles that must be sent by a partition is proportional to the surface area of faces exposed by partitioning, and to the total particle velocity, v_{tot}. The number of exposed faces is proportional to the average number of exposed cells, C_e/p. The total velocity is the sum of average thermal speed, v_t, and the stream speed, $\overline{v}$: $v_{tot} = v_t + \overline{v}$. The average number of particles sent by a partition, n_p, is then,

$$n_p = \frac{c_f c_p C_e}{p}\left(v_t + \overline{v}\right), \tag{6.16}$$

where c_f is the fraction of particles in exposed cells that must be communicated. Using (6.15), this can be written,

$$n_p = \frac{c_f c_p 2q\left(p^{1/q} - 1\right)C^{\frac{D-1}{D}}}{p}\left(v_t + \overline{v}\right), \tag{6.17}$$

where q is the dimensionality of the partitioning, D is the dimensionality of the simulation, and p is the number of partitions. Knowing the number of particles that must be exchanged, it is possible to predict the amount of time that will be spent during the particle exchange phase of the concurrent DSMC algorithm.

6.3 Communication Modeling

Communication cost in a DSMC simulation can be predicted as a function of physical and computational parameters. Recall from (6.8) that the dependence of the concurrent execution time on the communication time,

$$T_{conc} = \frac{T_{one}}{Up} + \frac{T_{comm}^{ave}}{U}. \tag{6.18}$$

Communication takes place at two points in the concurrent algorithm in Program 5.1. The first is communication of particles between adjacent partitions, and the second is the global gathering of progress information, such as the number of particles in the domain. Assuming that particle exchange communication time is T_{ex}, and that the global communication time is T_g, the total communication time per processor, T_m, can be written,

$$T_m = T_{ex} + T_g. \tag{6.19}$$

Global communication, however, is typically only used for gathering a small amount of progress information, and can be performed infrequently. Its contribution to the total communication time can thus be neglected. In most simulations, particle exchange dominates communication, in terms of the number and size of messages and the time required for exchanging messages.

Particle exchange requires some fixed overhead time, as well as time for the actual transfer of messages. The overhead time, T_l, is required for communication latency and for transferring synchronization messages that contain no particles. The time required for the actual transfer of particles is proportional to the number of particles, and given by the product of particle transport time per dimension, T_p, the number

of dimensions, D, and the number of particles being transferred, n_p. The time T_p includes any additional computation that is required for incorporating particles into the data structures of their destination partitions. This gives a total communication time,

$$T_m = T_l + T_p n_p D. \tag{6.20}$$

Combining (6.20) and (6.17) yields,

$$T_m = T_l + T_p n_p D = T_l + T_p \frac{c_f c_p 2q \left(p^{1/q} - 1\right) C^{\frac{D-1}{D}}}{p} \left(v_t + \overline{v}\right). \tag{6.21}$$

Using the value for the number of cells, C, computed in (4.2), this can be written,

$$T_m = T_l + T_p \left(\frac{2c_f c_p c_v^{D-1}}{c_\lambda^{D-1}}\right) \frac{q\left(p^{1/q} - 1\right)}{p} \left(v_t + \overline{v}\right) n^{D-1} \sigma^{D-1} V^{\frac{D-1}{D}}. \tag{6.22}$$

This equation gives the time that is required for a single processor to perform its portion of the communication. The total communication time for all processors during one timestep depends on the type of interconnection architecture and the degree to which communication bandwidth must be shared. The following sections consider the communication cost implications of both mesh-based and bus-based architectures.

6.4 Model Parameters

Table 6.1 summarizes the model parameters used in the sequential and concurrent models. For each parameter, representative values are listed. The first three values in this table, c_λ, c_p, and c_v, were obtained in Section 4.3.1. The value for c_f was obtained by measuring exposed cells and communicated particles for a variety of partitionings of box grids.

For illustrative purposes, typical implementation-specific values were obtained for an actual DSMC implementation [79]. Other DSMC implementations will have different associated constants, but must obey the same dependences on the physical and architectural parameters. For each architecture, a series of simulations were con-

Table 6.1: General Model Parameters

Parameter	Description	Typical Values
c_λ	Ratio of cell length to local mean free path, or minimum local Knudsen number	0.3-1
c_p	Ratio of particles to cells	3-10
c_v	Ratio of cell length to cube root of cell volume	1-5
c_f	Fraction of particles in exposed cells that are communicated	0.51

ducted of gas flow in a simple box grid, in order to estimate architecture-specific parameters. Simulations were conducting using Argon gas and the Variable Soft Sphere (VSS) collision model [15].

In order to determine constants for the model in the previous sections, and to evaluate its accuracy, scalability tests were conducted on two architectures, the SGI Power Challenge and the Cray T3D. The SGI Power Challenge has 14 75-MHz R8000 processors and 2 GBytes of shared RAM, while the Cray T3D has 256 200-MHz Dec Alpha processors with 64 GBytes of RAM per processor. These machines represent two classes of architectures, and the same modeling techniques could be extended to address other concurrent architectures.

The value of T_t was obtained by measuring simulation timestep time with the collision phase disabled, and dividing this by the number of particles and the number of dimensions (3). Similarly, T_c was calculated by timing simulation timesteps with the transport model disabled, and dividing by the number of collisions and the number of dimensions. In order to estimate communication costs, the time required for the communication phase was measured for several different partitionings and numbers of processors. Linear regression was then used to estimate the overhead time, T_l, and the particle transfer time, T_p. Because multiple rounds of communication may be required for the particle exchange phase, and computation is required for sending and receiving particles, these numbers are significantly larger than would be expected

from the performance characteristics of the underlying communication architecture.

Table 6.2: Implementation-Specific Parameters

Parameter	Description	Pwr. Chal.	T3D
T_t	Single-particle transport time per dimension	$11\mu s$	$16\mu s$
T_c	Collision time per dimension	$13\mu s$	$41\mu s$
T_l	Communication overhead time	.0138 s	0.155 s
T_p	Particle transfer time per dimension	$26\mu s$	$60\mu s$

Table 6.2 shows the values of T_t, T_c, T_l, and T_p for the SGI and Cray architectures, obtained from these experiments. In the following sections, communication modeling is extended to predict timestep duration and parallel efficiency on bus-based and mesh-based architectures. The models use these values to predict performance on concurrent architectures, and predictions are compared with experiment.

6.5 Mesh-Based Modeling

Mesh-based architectures, such as the Cray T3D and the Intel Paragon, have dedicated connections between adjacent processors. For a q-dimensional mesh, each processor has $2q$ such connections, and must send q messages, each with n_p/q particles. The communication time is then, using (6.22),

$$T_{comm} = T_m = qT_l + T_p \left(\frac{2c_f c_p c_v^{D-1}}{c_\lambda^{D-1}} \right) \left(\frac{p^{1/q} - 1}{p} \right) (v_t + \overline{v})\, D\, (n\sigma L)^{D-1}. \qquad (6.23)$$

This value can be used for the communication time in (6.10) in order to calculate the parallel efficiency, E,

$$\begin{aligned} E &= \frac{1}{1+T_m/T_{comp}} \\ &= U \left[1 + \frac{qT_l + T_p \left(\frac{2c_f c_p c_v^{D-1}}{c_\lambda^{D-1}} \right) \left(\frac{p^{1/q}-1}{p} \right) (v_t+\overline{v}) D (n\sigma L)^{D-1}}{\frac{c_v^D}{c_\lambda^D} \left[c_p T_t + \frac{c_t (c_p - 1) c_\lambda}{\sqrt{2}} \left(\frac{v_t}{\overline{v}+v_t} \right) T_c \right] D \frac{(n\sigma L)^D}{p}} \right]^{-1}. \end{aligned} \qquad (6.24)$$

While the full form of this equation must be used for accurate modeling of realistic

simulations, it is too complex to permit intuitive analysis. With the application of three assumptions, it can be reduced to a more manageable form. First, assume zero-overhead communication, in which case the term qT_l can be neglected. Second, assume a large number of particles per cell, such that $c_p >> 1$. Third, assume low Mach-number flows, for which $\overline{v} << v_t$. With these assumptions, the equation for sequential processing time, (6.1), can be reduced to,

$$T_{one} = T_s c_p D \left(n\sigma L\right)^D , \tag{6.25}$$

where T_s is an implementation- and architecture-dependent constant, and subsumes the constants T_t, T_c, c_v, c_λ, and c_t. With these approximations, (6.24) can be simplified to,

$$E \approx U \left[1 + \frac{T_p}{T_s}\left(\frac{2c_f c_v^{D-1}}{c_\lambda^{D-1}}\right)\left(p^{1/q} - 1\right)\left(\frac{v_t + \overline{v}}{n\sigma L}\right)\right]^{-1} . \tag{6.26}$$

This equation explains how model and physical parameters affect the computational efficiency of a simulation. The dependence on the dimensionality of the mesh, q, shows the advantage of higher-dimension meshes. As q increases, the communication term decreases, thus improving efficiency. The dependence on the number of processors, p, suggests an inherent limit in the scalability of the algorithm. Because computation time decreases faster than the communication time (for a fixed-size problem), efficiency decreases with the number of processors. Once the number of processors is large enough, the number of cells per processor becomes small, the fraction of the particles that must be exchanged approaches unity. In this case, almost all of the simulation time is spent on communication and overhead. On the other hand, as the problem size (measured in terms of the quantity $n\sigma L$) grows, the number of required cells, and thus the computation, grows faster than the number of exposed faces, and thus the communication. Efficiency therefore increases with problem size.

Note that there is no strong dependence of efficiency on the dimensionality of the simulation. For a given partitioning and problem size, both communication and computation scale with problem dimensionality so that efficiency is unaffected. Because the dimensionality of the partitioning can be no greater than the dimensionality of

the grid, however, the maximum achievable efficiency is higher for three-dimensional simulations than for one- and two-dimensional simulations.

It is also important to note that the efficiency is directly proportional to the load balance, U. The initial load balance is determined by the initial partitioning and the initial distribution of particles. Assuming a uniform distribution of particles, however, the load balance can be reasonably predicted by the balance of the volumes of the initial partitions. This value is determined by the geometry and the specific partitioning technique, but may be determined statically, and thus used for predictive modeling. During a real simulation, especially one involving gas injection and exhaust, the load balance may not correspond to the volume balance of the initial partitioning, and may change during the course of a simulation.

Equations (6.24) and (6.26) also show the effects of potential optimizations. Improvements to the sequential portion of the application will reduce T_t and T_c and therefore the total simulation time, but will also decrease the efficiency. Optimizations to the communication hardware and software will reduce communication costs, T_p, and therefore improve efficiency. While the extension to more sophisticated chemistry models can increase the amount of sequential work and thus the parallel efficiency, any corresponding increases in the size of particle data structures will result in increased communication and thus lower parallel efficiency.

6.6 Mesh-Based Experiments

In order to test the predictive capability of the mesh-based performance model, several simulations were conducted for a box geometry, with 12,540 tetrahedral cells of roughly uniform size, using 10 particles per cell, a particle density of $7.3 \times 10^{18} m^{-3}$, and temperature of 300 K. The same problem was simulated on 1, 2, 4, 8, 32, 64, and 128 processors. Before each simulation, the balance of partition volumes was computed in order to estimate the load balance of the computation. The time per timestep was measured and compared with both predicted and ideal speedups.

Figure 6.1 (left) shows the time per timestep as a function of the number of pro-

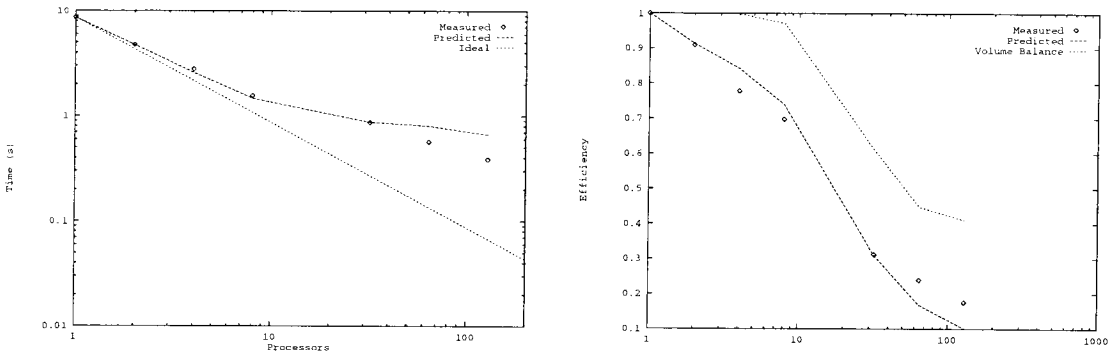

Figure 6.1: Predicted and measured scaling of a box simulation the Cray T3D

cessors for measured, predicted, and ideal cases. Up to 32 processors, the agreement between the model and experiment is excellent. For 64 and 128 processors, the measured step time was slightly less than predicted, but the trend is correct. Figure 6.1 (right) shows the parallel efficiency, E, for the same experiments. The trend is again correctly captured, and the predictions are accurate up to 32 processors. On 128 processors, each cell has fewer than 100 cells and thus fewer than 1000 particles. The amount of computation on each processor is therefore very small. Most data structures might even be able to fit in cache memory, thus resulting in an additional speedup over predictions. This figure also shows the volume balance of the initial grid. For large numbers of processors, the efficiency loss due to load imbalance is significantly larger than the loss due to communication.

Simulations of the GEC reactor were also conducted on the T3D. The GEC Reactor grid, shown in Figure 1.2 (right), has a volume of $0.013m^3$. It was simulated at an operating pressure of 0.291 Pa (2.2 mTorr) and operating temperature $300K$, which corresponds to a particle density of $7.0 \times 10^{19}m^{-3}$. The grid was partitioned for 16, 32, 64, and 128 processors. Due to memory constraints, it was not possible to simulate this grid on fewer than 16 processors. For each simulation, the step time and efficiency were both predicted and measured as functions of the number of processors.

The scalability results for the GEC grid are shown in Figure 6.2, with step time on the left and efficiency on the right. The model consistently underpredicts the timestep

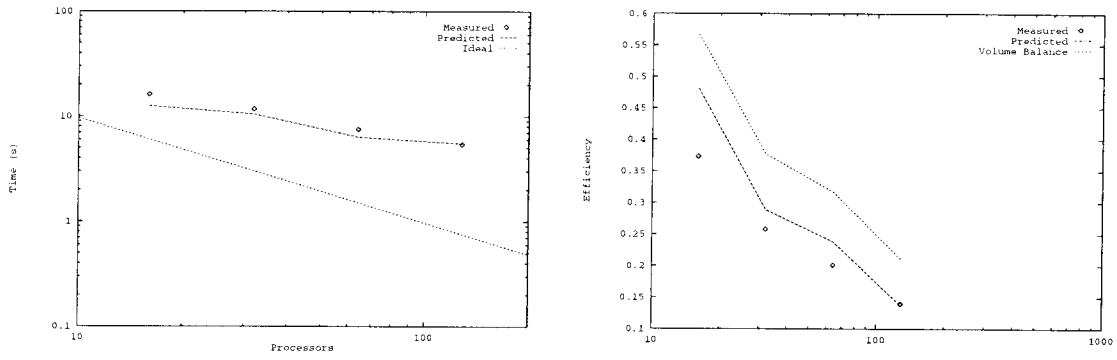

Figure 6.2: Predicted and measured scaling of a GEC simulation on the Cray T3D

time, but only by a small margin. One reason for this may be a poor estimate of uniprocessor time. Since it was impossible to simulate this grid on one processor, the per-cell processing rate for the box problem was used to estimate the uniprocessor time. As has been shown in Section 4.5, however, this procedure may underestimate the time required to simulate complex geometries by as much as 25%. Increasing the step time by 25% for each case would give a much better agreement between the model and experiments.

The primary result from this set of experiments is that the model accurately predicts the performance trend as a function of the number of processors. This is also shown in the efficiency plot, Figure 6.2 (right). Here again it can be seen that the loss in efficiency due to load imbalance is greater than the loss due to communication. The communication costs, however, are accurately predicted by the model. The model is therefore an effective tool for predicting parallel performance of large, complex-simulations.

6.7 Bus-Based Modeling

The analysis for the efficiency of bus-based systems is similar to that of mesh-based communications, except that the bandwidth of the network must be shared by all processors. Bus-based systems include shared-memory systems, where the processors all

share the same path to main memory, and networks of workstations, where all of the machines are on the same network segment and must share the network bandwidth. The communication component of the algorithm in these cases is determined by the total number of messages exchanged, not the number of messages per processor. The messaging time is then,

$$T_m = pqT_l + T_p \left(\frac{2c_f c_p c_v^{D-1}}{c_\lambda^{D-1}} \right) \left(p^{1/q} - 1 \right) (v_t + \overline{v}) \, D \, (n\sigma L)^{D-1}, \tag{6.27}$$

where q is the dimensionality of the partitioning. The efficiency can then be computed as,

$$E = U \left[1 + \frac{qpT_l + T_p \left(\frac{2c_f c_p c_v^{D-1}}{c_\lambda^{D-1}} \right) \left(p^{1/q} - 1 \right) (v_t + \overline{v}) \, D \, (n\sigma L)^{D-1}}{\frac{c_v^3}{c_\lambda^3} \left[c_p T_t + \frac{c_t (c_p - 1) c_\lambda}{\sqrt{2}} \left(\frac{v_t}{\overline{v} + v_t} \right) T_c \right] \frac{D(n\sigma L)^D}{p}} \right]^{-1}. \tag{6.28}$$

As with mesh-based modeling, it is interesting to consider the simplification resulting from the assumptions of zero-overhead, low Mach number, and many particles per cell,

$$E \approx U \left[1 + \frac{T_p}{T_s} \left(\frac{2c_f c_v^{D-1}}{c_\lambda^{D-1}} \right) p \left(p^{1/q} - 1 \right) \left(\frac{v_t + \overline{v}}{n\sigma L} \right) \right]^{-1}. \tag{6.29}$$

Computational efficiency therefore improves with the problem size and the dimensionality of the partitioning. Unlike with mesh-based computations, however, efficiency decreases rapidly with the number of processors, by an additional factor of p. This result demonstrates the clear advantage of mesh-based networks over bus-based networks for DSMC computations.

6.8 Bus-Based Experiments

DSMC performance on shared-memory systems was evaluated on a 14-processor Silicon Graphics Power Challenge. At the lowest level, this machine uses a bus architecture, where all processors must share a fixed amount of bandwidth to main memory. Unless the bus is saturated, however, communication between processors may effec-

tively take place in parallel. Because a significant portion of the communication may be performed in parallel, the machine may behave like a mesh architecture under certain conditions. For these reasons, the performance of this machine was predicted using both bus-based and mesh-based communication models.

Simulations were conducted on a 54145-cell box geometry. Simulations were conducted using 1, 2, 4, 8, 12, and 14 processors on this system. For each case, the timestep time and efficiency were predicted and measured.

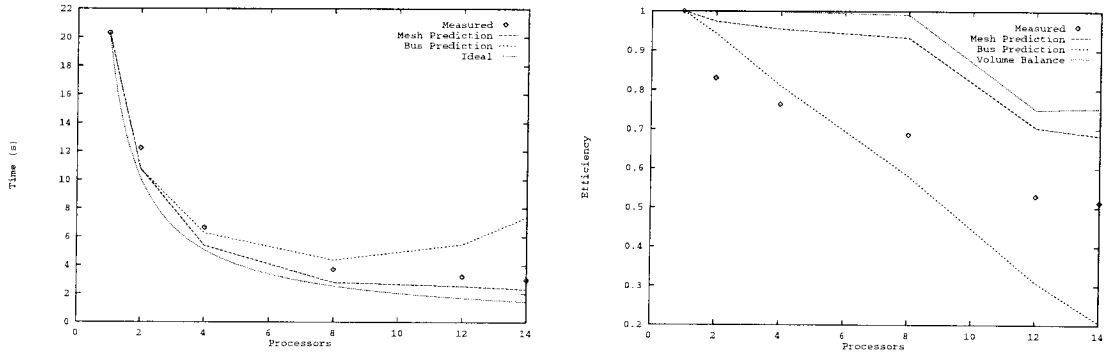

Figure 6.3: Predicted and measured scaling of a box simulation the SGI Power Challenge

The results of the box simulations on the SGI Power Challenge are shown in Figure 6.3. Figure 6.3 (left) shows time per timestep as a function of the number of processors, for ideal speedup, mesh- and bus-based predictions, and experimental results. Up to 4 processors, both models slightly underpredict the time per timestep. For 8-14 processors, the bus-based prediction is significantly higher than the mesh-based prediction. This is because the model predicts an increase in contention for the shared bus. Experimental results, however, fall between the two models. This suggests that the actual behavior of the system is most accurately modeled by combination of bus- and mesh-based approaches. Since the performance trend is more closely matched by the mesh-based model, however, this suggests that the majority of communication operations can occur concurrently.

The computational efficiency of the box simulation on the SGI Power Challenge is

shown in Figure 6.3 (right). This shows that the models overpredict the efficiency for 1-4 processors, but bound it for 8-14 processors. This figure also shows the balance of the initial partitioning, which accounts for a substantial fraction of the efficiency loss on 12-14 processors.

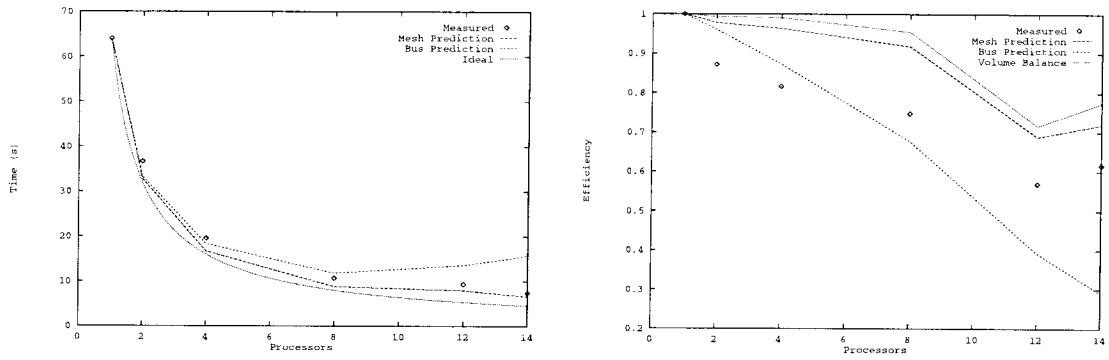

Figure 6.4: Predicted and measured scaling of a GEC simulation on the SGI Power Challenge

The GEC grid was also simulated on the SGI Power Challenge. The time per timestep as a function of number of processors is shown in Figure 6.4 (left). As with the box grid, both models underpredict the time per timestep up to 4 processors, but for 8-14 processors, step time is between the two predictions. The mesh-based model is generally more accurate than the bus-based model. Figure 6.4 (right) shows the efficiency and volume balance for the same set of experiments.

6.9 Model Predictions

The preceding sections presented a model for concurrent DSMC performance, validated it on small test problems, and tested its predictive abilities on a large-scale problem. The same model can also be used to predict the efficiency of more complex simulations on larger machines. Consider three-dimensional simulations of the GEC Reference Cell at operating pressures of 2.66 Pa/20 mTorr, 6.65 Pa/50 mTorr, and 13.3 Pa/100 mTorr. Because the time required per timestep is proportional to the

number of simulated cells and the number of simulated cells required is proportional to the cube of the pressure (for three-dimensional simulations), sequential runtime of these simulations would be substantially greater than that of the 0.29 Pa/2.2 mTorr simulation.

Table 6.3 lists the number of grid cells required for simulations at each of these pressures. The higher-pressure simulations require more cells than can be stored on available sequential systems, so uniprocessor time must be estimated by scaling the per-particle processing rate from the lowest-pressure case. Because the grids would be too large to partition statically, it is also impossible to measure the volume balance of initial partitioning. Load balance was therefore assumed to be 90% for 2 or more processors. In practice, achievable load balance would depend on both the initial partitioning and any dynamic load balancing techniques that are used.

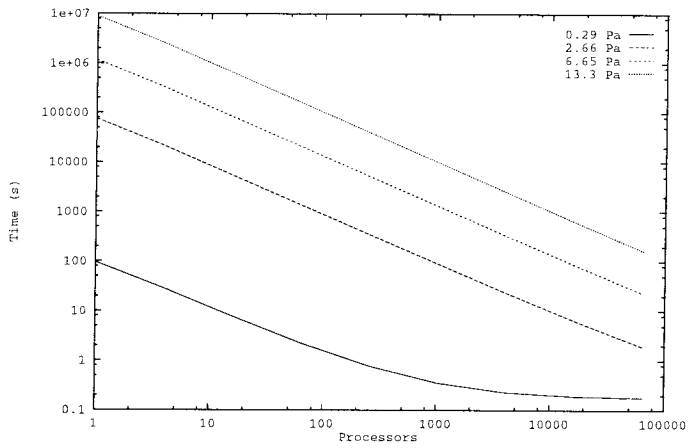

Figure 6.5: Predicted timestep time as a function of number of processors for 0.29 Pa/2.2 mTorr, 2.66 Pa/20 mTorr, 6.65 Pa/50 mTorr, and 13.3 Pa/100 mTorr simulations

Figure 6.5 shows the predicted timestep times for each of the GEC simulations, for 1-65,536 processors processors. Note that the step time increases substantially for the higher pressures. The scaling for high pressures is excellent. No matter how many processors are used, however, the timestep time can never be less than the communication overhead time, $T_l = 0.155$ seconds. The dominance of this term can be seen in the predictions for the 0.29 Pa/2.2 mTorr case for more than 1000

processors. For this simulation, the use of more than 1000 processors would not result in any further speedup.

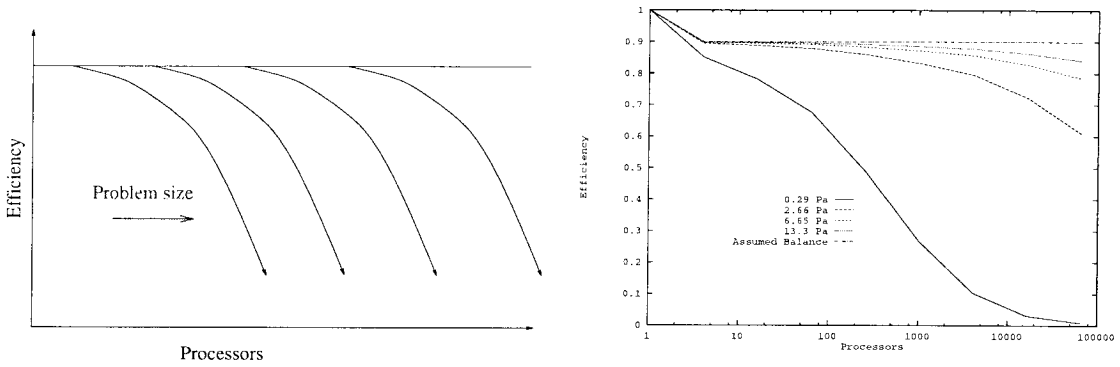

Figure 6.6: Parallel efficiency as a function of problem size and number of processors (left) and predicted parallel efficiency (right) for 0.29 Pa/2.2 mTorr, 2.66 Pa/20 mTorr, 6.65 Pa/50 mTorr, and 13.3 Pa/100 mTorr simulations

For most realistic problems, parallel efficiency decreases with larger numbers of processors. This is the result of fixed overhead, or inherently sequential aspects of an algorithm, that remain constant as the amount of useful work per processor decreases. On the other hand, as the problem size is increased, the amount of useful work per processor increases, further amortizing the fixed overhead, and improving parallel efficiency. This phenomenon is represented schematically in Figure 6.6 (left).

Predictions of efficiency, assuming a 90% load balance, are shown in Figure 6.6 (right). The efficiency of the low pressure simulations decreases more rapidly than that of the high pressure simulations. For the 0.29 Pa/2.2 mTorr case, efficiency drops below 30% for more than 1000 processors. For the 13.3 Pa/100 mTorr case, however, efficiency is predicted to remain above 80%, even on 65,536 processors. Note that

It is interesting to consider the number of Cray T3D processors that would be required to complete a GEC simulation at a given pressure in a given amount of time. While the sequential time required for a timestep is proportional to the cube of the pressure, the number of steps required to reach convergence is also proportional to the pressure. The total sequential time required to reach convergence is therefore

proportional to the fourth power of the pressure.

For reactor simulations to be useful for industrial purposes, they must be completed in reasonable timeframes. Assume that the maximum time available for a simulation is one week, or 604,800 seconds. The number of steps required for a simulation can be calculated as in Section 4.5. Dividing the time available by the number of steps required gives the maximum allowable time per timestep, T_{max}. Using the predicted step times from Figure 6.5, this can be used to determine the minimum number of processors required to give a solution in the desired time, p_{time}. Values for T_{max} and p_{time} are shown in Figure 6.3. As the pressure increases, the number of processors required quickly becomes very large. Above 6.65 Pa/50 mTorr, the minimum number of processors is much larger than the size of currently available systems.

The number of processors required for a simulation, however, is also constrained by the amount of available memory. If a single Cray T3D processor had enough available memory, it could complete a simulation at 0.29 Pa/2.2 mTorr in less than a week. A 64-MByte T3D processor, however, only has approximately 54 MByte of available memory, and even with completely uniform memory usage on each processor, at least 5 processors would be required for this simulation. Because of memory imbalances resulting from uneven grid density and uneven initial partitioning, actual simulations at this pressure cannot be conducted with fewer than 16 processors. Memory requirements for a DSMC simulation are proportional to the number of cells required, which is proportional to the cube of the simulated pressure. By scaling the memory requirements accordingly, it is possible to predict the minimum number of processors, p_{mem}, required for completing a simulation at a given pressure.

Table 6.3: GEC Simulation Predictions

Press.	Press.	Cells	T_{max}	p_{time}	p_{mem}
Pa	mTorr		sec.		
0.291	2.2	1.4×10^5	7,760	1	5
2.66	20	1.1×10^8	850	128	3,800
6.65	50	1.7×10^9	339	650	60,500
13.3	100	1.3×10^{10}	170	50,000	479,000

Estimated values for p_{mem} are shown in Table 6.3. For each case, $p_{mem} > p_{time}$, therefore these simulations are memory-bound. In other words, the addition of more memory could substantially reduce the number of required processors. While a 2.66 Pa simulation would require 3,800 64-MByte processors, it would be possible with 1750 128-MByte processors, or 841 256-MByte processors. The effect of pressure on computational requirements is also clearly demonstrated in this table. As pressure increases, the required number of processors grows very rapidly. As explained in Section 4.5, these predictions are conservative estimates of runtime and storage requirements. In practice, it may be possible to obtain satisfactory results using substantially fewer processors and less memory.

6.10 Heterogeneous Modeling

The preceding analysis has been developed for *homogeneous* collections of processors. In order to make use of all available computational resources, including networks of workstations, it is important to also consider *heterogeneous* collections of computers. In general, computers may differ in speed of processors and amount of available memory. Consider a collection of p computers, each with a different processing speed and memory, M^i. The time required for single-particle transport, T_t^i, and for single-collisions, T_c^i, also varies between computers. The speed at which a computer can execute a DSMC timestep, measured in terms of particles per second, is given by,

$$S^i = p^i \frac{N^i}{T_{one}^i} = \frac{c_p C^i}{\left[c_p T_t^i + \frac{c_t(c_p-1)c_\lambda}{\sqrt{2}}\left(\frac{v_t}{\bar{v}+v_t}\right)T_c^i\right] DC^i} = \frac{1}{\left[T_t^i + \frac{c_t(c_p-1)c_\lambda}{c_p\sqrt{2}}\left(\frac{v_t}{\bar{v}+v_t}\right)T_c^i\right] D}. \tag{6.30}$$

The parameters T_t^i and T_t^i are likely to scale between machines in the same manner. In this case, they can be written in terms of reference times, T_t and T_t, and processor speeds, s^i,

$$T_t^i = \frac{T_t}{s^i}, T_c^i = \frac{T_t^i}{s^i}. \tag{6.31}$$

The DSMC speed of a computer is then given by,

$$S^i = \frac{s^i}{\left[T_t + \frac{c_t(c_p-1)c_\lambda}{c_p\sqrt{2}}\left(\frac{v_t}{\overline{v}+v_t}\right)T_c\right]D}. \tag{6.32}$$

The total DSMC speed of a collection of computers, S, is the sum of the individual speeds,

$$S = \sum_i S^i = \frac{\sum_i s^i}{\left[T_t + \frac{c_t(c_p-1)c_\lambda}{c_p\sqrt{2}}\left(\frac{v_t}{\overline{v}+v_t}\right)T_c\right]D}. \tag{6.33}$$

As with homogeneous simulations, the time required for a timestep, T_{conc}, is the sum of communication and computation times. The computation time for a given partition, T^i_{comp}, is proportional to the number of cells in that partition, C^i,

$$T^i_{comp} = \left[c_pT_t + \frac{c_t(c_p-1)c_\lambda}{\sqrt{2}}\left(\frac{v_t}{\overline{v}+v_t}\right)T_c\right]D\frac{C^i}{s^i}. \tag{6.34}$$

The communication time is a linear function of the number of particles communicated,

$$T^i_{comm} = qT^i_l + T^i_pDn^i_p. \tag{6.35}$$

Assuming that communication times scale with processing times, this can be written,

$$T^i_{comm} = \frac{qT_l + T_pDn^i_p}{s^i}. \tag{6.36}$$

Recall from Section 6.2 that the number of particles communicated by a partition is proportional to the number of exposed faces,

$$n_p = c_fc_pC^i_e\left(v_t + \overline{v}\right), \tag{6.37}$$

For homogeneous networks, it is possible to assume that the number of exposed faces of a partition is the average number of exposed faces, or the total number of exposed faces divided by the total number of partitions. For heterogeneous networks, however, partitions will differ both in terms of the number of total cells and the number of

exposed cells. It is therefore more appropriate to write the number of exposed cells in terms of the total number of cells and the dimensionality of the partitioning,

$$C_e^i = c_h \left(C^i\right)^{\frac{q-1}{q}}, \tag{6.38}$$

where c_h is a proportionality constant that reflects the shape of the partitions. The communication time for the i-th processor can then be written,

$$T_{comm}^i = \frac{1}{s^i}\left(qT_l + T_p D n_p^i\right) = \frac{1}{s^i}\left[qT_l + T_p D c_p c_f c_h \left(v_t + \overline{v}\right)\left(C^i\right)^{\frac{q-1}{q}}\right]. \tag{6.39}$$

The total time for a timestep is then given by,

$$\begin{aligned} T_{conc}^i &= T_{comp}^i + T_{comm} \\ &= \left[c_p T_t + \tfrac{c_t(c_p-1)c_\lambda}{\sqrt{2}}\left(\tfrac{v_t}{\overline{v}+v_t}\right)T_c\right] D\tfrac{C^i}{s^i} \\ &\quad + \tfrac{1}{s^i}\left[qT_l + T_p D c_p c_f c_h \left(v_t + \overline{v}\right)\left(C^i\right)^{\frac{q-1}{q}}\right]. \end{aligned} \tag{6.40}$$

The optimal assignment of cells to partitions will result in all partitions having the same concurrent execution time, T_{conc}. Consider a partitioning where each partition has a fraction, f^i, of the total number of cells. This equation can be solved for f^i, and the constraint that the sum of these fractions must equal one gives the total concurrent execution time. For arbitrary q, this cannot be solved analytically. As illustration, consider a simplified one-dimensional partitioning for which communication overhead, T_l, can be neglected. Let $T_s = T_t + \frac{c_t(c_p-1)c_\lambda}{\sqrt{2}}\left(\frac{v_t}{\overline{v}+v_t}\right)T_c$. In this case, the amount of communication per partition is constant, and the total step time is,

$$T_{conc}^i = \frac{Dc_p}{s^i}\left[T_s f^i C + T_p c_f c_h \left(v_t + \overline{v}\right)\right]. \tag{6.41}$$

Solving this equation for f^i, subject to the constraint $\sum f^i = 1$ yields the total concurrent time,

$$T_{conc} = \frac{Dc_p}{\overline{s}}\left[\frac{T_s C}{p} + T_p c_f c_h \left(v_t + \overline{v}\right)\right], \tag{6.42}$$

and the optimal cell allocation,

$$f^i = \frac{s^i}{\overline{s}} \left[\frac{1}{p} + \frac{T_p}{T_s C} c_f c_h \left(v_t + \overline{v} \right) \right] - \frac{T_p}{T_s C} c_f c_h \left(v_t + \overline{v} \right). \tag{6.43}$$

While intuition would suggest that the optimal allocation of cells is simply $f^i = \frac{s^i}{\overline{s}p}$, this is not correct. Because communication does not increase at the same rate as computation, work allocations must be adjusted accordingly. Note that these equations reduce as would be expected. If communication costs can be neglected ($T_p \approx 0$), the concurrent timestep time corresponds to ideal scaling, $T_{conc} = \frac{D c_p T_s C}{\overline{s}p}$, and the optimal cell allocation is proportional to the processing speed, $f^i = \frac{s^i}{\overline{s}p}$. This also correctly reduces to the homogeneous case, for which $f^i = \frac{1}{p}$.

It is possible for some processors to be so slow that they spend all of their time on communication and can not contribute useful work to the computation. This corresponds to the case

$$f^i = \frac{s^i}{\overline{s}} \left[\frac{1}{p} + \frac{T_p}{T_s C} c_f c_h \left(v_t + \overline{v} \right) \right] < \frac{T_p}{T_s C} c_f c_h \left(v_t + \overline{v} \right). \tag{6.44}$$

Any such processor should be removed from the computation. Another case in which work assignment needs to be different from that specified in (6.43) is when memory constraints are exceeded. If the amount of memory required for $f^i C$ cells is greater than the amount of available memory on processor i, "excess" work from this processor must be reassigned to other processors.

Having considered the case of one-dimensional partitioning and zero-overhead communication, it is possible to make qualitative statements about more general cases. It is always true that as the size of a partition grows, the amount of computation that it requires will grow faster than the amount of communication that it requires. For this reason, multi-dimensional partitioning will not avoid the situation wherein a processor may be too slow to make a useful contribution to the computation.

6.11 Related Work

Several DSMC implementations on concurrent architectures have been presented [26, 50, 51, 91, 67], and scalability analysis has been presented in [104]. The scalability of similar techniques has been studied in [31, 25]. Load balancing and related performance optimizations for particle-based simulations are discussed in [87, 108, 50, 99, 100, 66]. The use of automatic granularity control for DSMC simulations is presented in [84].

Parallel scalability analysis of the DSMC method is presented in [104]. The high cost of communications and simplistic global remapping strategy used in this implementation limit efficiency to about 20% on 64 processors. The advantage of increasing problem size with machine size is discussed, and performance predictions are made for larger problem sizes and larger machines. While this work elucidates the effects of communication latency and bandwidth, as well as sequential execution time and problem size, on a concurrent simulation, it does not explain the dependence of these quantities on physical simulation parameters.

Previous work has been conducted to show that DSMC simulations do not scale well for fixed-size problems [73]. This technique uses dynamic remapping techniques such as recursive coordinate bisection (RCB), recursive inertial bisection (RIB), and one-dimensional chain partitioning in order to maintain load balance. When scaling the number of particles and cells with the number of processors, these techniques achieve better than 70% utilization on up to 64 processors. These load balancing techniques, however, require substantial amounts of global computation and communication, and are therefore a fundamental obstacle to further scalability. Another interesting result of this work is an estimation of the runtime profile of different components of the algorithm, showing transport operations requiring about 48% of the timestep time, collisions 10.6%, and communications and parallel overhead consuming 24-40%. While the ratio of transport costs to collision costs here is similar to that presented in this thesis, the communication overhead is much larger, suggesting a less efficient concurrent implementation.

6.12 Summary

The results of this work show that the parallel scalability of DSMC implementations is predictable, using a simple model with a small number of parameters. The two primary factors that impact parallel efficiency are load imbalance and communication cost. Load imbalance is primarily determined by initial partitioning, but may be improved with the use of dynamic load balancing. Communication costs increase with the number of processors, and with the size of the problem. The fraction of time spent on communication, however, decreases with problem size. For this reason, large problems scale more efficiently than small problems.

The extension of the DSMC method to include more sophisticated collision and chemical models increases the amount of sequential computation required, which improves scaling properties. On the other hand, certain models, such as discrete Larsen-Borgnakke, may require large and complex data structures for each particle. The cost of particle communication will therefore be greater for these simulations, decreasing parallel efficiency.

The amount of sequential simulation time required for a three-dimensional DSMC timestep is proportional to the cube of gas pressure, and the computation required for a three-dimensional simulation to converge is proportional to the fourth power of pressure. Even for very efficient parallel implementations, computational requirements for high-pressure simulations quickly become prohibitive. Simulations above 6.65 Pa/50 mTorr are infeasible, even on the largest parallel machines currently available.

The models presented here made it possible to answer the questions in at the beginning of the chapter. For a given machine and physical problem, it is possible to predict parallel performance and efficiency. The effects of changes in physical, algorithmic, and architectural parameters on parallel performance have been quantified. For a given physical problem size, it is also possible to predict runtime and memory requirements. These predictions, along with the model for parallel performance and machine-specific parameters, can be used to estimate the number of processors required to satisfy the constraints of available time and memory.

Chapter 7 Related Experimental Studies

This chapter presents several additional experimental studies of concurrent DSMC simulations. A model of partition connectivity is presented and aspects of concurrent transport in unstructured grids are explored. The effectiveness of dynamic load balancing and automatic granularity control is demonstrated. Parallel scalability is also measured on several different platforms.

7.1 Partition Connectivity

The choice of parameters for initial grid partitioning can have a significant impact on the performance of a simulation. Poor choices can increase both the load imbalance and the communication cost. While dynamic load balancing and partitioning can recoupe most of these costs, it is still important to have a versatile initial partitioner and a good choice of partitioning parameters.

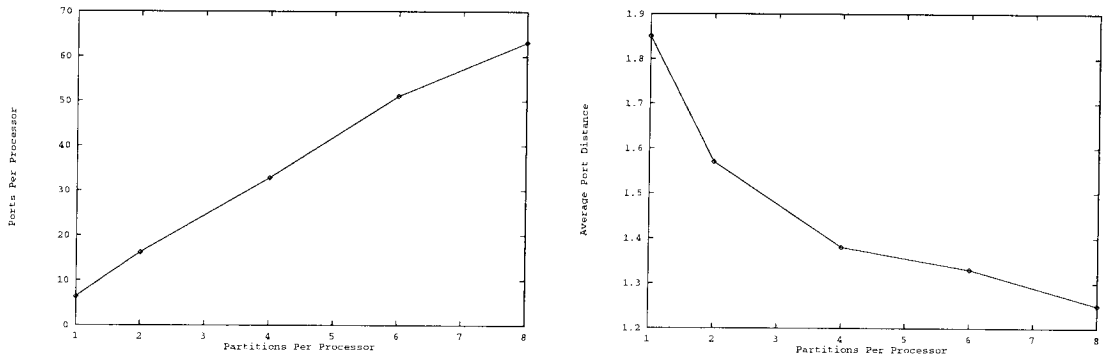

Figure 7.1: Connections Per Processor (left) and average connection distance (right) as a function of Partitions Per Processor

Figure 7.1 (left) shows the number of connections per processor as a function of number of partitions per processor, and Figure 7.1 (right) shows the average con-

nection distance as a function of the number of partitions per processor. Because adding more partitions to a processor increases the number of local (same computer) connections, the average distance decreases with the number of computers.

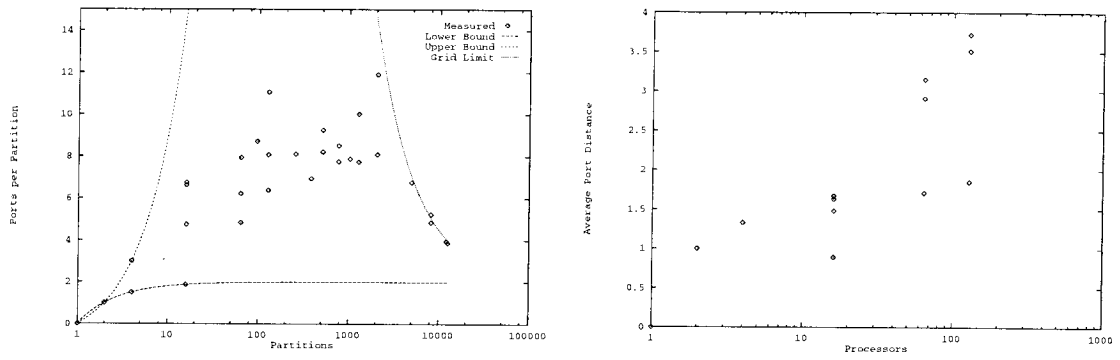

Figure 7.2: Connections Per partition as a function of the number of partitions(left) and average connection distance as a function of number of processors (right)

Figure 7.2 (left) shows the number of connections per partition as a function of the number of partitions, for a 12,540-cell cube-shaped grid. Different values for a particular number of partitions are the result of different partitioning factors. The points on this plot are bounded by three curves, only one of which is grid-specific. The lower bound on the number of connections is given by the equation,

$$s >= 2 - 2/p, \tag{7.1}$$

where s is the number of connections per partition, and p is the number of partitions. This is the number of connections that result from an optimal one-dimensional partitioning. The upper bound, for all-to-all connectivity, is specified by the equation,

$$s <= p - 1. \tag{7.2}$$

As seen in the figure, for small p, the number of connections per partition increases in between the upper and lower bounds. For intermediate p, between 100 and 1000, p holds an approximately constant value in the range 6-10. When p is large, the number

of cells per partition becomes small. When p is equal to the number of grid cells, each shared face requires a connection, resulting in an average of 4 connections per partition for a grid with no external boundaries, and 3.89 connections per partition for this grid. This limit can be approximated with the equation,

$$s < 2 + 2 * \frac{C}{p}, \tag{7.3}$$

where C is the number of grid cells. This limit is also shown in Figure 7.2 (left).

7.2 Unstructured Grid Transport

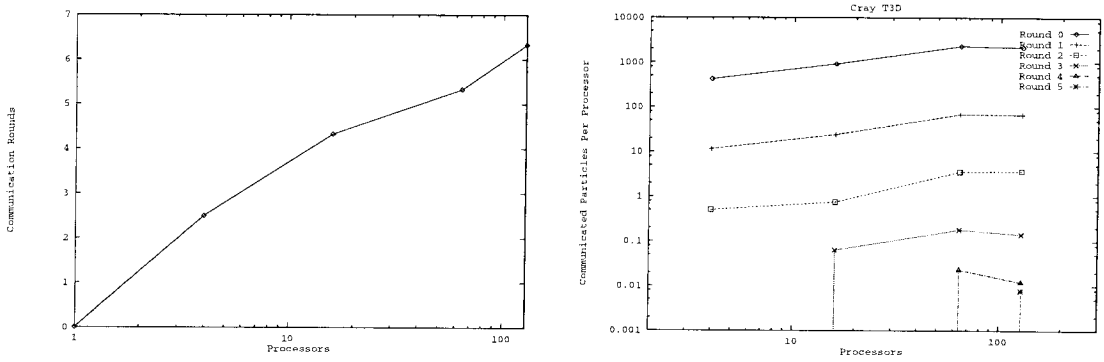

Figure 7.3: Number of Communication Rounds (left) and Particles exchanged, per processor, in each round (right) as a function of the number of processors

As the number of processors used for a computation increases, the number of partitions must also increase. This results in increased communication in two ways. First, it leads to an increase in the surface area of partition boundaries, and thus the probability that a particle will cross a partition boundary. Second, as partitions are smaller and more numerous, particles are more likely to cross the edges and corners of partitions, and thus travel through several partitions in one timestep. This requires additional rounds of nearest-neighbor particle exchanges.

Figure 7.3 (left) shows the number of rounds of particle exchanges as a function of the number of processors. These results are from the simulation of a 12,540-cell

box grid with no net flow, using a number of particles proportional to the number of processors. It can be seen that the number of communication rounds increases to 6.3 for 128 processors. In the next experiment, the number of communicated particles was computed for each round of exchanges, and averaged over all of the processors. The results, shown in Figure 7.3 (right) show that each subsequent round exchanges 10-30 times fewer particles than the previous round. The number of communicated particles, for any given round, can also be seen to increase slowly with the number of processors.

7.3 Parallel DSMC Scalability

Recall that practical DSMC simulations typically involve two phases: startup and statistics-collection. During the startup phase, macroscopic properties change over time as the solution emerges. Once the macroscopic parameters have converged, statistics are collected over several thousand steps in order to obtain smooth and accurate results. During the startup phase, a small number of particles are used (only as many as are required to reach a correct solution). As the number of processors is increased, there is no need to increase the number of particles.

During the statistics-collection phase, however, the goal is to maximize the number of "samples", where a sample is one timestep for one particle. As the ratio of particles to cells increases, the computational overhead associated with each cell is amortized over a larger number of "useful" particle computations. Maximizing the particle processing rate therefore results in the shortest wall-clock-time required for a given level of smoothness. On distributed-memory machines, the use of additional processors makes possible the use of additional particles. It is therefore useful to consider a *scaled speedup*, where the number of particles used is proportional to the number of processors.

In both phases of a computation, the rate of productive work can be measured and compared in terms of the number of particles that can be simulated in a given amount of time. Because of the reduced overhead, this processing rate can actually increase

super-linearly with the number of processors. This is particularly true on machines with small amounts of memory per processor, where single-processor simulations are only possible with very small numbers of particles. While this metric of performance may be misleading from an algorithmic scalability perspective, it is nevertheless a meaningful measure of the amount of "useful work" that can be achieved on existing platforms.

7.4 Load Balancing Experiments

In order to investigate the effectiveness of dynamic load balancing with automatic granularity control, a series of GEC simulations was conducted on the Cray T3D. A high-pressure (13.3 Pa / 100 mTorr), uniform-flow case was considered. Due to the relatively large size of the grid (140,000 grid cells), and the small amount of memory per processor (32 MB), this problem could not be run on fewer than 16 processors. For this reason, the uniprocessor speed could not be determined exactly. An estimate of the uniprocessor speed was obtained by running the full uniprocessor case on one Avalon A12 processor (with 512 MB RAM), then timing small uniprocessor test cases on both the A12 and the T3D. The T3D uniprocessor time was then computed as the A12 time scaled by the ratio of times for the small problem on the two machines. Based on several different tests, this figure is believed to be accurate to within 10%.

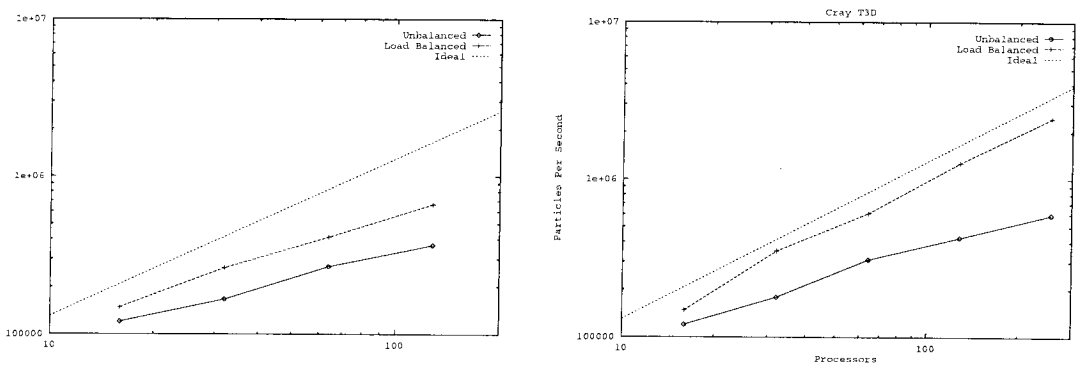

Figure 7.4: Unscaled (left) and scaled (right) performance on the GEC problem

Simulations were conducted on varying numbers of processors for both scaled and unscaled cases. For the unscaled simulations, 200,000 particles were used. The unscaled results are shown in Figure 7.4 (left). Three lines are shown here: the measured speed without load balancing or automatic granularity control; the measured speed with load balancing and granularity control; and the ideal speed, computed by scaling the estimated uniprocessor speed. For these tests, the combination of load balancing and granularity control improved performance by 50-100%, but performance still dropped below 40% of ideal on 128 processors. This can be attributed to the small number of particles per processor for the unscaled case on large numbers of processors. As the number of particles per processor decreases, the fraction of time spent on computational overhead increases, resulting in poor scaling.

Several scaled-particle simulations were also conducted, using 12,500 particles per processor. These results are shown in Figure 7.4 (right), again with unbalanced, balanced, and ideal speeds. Here, the unbalanced performance quickly drops to 26% of ideal, but with load balancing and granularity control, performance remains above 70% of ideal. On 128 processors, the combination of load balancing and granularity control resulted in a 3x performance improvement, resulting in performance that was 77% of ideal.

7.5 Automatic Granularity Control Experiments

This section presents tests that were conducted in order to evaluate the effectiveness of the automatic granularity control technique described in Section 5.5. The scaled-particle simulation was executed on 128 processors with 3.2 million particles, both with automatic granularity control, and without, using different numbers of partitions per processor. The GEC Grid was initially statically partitioned for one partition on each of 128 processors, and tests were conducted using the same initial partitioning, both with and without automatic granularity control. For the simulation with automatic granularity control, partitions were automatically divided only when deemed appropriate by the load balancing technique. For the simulations without automatic

granularity control, each initial partition was repeatedly split in order to obtain a specified number of partitions per processor (1,2,4,8, or 16), and then the load balancing method continued without any further splits. This approach yields the most uniform granularity possible for the given initial partitioning. In each of these cases, the performance, in particles per second, was measured both before and after dynamic load balancing.

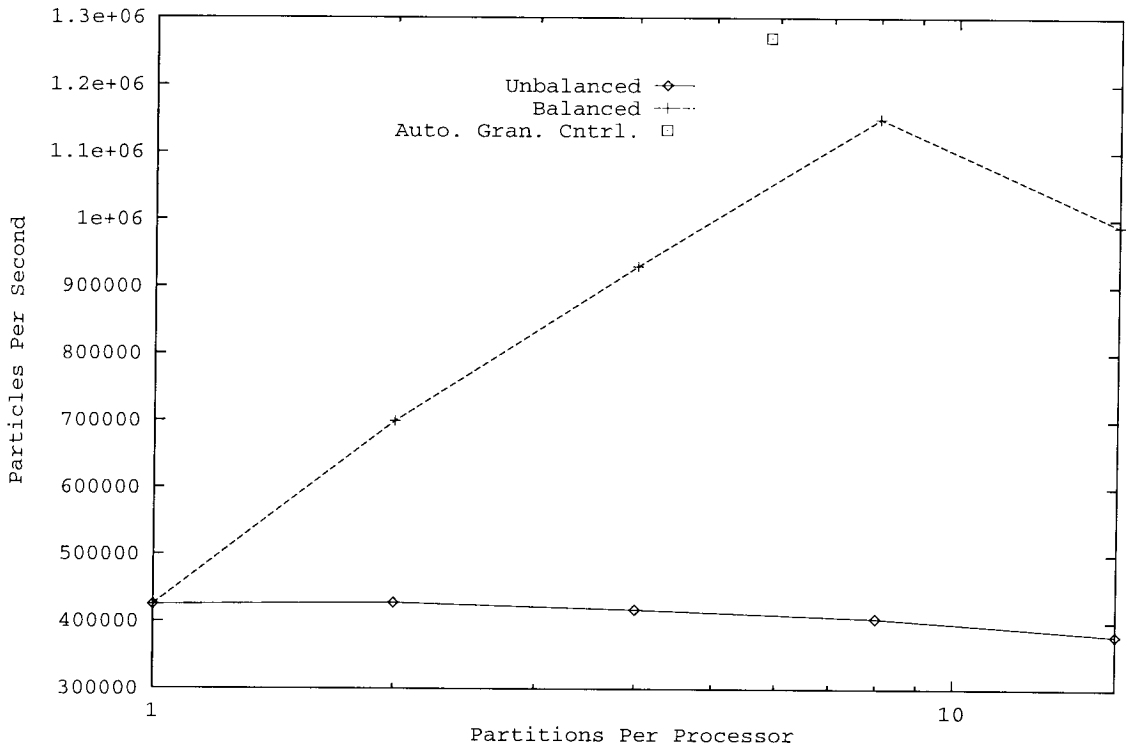

Figure 7.5: Performance as a function of partitions per processors

Figure 7.5 shows the results of these experiments. The decrease in performance of the unbalanced case reflects the increased overhead of the additional partitions on the same processor, which is fairly small, as inter-processor communication is not increased. With only one partition per processor, load balancing cannot make any improvement. Up to 8 partitions per processor, performance improves with more partitions per processor, as load balancing has more flexibility in transfer selection. Above 8 partitions per computer, however, the increased overhead of non-local communication is greater than any improvements from load balancing, and performance

deteriorates.

Without automatic granularity control, the best performance is obtained with the use of 8 partitions per processor. The automatic granularity control technique yielded an average of 5.84 partitions per processor, and a 10% better performance, with 27% fewer partitions. The performance improvement is the result of two factors. First, fewer partitions are required and thus the volume of communication is reduced; second, the approach guarantees that no partition is so large as to impede the load balancing method.

In addition to the performance improvement that results from the use of automatic granularity control, it is important to note the reduction of parameters. Without automatic granularity control, it is necessary to specify the desired number of partitions per processor. An optimal value for this parameter can only be determined by extensive tests. A sub-optimal number of partitions per processor could further reduce performance by 12%. In general, dynamic load balancing and automatic granularity control will yield substantially better performance than static manual partitioning.

7.6 Architecture Comparison

This section evaluates the performance and scalability of DSMC simulations on a variety of platforms. These simulations are at an operating pressure of 100mTorr. Due to memory constraints, it was only possible to use small fraction of the number of grid cells required to satisfy the cell size criteria described in Section 4.1.1. Though the communication requirements of these examples may underestimate the communication requirements of more stringent simulations, general simulation properties are still representative. In each case, the number of particles used in the simulation is proportional to the memory available to those processors. In other words, as the number of processors is increased, the number of particles is increased so as to maintain a constant fraction of memory usage. Performance is measured in terms of particles per second, or the rate at which particles can be processed for each timestep. For ideal parallel scaling, this quantity will increase linearly as a function of number of

processors.

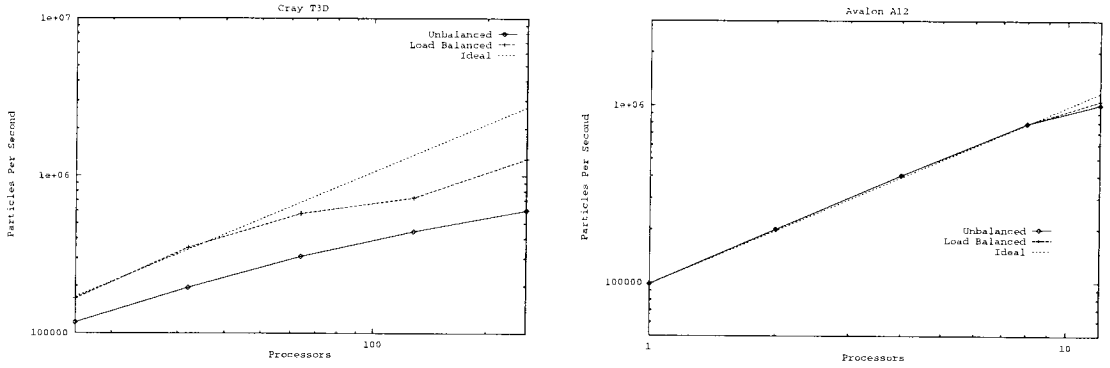

Figure 7.6: Scalability results for the Cray T3D (left) and the Avalon A12 (right).

Figure 7.6 (left) shows performance on the Cray T3D, which is composed of 256 200-MHz 21064 DEC Alpha processors, with 64 MB RAM each, but without secondary or tertiary caches. Due to the small memory size, it is impossible to run this problem on fewer than 16 T3D processors, as the GEC grid alone requires approximately 16 MB. The ideal T3D performance is therefore calculated by linearly scaling the balanced 16-processor performance. This figure clearly shows the benefits of load balancing on large numbers of processors.

Figure 7.6 (right) shows performance on the Avalon A12, which is composed of 12 400 MHz 21164 DEC Alpha processors, each with 500 MB RAM, a 96k secondary cache, and a 1 MB tertiary cache. Due to the small number of processors on the A12, the effect of load balancing is much smaller than on the T3D, while the A12 processors are 3 to 4 times faster than the T3D processors. Due to severe memory constraints on the T3D, it is only possible to achieve 50% load balance there, compared to 85% on the A12. These differences account for what is almost an order of magnitude difference in per-processor speed between the A12 and the T3D.

Figure 7.7 (left) shows the performance results on a network of 5 Dell PC's, each containing 4 200 MHz Intel Pentium Pro processors and 1 GB RAM. Note that this platform uses a heterogeneous communication system, with socket communication between cabinets and shared-memory communication within cabinets. For this plat-

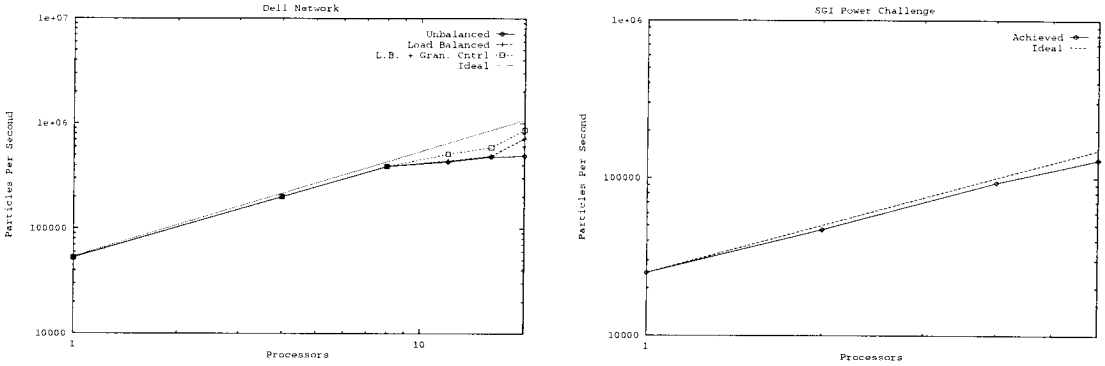

Figure 7.7: Scalability results for a network of multiprocessor Dell PC's (left) and the SGI Power Challenge (right).

form, the effects of dynamic load balancing are shown, both with and without automatic granularity control. Due to the combination of multiple cabinets and multiple processors within a cabinet, it is advantageous to start simulations with one partition per cabinet, and to allow the load balancing algorithm to split partitions in order to achieve both full utilization within a cabinet and load balance across cabinets. Efficient simulation was possible with all 20 processors.

Performance results on the SGI Power Challenge are shown in Figure 7.7 (right). This machine uses 14 75-MHz R8000 processors and 2 GB RAM. With just one cabinet, there is no need for dynamic load balancing.

The results for each platform are combined in Figure 7.8. The highest raw performance was achieved on the Cray T3D, with 1.28 million particles per second on 256 processors. More available memory for each processor would allow for a more efficient load balance, which could as much as double the performance on 256-processors. Of the machines considered in this study, the Avalon A12 demonstrated the highest per-processor performance, with 94,000 particles per second per processor. The Dell network also achieved excellent performance, the use of commodity components making it a particularly cost-effective platform.

As predicted by the models presented in Chapter 6, the primary constraint on the efficiency of concurrent DSMC simulations is the load balance. For adequately sized

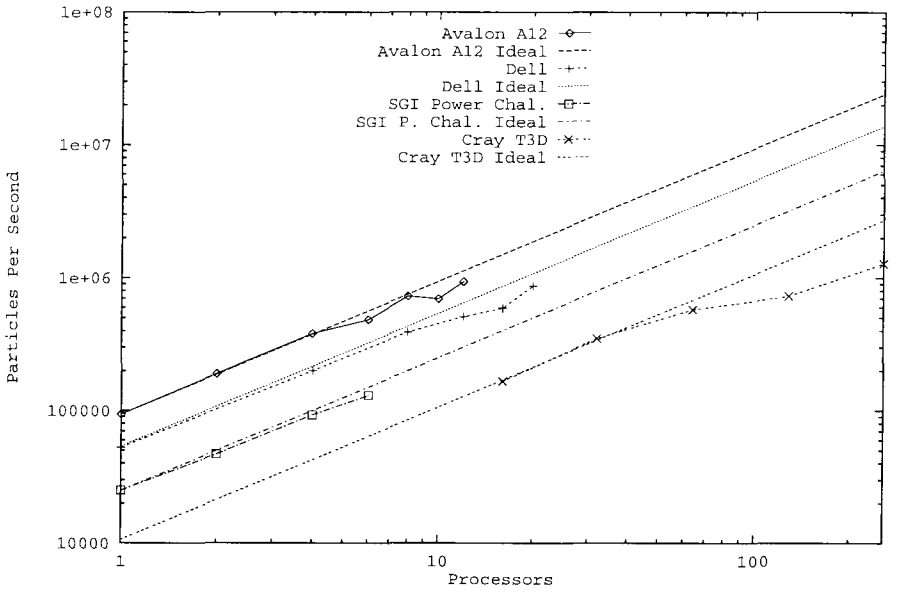

Figure 7.8: Performance on the Cray T3D, Avalon A12, Dell network, and SGI Power Challenge

problems, the losses due to communication are generally less than those due to load imbalance. The minimum number of processors on which simulations were possible for the Cray T3D is also consistent with predictions from the model in 4. Calculation of the architecture-specific parameters for the Avalon A12 and Dell platforms would also allow for the prediction of scalability of these simulations on larger numbers of processors.

7.7 Related Work

The concurrent DSMC method developed by Dietrich and Boyd uses domain decomposition, unstructured grids and ray tracing [26]. The requirement of multiple communication rounds is described. A parallel efficiency above 90% is achieved on up to 400 IBM SP2 processors when scaling the number of particles with the number of processors. Using a constant total number of particles, efficiency drops to 20% on 400 processors.

Yokokawa has developed a parallel DSMC implementation based on grid parti-

tioning [107]. Two-dimensional results were obtained on a Fujitsu AP1000, and a speedup of 42 was obtained on 64 processors. Wang *et al.* present a scalability study of their PIC/MC technique that couples self-consistent field solution with Monte Carlo collisions. Parallel efficiencies were measured to be above 90% for up to 256 Cray T3D processors, 512 Intel Paragon processors, and 512 Intel Delta processors. This technique uses Cartesian grids with spatial domain decomposition.

Matsumoto and Tokumasu have performed parallel molecular dynamics calculations of internal energy exchanges for N_2 collisions. Using these results in a three-dimensional parallel DSMC implementation, they achieved 60% parallel efficiency on 64 processors. This work also demonstrates the fact that DSMC efficiency on a given number of processors improves with problem size [65].

7.8 Summary

This chapter demonstrated a model for partition connectivity as a function of partitioning parameters. Concurrent particle transport in unstructured grids was discussed. Experiments were presented that demonstrated the practical utility of dynamic load balancing and automatic granularity control on realistic simulations. Simulation performance on several concurrent architectures was also presented.

Chapter 8 Technology Demonstrations

This chapter presents several applications of the concurrent DSMC technology. A variety of systems, including semiconductor manufacturing and high-altitude spacecraft flight, can be addressed with this technique.

8.1 Semiconductor Manufacturing Applications

One of the primary goals in the design of a plasma reactor is to ensure flow uniformity above the wafer. If flow parameters, such as density and speed, are different across the wafer, etching and deposition rates will vary accordingly. Non-uniform etching and deposition rates, in turn, can drastically reduce the yield of a production line.

This section presents results from a series of simulations of low-pressure neutral flow in the GEC Reference Cell reactor shown in Figure 8.1 (left). The same initial grid, shown in Figure 8.1 (right), was used for each configuration. Simulations were conducted in several different flow configurations, and the uniformity of gas properties in the vicinity of the wafer was compared between configurations.

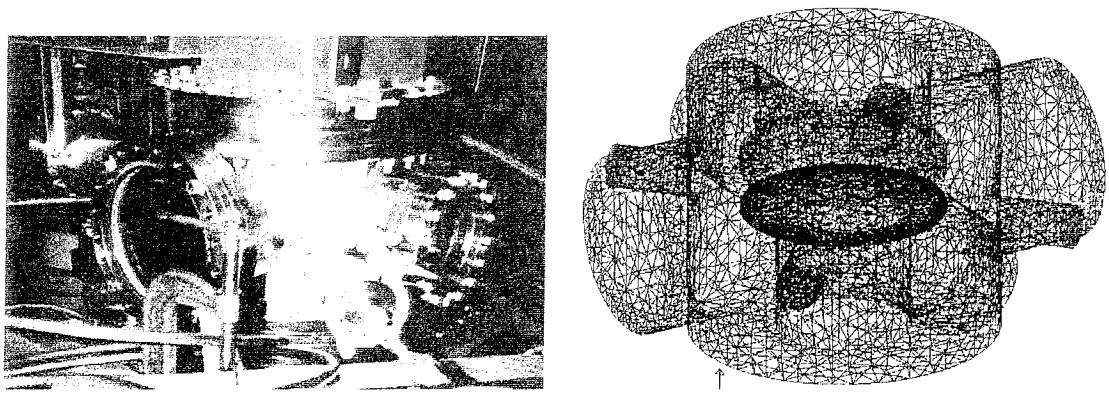

Figure 8.1: The Gaseous Electronics Conference (GEC) Reference Cell Reactor (left) and tetrahedral grid used to represent it (right)

8.2 Simulation Results

This section presents results from six simulations of the GEC Reference Cell Reactor, used to determine the effects of chamber configuration on flow uniformity above the wafer. In each of the simulations, the walls of the reactor are assumed to be accommodating at 300K. Simulations were performed using argon with $\alpha = 0.31$, $\beta = 0.714$, $T_{ref} = 273K$, and $\sigma_{ref} = 5.307 \times 10^{-19} m^2$. In the first four configurations, the wafer was held at 300K, while it was heated to 400K for the final configuration.

The outflow surface in each configuration was specified to adapt in order to maintain a pressure of 2.66 Pa (20 mTorr) at a point just above the center of the wafer, as described in Section 3.2.1. Gas was injected at 300K, and a timestep of 4×10^{-6} seconds was used.

Table 8.1: GEC Reference Cell Simulation Configuration Parameters

Configuration	Inflow	Flow Angle	n_i	v_i	T_w
		Degrees	m^{-3}	m/s	K
Horizontal-45	Small	45	5×10^{21}	37.6	300
Horizontal-135	Small	135	5×10^{21}	37.6	300
Vertical	Top	180	6.4×10^{20}	29.4	300
Showerhead	Showerhead	180	6.4×10^{20}	29.4	300
Heated Wafer	Showerhead	180	6.4×10^{20}	29.4	400

The following sections consider the different flow configurations in further detail. Table 8.1 below shows the parameters used in each of the configurations, the inflow port, the angle between the inflow and outflow ports, the inflow number density n_i, the inflow speed v_i, and the wafer temperature T_w. For each case, two 2-D slices through the reactor are shown, one above and parallel to the wafer (horizontal), and the other perpendicular to the wafer (vertical). Projections of CAD surfaces onto the cut planes appear as solid lines. In the horizontal cut, the largest complete circle in the center of the reactor indicates the edge of the wafer. In the vertical cut, the wafer appears as a thin horizontal element in the center of the reactor.

The plots are colored by pressure, with a scale from 0 (blue) to 5 (red) Pascal,

and 20 contour lines are shown for this interval. A color key is shown in Figure 2.17 (right). The orientation and spacing of the contour lines provides an indication of the uniformity of the flow above the wafer. Closely spaced parallel contour lines indicate steep gradients in density, while scattered and separated lines suggest reasonable uniformity. Concentric contour lines centered at the center of the wafer indicate a radial flow pattern. These results suggest which configurations are most appropriate for semiconductor manufacturing purposes.

Horizontal 45-Degree Configuration

In this configuration, gas was injected through the small port appearing on the upper left of the plane, and exhausted through the large port on the left, 45-degrees apart. The inflow density was $5 \times 10^{21} m^{-3}$ and the inflow speed was 37.6 m/s. Figure 8.2 shows the gas pressure in horizontal and vertical planes for this configuration. The pressure in the inflow region is high compared to the rest of the reactor and decreases steadily towards the outflow region. The effects of the inflow can be clearly seen above the portion of the wafer closest to the inflow port.

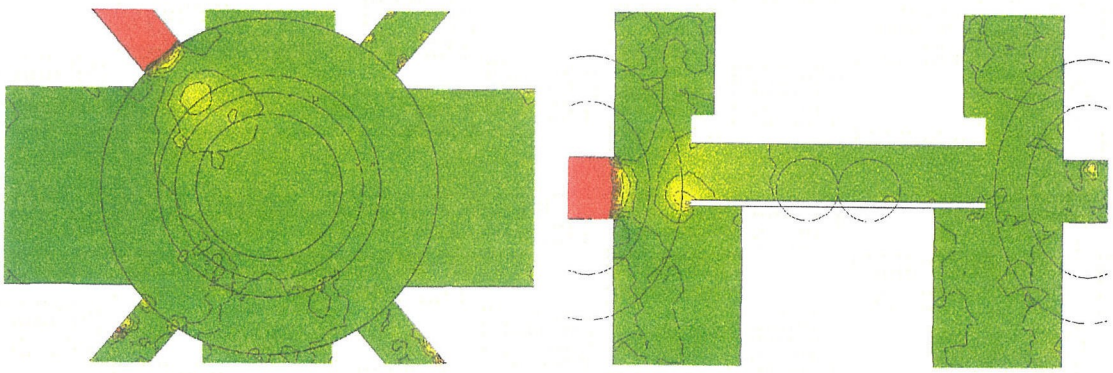

Figure 8.2: Pressure in the horizontal (left) and vertical (right) planes for the horizontal 45-degree configuration

Horizontal 135-Degree Configuration

In this configuration, gas was injected through the small port at the bottom left of the plane, and again exhausted through the large port on the left, separated by 135 degrees. This configuration is almost half-symmetric, with gas flowing across the wafer. Figure 8.3 shows the gas pressure in horizontal and vertical planes. Notice the high density in the inflow region, and a slightly lower density in the exhaust region. Since the effects of the inflow port are much stronger than the effects of the exhaust port, the results of this configuration are almost a mirror image of those in the previous section. In contrast, however, the effect of the inflow port continues further across the wafer than in the previous case, resulting in a stronger density drop and less uniformity.

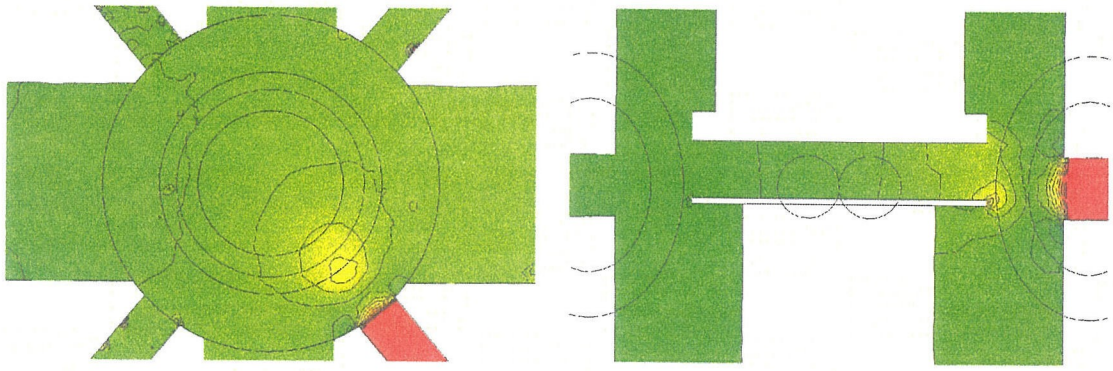

Figure 8.3: Pressure in the horizontal (left) and vertical (right) planes for the horizontal 135-degree configuration

Vertical Configuration

In this configuration, gas was injected at the top of the reactor with a density of $6.4 \times 10^{20} m^{-3}$ and a speed of 29.4 m/s, and exhausted at the bottom of the reactor. This configuration is quarter-symmetric, and could be reasonably approximated as rotationally symmetric about the vertical axis. Figure 8.4 shows uniform gas pressure across the entire width of the reactor, especially above the wafer. Because the flow is vertical in this configuration, there is no reason to expect non-uniformities above the

horizontally oriented wafer.

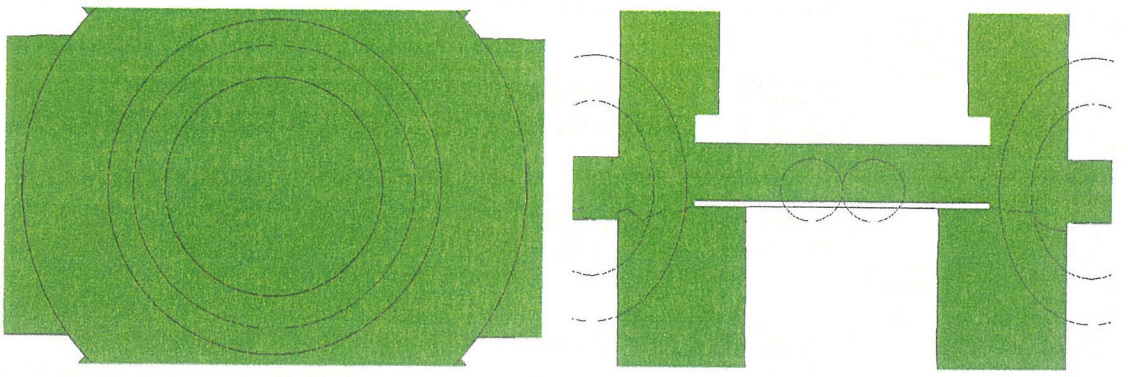

Figure 8.4: Pressure in the horizontal (left) and vertical (right) planes for the vertical configuration

Showerhead Configuration

In this configuration, gas was injected above the wafer, with a density of $6.4 \times 10^{20} m^{-3}$ and a speed of 29.4 m/s, and exhausted at the bottom of the reactor. This configuration is also quarter-symmetric, and could be reasonably approximated as rotationally symmetric about the vertical axis. Figure 8.5 shows the gas pressure in horizontal and vertical planes. The concentric contour lines in the horizontal plane, and vertical lines in the vertical plane, are clear indication of a radial flow profile, where pressure is highest at the center and lowest at the edges of the wafer. The asymmetry of the horizontal ports does not noticeably affect the rotational symmetry near the wafer for this configuration.

Heated Wafer Configuration

This configuration is identical to the previous configuration, with the exception that the wafer was heated to 400K, while the other solid surfaces of the reactor were held at 300K. Figure 8.6 shows the gas pressure in horizontal and vertical planes. Again, the flow is predominantly radial and axisymmetric. Because of the wafer heating, the temperature above the wafer is greater than in the previous case, but the density is

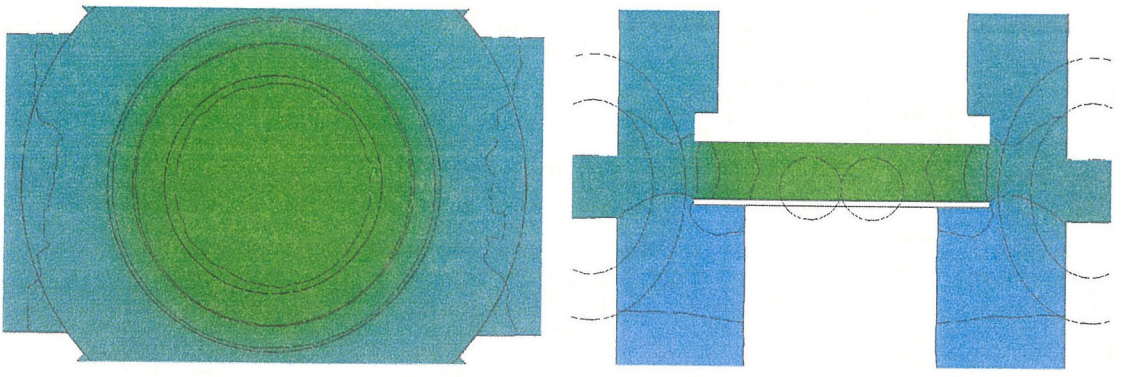

Figure 8.5: Pressure in the horizontal (left) and vertical (right) planes for the showerhead configuration

proportionally lower, yielding the same pressure above the wafer. This is enforced by adaptive pressure regulation, which maintains a constant pressure above the wafer for all configurations.

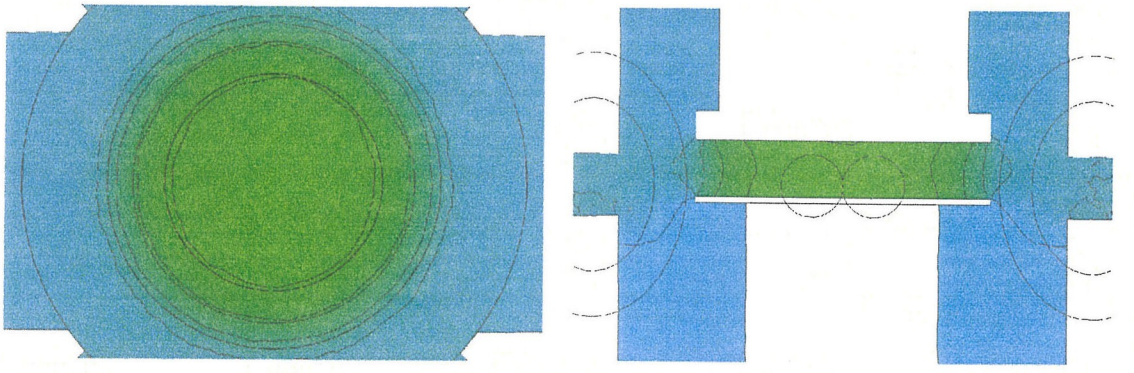

Figure 8.6: Pressure in the horizontal (left) and vertical (right) planes for the heated wafer showerhead configuration

8.2.1 Convergence

For the purposes of this study, convergence was ascertained from examination of overall system parameters. The longest timescale in these simulations is the convergence of the pressure regulation method, because it is governed by the acoustic timescale. Changes in the exhaust probability must propagate through the entire system be-

fore the regulation can converge. Convergence of the pressure regulation algorithm therefore implies convergence of the system as a whole.

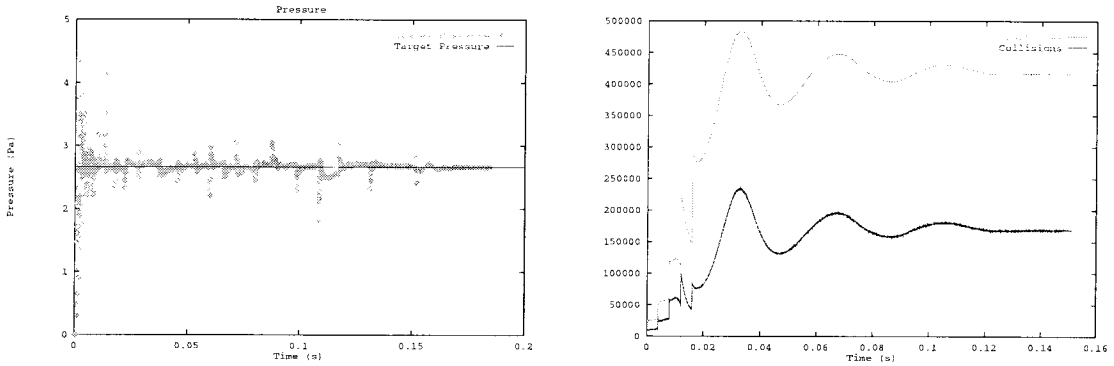

Figure 8.7: System energy (left) and particles and collisions (right) as functions of simulation time

Figure 8.7 (left) shows system pressure as a function of time for the horizontal 135-degree simulation. Figure 8.7 (right) shows the number of particles and collisions per timestep for the same simulation. The pressure regulation-induced oscillation can be seen here as well. The other feature of this plot is the periodic doubling in the number of particles, as described in Section 3.2.2. Initially, a small number of particles is used to rapidly obtain an approximate solution. Particles are periodically replicated in order to provide smoother solutions. Using this particle replication method, convergence can be obtained substantially faster than by running a simulation with a large number of particles from the beginning.

The results presented in Figures 8.2-8.6 were averaged over several thousand steps, starting from when the pressure settled to within 10% of the target pressure. After pressure convergence, the number of particles was further increased to several million, to ensure accuracy and smoothness of the results.

8.2.2 Analysis

While the preceding figures serve to provide a qualitative description of flow uniformity above the wafer, quantitative comparisons of simulation results are also possible.

The following figures show flow parameters in the horizontal plane just above the wafer. Two lines are considered, both passing through the center of the chamber: one is parallel to the centerline of the large outflow port in the horizontal configurations (from left to right), and the other is parallel to the centerline of the inflow ports for the horizontal configurations (from upper left to bottom right).

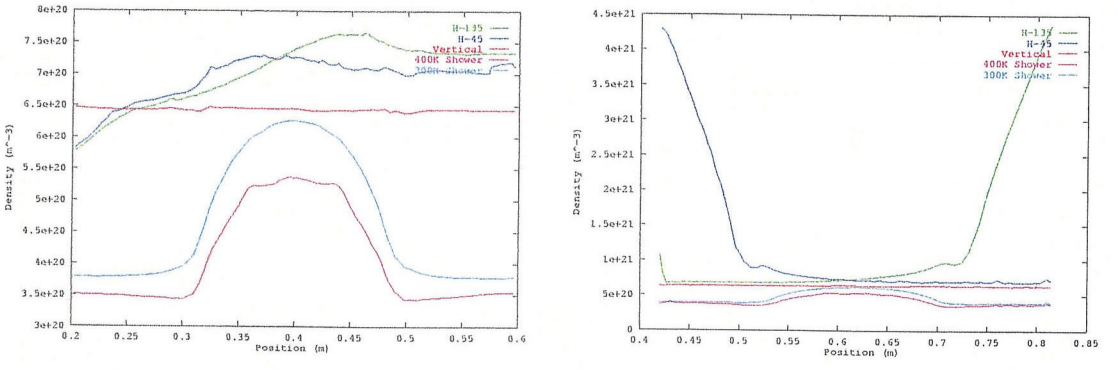

Figure 8.8: Particle density as a function of position along a line above the wafer, passing through the inflow ports in the horizontal configurations (left), and along a line through the outflow port in the horizontal configurations (right)

Figure 8.8 (left) shows the density along the first line, where the wafer extends from approximately 0.33 m to 0.47 m. The decrease in density towards the left of the graph corresponds to the pressure drop along the outflow port. The density peak for the two showerhead configurations is a direct result of the showerhead inflow, which causes a radial flow pattern. The density of the heated-wafer showerhead configuration is lower than that of the regular showerhead configuration, as is required for both to achieve the same pressure above the wafer. In the vertical configuration, very little variation is seen across the entire width of the reactor.

The second line, running through the inflow ports, is shown in Figure 8.8 (right), where the wafer extends from 0.53 m to 0.67m. The most noticeable feature in this plot is the large density drop along the inflow tubes for the horizontal configuration. As in the previous plot, the radial flow profile can be observed for the showerhead configurations, and uniform flow persists for the vertical configuration.

Average particle speed along the horizontal line is shown in Figure 8.9 (left). For the horizontal configurations, the increase in speed from right to left corresponds to the acceleration towards the outflow port. The profiles for the showerhead configurations are indicative of a radial flow pattern. Unlike the other configurations, the speed peaks for the vertical configuration correspond to increases in vertical, or downward, speed, and these occur just past the edges of the wafer, where gas has an unobstructed path from inlet to outlet.

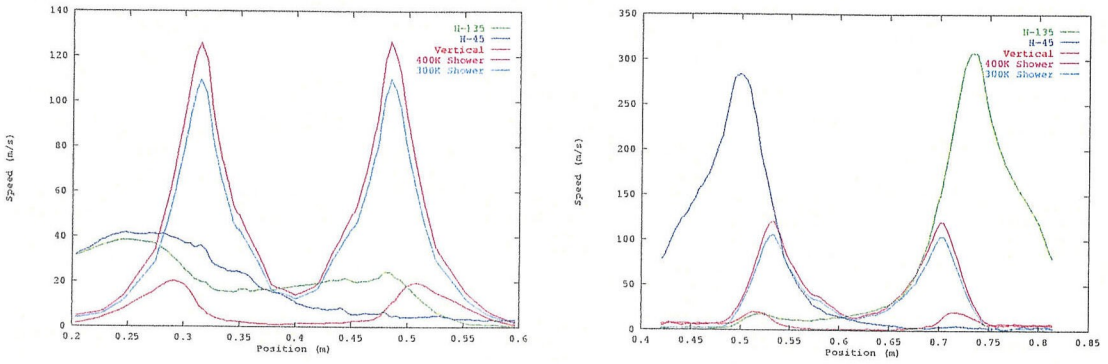

Figure 8.9: Average particle speed as a function of position along a line above the wafer, passing through the inflow ports in the horizontal configurations (left) and through the outflow port in the horizontal configurations (right)

Speed along the diagonal line is shown in Figure 8.9 (right). Here, the flow can be seen to accelerate along the inflow tubes for the horizontal configurations, from left to right in the 45-degree configuration, and from right to left in the 135-degree configuration. The showerhead configurations again display radial flow profiles, and the vertical configuration shows little flow above the wafer.

From the preceding figures, several general conclusions can be drawn. For the vertical and showerhead configurations, flow is generally axisymmetric, as demonstrated by the similarity between the plots along two different lines. The flow is primarily radial above the wafer for the showerhead configurations, while it is directly downward for the vertical configuration. The expected density drops are seen from inflow to outflow for the horizontal configurations, as are the speed changes in response to the

changing geometry of the reactor. Above the wafer, the showerhead configurations exhibit a 50% radial drop in density, and a 20% decrease in speed. The horizontal configurations exhibit a 30-50% variation in density, and a 10% variation in speed. The 135-degree configuration shows better uniformity than the 45-degree configuration, along the horizontal line across the wafer. In all cases, the vertical configuration provides extremely uniform flow above the wafer.

8.2.3 Related Work

Bukowski, Graves, and Vitello have conducted two-dimensional simulations of inductively coupled plasma in the GEC Reference Cell Reactor [22]. Separate equations are solved for ions and neutral species, while the electromagnetic fields are solved self-consistently. Results are compared with experimental data.

Font and Boyd have conducted simulations of a helical etch reactor in a variety of configurations [30]. The DSMC method is used for modeling neutrals and ions, while electrons are modeled with a background condition. Their results show the effects of different nozzle locations.

An inductively coupled plasma reactor with complex geometry have been studied using the Hybrid Plasma Equipment Model (HPEM), which incorporates electromagnetic field solution, electron Monte Carlo simulation, and fluid-chemical kinetics simulation [59]. These results show excellent agreement with experimental data.

Hitchon, *et al.*, have implemented a nonstatistical technique for studying the transport of sputtered neutral particles [76]. Their "convective scheme" (CS) has also been use to solve the Boltzmann equation in a cylindrical geometry [75]. Hitchon has performed a detailed study of the sensitivity of simulations to collision techniques [40]. Two-dimensional results have been obtained for ion densities in plasma chambers with simple geometries, solved self-consistently with the electrostatic potential [54, 41]. Techniques have also been presented that permit the results of computationally inexpensive simulations to be extrapolated to obtain results comparable to those from lengthy simulations [39].

Wadsworth has applied a parallel, three-dimensional code to simulation of a simplified, quarter-symmetric version of the GEC reference cell reactor [96]. This implementation uses ray tracing and grid-based partitioning under PVM, and includes static load balancing. Bartel has performed axisymmetric simulations of plasma reactors, with multiple species and constant electromagnetic fields [9]. In this technique, electron distributions are solved by a separate fluid model. Economou and Aydil have monitored etch rates and surface chemistry processes in experimental plasma reactors [27, 4, 5].

Nanbu and Uchida have combined the PIC and DSMC techniques to support sputtering simulations with self-consistent electric fields and multiple species [71]. This technique has been applied to one-dimensional problems with decoupled neutral flow, iion formation, sputtering, and sputtering atom transfer. Nanbu and Kondo analyzed a three-dimensional magnetron discharge using the same PIC/MC technique, determining the effects of magnetic field and pressure on plasma density [72].

A number of techniques have been used to model plasmas. Graves *et al.* have modeled microwave plasmas in cylindrical geometries using a hybrid approach [35]. This technique uses particles to model ions and a fluid model for electrons. Microwave power deposition is modeled with an *ad hoc* technique. Barone and Graves simulated plasma-surface interactions using molecular dynamics (MD) techniques [8].

Currently, three methods are being investigated for handling trace species and the difference in timescales between ions and neutrals. These are species-selective particle transport and weighting [13, 21], spatially dependent weighting factors, and overlay techniques. Validation of these techniques is difficult in the absence of high-quality experimental data for realistic systems.

8.2.4 Summary

This chapter has presented the basic concepts behind a novel concurrent DSMC method and demonstrated the method on three-dimensional simulations of the GEC reference cell. While much progress has been made, this work is a first step in a

comprehensive plasma-simulation project. A number of directions are active areas of research. The use of tetrahedral grids is consistent with finite volume continuum methods, facilitating the integration of the two techniques. The inclusion of ions and self-consistent electric fields will significantly enhance the ability to model a variety of engineering problems associated with plasma reactors.

The main advantage of this technique is that it addresses the entire design cycle of a simulation. Each step in the process has been optimized in order to minimize the amount of human effort and computer time required. Industry-standard commercial tools are used where possible, and *scalable* concurrent algorithms enable the efficient use of a wide variety of platforms. Modular software design facilitates the incorporation of new or proprietary models for physical and chemical processes. Methodologies that do not address each of these issues are limited in their viability for widespread industrial application.

The results of this chapter show that large-scale simulations of realistic plasma reactors are possible within realistic engineering timescales. Using a relatively small parallel machine such as a 14-processor SGI Power Challenge or a 72-processor Intel Paragon, simulations have been conducted on a variety of proprietary Intel and Tegal reactors that have had a direct impact on the efforts of process engineers. These problems can typically be configured and simulated in about a week using several million particles. On newer architectures, such as the 12-processor Avalon A-12, simulations can be completed in 2-3 days. Good performance and scaling have been observed on as many as 512 processors of a parallel computer and 25 networked workstations.

While only single-species neutral flow has been presented here, significant progress has been made in support of complex inelastic collisions, internal energy modes, and reacting chemistry. Full support for all of these features will allow industrial designers and process engineers to evaluate new reactor designs and configurations quickly, accurately, and cost-effectively.

8.3 Aerospace Applications

Because Earth's upper atmosphere is composed of low-density gas, the flight of spacecraft through this regime is also in the transition region. The DSMC technique is therefore useful for the simulation of spacecraft reentry. One example of this application is the Skipper mission, a space experiment designed to measure the ultraviolet radiation from the shock-heated gas in the nose region of a small satellite. Figure 8.10 shows diagrams of the inside (right) and outside (left) of the satellite.

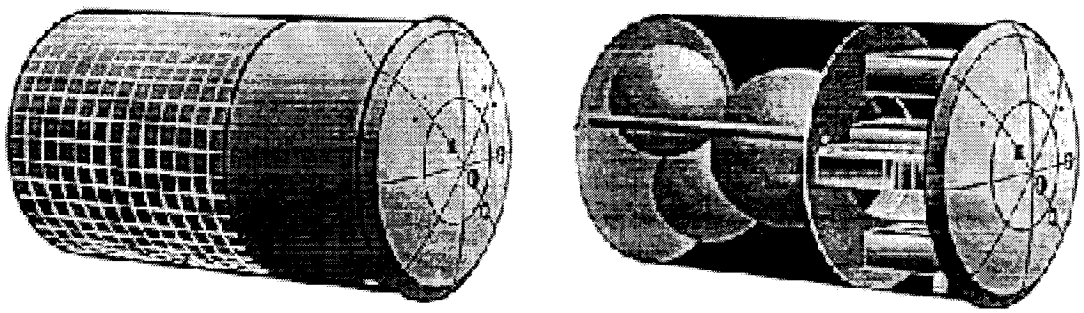

Figure 8.10: Outside (left) and inside (right) diagrams of the Skipper satellite

A central problem is to optimize the vehicle trajectory so as to maximize the data acquired by detectors. Flow field predictions are required at a wide variety of angles of attack; these may then be used by designers to conduct a tradeoff study to obtain the optimal trajectories for data collection.

8.4 Simulation Results

A simulation of the Skipper satellite was conducted using a 7 km/s argon flow. The spacecraft body was configured to be fully accommodating at 300K. The simulations were conducted in 3D, and 2D cuts through the center of the satellite body are shown.

Figure 8.11 shows the temperature results from this simulation. A low temperature region around the surface of the spacecraft is the result of thermal accommodation. The high temperature in the bow shock can also be seen in this figure.

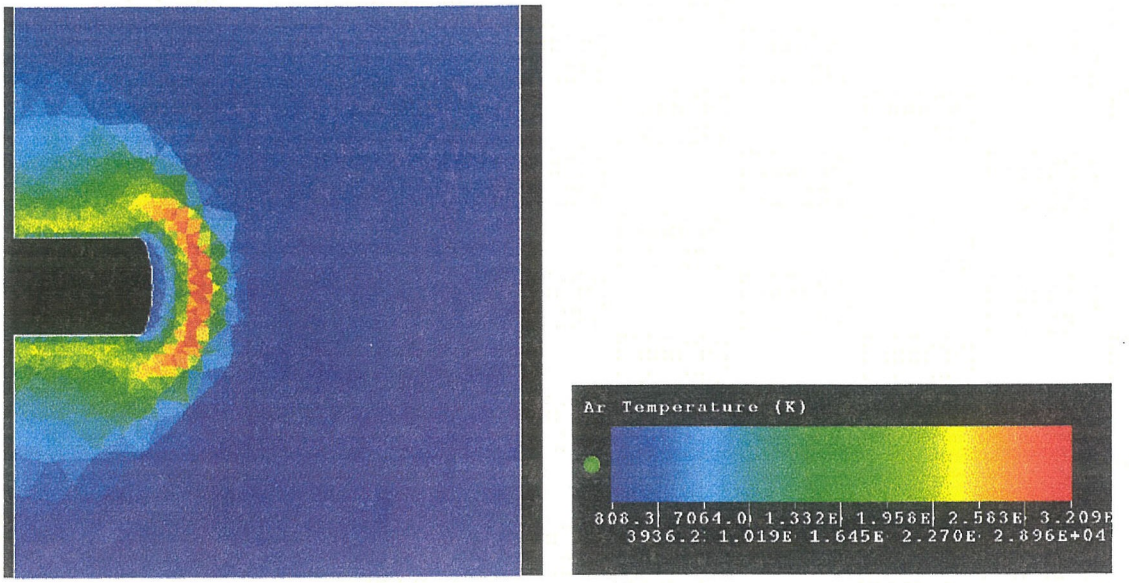

Figure 8.11: Temperature solution (left) and key (right)

Figure 8.12 shows the speed results in the same plane. The 7km/s freestream speed can be seen. The speed drops to zero immediately in front of the satellite.

8.4.1 Related Work

The DSMC method was originally designed for use with aerospace applications. Bird has used sophisticated internal energy models in the modeling of reentry of the Space Shuttle [14]. Ivanov, *et al.*, have performed extensive studies of high-altitude reentry capsules [51, 47] and other hypersonic rarefied flows [52]. They have also studied the hysteresis effect in the transition between regular and Mach reflection of planar shock waves [33, 49]. Nguyen *et al.* present simulations of hypersonic channel-wedge flows, along with grid resolution and convergence studies for the monotonic Lagrangian grid method [74].

Nanbu presents simulations of hypersonic flow around a disk, using regular multi-block grids [70]. Koura has calculated coefficients for the VSS model for air species [58]. In collaboration with Legge, they have also performed simulations of force and heat transfer for flow around a disc and shown good agreement with experimental

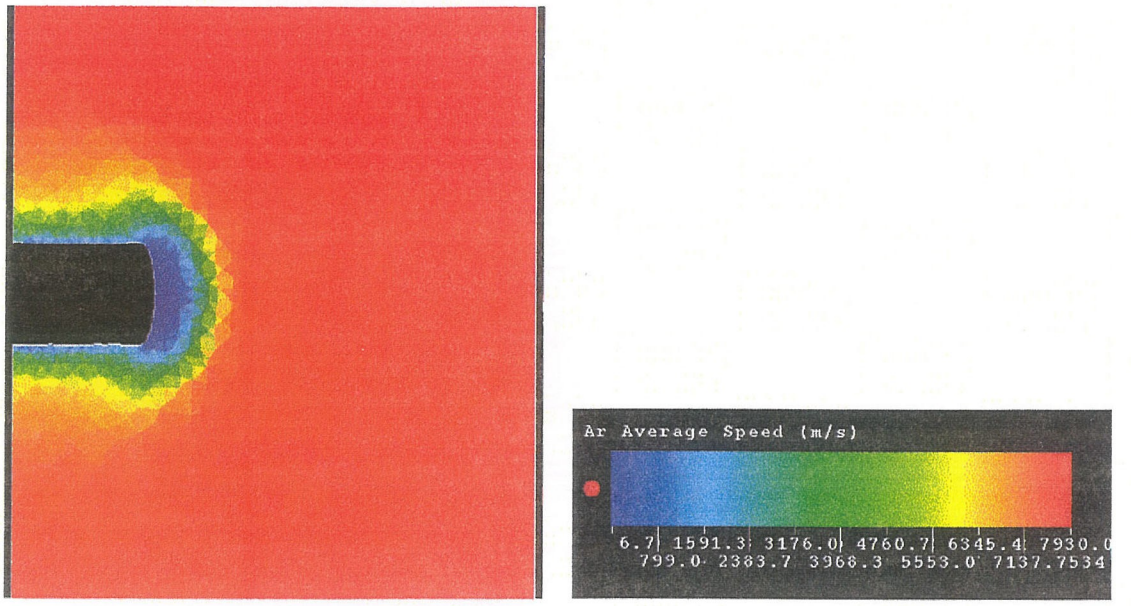

Figure 8.12: Speed solution (left) and key (right)

data [60].

Marriott and Bartel have compared two sophisticated chemistry techniques, Maximum Entropy (ME) and Larsen-Borgnakke, on two-dimensional axisymmetric grids [63]. Their results show that the two approaches yield significantly different solutions under certain conditions. Boyd has developed models for energy transfer between vibrational and translational modes, simulating flow over a two-dimensional wedge [19]. He has also studied the effects of rotational degrees of freedom for jets of iodine vapor impinging on blunt bodies [20]. These results agree well with experimental data for moderate temperatures (100 - 500K), but less satisfactorily for high temperatures.

Another particle technique, Particle-In-Cell, has been used to model ion thruster backflow contamination [87]. The parallel programming methodology in this work is very similar to that of that described in this thesis.

8.4.2 Summary

The use of the DSMC method is particularly appropriate for simulations such as those of the Skipper satellite. The results of this section show that these simulations are

feasible with available computational resources.

Chapter 9 Conclusion

This thesis has presented a framework for understanding the performance characteristics of particle simulations. Application- and architecture-independent models have been developed for predicting runtime and storage requirements for sequential and concurrent simulations. These models are applicable to a wide variety of homogeneous and heterogeneous computational platforms. The models are demonstrated and evaluated in the context of Direct Simulation Monte Carlo (DSMC) simulations for semiconductor manufacturing and aerospace applications.

In addition to the work presented here, a number of additional models have been implemented but have not yet been validated. A framework for supporting arbitrary chemical reactions has been developed, including calculations of reaction probabilities from rate constant information. Binary reactions and dissociations are supported. Infrastructure for supporting surface transport and surface reactions has also been developed. Simplistic models for particle absorption, reemission, surface reactions have been implemented. Several novel reaction types have been developed to support specific industrial needs. Numerical integration of arbitrary particle trajectories has been developed, using the Gaussian-Quadriture method.

One of the chief elements of complexity of this work is that algorithmic analysis alone is insufficient for understanding the performance characteristics of particle simulations. The interaction between the physical and chemical properties of a system and the numerical method being used to model it must be considered in detail.

Equipped with these predictive models, scientists can determine whether DSMC simulations of a given system will be feasible with existing computational resources, or what additional resources will be required. The feasibility boundaries for realistic simulations have been determined. The same analysis techniques could be applied to other numerical methods, such as PIC and Navier Stokes. By combining these analyses, it would be possible to determine which numerical method is most appropriate

for obtaining a desired level of accuracy on a given physical system.

Bibliography

[1] A. Aho, J. Hopcroft, D. Ullman. "The Design and Analysis of Computer Algorithms." Addison-Wesley. Reading, Mass., 1974.

[2] J. Austin and D. Goldstein. "Direct Numerical Simulation of Low-Density Atmospheric Flow on Io." *Bulletin of the American Physical Society Series II.* **39**(9). 1994.

[3] D. J. Alofs, R. C. Flagan, G. S. Springer. "Density Distribution Measurements in Rarefied Gases Contained between Parallel Plates at High Temperature Differences." *Physics of Fluids.* **14**(3). 1971.

[4] E. Aydil, R. Gottscho, Y. Chabal. "Real-time Monitoring of Surface Chemistry during Plasma Processing." *Pure and Applied Chemistry.* **66**(6). 1994.

[5] S. Han, E. Aydil. "A Study of Surface Reactions during Plasma Enhanced Chemical Vapor Deposition of SiO_2 from SiH_4, O_2, and Ar Plasma." *Journal of Vacuum Science and Technology.* 1995.

[6] S. Baase. "Computer Algorithms: introduction to design and analysis." Addison Welsey. Reading, Mass, 1978.

[7] S. Barnard and H. Simon, "A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," *Concurrency: Practice and Experience,* vol. 6, pp. 101–117, 1994.

[8] M. Barone and D. Graves. "Molecular dynamics simulations of plasma-surface chemistry." *Plasma Sources Science and Technology.* 1996.

[9] T. Bartel. "Low Density Gas Modelling in the Microelectronics Industry." *Rarefied Gas Dynamics 19.* Volume 1, Oxford University Press. 1995.

[10] S. Barnard and H. Simon, "A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," *Concurrency: Practice and Experience*, Vol. 6, pp. 101–117, 1994.

[11] G. A. Bird. "Breakdown of Translational and Rotational Equilibrium in Gaseous Expansions." *AIAA Journal.* **8**(11). 1970.

[12] G. A. Bird. "Direct Simulation of the Boltzmann Equation." *Physics of Fluids.* **13**(11). 1970.

[13] G. A. Bird. "Molecular Gas Dynamics." Oxford University Press, 1976.

[14] G. Bird. "Simulation of Multi-Dimensional and Chemically Reacting Flows." *Rarefied Gas Dynamics.* 1979.

[15] G. Bird. "Molecular Gas Dynamics and the Direct Simulation of Gas Flows." Clarendon Press. Oxford, 1994.

[16] C. Borgnakke, P. Larsen. "Statistical Collision Model for Monte Carlo Simulation of Polyatomic Gas Mixture." *Journal of Computational Physics.* (18) 1975.

[17] A. Borodin and I. Munro. "The Computational Complexity of Algebraic and Numeric Problems." American Elsevier. New York, 1975.

[18] J. F. Bourgat, P. Le Tallec, and M. D. Tidriri. "Coupling Boltzmann and Navier-Stokes Equations by friction." *J. Comp. Phys..* 1996.

[19] I. Boyd. "Analysis of vibrational-translational energy transfer using the direct simulation Monte Carlo method." *Physics of Fluids.* **3**(7). 1991.

[20] I. Boyd, G. Pham-Van-Diep, E. Muntz. "Monte Carlo Computation of Nonequilibrium Flow in a Hypersonic Iodine Wind Tunnel." *AIAA Journal.* **32**(5). 1994.

[21] I. Boyd. "Conservative Species Weighting Scheme for the Direct Simulation Monte Carlo Method." *Journal of Thermophysics and Heat Transfer.* **10**(4). 1996.

[22] J. Bukowski, D. Graves, and P. Vitello. "Two-dimensional fluid model of an inductively coupled plasma with comparison to experimental spatial profiles." *Journal of Applied Physics.* **80**(5). 1996.

[23] C. Cercignani. "Theory and Application of the Boltzmann Equation." Scottish Academic Press. Edinburgh/London, 1975.

[24] S. Chapman, T. G. Cowling. "The Mathematical Theory of Nonuniform Gases." Cambridge University Press. New York, 1952.

[25] M. Crowley, T. Darden, T. Cheatham, and D. Deerfield. "Adventures in Improving the Scaling and Accuracy of a Parallel Molecular Dynamics Program." *Journal of Supercomputing.* 1997.

[26] S. Dietrich and I. Boyd. "Scalar and Parallel Optimized Implementation of the Direct Simulation Monte Carlo Method." *Journal of Computational Physics.* 1996.

[27] D. Economou, E. Aydil, G. Barna. "In Situ Monitoring of Etching Uniformity in Plasma Reactors." *Solid State Technology.* April, 1991.

[28] G. S. Fishman, "Multiplicative congruential random number generators with modulus 2**b: an exhaustive analysis for b = 32 and a partial analysis for b = 48," *Math. Comp.*, Vol. 189, pp. 331–344, 1990.

[29] J. Foley. "Introduction to Computer Graphics." Addison-Wesley. Reading, 1994.

[30] G. Font and I. Boyd. "Numerical study of the effects of reactor geometry on a chlorine plasma helicon etch reactor." *Journal of Vacuum Science and Technology.* **15**(2). 1997.

[31] I. Foster, W. Gropp, and R. Stevens. "The Parallel Scalability of the Spectral Transform Method." *Monthly Weather Review.* 1992.

[32] S. Gimelshein, G. Markelov, M. Rieffel. "Collision Models in the Hawk DSMC Implementation." California Institute of Technology Technical Report CS-96-16. 1996.

[33] S. Gimelshein, G. Markelov, and M. Ivanov. "Real Gas Effects on the Transition Between Regular and Mach Reflections in Steady Flows." *AIAA Paper 98-0877.* 1998.

[34] A. S. Glassner. "Graphics Gems." Academic Press, Inc.. San Diego, 1990.

[35] D. Graves, H. Wu, R. Porteous. "Modeling and Simulation of High-Density Plasmas." *Japanese Journal of Applied Physics.* **32**(6B). 1993.

[36] J. K. Haviland and M. L. Levin. "Application of Monte Carlo method to heat transfer in rarefied gases." *Phys.Fluids.* 1962.

[37] R. Haimes, M. Giles. "Visual3: Interactive Unsteady Unstructured 3D Visualization." *AIAA Paper 91-0794.* Reno, NV.1991

[38] A. Heirich, S. Taylor. "Load Balancing by Diffusion." *Proceedings of 24th International Conference on Parallel Programming.* 1995.

[39] W. N. G. Hitchon, T. J. Sommerer, J. E. Lawler. "A Self-Consistent Kinetic Plasma Model with Rapid Convergence." *IEEE Transactions on Plasma Science.* **19**(2). 1991.

[40] W. N. G. Hitchon, G. J. Parker, J. E. Lawler. "Accurate Models of Collisions in Glow Discharge Simulations." *IEEE Transactions on Plasma Science.* **22** (3). 1994.

[41] W. N. G. Hitchon, E. R. Keiter. "Kinetic Simulation of a Time-Dependent Two-Dimensional Plasma." *Journal of Computational Physics.* **112** (2). 1994.

[42] *ICEM-DDN User's Manual.* ICEM-CFD Engineering, Berkeley, CA.

[43] *ICEM-Tetra User's Manual.* ICEM-CFD Engineering, Berkeley, CA.

[44] M. Ivanov, S. Rogasinsky. "Analysis of numerical techniques of the direct simulation Monte Carlo method in the rarefied gas dynamics." *Soviet Journal on Numerical Analysis and Mathematical Modelling.* **2**(6). 1988.

[45] M. Ivanov, S. Rogasinsky, V. Rudyak. "Direct statistical simulation method and master kinetic equation." XVI International Symposium on Rarefied Gas Dynamics, Pasadena. 1988.

[46] M. S. Ivanov, S. V. Ragasinsky. "Theoretical Analysis of Traditional and Modern Schemes of the DSMC Method." Invited Paper, Rarefied Gas Dynamics, 1991.

[47] M. Ivanov, S. Antonov, S. Gimelshein, A. Kashkovsky. "Rarefied Numerical Aerodynamic Tools for Reentry Problems." Proceedings of the First European Computational Fluid Dynamics Conference, Burssels, Belgium. 1992.

[48] M. Ivanov, S. Antonov, S. Gimelshein, A. Kashkovsky. "Computational Tools for Rarefied Aerodynamics." Proceedings of the XVII International Symposium on Rarefied Gas Dynamics, Vancouver, Canada. 1994.

[49] M. Ivanov, S. Gimelshein, A. Beylich. "Hysteresis effect in stationary reflection of shock waves." *Physics of Fluids.* **7**(4). 1995.

[50] M. Ivanov, G. Markelov, S. Taylor, J. Watts. "Parallel DMSC Strategies for 3D Computations." *Proceedings of Parallel CFD '96.* 1996.

[51] M. Ivanov, G. Markelov, S. Gimelshein, and S. Antonov. "DSMC Studies of High-Altitude aerodynamics of Reentry Capsule." *Proc. 20th International Symposium on Rarefied Gas Dynamics.* 1997.

[52] M. Ivanov, S. Gimelshein. "Computational Hypersonic Rarefied Flows." *Ann. Rev. Fluid Mech..* 1998.

[53] Y. Kallinderis, P. Vijayan. "Adaptive Refinement-Coarsening Scheme for Three-Dimensional Unstructured Meshes." *AIAA Journal.* **31**(8). 1993.

[54] E. R. Keiter, W. N. G. Hitchon, M. J. Goeckner. "A kinetic model of pulsed sheaths." *Physics of Plasmas.* **1**(11). 1994.

[55] D. Knuth. "The Art of Computer Programming." Addison Welsey. Reading, Mass., 1968.

[56] M. N. Kogan. "Rarefied Gas Dynamics." Plenum Press. New York, 1969.

[57] K. Koura and H. Matsumoto. "Variable soft sphere molecular model for inverse-power-law or Lennard-Jones potential." *Physics of Fluids.* **3**(10). 1991.

[58] K. Koura, H. Matsumoto. "Variable soft sphere molecular model for air species." *Physics of Fluids.* **4**(5). 1992.

[59] M. Kushner, W. Collison,, M. Grapperhaus, J. Holland, M. Barnes. "A three-dimensional Model for Inductively-Coupled PLasma-Etching Reactors - Azimuthal Symmetry, Coil Properties, and Comparison to Experiments." *Journal of Applied Physics.* **80**(3). 1996.

[60] H. Legge, K. Nanbu, S. Igarashi. "Force and Heat Transfer on a Disc in Rarefied Flow." *Rarefied Gas Dynamics 17.* Volume 1, Weinheim, NY. 1990.

[61] F. Lumpkin III, B. Haas, I. Boyd. "Resolution of the differences between collision number definitions in particle and continuum simulations." *Phys. Fluids A.*. **3**(9). 1991.

[62] J. Machta. "The Computational Complexity of Pattern Formation." *Journal of Statistical Physics.* **70**(3). 1992.

[63] P. Marriott, T. Bartel. "Comparisons of DSMC Flow Field Predictions using Different Models for Energy Exchange and Chemical Reaction Probability." *Rarefied Gas Dynamics 19.* Volume 1, Oxford University Press. 1995.

[64] M. Matsumoto, T. Nishimura. "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator." To appear in *ACM Transactions on Modelling and Computer Simulation.*

[65] Y. Matsumoto and T. Tokumasu. "Parallel computing of diatomic molecular rarefied gas flows." *Parallel Computing.* (23) 1997.

[66] B. Moon, M. Uysal and J. Saltz. " Index Translation Schemes for Adaptive Computations on Distributed Memory Multicomputers." *University of Maryland Technical Report CS-TR-3428.* 1995.

[67] B. Moon and J. Saltz. "Adaptive Runtime Support for Direct Simulation Monte Carlo Methods on Distributed Memory Architectures." *University of Maryland Technical Report CS-TR-3427.* 1995.

[68] E. P. Muntz. "Rarefied gas dynamics." *Ann. Rev. Fluid Mech.*. 1989.

[69] K. Nanbu, Y. Watanabe. "Relaxation Rates of Inverse-Power and Rigid-Sphere Molecules." *Rep. Ins. High Speed Mach.*. (43) 334.1981

[70] K. Nanbu, S. Igarashi, Y. Watanabe. "Three-Dimensional Hypersonic Flow Around a Disk with Angle of Attack." *Proceedings of the XVI International Symposium on Rarefied Gas Dynamics.* 1989.

[71] K. Nanbu, S. Uchida. "Application of Particle Simulation to Plasma Processing." *Rarefied Gas Dynamics 19.* Volume 1, Oxford University Press. 1995.

[72] K. Nanbu and S. Kondo. "Analysis of Three-Dimensional DC Magnetron Discharge by the Particle-in-Cell/Monte Carlo Method." *Japanese Journal of Applied Physics.* **36**(1). 1997.

[73] R. Nance, R. Wilmoth, B. Moon, H. Hassan, and J. Saltz. "Parallel Monte Carlo Simulation of Three-Dimensional Flow over a Flat Plate." *Journal of Thermophysics and Heat Transfer.* **9**(3). 1995.

[74] T. Nguyen, C. Oh, R. Sinkovits, J. Anderson Jr., and E. Oran. "Simulations of High Knudsen Number Flows in a Channel-Wedge Configuration." *AIAA Journal.* **35**(9). 1997.

[75] G. J. Parker, W. N. G. Hitchon, J. E. Lawler. "Numerical solution of the Boltzmann equation in cylindrical geometry." *Physical Review* , **50**(4): 3210-3219. 1994.

[76] G. J. Parker, W. N. G. Hitchon, D. J. Koch. "Transport of sputtered neutral particles." *Physical Review* pre-print, April 1995.

[77] T. Parsons, J. Harvey. "Object-Process Paradigms in Molecular Computation." *Rarefied Gas Dynamics 19.* Volume 1, Oxford University Press. 1995.

[78] W. Press, S. Teukolsky, W. Vetterling, B. Flannery. "Numerical Recipes in C." Cambridge University Press. Cambridge, 1996.

[79] M. Rieffel. "Concurrent Simulations of Plasma Reactors for VLSI Manufacturing." California Institute of Technology Masters Thesis CS-95-012. 1995.

[80] M. Rieffel, S. Taylor, and J. Watts. "Concurrent Simulation of Plasma Reactors." *Proceedings of High Performance Computing '97.* pp. 163-168. 1997.

[81] M. Rieffel, S. Taylor, J. Watts, S. Shankar, "Concurrent DSMC on Adaptive Tetrahedral Grids. "Invited talk for the 1997 ICEM-CFD Users' Group Meeting, Berkeley, CA, 1997.

[82] M. Rieffel, S. Taylor, and S. Shankar. "Reactor Simulations for Semiconductor Manufacturing." *Proceedings of High Performance Computing '98.* 1998.

[83] M. Rieffel. "The Computational Complexity of the DSMC Method." *Journal of Computational Physics.* (Submitted) 1998.

[84] M. Rieffel, J. Watts, S. Taylor. "Automatic Granularity Control for Load Balancing of Concurrent Particle Simulations." *Proceedings of High Performance Computing '98.* 1998.

[85] S. Taylor, M. Rieffel, J. Watts, and S. Shankar. "Computational Techniques for the Concurrent Simulation of Plasma Reactors." *Parallel Computing for Industrial Applications.* Morgan Kaufmann, 1998.

[86] M. Rieffel, S. Taylor. "The Parallel Scalability of the DSMC Method." *IEEE Transactions on Parallel and Distributed Systems.* (Submitted) 1998.

[87] R. Samanta Roy, G. Hastings, and S. Taylor, "Three-Dimensional Plasma Particle-in-Cell Calculations of Ion Thruster Backflow Contamination," *Journal of Computational Physics*, Vol. 128, pp. 6–18, 1996.

[88] E. M. Shakhov. "Method of studying the rarefied gas motion." Nauka. Moscow, 1974. (In Russian.)

[89] S. Shankar, M. Rieffel, S. Taylor, D. Weaver, A. Wulf. "Low Pressure Neutral Transport Modelling for Plasma Reactors." Invited Paper for 12th International Symposium on Plasma Chemistry, August 21-25, 1995.

[90] S. Shankar, M. Rieffel, S. Taylor, L. Jerde, R. Ditizio. "Three-Dimensional Flow Simulations in Low Pressure Etch Reactors." Tegal Plasma Symposium, Santa Clara, CA. 1996.

[91] S. Sharma, R. Ponnusamy, B. Moon, Y. Hwang, R. Das, J. Saltz. "Run-time and Compile-time Support for Adaptive Irregular Problems." *Supercomputing '94.* 1994.

[92] S. Taylor, J. Watts, M. Rieffel and M. Palmer. "Large-Scale Irregular Calculations using Parallel Architectures. Invited paper to the 6th International Symposium on Computational Fluid Dynamics" 1995.

[93] S. Taylor, J. Watts, M. Rieffel, M. Palmer. "The Concurrent Graph: Basic Technology for Irregular Problems." *IEEE Parallel and Distributed Technology.* **4**(2). 1996.

[94] R. Van Driessche and D. Roose, "An improved spectral bisection algorithm and its application to dynamic load balancing," *Parallel Computing,* vol. 21, pp. 29–48, 1995.

[95] Dean C. Wadsworth. "Slip effects in a confined rarefied gas. I: Temperature slip." *Physics of Fluids.* **5**(7). 1993.

[96] D. Wadsworth. "Development and Application of a Three-dimensional Parallel Direct Simulation Monte Carlo Code for Materials Processing Problems." Proceedings of Parallel CFD '95, Pasadena. 1995.

[97] C. Walshaw and M. Berzins, "Dynamic load-balancing for PDE solvers on adaptive unstructured meshes," *Concurrency: Practice and Experience,* vol. 7, pp. 17–28, 1995.

[98] J. Watts. "A Practical Approach to Dynamic Load Balancing." California Institute of Technology Masters Thesis CS-95-013. 1995.

[99] J. Watts, M. Rieffel, S. Taylor. "Practical Dynamic Load Balancing for Irregular Problems." *Parallel Algorithms for Irregularly Structured Problems: IRREGULAR '96 Proceedings.* Volume 1117, Springer-Verlag LNCS. 1996.

[100] J. Watts, M. Rieffel, S. Taylor. "A Load Balancing Technique for Multiphase Computations." *Proceedings of High Performance Computing '97.* pp. 15-20. 1997.

[101] J. Watts, M. Rieffel and S. Taylor. "Dynamic Management of Heterogeneous Resources." *Proceedings of High Performance Computing '98.* 1998.

[102] M. Willebeek-LeMair and A. Reeves, "Strategies for dynamic load balancing on highly parallel computers," *IEEE Trans. on Parallel and Distributed Systems,* vol. 4, pp. 979–993, 1993.

[103] R. Williams, "Performance of dynamic load balancing algorithms for unstructured mesh calculations." *Concurrency: Practice and Experience,* vol. 3, pp. 457–481, 1991.

[104] R. Wilmoth. "Direct Simulation Monte Carlo Analysis of Rarefied Flows on Parallel Processors." *J. Thermophysics.* 1991.

[105] R. Wilmoth, A. Carlson, and G. Bird. "DSMC Analysis in a Heterogeneous Computing Environment." *AIAA Journal.* 1992.

[106] M. Woronowicz, R. Wilmoth, A. Carlson, and D. Rault. "Procedure for Adapting Direct Simulation Monte Carlo Meshes." *Proceedings of Rarefied Gas Dynamics.* 1992.

[107] M. Yokokawa, K. Watanabe, H. Yamamoto, M. Fujisaki, H. Kaburaki. "Parallel Processing for the Direct Simulation Monte Carlo Method." *Computational Fluid Dynamics Journal.* **1**(3). 1992.

[108] X. Yuan, C. Salisbury, D. Balsara, and R. Melhem. "A load balancing package on distributed memory systems and its application to particle-particle particle-mesh (P3M) methods." *Parallel Computing.* 1997.

[109] X. Zhong, K. Koura. "Comparison of Solutions of the Burnett Equations, Navier-Stokes Equations, and DSMC for Couette Flow." *Rarefied Gas Dynamics 19.* Volume 1, Oxford University Press. 1995.