Particle-In-Cell Simulations Of Highly Collisional Plasmas On The GPU In 1 And 2 Dimensions

A thesis for the degree of

PHILOSOPHIAE DOCTOR

Presented to

DUBLIN CITY UNIVERSITY

By

Nina Hanzlikova B.Sc.
School of Physical Sciences
Dublin City University

Research Supervisor:
Prof. Miles M. Turner

March 2015

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Philosophiae Doctor is entirely my own work, and that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: .....

Nina Hanzlikova

ID No.: 56459501

Date: 27^{th} December 2014

Contents

List of Figures

List of Symbols

n_i	electron density (m^{-3})	4
n_e	electron density (m^{-3})	4
e	absolute electron charge ($\simeq 1.6022 \times 10^{-19}$ C)	5
E	electric field (Vm^{-1})	5
λ_D	electron Debye length (m)	5
T_e	electron temperature (K, Volts or Joule)	4
ε_0	vacuum permittivity ($\simeq 8.8542 \times 10^{-12}$ Fm^{-1})	5
n_0	plasma density (m^{-3})	4
ω_{pi}	ion plasma frequency (rad s^{-1})	6
u_B	Bohm velocity (ms^{-1})	13
k_B	Boltzmann's constant ($1.3807 \times 10^{-23} JK^{-1}$)	9
m_i	ion mass (kg))	6
q	electric charge (C)	10
u	velocity(ms^{-1}); average velocity	13
ω_{pe}	electron plasma frequency (rad s^{-1})	6

LIST OF SYMBOLS

Abstract

During 20th century few branches of science have proved themselves to be more industrially applicable than Plasma science and processing. Across a vast range of discharge types and regimes, and through industries spanning semiconductor manufacture, surface sterilisation, food packaging and medicinal treatment, industry continues to find new usefulness in this physical phenomenon well into 21st century. To better cater to this diverse motley of industries there is a need for more detailed and accurate understanding of plasma chemistry and kinetics, which drive the plasma processes central to manufacturing. Extensive efforts have been made to characterise plasma discharges numerically and mathematically leading to the development a number of different approaches[1].

In our work we concentrate on the Particle-In-Cell (PIC) - Monte Carlo Collision (MCC) approach to plasma modelling [5][9]. This method has for a long time been considered computationally prohibitive by its long run times and high computational resource expense. However, with modern advances in computing, particularly in the form of relatively cheap accelerator devices such as GPUs and co-processors, we have developed a massively parallel simulation in 1 and 2 dimensions to take advantage of this large increase in computing power. Furthermore, we have implemented some changes to the traditional PIC-MCC implementation to provide a more generalised simulation, with greater scalability and smooth transition between low and high (atmospheric) pressure discharge regimes. We also present some preliminary physical and computational benchmarks for our PIC-MCC implementation providing a strong case for validation of our results.

Acknowledgements

Over the years of creation of this work many people deserve my thanks for their expertise and support. Chief amongst them is my supervisor, Prof. Miles Turner, whose guidance down the years has been instrumental in my completing this work. Your recognition of the importance for us to understand computer science as well as physics helps push this field forward and certainly helped in pushing me along with it.

A great thanks is owed to my colleagues from the plasma physics research group, Huw Leggate, Seán Kelly, Samir Kechkar, Sarveshwar Sharma, Gurusharan Singh, Mubarak Mujawar, Zhenning Su, Nishant Sirse, Bert Ellingboe, Eamonn Monaghan, Cezar Gaman and David O'Farrell. Not only was their technical expertise invaluable, their friendship and company truly enriched my time as a PhD student.

Many friends have provided welcomed distractions down the years and I'd like to thank them all, in particular Damien, Mary, Cathal O B., Carri, d_fens, Conor, Meabh, Oisín, Cathal M., Hugh, William & Alison, the Sheerin siblings, Marissa, Rachel, Jen, Ciarán, Adam, Mike, Sheila & Tiernan, Emma & Hugh, Alex & Jen, Dave, Helen & Orla, Declan, Selina & Alice, bunbun, fun and anyone else I've inadvertently forgotten. The Taylor family deserves thanks as well, for countless family dinners they've let me gate crash.

Finally I'd like to thank Liam and my own family for putting up with me so far. They have all been more supportive of me than I ever could hope, despite in the latter's case still not being sure what it is I do for the most part. Thanks to mum and Liam for reading over my work with fresh eyes and thanks to them and Jan for making life exciting every step of the way.

This research was funded by a grant from the Irish Research Council (IRC) and Science Foundation Ireland (SFI).

CHAPTER 1

Introduction

As discussed in the abstract, plasmas are very industrially useful but physically complex systems. The range of different types of plasmas is extensive, each with its unique set of dominant phenomena, as well as a set of boundary conditions defining the plasma, particular to every plasma chamber. It is not unreasonable to ask where does one begin when faced with a problem of such complexity.

Since plasmas are a large collection of kinetically interacting charged particles, to characterise them both electromagnetic and collisional approaches have to be considered. These allow us to calculate a number of characteristic parameters and thus give us an overview of the behaviour of the plasma. From electromagnetic considerations we can establish a characteristic length for electromagnetic interactions in the plasma, the response frequency of electrons to applied force (i.e. electric field) and macroscopic charge charac-

teristics of the discharge. From kinetic interactions we obtain the minimal collision parameter and diffusion characteristics for the plasma[1]. These provide us with a considerable tool set for understanding plasma interactions and modelling of discharges[5, 9].

1.1 Overview

In Chapter 1 of this thesis we discuss the fundamental concepts associated with our work. In Section 1.2 we outlined some of the fundamental parameters associated with plasmas as well as the base processes taking place and the expected plasma characteristics. In Section 1.3 we extend this description to the special case of atmospheric pressure industrial plasmas, listing some common atmospheric sources as well as how these differ from the low pressure cases. An overview of plasma modelling is provided in Section 1.4.

Chapter 2 provides a brief overview of high performance computing (HPC), with particular focus on the GPU Nvidia CUDA environment. The GPU hardware architecture is described in Section 2.1.1, followed by a short summary of particle-in-cell simulation developments on the GPU in Section 2.1.2. Finally in Section 2.2 we discuss some HPC alternatives to GPU and CUDA technologies.

In Chapter 3 we outline the conventional PIC-MCC modelling procedure in greater detail, beginning with high level overview of the model structure in Section 3.1. The equations to be solved are discretised in Section 3.2, with numerical smoothing to improve the simulation quality being discussed in Section 3.3. Normalisation imposed on the system is defined in Section 3.4. The chapter concludes with a description of the collision simulation technique in Section 3.5.

1.1 Overview

A detailed description of our model design in 1 and 2 dimensions is presented in Chapter 4. A high level overview of the model procedure is shown in Section 4.1, with each of the features listed being described in more detail in the following sections. The data structure is detailed in Section 4.2, with the algorithmic and numerical procedures described in Sections 4.3-4.7.

Special attention is given to the description of the particle pusher/collider. This is described in detail in Chapter 5. Here we outline some non-obvious issues with the naive direct implementation of collisions and propose an alternative implementation (Section 5.1). In Section 5.2 we then provide some simulation results to support our arguments regarding the severity of the issue and its resolution with our modifications.

In Chapter 6 we provide verifications of our 1 and 2 dimensional models. We outline the benchmark parameters in Section 6.1 and offer a brief discussion of the deviations of the benchmark models from our own implementation at the start of the 1D model verifications in Section 6.2. We offer a preliminary verification benchmark for the 2D model and discuss its limitations in Section 6.3.

The computational performances of our models are presented in Chapter 7. Here we present the parallel scaling benchmarks for the 1 dimensional and 2 dimensional models as well as for the 2 dimensional field solver. In addition we also provide scaling benchmarks of the PIC-MCC models with gas pressure. The benchmarks for the 1D and 2D PIC-MCC models and the 2D field solver are presented in Sections 7.1, 7.2 and 7.3 respectively.

The work concludes in Chapter 8 with a summary of our results in Section 8.1 and an outline of future works to be carried out with our models in Section 8.2.

1.2 Plasma Fundamentals

As mentioned in the introduction to this chapter, plasmas consist of a collection of charged particles moving under kinetic interactions and electromagnetic forces exerted on them[1]. The overall total charge of this system is approximately zero so on large scales we can say that deviation from neutrality has to be small in comparison to the electron density and the plasma is said to be quasi-neutral. On local scales, however, we can see accumulation of charges and thus a potential can be observed.

At the same time plasma particles can freely traverse the plasma and interact kinetically with each other. As a result of these particle collisions many chemical processes take place in the plasma, possibly foremost of these being ionization. In addition, excitation and elastic collisions affect the individual particle energies and thus modify the rates of chemical reactions taking place.

Finally plasma boundary also plays an important role to characterising plasma behaviour. It is usually at the boundary that materials processing takes place and therefore diffusion from the plasma bulk to the boundary sheath region as well as particle confinement due to induced electric fields in the sheath are of great interest to our understanding of plasma processes.

1.2.1 Debye Length

It is reasonable to attempt to characterise these scales in a more vigorous fashion. Assuming the simple case of singly charged ions and electrons, where ion density $n_i = n_0$ and electrons follow the Boltzmann relation $n_e = n_0 \exp(\Phi/\mathrm{T}_e)$, where T_e is given in eV, we can solve the Poisson equation to obtain an expression for Φ[1]. The Poisson equation for singly charged

positive ion and electron system in one dimension is given as

$$\frac{d^2\Phi}{dx^2} = \frac{e}{\epsilon_0}(n_e - n_i) \tag{1.1}$$

Solving for Φ we see a sharp drop for values larger than a critical length λ_D, which we call Debye length and can calculate from

$$\lambda_D = \left(\frac{\epsilon_0 \mathrm{T}_e}{e n_0}\right)^{1/2} \tag{1.2}$$

Therefore Debye length gives us a measure of the effective screening of charged plasma particle from the electric field they exert on one another. In typical low pressure discharge conditions where $\mathrm{T}_e = 4$ eV and $n_e = 10^{10}$ cm^{-3} Debye length is equal to 0.14 mm.

1.2.2 Plasma Frequency

Since plasmas consist of charged particles, it is apparent that applying an electric field to a plasma causes a displacement of the charges. Electrons are much more mobile than ions due to their much smaller mass so at a first approximation the case can be simplified by considering only their movements. When an electric field is applied, the charges will move so as to create their own field to oppose it, and thus restore equilibrium. However, due to inertia they will overshoot this position and experience a force in the opposite direction. This results in oscillation motion of the electrons around the equilibrium point, with a characteristic frequency called plasma frequency.

To calculate this, if we consider a small displacement s of charged particles from their equilibrium position, the induced electric field is determined from Gauss's law to be

$$E = \frac{e n_0 s}{\epsilon_0} \tag{1.3}$$

This gives the force equation as

$$m_e \frac{d^2 s}{dt^2} = -eE \tag{1.4}$$

since the force is always directed opposite to the displacement of the charge to restore the particle to its equilibrium position. Combining these two equations we see this gives the equation for a simple harmonic oscillator with the frequency ω_{pe} given by

$$\omega_{pe} = \left(\frac{e^2 n_0}{\epsilon_0 m_e}\right)^{1/2} \tag{1.5}$$

In this case we considered electrons to be our oscillating species but the same argument holds for a centre of mass system with oscillating ions, with particle specific parameters such as ion charge q_i and mass m_i replacing the electron values in Equation 1.5 to give the ion plasma frequency ω_{pi}. The plasma frequency in the case where both ion and electron plasma frequencies are being considered is then given by

$$\omega_p^2 = \omega_{pe}^2 + \omega_{pi}^2 \tag{1.6}$$

However since the characteristic frequency of the species is inversely proportional to the mass of the species and the mass of the ions is several orders of magnitude larger than that of electrons, the plasma electron frequency dominates and therefore plasma frequency can be approximated to a high degree of accuracy as electron plasma frequency value. Discharge plasma frequencies are usually in the microwave range 1-10 GHz[1].

An interesting effect of this oscillation response frequency is the limiting behaviour it sets on the interaction of the plasma with an electromagnetic wave. Since the electrons can only respond to an electric field on timescales equal to or in excess of the plasma oscillation period, if we subject the plasma to an EM wave oscillating at a higher frequency, the charges will not have

reacted fully to the field before the field has altered. Thus the EM wave will be able to pass through the plasma without any attenuation or disruption and the plasma will be effectively transparent. Conversely, at lower frequencies the electrons will be able to respond to the field and thus attenuation of the wave will be observed after passing through the plasma.

1.2.3 Collisions

Next let us consider collisions in our particle system. Since we are dealing with a collection of particles there is a large number of different types of collisions taking place. Of major importance in particular are ionization reactions with neutral feed gas, which drive the plasma particle creation mechanism and sustain the discharge through primary and secondary collisions. These processes also clearly determine the plasma density, which in turn crucially effects material processing features of a plasma, such as surface etch rates. Ionization rates within the plasma are in turn highly dependent on the plasma particle energies, a parameter not only affected by the driving potential applied to the plasma, but also through excitation and elastic collisions between particles. In addition, collisions can also cause recombination or quenching of excited states and thus affect the particle kinetic properties further. For a further discussion of the different collision mechanisms the reader is referred to Lieberman[1].

However we can relatively easily consider the simple case of Coulomb scattering. It is useful to define the collision parameter b as the radial (perpendicular) distance between the initial trajectory of an incident particle and the centre of its collision partner, which results in a 90° deviation in direction from the original path, as shown in Figure 1.1. Assuming a stationary colliding partner, which can be generalised by taking the centre of mass system,

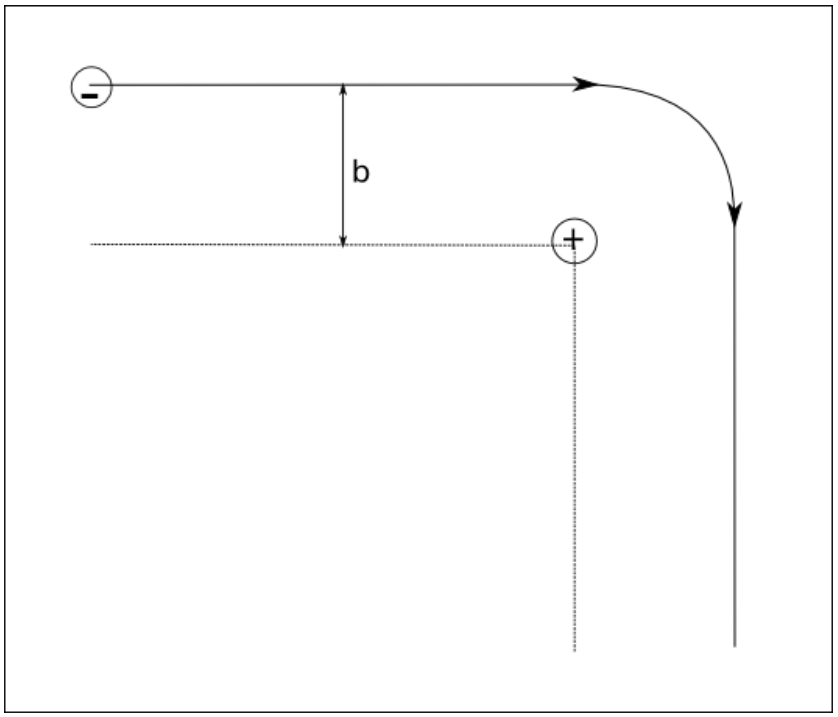


Figure 1.1: *Collision parameter schematic for a 90° deflection from original trajectory*

for the incident particle to just avoid capture its kinetic energy has to be equal to the potential between the charges

$$k_b T_e = \frac{e^2}{4\pi\epsilon_0 b} \quad => \quad b = \frac{e^2}{4\pi\epsilon_0 k_b T_e}$$

where T_e is electron temperature in Kelvin.

In reality the deflection happens as a result of multiple small angle scattering collisions and a solution for the scattering parameter can be obtained from integrating over the small angles as carried out by Lieberman[1].

The scattering parameter is useful in determining the cross section σ_{sc} for scattering collisions, which in turn allows us to calculate the rate constant K. For cumulative collision angles the cross section is given by

$$\sigma_{90} = \frac{8}{\pi} b^2 \ln(\Lambda),$$

and the rate constant for collisions can be found from

$$K = \sigma v, \tag{1.7}$$

allowing for easy calculation of the number of the particular collisions based on the density of colliding species. Typically, while Λ is a large number, ln $\Lambda \approx 10$[1].

1.2.4 Diffusion

In a multi-species system the movement of particles will result in mixing of the different species and thus diffusion. Diffusion can be defined in terms of particle fluxes, where particles being accelerated to higher speeds mix faster and collisions act to decelerate them and reduce the particle flux. Combining the acceleration due to induced electric field in the discharge with Fick's law for diffusion we get the particle flux expression

$$\Gamma = \pm\mu n E - D\nabla n \tag{1.8}$$

where

$$\mu = \frac{|q|}{m\nu_c} \qquad\qquad D = \frac{k_b T}{m\nu_c}$$

are the mobility for momentum transfer frequency ν_c and the diffusion coefficient respectively. The actual range of ν_c values encountered in industrial plasmas is large. In the low pressure limit the collision frequency is much lower than the electron plasma frequency described in Section 1.2.2, while at atmospheric pressures it comes to dominate it. Therefore particle mobility can vary significantly with the plasma source. In addition the relation above does not assume a particular particle species and thus has to hold for both ions and electrons.

However for quasi-neutrality to be satisfied one species cannot have a larger flux than the other specie. A greater flux in one specie would result in depletion of the species in the region and thus to a buildup of the opposite charge, inducing an electric field to oppose this flux motion. It is therefore necessary to equate the ion and electron fluxes as given by Equation 1.8. Solving for E and substituting into the ion or electron flux equation results in

$$\Gamma = -\frac{\mu_i D_e + \mu_e D_i}{\mu_i + \mu e} \nabla n$$

This is just Fick's law with

$$D_a = \frac{\mu_i D_e + \mu_e D_i}{\mu_i + \mu e} \tag{1.9}$$

being the ambipolar diffusion coefficient. Ambipolar diffusion, combined with a suitable boundary condition allows for the calculation of the density as a function of time and position. It may be tempting to set the boundary as a perfectly absorbing wall, with species density tending to zero at this boundary, but this would lead to a particle flux of zero at the wall for any

finite velocity value, which is inconsistent with any physical solution. It is therefore necessary to take a closer look at the conditions at the wall.

1.2.5 Plasma Sheath

The simplest case to consider is that of a uniform species distribution confined in a chamber with absorbing walls. Initially particles will begin diffusing outwards from the bulk. The electrons, being lighter and thus having higher mobility than the ions will be lost faster and thus there will be a build up of positive charge at the walls. This will result in a potential gradient forming at the boundary, inducing an electric field directed out of the discharge. Electrons will therefore be accelerated in the opposite direction, reducing the electron loss at the wall and effectively confining them in the bulk of the plasma. The positive charge region at the edge of a plasma is referred to as the plasma sheath.

At the same time the buildup of positive charge in the sheath region means any positive charges in the bulk of the discharge will also be repelled from entering it. However a flux into the sheath is required by the ion continuity equation. Therefore only ions of a particular minimum velocity will be able to enter the sheath. Due to this there has to exist a region at the boundary between the bulk plasma and the sheath where the plasma is still essentially quasi-neutral but a small potential gradient exists to accelerate the ions to the required velocity. This acceleration takes place in the so called presheath.

To treat this mathematically, let us consider the potential and density profile shown in Figure 1.2. Taking the potential as zero at the sheath-presheath boundary and using ion continuity equation at the sheath edge combined with energy conservation, the expression for the ion density is

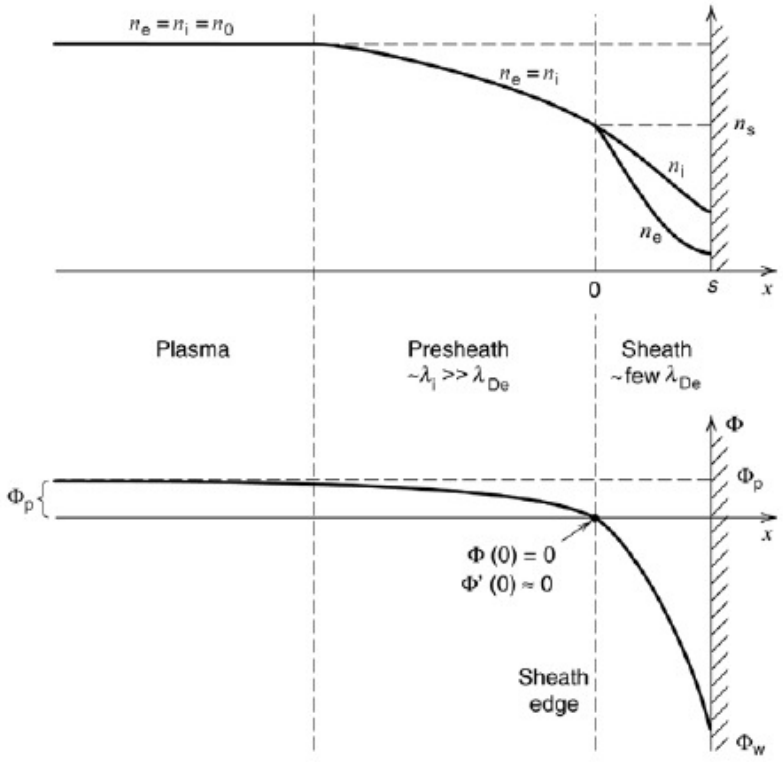


Figure 1.2: *Sheath and presheath formation at a wall.[1]*

given as

$$n_i(x) = n_{is}\Big(1 - \frac{2e\Phi(x)}{Mu_s^2}\Big)^{-1/2} \tag{1.10}$$

where n_{is} and u_s are the density and the ion velocity at the sheath edge respectively. Using the Boltzmann relation for electrons, the Poisson equation becomes

$$\frac{d^2\Phi}{dx^2} = \frac{en_s}{\epsilon_0}\left[\exp\frac{\Phi}{T_e} - \Big(1 - \frac{\Phi}{\mathcal{E}_s}\Big)^{-1/2}\right] \tag{1.11}$$

where $\mathcal{E}_s = \frac{1}{2}Mu_s^2/e$ and by multiplying both sides by $d\Phi/dx$ and integrating over dx we can get the expression for the first derivative of Φ as

$$\frac{1}{2}\Big(\frac{d\Phi}{dx}\Big)^2 = \frac{en_s}{\epsilon_0}\left[T_e\exp\frac{\Phi}{T_e} - T_e + 2\mathcal{E}_s\Big(1 - \frac{\Phi}{T_e}\Big)^{1/2} - 2\mathcal{E}_s\right] \tag{1.12}$$

Clearly to obtain any physically meaningful solution the right hand side of the equation has to be a positive number, so using Taylor expansion in the first two terms yields the inequality

$$\frac{1}{2}\frac{\Phi^2}{T_e} - \frac{1}{4}\frac{\Phi^2}{\mathcal{E}_s} \geq 0$$

or

$$u_s \geq u_b = \Big(\frac{eT_e}{M}\Big)^{1/2} \tag{1.13}$$

where u_b is called the Bohm velocity. This requirement is referred to as the Bohm criterion and sets the sheath ion velocity requirement of $u_{is} \geq u_b$[10]. Conversely, it is required that the bulk ions have speeds smaller than the Bohm velocity to avoid sheath formation. The sheath edge where the bulk joins the sheath is defined to have the ions with a velocity distribution around the Bohm velocity. At $T_e = 4$ eV in Helium plasma this is approximately 9000 m/s, while sheath width in nominally in the region of 1 cm (for Child Law sheath)[1].

1.2.6 Driving Potential and Heating

To achieve sustained plasma conditions, driving potential is applied to the discharge. In materials processing a particularly popular type of discharge is the so-called rf-diode, a capacitively coupled radio frequency driven discharge. The plasma is confined between two electrodes and driven by the applied radio frequency signal. The driving voltage is typically between 100-1000 V with an electrode separation of 2-10 cm[1].

Since the driving frequency is much higher than the ion plasma frequency, most applied potential energy is transferred to the electrons. The relatively heavy ions continuously bombard the electrodes over the rf cycle while electrons oscillate with the field and thus are lost only when the electron cloud approaches the electrode.

In addition to heating due to the applied electric field, electrons in the sheath are also heated through stochastic heating. Stochastic heating takes place due to accelerated electrons in the sheath colliding with the oscillating sheath and thus is sometimes referred to as the collisionless heating. This effect is most significant at low pressures.

On the other hand at high pressures, the bulk plasma Ohmic heating comes to dominate over stochastic heating. Ohmic heating takes place due to electron collisions with neutrals in the bulk. At low pressures these collisions are moderately rare, in comparison to the much more prevalent stochastic collisions. However as the pressure increases these collisions become much more common and come to dominate the heating processes.

1.3 Atmospheric Pressure Plasmas

Atmospheric pressure plasmas usually come in two broad varieties, thermal and cold plasmas. Thermal plasmas are characterised by the constituent particles being in local thermodynamic equilibrium (LTE). Conversely in cold plasmas the electron temperature is expected to be much larger than the ion temperature and the plasma is said to be a non-local thermodynamic equilibrium plasma (non-LTE). Particularly in spectroscopic studies of plasmas the thermal equilibrium of particles becomes important[11]. In practice for considerations such as limiting energy losses through heating as well as not damaging treated surfaces, and in medical treatment applications it is desirable for the plasma to be non-LTE and moderately uniform over the treated surface area.

1.3.1 Atmospheric Plasma Sources

The variety of atmospheric plasma sources available is extensive and therefore providing an exhaustive list of these would extend outside the scope of this work. However a few examples of sources of different types of plasma are outlined below.

Arc Plasma Torch

Arc plasma torches come in two varieties, current-carrying arc and transferred arc[12]. In the case of the former the nozzle is positively biased and becomes the anode of the system, while in the latter case the nozzle is left as the floating potential and the treated material is biased to be the anode. Sources of this type include Plazjet[13], Plasmapen and Plasmapen Xtension[14] and Plasma-Jet[15].

Corona Discharge

Corona discharge[16] is an example of a pulsed working mode source. The discharge is driven by a DC power supply which is pulsed, rather than steady. The duration of this pulsing is shorter than the time needed for arc creation. The treated surface is biased as the anode, with the voltage driven between it and the wire cathode.

Dielectric Barrier Discharge (DBD)

Dielectric-Barrier discharges were first reported by Siemens in 1957[17], who concentrated on their use in ozone generation. Two distinct modes of operation are achievable with DBDs, the filamentary discharge[18] and the more uniform glow discharge[19]. The latter usually requires more specific running conditions such as feed gas composition and pressure pulsing[20], as well as electrode structure and power frequency[21].

Atmospheric Pressure Plasma Jet (APPJ)

This is a small, low power, radio frequency driven plasma torch[16]. Two common electrode configurations are with two concentric electrodes with working gas flow through the system[12] and two planar, perforated aluminum electrodes that allow gas flow through electrodes themselves [22].

1.3.2 Atmospheric Plasma Characteristics

The special case of atmospheric pressure plasmas is an attractive discharge regime for industrial applications, largely due to the potential ease of operation and low deployment costs. The vacuum conditions and airtight plasma chambers required in low pressure processing are expensive and cumbersome

and particularly in fields such as medicine impractical for widespread use. In recent years this has motivated a considerable interest in discharges characterised by higher pressures. However, as appealing as the premise of plasma plumes operational in open air is, there are many practical problems associated with the basic operation at atmospheric pressures.

To illustrate this it is necessary to consider the ionization mechanism in plasmas. As discussed in Section 1.2, the driving mechanism behind ionization in a plasma is collisions between charged particles and neutrals. Assuming we begin with a small number of electrons near the cathode, these are accelerated towards the anode and on their way they collide with neutral background gas causing more ionization and production of electrons. Therefore the electron density can be expressed as $n_e(x) = n_{e0}\exp(\alpha x)$, where α is the Townsend Ionization Coefficient. Meanwhile ions generated through ionization near the anode are accelerated towards the cathode, causing further ionization through collisions and resulting in additional $n_{e0}\gamma[\exp(\alpha d) - 1]$ electrons, where γ is the second Townsend coefficient. This then in turn leads to further avalanche effect, as discussed in literature[1, 2]. The first Townsend coefficient α can be semi-empirically determined from

$$\frac{\alpha}{p} = A\exp\Big(-\frac{B}{E/p}\Big) \tag{1.14}$$

where A and B are parameters dependent on the gas in the discharge and p is the pressure. At some point the creation of electrons through collisions starts to equal the loss rate at the walls and a self-sustained discharge is achieved.

This mechanism clearly heavily relies on diffusion of charged particles across the gas. However diffusion timescale is inversely proportional to the mean free path λ of the particles in the discharge, which in turn is itself inversely proportional to the pressure. This leads to slow diffusion of charges

and their localisation within the discharge. The increase in collisions with pressure also leads to more losses through dissociative recombination and it is in fact this timescale that dominates the loss processes at higher pressures.

In addition, collisions between electrons and neutrals lead to a larger energy transfer between the species at higher pressures. At low pressures collisions are relatively rare and most heating from the electric field is therefore confined to electrons, which are light enough to respond to the applied field as well as transfer very little of this obtained energy during the relatively rare collisions with the much heavier ions. However as the gas pressure increases towards atmospheric values, the increase in collisions results in sufficient energy transfer between the species for the low pressure assumption of temperature imbalance between ions (T_i) and electrons (T_e) to no longer hold. Therefore it can be seen that by increasing the ion density electron temperature reduction is achieved, resulting in reduced ionization and reaction rates and energy losses.

Returning to the study of the ionization at high pressure it can be seen that the localised region of high ionization can result in a localised temperature increase due to recombination and gas heating. Assuming constant pressure to conserve particle continuity this will result in the gas density reducing. As discussed above the electron temperature will increase as a result of reduction in energy losses and this will further increase the ionization rate in the region[23]. This situation dominates particularly in core of plasma columns and thus results in a tendency towards plasma contraction and filamentation.

Not all charge build-ups necessarily reach these instability conditions. Starting with a localised avalanche, the charged species will start heading towards the oppositely charged electrodes creating an electric field directed

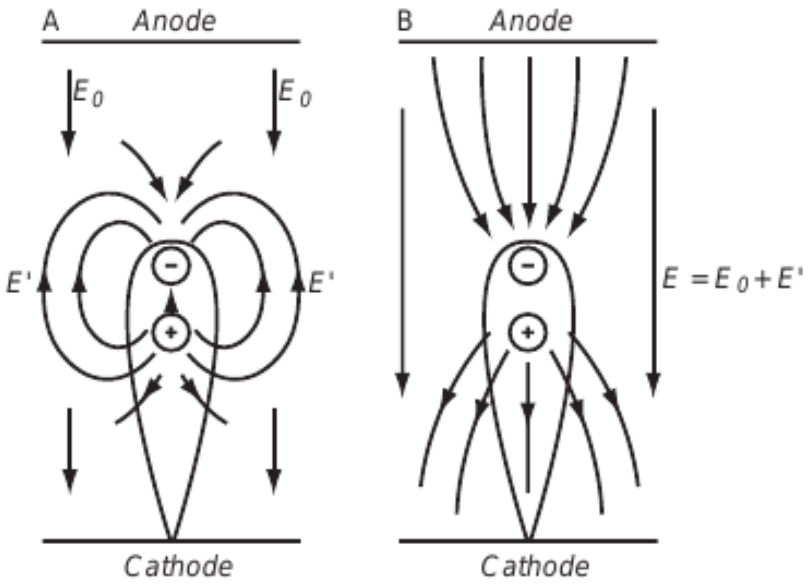


Figure 1.3: *(a) Externally applied electric field and electric field due to charge separation. (b) Superposition of the external and avalanche fields.[2]*

as shown in Figure 1.3. The field at the ends of the avalanche is directed in the same direction as the externally applied field while within the avalanche there is in fact a drop in the combined electric field. As a result an increased ionization rate can be observed at the edges while there is an ionization drop in the avalanche itself. From the requirement of quasi-neutrality, we can see a plasma streamer forming from the avalanche when the resultant electric field in the avalanche is zero, which corresponds to the induced electric field due to charge separation being equal to the applied electric field. Mathematically the condition for streamer formation is given by the Meek criterion

$$E_a = \frac{e}{4\pi\epsilon_0 r_a^2}\exp\left[\alpha\left(\frac{E_0}{p}\right) * d\right] \approx E_0 \tag{1.15}$$

where r_a is the avalanche head radius and d is the streamer gap.

For industrial applications plasma instabilities are generally to be avoided. The build up of localised ionized channel manifests as sparks or arcing, which leads to significant damage of the surface being processed. There are various ways of limiting the instability formations at high energy, generally focusing on keeping the imbalance between ion and electron temperatures large enough to allow for diffusion timescale to dominate the charged species loss processes. Imposing a limited time duration results in the avalanche not having enough time to transit into a streamer. Applying a high electric field on the other hand postpones the streamer formation since a larger electric field has to be induced in the avalanche for the Meek criterion to be satisfied. Artificially limiting current and heat removal will limit the ionization inside an avalanche region to avoid a buildup and streamer formation.

In practise a number of different techniques have been seen to be effective in producing stable atmospheric pressure discharges. This includes using distributed resistive electrodes to limit the current, as developed by Laroussi and Alexeff[24]. Similarly charge buildup can be limited by intro-

ducing a dielectric barrier at the electrodes, resulting in a Dielectric Barrier Discharge[25]. In more specialised cases high frequency excitation has also been seen to limit the formation of instabilities, as shown by Liu *et al*[26]. Additionally heat removal and diffusion can be encouraged by reducing the physical dimensions of the discharge. This is the mechanism for instability reduction in atmospheric pressure micro-discharges[27]. A more comprehensive discussion of all these discharges is given by Tendero [12]

Presuming one overcomes the problem of instabilities the resultant plasma system is still a lot more complex to characterise than the low pressure counterpart. The increase in collisions results in more complex chemistries where three body reactions become of importance in the system. Ionic processes also gain in importance over metastable ones since increase in collisions leads to both higher ionization rates and greater quenching of species. As mentioned, these conditions will favour dissociative recombination losses over diffusion losses leading to decrease in importance of ambipolar diffusion and heterogeneous reactions across large volumes.

1.4 Plasma Modelling

A number of different plasma modeling approaches have been developed in the field, each with its own advantages and disadvantages. Three major approaches to simulating plasmas can be summed up as fluid models, particle-in-cell models and hybrid models, the last of which combines the characteristics of the former two approaches[5]. In our work we concentrated on the particle-in-cell approach, due to our interest in highly collisional plasma regimes and their kinetic characterisation.

1.4.1 Particle Modelling

Particle-in-cell (PIC) codes have been a popular method of modelling plasmas since the 1960s[28–31]. Despite their computational cost usually making all but simple plasma chamber geometries and chemistries prohibitive, they none-the-less provide a direct simulation vehicle for the study of plasma kinetics.

As discussed by Lieberman[1], a central issue in characterising the kinetic interactions of plasmas is the non-Maxwellian nature of the electron distribution in non-LTE discharges. As a result electron energy distribution functions have to be calculated directly from the Boltzmann equation

$$\frac{\partial f_e}{\partial t} + \mathbf{v} \cdot \nabla f_e + \frac{\mathbf{F}}{m} \cdot \nabla_v f_e = \left.\frac{\partial f_e}{\partial t}\right|_c \tag{1.16}$$

The solution to this equation is notoriously complex, resulting in a set of coupled, non-linear, integro-differential equations in seven dimensions (x, y, z, v_x, v_y, v_z, t). Conveniently, the PIC method provides a direct numerical solution for this problem without the need for assumptions relating to the electron distribution itself. This is achieved through tracking samples of the particle phase space during the simulation and resolving the physical and chemical phenomena through direct collisions. Therefore the electron energy distribution function and electron probability distribution function can be calculated directly from the simulation.

The resultant discrete superparticles then allow for direct determination of trajectories of the phase space slices. At the same time the spatial charge accumulation is abstracted into a charge grid, solving the potential for the system in a much more efficient fashion than direct particle-particle interactions.

Collisions began being introduced into PIC codes a decade after the pio-

neering efforts listed above[32–35], mostly through the adoption of the Monte Carlo procedure. These simulations relied heavily on low collision probabilities, consistent with low pressure plasmas, requiring the assumption of low collision frequency in comparison to the plasma frequency. The resolution of these collisions took place at the end of each timestep, although timestep centering of collisions is currently also sometimes employed. A modification can be introduced for higher pressure cases, where a modified collision probability is calculated, but this presents a collision limit of one collision per particle per timestep[36]. In the case of high collision rates, an alternative is provided in texts such as Hockney's[9, 37], which still remains the definitive literature on the subject. In this approach, a particle subdivided into collision times leading up to the timestep boundary may undergo several collisions between each solution of the field equations. The global timestep Δt is then the interval between solutions of the field equation, while particles advances across these intervals in a sequence of intermediate steps punctuated by collisions.

1.4.2 Atmospheric Plasma Modelling

As discussed in Section 1.3.2, non-LTE atmospheric pressure discharges consist of streamers forming between the electrodes and dielectrics. It is therefore immediately apparent that to study this class of discharges a minimum of 2 dimensions is needed in our simulations. These models tend to be very complex and computationally intensive[9] and therefore various simplifying assumptions have been adopted in most studies.

At atmospheric pressure the discharge consists of a number of (overlapping) microdischarges. Therefore it becomes necessary to simulate injection of species. A mixture of global or fluid model approaches to this can be found in the literature[38, 39]. These models solve for the species continuity

equation assuming predefined rates for kinetic reactions. While much faster than kinetic models, the defined rates of reaction require defined energy distributions for the species. Most commonly we assume a Maxwellian electron distribution however this also imposes an inherent assumption of thermal equilibrium as well as equilibrium of electrons with the applied field. This potentially limits the time resolution usefulness of these models.

An alternative approach is that of hybrid models, where one species is approximated by the fluid approach while the other is treated kinetically through Particle-In-Cell/Monte Carlo methods. A discussion of the hybrid model approach is given in [40, 41]. The adaptation of this technique to the atmospheric pressure discharge can be found in the work of Kong[42, 43] where both cathode fall region and sheath regions in DBDs were examined with kinetic treatment of the electrons.

Probably the most accurate approach to the plasma discharge modelling problem is the Particle-In-Cell model, where all species are treated kinetically and thus no kinetics dependent coefficients need to be supplied from assumed particle distributions. These methods were particularly developed by Hockney[9] and Birdsall[5], which still remain the seminal texts on this subject. Unfortunately the kinetic treatment is computationally expensive both in terms of memory storage and arithmetic operations due to the species being treated in terms of large numbers of superparticles, with defined positions and velocities. This leads to problems in representing particularly complex plasma systems both in terms of the particle kinetics in the plasma as well as any more exotic plasma chamber geometries. Therefore kinetic models have not seen much popularity in atmospheric simulation as well as more realistic plasma conditions.

However with the advent of improved computational resources available

at reasonable prices this attitude has been changing, with efforts being made to examine some of the simplifying assumptions of the popular fluid models using the kinetic models in at least simplified scenarios. Of particular interest is the work of Avtaeva and Skornyakov[44], who examined the local field approximation for electrons in DBD and the comparisons of fluid and kinetic 1D models carried out by Lee *et al.*[45]. Both studies seemed to show moderate qualitative agreement between the different approaches but notable quantitative differences between the obtained results.

Most recently more analytical groundwork has been presented to explore dealing with more irregular geometries by Fichtl *et al.*[46]. This work approaches the problem by transforming the irregular physical grid into a rectangular logical grid, which reduces the problem into a familiar one with addition of certain coordinate transformations. However this formulation leads to the pusher position integration of particles becoming an implicit one and thus results in a much more complicated solver for the particle velocity and position advancement. The field solver also gains in complexity to account for the physical geometry. However Fichtl has presented results of his implementation showing agreement for some common periodic boundary condition problems.

1.5 Chapter Summary

In this chapter we outlined some important fundamental parameters in a plasma. We described the importance of Debye length and plasma frequency, as well as described the mechanics behind plasma collisions, diffusion and sheath formation. In the case of high pressure plasmas, streamer formation was briefly outlined. Finally a brief overview of plasma simulation outcomes

in literature was also provided.

CHAPTER 2

Introduction To High Performance Computing

In recent years industrial demands have resulted in increasingly more sophisticated plasma sources. This demand has in turn driven the need for more computationally complex models to characterise these sources theoretically. The resultant increase in computational intensity of the models presents a considerable challenge for many commercially available computational systems. Within the plasma simulation community this has highlighted the need for highly scalable algorithms and High Performance Computing (HPC) hardware infrastructures to support them.

In our work we have concentrated on the Graphical Processing Unit (GPU) architecture using Nvidia CUDA parallel computing platform. The GPU provides an inexpensive, highly scalable accelerator alternative to sophisticated CPU systems. In turn the Nvidia CUDA platform provides a convenient environment for C/C++ developers to transition into GPU devel-

opment, with a number of tools and libraries made available and maintained by Nvidia. Some alternatives to these technologies are discussed in Section 2.2.

2.1 Graphical Processing Unit (GPU)

The need for dedicated computer display hardware became apparent as early as 1983, with Intel's release of the iSBX 275 Video Graphics Controller Multimodule Board[47]. Since the display bitmap consists of a large array of points that need to be periodically updated with respect to the display colour and quality it becomes apparent that significant amounts of computation are involved in any visual output from the useful computation being carried out on the host machine. The Intel iSBX 275 Controller was the first dedicated piece of hardware to handle just this overhead, paving the way for more sophisticated display capabilities as well as freeing up the CPU for more useful computation. Since then a number of dedicated graphics hardware companies have been created with probably the most notable being the Nvidia company, which introduced the first "Graphical Processing Unit" with its GeForce 256 card[48], and AMD acquired ATI Technologies introduction of the Radeon R300 card.

2.1.1 Hardware Overview

The main computational premise in graphics programming revolves around having a large set of data where each element is updated periodically using a similar arithmetic procedure. This system is largely independent at update time on updates taking place in other elements simultaneously. To this effect the GPU architecture employs hugely parallel programming using the

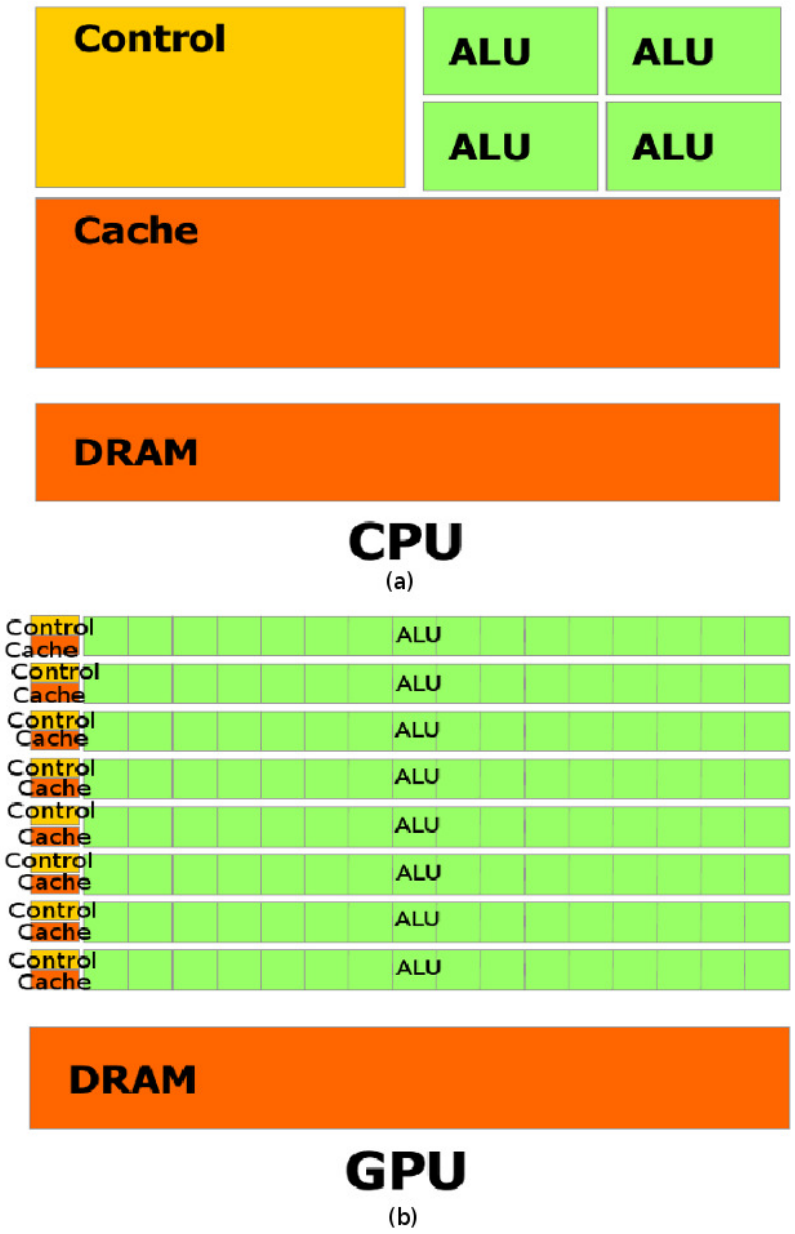


Figure 2.1: *Processing unit architectures. (a) A single core central processing units (CPU) hardware structure; (b) A graphical processing unit (GPU) hardware structure.[3]*

Single Instruction Multiple Thread (SIMT) model. Where the multicore hyperthreaded CPU can concurrently run at most number of threads equal to twice the number of cores, GPUs frequently run hundreds of threads at the same time (the exact number depends on the number of multiprocessors on the card, each of which executes 32 threads, known as a warp, concurrently).

A hardware schematic of a single core CPU vs. GPU is shown in Figure 2.1. Since the purpose of a modern CPU is exceedingly varied, including many applications revolving around memory accesses, string processing and threaded applications carrying out different procedures on individual threads, much more hardware is dedicated to controller and memory cache than in the GPU. Therefore the CPU allows for a lot of flexibility in usage and application building. On the other hand, the GPU was developed with a number-crunching purpose in mind and thus memory precision and flexibility were exchanged for additional arithmetic logic units. This trade-off is discussed at some lengths by various graphics card manufactures in release notes such as Nvidia CUDA Programming Guide[3]. In addition, since GPUs were originally designed to speed up the calculation in an application rather than carry the bulk of it, the cards lack a master controller that would allow for thread coordination from the graphics card, unlike their CPU cousins. Therefore GPU threads always have to be launched from the host (CPU implemented) code.

The premise of intensive arithmetic calculations on a large data set is of course not unique to determining the display bitmaps between monitor refresh rates. Many scientific problems also involve this process to converge on a numerical solution. To take advantage of this in the later part of 2000s[49, 50] more general purpose programming language extensions were being developed, with the most popular being Nvidia CUDA launched in

2007 and OpenCL in 2008.

The PIC simulation itself as previously discussed involves a simulation of a large number of particles being periodically updated with a set of parameters independent of the updates to other superparticles. However at low pressure most of the operations taking place are memory related, and therefore the arithmetic capabilities are not being utilised as much as could be desired. In addition since in most cases the CPU and GPU do not share the same physical memory space, particle data has to be copied between the two devices every time either requires access to the updated values. In more numerically expensive problems this performance overhead is offset by the speed improvement from the arithmetic portion of the procedure but in a memory access based application it may sometimes be more efficient to carry out the calculation on the host rather than the GPU[51].

2.1.2 Particle-In-Cell Models On GPU

The Particle-In-Cell code renaissance has been mostly made possible with advances in computer science in the last few decades. The basic structure of a PIC code has been largely unchanged until the advent of multi-core CPUs and GPUs. Importantly, the architectural shifts meant simulation code was no longer speeding up predominantly as a result of faster processor clock speeds. Instead it was the ability to carry out instructions simultaneously that has resulted in most of performance improvements in modern software.

With the 2007 launch of Nvidia CUDA computing architecture[49], first serious efforts were made to utilise some of the more exotic parallel computational acceleration methods in scientific computing. This included Nvidia's development of numerical method libraries for common numerical problems such as sparse matrix methods or Fourier transforms, as well as establishing

a registered developer community with extensive workshops and documentation to make up for any shortcomings in the range and maturity of products on offer.

In plasma science there has been a particular interest in adapting existing simulation techniques to take advantage of parallel processing capabilities in the last 8 years. Particularly in the case of Particle-In-Cell codes, where spatial cell positions are largely independent of each other within the timestep allowing for ease of parallelisation this has been a topic of interest. A deviation from the spatial independence is seen in the particle-to-grid interpolation for solving of the Poisson equation (see Sections 1.2 and 3.3), and was addressed by Stantchev *et al.* in [52]. This approach uses shared memory to reduce memory access times in the interpolation while allowing for a large amount of interthread communication in the GPU code. The same group also saw the GPU as a convenient aid in providing visual output for the PIC simulation, allowing for 2D display of plasma turbulances[53].

Since GPUs have latencies associated with memory access, it is convenient to keep the read/write operations organised within memory space for efficient calculations. In particular, this has been examined by Mertmann *et al.*[6], who give a number of different approaches to the organisation of particles in memory space with sorting grids of varying coarseness as well as their study of performance affects for different tread/block configurations. A 2D extension of a fully relativistic PIC simulation was benchmarked by Kong *et al.*[54], giving a measure of comparative performance to CPU code as achieved by the GPU implementation.

Finally, due to the moderately low expense associated with GPUs, a number of GPU clusters have become available to the scientific community as means of carrying out high performance computing, including the Irish

Centre for High-End Computing (ICHEC) GPU cluster[55]. As a result efforts have been made in plasma modelling research to take advantage of these facilities, with Burau *et al.* examining the case of a fully relativistic GPU PIC code on a cluster[56].

2.2 HPC Alternatives To GPU And CUDA

Some of the motivations for use of the GPU hardware architecture and CUDA platform were discussed at the start of this chapter. In this sections we will briefly list some of the HPC alternatives.

OpenCL

OpenCL is a popular alternative to Nvidia CUDA platform. It is a framework which allows execution across heterogeneous computing platforms consisting of not only CPUs and GPUs, but also digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors. It includes a language based on C99. Where CUDA is only available on Nvidia GPUs, OpenCL does not suffer this restriction, allowing for development on the slightly less costly AMD graphics cards. On the other hand OpenCL is a slightly newer technology than CUDA and thus some of the libraries available on the CUDA platform were not available on the OpenCL platform at the start of this project. In addition the CUDA C/C++ extensions are more integrated into C/C++ engineering standards than those of OpenCL.

Intel®Xeon Phi Coprocessor

A sophisticated alternative to the GPU is presented by the Intel®Xeon Phi architecture. This combines the accelerator approach of the GPUs with the

flexibility of the standard CPU processor. It supports MPI and OpenMP parallel platforms and provides the user with offload execution mode and native execution mode. The offload mode is similar to that of the GPU operation, where calculation is offloaded from the host system onto the accelerator device. In the native mode, the entire execution takes place on the accelerator, and the device emulates a many-core CPU system. Code designed for execution in the native mode has to be specified as such to compiler at compile time, thus making it fully portable to CPU execution as well as compatible with most standard development tools (e.g. debuggers and memory-checkers). Unfortunately the price of this accelerator is much higher than that of a GPU, being comparable to that of high end CPUs. The peak performances achievable on the Xeon Phi are also somewhat lower than those on the GPU.

Many-CPU Systems

Computer systems with many CPUs are largely outside the scope of this work, since they usually constitute supercomputer systems. The Irish Centre for High-End Computing (ICHEC) provides such facilities to Irish researchers, on successful requisitioning of computational resources.

2.3 Chapter Summary

In this chapter we gave a brief introduction to the topic of High Performance Computing. In particular we discussed the hardware design and specifications of the GPU architecture as well as the Nvidia CUDA parallel platform. A short overview of use of the GPU in PIC-MCC modelling was also given. Finally we outlined some alternative architectures and platforms for HPC.

CHAPTER 3

PIC-MCC Modelling

Since plasmas are a large collection of kinetically interacting charged particles, to characterise them both electromagnetic and collisional approaches have to be considered. These allow us to calculate a number of characteristic parameters and thus give us an overview of the behaviour of the plasma. From electromagnetic considerations we can establish a characteristic length at which electromagnetic force acts in a plasma, the response frequency of electrons to applied force (i.e. electric field) and macroscopic charge characteristics of the discharge. From kinetic interactions we obtain the minimal collision parameter and diffusion characteristics for the plasma[1]. These provide us with a considerable tool set for understanding plasma interactions and modelling of discharges[5, 9].

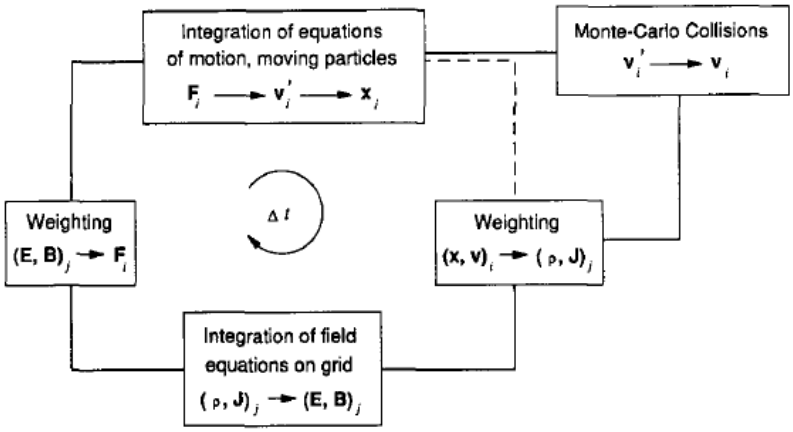


Figure 3.1: *Schematic flow diagram of a typical collisional Particle-In-Cell plasma model.[4]*

3.1 PIC-MCC Model Structure

Particle-In-Cell codes provide a very comprehensive physical view of a discharge. They employ very few simplifying assumptions, leading to detailed models of fundamental processes on even short timescales, as based on direct particle kinetics as opposed to energy distributions. Unfortunately this also leads to a large computational overhead for any more realistic laboratory set ups such as those employing complex feed gas mixtures, plasma chamber geometries, chemical (collisional) reactions or plasma-wall interactions.

PIC codes rely on treating the kinetic interactions directly on individual particle level. However since even moderately low pressure discharges consist of charged particles of density in the order of $10^{15}m^{-3}$[1], this number is really beyond the scope of a simulation. Instead the simulations track so called superparticles which correspond to an element of particle phase-space with appropriately adjusted charge and mass parameters. An individual su-

perparticle commonly represents on the order of 10^8 actual particles even at low pressure. This allows for representation of the plasma in terms of much more manageable superparticle densities, where each superparticle consists of a large number of real particles. This approach reduces the computational cost sufficiently to allow us to deal with the kinetic interactions of particles but the problem of electromagnetic interactions still remains. Summing up individual force contributions from each particle would result in operations on the order of N^2[9], which is, needless to say, unacceptable for any realistic simulation. Instead, PIC models use a particle-mesh interpolation, where superparticles contribute charge to a mesh point, with the electric field being calculated for individual positional mesh points rather than individual particles. The computational overhead thus reduces to the order of $N\log N$[9] operations, which is much more acceptable.

The actual flow diagram of a PIC model is shown in Figure 3.1. It is assumed that the simulation contains a finite number of superparticles with initialised positions and velocities. From the particle velocities we can interpolate the charges accumulated for each cell, which in turn allows for calculating the potential and electric field in each cell using the Poisson equation. Thus accelerations for particles in each cell can be determined. With this information the velocities and positions of each particle can be updated and the simulation can be advanced to the next timestep. Optionally during or after this particle push, collision simulation can be implemented. The appropriate method for simulating particle collisions depends on the collision frequency and the types of collisions present but is usually a variation on the Monte Carlo procedure incorporated into the PIC model, as described by Hockney and Eastwood[9] and Birdsall[4]. The method chosen in our code as well as the justification for this choice will be discussed in Chapter 5.

Finally, some numerical constraints have to be imposed on the PIC model to achieve satisfactory physical results rather than artificial numerical effects. A more detailed discussion of the reasons for these is given in Hockney's text[9], here we will only provide a brief statement of them. To begin, our cell size has to be small enough to allow for sufficient resolution of internal structures of the plasma and for correct solution of the Poisson equation. Due to charge shielding affects in plasmas, the cell width Δx has to be on the order of Debye length, with the common value in use being $\lambda_D/2$. The time step Δt also needs to be constrained, in this case to be short enough to resolve electron behaviour. These time scales are, as we have seen in Section 1.2, characterised by the plasma frequency ω_p. In this case the nominal values recommended are $\omega_p \Delta t \sim 0.2$. Finally, we require the length of our simulation to be much longer in comparison to Debye length to allow for proper resolution of the potential and electric field.

3.2 Discretisation Of Equations

In the PIC-MCC approach, the plasma model can be described through two major modelling components. The superparticles need to be advanced in space while simultaneously characterising their electromagnetic interactions. Therefore we need to obtain discrete expressions for the equations of motion and we also need to develop a numerical solution for the electric field equations to allow us to correctly determine localised forces on these particles.

3.2.1 Equations Of Motion

To determine the updates necessary for the velocity and position changes between timesteps, we have to consider the basic differential equations

$$m\frac{d\mathbf{v}}{dt} = \mathbf{F}$$

$$\frac{d\mathbf{r}}{dt} = \mathbf{v}$$

These can be written in the finite difference form as

$$m\frac{\mathbf{v}_{new} - \mathbf{v}_{old}}{\Delta t} = \mathbf{F}_{old} \tag{3.1}$$

$$\frac{\mathbf{r}_{new} - \mathbf{r}_{old}}{\Delta t} = \mathbf{v}_{new} \tag{3.2}$$

It should be readily apparent that the value of $\mathbf{F}_{old}$ and $\mathbf{v}_{new}$ on the right hand side (RHS) of Equations 3.1 and 3.2 are the average values of these parameters over the time interval Δt. It then follows that if Δt is small the variation of these parameters is approximately linear and the average value is the value given at the time interval mid-point. This is illustrated in Figure 3.2, where we see that the velocity parameter is always calculated at the half time step and the position at the full time step boundary.

3.2.2 Electric Field And Potential

To advance the charged particle velocities over the timestep, a solution to Maxwell's equations needs to be determined.

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0} \tag{3.3}$$

$$\nabla \cdot \mathbf{H} = 0 \tag{3.4}$$

$$\nabla \times \mathbf{E} = -\mu_0 \frac{\partial \mathbf{H}}{\partial t} \tag{3.5}$$

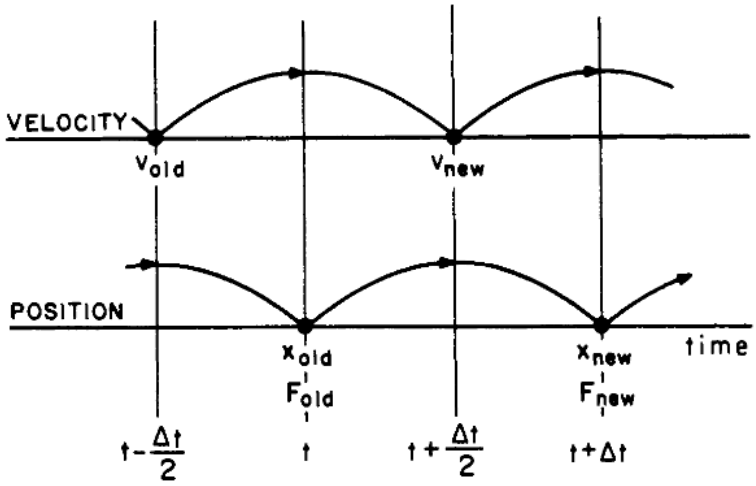


Figure 3.2: *Illustration of the leap-frog integration method, illustrating the time centering for the updating of position and velocity parameters.[5]*

$$\nabla \times \mathbf{H} = \epsilon_0 \frac{\partial \mathbf{E}}{\partial t} + \mathbf{J} \tag{3.6}$$

In the low-temperature, high-pressure, rf-driven discharge limit, Equation 3.5 can be further simplified as $\nabla \times \mathbf{E} \approx 0$. In the rf-driven discharge, the timescale of variation of the magnetic field is proportional to the frequency of the driving signal, ω_{rf}. This frequency is much lower than the electron plasma frequency, ω_{pe} (which in turn, in the high pressure limit is much lower than the collision frequency ω_c). Therefore on the timescales of the simulation timestep, the magnetic field only varies very slowly with time and thus becomes negligible.

Thus, since the curl of a gradient is zero, the electric field can be calculated from the potential and only the Poisson equation, as given in Equation 1.1, has to be solved for this simulation of this system. Applying the finite difference discretisation twice for the second derivative results in

$$\frac{\Phi_{j-1} - 2\Phi_j + \Phi_{j+1}}{(\Delta x)^2} = -\frac{\rho_j}{\epsilon_0} \tag{3.7}$$

and

$$E_j = \frac{\Phi_{j-1} - \Phi_{j+1}}{2\Delta x} \tag{3.8}$$

The PIC model allows for easy determination of the charge density ρ from the positions of the superparticles. To determine the potential Φ though, a boundary condition needs to be applied to the system. This is particular to the discharge being modelled, though common choices for the simplest system are the periodic boundary condition or the grounded boundary condition[9]. Based on the choice of boundary condition different approaches to finding Φ become appropriate. Many of these methods are discussed in the literature [5, 9] and therefore the discussion here will be limited to the method chosen in the implementation being presented here.

In two dimensions the solution for the potential becomes somewhat more complicated. The Poisson equation becomes non-linear in two dimensions and therefore an iterative approach to solving the system has to be adopted. With this in mind we decided to follow the approach of Vahedi *et al.*[57] and modify the Poisson equation in two dimension into its parabolic form, aiming to solve for steady state.

$$\nabla^2\Phi + \frac{\rho}{\epsilon_0} = \frac{\partial^2\Phi}{\partial x^2} + \frac{\partial^2\Phi}{\partial y^2} + \frac{\rho}{\epsilon_0} = \frac{d\Phi}{dt} \tag{3.9}$$

Each partial derivative can in turn be discretised as given in the left hand side (LHS) of Equation 3.7 resulting in

$$\frac{\Phi^{n+1}_{i-1,j} - 2\Phi^{n+1}_{i,j} + \Phi^{n+1}_{i+1,j}}{\Delta x^2} + \frac{\Phi^{n+1}_{i,j-1} - 2\Phi^{n+1}_{i,j} + \Phi^{n+1}_{i,j+1}}{\Delta y^2} + \frac{\rho_{i,j}}{\epsilon_0} = \frac{\Phi^{n+1}_{i,j} - \Phi^{n}_{i,j}}{\Delta t} \tag{3.10}$$

As is plain from the equation above, subscript i refers to the spatial coordinate in the x-direction, subscript j to the spatial coordinate in the y-direction and superscript n to the artificial time iteration. The system is considered to be in steady state when the time derivative on the RHS approaches zero.

3.3 Charge Accumulation And Weighting

The use of a finite grid for the purposes of calculating the potential and acceleration presents its own problems. Applying a simple binning procedure to our particle distribution results in a charge profile of the shape seen in Figure 3.3 (a). This is equivalent to finite sized particles of width δx occupying the cell. As a particle centre traverses the boundaries between cells we would observe a sharp jump between the densities of the two cells. The resultant potential would be very noisy due to the sharp charge density transitions at the wall boundaries. A similar problem is encountered when extrapolating between acceleration and velocity, where we observe a discontinuity in the variations of the accelerations in adjacent cells.

For this reason smoothing functions $\mathcal{W}_s$ are applied to the cell charge assignments and accelerations of particles during the weighting process, as based on the particle position. Most commonly these are referred to as zero-, first- and second-order weighting and the effective particle shapes associated with each are shown in Figure 3.3. The higher order schemes effectively turn each superparticle into a real particle cloud (the Cloud-In-Cell, CIC, model), where the clouds can move freely through each other. The first-order scheme can be obtained easily by applying a linear interpolation to the particle charge contribution as determined from its nearest grid point. In the second-order weighting a quadratic or cubic spline is applied, leading to better smoothing than the first-order weighting.

Unfortunately, the use of higher order weightings becomes more computationally costly with more complicated interpolation functions. However the smoother charge and acceleration profiles resulting from this procedure, while requiring more operations per particle, also allow for a coarser grid and fewer superparticles while producing the same physical results. This

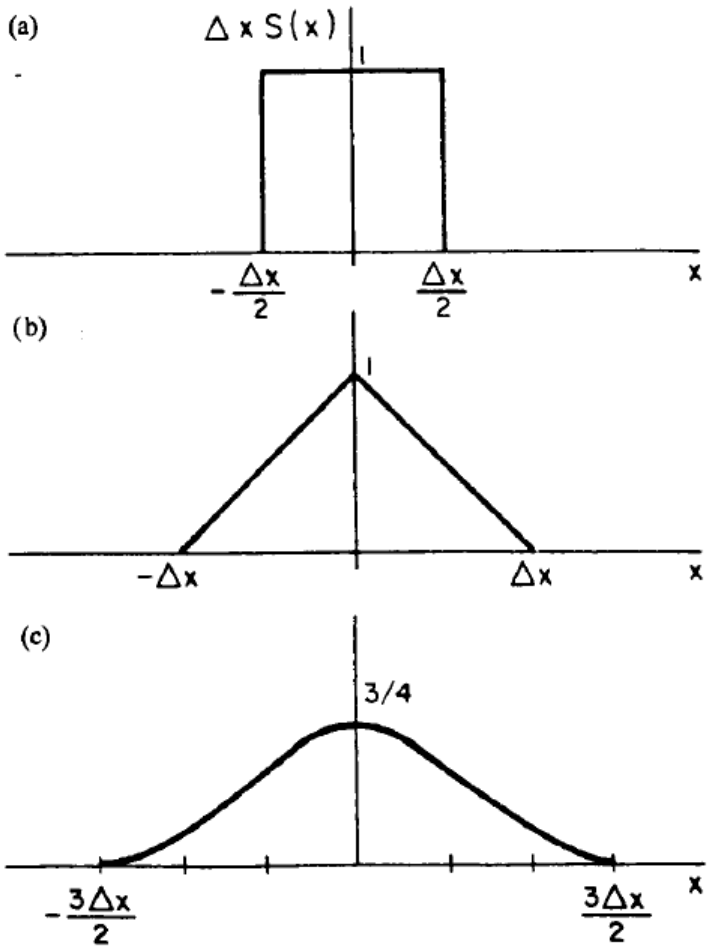


Figure 3.3: *Force and charge interpolation functions for PIC codes: (a) Zero-order (Nearest Grid Point); (b) first-order (cloud-in-cell, PIC); (c) second-order (parabolic or quadratic) spline.[4]*

goes some way to reduce the computational cost associated with smoothing functions. For most problems first order weighting is found to provide an acceptable compromise between simulation speed and smoothness of physical quantities[9].

It should be noted that the same weighting scheme should be applied to both the charge assignment and the calculation of a superparticle's acceleration. This is due to the smoothing function effectively setting the particle shape. Consecutive changes in the weighting function form would then result in alterations of this shape during the simulation. The outcome of this would be that particles could impose electric fields on themselves in addition to their contribution to the potential, resulting in a physical inconsistency in the simulation.

3.4 Normalisation

From the mathematics outlined in Section 3.2 it is clear that the equations being solved by the PIC model contain a number of physical constants throughout the simulation. Since these are essentially repetitions of the same operations it is computationally efficient to work in normalised units, where the cell width Δx and time step Δt are equal to 1. For simplicity the cell height is equal to the cell width. The above normalisation leads to the transformation

$$x' = \frac{x}{\Delta x}, \qquad t' = \frac{t}{\Delta t} \tag{3.11}$$

From these definitions it follows that

$$v' = v\frac{\Delta t}{\Delta x}, \qquad a' = a\frac{(\Delta t)^2}{\Delta x} \tag{3.12}$$

By relating the particle acceleration to the electric field and substituting the new normalised acceleration, a logical normalisation choice for electric field

presents itself as

$$a' = \frac{e(\Delta t)^2}{m_e \Delta x} E = E' \tag{3.13}$$

Substituting this normalised electric field value into the potential expression given in Equation 3.8, we find the normalisation constant for potential to be

$$\Phi' = -\frac{e(\Delta t)^2}{2m_e(\Delta x)^2}\Phi \tag{3.14}$$

Similarly, this can then be used in the Poisson equation (Equation 3.7) to find the normalisation for the charge density ρ. This is seen to be given by

$$\rho' = \frac{e(\Delta t)^2}{2m_e \epsilon_0}\rho \tag{3.15}$$

3.5 Monte Carlo Collisions

Collisions in PIC codes are implemented through the Monte Carlo procedure. The details of this implementation are dependent on the collision frequency as well as types of collisional processes being modeled. For collision cases of collision frequency up to once per timestep, the most frequently implemented technique involves decoupling of the collision handling from the velocity and position integration. This is achieved by carrying out the particle advancement as in the case of no collisions and applying the collision process at a fixed, constant point in the timestep. Most frequently the end of the push is chosen for these collisions however this is not a requirement of the scheme. In this case a pseudo-random uniform number is compared to the probability of collision for the particle, P, for the timestep. The collision probability P can be calculated from the mean free path $\lambda(v)$ as given by Birdsall[4]

$$P = 1 - \exp(\frac{-v\Delta t}{\lambda}) = 1 - \exp(-\nu_c \Delta t), \qquad \nu_c = \frac{v}{\lambda} \tag{3.16}$$

where ν_c is the collision frequency for the particle velocity. For $\nu_c \Delta t \ll 1.0$, the probability P becomes approximately equal to $\nu_c \Delta t$. Further selectivity

of collision processes can be achieved through the null collision method[58]. This requires obtaining a probability for each different type of collision being simulated and comparing the uniform number between these to select the appropriate collision process.

This procedure however limits the simulation to a maximum of single collision within a timestep. In many problems it is desirable to have a higher collision rate, which requires breaking down of the timestep into smaller segments. This can be implemented with the probability approach by comparing a uniform pseudo-random number to the collision probability at constant sub-timestep intervals, with the Leap Frog integration taking place between each comparison.

Alternatively a non-constant sub-timestep interval can be generated for each particle, producing the time until collision, as given by

$$\delta t = -\frac{\ln R}{\nu_c} \tag{3.17}$$

where R is a uniform pseudo-random number. In this approach the sub-timestep and timestep boundaries no longer have to align giving each particle an extra piece of data associated with it between each timestep. However this is a much more direct way of simulating collisions and at high collision frequencies can be more accurate. These techniques are discussed in further detail in Chapter 10 of Hockney *et al.* [9] or in the accompanying paper[37]. The schematic of the subdivision of the leap frog integration over the timestep is shown in Figure 5.1(b).

Generally collision frequency ν_c is determined from experimental collisional cross section σ data, through the relation

$$\nu_c = SN_g \qquad S = \sigma(\varepsilon)v \tag{3.18}$$

where N_g is the gas density, ε is the energy of the collision and v is the

relative velocity in the frame of reference for which the cross sections were measured. This is most frequently the centre of mass frame of reference but other frames may be chosen by different experiments.

In the case of collisions with other tracked particles instead of neutral gas density, the density of the collision partner species would be used. In addition, after successful resolution of the superparticle velocity modifications due to the collision, the collision partner particle would also have to undergo velocity modifications to account for this interaction.

3.6 Chapter Summary

In this chapter we provided a detailed overview of the particle-in-cell modelling technique, combined with Monte Carlo collision procedures when applied to plasmas. We discretised the equations of interest and discussed base techniques used to solve them. We have also introduced simulation techniques such as particle weighting and parameter normalisations, used to provide smoothing of values to counter the imposed discretisation and improve computational efficiency. We also introduced some of the restrictions on physical parameter sizes imposed by implicit assumptions within the computational procedures, in particular in relation to collision frequencies as compared to plasma frequency.

CHAPTER 4

GPU PIC-MCC Algorithm

As discussed in Section 2.1.1 the GPU uses the SIMT model for parallel processing. This is effectively just the Single Instruction Multiple Data (SIMD) model, which relies on having a large set of data whose every element is being operated on by the same set of instruction. Best performances are achieved when there is minimal instruction divergence between thread functions. In the case of the GPU, memory copy latencies also cause an operational overhead and thus it is profitable to minimise these in comparison to numerical operations. It is therefore desirable for the device code to consist of a large number of identical numerical operations on a sizable data set, very much like the case of calculating point transformations for graphics.

The collisionless PIC model would in fact be ideal for satisfying the first of these two criteria. The updating of the position and velocity of particles (particle push) takes the same form for all the particles and the cell specific

acceleration allows for a convenient thread assignment in the case of a large number of available threads. However the update of positions and velocities is mathematically straightforward and requires little computation so it is not very memory copy efficient. On the other hand collisional PIC codes improve the ratio of numerical operations to memory accesses due to recalculation of velocities necessitated by the particle collisions. As the collision frequency increases so does the arithmetic intensity. However since collisions are modelled using Monte Carlo methods, the interval between collisions is effectively randomised and the particles will experience varying numbers of collisions per timestep. This results in a deviation from the strict SIMT model and particularly at lower collision frequencies leads to load balancing problems and performance decrease.

At the end of each timestep, new accelerations have to be calculated from the updated positions of the superparticles. By this stage all particle positions and velocities need to have been updated before the simulation can proceed. This therefore presents the natural need for a global thread synchronisation point. While in general synchronisation points result in performance decrease they also effectively reset the loads for threads resulting in a smoother load balancing performance across the simulated timesteps (i.e. load balancing problem does not significantly disimprove with simulation time).

4.1 GPU PIC-MCC Architecture Overview

An outline of the our Particle-In-Cell model is given in Figure 4.1. This implements a slightly different architecture to the one normally encountered in PIC-MCC models (see Figure 3.1), since the latter usually decouple the push

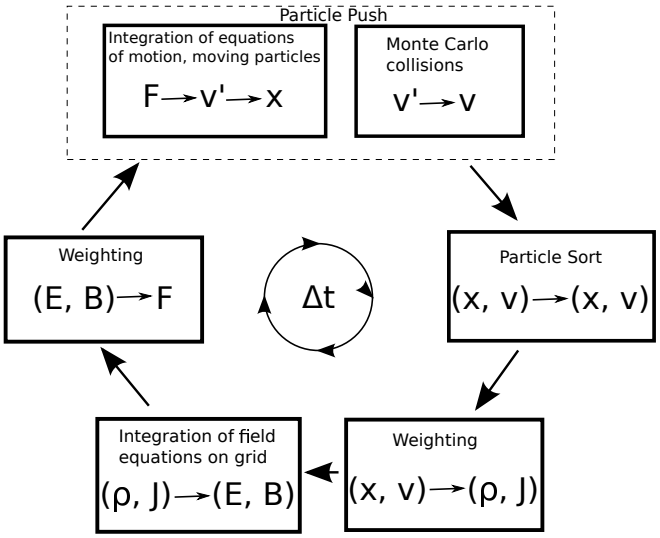


Figure 4.1: *Schematic of our PIC-MCC GPU implementation.*

and the collision procedures. However due to our wish to model collisions irregularly spaced within the timestep we required for these to be handled within the push itself.

Since coinciding reads/writes to a given memory location by multiple threads should be avoided due to the poor performance of atomic operations, it is necessary to determine a logical breakdown of the code into threads, with sorting of particles in memory according to their positions being a convenient way of improving the memory access patterns in the code. Therefore a rough outline of our code can be given as follows:

1. **Particle generation**. This part of the code loads particle data into memory and is only called in this form at the start of the simulation, though in the case of ionization can be reused to generate new super-particles.

2. **Field solver**. Here we calculate the accelerations of the particles in a cell as based on the charge density due to their positions. This operation needs to be initially carried out before the first push to determine starting accelerations. In the case of no weighting the charge density would be zero throughout the simulation but due to the application of higher order weightings to the particle shapes there will be very small non-zero acceleration in the cells.

3. **Particle pusher**. After determining the accelerations of particles for each cell, the velocities and positions need to be updated. This step accounts for all the kinetics of the particles and it is here that we implement our Monte Carlo collisions. In the case of atmospheric pressure plasma it is this step which is potentially most time consuming. The high density leads to a large number of collisions and thus velocity recalculations, significantly increasing the arithmetic intensity of the simulation. This step is also the most easily parallelisable, particularly in the presence of some form of sorting algorithm, allowing for easy treatment of particles within a given cell in bulk.

4. **Particle sort**. Once the particle positions have been updated it is useful to sort them in the continuous memory space according to their physical position. This reduces the latency associated with memory access across the particle data and allows for convenient hierarchy for parallelisation in the rest of the simulation (i.e. for summing over the charge contributions in the cell or for determining the appropriate acceleration as based on the particle cell). The acceptability of the performance overhead associated with this procedure, much like the memory copy latency, is dependent on the arithmetic intensity of the

rest of the simulation, so performance increases for more collisional PIC models.

All of the above listed steps are performed on the device as GPU functions (kernels) and are discussed in greater detail below. In the case of multiple superparticle species (such as in the case of electrons and ions), the CPU host code launching the kernels loops over serially for each specie. On the current hardware setup this is acceptable due to the relatively small number of multiprocessors available on our test card as well as the large number of cells in our simulation but would potentially present another convenient natural parallelisation division on multicard system.

4.2 Memory Allocation

Since PIC models rely on kinetic treatment of the plasma particles each superparticle has to have an associated position and velocity. In the 1D3V and 2D3V cases there is one float or float2 position variable respectively and one velocity float3 component kept in the associated storage, as discused in the following Section 4.2.1. In addition an integer specifying the species of the particle is kept in memory, and a float giving the generated time to next collision is necessary. Due to the GPU and CPU not sharing memory space the memory to contain these variables has to be allocated individually on both processing units.

4.2.1 Particle Data

The most important memory building block of our simulation is the particle class. An instance of this class contains all the information on the individual superparticle, including kinetic and positional information, specie, relevant

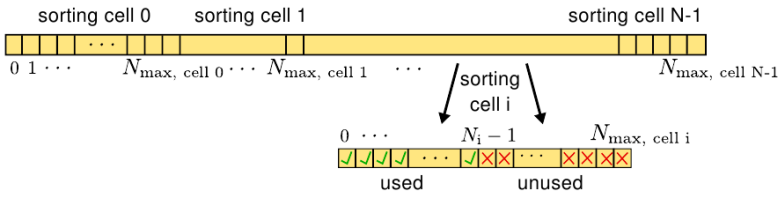


Figure 4.2: *Data array allocation. The cells are of set sizes with free space after valid particles to allow for sorting between cells.[6]*

constant physical parameters and time to next collision. Particle objects are arranged in an array, sorted by cell position in the plasma and with some empty space at the end of each cell, necessary to provide space for sorting particles after cell migration. This is illustrated in Figure 4.2. A copy of this array needs to be maintained in both device and host memory. Where possible vector variables (i.e. float2 and float3) are used to express physical parameters.

4.2.2 Cell Data

Since accelerations are calculated on per cell basis, for weighting purposes every cell stores acceleration values for each cell edge, which is then weighted according to position. In the case of ionizing collisions the number of charged particles created is stored as an integer in each cell. This variable is also used in the case of rescaling to allow for additional particle creation. As discussed above, the particle arrays contain empty particle holders at the end of each cell. Therefore the cell data structure stores the index of the first empty element in each cell and the first free element at the end of the cell, to be used for designating particles leaving the cell later in the simulation (this is described in detail in Section 4.5). All of these parameters also require a

CPU and a GPU copy to be useful on either processor.

4.2.3 Field Data

The field solver implemented in our model differs between the one and two dimensional case. The one dimensional case uses a Sine transform based solution, while the two dimensional solver finds the potential through a dynamic alternating direction implicit solver (DADI). Therefore the two cases have slightly different requirements for the memory space allocation.

The weighting of the charge density contributions requires access to the charge densities of cells handled by adjacent threads. Therefore local variables which provide for much faster memory access, cannot be employed in the charge summation. However performance during this step can be improved through the use of shared memory rather than global memory for summation charges due to the particles. Unlike local memory, shared memory can be accessed by all threads within a thread block. This implementation however requires the addition of boundary arrays to the field data structure. The charges at the thread block boundaries are then resolved on the host, thus avoiding the use of atomic operations in calculating the remaining charge at the shared memory block boundaries.

1 Dimension

In the one dimensional case, for the grounded electrode boundary condition, we use the Sine Transform to determine the solution to the Poisson equation. This is numerically obtained from a modified array for Fourier Transform, and uses the CUDA Fast Fourier Transform (CUFFT) library for the transformation itself. We therefore have a host and a device allocated special complex variable arrays required by this library, for the density and poten-

tial respectively. Charge accumulation is injected into the real component of the complex charge density variable. After the potential solution has beem obtained and arrays have been transformed back into the original domain, the real components of the potential elements contain the calculated potential values.

2 Dimensions

The two dimensional solver no longer requires complex variables for the charge and potential arrays and thus float arrays suffice. In addition an instance of the DADI solver is also created and maintained throughout the simulation, since it has to be reused on every field solution. This solver instance requires a memory overhead to allow for mathematical manipulation of the input arrays. This includes destination arrays for matrix transpose[59] and arrays for determining dynamic timestep modification[60]. The DADI solver itself uses a parallel cyclic reduction (PCR) solver to find the solution to the tridiagonal system. The PCR solver in turn requires storage space to record the tridiagonal matrix being solved.

4.2.4 Miscellaneous

It is useful to calculate the particle density per cell during the simulation and it is necessary prior to sorting particles to determine whether the designated cell memory contains sufficient space for the migrating particles. These values are stored in host integer arrays of the cell simulation length. The allocation of addition GPU and CPU particle memory has to take place on the host and therefore no GPU memory counterpart for the particle count is required.

4.3 Particle Generation

The particle and cell data is populated with values on the device. In our code the particle push and generation use the convenient parallelisation already inherent from the use of a cell structure for solving of the Poisson equation. The number of cells in a given 1D simulation is usually on the order of hundreds since the cell width is confined to be of the order of Debye length and most plasma systems of interest are much larger. This number only increases for higher dimensional simulations. Therefore it would be challenging to deploy a simulation of such parallelism efficiently on a workstation CPU, with only a small number of concurrent threads available. However this configuration is well suited to the GPU, where thread numbers are expected to be large even for a single card.

The initial number of superparticles per cell is constant, with positions being stored as absolute numbers in the memory. This allows for ease of debugging since the code relies less on the relative particle memory position in the data structure. The position between the cell boundaries is generated as a pseudorandom number from the CURAND device library API. Similarly the velocity of each superparticle is initialised using the normal distribution from this API, centered around the appropriate thermal velocity for the species. The electrons are assumed to be at 30 000K while the ions are in thermal equilibrium with the feed gas at 300K.

The species integer variable is set as 0 for empty memory particle slots to allow for quick checking for actual filled particle positions. It is set as 1 for electrons and higher numbers for other species. Time to last collision is generated from Equation 3.17 in the previous chapter. In the cell data structure an index is initialised to record the location of the next free particle space for that cell in the particle data array. The index recording the end of

the cell's particular particle data array section is also recorded.

Since each cell has to have particles followed by some amount of allocated free space to allow for particle sorting (see Fig. 4.2), after recording all the valid superparticles, this space is also initialised. All the particle data parameters in the empty slots are set to zero and, as mentioned, the space is marked as invalid particles by setting the species value. During the particle count procedure mentioned in Section 4.2.4 it is then simple to confirm the agreement between the index values and the actual particles contained in these designated spaces by checking the value parameters of the empty particles.

4.4 Particle Pusher

In our particle pusher we wanted to implement collisions without the traditional limit on collision frequency of $\nu_c \Delta t \ll 1.0$ and therefore we chose the second approach discussed in Section 3.5. Here time between collisions δt is calculated directly, and thus collisions do not uniformly align at a given point on the timestep. The repercussions of this complication on the position and velocity integration itself will be further discussed in Chapter 5, in this Section we will only give a brief outline of our pusher.

The particle push is designed to update the velocity and position coordinates of the superparticles. The GPU implementation parallelises this operation on thread per cell basis, much the way particle initialisation is handled. The list of particles in the cell is cycled through serially by each individual thread and particles are pushed to the beginning of the next timestep in sub intervals dependent on the collision frequency.

The general outline of the particle pusher can be summarised as follows:

1. Calculate thread-specific parameters, such as the cell position and acceleration gradient in the cell.

2. Iterating backwards (from the end of filled space) over all the particles in the cell, the acceleration of the particle at the start of the pusher is calculated using first order weighting, as discussed in Section 3.3. This value is used throughout the timestep as opposed to being recalculated for each hop.

3. Each particle is collisionally advanced to the end of the timestep using our custom integration procedure described in Chapter 5

4. Last advancement segment of the push check whether the particle has escaped the discharge boundary and if so removes it from the particle data array. To avoid empty spaces in the continuous memory block, this empty position is filled by the last particle in the block. Since the particles are iterated over backwards, this replacement particle has already been updated for this timestep and so our executing thread can move on with no further action necessary.

The push procedure takes place completely on the GPU device. This presents a very important limitation. The memory space allocated for the particle array cannot be changed at this point in time due to global dynamic memory allocation not being enabled on the device. Therefore at this point only the number of particle creating collisions in each cell are recorded for every particle specie. These particles are not actually created until control of the memory array is returned to the host and it has been ensured enough memory has been allocated for the new particle density.

4.5 Particle Sort

For book-keeping and memory access purposes it is useful to keep the particles positioned in a particular cell also close to each other in the memory. Therefore our code implements a particle sorting algorithm as based on the work of Mertmann *et al*[6]. This consists of three kernels and employs a single atomic operation in the last of these kernels. While this procedure does add an overhead to the simulation, particularly due to the process being focused on memory operations rather than arithmetic operations, it is seen, as will be later discussed, that at higher collision frequencies this operation takes up only a small fraction of the total computing time.

However before particle sorting can take place we need to determine whether the existing allocated particle data memory array contains sufficient space to accommodate the resolution of the particle flux through the cell. Due to the implicitly arbitrary order in which threads are cycled through on the hardware it is impossible to guarantee all the leaving particles have been moved before new ones enter. Thus each cell has to be simultaneously handle the maximum number of particles present if all transient particles have arrived at this destination before and of the leaving particles have been removed. It is therefore necessary that prior to initiating particles sorting, the host code sums over the particles leaving and particles entering a cell and allocates additional space if necessary. Any particles created during ionization are also calculated into this space allocation, to avoid having to repeat this particle summation after sorting. The actual ionization particles themselves are only added to the species data after the sorting, to minimise the number of particles traversed during the sort. This is carried out using the generation mechanism used in the initial data population during particle generation.

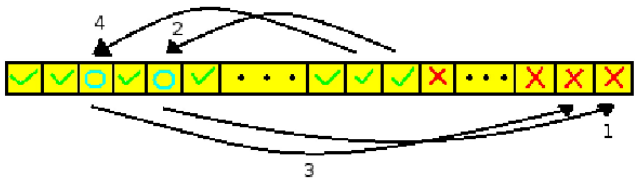


Figure 4.3: *First kernel of the sorting algorithm. Each thread checks the particles in its cell in reverse order. If the particle has left the cell it is moved to the end of the cell's free space (step 1). It then moves the last particle (first to be checked) into the freed position (step2). This is repeated for any further particles (steps 3 and 4).*

4.5.1 Sort Kernel 1 - Particles Leaving The Cell

The first kernel of the sorting algorithm is outlined in Figure 4.3. As in the case of particle push, an individual thread is launched for each cell and proceeds to iterate through the particles in backwards order (i.e. from last to first by index in the array). Each particle's position coordinate is checked with respect to the cell position and if the particle is found to now correspond to a different cell, its data is copied to the end of the free space in the cell. The data of the last indexed particle still in the cell is then moved to the now free position in the memory to avoid empty spaces in the memory chunk containing the particles still in the cell. The cell index parameters marking the end of the valid cell particles and start of particles found to be leaving the cell are adjusted accordingly for the particle movement. This is in contrast to the procedure outlined by Mertmann *et al.*, where the step of filling up freed spaces in the valid particle portion of the array is done in a separate particle iteration after any invalid position particles were moved to the end

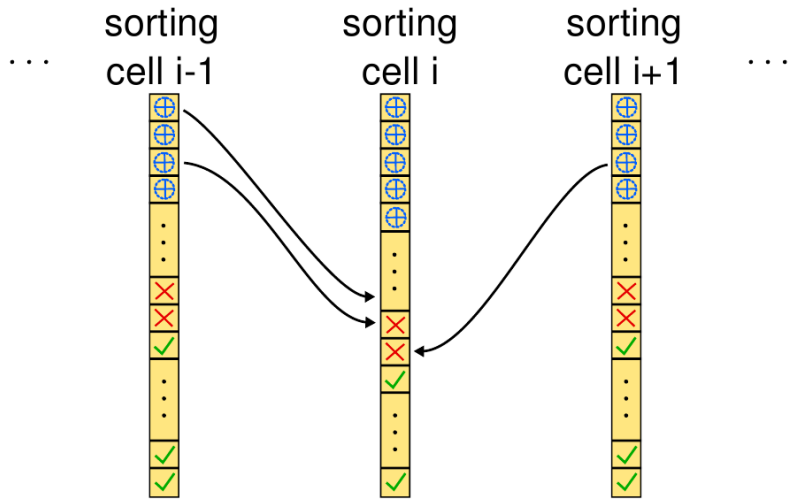


Figure 4.4: *Second kernel of the sorting algorithm. Each cell first looks to the cell to its right to iterate through the particles leaving the cell (i.e. the particles at the end of its free space). Any particles corresponding to the thread cell position are moved into the new cell and erased from the data of the right side cell. This procedure is then repeated for the particles of the cell to the left of the thread cell. Image taken from [6].*

of free space[6].

4.5.2 Sort Kernel 2 - Particles Moving To Adjacent Cells

Next, a kernel is launched to move particles from adjacent cells into their new cell. Every thread checks each of its adjacent cells in turn for particles whose position corresponds to that of the thread's cell. Any particles satisfying this criterion are moved to the thread's cell at the end of the valid data array, their data in the adjacent cell being erased. The index for valid data in the cell is also incremented. The whole procedure of the second kernel is

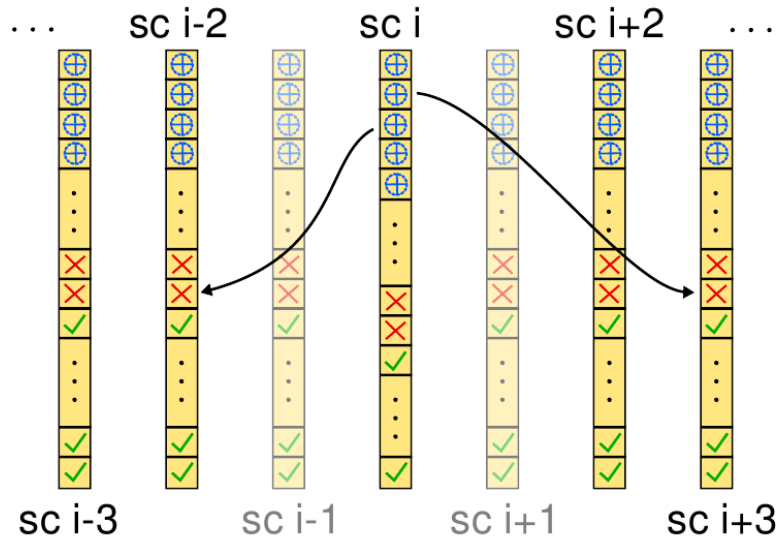


Figure 4.5: *Third (final) kernel of the sorting algorithm. Each thread checks its original cell for remaining particles (ones that traversed further than one cell) and moves them to the appropriate cell. This step uses one atomic operation. Image taken from [6].*

summarised in Figure 4.4. Since only one thread can ever be writing to its cell particle array and each moving particle can also only correspond to one cell, no atomic operations are necessary to implement this kernel. While in the 1 dimensional sort case only the cells left and right have to be checked, in the 2 dimensional case we extend this principle to cells above, below and diagonal to the current cell.

4.5.3 Sort Kernel 3 - Particles Migrating Over Multiple Cells

Finally any outstanding particles which have traversed more than one cell length are moved to their appropriate cells in the third kernel of the algo-

rithm. Each thread checks over the moving particle memory space at the end of the cell's particle data array for any remaining particles. The particle position is used to determine the new cell index and a single atomic operation is used to find the index of the first free space after valid, sorted particles in that cell. The thread then copies the particle's data to this new cell and erases it from its own cell's particle data. Figure 4.5 outlines this procedure.

4.6 Particle Addition

Once all the particles have been sorted any ionization which took place over the last timestep has to be resolved. The existing particles have had their velocity effects resolved during the push kernel, however the collisional products are now added to each cell, starting in the first free position after the freshly sorted particles. The new particles are initialised with a random position in the cell and energy normally distributed around the expected species temperature. This step is done in parallel on the device in a fashion analogous to the initial creation of particles. One side effect of this collision handling is that energy of the system is not conserved. However deviation from energy conservation should be relatively small, particularly in either low collisional cases and/or in cases of relatively good species thermal agreement with the assumed distribution.

4.7 Field Solver

Finding the electric field solution for the system is possibly the most mathematically complex part of the PIC model. The type of field solver used generally depends on the choice of boundary conditions of the plasma, geom-

etry of the chamber, and number of dimensions required in our simulation. Since we chose to implement both one and two dimensional models, each required a different solver. In both cases we chose to implement the grounded boundary condition, allowing for simple superposition of RF potential for a more reasonable physical model.

4.7.1 1 Dimension

As discussed in Hockney *et al.*[9], in the special case of a grounded boundary solution an efficient method of calculating the electric potential of the system is through the use of Sine transform. This useful transform is expressed as

$$\hat{\phi}_k = \sum_{x=1}^{N-1} \phi_p \sin\Big(\frac{\pi k x}{N}\Big) \tag{4.1}$$

with each boundary value tending to zero. This method is very similar to the use of the Fourier Transform

$$\phi_k = \sum_{x=1}^{N} \hat{\phi}_x \exp\Big(\frac{-2\pi i k x}{N}\Big) \tag{4.2}$$

to solve differential equations, with the exception that while in the case of an exponential we obtain the exponential back on every instance of differentiation, the Sine Transform requires two differential iterations for the Sine term to reappear.

Therefore the relation

$$\frac{d^2}{dx^2}\hat{\phi}_k = -\Big(\frac{\pi k}{N}\Big)^2 \hat{\phi}_k \tag{4.3}$$

can be used to change the integration procedure into a multiplication in the transformed domain. Substituting Equation 4.3 into the Poisson equation 1.1, where the charge density has had the Sine transform applied to it results in

$$\hat{\Phi}_k = \frac{N^2 \rho(k)}{\epsilon_0 \pi^2 k^2} \tag{4.4}$$

To obtain the potential in the x rather than the k domain we apply the inverse Sine transform to $\hat{\Phi}_k$. Conveniently, the Sine Transform is also its own inverse with the scaling factor of $N/2$ [61].

Due to the discretisation of our Poisson equation in 3.7, we also need to modify our scaling factor for differentiation in the k domain. Equation 4.4 then becomes

$$\hat{\Phi}_k = \frac{\rho(k)}{\epsilon_0 K^2} \tag{4.5}$$

where

$$K^2 = \kappa^2 \left[\frac{\sin \frac{\kappa \Delta x}{2}}{\frac{\kappa \Delta x}{2}} \right]^2, \qquad \kappa = \frac{\pi k}{N}$$

As the cell grid becomes finer the value of K approaches that of κ. For a discussion of the difference between the two parameters the reader is referred to Birdsall *et al*[5].

When actually implementing this solution computationally, firstly let us consider obtaining the RHS of Equation 3.7. In our field data structure we have a density array of length N_{cell} - 1 allocated. This is because we are effectively finding the charge density and thus potential on each cell boundary, with the outer extremities being grounded. From Equation 3.8 it is however obvious that a value for the potential is necessary for the imaginary cells just outside our chamber for us to be able to determine the acceleration at the boundaries. Here we approximate the potential value to be the inverse mirror image of the corresponding value on the valid side of the boundary, making the potential effectively half a cycle of a periodic pattern on the simulation length interval.

Since multiple species of different charges are present in the simulation, the charge density is initialised as 0. The different species are then iterated over on the host with their charge contributions added to the charge density array as appropriate. However the first order weighting being applied to

the charge summation also requires each cell to contribute some charge to its neighbouring cell. For this a temporary shared cell charge structure is declared in the kernel charge summation function, to account for the particle charge contribution to the left and right cell boundary as based on its position in the cell. Once the thread block has finished its summation, each thread looks to its right side neighbour and adds its left boundary contribution to its own right side value. This charge is then store in the charge density array. However a layer of complexity is introduced due to shared memory being only visible for threads of the same block. Therefore the left side boundary value for the left-most cell in the thread block is stored in global memory and added to the charge density sum on the host after the conclusion of the kernel.

As was discussed, we have opted for a grounded boundary condition and chose the Sine Transform method to solve for the potential. The Sine Transform given in Equation 4.1 above, makes its similarity to the Fourier Transform readily apparent. In fact the Fourier Transform can be used to conveniently obtain the Sine Transform and thus we can take advantage of the parallel Fast Fourier Transform implementation available from the official CUDA toolkit libraries. This procedure is described in detail by Press *et al.*[61] and requires the construction of an auxiliary array from the original data. This new array then undergoes the Fourier Transform, output of which can then be used to re-construct the Sine Transform of the original array.

The auxiliary array y_j is constructed from the Sine Transform data $f_j =$

$0, ..., N$ as follows

$$\begin{aligned} y_0 &= 0 \\ y_j &= \sin(j\pi/N)(f_{j-1} + f_{N-j}) + \frac{1}{2}(f_{j-1} - f_{N-j}) \\ j &= 1, ..., N \end{aligned} \tag{4.6}$$

After applying the Fourier Transform to this array it can be seen that the Sine Transform is given as

$$\begin{aligned} S_{2k+1} &= I_{k+1} \\ S_{2k} &= S_{2k-2} + R_k \\ k &= 0, ..., N/2 \end{aligned} \tag{4.7}$$

where I_k and R_k are the imaginary and real component of the Fourier Transform respectively. Since the even terms of the Sine Transform require recursion, we require a starting point for the elements, in this case

$$S_0 = \sum_{j=0}^{N-1} f_j \sin(j\pi/N) \tag{4.8}$$

After obtaining the sine transform of the charge density, solving the differential equation in the frequency domain becomes a problem of multiplication, as described in Equation 4.7.1.

To obtain the differential equation solution in the spatial coordinate domain we need to apply the inverse Sine Transform to the data set. Fortunately the inverse transform can be obtained by applying the Sine Transform a second time, with the resulting data being scaled by a factor of $N/2$ with respect to the original data. Therefore the data has to be normalised for this scale factor to obtain the potential due to the charge distribution.

Finally we use the finite difference method applied to the potential to find the electric field at each cell boundary. This gives us the force on each species and thus the accelerations to be applied to each particle.

4.7.2 2 Dimensions

As discussed in Section 3.2.2, the solution to the Poisson equation becomes more complex in 2 dimensions. The elliptic form of the Poisson equation is given in Equation 3.10. As discussed by sources such as Vahedi *et al.*[57], by introducing the artificial time dependence we can solve the Poisson equation for the steady state (i.e. the time derivative term goes to zero). The iterative procedure itself takes place around the artificial time parameter Δt, which can be dynamically adjusted to provide faster convergence.

In our parallel implementation we based our solver on the work presented by Wei *et al.*[62]. Taking Equation 3.10 and rearranging to group expressions in terms of spatial and time coordinates we obtain

$$\Phi^n_{i,j} = \Big(1 + \frac{2\Delta t}{\Delta x^2} + \frac{2\Delta t}{\Delta y^2}\Big)\Phi^{n+1}_{i,j} - \frac{\Delta t}{\Delta x^2}\Phi^{n+1}_{i+1,j} - \frac{\Delta t}{\Delta x^2}\Phi^{n+1}_{i-1,j} - \frac{\Delta t}{\Delta y^2}\Phi^{n+1}_{i,j+1} - \frac{\Delta t}{\Delta y^2}\Phi^{n+1}_{i,j-1} - \frac{\rho_{i,j}}{\epsilon_0}\Delta t \quad (4.9)$$

Calculating a solution to this expression then relies on trying to converge onto the steady state through a series of successive double sweeps. By breaking a single timestep iteration up into half steps, where each half step sweeps over one spatial dimension in turn, while assuming values from the previous half sweep for the other spatial dimension, an approximation to the solution can be found. For the first sweep we reformulate Equation 4.9 as

$$a_i\Phi^{n+1/2}_{i-1,j} + b_i\Phi^{n+1/2}_{i,j} + c_i\Phi^{n+1/2}_{i+1,j} = d_i \quad (4.10)$$

where $a_i = c_i = -\frac{\Delta t}{\Delta x^2}$, $b_i = \Big(1 + \frac{2\Delta t}{\Delta x^2} + \frac{2\Delta t}{\Delta y^2}\Big)$ and $d_i = \Phi^n_{i,j} + \frac{\Delta t}{\Delta y^2}\Phi^n_{i,j-1} + \frac{\Delta t}{\Delta y^2}\Phi^n_{i,j+1} + \frac{\rho_{i,j}}{\epsilon_0}\Delta t$. For the second sweep the expression instead becomes

$$a_j\Phi^{n+1}_{i,j-1} + b_j\Phi^{n+1}_{i,j} + c_j\Phi^{n+1}_{i,j+1} = d_j \quad (4.11)$$

with the coefficients taking the form $a_j = c_j = -\frac{\Delta t}{\Delta y^2}$, $b_j = \Big(1 + \frac{2\Delta t}{\Delta x^2} + \frac{2\Delta t}{\Delta y^2}\Big)$ and $d_j = \Phi^{n+1/2}_{i,j} + \frac{\Delta t}{\Delta x^2}\Phi^{n+1/2}_{i-1,j} + \frac{\Delta t}{\Delta x^2}\Phi^{n+1/2}_{i+1,j} + \frac{\rho_{i,j}}{\epsilon_0}\Delta t$.

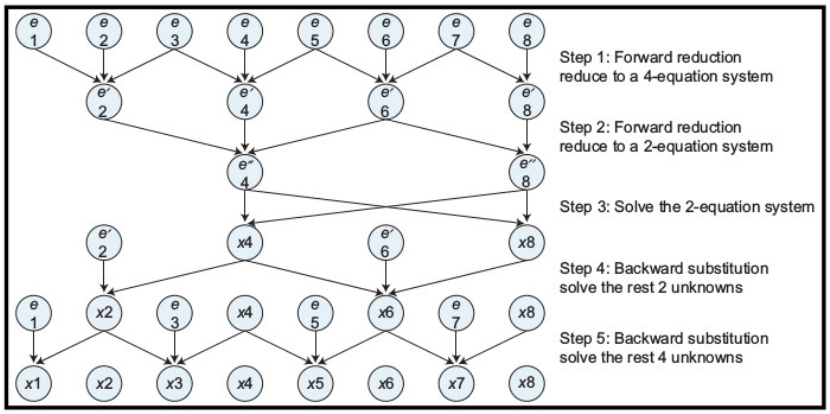


Figure 4.6: *Schematic of the cyclic reduction solver for tridiagonal equation systems, as described by Zhang et al.[7]*

As can be seen from Equations 4.10 and 4.11, in both parts of the sweep the resultant is a series of tridiagonal systems across the y- and x- direction, respectively. Much work has been dedicated to the problem of efficiently solving tridiagonal systems in the literature and with consideration for our computer architecture we opted for the parallel cyclic reduction solver (PCR), describe by Zhang *et al.*[7].

The classical cyclic reduction (CR) solver is illustrated in Figure 4.6. At the first step of the cycle each set of adjacent 3 equations in the linear system will only consist of a small number of elements of Φ. For example, for the first sweep of the double sweep a typical set of three adjacent equations can be expressed as

$$\begin{aligned} a_{i-1}\Phi_{i-2,j}^{n+1/2} + b_{i-1}\Phi_{i-1,j}^{n+1/2} + c_{i-1}\Phi_{i,j}^{n+1/2} &= d_{i-1} \\ a_i\Phi_{i-1,j}^{n+1/2} + b_i\Phi_{i,j}^{n+1/2} + c_i\Phi_{i+1,j}^{n+1/2} &= d_i \\ a_{i+1}\Phi_{i,j}^{n+1/2} + b_{i+1}\Phi_{i+1,j}^{n+1/2} + c_{i+1}\Phi_{i+2,j}^{n+1/2} &= d_{i+1} \end{aligned} \tag{4.12}$$

In this expression it would be simple to eliminate most of the terms on the

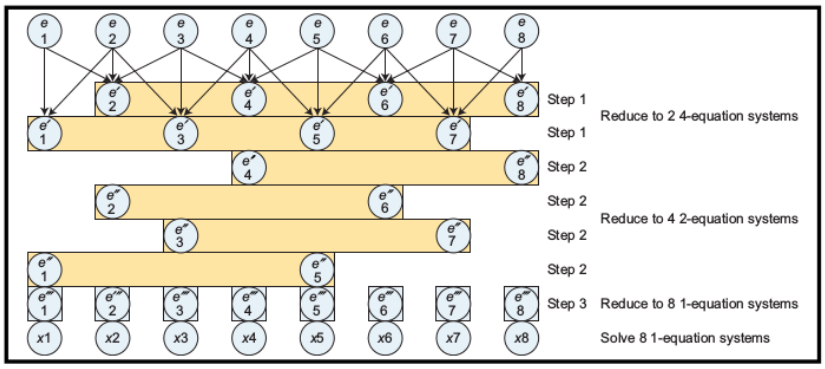


Figure 4.7: *Schematic of the parallel cyclic reduction solver for tridiagonal equation systems, as described by Zhang et al.[7]*

LHS, leaving only the terms including $\Phi_{i-2,j}^{n+1/2}$ and $\Phi_{i+2,j}^{n+1/2}$ term. As shown in the figure, by applying this reduction, centered on every even numbered equation we immediately half the number of systems. The same can then be repeated for the new set of equations to half the number of these and so on until the problem is trivialised to a set of two equations, which can be solved quite easily. Afterwards we can apply backwards substitution to recover the other values of Φ in the system.

The parallel cyclic reduction (PCR) is an extension of the ideas of the CR solver. In this approach rather then only aiming to obtain a direct reduction solution at 2 points in the system, relying on backwards substitution for the rest, we instead hope to utilise parallelisation to solve the system simultaneously. As shown in Figure 4.7, reduction now takes place centered on every point in the system. We rely on the parallelism managing the overhead of the increase in calculation needed, while gaining the benefit of not needing the backwards substitution procedure.

The algorithm presented by Zhang *et al.*[7] expects multiple linear system sets to require solving, which is consistent with our problem, where each

row or column represents a distinct linear system. The partitioning of this system is done by dedicating a thread block to each system, while each equation in a single system is handled by a single thread. This allows for each equation system to be able to impose thread synchronisation points in the execution after each reduction step - a necessity due to the next reduction cycle relying on equations resultant from the previous step. In addition, in Zhang's implementation the system data is copied into shared memory to speed up memory accesses.

Unfortunately the fixed maximum number of threads available to a thread block and the limited amount of shared memory available on the GPU also limit the size of each linear system in our problem. Therefore we decided to use the much larger global memory to hold our intermediate solution steps and also to generalise the procedure to allow for each thread to handle more than a single equation during each reduction cycle. Synchronisation, still necessary due to each new reduction cycle using a previously reduced equation, is only called after each thread has looked after every equation assigned to it. We have also generalised our procedure from Zhang's to handle system sizes other than power-of-two sized ones through ensuring that each "step" between equations being used in the current reduction does not extend our read to outside the system size.

As discussed by Wei *et al.*[62] since ADI procedure sweeps alternately over rows and columns, for one half of the sweep the memory arrangement of our linear system will not be aligned. Each element will effectively be offset by the length of the system in the perpendicular direction. This can be problematic for performance since reads from memory take place in bursts [63]. A memory burst delivers to the function not just the requested data but also some of the data adjacent to the location initially queried. This

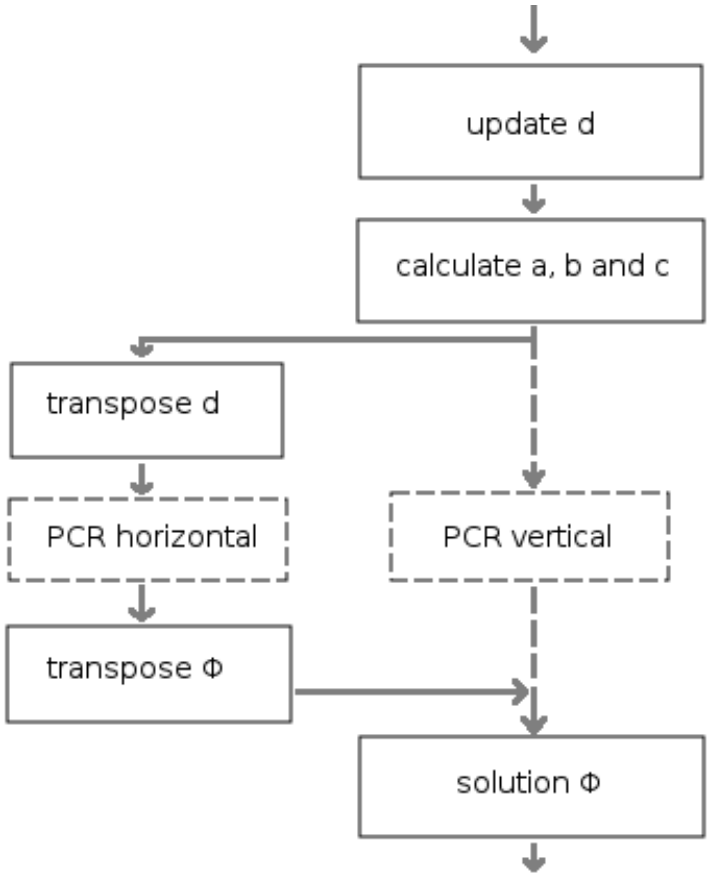


Figure 4.8: *Schematic of the parallel ADI solver with matrix transpose to allow for better memory alignment during each half cycle.*

is utilised to significantly reduce reads overhead on the GPU when threads simultaneously request adjacent memory by the function taking the data from the first burst, rather than reissuing a read request. Therefore for half the ADI sweep, the memory mis-alignment results in our reads being significantly slower.

Therefore it is beneficial to the performance of the solver to first transpose the linear systems before carrying out the unaligned part of the sweep. An efficient matrix transpose algorithm has been developed by Ruetsch *et al.*[59] and can be used to great effect to remove the memory alignment problem. By transposing the matrix, we reverse the organisation of the elements in the memory so for the purposes of cyclic reduction each new system is now again adjacent in memory. The whole ADI procedure is summarised in Figure 4.8.

As mentioned above, the rate of conversion on the steady state solution can be significantly improved by dynamically adjusting the artificial timestep parameter size[60]. We define a parameter TP as

$$TP \equiv \frac{\| \Phi^{n+2} - {}^{-}\Phi^{n+2} \|}{\| \Phi^{n+2} - \Phi^{n} \|} \tag{4.13}$$

Φ^{n+2} is obtained by applying 2 double sweeps to Φ^n, while ${}^{-}\Phi^{n+2}$ is calculated by applying a single double sweep of timestep size 2Δt. Doss *et al.*[60] determined that depending on the range between which the parameter TP falls it is advantageous to adjust the timestep Δt by a constant factor. For intervals (-∞, 0.05], (0.05, 0.1], (0.1, 0.3], (0.3, 0.4] and (0.4, 0.6] these factors are 4, 2, 1, 0.5 and 0.25 respectively. In the event that TP falls above 0.6, it is recommended that the new value of Φ is rejected and Δt is adjusted by the factor of 0.0625.

4.8 Chapter Summary

A detailed description of out PIC-MCC models in 1 and 2 dimensions was provided in this chapter. A high level overview of the procedure was described to allow the reader to better understand the component architecture of our models, with each component with the exception of the collisional pusher being described at length. A detailed account of the data structures used in our model was given, and algorithmic details were provided for each component, with the above exception. Some specific limitations due the GPU computing architecture were introduced and overheads associated with their handling as well as additional physical assumptions required for their handling were discussed. Detailed accounts were also given of the moderately complex field solver in both 1 and 2 dimensions, as implemented for the highly parallel architecture of the GPU. The mostly omitted collisional pusher will be discussed in detail in the following chapter.

CHAPTER 5

Particle Pusher - Leap Frog Integration

In the previous chapter we touched on an important part of the PIC-MCC simulation procedure - the need to advance the particle positions and velocities in time. In the collisionless case this is a moderately straight forward integration procedure. Collisions themselves are added through incorporation of the Monte Carlo procedure before, after or during this integration. The collision procedures themselves can be variously resolved at evenly spaced intervals, most commonly the duration of a single timestep, or at irregular intervals generated aroud the free flight time of the particle (see Section 3.5). The point in integration at which to implement collisions has generally been viewed as arbitrary, at least for the electrostatic simulation case[9], however on our closer examination of generalised treatment of collisions, independent of collision frequency, we discovered that this view is not completely accurate. In this chapter we describe the coupling of the leap frog integrator to collision

simulation and some difficulties in preserving accuracy of this scheme.

5.1 Leap Frog Integration Algorithm

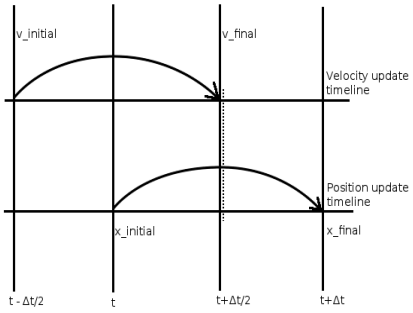
Leap frog integration is a simple yet powerful scheme for velocity and position integration. It is fast and accurate, with low complexity, making it very popular in the field of particle and plasma simulation[5, 9]. In particle simulation it is frequently coupled to the Monte Carlo procedure to characterise inter-particle interactions[32–35]. Generally these collisions tend to be limited to relatively low collision frequencies[36] but alternative approaches focused on handling of higher collision frequencies have been proposed most notably by Hockney *et al.*[9, 37].

In Hockney's approach, a particle subdivided into collision times leading up to the timestep boundary, may undergo several collisions between each solution of the field equations. The global timestep Δt denotes the interval between solutions of the field equation. Particles are advanced across a timestep through a number of intermediate steps, with a Monte Carlo collision being resolved at the end of each sub-step.

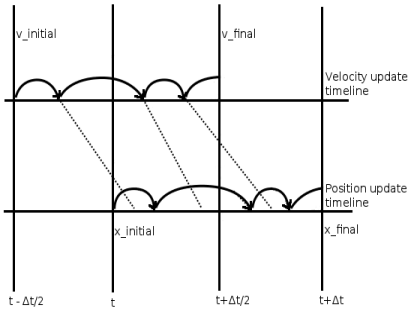
While attractive in its simplicity, it can be shown that the subdivision method results in inaccurate estimation of the velocity and position integration over time[64]. Here we present both this accuracy analysis and the modifications implemented in our push simulation to restore the numerical validity of our scheme.

5.1.1 Classical Leap Frog Implementation

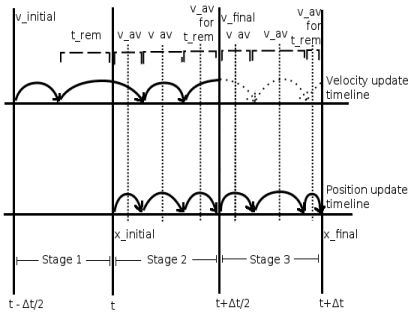
The discrete solution to the equations of motion has been described in Section 3.2.1, with a strategy for numerical implementation given in Section 4.4.



(a) Graphical representation of the leap frog integration method over one timestep. The top horizontal line shows the velocity update timeline while the lower shows the position update timeline. The vertical dotted line projects the velocity value used for the position update onto the position timeline. In the classical case, the final velocity value after velocity update, is the mid-point value, or average velocity for the position update.



(b) Graphical presentation of the velocity projection onto the position timeline for the unmodified leap frog integration method. Each "hop" on the velocity update timeline represents free flight of the particle. As in Figure 5.1(a), we present the projection of the velocity value being used to update the position integration. However, here the projection dotted lines are not vertical, showing that the velocity values being used do not actually correspond to the particle velocity mid-update (i.e. average update velocity). Therefore these velocity values are incorrect for the position update.



(c) Graphical presentation of the velocity projection onto the position timeline for our modified leap frog integration method. Each "hop" on the velocity update timeline represents free flight of the particle. Once again the dotted vertical lines show the velocity value projections onto the position update timeline. Here we see that the projection lines are once again vertical, restoring the accuracy of the leap frog method.

As mentioned in the former, the force value used to determine the velocity update as well as the velocity value used for the position update are the average values for their updates. During the timestep the force being used, and therefore the acceleration is constant. Therefore if no collisions take place during the timestep, the velocity value midway through the timestep is also the average velocity for the update, as shown in Figure 5.1(a). This is the central idea in the leap frog integration, where the recorded position and velocity parameters have a half timestep phase difference introduced between them. Equations 3.1 and 3.2 can then be subscripted in terms of the timestep position as

$$\mathrm{F}_t = \frac{v_{t+\Delta t/2} - v_{t-\Delta t/2}}{\Delta t} \tag{5.1}$$

$$v_{t+\Delta t/2} = \frac{x_{t+\Delta t} - x_t}{\Delta t} \tag{5.2}$$

This is a very elegant implementation of the integration itself. The use of average values provides a high degree of accuracy in comparison to an identical procedure without the phase shift, where the final (as opposed to middle) velocity values would be used for the calculation. No additional memory needs to be used to store velocity values from the previous step and the straight forward implementation of this procedure makes it one of the most popular schemes for particle push.

5.1.2 Collisions

The Monte Carlo procedure for collisions itself is described in Section 3.5. As mentioned, there are two distinct approaches to modelling collisions, one centering around resolving collisions of a calculated probability after a constant particle flight time and the second requiring collision time calculation and allowing for resolution after distinct time period for each particle. In the

former method the collision probability after a constant time interval Δt is given as

$$P = 1 - \exp(\frac{-v\Delta t}{\lambda}) = 1 - \exp(-\nu_c \Delta t), \qquad \nu_c = \frac{v}{\lambda}$$

and can simplify to

$$P = \nu_c \Delta t$$

in the $\nu_c \Delta t \ll 1.0$ limit. Ultimately even the first formulation of the probability calculation is limited to relatively low normalised collision frequencies since the maximum number of collisions being resolved in the interval Δt is one.

At low pressures collision frequency is fairly small and therefore it is not the free flight time but rather electron reaction time to electric field (i.e. as determined from plasma frequency) which dominates the accuracy of the simulation. However at atmospheric pressures, the collision frequency begins to dominate over the plasma frequency and the free flight time starts to constrain the simulation timescale. As a result this type of Monte Carlo collision resolution sometimes chooses a smaller collision sampling time δt than the push timestep Δt and thus carries out Monte Carlo collisions multiple times per timestep.

A more general method of modelling Monte Carlo collisions is presented by the second method of Section 3.5, which calculates the time between individual collisions

$$\delta t = -\frac{\ln R}{\nu_c}$$

This method clearly allows for different intervals and thus does not suffer from the issues of the former method. However conversely we now have to handle sampling flight times that project the particle across the timestep

boundary. Therefore more information, namely the flight time remaining to next collision, has to be stored for each particle.

Multiple collision types can be simulated using the null collision method[58]. In this the maximum total collision frequency is calculated from a given set of collisional cross sections. The free flight times are then generated for this frequency, to account for the "most collisional" scenario. Not all of these collision times result in collisions, of course, since this is the 'worst case' scenario. Instead at collision resolution time we calculate the probability associated with each type of collision. This probability is just the ratio of the actual collision frequency for the process at the current velocity, to the maximum total collision frequency.

Both of the collision methods designed for higher collision frequency handling clearly cannot be decoupled from the push procedure. This is generally implemented through free flight leap frog sub-hops within each timestep, accompanied by a Monte Carlo collision to resolve the velocities at the end of each sub-hop. This is illustrated in Figure 5.1(b).

5.1.3 Leap Frog Modifications

As seen from Figure 5.1(b), the subdivision of the timestep into smaller segments presents a problem for the integrator. In Section 5.1.1 we discussed the positioning of the evaluation of the position and velocity on the timeline of the simulation. Here we outlined the importance of the half-timestep phase difference between the velocity and position value, as well as how, in the collisionless case, this relates to the average velocity over the position update. In this case, as illustrated in Figure 5.1(a), we see that the evaluated velocity position align exactly with the mid-point of the position update when projected onto the timeline.

This is in contrast to the case where the timestep is divided into smaller hops, due to the presence of multiple collisions in one timestep (Figure 5.1(b)). Here we also project the velocity value being used for the update onto the each corresponding position hop. We see that these evaluation times do not agree between the two timelines, with the velocity timeline being treated as if it was shifted forward into the position update timeline. This in effect also invalidates the half timestep phase shift between the velocity and position, as imposed for the accuracy of the integration schemes. Unmodified this results in a significant reduction of the approximation accuracy where the velocity value at $v_{t+\Delta t/2}$ is being treated as final velocity for the total position integration over the timestep. The change in position over the timestep is therefore underestimated, an effect which can be reduced by making the timestep smaller so the continuous integrations are longer within. This however also requires longer runtimes for a set simulation time than the collisionless or weakly collisional probability case. For these reasons more complex integration solvers, parallelisation, and other methods are common[65, 66].

In our work we developed an alternative leap frog approach to remedy this issue. As seen from Figure 5.1(a), during every timestep we need to consider times between $t - \Delta t/2$ and $t + \Delta t$ with 3 distinct stages:

1. Update of velocity parameter from $t - \Delta t/2$ to t, with no update to position.

2. Update of velocity and position from t to $t + \Delta t/2$.

3. Update of position only from $t + \Delta t/2$ to $t + \Delta t$.

Therefore a reasonable sub-partitioning can be implemented by treating each of these stages individually. In the first stage we apply collisions to the ve-

locity parameter, leaving the position parameter unchanged. For the second stage we update the velocity while calculating its average value for the collision sub-timestep and use this parameter to advance the position for the collision sub-step. Lastly we use a local velocity variable to replace the actual velocity variable and use this in the procedure outlined for stage 2, thus avoiding updating of the actual velocity for this part of the integration.

In Figure 5.1(c), the projections of the velocity update onto the position update timeline are illustrated. In the third stage update we include the velocity update projection necessary to integrate the position value to the end of the timestep. We see that the sub-hops now once again align with each other and that average velocity values project directly onto the corresponding times for mid-position update times. This modification clearly restores the update timeline projection characteristic of the leap frog integration method, as shown in Figure 5.1(a).

The pseudo-code outline of the algorithm can be seen below:

```
\* Stage 1 *\
time = t - Δ t/2
while time + t_coll <= t:
   update velocity up to t_coll
   collide velocity
   time += t_coll
   t_coll = new t_coll
t_remainder = t - time \* t_rem in Figure 5.1(c) *\
update velocity up to t_remainder
time += t_remainder       \* time now equals t *\
```

```
\* Stage 2 *\
while time + t_coll <= t + Δ t /2:
   v_old = velocity
   update velocity up to t_coll
   calculate v_average
   collide velocity
   update position with v_average up to t_coll
   time += t_coll
   t_coll = new t_coll
t_remainder = t + Δ t/2 - time \* t_rem in Figure 5.1(c) *\
v_old = velocity
update velocity up to t_remainder
calculate v_average
update position with v_average up to t_remainder
time += t_remainder       \* time now equals t + Δ t/2 *\

\* Stage 3 *\
v_temp = velocity
while time + t_coll <= t + Δ t:
   v_old = v_temp
   update v_temp up to t_coll
   calculate v_average
   collide v_temp
   update position with v_average up to t_coll
   time += t_coll
   t_coll = new t_coll
t_remainder = t + Δ t/2 - time \* t_rem in Figure 5.1(c) *\
```

```
v_old = velocity
update v_temp up to t_remainder
calculate v_average
update position with v_average up to t_remainder
time += t_remainder        \* time now equals t + Δ t *\
```

One important feature of note is that collisions are now effectively resolved over all three stages. This distinction is of importance in stage 3, where the collisions we consider are only simulated for the position advancement of the particles. Therefore in this stage we take care not to create any particles through ionizing processes or destroy any particles through recombination. Without this precaution the ionization or capture reactions would be overestimated by factor 1.5 of the actual value.

Also of importance to note for this approach is the requirement for a greater amount of pseudo-random number generation, as well as the effective discarding of random numbers in Stage 3 without using them to update the actual velocity component. With a good random distribution and high collisions this should not be a problem, however in our code we take the precaution of saving the state of the random number generator at the start of Stage 3 and reverting the generator to this state at the end of the timestep.

5.2 Validation

In Section 5.1 above we outlined the mathematical issue with the segmented and unmodified leap frog integration method. In this section we examine how much of an impact this behaviour has on the numerical results of the integration.

In Sections 5.2.1 and 5.2.2 we examine the motion of a single particle under constant acceleration and under simple harmonic oscillation (SHM) respectively. In these 2 cases we wished to illustrate the issue of dividing the single step into multiple sub-hops and thus we set the sub-hop interval δt to a constant value. This corresponds to the sampling described for the constant interval Monte Carlo probability method in Section 5.1.2. However we did not carry out a collision at the end of a sub-hop, leaving the velocity of the particle unchanged. Thus in principle the correct integration procedure should result respectively in the well known collision-less trajectory of a particle moving linearly at a constant acceleration and a particle harmonically oscillating with time. We further also examined the effects of changing the sub-hop interval δt to illustrate the sensitivity of the calculated trajectories to the degree of fragmentation of the integration timestep. These results are presented in Section 5.2.3.

Finally in Section 5.2.4 we take a look at the effect this issue has on a full Particle-In-Cell simulation. For comparison purposes we implemented three versions of the particle push. Firstly, for comparison purposes we constructed a simple MC-PIC probability pusher with collision interval δt equal to the push timestep Δt. This was compared to the unmodified, fragmented pusher implementing the irregular collision interval, as illustrated in Figure 5.1(b) and our modified leap frog pusher of Figure 5.1(c).

5.2.1 Integration Under Constant Acceleration

As mentioned above, to examine the numerical results of the integration we tracked the variation of position with time for a single particle under constant acceleration. We compared the deviations of the unmodified and the modified leap frog procedure to the analytical solution for the displacement under the

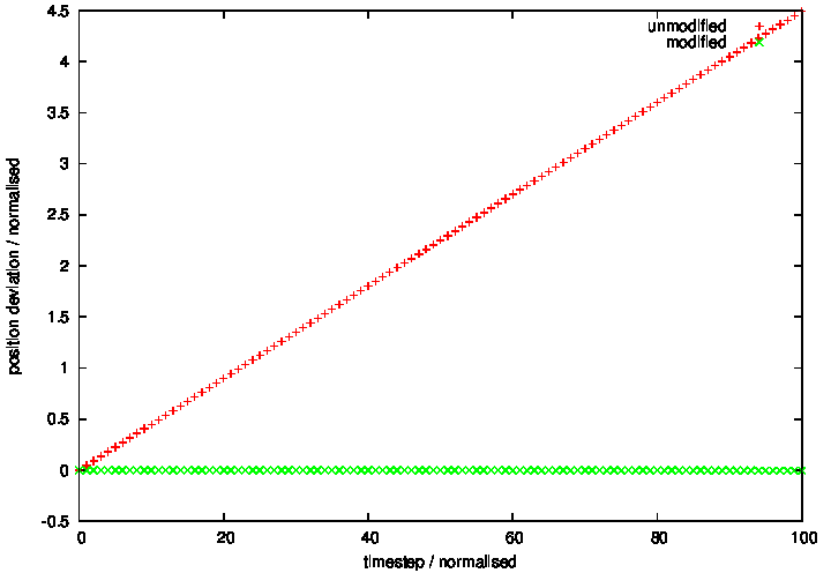


Figure 5.2: *Plot of position deviation from classical leap frog integration for a particle under constant acceleration. The deviation from analytical solution for position is plotted for the unmodified, fragmented leap frog method, and for the modified version. Each method divides the timestep into 10 sub-steps, where sub-division $\delta t = 0.1$.*

set parameters ($\text{position}_{analytical}$ - position_{LF}), and these results are shown in Figure 5.2. The simulation used single precision accuracy and dimensionless units which roughly correspond to the normalised values used in our PIC code, where the cell size and timestep size are 1 and electron thermal velocity is 0.4. The sub-hop interval δt was set as 0.1. The acceleration itself was set at a constant value of 0.1, which was also comparable to the initial acceleration calculated for the sheaths in our PIC model. Therefore this acceleration can be seen as indicative of the conditions experienced in the sheath and the associated electron confinement and ion diffusion.

In the Figure, the unmodified integrator shows a linear, cumulative increase in the deviation from the analytically determined position value. This linearity in the deviation can be attributed to the constant value of the sub-hop interval parameter δt. The value of δt defines the constant amount of fragmentation of each push timestep and under constant acceleration we expect the cummulative position deviation to increase linearly. In contrast the modified pusher deviation value is seen to be negligible, both highlighting the error in estimation due to the unmodified integrator and the validity of our approximation.

5.2.2 Simple Harmonic Motion

A more interesting acceleration case is that of simple harmonic motion. Since particle in SHM is effectively confined to oscillate around a point, this acceleration can illustrate the effects of the timestep fragmentation on the positional confinement of a particle to a particular area. In this analysis we again tracked the motion of a single particle. Much like in the constant acceleration case in Section 5.2.1 we used physical values similar to those of the normalised PIC model. The timestep and cell size were set to 1, with the

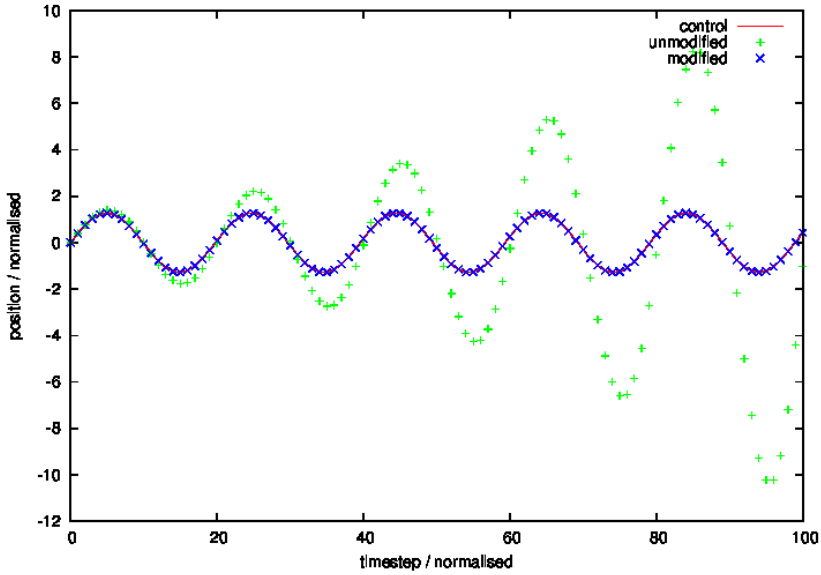


Figure 5.3: *Plot of position trajectories for the control case with no timestep divisions, the unmodified leap frog method with divisions of $\delta t = 0.1$ and the modified leap frog method, presented as a function of time.*

particle velocity corresponding to the normalised electron thermal velocity of 0.4. The sub-hop interval δt was again set to 0.1. The factor k/m, the ratio of the spring constant to the particle mass, is set at 0.1.

Figure 5.3 shows the oscillation trajectory for the unfragmented leap frog position integration (labelled as control), the unmodified fragmented pusher and our modified leap frog pusher. It is immediately apparent that the unmodified pusher oscillation does not remain constant around the equilibrium point. Instead the amplitude of the oscillation increases, showing decay of the containment of the particle around the equilibrium. On closer investigation the period of the oscillation is also seen to increase. These effects are due to underestimation of the motion of the particle over a single timestep leading to underestimation of the position-based determination of the particle acceleration, thus artificially increasing the period and amplitude of the oscillation. This can be catastrophic for a plasma simulation since it is the harmonic oscillator behaviour that determines the motions of the plasma particles, potentially resulting in an instability in the simulation and overestimation of the loss of particles at the boundary. In contrast we see that our modification removes this problem. The particle oscillation amplitude remains constant throughout our advancement and agrees well with the unfragmented calculation of oscillation values.

5.2.3 Effects Of Degree of Leap Frog Fragmentation

In the previous comparison examples we used a constant sub-hop interval δt of 0.1. Since our analysis centers on examining the effects of the fragmentation of the particle pusher we considered it important to take a closer look at the effects of changing the value of δt. We chose to re-examine the deviations observed for the single particle, constant acceleration case presented

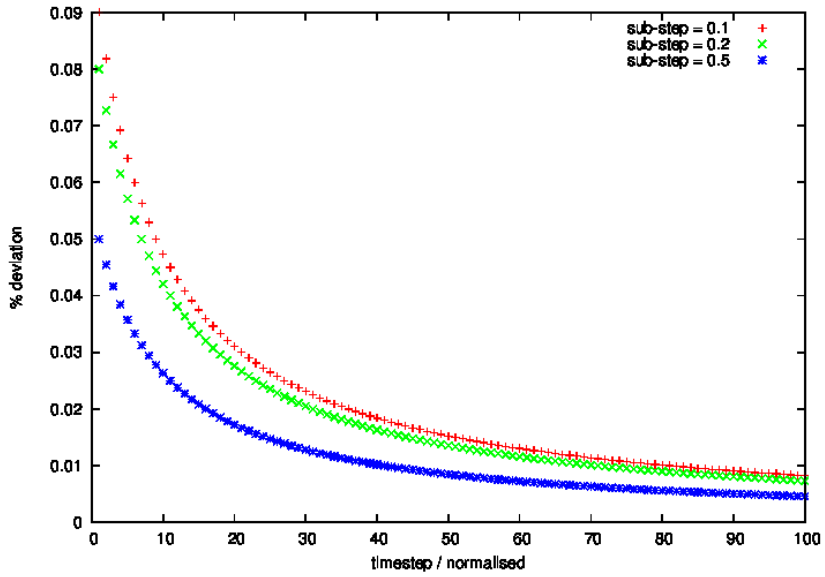


Figure 5.4: *Plot of percentage position deviation of the unmodified pusher from the analytical solution as a function of time for varying number of sub-steps.*

in Section 5.2.1, but this time we used sub-hop interval values, δt, of 0.5, 0.2 and 0.1. The other parameters remained identical to those described in Section 5.2.1.

In Figure 5.4 we plot the percentage deviation from the analytical solution versus the timestep. A trend can be observed corresponding to great deviation from solution with finer sub-stepping, a case interesting to highly collisional systems where more collisions occur and thus more divisions of the position and velocity updates are desirable. The plots show non-linear behaviour since this is a percentage value of deviation, as calculated from the total distance travelled up to that point. In practice, since acceleration remains rarely constant in PIC models, particle will be constantly subjected to the deviations seen in the early timestep regime in this plot.

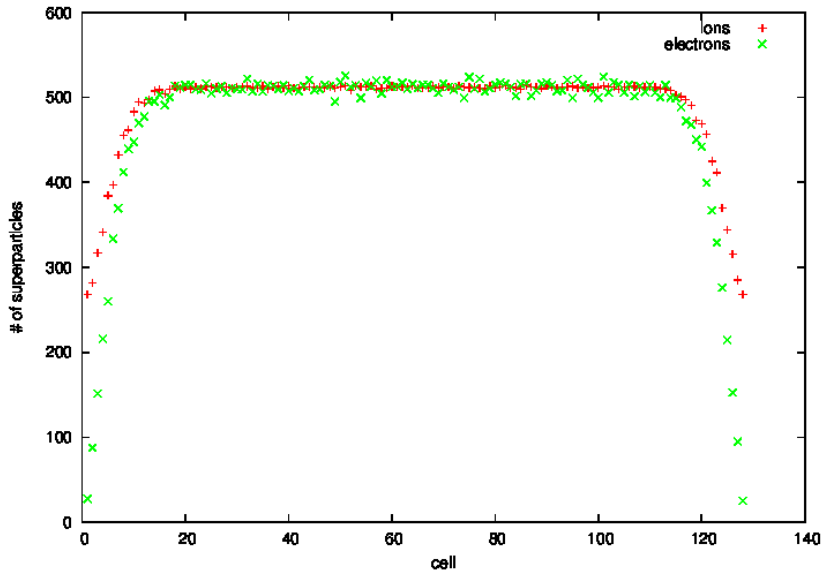


Figure 5.5: *Plot of superparticle densities for collision rate of $\nu_c = 0.01$, estimating collisions by probability, after 25 000 timesteps with $\omega_p \Delta t = 0.2$.*

5.2.4 Effects On Stability Of Simulation

Finally we also examined the effects sub-divisioning of the pusher has on the PIC-MCC simulation. We concentrated on evaluating the charged particle density profile for a simple, weakly collisional case plasma case. The test simulation consisted of a 1D GPU particle-in-cell code with grounded electrode boundary condition. The species simulated consisted of electron and Argon superparticles with plasma density of 10^{15} m^{-3}. The usual numerical parameters from literature[5, 9] of $\omega_p \Delta t = 0.2$ and $\Delta x / \lambda_D = 0.5$ were used, which in turn translated to timesteps of the order of 10^{-10}s and cell width of order 10^{-4}m. The system was simulated for the total cell length of 128 cells with collision frequency $\nu_c = 0.01$ and at 25 000 timesteps data on the density profiles of the system was collected. The system simulated elastic backscatter collisions only.

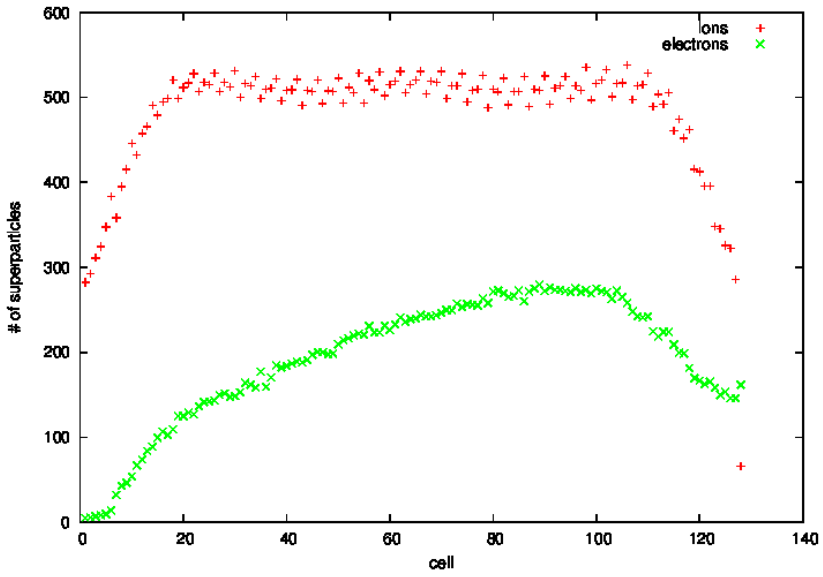


Figure 5.6: *Plot of superparticle densities for collision rate of $\nu_c = 0.01$ using the unmodified leapfrog integrator, after 25 000 timesteps with $\omega_p \Delta t = 0.2$. The electron density profile is clearly skewed towards the right as a result of the unaveraged acceleration of the superparticles and has not reached a steady ion to electron density ratio. The ion density is also more perturbed than in Figures 5.5 and 5.7 due to the particles being effectively more unresponsive to the magnetic field.*

For our expected benchmark case we implemented a simple leap frog pusher with Monte Carlo probability collisions resolved at the end of the timestep. This case clearly did not contain subdivision of the leap frog integrator and at low collision frequencies is expected to agree well with the more direct Monte Carlo collision interval calculation (see Section 5.1.2). The density profile of this pusher is shown in Figure 5.5.

The fragmentation of the pusher, on the other hand, was achieved by implementing a collisional particle pusher using the direct sub-hop interval calculation, as described in Section 5.1.2. In the unmodified case we do a direct leap frog advancement for each sub-hop as was schematically described

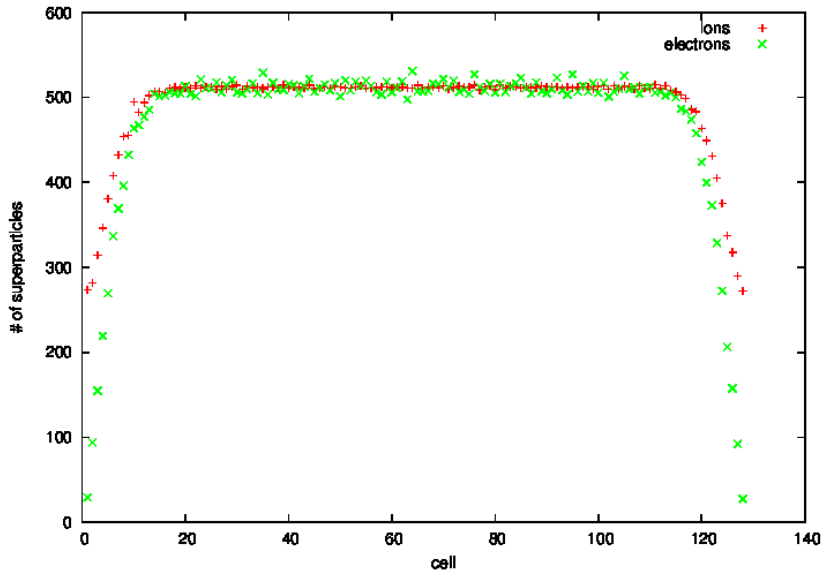


Figure 5.7: *Plot of superparticle densities for collision rate of $\nu_c = 0.01$ implementing the modifications outlined in Section 5.1.3, after 25 000 timesteps with $\omega_p \Delta t = 0.2$. The behaviour of the probability simulation with timesteps of Δt is restored and the simulation gives the expected quasineutrality and symmetricity.*

in Figure 5.1(b). The resultant density profile for this case can be seen in Figure 5.6.

Finally we modified the pusher as described in Section 5.1.3 to correctly simulate the variable collision interval Monte Carlo method. This modification provides the pusher with greatest flexibility in valid collision frequency values, since there is no maximum limit on the number of resolved collisions per timestep. The density profile of this implementation is shown in Figure 5.7.

From the density plots of the three simulations it is readily apparent that for commonly accepted PIC simulation parameters the unmodified pusher implementation is unstable. Insufficient electron confinement is observed at

the boundary, resulting in no establishment of quasineutrality in the bulk plasma. Since the Poisson equation solution resembles the simple harmonic motion case examined in Section 5.2.2 above, this is believed to be due to the increasing overestimation of the particle motion and the decay of the oscillation observed in that case. This leads to very little electron trapping at the plasma boundary. From the examination of the effect of increased substepping, as shown in Figure 5.4, it is expected that to achieve any reasonable numerical stability the timestep value would need to be reduced significantly, adding to the computational overhead.

This effect gives some explanation why meaningful results were achievable in previous simulation works on the subject. Taking the case of Hockney's work[9, 37], on which ours is based, the timestep cited is smaller than the one used in our simulation. In addition, with a sufficiently high frequency of velocity reducing collisions the displacement of a particle over the timestep is lower than for a less collisional model. Therefore with less position and velocity change over the timestep the discrepancies between the expected and calculated results will require a longer runtime to become apparent.

5.3 Chapter Summary

In this chapter we presented a detailed outline of the collisional particle pusher used in our procedure. We outlined the requirements on our pusher in light of application to highly collisional simulation regimes and described our resultant choice of direct collision-time Monte Carlo pusher. Problems with the naive implementation of this pusher were demonstrated, both conceptually and through simulation and an alternative pusher was developed to correct for these issues. This alternative implementation was described

both graphically and through pseudo-code. The correction to the procedure through our modifications was demonstrated through simulation of base acceleration cases. A comparison of the effects on simulation of this issue when uncorrected were shown through 1D grounded boundary condition density profiles.

CHAPTER 6

Benchmarking And Verification

In Chapters 3 and 4 we outlined the algorithm used in Particle-in-Cell modelling with Monte Carlo collisions. In these simulations we are clearly modelling a very complex system, requiring resolution of multiple different physical features and processes. As such this simulation presents us with the challenge of validating our simulation as physically meaningful after combining all these complex computational elements into a whole.

The need for comprehensive benchmarking and verification procedures for scientific simulations has been periodically highlighted since 1990s [67, 68]. Issues were raised about the validity of a number of widely accepted and professionally maintained codes after the demonstration of a number of errors in these simulations. Due to these effects calls were made for more rigorous validation and verification procedures in scientific simulation software [67–69]. In the plasma community these calls were until recently only addressed

sporadically, predominantly in the swarm physics community [70–75] or in works on positive column development [76]. However in 2013 Turner *et al.*[8] proposed a set of benchmarks for low pressure discharge models to address this concern. The benchmarks presented in that work were also used to validate our PIC-MCC plasma model and results of this analysis are outlined below.

6.1 Benchmark Parameters Outline

The benchmark conditions used to validate our model were determined based on experimental set up utilised by Godyak *et al.*[77]. As argued by Turner, Godyak's experiments are well-characterised and reproducible, thus enabling a relative ease of further future experimental validation. These benchmarks are also quite similar to Turner's earlier benchmark set, as determined from the experimental work of Surendra[78]. Four benchmark cases were outlined in Turner's work [8]. Of these, benchmark number 4 is significantly more time-consuming to carry out in practice than the other cases due to the excessively long runtime of this case. Therefore due to significant constraints on the time available for the completion of this work only the first 3 cases were applied to our simulation. Turner's complete benchmarks are summarised in Table 6.1.

The chemistry used in this simple model is that of a Helium plasma, with singly charged ions and electrons being the actively modelled species. In the case of electron superparticles the simulation handles ionization, elastic momentum transfer collisions and two excitation reactions between the electrons and neutrals. The elastic collisions are assumed to be isotropically

Table 6.1: *Benchmark parameters for the verification of our PIC-MCC model*

case:			1	2	3	4
Electrode separation	L	(10^{-2} m)			6.7	
Neutral density	N	(10^{20} m^{-3})	9.64	32.1	96.4	321
Neutral temperature	T_n	(K)			300	
Electron temperature	T_e	(K)			30 000	
Ion temperature	T_i	(K)			300	
Applied frequency	f	(10^6 Hz)			13.56	
Applied voltage	V	(V)	450	200	150	120
Electron mass	m_e	(10^{-31} kg)			9.109	
Ion mass	m_i	(10^{-27} kg)			6.67	
Plasma density	n_0	(10^{14} m^{-3})	2.56	5.12	5.12	3.84
Particles per cell	N_c		512	256	128	64
Cell size	Δx	(m)	L/128	L/256	L/512	L/512
Time step size	Δt	(s)	$(400\text{f})^{-1}$	$(800\text{f})^{-1}$	$(1600\text{f})^{-1}$	$(3200\text{f})^{-1}$
Steps to execute	N_S		512 000	4 096 000	8 192 000	49 152 000
Steps to average	N_A		12 800	25 600	51 200	102 400

scattered in the centre of mass frame of reference. The ion-neutral collisions also model isotropic elastic collision reactions but in addition they introduce an anisotropic back-scatter elastic collision component[79]. Cross sections used for determining the collision frequency were obtained from the Biagi v7.1 set available from the LxCat cross section repository[80]. For most of the reactions of interest in our benchmarks these were compiled from experimental data but for the back-scatter elastic collisions of ions with neutrals, cross sections can be calculated from analytic expressions[81]. The neutral gas density and temperature were approximated to be constant during the simulation.

The driving potential is applied as a sinusoidal function of frequency 13.56 MHz for each of the four cases. As discussed by Turner [8] the peak voltage for each case was selected to give an approximately constant current density amplitude of 10 Am^{-2} between the cases. The plasma in our model is confined between two planar electrodes.

6.2 1D Model Verification

General benchmarks applied to our model were described above. The four cases outlined in Turner's work[8] were however applied to slightly differently characterised PIC simulation procedures than the one presented in our work. Therefore we expect that while our results are not likely to be identical, if a good agreement is achieved this provides not only a verification for our model but a further support for the effectiveness of the selected benchmark parameters in validating these types of plasma models.

The simulation model differences between our PIC-MCC model and the ones outlined in Turner's paper are further addressed in Section 6.2.1. These

are followed by the actual results of our benchmarking cases in Section 6.2.2. Here we compare our results to those presented by Turner *et al.* and discuss agreements and deviations between the models and the different benchmark cases.

6.2.1 Comparison Of Our Simulation Techniques To Benchmark Models

As outlined in Chapter 4 from the beginning our PIC-MCC design was developed with GPU execution in mind. In Section 4.2.4 we mentioned the need to allocate global memory for the particle array on the host since these types of allocations are not available on the device. Therefore the extra space available in the memory particle array is fixed for the duration of the particle push procedure.

Since our collision processes are resolved in the particle push procedure, this presents a problem for the ionization processes. Ionizing electron collisions not only clearly generate additional electrons but they also create ion product partners from the neutral feed gas. In the benchmark cases from Turner *et al.*[8] these newly created particles are added automatically during the ionization collision procedure since this is generally implemented on the CPU, where re-allocations can be handled automatically. However, as we discussed in Section 4.6, to be able to create new particles through ionization during the collision procedure itself we would have to be able guarantee that sufficient memory has been allocated in all collision product arrays to accommodate these new particles.

This issue is further compounded by our choice of direct collision resolution Monte Carlo procedure. Where the benchmark cases used the simple probability method, outlined in Equation 3.16 of Section 3.5, our model was

designed for a more generalised handling of collision frequencies, calculating the times to next collision instead. The types of collisions were resolved using the null collision method[58] at the end of each free flight interval. While this effectively removes the accuracy constraint of low collision frequencies, as imposed on the benchmark cases, from our own simulation, this generic handling also removes any guarantee of the maximum number of ionization collisions experienced by a particle within a timestep. Where the benchmark cases guarantee at most one particle creation collision per timestep, our model can experience an unspecified number of such collisions. Therefore even were it practical to allocate enough space in the particle data arrays to accommodate every possible ionization (in the simple benchmark case this would double our memory space requirement for the particle arrays), our model does not give us any guarantees of what this maximum number would be.

As discussed in Section 4.6 we instead chose to record the number of particles created in each cell for each superparticle specie. This information was then used to add the new particles after the particle push for the timestep. The ionization collision kinetics are still resolved during the particle push procedure however unlike in the case of the benchmarks, all the remaining energy after ionization is deposited into the original colliding electron. The new particles are then generated with energies normally distributed around the set species temperature.

Finally it should be noted that Turner's choice of benchmark cases was determined through selecting four plasma operational pressure points approximately spanning the range of convenient values for their simulation model limits. At the lower end this was determined from limits of discharge sustainability through ionization, which agrees with the limits of our simulation. However the upper limit was determined from their Monte Carlo

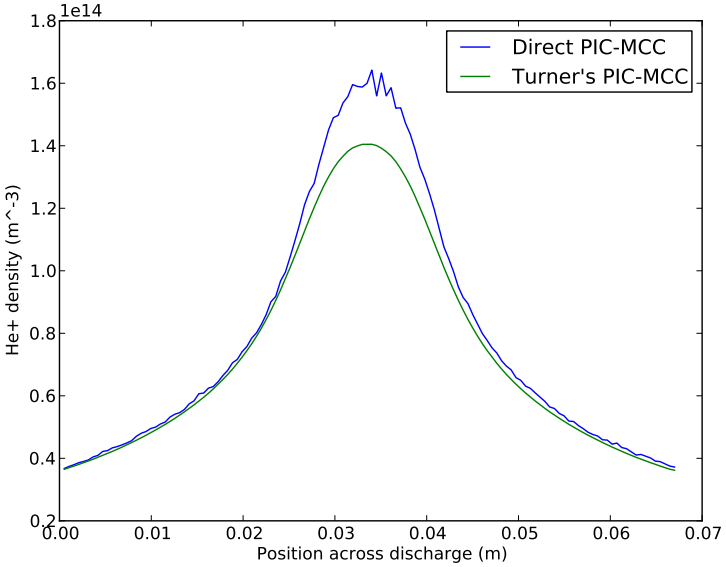


Figure 6.1: *Time averaged ion density profile for benchmark case 1. Both our direct PIC-MCC simulation result and that of Turner's benchmark PIC-MCC implementation (labeled as implementation E in his original paper[8] are shown.*

collision procedure requirement of $\nu_c \Delta t \ll 1.0$, which is not necessary for our direct Monte Carlo procedure.

6.2.2 Simulation Results

In Figure 6.1 we show the comparison between ion density profiles obtained from our direct PIC-MCC and from Turner's benchmark model using the first benchmarking case. Overall there is significant qualitative and quantitative agreement between the two density profiles. However in the centre of the discharge we observe a deviation of our model from the accepted expected profile, with out model slightly overestimating the ion density. As discussed in Section 6.2.1, the expected reason for this is the difference in ionization

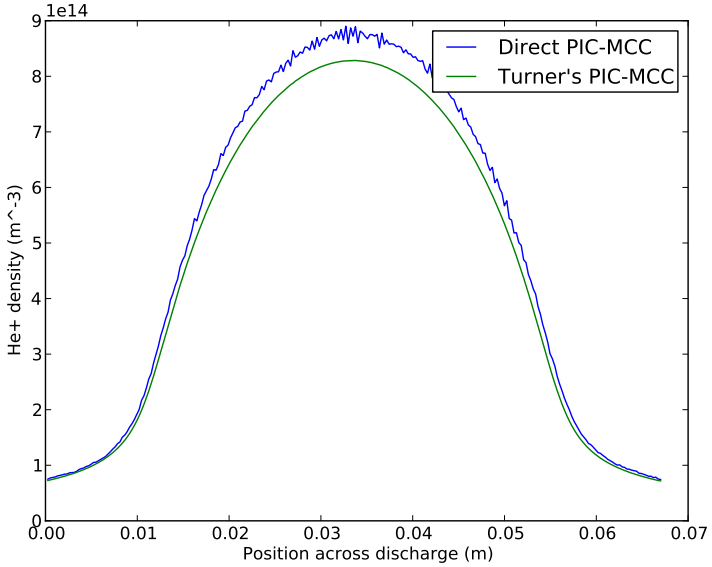


Figure 6.2: *Time averaged ion density profile for benchmark case 2. Both our direct PIC-MCC simulation result and that of Turner's benchmark PIC-MCC implementation (labeled as implementation E in his original paper[8] are shown.*

collision handling between the two models.

The ionization profile for benchmark case 2 is shown in Figure 6.2. Here, similarly as in the case 1 benchmark from Figure 6.1, we observe mostly good agreement between the two models. However in the centre of the discharge once again our direct model results in slightly higher ion density than would be expected. The ion density observed in the second benchmark case is larger than that of case 1 and as a result the absolute value of the ion density difference between our model and the benchmark model is larger in the second case. The percentage ion density deviation value however is seen to reduce, showing a clear improvement in agreement between the two models at the higher pressures of the case 2 benchmark.

Benchmark case 3 ion density profile is illustrated in Figure 6.3. As

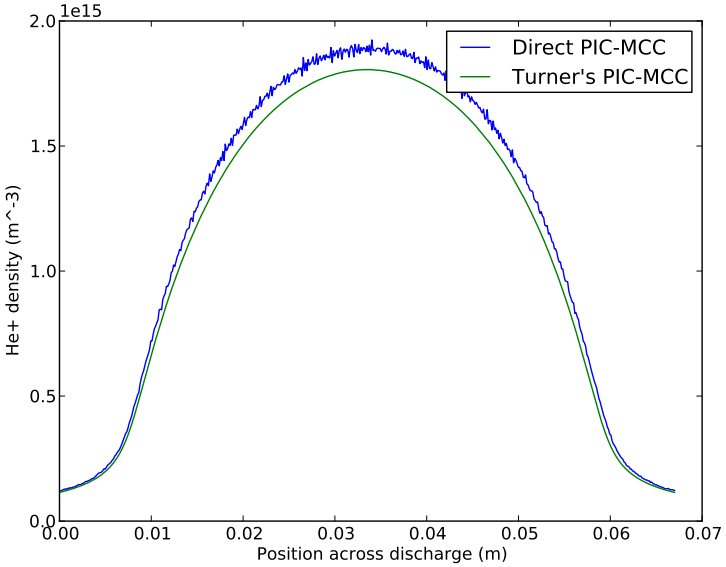


Figure 6.3: *Time averaged ion density profile for benchmark case 3. Both our direct PIC-MCC simulation result and that of Turner's benchmark PIC-MCC implementation (labeled as implementation E in his original paper[8] are shown.*

observed in the previous cases, here too we see a good agreement with the benchmark case in general, with a slight deviation from the benchmark ion density in the plasma bulk. Once again, as expected from a higher pressure case, the overall ion density has increased in comparison to the previous two cases. We also see a further decrease in the percentage deviation of the two densities in the bulk of the modelled discharge, resulting in closer agreement of our calculated ion density to the benchmark for case 3 than for the two previous cases.

Of note in all of these cases is a slight asymmetry in the the density profiles around the centre. This is believed to be a numeric effect produced by the pseudo-random number generator (RNG). While not well documented in peer reviewed literature, other research groups have raised the issue as prevalent in PIC-MCC models using standard C/C++ random number generator. This is also the generator used as base for the Nvidia CUDA RNG employed in our model. The asymmetry is reported to disappear with the change of RNG to an alternative such as the Mersenne Twister.

Likewise of note are the oscillatory artifacts observed at the peaks of our density profiles. These are most apparent in case 1 but can be discerned in the other cases. The origin of these oscillations is not well understood at the moment. It is possible that they are the result of our using our modified Monte Carlo collision procedure. The new collision procedure is expected to provide greater perturbation to the system than the conventional implementation, which resolves all collisions at a well-defined time. In addition, our alternative Monte Carlo collision implementation presents a greater overhead on the RNG, with free flight times being determined from the uniform distribution. Therefore any issues with the RNG distribution, such as the one discussed for the asymmetry case above are expected to affect our collision

procedure to a greater extent than the traditional collision implementation.

6.3 Preliminary 2D Model Verification

Unfortunately at the time of the writing comprehensive benchmarks for 2D simulations were unavailable. 2 dimensional simulations are comparatively time-consuming, with long runtimes due to the non-linear increase in problem domain on scaling from 1 dimensional case to the 2 dimensions, as well as the need for a significantly more complex field solver in 2 dimensions. As noted in the reasoning for omitting benchmark case 4 from our current results, in obtaining physical benchmark data we were under some time constraints with respect to measurements and as such proposed preliminary 2D PIC-MCC model benchmarks rather than a full suite of measurements.

In these preliminary measurements we collapse the 2 dimensional model into 1 dimension. This allows us to confidently use Turner's benchmarks in measurements of the 2D model and reduce the runtimes to approximately those of the 1D case. The majority of the procedures within our model remain the same and thus get tested for correctness. The exception of note is the 2 dimensional field solver. With the field solver we confined ourselves to re-using the parallel cyclic reduction solver (PCR) for solving the potential in 1D and separately testing the DADI solver with respect to the analytical solution for the case. The technique of projecting the 2D model onto 1D is discussed in Section 6.3.1, with the simulation results shown in Section 6.3.2.

6.3.1 2D Model Projection To 1 Dimension

In our projection procedure we adopted two philosophies:

(i) Where possible use all existing code with addition of boundary han-

dling.

(ii) If (i) is not feasible use existing code/solver with problematic section omitted.

In practice point (i) was sufficient for most of the code except the 2D field solver. To implement these changes we defined a global parameter ONE_DIM, to be defined in our constants for 1D execution. At critical, dimension-specific portions our model checks whether this parameter is defined and takes appropriate action accordingly. Critical sections included for instance disabling particle move in y-directions during push and correct scaling of the acceleration and charge distribution, as determined from superparticle position. For each of these changes only the scaling parameter for y-direction had to be adjusted. An advantage of this approach was that since our particles no longer moved in the y-direction, particle sorting for 2D simulation could be applied in its completeness while still returning correct results.

As mentioned, the 2 dimensional DADI field solver is not easily reduced to 1D without removal of a bulk of its procedure. Therefore we decided instead to re-purpose the PCR solver, used by the DADI solver, to solve the tridiagonal formulation of the Poisson equation in 1 dimension. Since the PCR solver is instrumental to the DADI solver itself, correct performance of this procedure provides a significant partial verification of the DADI solver itself. For the verification of the correctness of the DADI solver overall we confined ourselves to calculating the solution to the constant charge density 2D problem and comparing it to the analytical solution. These results are presented at the end of Section 6.3.2.

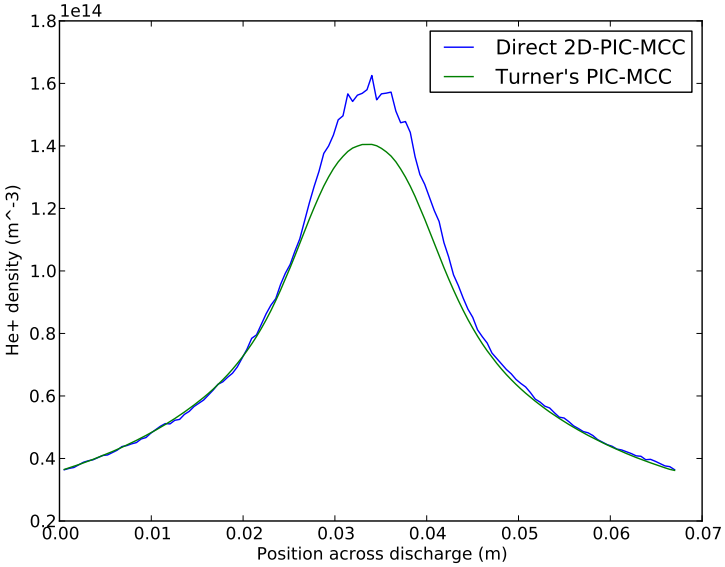


Figure 6.4: *Time averaged ion density profile for benchmark case 1 applied to the collapsed 2D simulation. Both our direct PIC-MCC simulation result and that of Turner's benchmark PIC-MCC implementation (labeled as implementation E in his original paper[8] are shown.*

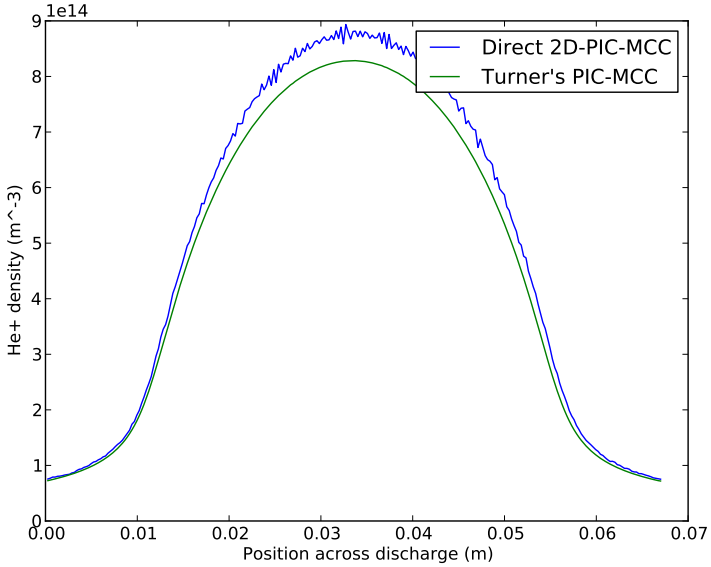


Figure 6.5: *Time averaged ion density profile for benchmark case 2 applied to the collapsed 2D simulation. Both our direct PIC-MCC simulation result and that of Turner's benchmark PIC-MCC implementation (labeled as implementation E in his original paper[8] are shown.*

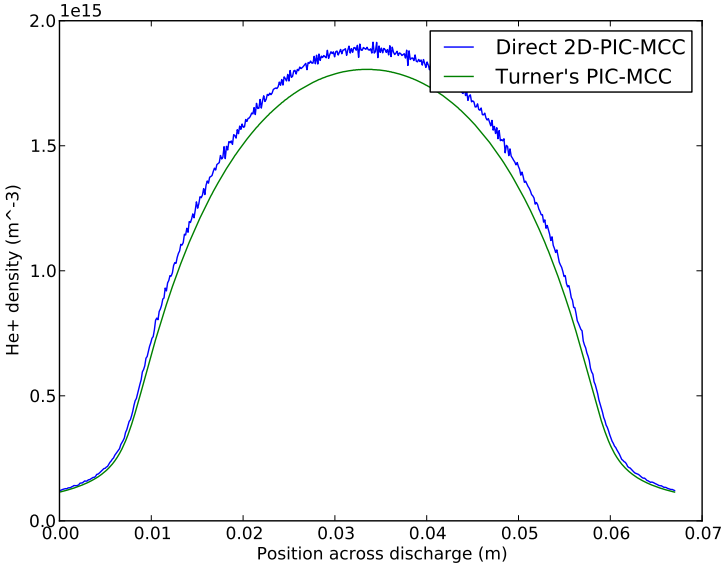


Figure 6.6: *Time averaged ion density profile for benchmark case 3 applied to the collapsed 2D simulation. Both our direct PIC-MCC simulation result and that of Turner's benchmark PIC-MCC implementation (labeled as implementation E in his original paper[8] are shown.*

6.3.2 Simulation Results

The ion density profile results for cases 1, 2 and 3 are shown in Figures 6.4, 6.5 and 6.6 respectively. As with the 1 dimensional model benchmark plots, we also include the profiles for the same benchmarks obtained from the work of Turner *et al.*[8]. It is immediately apparent that the 2D collapsed model profiles strongly agree with the profiles from the 1D model analysis, demonstrating a strong self-consistency between the 2 implementations.

It follows that similarly to the 1 dimensional results, in the 2D case we also observe that while our cases show fairly good agreement with the benchmark, the ion density tends to be higher for our model than for the benchmark cases.

This effect reduces in significance as we move to the higher pressure regimes. Once again we believe the discrepancy is due to the alternative handling of energy splitting during ionization implemented in our model as well as the use of distributions for initialising the ionization particle energies. As discussed in Section 6.2.1 above, this results in energy not being conserved during these collisions and can thus explain this deviation from the expected benchmark.

DADI Solver

An analytical solution to the 2 dimensional Poisson equation can be calculated for the special case of constant charge density. Using the normalised form of the Poisson equation, where $-\frac{\rho}{\epsilon_0} = f(x, y) = 1$, our equation takes the form

$$\Delta\Phi = 1 \tag{6.1}$$

which is of the form

$$\Delta u = \lambda u$$

where λ is an eigenvalue of matrix u. Let us denote eigenfunction corresponding to eigenvalue λ_k as Λ_k. We can thus express function f as

$$f = \sum_{n=1}^{\infty} F_n \Lambda_n \tag{6.2}$$

This is effectively an expression for Fourier series with coefficients F_n being calculated in the usual way as

$$F = \frac{4}{ab} \int_0^b \int_0^a f(x, y) \Lambda dx dy \tag{6.3}$$

Similarly, we expect u to be expressible as a series of terms of the eigenfunctions

$$u = \sum_{n=1}^{\infty} U_n \Lambda_n \tag{6.4}$$

Substituting these results into the Poisson equation above, along with the fact $\Delta\Lambda_n = -\lambda_n\Lambda_n$ we obtain the result

$$\sum_{n=1}^{\infty} -\lambda_n U_n \Lambda_n = \sum_{n=1}^{\infty} F_n \Lambda_n$$

where the coefficients U_n are given as

$$U_n = -\frac{F_n}{\lambda_n} \tag{6.5}$$

When this procedure is applied to our test Equation 6.1 above we can easily show that the eigenfunctions are $\sin(\frac{m\pi}{l}x)\sin(\frac{n\pi}{l}y)$ for a square problem domain of length l. The eigenvalues for these eigenfunctions are $\frac{\pi^2}{l^2}(m^2+n^2)$. Coefficients F then become

$$F_{m,n} = \frac{4}{\pi^2}\frac{1}{mn}\Big[1-(-1)^m\Big]\Big[1-(-1)^n\Big] \tag{6.6}$$

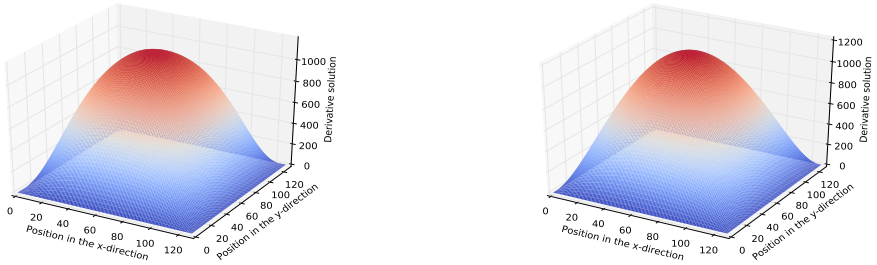
and using Equation 6.5 we find the coefficients of u as

$$U_{mn} = -\frac{4}{\pi^2}\frac{l^2}{\pi^2}\frac{1}{mn(m^2+n^2)}\Big[1-(-1)^m\Big]\Big[1-(-1)^n\Big] \tag{6.7}$$

Thus the analytical solution for u $(= \Phi)$ up to terms $m = n = 3$ is given as

$$u = \Phi = -\frac{8}{\pi^2}\frac{l^2}{\pi^2}\Big[\sin\Big(\frac{\pi}{l}x\Big)\sin\Big(\frac{\pi}{l}y\Big) + \frac{1}{15}\sin\Big(\frac{3\pi}{l}x\Big)\sin\Big(\frac{\pi}{l}y\Big) + \frac{1}{15}\sin\Big(\frac{\pi}{l}x\Big)\sin\Big(\frac{3\pi}{l}y\Big) + \frac{1}{81}\sin\Big(\frac{3\pi}{l}x\Big)\sin\Big(\frac{3\pi}{l}y\Big)\Big] \tag{6.8}$$

The solution of the analytical formulation of the Poisson equation (Equation 6.8) and the DADI procedure results are shown in Figures 6.7(a) and 6.7(b) respectively. In these cases we set the problem size to $l = 128$, with $\Delta x = \Delta y = 1$. The ratio of charge density to permittivity of free space is $\frac{\rho}{\epsilon_0} = 1$, introducing a factor of -1 to the solution in Equation 6.8 (as seen from calculation of coefficients F, where f = -1). The maximum value calculated from the analytical solution is 1182.8 while the maximum reached by our



(a) Potential profile calculated from the analytical solution.

(b) Potential profile obtained from our DADI solver.

Figure 6.7: *Plots of solution to the Poisson equation for the special case of constant charge density.*

DADI implementation is 1205.2. This is clearly a good agreement between the two solutions, an agreement that can be further fine-tuned by reducing the tolerance value used by the DADI solver to determine convergence, which was set at a fairly relaxed value for this test case.

6.4 Chapter Summary

In this chapter we have outlined the benchmarks used to validate our model for physical correctness. We have discussed the benchmark choice of Turner *et al.*[8] as well as the experimental data used to generate the benchmark parameters. We have presented our 1 dimensional model's results for these benchmarks, compared them to results seen in literature and discussed sources of discrepancy between our model and the benchmark cases. Finally we have presented preliminary benchmarks for the 2D model under 1D conditions, with separate verification results for the 2D field solver and compared these

results to both our 1D model and the literature benchmark.

CHAPTER 7

Performance

In this thesis we have outlined the architecture design and implementation of the 1 and 2 dimensional PIC-MCC procedure for arbitrary collision frequencies, as designed for massively parallel accelerator devices such as the GPU. With this focus it is important to examine the computational performance and scalability of our models. As is the case with an overwhelming majority of numerical models, the PIC simulation procedure is not fully scalable. Some of the bottlenecks for the parallelism were discussed in greater detail in Chapter 4 and will be summarised in individual sections below.

For characterising parallel scaling of our models we have measured their strong and weak scaling characteristics. These provide a standard performance measure in the parallel computing community and reader is referred to Appendix A for a more detailed description of their conditions. In addition since in the majority of the parallel scaling cases there is a large gradient

between the thread number extremes we have decided to reproduce the raw performance measurements in Appendix B.

The 1 dimensional PIC-MCC model scaling is outlined in Section 7.1. This includes both the parallel scaling analysis, where we examine how the base model scales onto the GPU hardware and a simple scaling measurement of the model with neutral gas density, to examine the performances expected at higher pressures and thus in effect higher computational intensities. These same measurements are reproduced for the 2D model in Section 7.2. Finally due to the complexity of its implementation we also separately examine the parallel scaling of the 2D field solver and present these results in Section 7.3.

All the measurements presented in this chapter were carried out on a single Dell Precision T5500 Workstation. The workstation hosted one Nvidia® GeForce® GTX 760 graphics card with 4GB of GDDR5 memory. The host system consisted of two Intel® Xeon® X5650 CPUs and 6144 MB of DDR3 memory, however throughout our study we have endeavoured to minimise the host portions of the code and therefore these were not scaled for our performance results.

7.1 1 Dimensional Model

The test parameters for 1D PIC-MCC model scaling were derived based on the benchmark ranges presented in Table 6.1 in Chapter 6. The total system length was 6.7 cm, with a grid cell total of 512. The peak of the applied sinusoidal voltage was 150V. This signal was of 13.56MHz frequency (f), with timestep of $(800f)^{-1}$. The initial plasma density was chosen as 5.12×10^{14} m^{-3} and each cell was initialised with 64 superparticles. Since these measurements were focused on measuring the computational performance of the

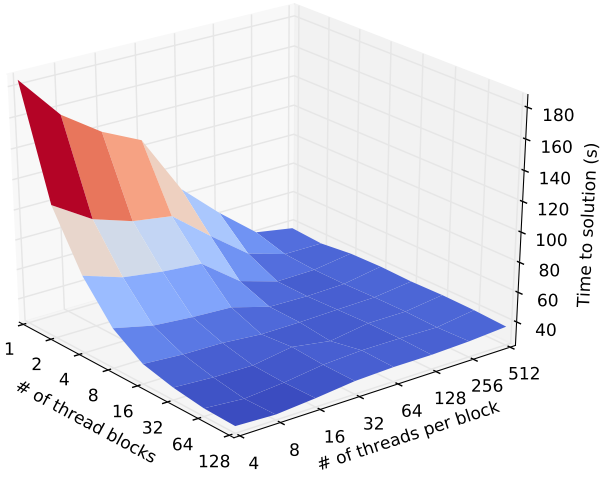


Figure 7.1: *Strong scaling of the 1D PIC-MCC model.*

model rather than its physical accuracy, the system was simulated for 1000 timesteps per measurement rather than until steady state is observed.

7.1.1 Parallel Scaling

The parallel scaling measurements were carried out under a constant neutral gas density of 9.64×10^{20} m^{-3}. Of interest in these measurements was not only performance scaling of the model with increasing total number of threads but also the affect of different thread-block configurations on the performance. The total number of threads in a simulation run is found from the product of the number of threads per block and the number of blocks in a simulation run.

Strong Scaling

The strong scaling results for the 1D PIC-MCC model are shown in Figure 7.1. As is expected, a stark performance increase is seen as we increase the total number of threads available to the execution. The minimum number thread grid configuration for our measurement is 1 block $\times$ 4 threads, with maximum scaling to 128 blocks $\times$ 512 threads. Clearly in a system of 512 cells, this maximum significantly exceeds the maximum parallelisation possible for our model at this systems size. Instead, for block/thread configurations roughly exceeding the maximum parallelisation of the problem we obtain some measure of the overheads associated with creating idle threads and blocks during our execution.

As can be seen from Figure 7.1 performance increases until roughly each cell is serviced by an individual thread. This is not a linear increase even for the power-of-two axes used in this figure, showing that performance increase is most notable at lower thread values. This behaviour is similar to scaling profiles noticed for other real-world irregular numerical applications on transitions to massively parallel regimes[82, 83]. Severity of the non-linearity of these types of scalings are largely hardware architecture-dependant. Most high-end hardware is designed and optimised for a standard set of high performance numerical benchmarks called LINPACK[84]. This standard benchmark looks at the performance of a given hardware system in solving a system of linear equations in a general dense matrix for a selection of different sizes. While a powerful solver in itself, as discussed by Flynn *et al.*[85] the simple linear system solution is a very regular application, in as far as the locality of data in memory is concerned, as well as memory accesses associated with the solution. Therefore it is a poor reflection of a realistic numerical application which is generally irregular in data locality and memory accesses and

therefore cannot make any of these guarantees.

Another important feature to note is that the application does not scale symmetrically for threads and blocks. In fact our application scales more efficiently with additions of further thread blocks over increase in the number of threads per block. This effect is most likely hardware-specific to GPU architecture implementation, as well as our specific card type. As mentioned above, our measurements were carried out on a GeForce GTX 760 GPU. According to its release notes[86], this card contains 1152 cores, divided among 6 streaming multiprocessors. Cores executing on a given multiprocessor all share data cache[3]. The block/thread software space translates onto this hardware as follows:

1. A block of threads is assigned to a single multiprocessor.

2. The thread block executes its threads on the cores available on the multiprocessor. These threads share the multiprocessor data cache.

3. Threads are executed in warps of 32. While multiple warps run concurrently if cores are available, each warp is treated as a distinct portion of the computation.

4. Within a warp, any conditional if statements are serialised during execution.

5. If a multiprocessor does not have resources available for all the warps of a given block, the idle warps have to wait until threads become available.

Since the entire (complex) particle push procedure takes place on the GPU it is clear that our PIC-MCC model presents a significant memory

and instruction overhead per thread. Therefore while many-thread execution optimises the core utilisation for computation, the memory and instruction requirements of this configuration for a single multiprocessor data cache presents an unacceptable overhead for our performance. The data no longer fits in the fast cache and instead the execution suffers from cache misses and associated memory loading overheads. In addition our collisional Monte Carlo implementation clearly suffers from conditional divergence as well. Reducing the number of threads per block also reduces the chance of a divergence within the threads of a particular block. In fact the peak performance was observed for 8 threads, 64 blocks configuration for our test case, however the optimal configuration is expected to vary for model systems of different grid sizes and superparticle densities.

Weak Scaling

The weak scaling of our PIC-MCC model is presented in Figure 7.2. In this measurement the computational load per thread is kept constant as the number of threads scales upwards. In an ideal case this scaling would show constant performance independent of thread/block configuration. As discussed in Appendix A, this measurement was concerned with replicating these parameters and thus modifications to the computation were carried out to reflect the computation load rather than physical accuracy on these runs. Crucially, the total grid size of the system becomes the product of threads per block and blocks, rather than the specified 512 cells from the strong scaling measurements.

As we see the scaling shows a fairly constant profile until the upper end of system size, where the run time starts increasing significantly. This cut-off is seen at a few thousand cells size and can be attributed to the serial

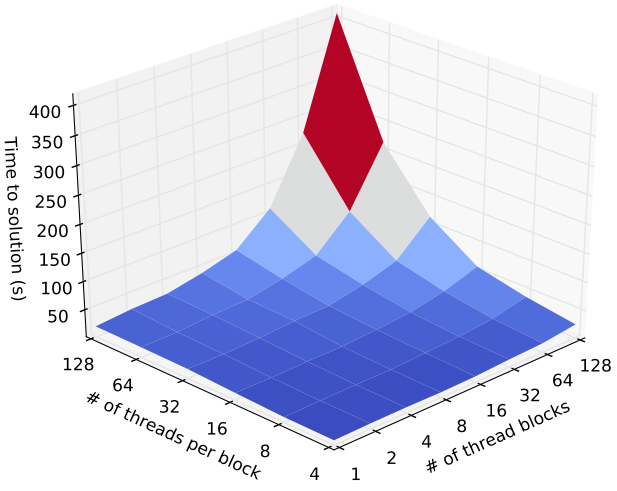


Figure 7.2: *Weak scaling of the 1D PIC-MCC model.*

portions of the code (such as memory space checks) beginning to dominate the execution time for the constrained load parallel sections. These portions are generally proportional to the simulation grid size and are predominantly necessitated due to memory allocation limitations during device execution (see Chapter 4).

It should also be noted that in this case we once again notice an asymmetry in the scaling of threads and blocks. We see that the scaling along threads performs worse than scaling along blocks. This effect can once again be attributed to the combination of data cache saturation and thread divergence serialisation, as was discussed for the strong scaling case.

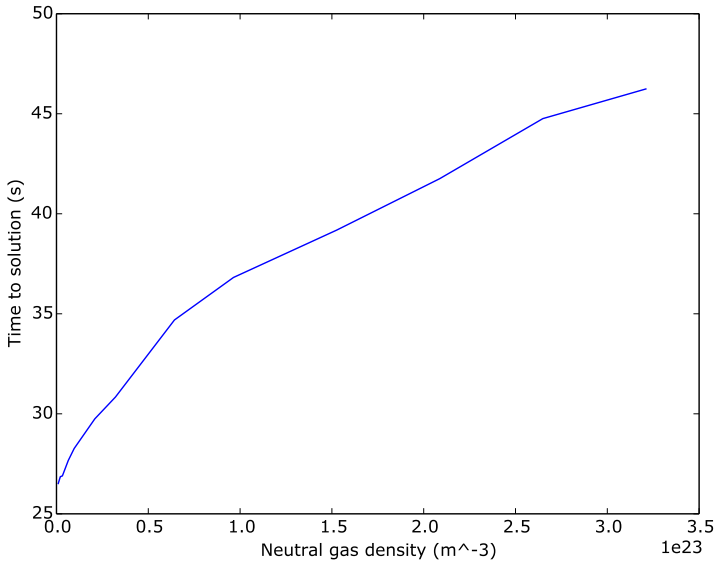


Figure 7.3: *Scaling of the 1D PIC-MCC model with pressure of the neutral feed gas.*

7.1.2 Scaling With Pressure

Since our model was designed to scale freely to highly collisional regimes we have decided to provide some preliminary measurement for the performance scaling of the model with varying neutral gas density. As discussed in Section 3.5, the null collision frequency is proportional to the neutral density, which itself is a function of gas pressure under controlled temperature conditions (which is a reasonable approximation of the case of non-LTE plasmas). Unfortunately since our modified pusher was developed during the design of our PIC-MCC model, at the time of the writing an alternative generic implementation was unavailable for comparison. Therefore our discussion has to be confined to qualitative trends associated with high-pressure PIC-MCC simulations rather than quantitative ones.

The performance for scaling with neutrals density is shown in Figure 7.3. We observe a steeper gradient in the time increase during the lower density regime, with the curve levelling off somewhat around density of 6.0×10^{22} m^{-3}. The total density range over which measurements were taken extends from 9.64×10^{20} m^{-3} to 3.21×10^{23} m^{-3}. Needless to say this is a fairly huge range corresponding to pressures from 3.99 Pa to 1.33×10^{3} Pa respectively. As can be seen, within this range we observe only a relatively small change in the runtime per 1000 timesteps, with the runtime difference on the smaller gradient portion of the graph (range 6.0×10^{22} m^{-3} - 3.21×10^{23} m^{-3}) being 10 seconds.

The change in the gradient observed in the plot above is hard to diagnose due to some uncertainty introduced into the absolute values of these measurements due to other simulation processes running on the card simultaneously (these effects can be better observed in some of the figures below). However two effects that are likely to contribute to this discrepancy are load balancing between threads and divergence between warps.

In lower collisional regimes, a collision taking place near the end of the timestep can cause a significant delay in execution as the other threads idly wait for this single execution to terminate. As the collision probability increases, the likelihood of multiple threads experiencing a collision near the end of the timestep increases, and so computation becomes more balanced than previously, with a more even load spread between threads. Thus the performance is expected to begin in a state of good load balance at very low collisions, transition into a poorly load balanced regime as the collision frequency increases and return to better load balance after some critical value of collision frequency.

As discussed in the strong scaling section above, any conditional opera-

tions within a thread warp get serialised. Therefore the load balancing issue described in the paragraph above also affects the performance in this fashion. At low collision frequencies where only a relatively small total number of collisions take place per timestep, this divergence between executions (i.e. whether a collision takes place) can be fairly apparent. As the simulation progresses to a higher collision regime, this situation becomes less divergent since statistically all threads start to experience collisions and the difference in the number of collisions effectively only impacts the latter part of the push. Therefore the more collisional case results in smaller divergence between the threads in a given warp, and thus only a smaller section of the push execution has to be serialised.

7.2 2 Dimensional Model

As discussed in the case of the 1D model scaling the focus of this analysis was the measurement of computational performance rather than physical validation. Therefore the test measurements for the 2D PIC-MCC model were carried out over 100 timesteps, rather than over a physical solution range. The 2D model is more computationally complex than the 1D implementation, mainly due to the more complex field solver, and therefore it was decided a more accurate approach to performance benchmarking would be to examine the 2D PIC-MCC model separately to its field solver. Therefore for these measurements the field solver was switched off. Measurements in this section only represent the scaling of the kinetic parts of the solution, sorting, charge accumulation and acceleration calculation (albeit set to zero due to the potential being set at this value). The scaling of the field solver is covered separately in Section 7.3.

The test parameters used for this measurement were also fairly similar to the 1D model performance benchmark. The discharge boundary was a square of side length 6.7 cm, divided over a cell grid of 128×128. The timestep was kept at $(800\times13.56\text{MHz})^{-1}$ and the initial plasma density was 5.12×10^{14} m^{-3}, to be consistent with the 1D simulation parameters. Each cell was initialised with 100 superparticles.

7.2.1 Parallel Scaling

The neutral gas density was once again kept constant for this part of the performance analysis, at the value of 9.64×10^{20} m^{-3}. As was the case in Section 7.1.1, multiple thread and block configurations were examined to provide us with not only information on scaling as a function of total number of threads but also as a function of the block/thread GPU hierarchy. However, in our 2D implementation we now use a 2 dimensional grid of blocks and threads. Therefore in this section our axes are presented in terms of the 2 dimensional thread and block configurations specified for the execution.

Strong Scaling

Figure 7.4 shows the strong scaling of the 2D PIC-MCC model over 100 timesteps. It is readily apparent that the scaling profile of the 2D model demonstrates many similarities with that of the 1D model. As was the case for the latter, at the upper scale of total thread values we exceed the maximum parallelisation values for our 128×128 cell grid model. In addition our maximum number of threads per block for which measurements were obtained (16×16, or total 256 threads per block) is constrained by the hardware limit on the total supported number of threads per block, as dependent on the compute capability of the GPU.

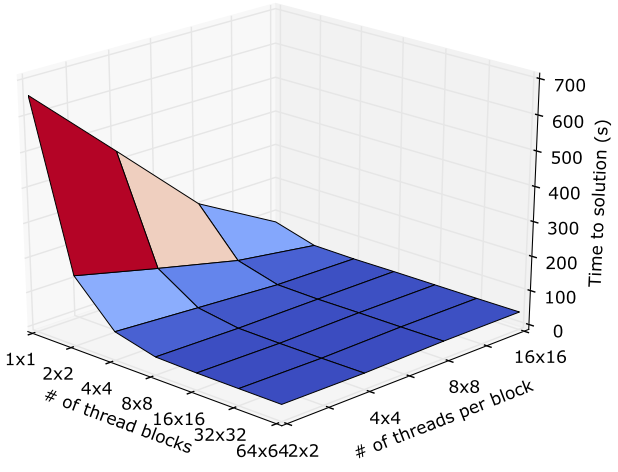


Figure 7.4: *Strong scaling of the 2D PIC-MCC model. In this test case the field solver was switched off. This was due to the significant computational complexity of the field solver and the computational overhead it represents, thus allowing us to focus on the PIC procedure itself. The field solver scaling is examined in Section 7.3.*

As described for the 1 dimensional case, here too we see a non-linear scaling of the application with addition of threads. Much like the 1 dimensional mode, the 2 dimensional problem is an irregular one and therefore it, too, suffers from non-optimal memory access patterns, which negatively impact its memory handling performance. In addition there is once again an asymmetry in scaling with blocks versus scaling with threads. The reader is referred to Appendix B for a clearer characterisation of this effect. As is the case for the 1D PIC-MCC model, the 2D model also presents a significant memory overhead as well as larger thread divergence, which in the case of low block numbers results in a large memory/instruction space being assigned to a small number of data caches as well as greater serialisation of warp threads. Data cache saturation increases the occurrence of cache misses for a set of processing threads and creates an overhead due to loading of memory pages. Therefore performance improvement is observed for configurations with smaller numbers of threads per block and larger numbers of blocks. In fact peak performance for our test case was observed for 32×32 blocks and 4×4 threads configuration, however once again these values are dependent on the specific test case parameters.

Weak Scaling

The 2 dimensional PIC-MCC model weak scaling is shown in Figure 7.5. As discussed in Section 7.2, here we also switched off the field solver and instead concentrated on benchmarking the kinetic framework of the system. The adjustments required to our code to better replicate the weak scaling conditions are discussed briefly in Appendices A and B. As was the case for the 1D model weak scaling, since load is being kept constant per thread, our total grid size effectively becomes a function of the total number of threads

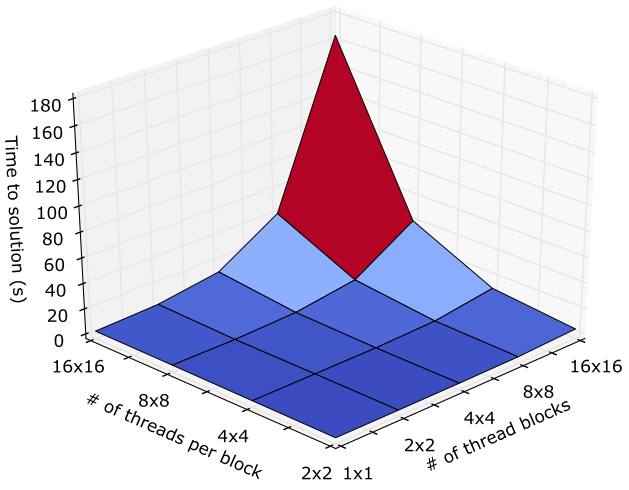


Figure 7.5: *Weak scaling of the 2D PIC-MCC model. In this test case the field solver was switched off. This was due to the significant computational complexity of the field solver and the computational overhead it represents, thus allowing us to focus on the PIC procedure itself. The field solver scaling is examined in Section 7.3.*

in the system. Therefore any serial portions of the code are expected to show scaling performance decrease.

In the 2D case we too observe the effects of the overhead associated with the serial portions of the code on the overall performance as we progress to higher grid sizes. However at the lower end of the grid spectrum we see a fairly consistent scaling profile, which presents an encouraging result for our parallelisation model. In addition we once again observe the asymmetry in scaling between thread-dominant and block dominant parallelisation, as seen more clearly from raw data in Appendix B. As in the previous cases this observation can be attributed to data cache saturation and warp thread divergence experienced at higher thread numbers.

7.2.2 Scaling With Pressure

In characterising the neutrals density scaling of the system we used similar parameters to those used for the parallel scaling characterisation. In this case we also disabled the field solver to better resolve the performance of the kinetic portions of this code. While the absence of the field solver and applied potential will affect the ionization rate in the system, since our collisions are modelled using the null collision method[58], the computation associated with resolving collisions remains characteristic of the full system. Once again the total density range over which measurements were taken extends from 9.64×10^{20} m^{-3} to 3.21×10^{23} m^{-3}, corresponding to 3.99 Pa and 1.33×10^{3} Pa respectively.

Figure 7.6 shows the neutrals density scaling profile for our 2D model. In this case we observe a broadly similar trend to that of the 1D pressure scaling, with an overall performance change between 5.0×10^{22} and 3.21×10^{23} showing a fairly smooth profile and being around 12 seconds. The profile at

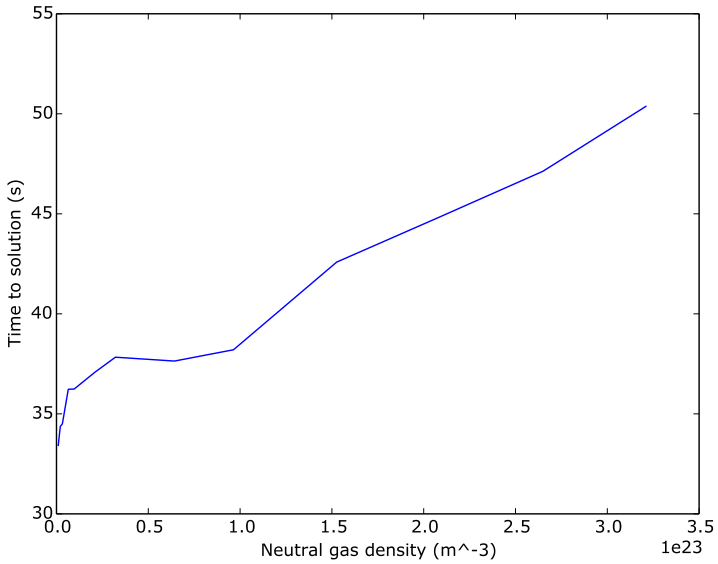


Figure 7.6: *Scaling of the 2D PIC-MCC model with pressure of the neutral feed gas. In this test case the field solver was switched off. This was due to the significant computational complexity of the field solver and the computational overhead it represents, thus allowing us to focus on the PIC procedure itself. The field solver scaling is examined in Section 7.3.*

the lower end of the spectrum is not as well characterised, as is apparent from the figure. This portion was problematic to measure due to other simulation running on the test card during these measurements as well as background tasks such as driving of display introducing an uncertainty into our results. However we do see some signs of a similar drop off in the computation time, as seen in the 1D case, which would correspond to the transition across the poorly load balanced regime, as discussed in Section 7.1.2.

7.3 2 Dimensional Field Solver

A very important portion of our 2D PIC-MCC implementation is the field solver. As discussed in Chapter 4 for this we wrote a dynamic alternating direction implicit solver (DADI) for the GPU. This solver in turn requires an efficient linear system solver and since we desire for this to be as parallelisable as possible, we chose the GPU parallel cyclic reduction solver of Zhang *et al.*[7] with some generalisations to allow for the handling of more physical systems. The background to both these implementations is discussed in Section 4.7.2. As a result in this section we present the parallel scaling of both the PCR solver and the DADI solver with constant PCR solver parallelisation values.

7.3.1 Strong Scaling

The DADI strong scaling profile was generated for a 128×128 system, with cell width and input function values normalised to 1. These scaling results are shown in Figure 7.7. In contrast the PCR solver scaling was measured over a 512×512 grid with the Poisson equation in 1 dimension being the test system of equation for each of the 512 systems and the source term $\frac{\rho}{\epsilon_0} = 1$.

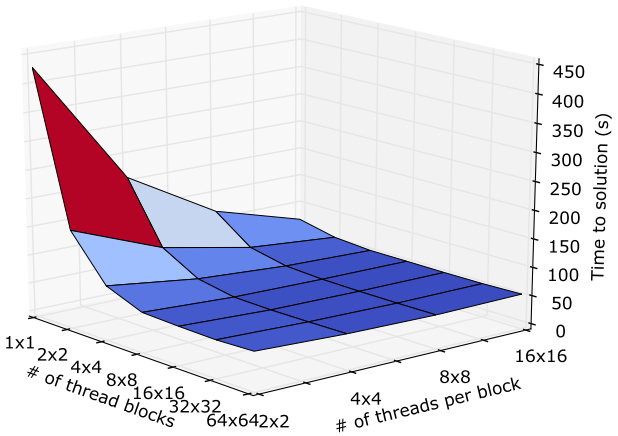


Figure 7.7: *Strong scaling of the DADI field solver. The parallel cyclic reduction solver, employed by the DADI solver, was not being scaled but instead supplied with constant values for blocks and threads. PCR scaling is addressed separately below.*

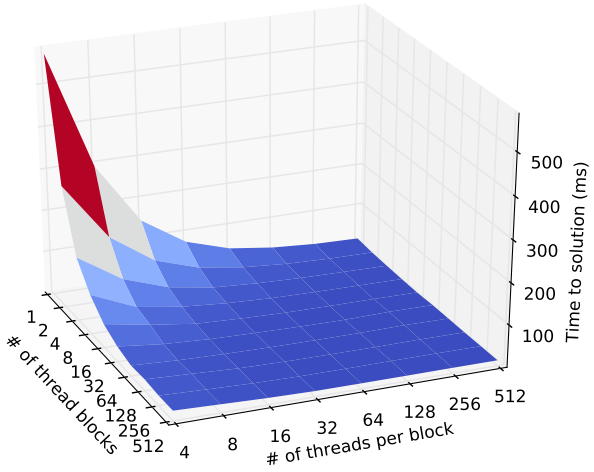


Figure 7.8: *Strong scaling of the parallel cyclic reduction solver (PCR).*

These strong scaling results are shown in Figure 7.8. The choice of a smaller domain for the DADI scaling tests was mainly due to the excessive execution times associated with convergence calculations for the low threading cases in this solver.

Both these cases show a broadly similar scaling characteristic, with a non-linear scaling profile associated with our previous application measurements. Of interest to note is that unlike the PIC-MCC models examined above, in the PCR case we see a reverse asymmetry profile between thread and block scaling in comparison to the latter cases. In the PCR case we see that performance improves more with addition of threads rather than blocks to the execution system, as shown more clearly in the raw data in Appendix B. In the DADI scaling overall we see an improvement of scaling with addition of blocks, as was the case in the PIC-MCC scaling in 1 and 2 dimensions.

As discussed in Section 7.1.1, the optimal execution on GPU takes place when non-divergent full warps of threads access contiguous portions of memory, carry out a reasonably straightforward calculation on the data set and return new values to contiguous portions of memory. The PCR solver fits this computation model very well and therefore we see optimal scaling profile for this case, with larger thread numbers filling warps more fully, optimal memory access patterns and moderately small requirements on storage in data cache. On the other hand the DADI solver is a much more complex numerical solver, with a parallel GPU transpose function (irregular) and a coefficient recalculation GPU function (also irregular) as well as the convergence procedure. Therefore this solver does not fit the hardware-optimised model of computation for the device and we see a different scaling pattern asymmetry than in the PCR case.

7.3.2 Weak Scaling

The weak scaling profiles for the DADI and the PCR solver are shown in Figures 7.9 and 7.10 respectively. While the DADI solver shows a profile much more reminiscent of those seen for the two PIC-MCC models, the PCR solver shows a very constant weak scaling performance over the measurement range. As was discussed in Section 7.3.1 above, the PCR solver is expected to show a much more optimal scaling profile since the GPU hardware itself is designed with solving these types of problems in mind. Indeed in the ideal case weak scaling performance is expected to be relatively constant over a large thread range.

On the other hand we see that as we progress to higher thread regimes the DADI solver performance suffers from greater overheads. These are both due to the serialised portions associated with the convergence solver as well

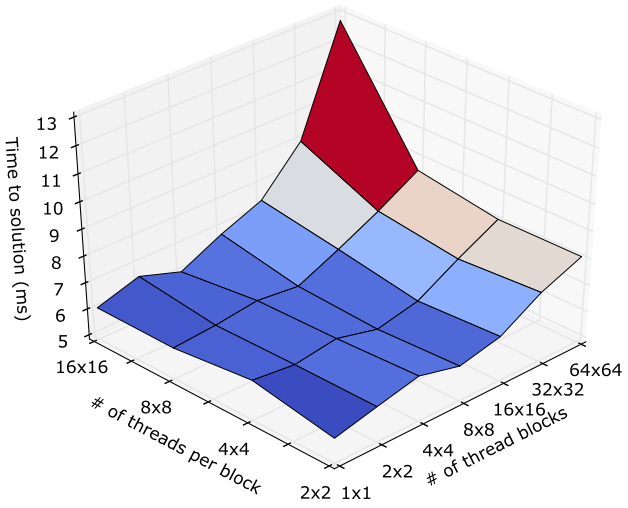


Figure 7.9: *Weak scaling of the DADI solver.*

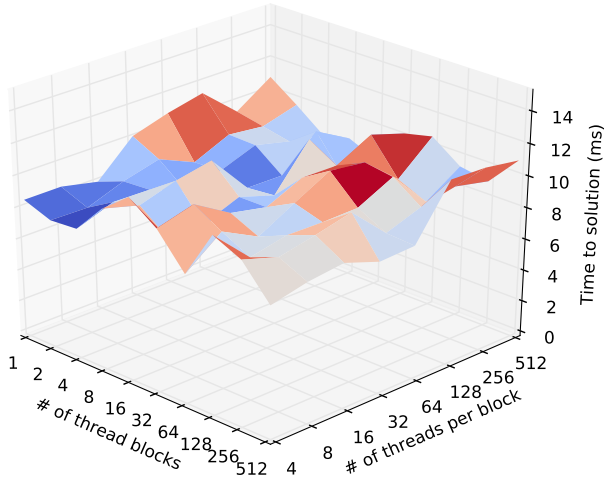


Figure 7.10: *Weak scaling of the parallel cyclic reduction solver (PCR).*

as the computation execution patterns not fitting the hardware-optimised computation pattern as well as the PCR solver. Therefore the DADI weak scaling profile is much more characteristic of the profiles expected from irregular real-world applications such as the 1 and 2 dimensional PIC-MCC models.

7.4 Chapter Summary

In this Chapter we examined the parallel scaling of our PIC-MCC models and their significant constituents. We have discussed at some length the computing models optimised for at hardware level and the performance challenges they present for irregular applications consistent with system models characteristic of real world problems, such as the PIC-MCC model. We have also discussed the hardware utilisation expected from real-world applications, allowing for more optimised choice of computation hardware in the future. Finally by presenting a computation component considered representative of optimal GPU hardware utilisation we have illustrated the divergence of the manufacturer-expected computation model from the realistic computation model.

CHAPTER 8

Conclusions And Future Work

8.1 Current Outcomes

In this work we have presented the research methodology and outcomes obtained when adjusting the existing Particle-In-Cell (PIC) modelling techniques to the high performance computational regime on the GPU and the highly collisional physical regime at atmospheric pressures. In the process we have also discussed and presented some measurements benchmarks to accompany our architectural studies and allow free reproduction of our results.

In Chapters 1 and 3 we have discussed the basic physical characterisations of industrial plasmas and the practical implementation requirements of collisional Particle-In-Cell modelling respectively. In these chapters we not only concentrated on outlining the physics to be characterised, we have also discussed the computational hardware used to implement these models and

the practical constraints it presents on our computational implementation. We also showed the numerical constraint imposed on the model by the characterisations formerly adopted by the plasma modelling community and the restrictions these impose on the validity of the simulation.

In Chapter 4 we presented in detail the architecture of 1 and 2 dimensional PIC-MCC models as developed for the computational constraints presented by the GPU. We have explicitly detailed our data structures and discussed the memory allocation difficulties presented by the GPU capabilities and the physical model requirements. We have developed consistent handling of memory allocations for non-particle-conserving model. Using Mertmann *et al.*[6] sorting procedure we exploited the data localisation to optimise the charge accumulation procedure. Using shared memory we designed a convenient weighting procedure to allow for better particle cloud characterisation of the charge in each cell.

In addition we also developed a fully GPU-utilising stand alone field solver for both 1 and 2 dimensional PIC procedures. Of particular interest is the 2 dimensional dynamic alternating direction implicit (DADI) solver. Due to our focus on physically meaningful utilisation of this solver we generalised the designs presented in literature[7, 62] to robustly handle more general systems. This resulted in our removing some of the computational constraints on the system size formerly imposed by these solvers.

In our attempt at extending our PIC-MCC model to highly collisional physical regimes consistent with atmospheric pressures, we developed an alternative particle pusher to facilitate for more direct simulation of Monte Carlo (MC) collisions. We showed the numerical inconsistencies inherent in the naive simulation of irregularly occurring Monte Carlo collisions within the leap frog pusher procedure and developed a technique to correct for these

issues. We have also presented numerical accuracy measurements for the unmodified leap frog pusher, naive collisional implementation pusher and our modified collision pusher under null collisions, and illustrated how the naive implementation diverges from the expected values. At the same time we have shown our modifications restore the numerical accuracy of the scheme for irregularly time-resolved MC collisions. In addition we also presented the effects the naive implementation of the pusher fragmentation has on the model stability, in particular the electron confinement in the bulk plasma. The restoration of the expected properties under our modified pusher regime was demonstrated. Finally these modifications in effect removed the simulation timestep constraints imposed by the classical probabilistic MC collisions procedure, while preserving the simulation accuracy, thus significantly increasing the absolute size of the timestep permissible for a self consistent simulation at higher gas pressures. These results were presented in Chapter 5.

With the development of our alternative, direct MC collisional procedure as well as the overall computationally complex PIC simulation design we recognised the need for extensive physical verifications. Therefore our PIC-MCC procedure was used in conjunction with a set of well-characterised physical simulation benchmarks at low pressure and our particle density profiles were compared to those obtained from multiple distinct models developed independently by different members of the plasma physics community. With our model we were able to obtain good agreement to other simulations of the given physical cases as well as highlight some quantitative differences seen between our implementation and that of other PIC-MCC models. Our work on physical verifications was discussed in Chapter 6.

Finally in Chapter 7 computational performance measurements of our

GPU PIC-MCC model were presented. Good parallel scaling was shown for both our PIC-MCC models in 1 and 2 dimensions and for the 2 dimensional field solver. Due to the complexity of the 2 dimensional model, to show more detailed performance analysis, the PIC-MCC model and the field solver were analysed separately. In this chapter we also showed the overheads accrued from serialised sections present in the PIC-MCC models due to the GPU computational constraints and presented their scalings over problem sizes. Scaling with gas pressure was measured to better predict the performance of our models on scaling to atmospheric pressure regimes.

In addition in this chapter we also discussed a number of hardware design choices in the GPU design and their affects on our scaling profiles. We discussed the application formulation expectations in the GPU hardware design and the realities of the applications seen in practice. Solvers within our PIC-MCC models consistent with these best-case applications were identified and their scaling profiles were compared to those of the more realistic, irregular portions of our applications. Scaling profile features of the models were identified and explained in terms of the GPU hardware design features.

Overall the aim of our study was to develop a two-fold extensive design and analysis of the PIC-MCC model capable of characterising a plasma at atmospheric pressures and specifically designed for the GPU hardware execution. In this work we have presented the detailed design of our GPU PIC-MCC model, at both 1 and 2 dimensions. The adjustments of our modelling procedures to account for arbitrary collision frequencies were presented, as were the computational architecture design choices necessitated by the GPU. The resultant models were subjected to a number of verification procedures to determine their physical accuracy as well as an extensive scaling analysis to characterise their performance on the given execution hardware.

8.2 Future Work

As discussed in Chapter 6, due to the comparatively long runtimes associated with the 2 dimensional PIC simulation we have only so far presented preliminary verifications of our 2D model. While these preliminary results are still valuable, it would be desirable to implement a suite of full 2D benchmarks, similar to those developed for the 1D PIC-MCC model. This is a complex challenge however, particularly since unlike for the 1D case, no systematic benchmarking effort has been as yet introduced for the 2D case. Therefore for this we require a wider collaboration with other groups and their independently developed 2D models, as was the case in the 1D verification benchmarks development.

The runtime lengths for certain 1D model conditions and virtually all realistic 2D model conditions present an additional obstacle. In our current formulation of the models, we assume that our computation does not get interrupted by a reboot or a system crash. As the runtime of a simulation is extended this guarantee is hard to facilitate due to random events such as brief grid outages or OS service crashes. Therefore in practice it is difficult to ensure uptime for the entire duration of the computation. To better handle these random real events we wish to introduce a checkpoint procedure into our model. While the writes to disk associated with a checkpoint procedure would present a performance overhead on the runtime, the checkpoint itself would provide a much more graceful recovery option after a crash for our models.

While a lot of energy has been dedicated to verification, it would be of interest to validate our 1 and 2 dimensional models directly against the experimental measurements of Godyak *et al.*[77, 87]. These experimental works were not only used to determine the verification parameters in Chapter 6,

there is also some preliminary evidence[88] that the current and plasma density profiles with voltage are not fully modelled by the standard 1 dimensional PIC models. Therefore it would be interesting to see what, if any effects transitioning to 2D simulation as well as the introduction of our alternate collisional pusher would have on these results.

Finally as discussed in Chapter 7, our performance measurements show strong signs of profile features associated with irregular applications. As such two interesting performance avenues present themselves. Firstly it would be interesting to examine the performances of CPU codes accelerated using only a subset of the GPU procedures developed for our model. This would allow for some of the GPU design specific overheads to be removed while also providing massive parallelism for performance-critical sections of a CPU model. Secondly it would be of great interest to see our design translated onto the Intel® Xeon® Phi architecture, which provides a more flexible alternative to the GPU. It is our expectation that some of the overheads due to memory allocation management would be unnecessary on the Xeon® Phi, thus potentially having significant effects on the performance of our base design on this device.

Appendices

APPENDIX A

Parallel Scaling Benchmarks

The subject of parallel scaling is quite extensively discussed in literature, with widely varying opinions and options of which approach best characterises the parallel performance of an algorithm or a computer system[89–91]. Indeed parallel scaling of an algorithm frequently varies drastically based on the hardware available for execution, a feature we have endeavoured to stress in the outline of our model and the justification for our architectural choices.

For basic parallel scaling measurements we have confined ourselves to the industry standard of strong and weak scaling measurements[92], which are outlined in more detail below. The general purpose of these measurements is to examine the scaling of the system as more threads become available as well as the degree of parallelism achieved in the examined code and the overhead presented on the execution device due to background overheads such as thread creation and non-locality of data in memory.

A.1 Strong Scaling

The so-called strong scaling is a very powerful, yet conceptually simple measurement. It provides a very accurate measure of the scaling of a code or a section of a code with addition of threads. In this approach the size of the system is kept constant, irrespective of the number of threads available for its execution. A process is considered to square linearly if the speedup observed is linearly proportional to the number of threads utilised by the computation. The efficiency of this scaling can be calculated as

$$Eff = \frac{t_1}{Nt_N} \times 100\% \tag{A.1}$$

where t_1 is the time taken to execute the entire test load by 1 thread, N is the number of threads and t_N is the time taken to execute the test load by these threads.

In practice in simulation of physical models, linear scaling is unlikely. Due to the complexity of these models, the system is usually divided into execution blocks for individual threads along logically convenient boundaries (such as cells in our model). It is usually very difficult to divide the load in this type of problem evenly among the available threads and thus load balancing can become an issue. In addition this type of dividing sets an upper bound on the number of threads a model can benefit from in practice. Finally the hardware computational resources of the execution device limit the number of concurrently available threads, eventually leading to a decline in performance as overhead of thread switching begins to dominate the execution.

A.2 Weak scaling

Weak scaling presents what is frequently considered a lesser measure of parallel scaling of an application. In this scaling the computational load per thread is kept constant. The performance of the code is measured with increasing number of threads (which also clearly increases the overall problem size). While this measurement does not necessarily provide information on the scaling of a problem of given size onto a more parallel system, it is instead a measure of scaling a problem up in size. This is of particular interest for problem too large to fit on a single node. In addition, as applied to measurements of our models, weak scaling also provides a measure of the overhead scaling with problem size for the critical sections of our procedures. As with strong scaling, a weak scaling efficiency can be calculated. This is given by

$$Eff = \frac{t_1}{t_N} \times 100\% \tag{A.2}$$

where t_1 is the execution time for 1 thread and t_N is the execution time for N threads.

In practice measuring weak scaling tends to be more complex than measuring strong scaling for most moderately complex solvers and models. For convergence and iterative solvers the iterations required for a solution vary on problem size and therefore the execution load per thread to reach solution would vary by numbers of threads. In addition, once again it is not straight forward to perfectly balance a load among threads as problem size increases. Therefore weak scaling should be considered a more rough measure than strong scaling.

APPENDIX B

Raw Performance Data

Here we present the raw data used in our performance analysis, presented in Chapter 7. This serves both, as documentation of our performance analysis as well as a reference for some of the points of our performance analysis discussed in the main work but not immediately apparent under the axes resolution of the accompanying plots.

In the sections below we also summarise the data collection conditions for each case. This is to allow for ease of reference and/or reproduction of our results.

B.1 1 Dimensional Model

Table B.1: *Performance values for the strong scaling of the 1D model. Times given in seconds.*

Blocks: / Threads:	1	2	4	8	16	32	64	128
4	184.59	112.83	75.818	50.877	37.410	32.363	29.405	28.728
8	155.71	96.238	67.025	44.586	35.543	30.735	27.235	27.616
16	137.88	87.565	63.179	42.313	35.082	30.414	29.519	29.866
32	125.49	83.587	60.824	40.971	32.827	34.571	33.089	32.793
64	85.544	62.674	41.210	33.467	32.947	30.414	34.508	33.784
128	61.909	42.931	33.263	32.868	33.657	34.454	34.413	33.712
256	40.888	35.050	33.674	33.601	34.243	34.448	33.852	34.644
512	36.399	35.007	37.425	37.966	37.390	37.898	37.582	36.470

B.1 1 Dimensional Model

Table B.2: *Performance values for the weak scaling of the 1D model. Times given in seconds.*

Blocks: / Threads:	1	2	4	8	16	32	64	128
4	9.9285	11.247	11.566	10.909	12.053	15.524	17.966	25.864
8	10.790	12.049	13.008	12.918	16.121	19.285	25.486	39.447
16	14.332	14.331	16.262	16.473	20.759	26.427	39.029	58.863
32	18.530	18.475	20.944	25.753	27.477	40.467	59.443	114.82
64	19.716	20.556	25.906	28.516	40.920	58.691	114.44	215.68
128	20.314	25.978	28.411	41.129	58.289	110.41	223.09	410.46

A test 1D plasma model was simulated over 1000 timesteps and execution times under different thread/block configurations were recorded. The total length of the system was 6.7 cm, divided onto a grid of 512 cells. The peak of the applied sinusoidal voltage was 150V. This signal was of 13.56MHz frequency (f), with timestep of $(800f)^{-1}$. The initial plasma density was chosen as 5.12×10^{14} m^{-3} and each cell was initialised with 64 superparticles. The neutral gas density was chosen as 9.64×10^{20} m^{-3}. As is readily apparent, these parameters fall comfortable within the physically meaningful benchmark cases described in Chapter 6. The strong scaling data for these conditions is presented in Table B.1 while the data for the weak scaling analysis is shown in Table B.2.

As discussed in Appendix A, the weak scaling analysis required some modification to the model to achieve better load balancing for each thread. Thus the numerical simulation results of this analysis were not expected to

Table B.3: *Performance values for the strong scaling of the 2D model. Times given in seconds.*

Blocks: / Threads:	1×1	2×2	4×4	8×8	16×16	32×32	64×64
2×2	659	189.05	68.561	37.993	35.069	34.388	34.089
4×4	432.68	129.38	53.700	34.785	32.761	32.218	32.581
8×8	204.53	72.351	39.241	36.325	36.815	36.494	37.084
16×16	72.830	38.546	36.321	37.013	36.827	36.706	37.057

be accurate or physically meaningful. Instead we concentrated on replicating operations expected to be carried out on a timestep pass so as to better examine the scaling behaviour under constant load per thread. Thus to conserve the overall number of particles, we disabled particle losses and particle creation for this analysis. This is certainly not a perfect weak scaling measurement however with our architecture in mind it gives a reasonable approximation of the weak scaling of our model.

B.2 2 Dimensional Model

Since the 2D model was also being measured for performance rather than physical validation, the test measurements were carried out over 100 timesteps. The 2D model was somewhat more computationally complex than the 1D

Table B.4: *Performance values for the weak scaling of the 2D model. Times given in seconds.*

Blocks: / Threads:	1×1	2×2	4×4	8×8	16×16
2×2	0.969	1.0287	1.2062	1.6482	3.8611
4×4	1.3829	1.7025	2.0718	3.6980	10.666
8×8	1.8682	2.1273	4.0922	11.204	40.461
16×16	2.1889	4.2118	11.370	40.398	164.322

implementation, mainly due to the more complex field solver and therefore it was decided a more accurate approach to performance benchmarking would be to treat the 2D PIC-MCC model separate to the field solver. Therefore for these measurements the field solver was switched off. As a result these measurements only represent the scaling of the kinetic treatment, sorting, charge accumulation and acceleration calculation (albeit set to zero due to the potential being set at this value). The field solver performance analysis is instead presented separately in Sections 7.3 and B.3.

The physical parameters chosen for this simulation were similar to the ones given for the 1D model performance analysis. The neutral gas density was set at 9.64×10^{20}, the timestep was chosen as $(800f)^{-1}$, where $f =$ 13.56MHz and plasma density was initialised as 5.12×10^{14} m^{-3}. The physical size of the system was chosen as a square 6.7 cm $\times$ 6.7 cm in size. A cell grid of 128 $\times$ 128 was imposed on these physical values and each cell was initialised with 100 superparticles. As was the case with the 1D performance benchmark above, here we also attempted to better approximate weak scal-

Table B.5: *Performance values for the strong scaling of the DADI solver. Times given in seconds.*

Blocks: / Threads:	1×1	2×2	4×4	8×8	16×16	32×32	64×64
2×2	429	164.063	86.728	61.435	58.894	57.443	60.161
4×4	208.19	101.112	65.301	53.433	51.874	52.325	53.383
8×8	116.31	68.909	54.869	51.064	50.028	49.851	50.399
16×16	69.978	55.357	50.448	50.332	49.975	50.309	52.159

ing by disabling particle gains and losses for that analysis. The raw data for the strong and weak scaling of the 2D model is given in Tables B.3 and B.4 respectively.

B.3 2 Dimensional Field Solver

Our DADI field solver can be characterised as a composite parallel application, consisting of a linear system solver, in this case the PCR solver, and the DADI convergence solver. It is clear that the PCR solver is a non-trivial parallel implementation[7] and therefore merits separate parallel performance evaluation. Therefore in this analysis we examined both the PCR solver alone and (in the case of strong scaling) the DADI solver with fixed

Table B.6: *Performance values for the weak scaling of the DADI solver. Times given in milliseconds.*

Blocks: / Threads:	1×1	2×2	4×4	8×8	16×16	32×32	64×64
2×2	5.7857	6.1777	6.5539	6.1633	6.5277	7.4544	8.0819
4×4	6.3551	6.1804	6.4148	6.0265	6.3700	7.2567	8.0625
8×8	6.0215	6.1438	6.3527	6.1795	6.8876	7.6580	8.5372
16×16	6.0430	6.5244	5.9936	6.7466	7.3640	8.9542	12.771

Table B.7: *Performance values for the strong scaling of the PCR solver. Times given in milliseconds.*

Blocks: / Threads:	1	2	4	8	16	32	64	128	256	512
4	573.7	292.3	151.4	93.24	55.44	35.23	23.94	25.77	22.52	22.71
8	294.4	152.7	81.81	46.21	36.19	23.03	18.01	20.46	18.06	18.70
16	146.0	78.67	44.46	29.17	20.58	16.84	14.78	14.85	14.09	14.62
32	76.82	43.69	26.55	20.34	15.65	13.86	12.71	12.50	12.01	12.04
64	43.49	27.64	19.72	15.28	12.38	11.48	11.93	11.72	11.78	11.68
128	28.93	19.35	15.86	13.22	11.39	10.37	11.72	11.24	11.98	11.55
256	21.49	15.45	14.06	11.94	11.05	12.24	12.81	11.71	11.73	11.64
512	18.68	14.47	12.48	10.97	10.06	11.50	11.33	11.00	11.37	11.41

PCR thread/block configuration parameters. Unlike the PIC models, the DADI solver reaches convergence comparatively quickly so therefore in the strong scaling case instead of setting a constant number of passes manually we instead measured the time taken to converge onto a solution under a constant accuracy requirement. Since the problem size and the source term were kept constant for the strong scaling case, the number of passes required for convergence remains the same.

The DADI strong scaling measurements were carried out on a 128×128 grid and are presented in Table B.5. In contrast the strong scaling measurements of the PCR solver were calculated for a system size of 512×512 and are shown in Table B.7. This discrepancy between the sizes does not affect the validity of the measurements much since both solvers were treated separately. The DADI problem size is smaller due to the parallel scaling measurements taking excessively long for the low number of total threads configurations, as seen both from the plot in Figure 7.7 and the table above. The numerical parameters supplied to the DADI solver were for a normalised system of $\Delta x = \delta y = 1$ and $\rho/\epsilon_0 = 1$. For the PCR solver we used a Poisson solution in 1D as our test problem, where $\rho/\epsilon_0 = 1$ as well.

The weak scaling raw data for DADI and PCR solvers are shown in Tables B.6 and B.8 respectively. As described for the previous weak scaling cases, in our measurement we were more interested in keeping our operation load constant, rather than obtaining correct mathematical result. Therefore for the DADI solver we constrained our measurement to a single DADI pass, and for the PCR solver a single cyclic reduction.

Table B.8: *Performance values for the weak scaling of the PCR solver. Times given in milliseconds.*

Blocks: / Threads:	1	2	4	8	16	32	64	128	256	512
4	8.493	7.855	7.980	9.891	11.25	9.991	7.869	10.93	10.12	8.057
8	8.499	7.809	8.813	9.257	7.941	10.62	9.612	8.707	9.397	8.577
16	7.992	8.656	8.764	9.963	7.887	9.250	8.443	9.192	9.871	8.850
32	10.10	9.248	7.897	10.42	8.766	8.773	9.675	9.147	10.40	8.048
64	10.79	8.254	8.350	7.876	8.041	9.784	10.86	9.198	8.282	8.164
128	11.30	9.240	9.179	7.850	10.77	9.743	10.57	10.29	9.062	11.41
256	9.354	8.993	8.629	9.881	7.645	10.89	8.701	7.916	9.516	11.32
512	10.85	9.618	8.286	8.647	7.883	10.19	10.60	9.333	9.025	10.96

APPENDIX C

Conferences And Publications

C.1 Publications

Leap frog integrator modifications in highly collisional particle-in-cell codes

N. Hanzlikova, M.M. Turner

Journal of Computational Physics, **268**, 2014

C.2 Conferences

Paper and talk presentation (*A novel finite element method assembler for co-processors and accelerators*), IA^3 2013, Denver, CO, USA.

Attendant, Supercomputing 2013, Denver, CO, USA.

Poster presentation (*1 dimensional atmospheric Particle-In-Cell plasma sim-*

ulation on the GPU), ICOPS 2012, Edinburgh, UK.
Poster presentation (*Highly Parallel Particle-In-Cell Simulations Using CPUs and GPUs*), ICPIG 2011, Belfast, UK.
Poster presentation (*Simulations of Atmospheric Pressure Plasma using the Kinetic Approach*), Globe Forum 2010, Dublin, Ireland.
Attended Plasma Summer School and Masterclass 2010, Bad Honnef, Germany.

C.3 Solvers

- Parallel cyclic reduction solver:
  https://github.com/geekity/PCR
- Dynamic alternating direction implicit solver:
  https://github.com/geekity/ADI

Bibliography

[1] M. A. Lieberman and A. J. Lichtenberg, *Principles of Plasma Discharges and Materials Processing*. John Willey & Sons, Inc., 1994.

[2] A. A. Fridman, *Plasma Chemistry*. Cambridge University Press, 2008.

[3] Nvidia, *Nvidia CUDA programming Guide (V 4.0)*, 2011.

[4] C. K. Birdsall, "Particle-in-cell charged-particle simulations, plus Monte Carlo collisions with neutral atoms, PIC-MCC," *IEEE Transactions on Plasma Science*, vol. 19, pp. 65–85, Apr. 1991.

[5] C. Birdsall and A. Langdon, *Plasma Physics via Computer Simulation*. Institute Of Physics, 1991.

[6] P. Mertmann, D. Eremin, T. Mussenbrock, R. P. Brinkmann, and P. Awakowicz, "Fine-sorting one-dimensional particle-in-cell algorithm with monte-carlo collisions on a graphics processing unit," *Computer Physics Communications*, vol. 182, no. 10, pp. 2161 – 2167, 2011.

[7] Y. Zhang, J. Cohen, A. Davidson, and J. Owens, "A hybrid method for solving tridiagonal systems on the gpu," in *GPU Computing Gems Jade Edition* (W. Hwu, ed.), Elsevier, 2011.

[8] M. M. Turner, A. Derzsi, Z. Donkó, D. Eremin, S. J. Kelly, T. Lafleur, and T. Mussenbrock, "Simulation benchmarks for low-pressure plasmas: Capacitive discharges," *Physics of Plasmas*, vol. 20, p. 013507, Jan. 2013.

[9] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*. Taylor & Francis, Jan. 1989.

[10] D. Bohm, "Minimum ionic kinetic energy for a stable sheath," *The Characteristics of Electrical Discharges in Magnetic Fields, MacGraw-Hill, New York*, vol. Chapter 3, pp. 77–86, 1949.

[11] H. R. Griem, "High-density corrections in plasma spectroscopy," *Phys. Rev.*, vol. 128, pp. 997–1003, Nov 1962.

[12] C. Tendero, C. Tixier, P. Tristant, J. Desmaison, and P. Leprince, "Atmospheric pressure plasmas: A review," *Spectrochimica Acta*, vol. 61, pp. 2–30, Jan. 2006.

[13] P. Fauchais, A. Vardelle, and B. Dussoubs, "Quo vadis thermal spraying?," *Journal of Thermal Spray Technology*, vol. 10, no. 1, pp. 44–66, 2001.

[14] "Pva-tepla," 2014.

[15] "Corotec," 2014.

[16] A. Schutze, J. Jeong, S. Babayan, J. Park, G. S. Selwyn, and R. F. Hicks, "The atmospheric-pressure plasma jet: a review and comparison to other plasma sources," *Plasma Science, IEEE Transactions on*, vol. 26, pp. 1685–1694, Dec 1998.

[17] W. Siemens *Poggendorff's Ann. Phys. Chem*, vol. 102, p. 66, 1857.

[18] K. Buss, "Die elektrodenlose entladung nach messung mit dem kathodenoszillographen," *Electrical Engineering (Archiv fur Elektrotechnik)*, vol. 26, no. 4, pp. 261–265, 1932. 10.1007/BF01657192.

[19] F. Massines, A. Rabehi, P. Decomps, R. B. Gadri, P. Segur, and C. Mayoux, "Experimental and theoretical study of a glow discharge at atmospheric pressure controlled by dielectric barrier," *J. Appl. Phys.*, vol. 83, no. 6, pp. 2950–2957, 1998.

[20] K. G. Donohoe and T. Wydeven, "Plasma polymerization of ethylene in an atmospheric pressure-pulsed discharge," *Journal of Applied Polymer Science*, vol. 23, no. 9, pp. 2591–2601, 1979.

[21] S. Kanazawa, M. Kogoma, T. Moriwaki, and S. Okazaki, "Stable glow plasma at atmospheric pressure," *Journal of Physics D: Applied Physics*, vol. 21, no. 5, p. 838, 1988.

[22] J. Park, I. Henins, H. Herrmann, G. Selwyn, J. Jeong, R. Hicks, D. Shim, and C. Chang, "An atmospheric plasma source," *Applied Physics Letters*, vol. 76, no. 3, pp. 288–290, 2000.

[23] Y. P. Raizer, *Gas Discharge Physics*. Springer, 1997.

[24] I. Alexeff and M. Laroussi, "The uniform, steady-state atmospheric

pressure DC plasma," *IEEE Transactions on Plasma Science*, vol. 30, pp. 174–175, Feb. 2002.

[25] F. Massines, A. Rabehi, P. Decomps, R. B. Gadri, P. Ségur, and C. Mayoux, "Experimental and theoretical study of a glow discharge at atmospheric pressure controlled by dielectric barrier," *Journal of Applied Physics*, vol. 83, pp. 2950–2957, Mar. 1998.

[26] D. W. Liu, F. Iza, and M. G. Kong, "Evolution of Atmospheric-Pressure RF Plasmas as the Excitation Frequency Increases," *Plasma Processes and Polymers*, vol. 6, no. 6-7, pp. 446–450, 2009.

[27] K. H. Becker, K. H. Schoenbach, and J. G. Eden, "TOPICAL REVIEW: Microplasmas and applications," *Journal of Physics D Applied Physics*, vol. 39, p. 55, Feb. 2006.

[28] O. Buneman, "Dissipation of currents in ionized media," *Phys. Rev.*, vol. 115, pp. 503–517, Aug 1959.

[29] C. K. Birdsall and W. B. Bridges, "Space-Charge Instabilities in Electron Diodes and Plasma Converters," *Journal of Applied Physics*, vol. 32, pp. 2611–2618, Dec. 1961.

[30] J. Dawson, "One-dimensional plasma model," *Physics Of Fluids*, vol. 5, pp. 445–59, 1962.

[31] P. Burger, "Theory of Large-Amplitude Oscillations in the One-Dimensional Low-Pressure Cesium Thermionic Converter," *Journal of Applied Physics*, vol. 36, pp. 1938–1943, June 1965.

[32] P. Burger, "Elastic Collisions in Simulating One-Dimensional Plasma

Diodes on the Computer," *Physics of Fluids*, vol. 10, pp. 658–666, Mar. 1967.

[33] R. Shanny, J. M. Dawson, and J. M. Greene, "One-Dimensional Model of a Lorentz Plasma," *Physics of Fluids*, vol. 10, pp. 1281–1287, June 1967.

[34] T. Takizuka and H. Abe, "A binary collision model for plasma simulation with a particle code," *Journal of Computational Physics*, vol. 25, pp. 205–219, Nov. 1977.

[35] R. J. Procassini, C. K. Birdsall, E. C. Morse, and B. I. Cohen, "A relativistic Monte Carlo binary collision model for use in plasma particle simulation codes," tech. rep., May 1987.

[36] X. Wang, C. Li, M. Lu, and Y. Pu, "Study on an atmospheric pressure glow discharge," *Plasma Sources Science Technology*, vol. 12, pp. 358–361, Aug. 2003.

[37] R. Hockney, R. Warriner, and M. Reiser, "Two-dimensional particle models in semiconductor-device analysis," *Electronics Letters*, vol. 10, pp. 484 –486, 14 1974.

[38] B. Eliasson, W. Egli, and U. Kogelschatz, "Modelling of dielectric barrier discharge chemistry," *Pure Appl. Chem.*, vol. 66, no. 6, pp. 1275–1286, 1994.

[39] R. Dorai and M. J. Kushner, "Consequences of propene and propane on plasma remediation of no[sub x]," *Journal of Applied Physics*, vol. 88, no. 6, pp. 3739–3747, 2000.

[40] A. Bogaerts and R. Gijbels, "Numerical modelling of gas discharge plasmas for various applications," *Vacuum*, vol. 69, no. 13, pp. 37 – 52, 2002. ¡ce:title¿12th International School on Vacuum, Electron and Ion Technologie s, 17-22 September 2001, Varna, Bulgaria¡/ce:title¿.

[41] M. J. Kushner, "Hybrid modelling of low temperature plasmas for fundamental investigations and equipment design," *Journal of Physics D: Applied Physics*, vol. 42, no. 19, p. 194013, 2009.

[42] J. J. Shi and M. G. Kong, "Cathode fall characteristics in a dc atmospheric pressure glow discharge," *Journal of Applied Physics*, vol. 94, no. 9, pp. 5504–5513, 2003.

[43] X. M. Zhu and M. G. Kong, "Electron kinetic effects in atmospheric dielectric-barrier glow discharges," *Journal of Applied Physics*, vol. 97, no. 8, p. 083301, 2005.

[44] S. Avtaeva and A. Skornyakov, "Effect of nonlocal electron kinetics on the characteristics of a dielectric barrier discharge in xenon," *Plasma Physics Reports*, vol. 35, pp. 593–602, 2009. 10.1134/S1063780X09070083.

[45] S. M. Lee, Y. J. Hong, Y. S. Seo, F. Iza, G. C. Kim, and J. K. Lee, "Simulations of biomedical atmospheric-pressure discharges," *Computer Physics Communications*, vol. 180, no. 4, pp. 636 – 641, 2009.

[46] C. A. Fichtl, J. M. Finn, and K. L. Cartwright, "An Arbitrary Curvilinear Coordinate Method for Particle-In-Cell Modeling," *ArXiv e-prints*, Jan. 2012.

[47] M. Swaine, "New Chip from Intel Gives High-Quality Displays," *Info World*, vol. 5, no. 11, p. 16, 1983.

[48] Nvidia, "NVIDIA: GeForce 256 - The World's First GPU." http://www.nvidia.com/page/geforce256.html, 1999.

[49] Nvidia, *Nvidia CUDA Programming Guide (V 1.0)*, 2007.

[50] Apple Inc., "Apple Previews Mac OS X Snow Leopard to Developers." http://www.apple.com/pr/library/2008/06/09Apple-Previews-Mac-OS-X-Snow-Leopard-to-Developers.html, 2008.

[51] J. Sanders and E. Kandrot, *CUDA By Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1 ed., 2010.

[52] G. Stantchev, W. Dorland, and N. Gumerov, "Fast parallel Particle-To-Grid interpolation for plasma PIC simulations on the GPU," *J. Parallel Distr. Com.*, vol. 68, pp. 1339–1349, 2008.

[53] G. Stantchev, D. Juba, W. Dorland, and A. Varshney, "Using graphics processors for high-performance computation and visualization of plasma turbulence," *Computing in Science Engineering*, vol. 11, pp. 52 –59, march-april 2009.

[54] X. Kong, M. C. Huang, C. Ren, and V. K. Decyk, "Particle-in-cell simulations with charge-conserving current deposition on graphic processing units," *Journal of Computational Physics*, vol. 230, no. 4, pp. 1676 – 1685, 2011.

[55] I. C. for High-End Computing, "GPGPU Research Projects." http://www.ichec.ie/research/gpgpu_projects.php, 2012.

[56] H. Burau, R. Widera, W. Ho andnig, G. Juckeland, A. Debus, T. Kluge, U. Schramm, T. Cowan, R. Sauerbrey, and M. Bussmann, "Picongpu: A

fully relativistic particle-in-cell code for a gpu cluster," *Plasma Science, IEEE Transactions on*, vol. 38, pp. 2831 –2839, oct. 2010.

[57] V. Vahedi and G. DiPeso, "Simultaneous potential and circuit solution for two-dimensional bounded plasma simulation codes," *J. Comput. Phys.*, vol. 131, pp. 149–163, Feb. 1997.

[58] K. Koura, "Null-collision technique in the direct-simulation Monte Carlo method," *Physics of Fluids*, vol. 29, pp. 3509–3511, Nov. 1986.

[59] G. Ruetsch, P. Micikevicius, and T. Scudiero, *Optimizing Matrix Transpose in Cuda.* Nvidia CUDA.

[60] S. Doss and K. Miller, "Dynamic ADI Methods for Elliptic Equations," *SIAM Journal on Numerical Analysis*, vol. 16, pp. 837–856, Oct. 1979.

[61] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C.* Cambridge University Press, 1 ed., Feb. 1988.

[62] Z. Wei, B. Jang, Y. Zhang, and Y. Jia, "Parallelizing alternating direction implicit solver on {GPUs}," *Procedia Computer Science*, vol. 18, no. 0, pp. 389 – 398, 2013. 2013 International Conference on Computational Science.

[63] I.-J. Sung, J. A. Stratton, and W.-M. W. Hwu, "Data layout transformation exploiting memory-level parallelism in structured grid many-core applications," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, (New York, NY, USA), pp. 513–522, ACM, 2010.

[64] N. Hanzlikova and M. Turner, "Leap frog integrator modifications

in highly collisional particle-in-cell codes," *Journal of Computational Physics*, vol. 268, no. 0, pp. 355 – 362, 2014.

[65] G. Chen, L. Chacón, and D. Barnes, "An energy- and charge-conserving, implicit, electrostatic particle-in-cell algorithm," *Journal of Computational Physics*, vol. 230, no. 18, pp. 7018 – 7036, 2011.

[66] E. Kawamura, C. K. Birdsall, and V. Vahedi, "Physical and numerical methods of speeding up particle codes and paralleling as applied to RF discharges," *Plasma Sources Science Technology*, vol. 9, pp. 413–428, Aug. 2000.

[67] L. Hatton and A. Roberts, "How accurate is scientific software?," *IEEE Trans. Softw. Eng.*, vol. 20, pp. 785–797, Oct. 1994.

[68] L. Hatton, "The t experiments: Errors in scientific software," *IEEE Comput. Sci. Eng.*, vol. 4, pp. 27–38, Apr. 1997.

[69] C. J. Roy and W. L. Oberkampf, "A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing," *Computer Methods in Applied Mechanics and Engineering*, vol. 200, no. 2528, pp. 2131 – 2144, 2011.

[70] I. D. Reid, "An investigation of the accuracy of numerical solutions of Boltzmann's equation for electron swarms in gases with large inelastic cross sections," *Australian Journal of Physics*, vol. 32, p. 231, June 1979.

[71] L. C. Pitchford, S. V. ONeil, and J. R. Rumble, "Extended boltzmann analysis of electron swarm experiments," *Phys. Rev. A*, vol. 23, pp. 294–304, Jan 1981.

[72] J. P. Verboncoeur, G. J. Parker, B. M. Penetrante, and W. L. Morgan, "Comparison of collision rates in particle-in-cell, Monte Carlo, and Boltzmann codes," *Journal of Applied Physics*, vol. 80, pp. 1299–1303, Aug. 1996.

[73] Z. Raspopovic, S. Sakadzic, S. Bzenic, and Z. Petrovic, "Benchmark calculations for monte carlo simulations of electron transport," *Plasma Science, IEEE Transactions on*, vol. 27, pp. 1241–1248, Oct 1999.

[74] N. R. Pinhão, Z. Donkó, D. Loffhagen, M. J. Pinheiro, and E. A. Richley, "Comparison of kinetic calculation techniques for the analysis of electron swarm transport at low to moderate E/N values," *Plasma Sources Science Technology*, vol. 13, pp. 719–728, Nov. 2004.

[75] Z. L. Petrović, S. Dujko, D. Marić, G. Malović, Ž. Nikitović, O. Šašić, J. Jovanović, V. Stojanović, and M. Radmilović-Radjenović, "REVIEW ARTICLE: Measurement and interpretation of swarm parameters and their application in plasma modelling," *Journal of Physics D Applied Physics*, vol. 42, p. 194002, Oct. 2009.

[76] J. E. Lawler and U. Kortshagen, "Self-consistent Monte Carlo simulations of the positive column of gas discharges," *Journal of Physics D Applied Physics*, vol. 32, pp. 3188–3198, Dec. 1999.

[77] V. A. Godyak, R. B. Piejak, and B. M. Alexandrovich, "Measurement of electron energy distribution in low-pressure RF discharges," *Plasma Sources Science Technology*, vol. 1, pp. 36–58, Mar. 1992.

[78] M. Surendra, "Radiofrequency discharge benchmark model comparison," *Plasma Sources Science Technology*, vol. 4, pp. 56–73, Feb. 1995.

[79] A. V. Phelps, "The application of scattering cross sections to ion flux models in discharge sheaths," *Journal of Applied Physics*, vol. 76, pp. 747–753, July 1994.

[80] S. F. Biagi, "Biagi v7.1 experimental cross section data set," Jan. 2014.

[81] A. V. Phelps, "Compilation of atomic and molecular data," 2005.

[82] D. Nikolopoulos, E. Ayguade, and C. Polychronopoulos, "Scaling irregular parallel codes with minimal programming effort," in *Supercomputing, ACM/IEEE 2001 Conference*, pp. 5–5, Nov 2001.

[83] N. Hanzlikova and E. R. Rodrigues, "A novel finite element method assembler for co-processors and accelerators," in *Proceedings of the 3rd Workshop on Irregular Applications: Architectures and Algorithms*, IA^3 '13, (New York, NY, USA), pp. 1:1–1:8, ACM, 2013.

[84] J. Dongarra and P. Luszczek, "Linpack benchmark," in *Encyclopedia of Parallel Computing* (D. Padua, ed.), pp. 1033–1036, Springer US, 2011.

[85] M. J. Flynn, O. Mencer, V. Milutinovic, G. Rakocevic, P. Stenstrom, R. Trobec, and M. Valero, "Moving from petaflops to petadata," *Commun. ACM*, vol. 56, pp. 39–42, May 2013.

[86] NVIDIA, "Nvidia geforce gtx 760 specifications," 2014.

[87] V. A. Godyak, R. B. Piejak, and B. M. Alexandrovich, "Evolution of the electron-energy-distribution function during rf discharge transition to the high-voltage mode," *Phys. Rev. Lett.*, vol. 68, pp. 40–43, Jan 1992.

[88] R. G. Houben and M. M. Turner, "Metastables and secondary emission in a particle in cell simulation of a helium plasma." 2014.

[89] R. Hockney, "Performance parameters and benchmarking of supercomputers," *Parallel Computing*, vol. 17, no. 1011, pp. 1111 – 1130, 1991. Benchmarking of high performance supercomputers.

[90] X.-H. Sun and J. L. Gustafson, "Toward a better parallel performance metric," *Parallel Computing*, vol. 17, no. 1011, pp. 1093 – 1109, 1991. Benchmarking of high performance supercomputers.

[91] D. Bailey, *The NAS Parallel Benchmarks*. NASA technical memorandum, National Aeronautics and Space Administration, Ames Research Center, 1993.

[92] Sharcnet-Consortium, "Measuring parallel scaling performance," 2014.