A Parallel Multigrid Poisson Solver for PINC, a new Particle-in-Cell Model

by

Gullik Vetvik Killie

Thesis

for the degree of

Master of Science

Faculty of Mathematics and Natural Sciences
University of Oslo

November 2016

Abstract

This thesis is about the development of a parallel multigrid solver to the Particle-in-Cell program PINC. The workings of the multigrid solver is described as well the most important parts of PINC. The solver is confirmed to work accurately on various test cases. The convergence rate of the algorithm was found to be between 0.149 and 0.203 for various grid sizes. A Langmuir oscillation was simulated with the PINC, where it performed the expected number of oscillations confirming that the program as a whole works.

Acknowledgements

I thank especially my girlfriend for keeping with me and supporting me through my high and lows. As well as my family for unconditional support.

I express sincere to my advisor Wojciech Miloch for support and more help than could be expected.

Sigvald Marholm recieves my thanks for teaching my to be a better programmer, in addition to the homebrewed beers he occasionaly brings. Shafa Aria deserves gratitude for being available for plasma discussions and being my office-mate.

Contents

Chapter 1

Introduction

As the computational capabilities of supercomputers increase, it is becoming more viable to simulate larger domains of plasma in more detail. This can help us foresee space-weather and also understand many fundamental phenomena of plasma. Particle-in-Cell is one widely used method to simulate space plasma (Lapenta, 2012). To efficiently use massive parallel computer the Particle-in-Cell method need efficient parallel field solvers. For the electrostatic case Multigrid solver are well suited to solve the Poisson equation. This thesis follows the development of a parallel multigrid solver to accompany the new Particle-in-Cell model PINC.

The thesis first introduces plasma and gives an overview the theoretical background, in chapter 2. Plasma as a state is defined, single particle motion in charged field is introduced and the fluid description of plasma shown. The fundamental Langmuir Oscillation is derived from the fluid description before a short overview of ways to numerically simulate plasma. The next chapter, chapter 3 gives an overview of the PiC method and our implementation is introduced. The theory behind the parallel multigrid is discussed afterwards. The implementation chapter, chapter 4, gives the details behind the actual implementation of the multigrid solver. The verification and performance of both PINC and the multigrid solver is discussed in chapter 5, before a summary and proposals for further developments follows in chapter 6.

Chapter 2

Theoretical Background

2.1 Plasma

This section presents a short overview of basic plasma theory, it can serve as a quick reminder for those already familiar with the subject, and necessary background to understand the numerical simulations in this work. For a more thorough introduction the books *Plasma Physics* (Fitzpatrick, 2014), *Introduction to Plasma Physics* (Goldston and Rutherford, 1995), *Waves and Oscillations in Plasmas* (Pécseli, 2012) or the classic *Introduction to Plasma Physics and Controlled Fusion* (Chen, 1984) can be consulted.

Plasma is the fourth, lesser known, state of matter. It is similar to a gas in that the particles are free to move, but it has the key distinction that a part of its constituent particles are electrically charged.

"A plasma is a quasineutral gas of charged and neutral particles which exhibits collective behaviour."

Francis F. Chen

The charge causes the particles to be subject to the Lorentz force, which changes the behaviour of the gas. The plasma state is a typical state of matter and appears in various environments, i.e. the Sun, and other stars, the upper parts of Earoth's atmosphere, and has many industrial applications as plasma cutters, argon light tubes or fusion. The plasma density and temperature can extend over several orders of magnitude as shown in section 2.1.

A better understanding of the mechanism governing a plasma's behaviour can help us predict spaceweather, improve design of spacecraft and instruments affacted by plasma, future fusion devices and plasma technologies.

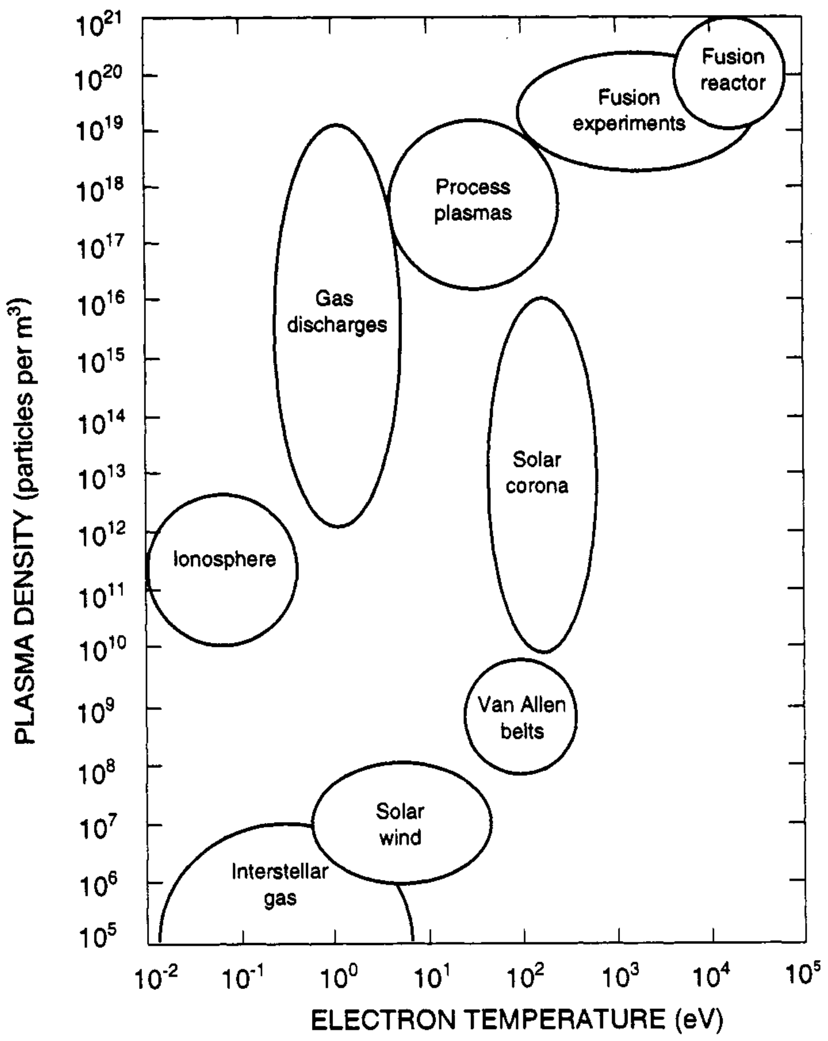


Figure 2.1: Plasmas occurs both in the hot and dense conditions in necessary for fusion, as well as in the cold and sparse interstellar environment. Figure after Goldston and Rutherford (1995).

2.1.1 Plasma Parameters

In this section we introduce basic plasma parameters that are also of concern for plasma simulations, which are the temperature, plasma frequency and Debye length. We also introduce the concept of quasineutrality.

Temperature

A modern view of temperature comes from kinetic theory developed by Maxwell and Boltzmann (Swendsen, 2006). We here provide only the final result of the detailed treatment that can be found in Goldston and Rutherford (1995). Temperature, T, is then related to the average kinetic energy of a particle, E_k. For an ideal monoatomic gas the kinetic energy is then

$$\bar{E}_k = \frac{1}{2}mv_{th}^2 = \frac{3}{2}kT, \tag{2.1}$$

where m is the mass and k is the Boltzmann constant. Here we have introduced $v_{th} \equiv \sqrt{kT/m}$ as the thermal velocity, i.e. the average velocity of a particle considering only one dimension. It should be mentioned that the fraction in front of the temperature is dependent on the degrees of freedom of the particle. A monoatomic particle can only move in three directions, but a diatomic particle can also vibrate and spin.

If the particles in a plasma collide often compared to the characteristic timescales of energy and particle changes, the particle velocity distribution can be approximated by a Maxwellian distribution. It is only then that the concept of temperature is valid (Goldston and Rutherford, 1995).

In high energy plasma physics it is also costumary to drop the Boltzmann factor, k, in eq. (2.1), and express temperature directly in electronvolt, eV. Electronvolt is defined as the energy it takes to move an elementary charge through a potential difference of 1 V, and corresponds to approximately 11600K.

Electron Plasma Frequency

A rather important frequency in plasma physics is the electron plasma frequency (Chen, 1984),

$$\omega_{pe} \equiv \sqrt{\frac{ne^2}{\epsilon_0 m_e}} \tag{2.2}$$

This frequency, ω_{pe}, is dependent on the number density, n, the fundamental charge, e, the vacuum permittivity, ϵ_0, and the electron mass, m_e. It can be thought of as a typical electrostatic oscillatory frequency. Consider an electrically neutral 1D slab, which is then disturbed, from its neutrality, by an infinitesimal charge density on one side.

$$\sigma = en\delta x \tag{2.3}$$

It will have an equal and opposite charge density on the other side. The slab will then have an electric field due to the charge density, caused by Gauss' Law.

$$\frac{\partial E_x}{\partial x} = -\frac{\sigma}{\epsilon_0} \qquad \rightarrow \qquad E_x = \frac{-en\delta x}{\epsilon_0} \tag{2.4}$$

Inserting this field as the only force in Newtons' law for a single particle yields

$$m\frac{\partial \delta x}{\partial t} = eE_x = -m\omega_{pe}^2 \delta x \tag{2.5}$$

The particle will then oscillate around its equilibrium position with the electron plasma frequency. The same phenomenon often happens in plasma as it tries to go back to its equilibrium and is called plasma oscillations, or Langmuir oscillations, see section 2.5 for a treatment of plasma oscillations.

An otherwise useful timescale is the reciprocal of the plasma frequency, the plasma period

$$\tau_p \equiv 2\pi/\omega_{pe} \tag{2.6}$$

Some researchers prefer to define the τ_p, without the 2π prefactor (Fitzpatrick, 2014).

Debye Shielding

Debye shielding length is the distance at which the electric influence from a particle is shielded out by the surrounding plasma. Consider a charged particle immersed in a plasma bath. The plasma is in a thermodynamical equilibrium, i.e. there is no significant temperature gradients. We artificially place a positively charged ion into the plasma. This ion will then attract electrons and repel positive ions. As a result There will be more negatively charged particles, and fewer positively charged particles, near the ion. This will form an electric shield around the ion. The distance away from a particle, where its field is reduced by a factor $1/e$, is called the Debye Shielding Length, λ_D, defined as:

$$\lambda_D \equiv \sqrt{\frac{\epsilon_0 k T_e}{n_e e^2}} \tag{2.7}$$

The above definition is often used, (Pécseli, 2012), neglecting the ion influence since they often have a much lower temperature. The shielding length is dependent on the ratio between the electron temperature, T_e, and electron density, n_e. In a warmer plasma the particles will move quickly and efficiently shield any charges, so the λ_D becomes smaller. In cases where we also need to account for ions, a more complete definition can be used

$$\lambda_D \equiv \sqrt{\frac{\epsilon_0 k T_e}{n_e e^2 (1 + Z\frac{T_e}{T_i})}} \tag{2.8}$$

Due to the earlier argument, and the statistical approach used when deriving it (Goldston and Rutherford, 1995), there must be a significant amount particles close to the ion to shield it out.

It should be noted that the shielding length is related (Fitzpatrick, 2014) to the plasma period and the thermal velocity through

$$\lambda_D \omega_{pe} = v_{th} \tag{2.9}$$

Quasineutrality

The assumption of quasi-neutrality is a crucial approximation for collective phenomena in plasma physics. By quasi-neutrality we assume that the electron density is equal to the ion density, $n_e \approx n_i$. This is often called the *"plasma approximation"* (Chen, 1984). This approximation is usually valid on length scales much larger than the shielding length. If we had a case where a large volume of plasma lost a significant amount of charge, a large electric field would accompany the density imbalance. This electric field would quickly correct the imbalance, and quasineutrality would be regained.

Plasma Classification

For a plasma description to be applicable the system we consider must have a typical length scale, L, and time scale, τ, larger than the Debye length and plasma period respectively.

$$\frac{\lambda_D}{L} \ll 1 \qquad\qquad \frac{\tau_p}{\tau} \ll 1$$

A plasma treatment aims to describe the collective behaviour of many particles. If the length scale is smaller than the λ_D the charges from individual particles are not effectively shielded out and the trajectories of each particle should be considered. On a timescale shorter than τ_{pe}, the collective plasma oscillation cannot be observed.

2.2 Single Particle Motion

To better understand the collective motion of plasma it is useful to consider the motions of single particles that the plasma consists of. By at first treating only one particle we can ignore the electromagnetic influence from other particles which greatly simplifies the situation. The Lorentz force, eq. (2.10), governs the dynamics of a charged particle in a plasma, provided that other forces, e.g. gravity, are neglible. The Lorentz force is due to charged particles in the electric field, **E**, and charged particles moving across the magnetic field **B**

$$\mathbf{F} = q\left(\mathbf{E} + \mathbf{v} \times \mathbf{B}\right) \tag{2.10}$$

To simplify matters we will only consider particles in static electric and magnetic fields, as that is often a valid approximation on the time and spatial scales of interest.

2.2.1 Gyration

Let us consider a situation with a single moving particle in a static and isotropic external magnetic field, i.e. $\mathbf{E} = \mathbf{0}$, $\mathbf{B} \neq 0$, a similar set up as in Baumjohann and Treumann (1997). Newton's Second law together with eq. (2.10) then gives

$$m\frac{\partial \mathbf{v}}{\partial t} = q\mathbf{v} \times \mathbf{B} \tag{2.11}$$

We should note that the velocity component parallel to the magnetic field, is not affected by the field and will remain constant, $\frac{\partial \mathbf{v}_{\|}}{\partial t} = 0$. The cross product of two parallel vectors is always zero, so $\mathbf{v}_{\|} \times \mathbf{B} = 0$. Using these two notions we can write the equation only in terms of the perpendicular, with respect to **B**, velocity.

$$m\frac{\partial \mathbf{v}_{\perp}}{\partial t} = q\mathbf{v}_{\perp} \times \mathbf{B} \tag{2.12}$$

Then we perform a temporal derivative.

$$\frac{\partial^2 \mathbf{v}_{\perp}}{\partial t^2} = \frac{q}{m}\frac{\partial \mathbf{v}_{\perp}}{\partial t} \times \mathbf{B} \tag{2.13}$$

Then we insert eq. (2.12) into the equation and use the vector relation $a \times b \times c = b(a \cdot b) - c(a \cdot b)$.

$$\frac{\partial^2 \omega_{\perp}}{\partial t^2} + \left(\frac{qB}{m}\right)^2 \omega_{\perp} = 0 \tag{2.14}$$

In the last equation we also changed the term, $v_{\perp}$, describing the rotational motion, to $\omega_{\perp}$, which will from now on signify gyrational motion. This differential equation corresponds to a gyration around the magnetic field lines with the gyration frequency, $\omega_c = \frac{qB}{m}$, as the frequency. The particles are free to move parallel to the magnetic field lines causing a spiralling motion along the magnetic field lines, as illustrated in fig. 2.2a. The high mobility along mangetic field lines is often an important part of why there are field-aligned currents, such as 'Birkeland Currents' (Cummings and Dessler, 1967), transporting plasma along magnetic field lines.

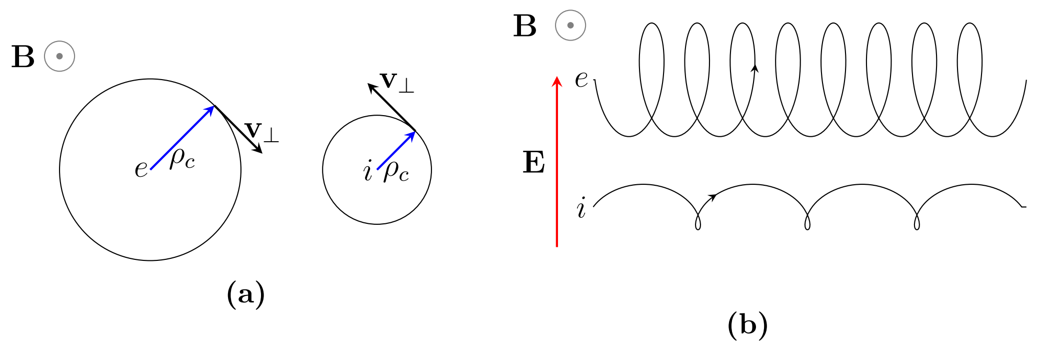


Figure 2.2: (a) The trajectories of an electron, left, and a positive ion, right, are shown. In both cases the trajectories of particles are are gyrating around the magnetic field lines. (b) Here we can see examples of particles experiencing the E-cross-B drift. The particle motion consists of a gyration as well a constant drift in the $E \times B$ direction.

2.2.2 E-cross-B Drift

A drift called E-cross-B drift can appear when a particle is moving within static and isotrop electric and magnetic fields. The equation of motion, neglecting all forces except the electromagnetic, is then

$$m\frac{\partial v}{\partial t} = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \tag{2.15}$$

In plasma physics it is often a good strategy to decompose quantities into parallel and perpendicular, with respect to $\mathbf{B}$. We start by seperating the velocity into $\mathbf{v} = \mathbf{v}_{\parallel} + \mathbf{v}_{\perp}$ and the electric field into $\mathbf{E} = \mathbf{E}_{\parallel} + \mathbf{E}_{\perp}$. Inserting this and using that $\mathbf{v}_{\parallel} \times \mathbf{B} = 0$ the equation, eq. (2.15), becomes

$$m\frac{\partial}{\partial t}\left(\mathbf{v}_{\parallel} + \mathbf{v}_{\perp}\right) = q\left(\mathbf{E}_{\perp} + \mathbf{E}_{\parallel} + (\mathbf{v}_{\perp}) \times \mathbf{B}\right) \tag{2.16}$$

The parallel motion consists of an acceleration caused by the parallel part of the electric field and is given by:

$$m\frac{\partial \mathbf{v}_{\parallel}}{\partial t} = q\mathbf{E}_{\parallel} \tag{2.17}$$

The remaining part of the equation describes the perpendicular motion:

$$m\frac{\partial \mathbf{v}_{\perp}}{\partial t} = q\left(\mathbf{E}_{\perp} + (\mathbf{v}_{\perp}) \times \mathbf{B}\right) \tag{2.18}$$

Now we assume that there is a time-invariant drift $\mathbf{v}_D$, i.e. not dependent on time, and we seperate the perpendicular motion into a drift and gyration, $\mathbf{v} = \mathbf{v}_{\parallel} + \boldsymbol{\omega}_{\perp} + \mathbf{v}_D$.

$$m\frac{\partial}{\partial t}\left(\boldsymbol{\omega}_\perp + \mathbf{v}_D\right) = q\left(\mathbf{E}_\perp + \left(\boldsymbol{\omega}_\perp + \mathbf{v}_D\right) \times \mathbf{B}\right) \tag{2.19}$$

From section 2.2.1 we know that the gyration part is given by

$$m\frac{\partial \omega_\perp}{\partial t} = q\omega_\perp \times \mathbf{B} \tag{2.20}$$

Taking this out of the equation we have

$$\frac{\partial \mathbf{v}_D}{\partial t} = \frac{q}{m}\left(\mathbf{E}_\perp + \mathbf{v}_D \times \mathbf{B}\right) \tag{2.21}$$

Then we use the previous assumption that the drift velocity is constant, cross the equation with **B** and simplify, see Goldston and Rutherford (1995), to arrive at

$$\mathbf{v}_D = \frac{\mathbf{E} \times \mathbf{B}}{B^2} \tag{2.22}$$

As we can see the E-cross-B drift is independent of the particle charge and mass, which means that both the ions and electrons will be drifting in the same direction and speed perpendicular to the electric and magnetic fields, as it is also shown in fig. 2.2b.

The $E \times B$ drift is an example of a single particle motion in electric and magnetic fields. There are many other important concepts and drifts when considering single particle motion, such as gradient-B drift, curvature drift, polarization drift and magnetic mirroring. We will not present them here and refer to Fitzpatrick (2014), or other introductionary plasma physics book for details. We note that the understanding of motions of single particles is necessary to study the collective behaviour of large amounts of particles constituting a plasma.

2.3 Kinetic Theory

In the previous section we provided examples of single particle motion. To deal with collective phenomena we need to account for large amount of particles simultaneously. This can be done with the kinetic theory that we will introduce here.

To consider a large amount of particles we consider a charge and current density instead of the individual particles. Let $\mathcal{F}_s$ be the exact phase-space density of a particle species, it contains all the positions, velocities for all the particles at all times. By integrating over all velocities and multiplying with the charge for all species we obtain the charge density, ρ_c.

$$\rho_c = \sum_s e_s \int \mathcal{F}_s(\mathbf{r}, \mathbf{v}, t)\,\mathrm{d}^3\mathbf{v}$$

Likewise we find the current density, $\mathbf{j}$ by:

$$\mathbf{j} = \sum_s e_s \int \mathbf{v}\mathcal{F}_s(\mathbf{r}, \mathbf{v}, t)\,\mathrm{d}^3\mathbf{v}$$

Then its seems we can derive all plasma interaction from considering the conservation of the phase-space density, coupled with Maxwells equations. The phase-space conservation is given by what is known as the Vlasov equation eq. (2.23) (Pécseli, 2012)):

$$\frac{\partial \mathcal{F}}{\partial t} + \mathbf{v} \cdot \nabla \mathcal{F}_s + \frac{e_s}{m_s}\left(\mathbf{E} + v \times \mathbf{B}\right) \cdot \nabla_v \mathcal{F}_s = 0 \tag{2.23}$$

where $\nabla_v = \left(\frac{\partial}{\partial v_x}\hat{i} + \frac{\partial}{\partial v_y}\hat{j} + \frac{\partial}{\partial v_z}\hat{k}\right)$ is the velocity grad-operator. Unfortunately this expression, combined with Maxwells equations, is only solvable for special simple geometries.

2.4 Fluid Description

In another limit one can consider plasma as conducting fluids. Here we are not interested in single particle motion anymore, but in the motion of a small volume of plasma. This section provides an overview of the derivation of the fluid equations from the kinetic theory, by taking different order moments of the Vlasov equation. This can help in understanding the limitations of the fluid model of plasma. Lastly a few different approximations are introduced to make the fluid equations closable.

2.4.1 Velocity Moments

To investigate plasma as a fluid we have to make certain fluid approximations. The plasma is then characterized by local parameters describing particle density, kinetic temperature, flow velocity and so on. These parameters refer to a small volume of plasma, in contrast with the discussions earlier about single particle motion, section 2.2. The time evolution is then governed by the fluid equation, but unfortunately the resulting equations are generally less tractable than the usual hydrodynamical equations. This is because they need to be closed with Maxwell's equations.

The first order moment is given by mass times velocity, in introductionary physics literature this is usually refered to as the moment (Fitzpatrick, 2014).

Here we will introduce a more general form of moment. This will help understand how the fluid equations result from averaging over different moments of the general transport equation (Pécseli, 2012; Fitzpatrick, 2014). The zeroth, first and second order moments are respectively given by:

$$\Phi^0(\mathbf{v}) = m \tag{2.24a}$$

$$\Phi^1(\mathbf{v}) = m\mathbf{v} \tag{2.24b}$$

$$\Phi^2(\mathbf{v}) = m\mathbf{v}\mathbf{v} \tag{2.24c}$$

By integrating the moment functions and the distribution function $\mathcal{F}$, over the velocity space we can retrieve different quantities.

Integrating the zeroth order moment gives the density, if we divide by the mass.

$$n = \frac{1}{m}\int m\mathcal{F}\mathrm{d}\mathbf{v} \tag{2.25}$$

Integrating over the first order moment gives the momentum, if we divide by density.

$$m\mathbf{v} = \frac{1}{n}\int m\mathbf{v}\mathcal{F}\mathrm{d}\mathbf{v} \tag{2.26}$$

We can in fact find the mean of any order moment by integrating the distribution function over $\mathcal{F}$.

$$\langle\Phi^n(\mathbf{v})\rangle = \frac{1}{n}\int \Phi^n\mathcal{F}\mathrm{d}\mathbf{v} \tag{2.27}$$

2.4.2 Transport Equation

By multiplying the moment function, Φ, with the Vlasov equation, eq. (2.23), we obtain the general momentum transport equation (Shu, 2010).

$$\frac{\partial n\langle\Phi^n(\mathbf{v})\rangle}{\partial t} + \nabla\cdot(\langle\Phi^n(\mathbf{v})\mathbf{v}\rangle) = \frac{n}{m}\langle\mathbf{F}_L\cdot\nabla_v\Phi^n(\mathbf{v})\rangle \tag{2.28}$$

This then becomes a conservation equation for the average macroscopic quantity $\langle\Phi\rangle$. By multiplying this equation with the moments the fluid equations can be obtained, see section 2.4.3.

2.4.3 Fluid Equations

From the Vlasov equation and the zeroth, first and second order moments we obtain the fluid equations. The generalized fluid equations are given in eqs. (2.29a) to (2.29c), where the three equations describe the conservation of mass, momentum and energy respectively. The collision term is neglected. We refer to Fitzpatrick (2014), although some notation differ, for the rather involved derivation of these equations.

$$\left(\frac{\partial}{\partial t} + \mathbf{u}_s \cdot \nabla\right) n_s + n_s \nabla \cdot \mathbf{u}_s = 0 \tag{2.29a}$$

$$m_s n_s \left(\frac{\partial}{\partial t} + \mathbf{u}_s \cdot \nabla\right) \mathbf{u}_s = -\nabla p_s - \nabla \cdot \pi + n_s \mathbf{f}_s \tag{2.29b}$$

$$\left(\frac{\partial}{\partial t} + \mathbf{u_s} \cdot \nabla\right) p_s = -\frac{5}{3} p_s \nabla \cdot \mathbf{u}_s - \frac{2}{3}\pi_s : \nabla \mathbf{u_s} - \frac{2}{3}\nabla \cdot \mathbf{q}_s \tag{2.29c}$$

The first equation, eq. (2.29a), is the continuity equation, it states that the total mass in a volume should be preserved. $\mathbf{u}_s$ is the flow velocity and n_s is the number density, i.e. number of particles in a volume. The divergence terms signify change due to the compressability of the fluid and can in many cases be set to 0. The total derivative, i.e. $\left(\frac{\partial}{\partial t} + \mathbf{u}_s \cdot \nabla\right)$ accounts for change in density in a volume taking into account substance exiting and entering. The momentum equation, eq. (2.29b), shows that the fluid momentum change (left hand side) is due to pressure gradients, ∇p_s, visceous forces, $\nabla \cdot \pi$ and external forces, $n_s \mathbf{f}_s$, per unit volume. Lastly we have the energy equation, in its pressure form, which shows that changes to thermal energy, $p = nkT$, are caused by compression, $p_s \nabla \cdot \mathbf{u}_s$, visceous effects $\pi_s : \nabla \mathbf{u_s}$ and heat transport $\frac{2}{3}\nabla \cdot \mathbf{q}_s$. The fluid equations are in general not closeable and adding higher order moments always introduces more unknowns. Due to this one generally uses different closing schemes to make them tractable.

One example is the *local thermodynamic equalibrium* (LTE). A plasma is said to be in LTE if the phase-space distribution is locally Maxwellian. This means the variations in temperature are slow enough that we can neglect heat conduction in the plasma. We can also ignore the viscosity due to there being little local variations to the momentum flow:

$$\mathcal{F}_m = \frac{n}{(2\pi)^{3/2} v_t^3} \exp\left\{-\frac{(v-u)^2}{2v_t^2}\right\} \tag{2.30}$$

Since the viscosity tensor, π, and the heat flux tensor, $\mathbf{q}$ contains odd integrals over the distribution, see Fitzpatrick (2014), they dissappear.

Another ways of closing the set of equations is to assume cold plasma or isothermal plasma. In a cold plasma the temperature is assumed to be 0, this causes the pressure, p, and viscosity, π, to be zero. This can be useful if the velocities of interest far exceed the thermal velocities. In isothermal plasma we assume infinite heat conductivity. This means the temperatures is constant in all space and time and can be useful in describing large scale plasma.

2.5 Langmuir Oscillations

The simplest example of plasma collective phenomena can be obtained from the fluid description. Plasma oscillations, also called Langmuir oscillations, is the basic resulting oscillation that happens as a plasma tries to reach a stable equilibrium, due to a small perturbation of its density. We will use this to show how the fluid equations can be closed for a simple system using assumptions. This is also very suited to test simulations, as we do later in section 5.4.

Here we will consider an one specie plasma fluid consisting of electrons under local thermal equilibrium, LTE. The electron density, n_0 and pressure, p_0, is initally homogenous. The fluid has a vanishing flow, $\mathbf{u}_0 = 0$, and no initial potential gradient $\phi_0 = 0$. See Pécseli (2012) for a more detailed discussion.

A small perturbation of the electron density will cause the electric field to try to restore the equilibrium. When the electrons reach the equilibrium position they will have a kinetic energy and will overshoot. This will cause a new perturbation away from the equilibrium.

Under the LTE conditions the fluid equations simplify to

$$\frac{\partial n_e}{\partial t} + \nabla \cdot (n_e \mathbf{u}_e) = 0 \tag{2.31a}$$

$$m_e n_e \left(\frac{\partial}{\partial t} + \mathbf{u}_e \cdot \nabla \right) \mathbf{u}_e = e n_e \nabla \phi - \nabla p_e \tag{2.31b}$$

$$\left(\frac{\partial}{\partial t} + \mathbf{u}_e \cdot \nabla \right) p_e + \frac{5}{3} p_e \nabla \cdot \mathbf{u}_e = 0 \tag{2.31c}$$

Since this set of equations have more unknowns than equations so we need additional information to close the set. Here we can use the Poisson equation to close it.

$$\epsilon_0 \nabla^2 \phi = e \left(n_e - n_0 \right) \tag{2.32}$$

Now we let a small perturbation, denoted by a tilde, happen to the equilibrium. Since we are free to choose an inertial reference frame, we select one co-moving with the plasma so the inital fluid velocity is $\mathbf{u}_0 = 0$. We also select the reference potential so the initial potential, ϕ_0, is 0.

$$\text{Perturbation} \rightarrow \begin{cases} n_e = n_0 + \tilde{n}_e \\ p_e = p_0 + \tilde{p}_e \\ \mathbf{u}_e = \tilde{\mathbf{u}}_e \\ \phi = \tilde{\phi} \end{cases}$$

Since the pertubation is small, we can say that any part that contains second order terms of perturbation of a quantity will be much smaller than the value of the quantity, $q \gg \tilde{q}\tilde{q}$. So even though we may miss some processes by doing this, we can drop the second order perturbation terms. This process is called linearization (Pécseli, 2012).

Inserting the perturbation and linearizing the equations we get:

$$\frac{\partial \tilde{n}_e}{\partial t} + \nabla \cdot (n_0 \tilde{\mathbf{u}}_e) = 0 \tag{2.33a}$$

$$m_e \frac{\partial \tilde{\mathbf{u}}_e}{\partial t} = e\nabla\tilde{\phi} - \frac{\nabla \tilde{p}_e}{n_0} \tag{2.33b}$$

$$\frac{\partial \tilde{p}}{\partial t} + \frac{5}{3} p_0 \nabla \cdot \tilde{\mathbf{u}}_e = 0 \tag{2.33c}$$

$$\epsilon_0 \nabla^2 \tilde{\phi} = e\tilde{n}_e \tag{2.33d}$$

Then we combine the continuity and energy equations, eq. (2.33a) and eq. (2.33c).

$$\frac{\partial}{\partial t}\left(\frac{\tilde{p}_e}{p_0} + \frac{5}{3}\frac{\tilde{n}_e}{n_0}\right) = 0 \tag{2.34}$$

The perturbed pressure and density are proportional, $\nabla \tilde{p}_e = (5p_0/3n_0)\,\nabla \tilde{n}_e$. Assuming plane wave solutions along the x-axis, the differential operators become $\nabla \to ik$ and $\frac{\partial}{\partial t} \to -i\omega$, we can solve for the dispersion relation.

$$\epsilon(\omega, k) = 1 + \frac{5}{3}\lambda_{De}^2 k^2 - \frac{\omega^2}{\omega_{pe}^2} \tag{2.35}$$

Here the electron Debye length, λ_{De}, and the plasma frequency, ω_{pe}, have been inserted. In the limit of short k, or vanishing temperature, we can see that the dispersion relation simplifies to $\omega = \omega_{pe}$, which we have also derived from the simplified perspective of a cold plasma slab (section 2.1.1). We will use the Langmuir oscillations in benchmarking our numerical model in this thesis.

2.6 Magnetohydrodynamics

In a plasma there are usually several types of species, then it follows that each specie needs its own set of fluid equations. Magnetohydrodynamics, (MDH), is an attempt to simplify this situation by combining it into one electrically conducting fluid. Conventional MHD assumes local thermodynamical equilibrium, negligable electron inertia and quasi-neutrality (Goldston and Rutherford, 1995). This simplifies Maxwell's equations to

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{j} \tag{2.36a}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{2.36b}$$

$$\nabla \cdot \mathbf{B} = \nabla \cdot \mathbf{E} = 0 \tag{2.36c}$$

The MHD fluid can be considered a neutral fluid with a current running through it (Hockney and Eastwood, 1988). The current is described by the conductivity σ and the bulk velocity $\mathbf{v}$ and is given as

$$\mathbf{j} = \sigma \mathbf{v} \tag{2.37}$$

With the condition that the conductivity is high and a finite current eq. (2.36b) becomes

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{v} \times \mathbf{B}) \tag{2.38}$$

Then it remains to close the MHD equations by the continuity and momentum equations, where ρ is the mass density and p is the scalar pressure.

$$\frac{\partial \rho}{\partial t} = \nabla \cdot (\rho \mathbf{v}) \tag{2.39a}$$

$$\rho \frac{\partial \mathbf{v}}{\partial t} = -\nabla p + \mathbf{j} \times \mathbf{B} \tag{2.39b}$$

2.7 Numerical Simulations

The mathematical description of plasma is useful to improve our understanding of the physics, but many problems doesn't fall neatly into convenient assumptions or are untractable. Then we need to turn to experiments and computer simulations to solve them. All of these methods work in symbiosis and are interdependent of each other. Numerical simulations bear many similarities to experiments, but they also have the advantage of being applied to situations that no experiment can reproduce. In addition, physical experiments often have constraints on what can be measured directly, while numerical experiments have all the needed data readily available. Modelling generally needs to be validated against against experiments and has a foundation built upon theory. As the computational resources have improved, more sophisticated simulations have been possible. Plasma simulations vary from fluid descriptions, as MHD codes, to kinetic descriptions, as Particle-in-Cell and Vlasov codes, with hybrid codes inbetween as well. This thesis focuses on the development of a Particle-in-Cell code, but here we will give a brief overview of other modelling approaches as well.

MHD

Magnetohydrodynamical codes solve the one-fluid equations, given in section 2.6, with various approaches and has similarities to Computational Fluid Dynamics. For the fluid equations to be a reasonable description of plasma, the dynamics needs to happen at much larger scales than the Debye shielding length and large time scales (slow phenomena). This approach has been widely used in large scale plasma simulations such as astrophysics, see Hawley and Stone (1995) where an MHD approach is discussed with regards to astrophysical problems, and space physics, see Watanabe and Sato (1990) for an investigation of the solar wind-magnetsphere interaction.

Particle-in-Cell

Particle-in-Cell (PiC) simulations model the particles directly. This has an advantage that few approximations are made, but computational cost increases fast with more particles. Numerical PiC codes have been used extensively study small scale plasma phenomena. Details of the algorithm are given in the next chapter.

Vlasov

Vlasov codes take the kinetic description as the starting point and are used in plasma laser modelling (Bertrand et al., 1990). They have an advantage over PiC in low density zones, where there are often too few particles for PiC. However computational costs are often very high.

Chapter 3

Method

The previous chapter provided the basic concepts in plasma physics and emphasized the need for numerical plasma models. This chapter goes through the theory behind a Particle-In-Cell (PiC) model, with a focus on the multigrid Poisson solver. First there is a general overview of a PiC model and the different building blocks needed. The stability criteria needed in a PiC model are then introduced. Next we go into a more detailed overview of the PINC model, this thesis was a part of building. Then there is an overview of the normalization scheme, in PINC, designed to minimize floating point operations. Domain partitioning as a strategy to parallelize the model is then considered. We continue with providing details on the multigrid solver including boundary conditions.

3.1 Particle-in-Cell

Particle based plasma simulations have been in use since the 1960s, (Verboncoeur, 2005), and the goal of this project was to design, and implement, a massively parallel code, with a focus on the Poisson solver. The aim here is to describe simple and fast PiC model, with good scaling properties, as a baseline and rather add in extra functionality later. Thus, we focus on an electrostatic model and we ignore relativistic effects, which makes it faster and more suited to certain tasks, such as space plasmas and plasma discharges. For an example of a modern relativistic full electromagnetic model see Sgattoni et al. (2015).

The first particle based plasma calculations was done by Dawson (1962) and Buneman (1959). They computed the electrical force directly between the particles leading to a computational scaling of $\mathcal{O}((\#particles)^2)$. Since a large number of particles is often needed, the PiC method seeks to improve the scaling by computing the force on the particles from an electric field instead. The electric field is computed on a grid from the charge distribution obtained from the plasma particle distribution. For an electrostatic model, which this thesis focuses on, this is usually done by solving the Poisson equation, eq. (3.1), over the whole

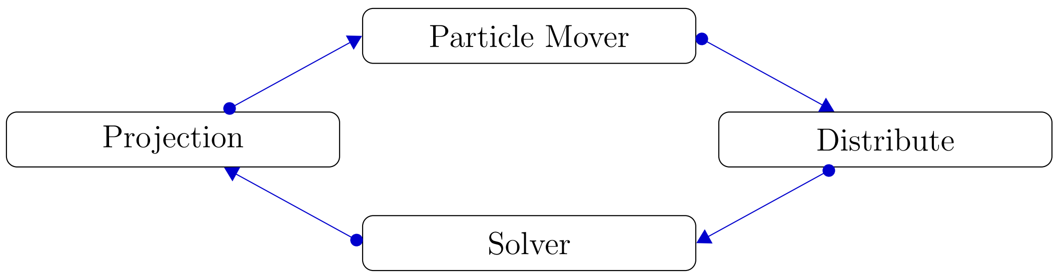


Figure 3.1: Schematic overview of the electrostatic PIC cycle. The mover moves all the particles and updates their velocities. Next the particle charges are distributed to a charge density grid. The solver then obtains the electric field on the grid (and magnetic field in a full electromagnetic model when also the currents are weigthed to the grid). Lastly the field values are projected onto the particles.

domain, Ω.

A PiC model has 4 main components: the mover, the weighting scheme (distribute), the field solver and the projection. See fig. 3.1 for an overview of the PiC cycle. The mover is responsible for moving the particles and updating the velocities of the particles. The input to the solver, in the electrostatic case, is the charge density, ρ, and the output is the potential, ϕ.

$$\epsilon_0 \nabla^2 \phi = -\rho \qquad \text{in} \qquad \Omega \tag{3.1}$$

The distribute module computes a charge distribution on a grid from the particle distribution, This is often done with 1st order interpolation, resulting in second order accuracy. Different order interpolation can also be used. The solver then computes the electric field. Lastly the fields are projected onto the particles.

3.1.1 Movers

The mover in a PiC model has the task of moving all the particles according to the velocity of the particles, as well as the electric and magnetic fields. An often used mover is the *Leapfrog* algorithm (Birdsall and Langdon, 2004), derived from a forward finite difference discretization of the timestep. Then the velocity is shifted half a timestep forward improving the accuracy, with no extra computations needed compared with the *Euler* integration. When the magnetic force also needs to be considered the most used algorithm is the *Boris* algorithm (Qin et al., 2013), employing rotations to effieciently deal with the cross product. While the aforementioned movers are explicit, based on a forward discretization of the time, various projects based on partial and full implicit algorithms also exist (Friedman et al., 1981; Lapenta, 2016). Since implicit algorithms allows a relaxation of the stability restrictions (to be introduced later) they allow the model to resolve closer to the investigated phenomena.

3.1.2 Field Solvers

The Poisson equation, eq. (3.1), is a well known and investigated problem. Here we will mention some advantages and disadvantages of different field solvers before we describe our choice of a multigrid solver. It should also be mentioned that a implicit methods require a different approach, with the preconditioned Jacobian-Free-Newton-Krylov as the most promising approach. Lapenta (2012) can be consulted for an overview.

Spectral Methods

The spectral methods are based on rewriting the problem into a sum of base functions and solving the problem on the basis functions form, see Israeli and Sherman (2005) for an implementation of an spectral Poisson solver. Often the basis functions chosen are sinusoidal, allowing the Fourier Transform to be used. Other basis functions can also be used as in Shen (1994). They are efficient solvers that can be less intricate to implement, but can be inaccurate for complex geometries.

When looking for a solution with a spectral method we first rewrite the problem in the form of the basis functions, in this case sinusoids, which for the three-dimensional Poisson equation would be

$$\nabla^2 \sum A_{j,k,l} e^{i(jx+ky+lz)} = \sum B_{j,k,l} e^{i(jx+ky+lz)} \tag{3.2}$$

where $A_{j,k,l}$ and $B_{j,k,l}$ are the coeffecients of the sinusoids, or otherwise the basis functions. From there we get a relation between the coefficients

$$A_{j,k,l} = -\frac{B_{j,k,l}}{j^2 + k^2 + l^2} \tag{3.3}$$

Then we compute the Fourier transform of the right hand side obtaining the coefficients $B_{j,k,l}$. We compute all the coefficients $A_{j,k,l}$ from the relation between the coefficients. At last we perform an inverse Fourier transform of the left hand side obtaining the solution.

$$\tag{3.4}$$

Finite Element Methods

The finite element (FEM) is a method to numerically solve a partial differential equations (PDE), by first transforming the problem into a variational problem and then constructing a mesh and local trial functions, see Alnæs et al., 2011 for a more complete discussion. FEM is similar to a spectral solver, with the main difference that FEM's basis functions are only locally nonzero.

To transform the PDE to a variational problem we first multiply the PDE by a test function v, then it is integrated using integration by parts on the second order terms. Then the problem is separated into two parts, the bilinear form $a(u,v)$ containing the unknown solution and the test function, as well as the linear form $L(v)$ containing only the test function.

$$a(u,v) = L(v) \qquad v\epsilon\hat{V} \tag{3.5}$$

Next we construct discrete local function spaces of that we assume contain the trial functions and test functions. The function space, $\hat{V}$, often consists of locally defined functions that are 0 except in a close neighbourhood of a mesh point, so the resulting matrix to be solved is sparse and can be computed quickly. The matrix system is then solved by a suiting linear algebra algorithm, before the solution is put together. The FEM method is very suited to tackling problems on complicated grids.

Multigrid

The multigrid method used to solve the Poisson equation and obtain the electric field is a widely used and highly efficient solver for elliptic equations, having a theoretical scaling of $\mathcal{O}(N)$ (Press et al., 1988), where N is the number of grid points. It is very well suited to simple geometries that can easily be translated to coarser problems. The multigrid method is based on iterative solvers such as Gauss-Seidel, section 3.3.3, these have the property that they quickly eliminate local errors in the solution, while far away influences takes longer to incorporate. Multigrid algorithms try to lessen this problem by transforming it into a coarser grid so the distant errors gets solved in fewer iterations. Due to this it needs operators to transfer the problem between coarser and finer grids, which is called restrictors and prolongators. The multigrid algorithm is a topic of this thesis and is described in more detail in section 3.3.

3.1.3 Stability

A PiC model has stability criteria that need to be fulfilled for the model to work correctly. This is caused by the inherent discretization of the problem in a numerical method. Here we will investigate a harmonic oscillator and a wave to find the stability criteria for the time and spatial discretization. A short discussion on the finite grid instability, caused by representing the charge distribution on a mesh, is included as well.

Time Stability Criterion

A one-dimensional harmonic oscillator (e.g. a pendulum in a gravity field) is described by

$$\frac{\partial^2 x}{\partial t^2} = -\omega_0^2 x, \tag{3.6}$$

and has a solution of the form

$$x(t) = Ce^{-i\omega t}. \tag{3.7}$$

Then we replace the temporal derivative with a centered finite difference:

$$\frac{x^{n+\Delta t} - 2x^n + x^{n-\Delta t}}{\Delta t^2} = -\omega_0^2 x^n. \tag{3.8}$$

Inserting the harmonic solution in place of the x^n, $x^{n+\Delta t}$ and $x^{n-\Delta t}$, we obtain:

$$\frac{e^{-i\omega(t+\Delta t)} - 2e^{-i\omega t} + e^{-i\omega(t-\Delta t)}}{\Delta t^2} = -\omega_0^2 e^{-i\omega t}. \tag{3.9}$$

Using Eulers relation, ($e^{-ix} = \cos(x) + i\sin x$), this yields:

$$2\cos(\omega\Delta t) - 2 = -\omega_0 \Delta t, \tag{3.10}$$

which can be rearranged into

$$\sin\left(\frac{\omega\Delta t}{2}\right) = \pm\frac{\omega_0\Delta t}{2}. \tag{3.11}$$

It is clear that when $\frac{\omega_0\Delta t}{2} > 1$, ω has an imaginary component and the numerical solution is unstable. This puts limits on the timestep which should be much smaller then the characteristic timescale in a system, which in our case $\Delta t \ll \omega_{pe}^{-1}$.

Spatial Stability Criterion

A 1-dimensional wave equation is described by:

$$\frac{\partial^2 \varphi}{\partial t^2} = c^2 \frac{\partial \varphi}{\partial x} \tag{3.12}$$

Applying a centered difference

$$\frac{\varphi_j^{n+\Delta t} - 2\varphi_j^n + \varphi_j^{n-\Delta t}}{\Delta t^2} = c^2 \frac{\varphi_{j+\Delta x}^n - 2\varphi_j^n + \varphi_{j-\Delta x}^n}{\Delta x^2} \tag{3.13}$$

Let us assume sinusoidal waves, $\varphi_j^n = e^{i(\omega t - \tilde{k} j \Delta x)}$.

$$\frac{e^{i\omega\Delta t} - 2 + e^{-i\omega\Delta t}}{\Delta t^2} = c^2 \frac{e^{-i\tilde{k}\Delta x} - 2 + e^{i\tilde{k}\Delta x}}{\Delta x^2} \tag{3.14}$$

Which can be rewritten to

$$\cos(\omega\Delta t) = \left(c\frac{\Delta t}{\Delta x}\right)^2 \left(\cos\left(\tilde{k}\Delta x\right) - 1\right) + 1 \tag{3.15}$$

ω needs an imaginary part if $\left(c\frac{\Delta t}{\Delta x}\right) > 1$, this is called the *Courant-Lewy Stability criterion* (Courant et al., 1869). In general for more dimensions it becomes

$$\Delta t \leq \frac{1}{c}\left(\sum_i \Delta x_i^- 2\right)^{-\frac{1}{2}} \tag{3.16}$$

If this condition is not fullfilled, aliasing will appear and the wave propagation will be represented correctly.

Finite Grid Instability

The particles in a PiC simulation move in a continuous space, while they are represented on a discrete grid for the field calculations. This reduction allows a *Finite Grid Instability* to appear, due to a loss of information of representing a number of particles by a few grid points (Lapenta, 2016). This will cause aliasing of properties on a smaller scale than the resolution. The numerical analysis of the instability is complicated and we refer to Birdsall and Langdon (2004) and Hockney and Eastwood (1988) for original works. The instability introduces the following additional constraint on the grid resolution,

$$\Delta x < \varsigma\lambda_D \tag{3.17}$$

The constant ς is of order one and varies according to the details of the implementation. For a Cloud-in-Cell (CiC) scheme, i.e. first order weigthing, $\varsigma \approx \pi$. This means that the stepsize needs to resolve on a much smaller scale than the Debye Shielding length. Violation of this criteria will cause the simulated plasma to unphysically heat, increasing λ_D, untill the condition is fullfilled.

3.2 PINC

Now we will describe the PiC code, PINC, which part of was developed as a part of this work. Some important features of PINC include the ability to change modules without recompiling, full N-dimensional functions as well as special 3-dimensional functions and built in modularity. Presently PINC is not publicly available, contact 4DSpace at UiO for access.

3.2.1 Normalization

For most numerical codes significant computational gain can be achieved relatively easy by smart normalization. With a succesful normalization most of the multiplications with constants will dissappear. Numerical errors due to machine precision are smallest close to unity $\mathcal{O}(1)$ (Hjorth-Jensen, 2016) so we want to work with numbers as close to unity as we can. As a sidebenefit it also makes the code easier to write and cleaner to read. Consider a single particle, with mass m and charge q, in an electric field $\mathbf{E}$. Its equation of motion is then

$$m\frac{\partial^2 \mathbf{r}}{\partial t^2} = q\mathbf{E} \tag{3.18}$$

To compute the acceleration of this particle in completely naive way, would at each point cost 1 multiplications and 1 division, $m/q * E$. If we instead use normalized values the equation could look like this

$$\frac{\partial^2 \tilde{\mathbf{r}}}{\partial t^2} = \tilde{\mathbf{E}} \tag{3.19}$$

where $\tilde{\mathbf{r}}$ and $\tilde{\mathbf{E}}$ is normalized so the dimensionality of the equation works out. Here we have no multiplications and no divisions, but we do have the added task of transforming our variables first to the normalized variables and then back to the original for diagnostics.

Non-dimensionality PINC

A good dimensionalizing strategy is to first remove dimensionality from the fundamental quantities, and then work out the normalizations necessary for the derived quantities.

The fundamental quantities that are involved in our PiC simulation is mass m, position $\mathbf{r}$, time t and charge q. Since we are dealing with plasma it is useful to normalize with Debye-length, λ_D, and electron plasma frequency, ω_{pe}. The normalized quantities are then:

$$\tilde{\mathbf{r}} = \frac{\mathbf{r}}{\lambda_D} \tag{3.20a}$$

$$\tilde{t} = \omega_{pe} t \tag{3.20b}$$

$$\tilde{m} = \frac{m}{m_e} \tag{3.20c}$$

$$\tilde{q} = \frac{q}{e} \tag{3.20d}$$

Next we need the velocity, which is the temporal derivative of the position. This is normalized by transforming the position to the nondimensional position, by eq. (3.20a), as well as changing the temporal derivative to a nondimensional temporal derivate, by eq. (3.20b).

$$\frac{\partial \mathbf{r}}{\partial t} = v \quad \rightarrow \quad \frac{\partial \tilde{\mathbf{r}}}{\partial \tilde{t}} = \tilde{\mathbf{v}} = \frac{\mathbf{v}}{v_{th}} \tag{3.21}$$

Here we have introduced the thermal velocity, mentioned in section 2.1.1, $v_{th} = \lambda_{De}\omega_{pe}$.

Now we will use the Lorentz force to normalize the electromagnetic fields.

$$\frac{\partial \mathbf{v}}{\partial t} = \frac{q}{m}\left(\mathbf{E} + \mathbf{v} \times \mathbf{B}\right) \tag{3.22}$$

Swapping in all the nondimensional values from eqs. (3.20a) to (3.20d) we obtain

$$\frac{\partial(\tilde{\mathbf{v}} v_{th})}{\partial(\tilde{t}/\omega_{pe})} = \frac{(\tilde{q}e)}{(\tilde{m}m_e)}\left(\mathbf{E} + (\mathbf{v}v_{th}) \times \mathbf{B}\right) \tag{3.23}$$

$$\frac{\partial \tilde{\mathbf{v}}}{\partial \tilde{t}} = \frac{\tilde{q}}{\tilde{m}}\left(\frac{e}{v_{th}\omega_{pe}m_e}\mathbf{E} + \tilde{\mathbf{v}} \times \frac{e}{\omega_{pe}m_e}\mathbf{B}\right) \tag{3.24}$$

This suggests that we use the following nondimensional fields

$$\tilde{\mathbf{E}} = \frac{e}{v_{th}\omega_{pe}m_e}\mathbf{E} \quad \text{and} \quad \tilde{\mathbf{B}} = \frac{e}{\omega_{pe}m_e}\mathbf{B} \tag{3.25}$$

The electric field is related to the charge density ρ through Gauss' law.

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0} \tag{3.26}$$

Inserting normalized quantities for $\mathbf{E}$ and the gradient operator

$$\tilde{\nabla} = \left(\frac{\partial}{\partial \tilde{x}}, \frac{\partial}{\partial \tilde{y}}, \frac{\partial}{\partial \tilde{z}}\right) = \lambda_D \nabla$$

$$\frac{1}{\lambda_D}\tilde{\nabla} \cdot \frac{v_{th}\omega_{pe}m_e}{e}\tilde{\mathbf{E}} = \frac{\rho}{\epsilon_0} \tag{3.27}$$

$$\tilde{\nabla} \cdot \tilde{\mathbf{E}} = \frac{\lambda_D e}{v_{th}\omega_{pe}m_e}\frac{\rho}{\epsilon_0} \tag{3.28}$$

This gives the dimensionless charge density

$$\tilde{\rho} = \frac{\rho}{n_0 e} \tag{3.29}$$

Normalization PINC

It should be mentioned that the normalization scheme for PINC was mostly worked out by Sigvald Marholm, and I am mostly repeating his work here. It is still included here to give complete understanding of our PiC implementation. The general aim of the normalization scheme is to reduce the number of floating point operations on the particles. Since there are usually fewer grid points (i.e. values for fields such as ρ and $\mathbf{E}$) than particles in a simulation a multiplication should preferably be done to a field instead of each particle. From now on we will omit the tilde on dimensionless quantities and consider all quantities dimensionless.

Mover

We use the standard Leapfrog algorithm (Birdsall and Langdon, 2004). This has the advantage of second order accuracy and stability for oscillatory motion with the same number of function calls as Euler integration. It should be mentioned that the Leapfrog algorithm preserves momentum, but the energy can drift.
The finite-difference discretization of a leapfrog timestep is given by

$$\frac{\mathbf{r}^{n+1}-\mathbf{r}^{n}}{\Delta t}=\mathbf{v}^{n+\frac{1}{2}} \tag{3.30}$$

By discretizing time as $\bar{t}=t/\Delta t$ and the position and velocity as

$$\bar{\mathbf{r}}=\left(\frac{x}{\Delta x},\frac{y}{\Delta y},\frac{y}{\Delta y}\right) \tag{3.31}$$

$$\bar{\mathbf{v}}=\Delta t(\delta\mathbf{r})^{-1}\mathbf{v} \tag{3.32}$$

we obtain the simpler step equation

$$\bar{\mathbf{r}}^{n+1}=\bar{\mathbf{r}}^{n}+\bar{\mathbf{v}}^{n+\frac{1}{2}} \tag{3.33}$$

Accelerator

The accelerator sets a new velocity to the particles. For a case with no magnetic field the equation of motion becomes

$$\frac{\partial\mathbf{v}}{\partial t}=\frac{q_s}{m_s}\mathbf{E} \tag{3.34}$$

Discretizing the equation and normalizing the electric field as

$$\bar{\mathbf{E}}=\frac{\Delta t^2}{\Delta\mathbf{r}}\frac{q_0}{m_0}\mathbf{E} \tag{3.35}$$

the velocity step for a particle species is given by

$$\bar{\mathbf{v}}^{n+\frac{1}{2}} = \bar{\mathbf{v}}^{n-\frac{1}{2}} + \xi_s \xi_{s-1} \cdots \xi_1 \bar{\mathbf{E}} \tag{3.36}$$

where the specie specific normalization coefficient is:

$$\xi_s = \frac{q_s/m_s}{q_{s-1}/m_{s-1}} \tag{3.37}$$

By applying the cooefficient directly to the electric field this enables us to accelerate each particle with only 1 addition.

Distribute

The interpolation of the charged particles onto a charge density grid is handled by the distribute module. For each particle the charge is distributed to the nearby grid points according to the distance to the grid points. We will not go into the details of the implementation here, only mention the resulting normalization.

The normalized charge density at grid point j is given by adding together the contribution from each particle species

$$\bar{\rho}_j = \sum_i \omega_{ij} \bar{q}_i \tag{3.38}$$

$\bar{q}_i$ is the normalized charge for each particle given by

$$\bar{q}_i = \frac{\Delta t^2}{\Delta V} \frac{q_0}{m_0} q_i \tag{3.39}$$

where $\Delta V = \mathrm{tr}\,(\Delta \mathbf{r})$.

Solver

Due to normalization being already inherent in the charge distribution, $\bar{\rho}_j$, and in the application of the electric field, $\bar{\mathbf{E}}$, to each particle the solver can disregard the normalization.

3.3 Multigrid

Here we will go through the main theory and algorithm behind the multigrid solver, developed as a part of this thesis. See also Press et al. (1988) and Trottenberg et al. (2000) for the general description. This solver is developed for the wholly distributed storage model.

3.3.1 General idea

An iterative solver solves a problem by starting with an initial guess, then it performs an algorithm improving the guess and repeats with the improved guess. The difference between the guess and the correct solution, the residual, does not necessarily converge equally fast for different frequencies. An iterative solver can be very efficient on reducing the local error, while the errors due to distant influence is reduced slowly. A multigrid solver attacks this problem by applying iterative methods on different discretizations of the problem. By solving the problem on very coarse grids the error due to distant influence will be reduced faster, while solving on a fine grid reduces the local error fast. Thus, by solving the problem on both fine and coarse grids the needed cycles will be reduced. To implement a multigrid algorithm we then need algorithms to solve the problem on a grid, restriction and prolongation, see section 3.3.3 and **??**, operators to transfer the problem between grids, as well as a method to compute the residual.

3.3.2 Algorithm

We want to solve a linear elliptic problem,

$$\mathcal{L}u = f \tag{3.40}$$

where $\mathcal{L}$ is a linear operator, u is the solution and f is a source term. In our specific case the operator is given by the Laplacian, the source term is given by the charge density and we solve for the electric potential.

We discretize the equation onto a grid of size q.

$$\mathcal{L}_q u_q = f_q \tag{3.41}$$

Let the error, v_q be the difference between the exact solution,u_q, and an approximate solution, $\tilde{u}_q$, to the difference equation (3.41), $v_q = u_q - \tilde{u}_q$. Then we define the residual as what is left after using the approximate solution in the equation.

$$d_q = \mathcal{L}_q \tilde{u}_q - f_q \tag{3.42}$$

Since $\mathcal{L}$ is a linear operator the error satisfies the following relation

$$\mathcal{L}_q v_q = \mathcal{L}(u_q - \tilde{u}_q) + (f_q - f_q) \tag{3.43}$$

$$\mathcal{L}_q v_q = -d_q \tag{3.44}$$

The system can then be solved directly on this level with a chosen discretization. If we then increase the resolution to obtain a better solution, the system becomes

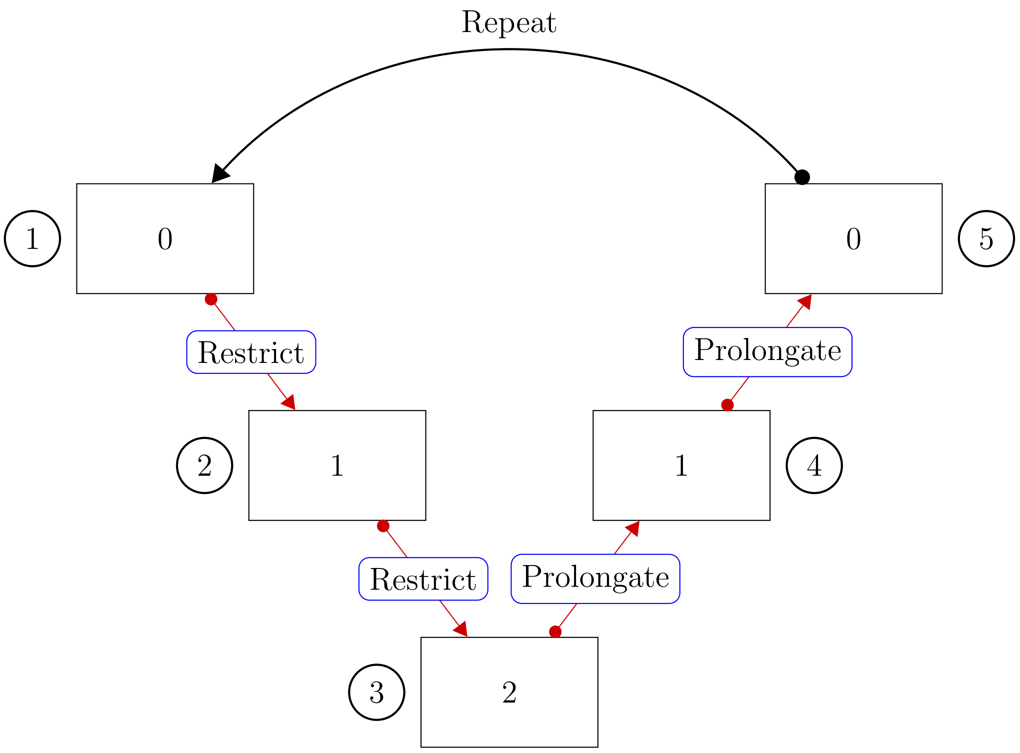


Figure 3.2: Schematic overview of the Multigrid cycle. In a three level MG implementation, there is 5 main steps in a cycle that needs to be considered.

harder to solve. The multigrid method approaches this problem by solving it on several different discretizations levels. We set up a system of nested coarser regular grids, $\mathfrak{T}_0 \supset \mathfrak{T}_1 \supset \cdots \supset \mathfrak{T}_\ell$, where $\mathfrak{T}_0$ is the finest and $\mathfrak{T}_\ell$ is the coarsest grid. Then an iterative solver, which has the property of quickly converging of high frequency errors, i.e. local errors, is used on the finest grid. The remaining error is then transferred to a coarser grid where lower frequency errors are more easily found. The errors found on the coarser levels are then transferred up to the finest level as a correction. To transfer between the discretization coarseness we use restriction, $\mathcal{R}$, and prolongation, $\mathcal{P}$, operators. Due to the fewer grid points the problem is faster to solve on the coarser grid levels than on the fine grid. Applying the restriction and prolongation operators on the grid gives us the grid discretized on a different level:

$$\mathcal{R}d_q = d_{q+1} \qquad \text{and} \qquad \mathcal{P}d_q = d_{q-1} \tag{3.45}$$

fig. 3.2 shows a schematic overview of a 3-level version of a multigrid V cycle. The needed operations on each level is described in greater detail in section 3.3.1.

V-Cycle

The simplest multigrid cycle is called a V-cycle, which starts at the finest grid, goes down to the coarsest grid and then goes back up to the finest grid (Press et al., 1988). First the problem is smoothed on the finest level, then we compute the residual, or the rest after inserting the guess solution in the equation. The residual is then used as the source term for the next level, and we restrict it down as the source term for the next coarser level and repeat until we reach the coarsest level. When we reach the coarsest level the problem is solved there and we obtain a correction term. The correction term is prolongated to the next finer level and added to the solution there, improving the solution, following by a new smoothing to obtain a new correction. This is continued until we reach the finest level again and a multigrid cycle is completed, see fig. 3.2 for a 3 level schematic.

In the following description of the steps in the MG method, we will use ϕ, ρ, d and ω to signify the solution, source, defect and correction respectively. A subscript means the grid level, where 0 si the finest level, while the superscript 0 implies an initial guess is used. Hats and tildes are also used to signify the stage the solution is in, with a hat meaning the solution is smoothed and a tilde meaning the correction from the grid below is added.

The overarching algorithm is shown in algorithm 1

Algorithm 1 Multigrid V cycle

```
if level = coarest then
    Solve                                 $\widehat{\phi}_l = \mathcal{S}(\phi_l, \rho_l)$
    Interpolate correction                $\omega_{l-1} = \mathcal{I}\phi_l$
else
    for each level do
        Smooth                            $\widehat{\phi}_l = \mathcal{S}(\phi_l, \rho_l)$
        Residual                          $d_l = \nabla^2\widehat{\phi}_l - \rho_l$
        Restrict                          $\rho_{l+1} = \mathcal{R}d_l$
        Go down, receive correction       $\omega_l = \mathrm{MG}(\phi_{l+1})$
        Add correction                    $\widetilde{\phi}_l = \widehat{\phi}_l + \omega_l$
        Smooth                            $\phi_l = \mathcal{S}(\widetilde{\phi}_l, \rho_l)$
        Interpolate correction            $\omega_{l-1} = \mathcal{I}\phi_l$
```

At the coarsest level the the problem is solved directly and the correction is propageted upward.

W Cycle

The W-cycle is similar to the V-cycle, with the difference that it spends longer time on the coarser grids, obtaining a better solution before returning to the finest grid.

Full Multigrid

Full Multigrid (FMG) is a multigrid cycle where the source term is known at all the levels. This is achieved usually by an interpolation scheme, or reuse the restriction algorithm, on the source term. Then the problem is first solved at the coarsest level before going up to the finest level.

3.3.3 Smoothing

The multigrid method prefers iterative solvers as smoothers which converges fast for high frequency errors. The low frequency convergence, i.e. the long range interaction, is improved by also solving it with coarser discretization. In this project we also wanted smoothers with good parallel scaling properties. We arrived at using Gauss-Seidel with Red and Black ordering. We ended with 1st order discretization of the Laplacian operator, as a compromise between simplicity in the program and the computational efficiency of hardcoded parts of the algorithms dealing with the Halo, i.e. the ghost layers. It may be that the higher order discretizations will yield better convergence and the project has some plans to expand to incorporate it.

Relaxation methods, such as Gauss-Seidel, work by looking for the setting up the equation as a diffusion equation, and then solving for the equilibrium solution.

So suppose we want to solve the elliptic equation

$$\mathcal{L}u = \rho \tag{3.46}$$

Then we set it up as a diffusion equation

$$\frac{\partial u}{\partial t} = \mathcal{L}u - \rho. \tag{3.47}$$

By starting with an initial guess for what u could be the equation will relax into the equilibrium solution $\mathcal{L}u = \rho$. By using a Forward-Time-Centered-Space scheme to discretize, along with the largest stable timestep $\Delta t = \Delta^2/(2 \cdot d)$, we arrive at Jacobi's method, which is an averaging of the neighbors in addition to a contribution from the source term. By using the already updated values for the calculation of the u^{new} we arrive at the method called Gauss-Seidel which for two dimensions is the following

$$u_{i,j}^{n+1} = \frac{1}{4}\left(u_{i+1,j}^{n} + u_{i-1,j}^{n+1} + u_{i,j+1}^{n} + u_{i,j-1}^{n+1}\right) - \frac{\Delta^2 \rho_{i,j}}{4} \tag{3.48}$$

A slight improvement of the Gauss-Seidel algorithm is achieved by updating every other grid point at a time, by using Red and Black Ordering. This allows a vectorization of the problem and avoids any uneccessary copying.

Jacobian and Gauss-Seidel RB

The main iterative PDE solver, in this version of the multigrid program, is a Gauss-Seidel Red-Black, in addition a Jacobian solver was developed as a stepping stone and for testing purposes. It is a modification of the Jacobian method, where the updated values are used where available, which lead to the convergence increasing by a factor of two (Press et al., 1988).

Our problem is given by $\nabla^2\phi = -\rho$. One way to think of the Jacobian method is as a diffusion problem, and with the equilibrium solution as our wanted solution. If we then discretize the diffusion problem by a Forward-Time-Centralized-Space scheme, we arrive at the Jacobian method, which is shown explicitly below for 1 dimension.

$$\frac{\partial\phi}{\partial t} = \nabla^2\phi + \rho \tag{3.49}$$

Discretizing this we obtain:

$$\frac{\phi_j^{n+1} - \phi_j^{n+1}}{\Delta t} = \frac{\phi_{j+1}^n - 2\phi_j^n + \phi_{j-1}^n}{\Delta x^2} + \rho_j \tag{3.50}$$

The subscript j indicates the spatial coordinate, and the superscript n is the 'temporal' component.

This is numerically stable if $\Delta t/\Delta x^2 \leq 1/2$, so using the timestep $\Delta t = \Delta x^2/2$ we get

$$\phi_j^{n+1} = \phi_j^n + \frac{1}{2}\left(\phi_{j+1}^n - 2\phi_j^n + \phi_{j-1}^n\right) + \frac{\Delta x^2}{2}\rho_j \tag{3.51}$$

Then we arrive at the Jacobian method

$$\phi_j^{n+1} = \frac{1}{2}\left(\phi_{j+1}^n + \phi_{j-1}^n + \Delta x^2\rho_j\right) \tag{3.52}$$

The Gauss-Seidel method uses updated values of ϕ where they are available.

$$\phi_j^{n+1} = \frac{1}{2}\left(\phi_{j+1}^n + \phi_{j-1}^{n+1} + \Delta x^2\rho_j\right) \tag{3.53}$$

Following the same procedure, we get the Gauss-Seidel method for for 2 dimensions:

$$\phi_{j,k}^{n+1} = \frac{1}{4}\left(\phi_{j+1,k}^n + \phi_{j-1,k}^{n+1} + \phi_{j,k+1}^n + \phi_{j,k-1}^{n+1} + \Delta x^2\rho_{j,k}\right) \tag{3.54}$$

and 3 dimensions:

$$\phi_{j,k,l}^{n+1} = \frac{1}{8}\left(\phi_{j+1,k,l}^n + \phi_{j-1,k,l}^{n+1} + \phi_{j,k+1,l}^n + \phi_{j,k-1,l}^{n+1} + \phi_{j,k,l+1}^n + \phi_{j,k,l-1}^{n+1} + \Delta x^2\rho_{j,k,l}\right). \tag{3.55}$$

Here we have implemented a different version of the Gauss-Seidel (similar to a chess board) algorithm called Red and Black ordering, which has conceptual similarities to the leapfrog algorithm, where usually position and velocity is computed at t and $t + (\delta t)/2$. Every other grid point is labeled a red point, and the remaining is black. When updating a red node only black nodes are used, and when updating black nodes only red nodes are used. Then a whole cycle consists of two halfsteps which calculates the red and black nodes seperately.

- For all red points:

$$\phi^{n+1}_{j,k,l} = \frac{1}{8}\left(\phi^n_{j+1,k,l} + \phi^n_{j-1,k,l} + \phi^n_{j,k+1,l} + \phi^n_{j,k-1,l} + \phi^n_{j,k,l+1} + \phi^n_{j,k,l-1} + \Delta x^2 \rho_{j,k,l}\right)$$

- For all black points:

$$\phi^{n+2}_{j,k,l} = \frac{1}{8}\left(\phi^{n+1}_{j+1,k,l} + \phi^{n+1}_{j-1,k,l} + \phi^{n+1}_{j,k+1,l} + \phi^{n+1}_{j,k-1,l} + \phi^{n+1}_{j,k,l+1} + \phi^{n+1}_{j,k,l-1} + \Delta x^2 \rho_{j,k,l}\right)$$

3.3.4 Restriction

The multigrid method (MG) has several grids of different resolution, and we need to convert the problem between the diffrent grids during the overarching the MG-algorithm. The restriction algorithm has the task of translating from a fine grid to a coarser grid. Direct insertion is the simplest way to do this, where coarse grid points correspond directly to its representation on the fine grid. In this implementation we chose to use a half weight stencil, which works well together with the 1 layer Halo, to restrict a quantity from a fine grid to a coarse grid. A higher order restriction algorithm could later be implemented if thought useful. The coarse grid values are obtained by giving half weighting to the fine grid point corresponding directly to the coarse grid point, and gives the remaining half to the adjacent fine grid values, see (3.56), for 1D, 2D and 3D examples:

$$\begin{aligned}
\mathcal{R}_{1\text{D}} &= \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}, \\
\mathcal{R}_{2\text{D}} &= \frac{1}{8}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \\
\mathcal{R}_{3\text{D}} &= \frac{1}{12}\left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 6 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right).
\end{aligned} \tag{3.56}$$

3.3.5 Prolongation

Along with the restriction operator described in the previous subsection, we also need prolongation operator to go from a coarse grid to a finer grid. As in the restriction operator, direct insertion is the simplest algorithm. Here we will use bilinear interpolation, as advised in Trottenberg et al. (2000), for two dimensions and trilinear interpolation for 3 dimensions. In the bilinear interpolation a seperate linear interpolation is done in the x- y- and z-directions, then those are combined to give a result on the wanted spot. The same concept is expanded to give trilinear interpolation. The two and three dimensional stencils are given in (3.57)

$$\begin{aligned}\mathcal{P}_{2\text{D}} &= \frac{1}{4}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}\\ \mathcal{P}_{3\text{D}} &= \frac{1}{8}\left(\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}, \begin{bmatrix}2 & 4 & 2\\ 4 & 8 & 4\\ 2 & 4 & 2\end{bmatrix}, \begin{bmatrix}2 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}\right)\end{aligned} \tag{3.57}$$

3.3.6 Grid Structs and Partitioning

In this section our overall parallelization strategy is discussed as well as the storage needs for the grids.

Data structures

The fields and quantities in PINC are discretized on a 3-dimensional grid. In the multigrid calculation several grids of varying spatial coarseness are used. So it will be useful for us to organize the data so that we have the grid stored as an independent structure available for the program, while the multigrid part uses an extended version where it also has access to different subgrids of different coarseness. Each multigrid struct will have an array of different subgrids, where the first is a pointer to the fine grid used in the rest of the calculations, this makes it easy to select a grid level to perform an algorithm on.

Domain partitioning

We have chosen to divide the physical domain onto different processors so that each takes care of a physical subdomain. This is known as Domain Partitioning, and suits our distribution algorithm as well as the multigrid method, see section 3.5.1 for the consequences of Domain Partitioning for multigrids. Since each subdomain only needs to store the particles, and grids, on its physical subdomain the model can be upscaled in principle without any additional need for memory storage, by adding more processors. The subdomains are dependent on each other and we need some communication between them, which we solve by letting each subdomain also store the edge of the neighboring subdomain. Depending on the boundary conditions it could also be useful to store an extra set of values on the outer domain boundary as well, which will be called ghost points, N_G. The extra grid points due the overlap between the subdomains we will call overlap points, N_O. Let us for simplicity consider a regular domain, with equal size in all dimensions, with N grid points per dimension, d and consider how many grid values we need to store as a singular domain and the grid values needed when it is divided amongst several processors. Such a 2 dimensional case is depicted in fig. 3.3.

3.3.7 Singular domain

In the case where the whole domain is worked on by one process we need N^d to store the values on the grid representing the physical problem, in addition we see that we also need to store values for the ghost points along the domain boundary. Given that we have one layer of ghost points on all the boundaries, and there is 2 boundaries per dimension, the total number of ghost points is given by $N_G = 2dN$. Since there is only 1 domain we don't need to account for any overlap between subdomains and the total number of grid points we need to store is:

$$N_{Tot} = N^d + N_G + N_O = N^d + 2dN^{d-1} \tag{3.58}$$

For the 2 dimensional case, shown in fig. 3.3, that adds up to $N_{Tot} = 8^2 + 2 \times 4 \times 8 = 128$.

3.3.8 Several subdomains

In the case where we introduce several subdomains, in addition to storing the grid values and the ghost points we also need to store an overlap between the subdomains. If we take our whole domain Ω and divide in up into several small domains Ω_S, the smaller domains only takes a subset of the grid points. For simplicity, and for equal load on processors, we let the subdomains as well be regular, with the whole domain being a multiple of the subdomains. Our whole domain has N grid points in each direction, if we then divide that domain into $\#\Omega$ domains, then each of those subdomains will have $N_S = N^d/\#\Omega$ grid points. Each of those subdomains will also need values representing the ghost points and overlap from the neighboring nodes. A boundary of a subdomain will either have overlap points, or ghost points, not both at the same time so for each boundary we need to 1 layer, N_S^{d-1}. Each subdomain will have 2 boundaries per dimension since we have regular subdomains. The total number of grid points needed per subdomain is then

$$N_{Tot,S} = N_S^d + (N_G + N_O) = N_S^d + 2dN_S^{d-1} \tag{3.59}$$

while the total number of grid points is

$$N_{Tot} = \#\Omega N_{Tot,S} \tag{3.60}$$

For the 2 dimensional case discussed earlier we need $N_{Tot,S} = 4^2 + 2 \times 2 \times 4^1 = 32$. Since the effect of the subdomain boundaries increases the coarser the grid is, we should not let the coarsest multigrid level be too small. We also don't need the spatial extent of the grid to be equal on all sides, but it was done here to keep the computations simple.

3.4 Boundary conditions

A simulation must necessarily have finite extent, we need to employ boundary condtions to deal with the edges of the simulation. Here we will go through 3 different schemes corresponding to periodic boundaries, depicted in fig. 3.4, Dirichlet conditions and von Neumann conditions. Periodic conditions used when we want to simulate an infinite plasma sheet. It is usually used when the plasma sheet is of a much larger extent than the length scale of the phenomena we want to investigate, or when the investigated dynamics happen away from the edges. Dirichlet conditions are useful when the voltage on the edge of the simulation can be known beforehand, as it is often in laboratory experiments. When the electric field, or alternatively gradient of the voltage, along the edges is known von Neumann conditions should be used. The boundary conditions must also be coupled with fitting boundary conditions applied to the particles in a full PiC simulation. Particle conditions include periodic, bouncing and absorbing boundaries. To maintain the design aim of inherent modularity of our PINC model, the boundary conditions are defined using ghost points, avoiding different discretization stencils at the boundary. This reduces the complexity the smoothers, makes the boundary conditions easier to implement and opens the possiblity of using them with other solvers.

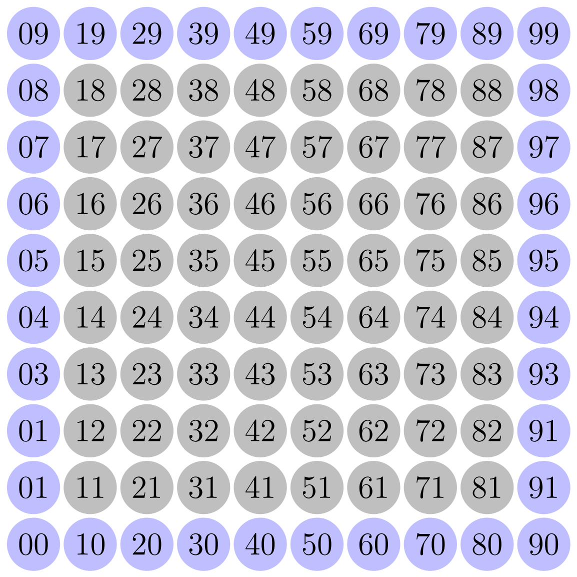
3.4.1 Periodic Boundaries

With periodic boundary conditions we want the boundary on one side to be equal to the field on the other side of the plasma. For the 1D case, see fig. 3.4b, this can be written as

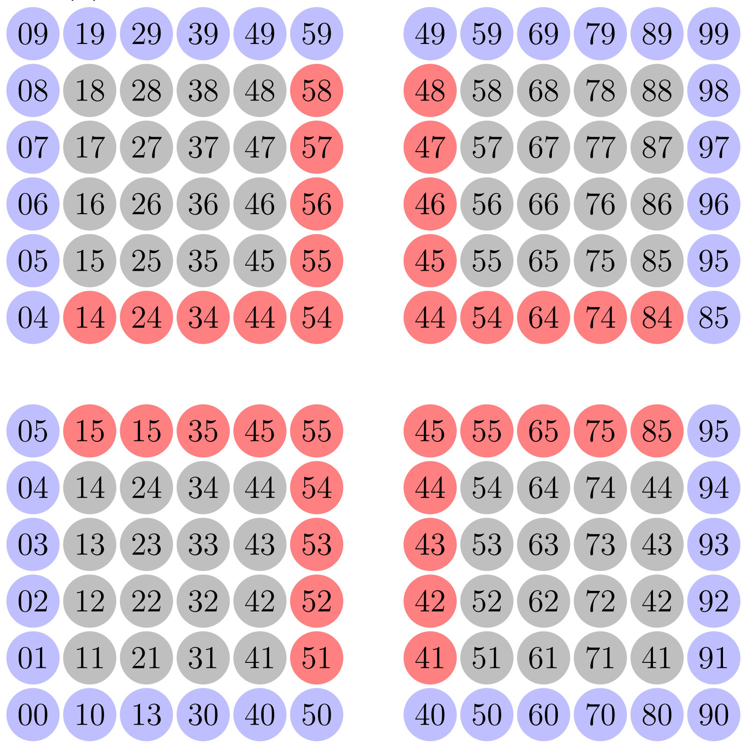
$$\nabla^2\phi = -\rho \qquad \Omega = [0, L] \tag{3.61}$$

With boundaries

$$\phi(0) = \phi(L) \tag{3.62}$$

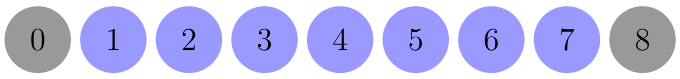


(a) The grid points needed for an 8×8 domain.

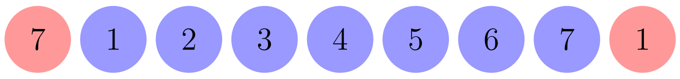


(b) The 8×8 grid divided into 4 subdomains

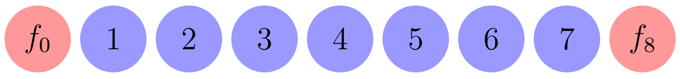
Figure 3.3: Each circle in the figures represents 1 grid point, and the first number is the column while the second is the row. The grey colour represents physical space the computational node works on, the blue color is the outer grid points for boundary conditions and the red colour is the overlapping grid points.



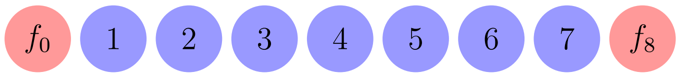
(a) A 1 dimensional domain with grid points before the boundary conditions has been applied. The numbers denote the indexes for the values in the grid array. The blue grid points, $(1-7)$ represents the true grid and the grey points, (0 and 8), is the ghost cell values. The concepts illustrated here can easily be expanded to more dimensions.



(b) Here periodic boundary conditions has been applied to the 1D domain above. Here and in the following the red color means a point has been changed. The ghost points at the edges has been set to equal the true grid points at the opposite edge giving periodic boundary conditions.



(c) For Dirichlet boundary conditions we have predetermined values along the edge, this are here represented as the ghost cells being set to a given value defined by $\partial\Omega_i$. The boundary function can be as simple as setting everything along the edge to a constant, but it could also be a spatially and time varying function. It is also possible to let it correspond to input given by coupled computer model.



(d) Von Neumann boundary conditions specifies what the derivative is on the edge. To achieve that we set the ghost points to a specified value that will give the wanted derivative when a finite difference method swipes over the point. For the left side the function should be set to $f_0 = u_2 - 2\Delta x A$ and to $f_8 = u_6 - 2\Delta x A$ for the right side. Here A is the predetermined values that the derivative should correspond to.

Figure 3.4: An overview of 3 boundary conditions applied to a 1D domain.

Here we should note that this is very similar to what happens between the subdomains in a Domain Partitioning parallelization scheme, so often the same algorithm and code can be reused to achieve periodic boundary conditions.

Solutions for the continuous problem with periodic boundary conditions exist only if the *compatibility condition* (Trottenberg et al., 2000)

$$\int_\Omega \rho \mathrm{d}\mathbf{x} = 0 \tag{3.63}$$

is held. This means that the total charge in the domain must be zero, which is often true in plasma due to quasi-neutrality.

To ensure a unique discretized solution, one needs to set the integration constant. This can be done by setting a *global constraint* on the solution

$$\sum_\Omega \phi = 0 \tag{3.64}$$

3.4.2 Dirichlet Boundaries

With Dirichlet conditions the boundaries of the potential are known and given by a function, $\partial\phi = f$. Then a 1D problem, fig. 3.4c , is represented by

$$\nabla^2\phi = -\rho \qquad \Omega = [0, L] \tag{3.65}$$

with boundaries

$$\phi(0) = f(0), \qquad f(L) = f(L) \tag{3.66}$$

3.4.3 von Neumann Boundaries

Let us assume that we know the gradient of the potential along the boundary, $\nabla\phi_{\partial\Omega} = f$. This is often used in hydrodynamics to represent reflecting boundaries. Then our 1D example problem will look like

$$\nabla^2\phi = -\rho \qquad \Omega = [0, L] \tag{3.67}$$

with boundaries

$$\partial\phi(0) = f(0), \qquad \partial\phi(L) = f(L) \tag{3.68}$$

The boundary condition is then stated as a gradient and we need to approximate it to ϕ to use it in the Poisson equation, eq. (3.67). We do this by the 2nd order central difference to the gradient.

$$\frac{\partial\phi(x)}{\partial x} = \frac{\phi(x+\Delta x) - \phi(x-\Delta x)}{2\Delta x} = f(x) \tag{3.69}$$

At the lower boundary, $x = 0$, this can then be written as

$$\phi(-\Delta x) = \phi(\Delta x) - 2\Delta x f(0) \tag{3.70}$$

and at the upper

$$\phi(L + \Delta x) = \phi(L + \Delta x) - 2\Delta x f(L) \tag{3.71}$$

With our discretization, where the internal cell sizes are 1, the $\phi(-\Delta x)$ corresponds directly to a ghost cell. So we can implement the von Neumann boundary conditions easily by setting the ghost cells equal to eqs. (3.70) and (3.71), see fig. 3.4d. This scheme completely avoids any modification of the smoother stencils at the boundaries.

3.4.4 Boundaries in Multigrid

The multigrid algorithm solves the problem on several discretization levels. Due to this we need to represent the boundaries on the coarser grid levels as well.

Periodic

Since the periodic boundary conditions can be thought of to set the domain next a copy of itself the boundary treatment will remain equal on the coarser grids. It should also be mentioned that the *global constraint*, eq. (3.64), only needs to be set on the coarsest grids to achieve good convergence rates (Trottenberg et al., 2000).

Dirichlet

The Dirichlet condition specifies the potential at the boundaries. Since the conditions should apply to the problem at all coarseness levels, we need a restriction operator specific to the boundary.

The easiest boundary restriction operator is direct injection, letting each coarse grid point correspond to a grid point on the boundary of the finer grid. This is often sufficient, especially in the case of spatially constant boundaries. If the boundaries are constant in time, the computing time can be saved by computing the boundaries for all levels once at the start of the simulation.

If the boundaries are more complicated first or second order interpolation could be used to restrict them.

von Neumann

The von Neumann conditions are dependent on the next to outermost grid point, i.e. grid point 2 in fig. 3.4d on the lower side, because of this they will have to be recomputed each time they are used. But still the function f should be restricted seperately from the finer grid restriction.

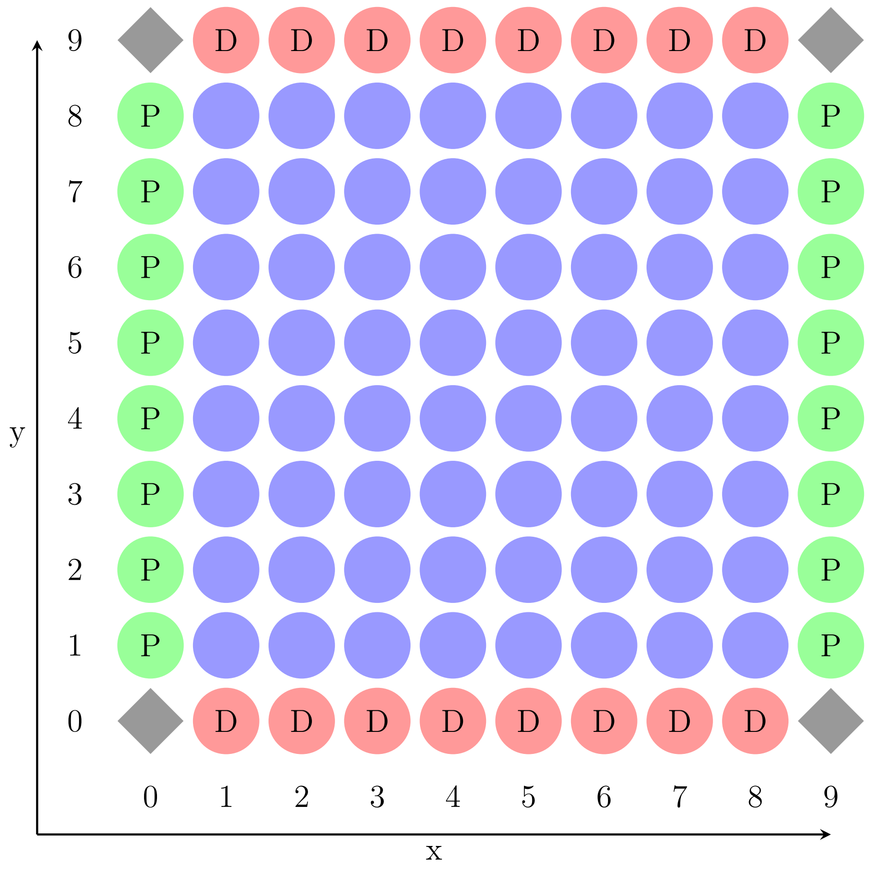


Figure 3.5: This is a 2D domain with mixed boundary conditions, along the x-axis there are periodic boundary conditions, green ghost points, and the y-axis there are Dirichlet boundary conditions, red ghost points. If the smoothers are using a 5 point discretization stencil the corners, grey diamonds, are neglected when computing the inner, i.e. true, grid.

3.4.5 Mixed Conditions

All the boundary conditions can be implemented with the use of ghost cells, this enables the use of the 7-point in 3D (and 5 pint in 2D), stencil for the smoothers. This greatly simplifies mixed boundary conditions, the boundaries are set on the ghost layer of the grid seperately, then the smoother runs. Here it should be noted that the boundaries need to be reset for each halfcycle in GS-RB.

As can be seen in fig. 3.5 we do not have to care for the corners of the domain, as long as we are using a 7-point stencil. This allows us to not need to care of which boundary conditions takes precedence when they clash. For a higher order, or different, stencils the corner ghost cells may be important and the mixing of boundary conditions need to be given extra care.

3.5 Parallelization

For the parallelization of an algorithm there are two obvious issues that need to be addressed to ensure that the algorithm retains a high degree of parallelization; communication overhead and load imbalance (Hackbusch and Trottenberg, 1982). Communication overhead means the time the computational nodes spend communicating with each other, if that is longer than the actual time spent computing the speed of the algorithm will suffer, and load imbalance appears if some nodes need to do more work than others causing some nodes to stand idle.

Here we will focus on multigrid of a 3D cubic grid, where each grid level has half the number of grid points. We will use grid partitioning to divide the domain, GS-RB (Gauss-Seidel Red-Black) as both a smoother and a coarse grid solver.

We need to investigate how the different steps: interpolation, restriction, smoothing and the coarse grid solver, in a MG algorithm will handle the parallelization.

3.5.1 Grid Partition

There are several well explored options for how a multigrid method can be parallized, for example Domain Decomposition (Arraras et al., 2015), Algebraic Multigrid (Stüben, 2001), see Chow et al. (2006) for a survey of different techniques. Here we will focus on Geometric Multigrid (GMG) with domain decomposition used for the parallelization, as described in the books Trottenberg et al. (2000) and Hackbusch and Trottenberg (1982).

With the domain partitioning we divide the grid into geometric subgrids, then we can let each processes handle one subgrid each. As we will see it can be useful when using the GS-RB smoothing, as well as other parts of a PiC program, to extend the subgrids with layers of ghost cells. The GS-RB algorithm will directly need the adjacent nodes, in its neighbour subdomain.

3.5.2 Distributed and accumulated data

One possible strategy to implement a parallel multigrid solver is to keep some of the quantities distributed, i.e. they are only stored locally on the local computational nodes, while the other are accumulated and shared between the nodes. In PINC the accumulated quantities are all stored only locally, but the processors obtain the needed information from the nearby subdomains when needed. Below follows an overview of which quantities need only be accessed locally and which need a global presence during a parallel execution of the code.

u	solution (Φ)
w	temporary correction
d	defect
f	source term (ρ)
$\mathcal{L}$	differential operator
$\mathcal{P}$	prolongation operator
$\mathcal{R}$	restriction operator
$\mathbf{u}$	Bold means accumulated vector
$\tilde{\mathbf{u}}$	is the temporary smoothed solution

- Accumulated quantities: $\mathbf{u}_q$, $\hat{\mathbf{u}}_q$, $\tilde{\mathbf{u}}_q$, $\hat{\mathbf{w}}_q$ $\mathbf{w}_{q-1}$, $\mathcal{P}$,$\mathcal{R}$
- Distributed quantities: f_q, d_q, d_{q-1}

To avoid the accumulated quantities, which can cause memory issues, we have therefore gone for a strategy where all the quantities are only locally distributed. Instead of gathering all of the accumulated quantities, each of the subdomains gathers only the needed part of the quantities from its neighboring subdomains. With this strategy the local memory needs should not increase as the number of processors grow.

3.5.3 Smoothing

We have earlier divided the grid into subgrids, with overlap, as described in subsection 3.5.1 and given each processor responsibility for a subgrid. We broadly follow the algorithm in M. F. Adams, 2001. A GS-RB algorithm start with a guess, or approximation, of the solution $u^n_{i,j}$. Then we will obtain the next iteration by the following formula, for a 2D case,

$$u^{n+1}_{i,j} = \frac{1}{4}\left(u^n_{i+1,j} + u^{n+1}_{i-1,j} + u^n_{i,j+1} + u^{n+1}_{i,j-1}\right) - \frac{\Delta^2 \rho_{i,j}}{4} \tag{3.72}$$

We can see that for the inner subgrid we will have no problems since all the surrounding grid points are known. On the edges we will need the adjacent grid points that are kept in the other processors. To avoid the algorithm from asking neighboring subgrids for adjacent grid points each time it reaches a edge we instead update the entire neighboring column at the start. So we will have a 1-row overlap between the subgrids, that needs to be updated for each iteration.

3.5.4 Restriction

For the transfer down a grid level, to the coarser grid we will use a half weighting stencil. In two dimensions it will be the following

$$\mathcal{R} = \frac{1}{8}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.73}$$

With the overlap of the subgrids we will have the necessary information to perform the restriction without needing communication between the processors (Hackbusch and Trottenberg, 1982).

3.5.5 Interpolation

For the interpolation we will use a bilinear interpolation stencil, which for 2 dimensions is:

$$\mathcal{P} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.74}$$

Since the interpolation is always done after GS-RB iterations the outer part overlapped part of the grid updated, and we can have all the necessary information. We note again that the is N-dimensional and the stencils are different for the 1D, 2D and 3D cases. However for clarity we present here a 2D case.

3.5.6 Scaling

Volume-Boundary effect

While a sequential MG algorithm has a theoretical scaling of $\mathcal{O}(N)$ (Press et al., 1988), where N is the number of grid points, an implementation will have a lower scaling efficiency due to interprocessor communication. We want a parallel algorithm that attains a high speedup with more added processors P, compared to sequential 1 processor algorithm. Let $T(P)$ be the computational time needed for solving the problem on P processors. Then we define the speedup $S(P)$ and the parallel efficiency $E(P)$ as

$$S(P) = \frac{T(1)}{T(P)} \qquad E(P) = \frac{S(P)}{P} \tag{3.75}$$

A perfect parallel algorithm would the computational time would scale inversely with the number of processors, $T(P) \propto 1/P$ leading to $E(P) = 1$. Due to the necessary interprocessor communication that is generally not achievable. The computational time of the algorithm is also important, if the algorithm is

	Cycle	Sequential	Parallel
MG	V	$\mathcal{O}(N)$	$\mathcal{O}(\log N \log \varepsilon)$
	W	$\mathcal{O}(N)$	$\mathcal{O}\left(\sqrt{N}\right)$
FMG	V	$\mathcal{O}(N)$	$\mathcal{O}\left(\log^2 N\right)$
	W	$\mathcal{O}(N)$	$\mathcal{O}\left(\sqrt{N} \log N\right)$

Table 3.1: The parallel complexities of sequential and parallel multigrid cycles.

very slow but has good parallel efficiency it is often worse than a fast algorithm with a worse parallel efficiency.

The parallel efficiency of an algorithm is governed by the ratio between the time of communication and computation, T_{comm}/T_{comp}. If there is no need for communication, like on 1 processor, the algorithm is perfectly parallel efficient. In our case the whole grid is diveded into several subgrids, which is assigned to different processors. In many cases the time used for computation is roughly scaling with the number of interior grid points, while the communication time is scaling with the boundaries of the subgrids. If a local solution method is used on a local problem it is only the grid points at the boundary that need the information from grid points on the other processors. Since the edges have lower dimensionality than the inner grid points, the boundary grows slower than the inner domain. As the size of the subdomains is increasing, the computational time increases faster than the time for communication. This causes a parallel algorithm to often have higher parallel efficiency on a larger problem. This is called the Boundary-Volume effect (Trottenberg et al., 2000).

Parallel complexity

The computational complexity of sequential and parallel MG cycles are calculated in Hackbusch and Trottenberg (1982) and are shown in table 3.1. In the table we can see that in the parallel case there is a substantial increase in the complexity in the case of W cycles compared to V cycles. In the sequential case the change in complexity when going to a W cycle is not dependent on the problem size, but it is in the parallel case.

3.5.7 Updating the Halo

All of the subgrids have a halo of ghostslayers around it, which is used to simplify boundary conditions and subdomain communication. Each computational node represents a subdomain of the whole, with the neighboring node being the boundary. So between two subdomains each subdomain updates the boundary according to the neigbouring subdomain. In addition the halo is used to facilitate boundary conditions on the whole domain. For some of the grid operators the ghost are not used, while some of them need updated values. All of the iterative solvers, that are used for smoothing, need updated values of the solution, ϕ. The prolongation and residual operators need updated values for the solution ϕ, and the restrictor need updated residual values, ρ, as long as direct insertion is not used. We also need to take into account that the smoothers outputs an updated halo for ϕ, to avoid unnecessary communication between the processors.

Chapter 4

Implementation

Presently PINC is not publicly available, contact 4DSpace at UiO for access[1]. The model has been implemented in the language c, and is the common git repository within 4DSpace strategic Research Initiative at the University of Oslo. The module for the Poisson solver is about 2000 and is a part of the PINC code which is still under development. For this reason the code is not explicitly included in this thesis. However once the code is completed it will be released publicly.

4.1 Implementation

In general there are 4 different quantities, that we need to keep track of: the source, the solution, the residual and the correction. On each grid level the residual is computed, then it is set as the source term for the next level and then it is not used anymore. The correction,which is the improvement to the finer grid, is only used when going to a finer grid. Due to this we can save some memory by letting the correction and the residual share the same memory, so both are stored in the same structure. There is a **regular** as well as a **recursive** implementation of the V-cycle, see fig. 4.1. The functions take the current level, the bottom of the cycle as well as the end point of the cycle. Thus, several different cycles can be built from the functions. A W cycle, see section 3.3.2, can be built from a V cycle that starts at the finest level and stops at a mid level, and then a new V-cycle is started at the mid level that ends at the finest level. A full multigrid algorithm (FMG) can also be implemented by first restricting the original source term down to the coarsest level and then run a V-cycle that ends at the finest level. The choice between different cycles can be selected in the input file of PINC, and more type of cycles can easily be constructed if needed.

The **regular** V cycle algorithm is quite straightforward, first it computes the residual and restrict it down to the bottom level, then it solves it directly on the

[1]www.mn.uio.no/4dspace

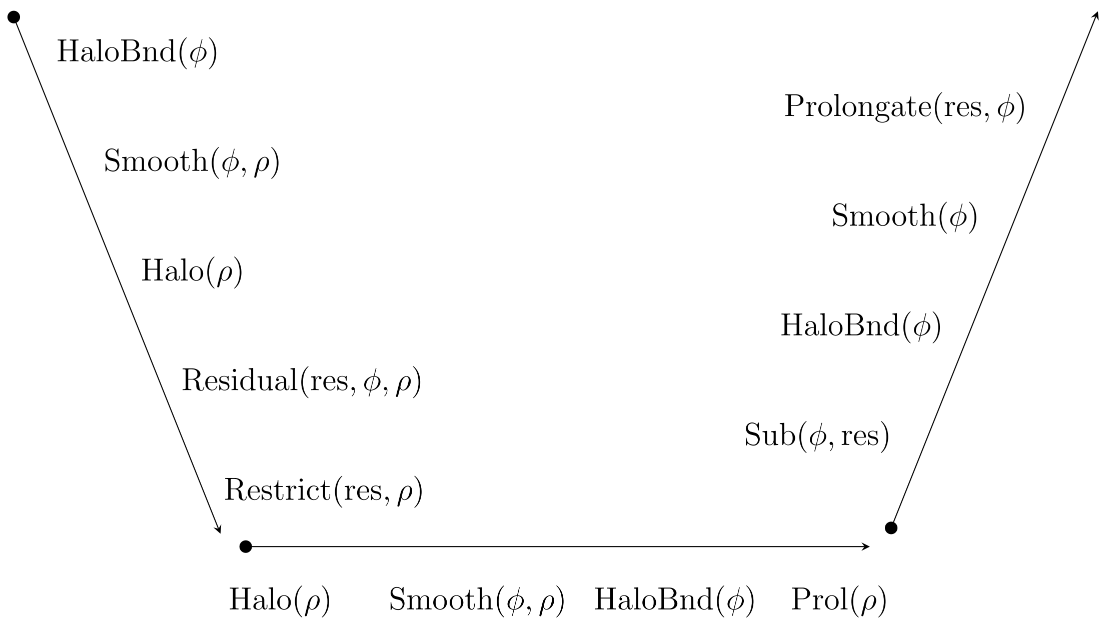


Figure 4.1: The functions used in 2 grid deep multigrid. The algorithm follows the steps needed for a complete V cycle.

bottom level. Then the correction is brought up and improved through the grid up to the top level. See **??** for an example code.

The **recursive** algorithm uses an algorithm similar to the one described in section 3.3.2. First it computes the steps necessary so the grid below has an updated source term, then it calls itself on a lower level. After receiving the correction from the lower level it is improved and sent to the level above. If the function is at the bottom level, it solves the problem directly and sends the correction up.

It should also be mentioned that there are both 3-dimensional algorithms, as well as a set of recursive N-dimensional algorithms that are built to handle $2-$ and $1-$dimensional simulations. The $N-$dimensional algorithms were easier to maintain than seperate algorithms for 1 and 2 dimensions.

4.2 Restriction

In our implementation we first cycle through all of the true coarse grid points, then the two main tasks are to find the specific fine grid point corresponding to the specific coarse grid point, and finding the indexes of the fine grid points surrounding the grid point.

Since the values in both grids are stored in the first order lexicographical array, we should treat the grid points in the same fashion, so the values are stored close to each other in the array. The first dimension is treated first, then the next dimension is incremented followed by treating the first dimension again, then increment the next and so on. The finer grid has twice the resolution of the coarser grid one level below, so for each time the coarse grid index is incremented, the fine grid index is incremented twice.

Along the x-axis each incrementation is by the number of values stored in the grid, which for scalars is 1, and 2 for the fine index. The fine index will in addition need to skip 1 row each time, each time the y-axis is incremented, due to the finer resolution and 1 layer each time the z-axis is incremented.

At the edges of the grid we have ghost layers, which have equal thickness for both the grids, so the coarse grid needs to be incremented over the ghost values, in the x-direction, each time y is incremented. When z is incremented the index needs to skip over a row of ghost values. The fine index follows the same procedure as the coarse index when dealing with the ghost layers.

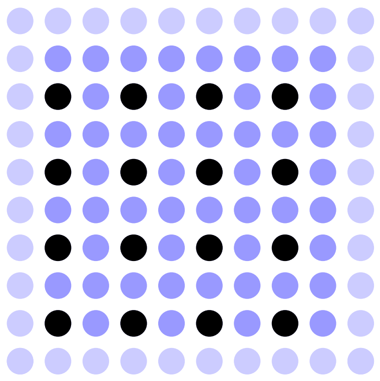
When correct fine grid index is found, corresponding to a coarse grid index, the stencil needs to be applied around that grid value. This is done by first calculating the index of the first coarse and find indexes and setting the correct indexes for the surrounding grid values, then the surrounding grid indexes can be incremented exactly as the fine grid index and they will keep their shape around the fine grid index. Since our indexes in x, y and z are labeled j,l,k, the next value along the x-axis is labeled 'fj' and the previous is labeled 'fjj'. The coarse and fine grid indexes are label 'c' and 'f' respectively.

4.3 Prolongation

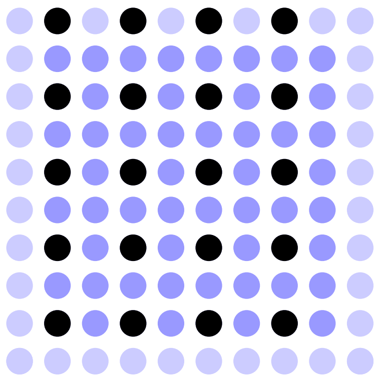
The algorithm implemented for the interpolation is based on the method, described in Press et al. (1988), has the following steps, which is also shown for a 2D case in fig. 4.2.

1. Direct insertion: Coarse$\rightarrow$ Fine
2. Interpolation on highest Dimension: $f(x) = \frac{f(x+h)+f(x-h)}{2h}$
3. Fill needed ghosts.
4. Interpolation on next highest Dimension

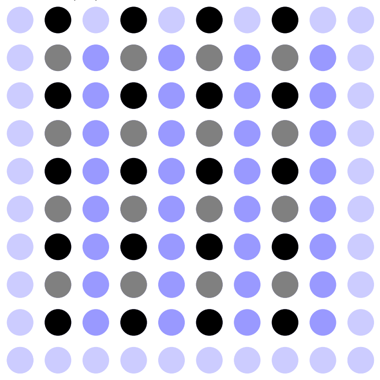
The interpolation should always first be done on the highest dimension, because the grid values are stored further apart along the highest axis in the memory, and the each succesive interpolation needs to apply to more grid points.



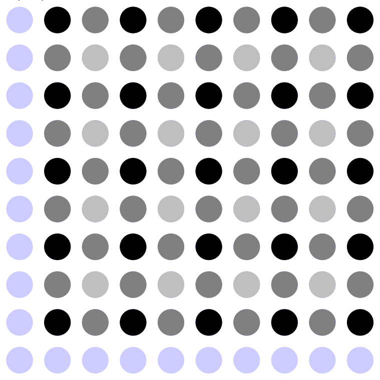
(a) Direct insertion



(b) Swapping X-Dim ghosts



(c) Y-swipe



(d) Swapping Y-ghostlayer and X-swipe

Figure 4.2: This figure shows the steps in computing the prolongation stencil in an $[8 \times 8]$ grid. First a direct insertion from the coarse grid is performed (4.2a), followed be filling the ghostlayer perpendicular to the x-axis from the neighbouring grid (4.2b). Then a swipe is performed in the y-direction filling the grid points between, taking half the value from the node above, and half from the node below (4.2c). Then a ghost swap is performed before doing a swap in the x-direction (4.2d).

4.4 Smoothers

Jacobi's method

The implementation of the Jacobian algorithm, which was described in section 3.3.3, is straightforward, but it has the downside of slow convergence and bad smoothing properties, in addition to the need of additional grid values. When ϕ_i^{n+1} is computed we need access to the previous value ϕ_{i-1}^n, so either the previous values need to stored seperately, or ϕ_i^{n+1} can be computed on a new grid and then copied over after completing the cycle. In this implementation we computed the solution on a temporary grid and then copied over, since it was mostly for debugging purposes in the early development, and efficiency was not a concern.

The computation is done by starting at index $g = 0$, computing the surrounding grid indexes, $gj, gjj, ...$, where gj is the next grid point along the x-axis, and gjj is the previous value. Then the entire grid is looped trough over, incrementing both g and the surrounding grid indexes $gj, gjj, ...$. The computation on the ghost layers will be incorrect but those will be overwritten when swapping halos. As mentioned Jacobi's method was used for development and testing, but in the final version we use the Gauss-Seidel Red and Black ordering.

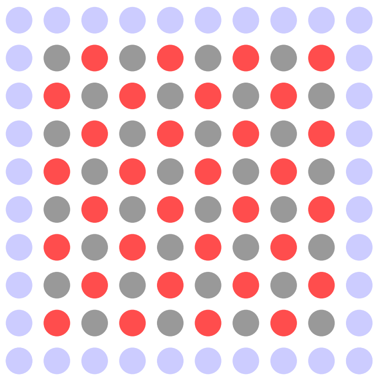
Gauss-Seidel Red and Black

In the implementation of Gauss-Seidel algorithm we use a clever ordering of the computations, called Red and Black ordering, both to increase the smoothing properties of the algorithm as well as avoiding creating a temporary grid to store ϕ^{n+1} in. Every grid point where the indexes sum up to an even number is labeled a red point and the odd index groupings are labeled black points, see fig. 4.3a. Then each red point is directly surrounded by only black points and vica versa.

A cycle is then divided into 2 halfcycles, where each halfcycle computes ϕ^{n+1} for the red and black points respectively.

1. for(int c = 0; c < nCycles; c++)
    - Cycle through red points and compute ϕ^{n+1}
    - Swap Halo
    - Cycle through black points and compute ϕ^{n+1}
    - Swap Halo

For the 2 dimensional case the iteration is done first for the odd rows and even rows seperately, due to the similarity between all the red points in the odd rows, and between the red points in the even rows. Then the cycling could be generalized into a static inline function used for all the cycling.


(a) This figure shows the Red and Black ordering, utlized by the GS-RB algorithm. Each color is done seperately.

For the 3 dimensional case there are two different approaches to the problem, one where the iteration through the grid is streamlined, but needs several loops through the grid to take care of a subset of the grid points each loop. The other algorithm uses one loop through the grid, with different conditions on the edges to make it go through the correct grid points in each line.

When the loops are streamlined the edges the loop go through the entire grid, but when it reaches and edge in it either needs to add or subtract 1 to the iterator index. In we want to do a red pass, computing all the red values, the grid is cycled through increasing by 2 each time. Then it will access up the indexes, $36, 38, 40, 42, \cdots$. In the second row we want it to use the index 43, instead of 42, so we need to increase it by 1 when it reaches the edge. When it reaches the end of the second line, we want it to increase from 47 to 48, so then we need to subtract 1. In the next layer we need to shift the behaviour on the edges to the opposite.

$$\begin{array}{cccccc}
. & . & . & . & . & . \\
48 & 49 & 49 & 50 & 51 & 52 \\
42 & 43 & 44 & 45 & 46 & 47 \\
36 & 37 & 38 & 39 & 40 & 41
\end{array}$$

In the other implementation I tried something similar to the 2D implementation, where it does several loops through the grid, computing an easier subgroup of the red nodes each time, so the iteratior index can increase by just to each time. So for the red points it computes the odd and even layers and rows seperately.

- Compute Odd layers, odd rows
- Compute Odd layers, even rows

- Compute Even Layers, odd rows
- Compute Even layers, even rows

4.5 Implementation of Boundary Conditions

Since the subdomains already need to exchange the halo, we already had made a suite of function dealing with halo-operations. These functions can get, set, add, etc. N-dimensional slices from the halo. As can be seen in **??** this is quite similar to the needs of the varying boundary conditions and we will reuse this capability during the implementation of boundary conditions. In addition we will also need methods to restrict the part of the conditions that need to be restricted. Here we will only deal with time-invariant boundary conditions so they can be set, and restricted, once during initializations of the grids. It should be easy to expand it to time varying conditions, just restrict the boundary function each timestep.

In general the condition types are stored as a $2-$dimensional array, in the order $x_{low}, x_{upper}, y_{low}, \ldots$. Each time the boundary conditions function is called it checks if each subdomain edge is at the total domain boundary. Since the outer subdomains need to do extra computations here, it shouldn't matter if the inner subdomains do some extra calculations. If the subdomain boundary is at the boundary it calls the a function depending on which boundary type the edge is.

4.5.1 Restriction

For now we have choosen to use straight injection, since we will not use any complicated boundary conditions in this project, other developers are welcome to expand it by more restriction algorithms.

4.5.2 Periodic

The periodic boundary conditions are just the same procedure as the halo exchange. To try too keep an even load, between the computational nodes, the halo exchange is also done between the the boundary subdomains.

To keep the convergence rate good we also need to keep the *global constraint* and *compatibality condition* in mind, see section 3.4.1. For this we have a N-dimensional parallel algorithm that neutralizes a grid. This adds up the values from all the subdomain and makes sure the total of the values is 0.

4.5.3 Dirichlet

Given that we have a slice, representing the dirichlet conditions on the relevant edges, the conditions are easily set by the use of slice-operations. The outermost slice, i.e. ghost layer, is set to be equal to the boundary slice.

4.5.4 Neumann

Neumann conditions are dependent the outer edge of the true domain, due to this they need to be reset at the finest level for each timestep and restricted down. Elsewise they are handled as the Dirichlet conditions.

Chapter 5

Verification and Performance

5.1 Verification

In this chapter we will go through different methods we used to test and verify the multigrid solver, as well as scaling measurements. Modular parts of the solver is tested with unittests where feasible. In addition the whole solver is tested with both analytically solvable test cases and randomly generated fields.

We also investigate the performance and scaling of the solver in the parallel environment. This is to obtain a better understanding of how the field resolution can be scaled up without hampering the perfomance of the particle-in-cell simulation to much. We are interested in both how well the solver performs on a larger number of processors, as well as the perfomance impact of the different parameters in the solver.

5.1.1 Error Quantification

In order to evaluate solutions we will primarily look at the normalized 2-norm of the error, eq. (5.1), and the residual, eq. (5.2). The $\|e\|_2$ is computed from comparing the numerical solution $\widehat{\phi}$ to the analytical solution ϕ and normalized with regards to the number of grid points, N. The residual is found by inserting the numerical solution into the Poisson equation, the remaining part is then the residual, and shows the difference of the current numerical solution and the optimal numerical solution.

$$\|e\|_2 = \sqrt{\frac{\sum\left(\hat{\phi} - \phi\right)^2}{N}} \tag{5.1}$$

$$\bar{r} = \frac{1}{N}\left(\sum_i \nabla^2 \widehat{\phi}_i + \rho_i\right) \tag{5.2}$$

5.2 Multigrid Solver

To test the solver itself we employ a couple of different techniques. For the main test we use a charge distribution with known analytical solution, and we then compare the numerical solution to the analytical solution.

Since constructed solutions can often behave too nice, we will also perform another test with a randomized charge distribution, here we will only look at the residual since we cannot compute the error due to not having an analytical solution. We expect the residual to approach 0.

Lastly we perform a few run on identical charge distributions, domain subdivisions and compare the solutions.

5.2.1 Analytical Solutions

We use a few different constructed charge density fields, which are analytically solvable, to test the performance and correctness of the solver. All the simulations here are ran on a grid of the size $128, 64, 64$ divided into $1, 2, 2$ subdomains, with PINC version 36ad. It uses 5 cycles when presmoothing, solving on the coarsest grid and postsmoothing, the MG solver is instructed to run for 100 MG V-cycles with 2 grid levels.

Sinusoidal function

A sinusoidal source term, ρ can be useful to test the solver since it can be constructed to have very simple derivatives and integrals. Here we use a sinusoidal function that has two positive maxima and two negative maxima over the total domain. We want the sinus function to go over 1 period over the domain, so we normalize the argument by dividing the grid point value, x_j, y_k, z_l, by the domain length in the direction, L_x, L_y, L_z.

$$\rho(x_j, y_j, z_l) = \sin\left(x_j \frac{2\pi}{L_x}\right) \sin\left(y_k \frac{2\pi}{L_y}\right) \tag{5.3}$$

A potential that fits with this is:

$$\phi(x, y, z) = -\left(\frac{2\pi}{L_x}\right)^2 \left(\frac{2\pi}{L_y}\right)^2 \sin\left(x_j \frac{2\pi}{L_x}\right) \sin\left(y_k \frac{2\pi}{L_y}\right) \tag{5.4}$$

Fig. 5.1 shows the results from running the MG-solver on the test sinusoidal test case described here. As can be expected the potential mirrors the charge distribution, except with an opposite sign and a larger amplitude. A decently large grid was simulated and the mean residual was found to be: $\bar{r} \approx 0.031$.

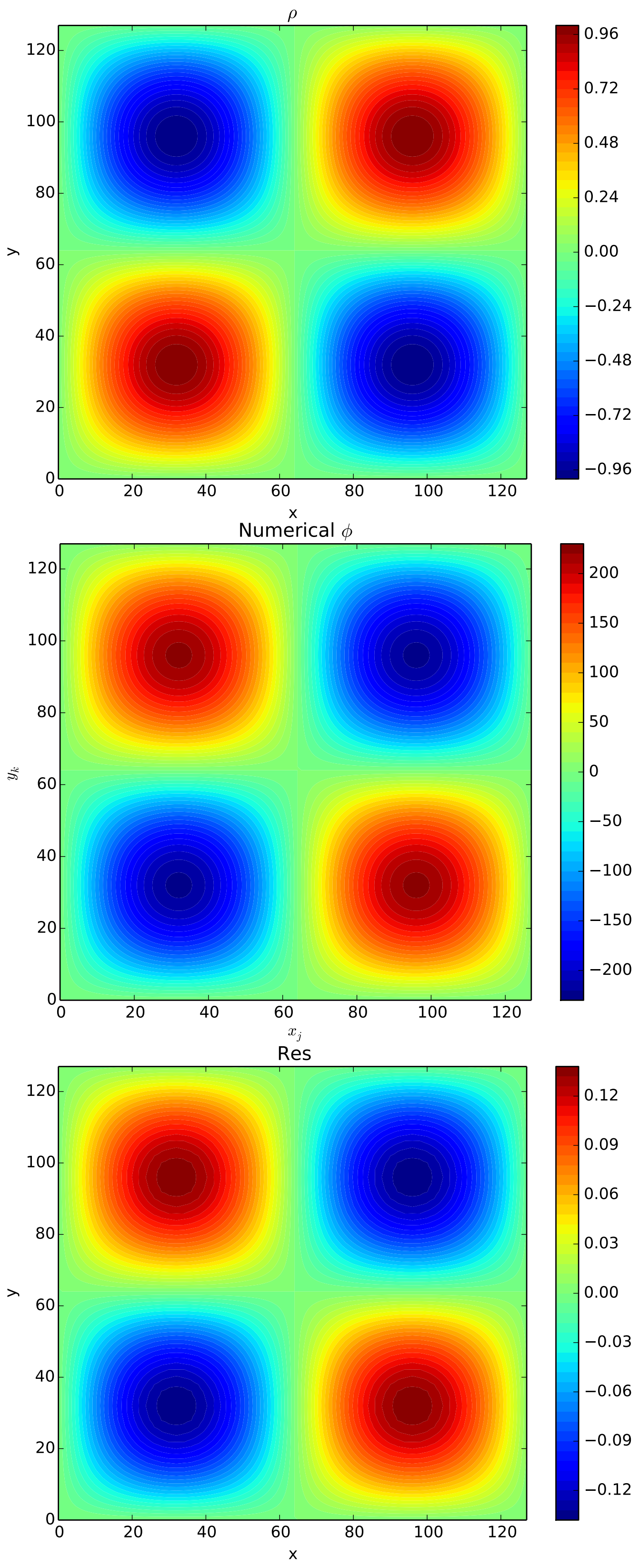


Figure 5.1: The x, y-plane is depicted from the grids cut along $z_l = 32$, from the sinusoidal test case described in section 5.2.1. The top plot shows the charge distribution, the center plot shows the numerical solution of the potential and the bottom plot depicts the residual. All the quantities are in normalized dimensionless units. x, y and z are given in grid points.

Heaviside Function

The solver is also tested with a charge distribution described by a Heaviside function. This is also suited to testing since the charge distribution is then constant planes, and we expect second order polynomial when integrating them. In the test case there are two planes with the value -1 and two planes with 1. In fig. 5.2 the test case, as well as the solution and residual is shown, and we can see the polynomials in the solution. The mean residual $\bar{r}$ was 0.0068.

$$\rho_(x_j, y_k, z_l) = \begin{cases} 1 & y_j\epsilon(0,32),(64,96) \\ -1 & y_j\epsilon(33,65),(97,127) \end{cases} \tag{5.5}$$

5.2.2 Random Charge distribution

To ensure that the solver performs well for less regular charge distributions, we carry out a test with randomized charge distribution. We use a standard random number generator from the gnu scientific library. Fig. 5.3 shows the charge distribution, numerical potential and the residual for this case. The mean residual was found to be $\bar{r} \approx 0.0039$, which is even smaller than for other more regular cases. This confirms that the solver performs well and accurately.

5.2.3 Additional Tests

In addition to the tests on shown in this section, we also ran the same tests obtaining similar results on various sizes and directions. Since we wanted to be sure that the program was working independently of the how the domain was divided into subdomains, we also performed tests on different subdomain divisions. There were no difference between different subdivisions and directions, which confirms that the merging between subdomains works well. These results are not shown here.

5.2.4 ND vs 3D algorithms

PINC is built to have two sets of algorithms, one N-dimensional and one 3-dimensional. The N-dimensional algorithms is meant to be used in cases where one want to do 1- and 2-dimensional simulations as well as a test for the 3-dimensional algorithm. The 3-dimensional algorithm is generally slightly faster than the N-dimensional due to some extra capabilities in hardcoding certain parts. On a laptop, using two 'Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz' processors, we ran the multigrid solver on a $128, 128, 128$ size problem, with both the 3D and ND algorithms. The ND algorithms used 80.08s to solve it to the given tolerance, while the 3D algorithms used 78.75s achieving only a slight

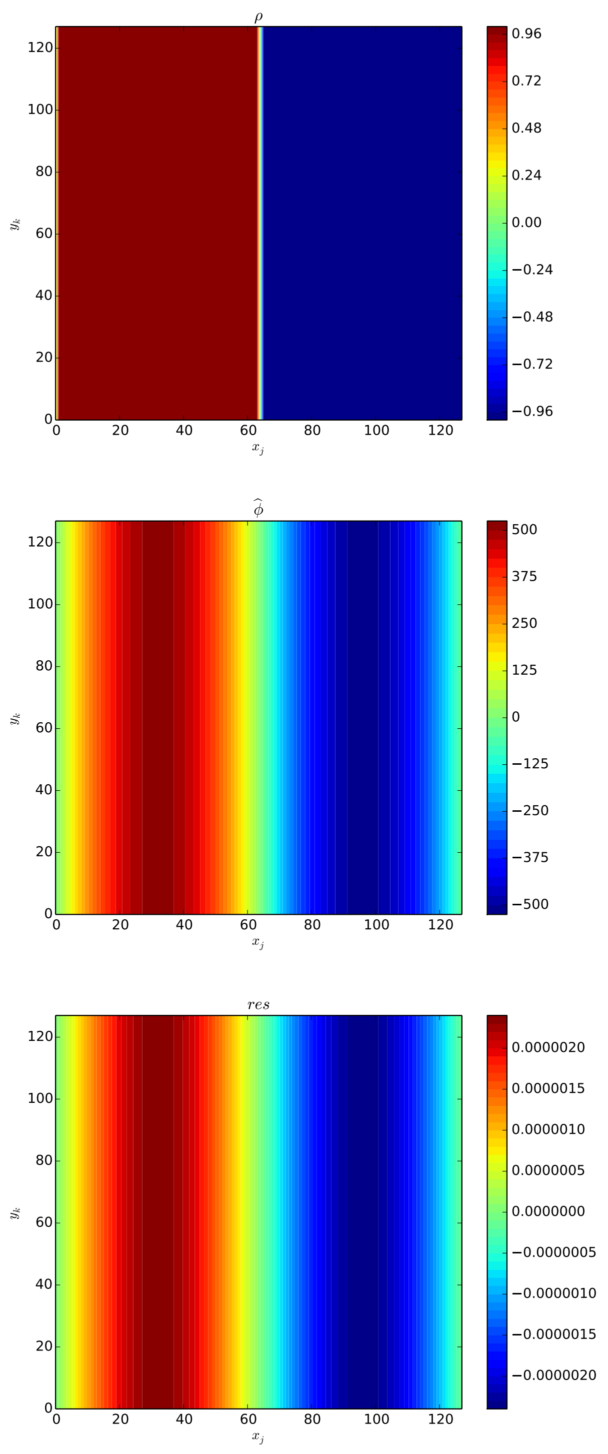


Figure 5.2: As earlier this is a x, y-plane cut along $x_k = 32$, of the grid. The plots show the charge distribution, numerical solution and the solution, from left to right. This is a test case constructed with Heaviside functions. In the solution of the potential the expected second degree polynomial can be seen.

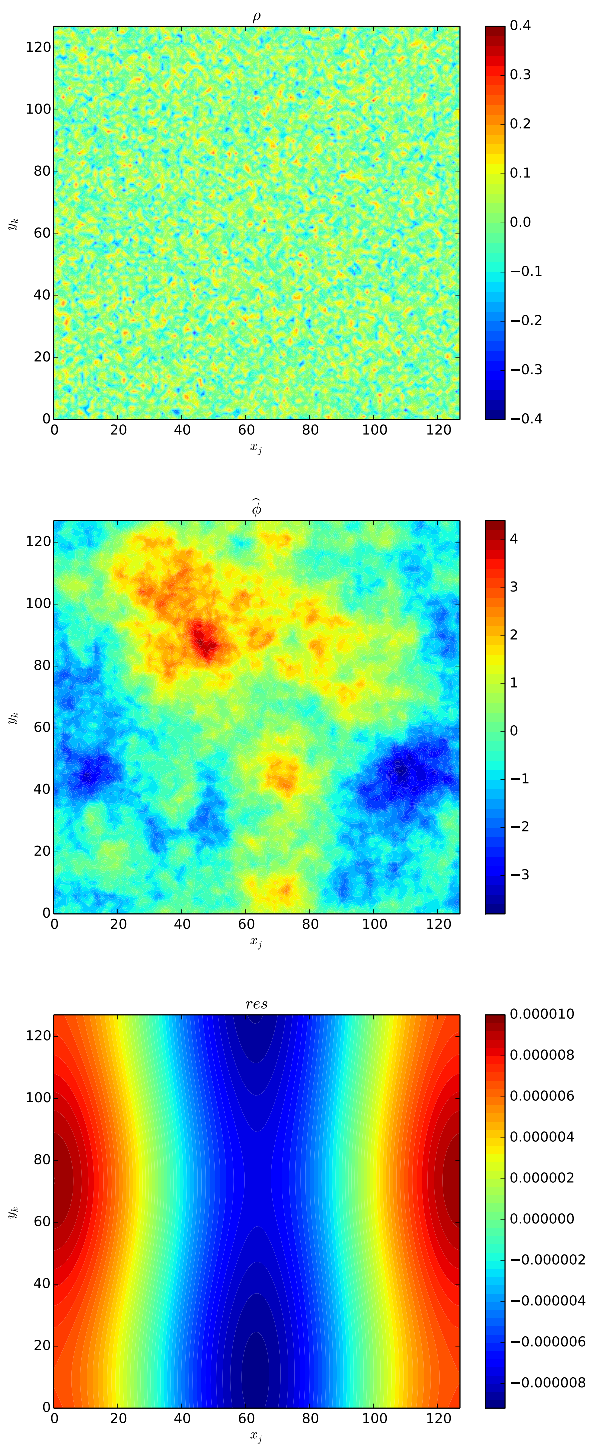


Figure 5.3: As earlier this is a x, y-plane cut along $x_k = 32$, of the grid. The plots show the charge distribution, numerical solution and the solution, from top to bottom. It is visible that the potential shows more structure than the ρ, since the integration smooths the original problem.

speedup. This is likely due to most of the computational time being spent on the interprocessor communication.

5.3 Scaling of the error compared to discretization

The charge density is represented on a discretized grid and due to this a numerical solution will have an inherent error. The error will be of second order, $\mathcal{O}(h^{-2})$, dependent on the stepssize, Δx, due to the first order solver, see section 3.3.3.

To investigate that the error of the solver follows a second order improvement as the stepsize decreases we construct a sinusoidal ρ as a test case.

$$\rho(x) = \sin\left(\frac{x}{2\pi}\right); \qquad x\epsilon[0, 2\pi] \tag{5.6}$$

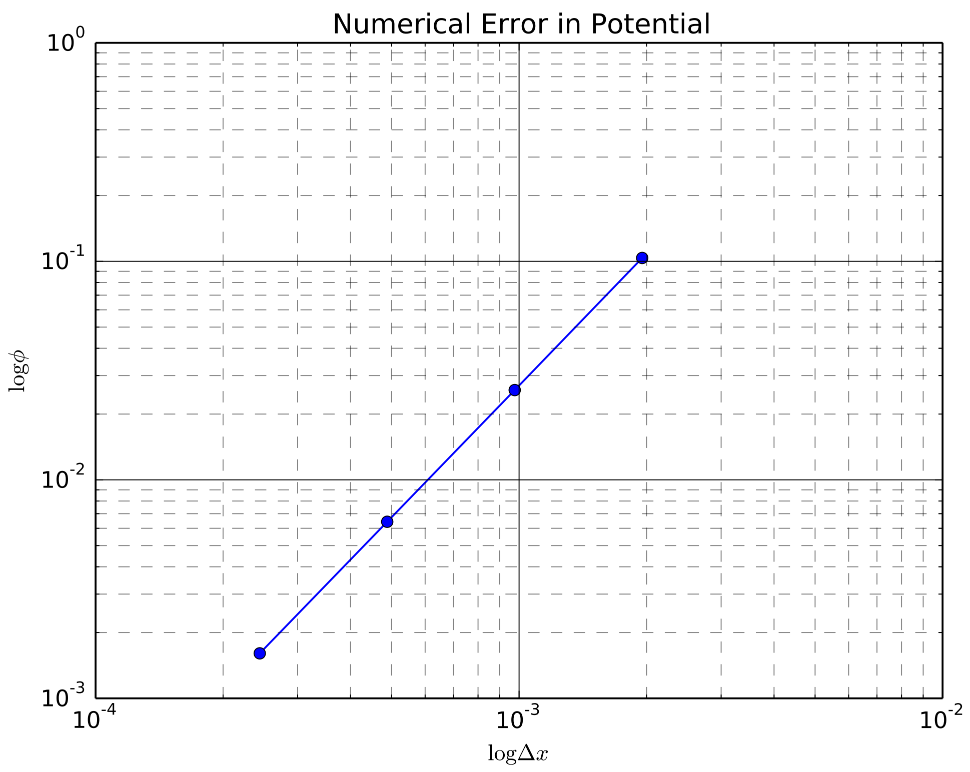
This ρ is analytically solvable for the Poisson equation so we can compute the 2-norm of the error, $\|err\|_2$. Then we gradually decrease the stepsize and compare the norm of numerical solutions. Since the normalization in PINC is normally done outside the multigrid solver, rho had to be suitably scaled to the stepsize. We expect the error to be proportional to the squared stepsize, $err(h) \approx Ch^2$, where C is a constant dependent on the geometry of the problem. By taking the logarithm we obtain

$$\log(err(h)) = 2\log(Ch) \tag{5.7}$$

Fig. 5.4 shows the the measured error when solving the sinusoidal charge distribution, eq. (5.6), for both the potential, ϕ, and the electric field, E_x. The problem was solved with different discretizations on a 3-dimensional domain, starting at $[8, 8, 8]$ doubling the grid points each time. The slope on the logarithmic plots was in both cases found to be 2.00, showing a second order error scaling $\mathcal{O}(h^{-2})$. The same test was also performed in 1 and 2 dimensional cases, with varying subdomain configurations and with the sinus shape along the other axes.

5.4 Plasma Oscillation

As a test of the validity of our PiC program, we can use the Langmuir oscillation, described in section 2.5. This test is inspired by a case set up in Birdsall and Langdon (2004), and modified to fit with our normalization and discretizations. To verify that the oscillation is properly simulated we will look at oscillations in the energy.



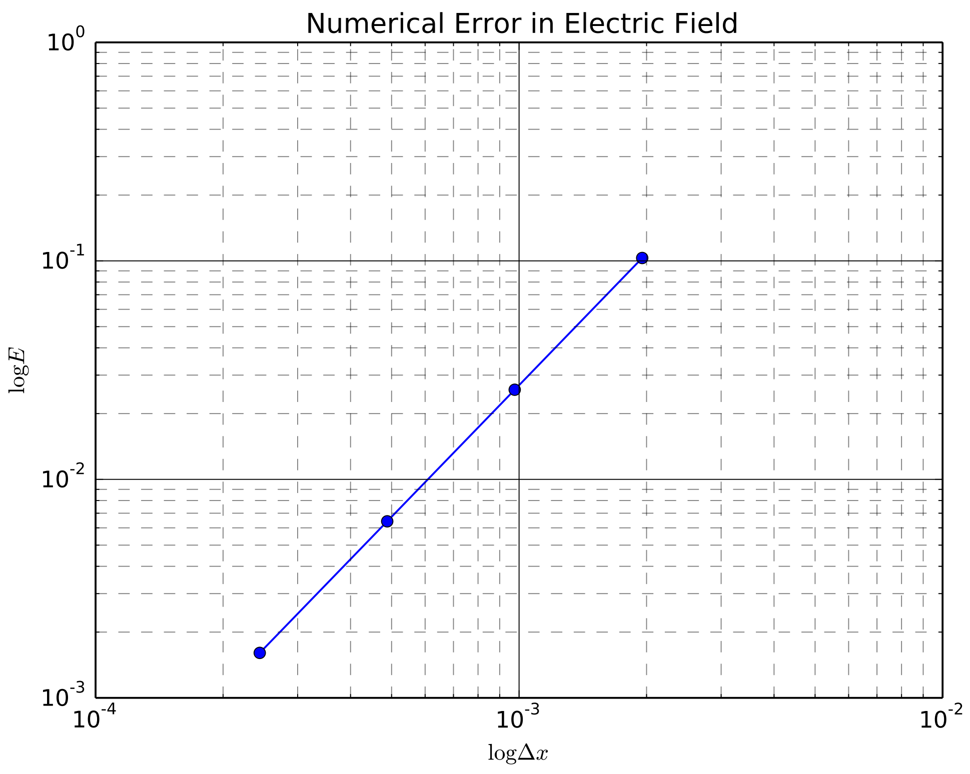


Figure 5.4: Logarithmic plot of the 2-norm of the error of the potential ϕ, top figure, and the x-component of the electric field. The solver was run on a scaled sinus-shaped charge distribution. Both of the plots show a straight line of the error, on the logarithmic plots, with a slope of 2.00. This corresponds to the error scaling with order -2 as a function of the stepsize as expected. All the units are in PINC normalized units.

5.4.1 Input parameters

Timestep

First we need to ensure that the simulation does not violate the time- stability criterion, section 3.1.3, $\Delta t \leq 2\omega_{pe}$. For computational reasons each particle in the program represents many real particles and we can adjust the number, of real particles represented, to ensure that the electron plasma frequency is 1.

$$\omega_{pe}^2 = \frac{nq_e^2}{m\epsilon_0} = \left(\frac{N}{V}\right)\left(\frac{q_{e*}}{m_{e*}}\right) q_{e*} \tag{5.8}$$

Here (N/V) is just the number of electron super-particles divided by the volume, q_{e*} and m_{e*}, represent super-particles and need to be multiplied by the number of particles in a super-particle, κ.

$$\omega_{pe}^2 = \left(\frac{N}{V}\right)\left(\frac{e}{m_e}\right) \kappa e \tag{5.9}$$

Since we want the electron plasma frequency to be 1, κ is set to

$$\kappa = \frac{Vm_e}{Ne^2} \tag{5.10}$$

Now the time is given in units of ω_{pe} and we use $\Delta t = 0.2 < 2$ easily.

Spatial-step

The spatial step need to satisfy the finite grid instability condition, $\Delta x < \varsigma\lambda_{De}$, section 3.1.3. We use a similar procedure as in normalizing the time with regards to ω_{pe} to normalize the length with regards to λ_{De}. The we end up with the temperature, or its other representation as thermal velocity, as the parameter we can adjust to make sure that it satisfies the condition.

$$\lambda_{De}^2 = \frac{\epsilon_0 kT_e}{nq_e^2} \tag{5.11}$$

Simulation

We ran a simulation for $10\omega_{pe}^{-1}$, on a $64, 64, 64$ size grid. The particles were first randomly distributed and given a Maxwellian velocity distribution. Then they were slightly perturbed to create an imbalance.

The simulation was set to run for 100 timesteps, where each timestep represents $0.1\omega_{pe}^{-1}$.

$$\tau_p \equiv 2\pi/\omega_{pe} \tag{5.12}$$

Using the plasma period this gives $10/2\pi \approx 1.6$ periods.

Size	Stepsize	Timestep	#Particles per cell	v_{th} (electrons, ions)
$(64, 64, 64)$	0.2	0.1	64	$0.002, 0.00046$

Table 5.1: Key settings for the Langmuir wave oscillation. The complete settings are stored as an input file, langmuirWarm, in the PINC directory.

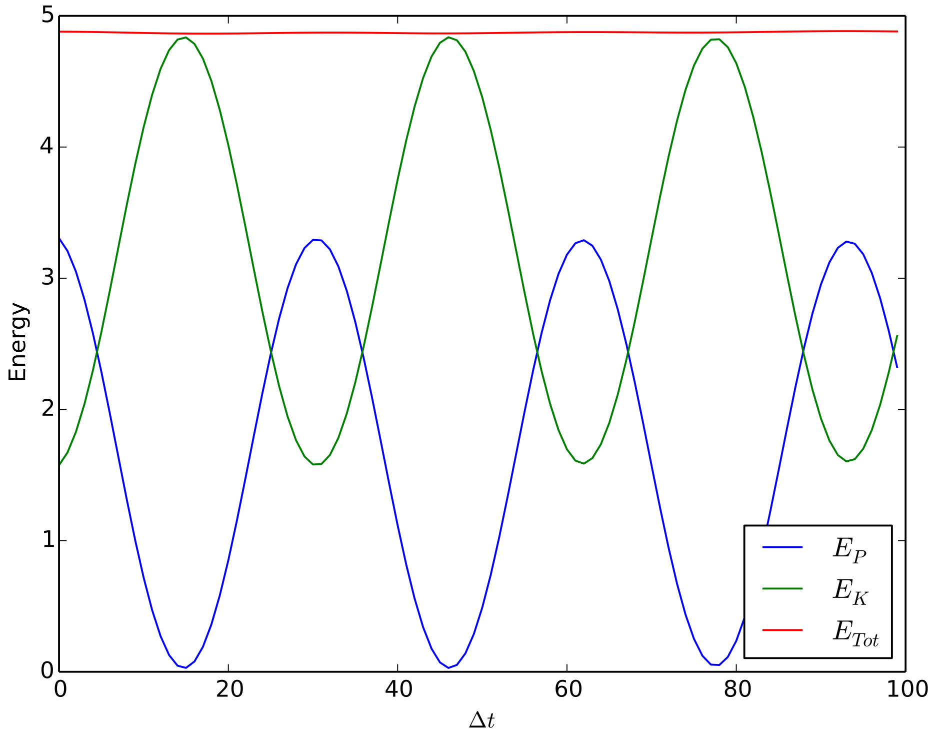


Figure 5.5: This shows the time-evolution of the energy in an perturbed plasma. The energies are in normalized units and $\Delta x = 0.1\lambda_{De}$. The total energy has a maximum variation of 0.22%. In the timespan of $10\omega_{pe}$ the plasma oscillates over 1.6 times.

The section 5.4.1 shows the energy fluctuations for a plasma oscillation, the total energy is stable conserved, with a maximal variation of 0.22%. The potential energy starts large and then it sinks as the potential energy is converted to kinetic energy. This is due to the particles attempting to neutralize the fields. The particles overshoots equilibrium position and the kinetic energy starts to convert back to potential energy. Thus, the PINC with the multigrid solver developed in this thesis, can simulate collective plasma phenomena and accurately reproduce Langmuir Oscillations, which we discussed in section 2.5 and which are the fundamental oscillations in plasma (Pécseli, 2012; Chen, 1984).

5.5 Performance

5.5.1 Perfomance Optimizer

A multigrid solver has several parameters that needs to be set correctly for a an optimal performance. These parameters are dependent on the problem size, as well as the computing architecture. Instead of attempting to estimate them beforehand we have included an external script that runs the program with different MG-solver settings on the wanted domain size and tries to optimize them, see appendix B.2. The parameters it tries to change is the number of grid levels and the cycles to run for presmoothing, postsmoothing and the coarse solver. It should be worth it to spend some computing power, finding close to optimal settings, prior to running a full scale simulation since the solver needs to run each time step. The performance optimizer naively runs the solver for a predetermined mesh of settings.

5.5.2 Convergence Rate

As an iterative solver a multigrid solver will gradually approach a solution, reducing the residual further each run. In this subsection we will measure the convergence rate, defined in similar way as in Zhukov et al. (2014),

$$p = \left(\frac{r_m}{r_0}\right)^{1/m} \tag{5.13}$$

where r_m, r_0 are the 2-norm of a the residual after m multigrid runs and the inital residual. We presume that each run of the multigrid solver will remove a proportion of the remaining residual. The tests are done on a sinusoidal problem for varying grid sizes. The smoothers run for 100 cycles on each of the 5 multigrid levels. Zhukov et al. (2014) found convergence rates, p, between 0.095 and 0.155 using a multigrid solver with a Chebyshev algorithms to smooth. While the convergence rates found here were worse, this is to be expected with the much simpler Gauss-Seidel smoothers.

Grid	p
64^3	0.149
128^3	0.192
256^3	0.203

Table 5.2: The convergence rate for the multigrid solver, running on 5 levels. The convergence rate becomes worse for larger grids.

5.5.3 Scaling of the MG Solver

One of the aims of building a parallell multigrid solver was to be able to enable simulating large plasma problems. To be able to achieve that the solver should be able to scale up very well, i.e. doubling the problem size and the number of available processors should only give a manageable increase in computational time. We don't expect to be able to achieve a perfect parallelization, since there is a certain amount of interprocessor communications necessary that will slow down the algorithm compared to a sequential algorithm. The exact parallel performance is also dependent on the communications channels and the topology between the processor clusters. In section 3.5.6 the parallel complexeties for the different multigrd algorithms is given and we will look at the parallel proprties for a V, W and FMG algorithm.

To investigate the scaling properties we will run set up a standard problem, and solve it with increasing resolutions. We keep the size of the domain unchanged but increase the resolution (reducing the spacing). We start with a 32^3 grid on 1^3 computaional core, then we increase the problemsize to 64^3 on 2^3 and so on. These tests were run on Abel, UiO's computer cluster, and the technical details can be found at the web page [1]. The results are shown on c**??**. From section 3.5.6 we expect theoretical optimal scaling of $\log N \log \varepsilon$. While this was not achieved, the settings were not optimized in these runs, for the various problem sizes and this may have caused it to scale worse than it should. When more processors are used the speed of the interprocessor communcation may have slowed down, as the processors are not part of the same node. The main problem with the test was that the subdomains was very small so the communication costs dominated. The communication costs scales linearly (Jung, 1997), so for the coarse grids of the 5-level multigrid the solver scaled close to linearly.

Baker et al. (2012) found it crucial to use assumed partition to achieve good scalability, as the interprocessor communication costs are the main obstacle to scalability for multigrid methods. By assumed partitioning the subdomains are divided into nearby groups to easy communication between them.

[1]http://www.uio.no/english/services/it/research/hpc/abel/more/index.html (Accessed 2016-11-15)

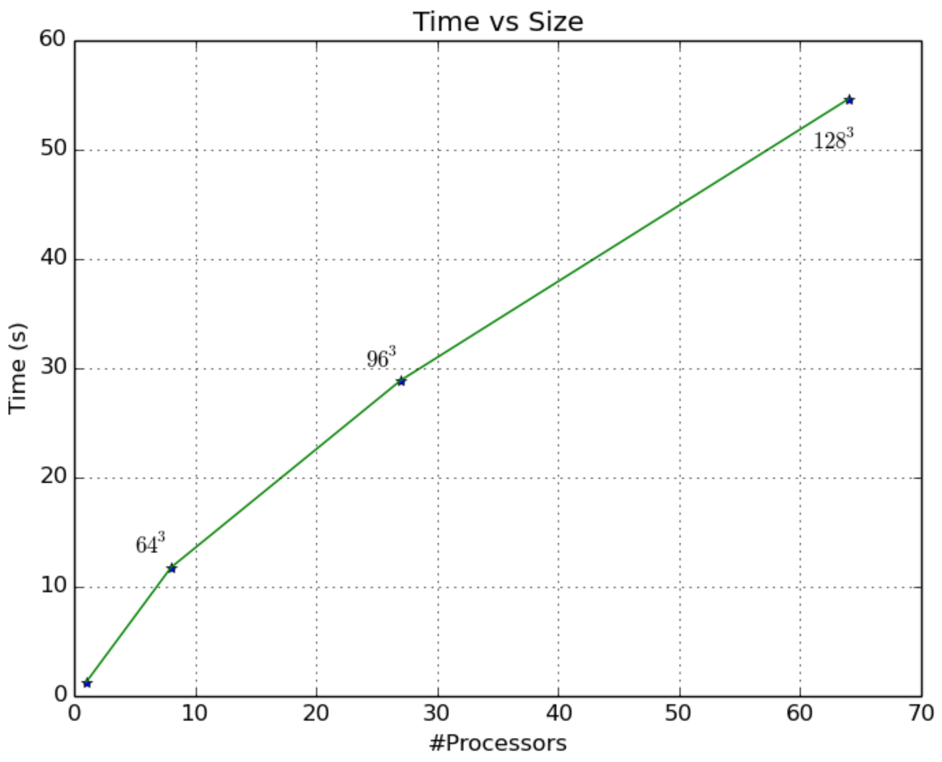


Figure 5.6: A Langmuir Oscillation were performed for 10 timesteps with a $(32, 32, 32)$ grid on each processor. This was repeated with increasing amount of processors, $(1, 8, 32, 64)$, to see how the multigrid solver scales.

Chapter 6

Summary and Conclusion

6.1 Summary

The main object of this project was to develop a new parallel multigrid solver for use in PINC.

The verification section goes through different ways the solver was tested. The main tests were done on charge distributions with known analytical solutions. The numerical solutions were confirmed to agree with the analytical solutions. These type of tests were done for various dimensions and subdivisions of the domain. The N-dimensional algorithms were then tested and compared with the specialized 3-dimensional algorithms. For the chosen parameters the N-dimensional algorithms were almost as fast as the 3D version, this may not be true for all configurations. Unfortunately the capability of the solver to use different boundary conditions, and mix them, was not well tested due to limited time and debugging of PINC to make it functional. The $\mathcal{O}(h^2)$ scaling of the 1st order discretization of the solver was then verified by measuring how the error decreased as the resolution increased. Next we wanted to look at the whole PINC program. It then confirmed to work correctly with a simulation of a Langmuir oscillation.

Since PINC is made to perform on a supercomputer we wanted to see how well it performs and scales on a multiple processors. First we measured the convergence rate of the algorithm, which was found to be between 0.149 and 0.203. For this we used UiO's supercomputer Abel. Due to bad choices in multigrid parameters, using 5 multigrid levels on small grids, the communication dominated the computational time. Due to time constraints further scaling measurements with better parameters were not conducted.

6.2 Concluding Remarks and Further Proposals

While the multigrid solver was shown to work and produce the correct results the scalability was satisfactorily shown. While the program is usable and under further development, it needs to be shown to scale well. A multigrid algorithm based on Gauss-Seidel should scale better than the performed tests show, as Jung (1997) showed. It should also be considered to implement the *processor-block Gauss-Seidel* from M. Adams et al. (2003). Further additions to increase the possible use of PINC include the possiblity to have objects in the plasma, which can be used to model the effects surrounding spacecraft and dust particles (Miloch, 2010; Miyake et al., 2013; Ergun et al., 2010). Collisions between particles will enable further studies of instabilities of plasma streams (Brackbill et al., 1995). Another way PINC can be further developed is with full implicit algorithms. Some of these suggestions are under planning and development.

Appendix A

Unittests

A.1 Unittests

Unittests are small tests that is used to check that the single pieces of the code work as they should. This serves a dual purpose in developing a software project. When a part of the code is developed it serves as a framework to create a standardized test of the piece of code that can easily be repeated. The unit tests are not maintained in the latest version of the software. It also helps when developing the higher level algorithms, in that the unittests ensures that the problem lies in the higher level algorithm and not in the lower level pieces it uses. When implementing wider changes, for example datastructures, the unittests can help making sure that the changes are not causing any unintended bugs. For information of how to use the unittests see the documentation, **documentation**

A.1.1 Prolongation and Restriction

The prolongation and restriction operators with the earlier proposed stencils will average out the grid points when applied. So the idea here is to set up a system with a constant charge density, $\rho(\mathbf{r}) = C$, and then apply a restriction. After performing the restriction we can check that the grid points values are preserved. Then we can do the same with the prolongation. While this does not completely verify that the operators work as wanted, it gives an indication that we have not lost any grid points and the total mass of the charge density should be conserved.

A.1.2 Finite difference

The finite difference operators is tested by setting up a test field based on a polynomial on which the operator should give an exact answer for. For example if we have a quantity $f(x) = 3x$, then a first order finite difference scheme will give $\hat{\nabla} f(x) = 3$.

A.1.3 Multigrid and Grid structure

We want the basic grid to be available through a grid datastructure and the stack of grids stored in the multigrid structure. To ensure that this will still work through changes in the the structs there is a simple unittest that uses a grid struct to set up a field, then it is changed in the multigrid struct. Then it confirms that the values in the grid struct is also changed.

A.1.4 Edge Operations

In the communication between the subdomains, as well as in the treatment of boundary conditions, there is a group of functions dealing with slice operations. These are tested by putting assigning each subdomain different constant values, then different slice operations is performed.

Appendix B

Scripts

B.1 PINC framework

```
import subprocess
import numpy as np

class Pinc(dict):
  def __init__(self, pinc="./mpinc.sh",
      ini="langmuirWarm.ini", path="../.."):
    # All commands will be executed from "path"

    self.pinc = pinc
    self.ini = ini
    self.path = path

  def run(self):
    cmd = self.pinc + " " + self.ini
    for key in self:
      cmd += " " + key + "=" + self.parse(key)
    self.runCommand(cmd)

  def runCommand(self, cmd):
    cmd = "cd " + self.path + "; " + cmd
    subprocess.call(cmd,shell=True)

  def clean(self):
    self.runCommand("rm -f data/*.h5")
    self.runCommand("rm -f data/*.txt")

  def parse(self, key):
    value = self[key]
    if isinstance(value,(list,np.ndarray)):
      string = str(value[0])
      if(len(value) > 1):
        for l in range(1,len(value)):
          string += "," + str(value[l])
      return string
    elif isinstance(value,(int,float)):
      return str(value)
    else:
      return value
```

Listing B.1: Framework to more easily run PINC with various settings

B.2 Multigrid Parameter Optimizer

```
  from pincClass import *
import subprocess
import h5py
import numpy as np
import sys as sys

if(len(sys.argv) > 1):
  path = "../../" + sys.argv[1]
else:
  path = "../../local.ini"
pinc = PINC(iniPath = path)

#Setting up wanted needed ini file
pinc.mode     = "mgRun"
pinc.mgCycles     = 1
pinc.startTime    = -1

class Settings:
  def __init__(self, nPre = 10, nPost = 10,
          nCoarse = 10, mgLevels = 3):
    self.nPre     = nPre
    self.nPost    = nPost
    self.nCoarse  = nCoarse
    self.mgLevels = mgLevels
    #Store results
    self.mgCycles   = 0
    self.time     = float('Inf')

  def copy(self, copy):
    self.nPre     = copy.nPre
    self.nPost    = copy.nPost
    self.nCoarse  = copy.nCoarse
    self.mgLevels   = copy.mgLevels

  def setPinc(self, pinc):
    pinc.preCycles      = self.nPre
```

```
      pinc.postCycles      = self.nPost
      pinc.coarseCycles    = self.nCoarse
      pinc.mgLevels        = self.mgLevels
      pinc.startTime       += 1


def formatTimeCycles(fileName, nRuns):
  data = h5py.File(fileName,'r')
  time = np.array(data['time'][nRuns,1])
  mgCycles= np.array(data['cycles'][nRuns,1])
  data.close()

  return time, mgCycles


def nCycleOptimize(count, nTries, nRun, bestRun, currentRun,
    pinc):
  pTime = float('Inf')
  preInc = 1
  for j in range(0,nTries): #nCoarse
    # print "Hello"
    if(count>0):
      nRun = nCycleOptimize(count -1, nTries, nRun, bestRun,
          currentRun, pinc)
    ##Run, retrieve time and cycles used
    currentRun.setPinc(pinc)
    pinc.runMG()
    time, mgCycles = formatTimeCycles('test_timer.xy.h5',nRun)

    #Check if best run
    if(time < bestRun.time):
      bestRun.copy(currentRun)
      bestRun.time = time
      bestRun.mgCycles = mgCycles

    if(preInc == 1):
      if(time < pTime):
        pTime = time
        if(count == 2):
          currentRun.nCoarse *= 2
        if(count == 1):
          currentRun.nPre *= 2
        if(count == 0):
          currentRun.nPost *=2
      else:
        if(count == 2):
          currentRun.nCoarse *= 0.5
        if(count == 1):
          currentRun.nPre *= 0.5
        if(count == 0):
          currentRun.nPost *=0.5
          preInc = -1
      else:
        if(time < pTime):
          pTime = time
          if(count == 2):
            currentRun.nCoarse *= 0.5
          if(count == 1):
            currentRun.nPre *= 0.5
          if(count == 0):
            currentRun.nPost *=0.5
        else:
          break
    nRun += 1
  return nRun


bestRun = Settings()
currentRun = Settings(10,10,10,5)

pinc.clean()
nTries  = 100
nRun  = 0
preInc  = 1



for i in range(1):  #mgLevels

  # nRun = nCoarseOpt(nTries, nRun, bestRun, currentRun, pinc)
  nRun = nCycleOptimize(2 ,nTries, nRun, bestRun, currentRun,
      pinc)

  currentRun.mgLevels += 1


print "\nBest runtime \t= " , bestRun.time*1.e-9, "s"
print "\nProposed run:"
print "mgCycles \t= "    , bestRun.mgCycles
print "mgLevels \t= "    , bestRun.mgLevels
print "nPreSmooth \t= "    , bestRun.nPre
print "nPostSmooth \t= "   , bestRun.nPost
print "nCoarseSolve \t= "  , bestRun.nCoarse
```

Listing B.2: Optimizer script

B.2.1 V-cycle, code

```
void inline static mgVRecursive(int level, int bottom, int
    top, Multigrid *mgRho, Multigrid *mgPhi,
                    Multigrid *mgRes, const MpiInfo *mpiInfo){

//Solve and return at coarsest level
if(level == bottom){
  gInteractHalo(setSlice, mgPhi->grids[level], mpiInfo);
  mgRho->coarseSolv(mgPhi->grids[level], mgRho->grids[level],
      mgRho->nCoarseSolve, mpiInfo);
  mgRho->prolongator(mgRes->grids[level-1],
      mgPhi->grids[level], mpiInfo);
  return;
}

//Gathering info
int nPreSmooth = mgRho->nPreSmooth;
int nPostSmooth= mgRho->nPostSmooth;

Grid *phi = mgPhi->grids[level];
Grid *rho = mgRho->grids[level];
Grid *res = mgRes->grids[level];

//Boundary
gInteractHalo(setSlice, rho, mpiInfo);
gBnd(rho,mpiInfo);

//Prepare to go down
mgRho->preSmooth(phi, rho, nPreSmooth, mpiInfo);
mgResidual(res, rho, phi, mpiInfo);
gInteractHalo(setSlice, res, mpiInfo);
gBnd(res, mpiInfo);

//Go down
mgRho->restrictor(res, mgRho->grids[level + 1]);
mgVRecursive(level + 1, bottom, top, mgRho, mgPhi, mgRes,
    mpiInfo);

//Prepare to go up
gAddTo( phi, res );
gInteractHalo(setSlice, phi,mpiInfo);
gBnd(phi,mpiInfo);
mgRho->postSmooth(phi, rho, nPostSmooth, mpiInfo);

//Go up
if(level > top){
  mgRho->prolongator(mgRes->grids[level-1], phi, mpiInfo);
}
return;
}
```

Listing B.3: Implementation of an recursive V-cycle

Appendix C

Examples

C.1 Ex: 3 level V cycle, steps necessary

① Compute defect on grid 0, the finest grid:

- $\widehat{\phi}_0 = \mathcal{S}(\phi_0, \rho_0)$
- $d_0 = \nabla^2 \widehat{\phi}_0 - \rho_0$
- Restrict defect: $\rho_1 = \mathcal{R} d_0$

② Compute defect on grid 1:

- $\widehat{\phi}_1 = \mathcal{S}(\phi_1^0, \rho_1)$
- $d_1 = \nabla^2 \widehat{\phi}_1 - \rho_1$
- Restrict defect: $\rho_2 = \mathcal{R} d_1$

③ Solve Coarse Grid for correction ω

- $\phi_2 = \mathcal{S}(\phi_2^0, \rho_2)$
- Interpolate as correction:$\omega_1 = \mathcal{I} \phi_2$

④ Add correction on level 1:

- $\widetilde{\phi}_1 = \widehat{\phi}_1 + \omega_1$
- $\phi_1 = \mathcal{S}(\widetilde{\phi}_1, \rho_1)$
- Interpolate correction:$\omega_0 = \mathcal{I} \phi_1$

⑤ Compute solution.

- $\widetilde{\phi}_0 = \widehat{\phi}_0 + \omega_0$
- $\phi_0 = \mathcal{S}(\widetilde{\phi}_0, \rho_0)$

Appendix D

Multigrid Libraries

Efficient computation of the poisson equation, or other elliptic equations, is a common problem with many applications, and there exists several predeveloped and optimized libraries to help solve it. These include Parallel Particle Mesh (PPM) (Sbalzarini et al., 2006), Hypre (Falgout and Yang, 2002), Muelu (), METIS (A fast and high quality multilevel scheme for partitioning irregular graphs — Karypis Lab 2016) and PETCs (manual.pdf 2016) amongst others. There is also PiC libraries that can be used PICARD and VORPAL to mention two.

If we want to have an efficient integration of a multigrid library into our PiC model we need to consider how easy it is to use with our scalar and field structures. To have an effiecient program we need to avoid having the program convert data between our structures and the library structures. Since our PiC implementation uses the same datastructures for the scalar fields in several other parts, than the solution to the poisson equation, we could have an efficiency problem in the interface between our program and the library.

We could also consider that only part of the multigrid algorithm uses building blocks from libraries. The algorithm is now using the conceptually, and programatically easy, GS-RB as smoothers, but if we implement compatibility with a library we could easily use several other types of smoothers which could improve the convergence of the algorithm

D.1 Libraries

D.2 PPM - Parallel Particle Mesh

Parallel Particle Mesh is a library designed for particle based approaches to physical problems, written in Fortran. As a part of the library it includes a structured geometric multigrid solver which follows a similar algorithm to the algorithm we have implemented in our project implemented in both 2 and 3 dimensions. For the 3 dimensional case the laplacian is discretized with a 7-point stencil, then it uses a RB-SOR (Red and Black Succesive Over-Relaxation), which equals GS-RB with the relaxation parameter ω set as 1, as a smoother. The full-weighting scheme is used for restriction and trilinear interpolation for the prolongation, both are described in (Trottenberg et al., 2000). It has implementations for both V and W multigrid cycles. To divide up the domains between the computational nodes it uses the METIS library. The efficiency of the parallel multigrid implementation was tested

D.3 Hypre

Hypre is a library developed for solving sparse linear systems on massive parallel computers. It has support for c and Fortran. Amongst the algorithms included is both structured multigrid as well as element-based algebraic multigrid. The multigrid algorithms scales well on up to 100000 cores, for a detailed overview see Baker et. al. (2012).

D.4 MueLo - Algebraic Multigrid Solver

MueLo is an algebraic multigrid solver, and is a part of the TRILINOS project and has the advantage that it works in conjunction with the other libraries there. It is written as an object oriented solver in cpp. For a investigation into the scaling properties see Lin et. al. (2014).

D.5 METIS - Graph Partitioning Library

METIS is a library that is used for graph partitioning, and could have been used in our program to partition the grids. The partitionings it produces has been shown to be 10% to 50% faster than the partionionings produces by spectral partitioning algorithms (A fast and high quality multilevel scheme for partitioning irregular graphs — Karypis Lab 2016). It is mostly used for irregular graphs, and we are not sure if it could be easily made to work with the datastructures used throughout the program.

D.6 PETSc - Scientific Toolkit

The PETSc is an extensive toolkit for scientific calculation that is used by a multitude of different numerical applications, including FEniCS. It has a native multigrid option, DMDA, where the grid can be constructed as a cartesian grid. In addition there is large amount of inbuilt smoothers that can be used.

Bibliography

A fast and high quality multilevel scheme for partitioning irregular graphs — Karypis Lab (2016). URL: http://glaros.dtc.umn.edu/gkhome/node/107 (visited on 07/08/2016).

Adams, M. F. (2001). "A distributed memory unstructured Gauss-Seidel algorithm for multigrid smoothers". In: Supercomputing, ACM/IEEE 2001 Conference. IEEE, pp. 14–14. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1592790 (visited on 04/21/2016).

Adams, M. et al. (2003). "Parallel multigrid smoothing: polynomial versus Gauss–Seidel". In: Journal of Computational Physics 188.2, pp. 593–610. URL: http://www.sciencedirect.com/science/article/pii/S0021999103001943 (visited on 11/15/2016).

Alnæs, M. S. et al. (2011). The FEniCS Manual. October. URL: http://mmc2.geofisica.unam.mx/acl/edp/Ejemplitos/FEniCs/fenics-manual-2011-10-31.pdf (visited on 04/21/2016).

Arraras, A. et al. (2015). "Domain decomposition multigrid methods for nonlinear reaction–diffusion problems". In: Communications in Nonlinear Science and Numerical Simulation 20.3, pp. 699–710. URL: http://www.sciencedirect.com/science/article/pii/S1007570414003074 (visited on 04/21/2016).

Baker, A. H. et al. (2012). "Scaling Hypre's Multigrid Solvers to 100,000 Cores". en. In: High-Performance Scientific Computing. Ed. by M. W. Berry et al. DOI: 10.1007/978-1-4471-2437-5_13. Springer London, pp. 261–279. ISBN: 978-1-4471-2436-8 978-1-4471-2437-5. URL: http://link.springer.com/chapter/10.1007/978-1-4471-2437-5_13 (visited on 11/15/2016).

Baumjohann, W. and R. A. Treumann (1997). Basic Space Plasma Physics. en. Google-Books-ID: e4yupcOzJxkC. World Scientific. ISBN: 978-1-86094-079-8.

Bertrand, P. et al. (1990). "A nonperiodic Euler–Vlasov code for the numerical simulation of laser–plasma beat wave acceleration and Raman scattering". In: Physics of Fluids B: Plasma Physics (1989-1993) 2.5, pp. 1028–1037. ISSN: 0899-8221. DOI: 10.1063/1.859276. URL: http://scitation.aip.org/content/aip/journal/pofb/2/5/10.1063/1.859276 (visited on 08/24/2016).

Birdsall, C. K. and A. B. Langdon (2004). Plasma Physics via Computer Simulation. en. CRC Press. ISBN: 978-1-4822-6306-0.

Brackbill, J. U. et al. (1995). "Particle Simulation MethodsA Monte Carlo collision model for the particle-in-cell method: applications to argon and oxygen discharges”. In: Computer Physics Communications 87.1, pp. 179–198. ISSN: 0010-4655. DOI: 10.1016/0010-4655(94)00171-W. URL: http://www.sciencedirect.com/science/article/pii/001046559400171W (visited on 08/18/2016).

Buneman, O. (1959). “Dissipation of Currents in Ionized Media”. In: Physical Review 115.3, pp. 503–517. DOI: 10.1103/PhysRev.115.503. URL: http://link.aps.org/doi/10.1103/PhysRev.115.503 (visited on 08/19/2016).

Chen, F. F. (1984). Introduction to Plasma Physics and Controlled Fusion. en. Boston, MA: Springer US. ISBN: 978-1-4419-3201-3 978-1-4757-5595-4. URL: http://link.springer.com/10.1007/978-1-4757-5595-4 (visited on 07/12/2016).

Chow, E. et al. (2006). “A survey of parallelization techniques for multigrid solvers”. In: Parallel processing for scientific computing 20, pp. 179–201. URL: http://www.edmondchow.com/pubs/parmg-survey-siam.pdf (visited on 04/21/2016).

Courant, R. et al. (1869). “Über die partiellen Differenzengleichungen der mathematischen Physik - PPN235181684_0100___log5.pdf”. In: Mathematische Annalen 100. URL: http://www.digizeitschriften.de/download/PPN235181684_0100/PPN235181684_0100___log5.pdf (visited on 09/27/2016).

Cummings, W. D. and A. J. Dessler (1967). “Field-aligned currents in the magnetosphere”. en. In: Journal of Geophysical Research 72.3, pp. 1007–1013. ISSN: 2156-2202. DOI: 10.1029/JZ072i003p01007. URL: http://onlinelibrary.wiley.com/doi/10.1029/JZ072i003p01007/abstract (visited on 07/19/2016).

Dawson, J. (1962). “One-Dimensional Plasma Model”. In: Physics of Fluids (1958-1988) 5.4, pp. 445–459. ISSN: 0031-9171. DOI: 10.1063/1.1706638. URL: http://scitation.aip.org/content/aip/journal/pof1/5/4/10.1063/1.1706638 (visited on 08/19/2016).

Ergun, R. E. et al. (2010). “Spacecraft charging and ion wake formation in the near-Sun environment”. In: Physics of Plasmas (1994-present) 17.7, p. 072903. ISSN: 1070-664X, 1089-7674. DOI: 10.1063/1.3457484. URL: http://scitation.aip.org/content/aip/journal/pop/17/7/10.1063/1.3457484 (visited on 10/27/2016).

Falgout, R. D. and U. M. Yang (2002). “hypre: A Library of High Performance Preconditioners”. en. In: Computational Science — ICCS 2002. Ed. by P. M. A. Sloot et al. Lecture Notes in Computer Science 2331. DOI: 10.1007/3-540-47789-6_66. Springer Berlin Heidelberg, pp. 632–641. ISBN: 978-3-540-43594-5 978-3-540-47789-1. URL: http://link.springer.com/chapter/10.1007/3-540-47789-6_66 (visited on 07/08/2016).

Fitzpatrick, R. (2014). Plasma Physics: An Introduction. English. 1 edition. Boca Raton: CRC Press. ISBN: 978-1-4665-9426-5.

Friedman, A. et al. (1981). “A direct method for implicit particle-in-cell simulation”. In: Comments on Plasma Physics and Controlled Fusion 6.6, pp. 225–236. ISSN: ISSN 0374-2806. URL: http://inis.iaea.org/Search/search.aspx?orig_q=RN:15000463 (visited on 11/06/2016).

Goldston, R. J. and P. H. Rutherford (1995). Introduction to Plasma Physics. en. CRC Press. ISBN: 978-1-4398-2207-4.

Hackbusch, W. and U. Trottenberg (1982). "Multigrid methods". In: URL: http://158.69.150.236:1080/jspui/handle/961944/66389 (visited on 04/21/2016).

Hawley, J. F. and J. M. Stone (1995). "Numerical Methods in Astrophysical HydrodynamicsMOCCT: A numerical technique for astrophysical MHD". In: Computer Physics Communications 89.1, pp. 127–148. ISSN: 0010-4655. DOI: 10.1016/0010-4655(95)00190-Q. URL: http://www.sciencedirect.com/science/article/pii/001046559500190Q (visited on 08/24/2016).

Hjorth-Jensen, M. (2016). Computational Physics - Lecture Notes Fall 2013. URL: http://www.physics.ohio-state.edu/~ntg/6810/readings/Hjorth-Jensen_lectures2013.pdf (visited on 08/22/2016).

Hockney, R. W. and J. W. Eastwood (1988). Computer Simulation Using Particles. en. Google-Books-ID: nTOFkmnCQuIC. CRC Press. ISBN: 978-1-4398-2205-0.

Israeli, M. and A. Sherman (2005). "An Accurate Fourier-spectral Solver for Variable Coefficient Elliptic Equations". In: Proceedings of the 5th WSEAS/IASME International Con ISTASC'05. Stevens Point, Wisconsin, USA: World Scientific, Engineering Academy, and Society (WSEAS), pp. 152–156. ISBN: 978-960-8457-35-5. URL: http://dl.acm.org/citation.cfm?id=1373616.1373642 (visited on 11/06/2016).

Jung, M. (1997). "On the parallelization of multi-grid methods using a non-overlapping domain decomposition data structure". In: Applied Numerical Mathematics 23.1, pp. 119–137. URL: http://www.sciencedirect.com/science/article/pii/S0168927496000645 (visited on 11/15/2016).

Lapenta, G. (2016). Particle In Cell Methods. With Application to Simulations in Space. Weather. URL: https://perswww.kuleuven.be/~u0052182/pic/book.pdf (visited on 09/27/2016).

Lapenta, G. (2012). "Particle Simulations of Space Weather". In: J. Comput. Phys. 231.3, pp. 795–821. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2011.03.035. URL: http://dx.doi.org/10.1016/j.jcp.2011.03.035 (visited on 11/10/2016).

manual.pdf (2016). URL: http://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf (visited on 07/08/2016).

Miloch, W. J. (2010). "Wake effects and Mach cones behind objects". en. In: Plasma Physics and Controlled Fusion 52.12, p. 124004. ISSN: 0741-3335. DOI: 10.1088/0741-3335/52/12/124004. URL: http://stacks.iop.org/0741-3335/52/i=12/a=124004 (visited on 10/27/2016).

Miyake, Y. et al. (2013). "Plasma particle simulations of wake formation behind a spacecraft with thin wire booms". en. In: Journal of Geophysical Research: Space Physics 118.9, pp. 5681–5694. ISSN: 2169-9402. DOI: 10.1002/jgra.50543. URL: http://onlinelibrary.wiley.com/doi/10.1002/jgra.50543/abstract (visited on 10/27/2016).

Pécseli, H. L. (2012). Waves and Oscillations in Plasmas. English. 1 edition. Boca Raton: CRC Press. ISBN: 978-1-4398-7848-4.

Press, W. H. et al. (1988). "Numerical recipes in C". In: Cambridge University Press 1, p. 3. URL: http://journals.cambridge.org/abstract_S0269964800000565 (visited on 04/21/2016).

Qin, H. et al. (2013). "Why is Boris algorithm so good?" In: Physics of Plasmas (1994-present) 20.8, p. 084503. ISSN: 1070-664X, 1089-7674. DOI: 10.1063/1.4818428. URL: http://scitation.aip.org/content/aip/journal/pop/20/8/10.1063/1.4818428 (visited on 11/06/2016).

Sbalzarini, I. et al. (2006). "PPM – A highly efficient parallel particle–mesh library for the simulation of continuum systems". en. In: Journal of Computational Physics 215.2, pp. 566–588. ISSN: 00219991. DOI: 10.1016/j.jcp.2005.11.017. URL: http://linkinghub.elsevier.com/retrieve/pii/S002199910500505X (visited on 07/08/2016).

Sgattoni, A. et al. (2015). piccante: arXiv:1503.02464. DOI: 10.5281/zenodo.16097. URL: http://zenodo.org/record/16097 (visited on 08/22/2016).

Shen, J. (1994). "Efficient Spectral-Galerkin Method I. Direct Solvers of Second- and Fourth-Order Equations Using Legendre Polynomials". In: SIAM Journal on Scientific Co 15.6, pp. 1489–1505. ISSN: 1064-8275. DOI: 10.1137/0915089. URL: http://epubs.siam.org/doi/abs/10.1137/0915089 (visited on 04/21/2016).

Shu, F. H. (2010). The Physics of Astrophysics Volume I: Radiation. English. Mill Valley, Calif.: University Science Books. ISBN: 978-1-891389-76-4.

Stüben, K. (2001). "A review of algebraic multigrid". In: Journal of Computational and Applied 128.1, pp. 281–309. URL: http://www.sciencedirect.com/science/article/pii/S0377042700005161 (visited on 04/21/2016).

Swendsen, R. H. (2006). "Statistical mechanics of colloids and Boltzmann's definition of the entropy". In: American Journal of Physics 74.3, pp. 187–190. ISSN: 0002-9505, 1943-2909. DOI: 10.1119/1.2174962. URL: http://scitation.aip.org/content/aapt/journal/ajp/74/3/10.1119/1.2174962 (visited on 07/22/2016).

Trottenberg, U. et al. (2000). Multigrid. Academic press. URL: https://www.google.com/books?hl=en&lr=&id=9ysyNPZoR24C&oi=fnd&pg=PP1&dq=trottenberg+2000&ots=rJCHSPzSMY&sig=sin3i-gmWOoykoTFnyHGIPZXT5Q (visited on 04/21/2016).

Verboncoeur, J. P. (2005). "Particle simulation of plasmas: review and advances". en. In: Plasma Physics and Controlled Fusion 47.5A, A231. ISSN: 0741-3335. DOI: 10.1088/0741-3335/47/5A/017. URL: http://stacks.iop.org/0741-3335/47/i=5A/a=017 (visited on 07/21/2016).

Watanabe, K. and T. Sato (1990). "Global simulation of the solar wind-magnetosphere interaction: The importance of its numerical validity". en. In: Journal of Geophysical Research 95.A1, pp. 75–88. ISSN: 2156-2202. DOI: 10.1029/JA095iA01p00075. URL:

http://onlinelibrary.wiley.com/doi/10.1029/JA095iA01p00075/abstract (visited on 08/24/2016).

Zhukov, V. T. et al. (2014). “Parallel multigrid method for solving elliptic equations”. en. In: Mathematical Models and Computer Simulations 6.4, pp. 425–434. ISSN: 2070-0482, 2070-0490. DOI: 10.1134/S2070048214040103. URL: http://link.springer.com/article/10.1134/S2070048214040103 (visited on 10/17/2016).