Development of a Parallel Multigrid Field Solver for Large-Scale Particle-in-Cell Applications

Inauguraldissertation

zur Erlangung des Doktorgrades
der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von

Michael Becker

geboren am 13. November 1987 in Giessen

Institut für Theoretische Physik
Justus-Liebig-Universität Giessen

Oktober 2018

Contents

List of Figures

Chapter 1

Introduction

The simulation of low temperature plasmas requires a particle-based approach, for which the movement and interaction of individual charged particles is modeled. For this, the forces on the particles need to be calculated. This is done by a so-called *field solver* module, which calculates the electromagnetic fields emerging from the present charge distribution and external sources such as electrodes and electric currents through conductors by solving Maxwell's equations.

1.1 Motivation

Depending on the range of application, the implementation of a field solver can be a difficult and complex task. While a wide range of methods for the numerical solution of the emerging partial differential equations exists, their computational costs mostly increase disproportionately with the size of the considered system. Such a behavior is generally undesirable because it results in parts of the simulation becoming increasingly dominant regarding their runtime, which is ultimately a bottleneck for the simulation of larger systems.
A large simulation domain further requires an efficient parallelization of every part of the program to cut down the otherwise excessive runtimes. For this, the field solver is the most critical component because the electromagnetic fields at every point are affected by the properties of the entire rest of the domain.

For the further development of PlasmaPIC, a 3D plasma simulation tool developed at the University of Gießen, a new field solver is needed to allow for progress towards the simulation of increasingly larger plasma discharges such as those inside radio-frequency ion thrusters. Since all other program parts already scale appropriately due to their localized character, this is the only significant algorithmic bottleneck limiting PlasmaPIC's applicability.
The fundamental requirements for this new solver are that the underlying solution process scales linearly with the size of the system and that it is sufficiently parallelizable in order to ensure a relatively constant runtime if the number of processors used for the simulation is scaled up proportionally to the system size.

Such characteristics are prominently featured by the class of multigrid methods, which use a hierarchy of nested grids with varying mesh size to improve the convergence of

the solution process.
Hence the goal of this thesis can be formulated: The development and implementation of a suitable parallel multigrid method that can be used as PlasmaPIC's field solver.

The effort of developing a solver specifically tailored for PlasmaPIC instead of implementing an external library or software suite such as the well-known PETSc is justified by the possibilities this approach opens up. Not only can the solver be optimized specifically for PlasmaPIC by using the same data structures, it can also be controlled and adjusted with the greatest flexibility.

1.2 Technical and Scientific Background

Satellites and other spacecraft rely on propulsion systems to manipulate their orbital parameters. Based on the principle of recoil, a propellant is accelerated and ejected to generate thrust, which is used for orbital maneuvers or position and attitude control. Many satellite missions, including the International Space Station (ISS), regularly require orbital correction to either counteract perturbations caused by lunar and solar gravity or to compensate the atmospheric drag in low Earth orbits.
Since the task of transporting additional weight in the form of fuel to space is first and foremost expensive, fuel efficiency is a matter of great interest. Chemical thrusters, while being a proven, sophisticated and reliable technology, are characterized by the fact that the energy used to generate thrust is stored in the propellant. The maximum exhaust velocity is therefore limited by the chemical processes in the combustion chamber, typically to 3 to 4 km/s [1].
Electrically powered propulsion systems on the other hand transfer electrical energy, generated, e. g., by solar cells, to the propellant and are therefore able to achieve much higher exhaust velocities of up to 100 km/s. The propellant mass is thus used much more efficiently, although the achievable thrust levels generally by far don't reach those of chemical thrusters due to mass flow and energy restrictions. If the desired adjustments to the spacecraft's orbital parameters don't have to be made in a very short time frame, electric propulsion, however, is usually the superior concept. Since many approaches allow for the thrusters to principally operate for very long periods of time (i. e., non-stop for multiple years), the cumulative thrust levels can even surpass those of chemical thrusters, which enables application even to interplanetary and deep space missions.

Aside from experimental designs, industry and academia focus mostly on research and development of ion thrusters, which are based on accelerating ions in electrostatic fields. Here, Xenon is usually the propellant of choice, as it offers many favorable properties, i. e., low ionization energy, high atomic mass and non-corrosive behavior. Since it's a rather rare trace gas in Earth's atmosphere and therefore expensive to obtain, finding viable alternatives is a matter of increasing interest.
Hall-effect thrusters (HET) consist of a vessel in the shape of a cylinder ring that is open at one end. A constant radial magnetic field and an electric field between the anodic backplate and an electron emitting cathode (*neutralizer*) at the open end are established. The electrons form an azimuthal Hall current, causing them to effectively be trapped in the vessel and ionizing the propellant gas entering through a gas inlet at the anode. The electric field then accelerates the ions out of the vessel. The so formed ion beam neutralizes itself by pulling the respective amount of electrons, emitted by the

neutralizer, with it.
Hall-effect thrusters distinguish themselves by producing comparably high thrusts at a given power level. Their design is furthermore rather simple and is thereby considered to be very reliable regarding the fail-safety of their electronics.

Gridded ion thrusters generate a plasma discharge inside a vessel and extract the ions through a biased grid system. Here, too, a dedicated neutralizer then emits an equal amount of electrons to prevent charging of the spacecraft. Various approaches on how to generate the plasma exist. For ion thrusters of *Kaufman type*, a hollow cathode inside the discharge chamber emits electrons, which are accelerated towards the innermost extraction grid. At high enough kinetic energy, these electrons can partake in electron impact ionization of the gaseous propellant, thus creating and sustaining a plasma (*direct current, DC*). A constant axial magnetic field (perpendicular to the grids) is generally used to lengthen the electrons' path in order to maximize the collision probability before contact with a wall or a grid. Additionally, the newly freed electrons can also reach the required energy for further ionizations.

The concept of *radio-frequency ion thrusters (RIT)* was first proposed in 1962 by H. W. Löb [2] and has been under investigation at the University of Gießen ever since. Here, the electrons are not accelerated between electrodes, but by an oscillating electric field induced by the alternating magnetic field of a coil wrapped around the discharge chamber (cf. figure 1.1). This can easily be described for an idealized set-up, where the coil is assumed to be cylindrical and long and end effects are neglected: Under these conditions, the magnetic field $\boldsymbol{B}$ can be expressed as

$$\boldsymbol{B}(t) = \mu_0 \, n_\mathrm{c} \, I_0 \exp\left(i \, \omega \, t\right) \, \boldsymbol{e}_\mathrm{z} \tag{1.1}$$

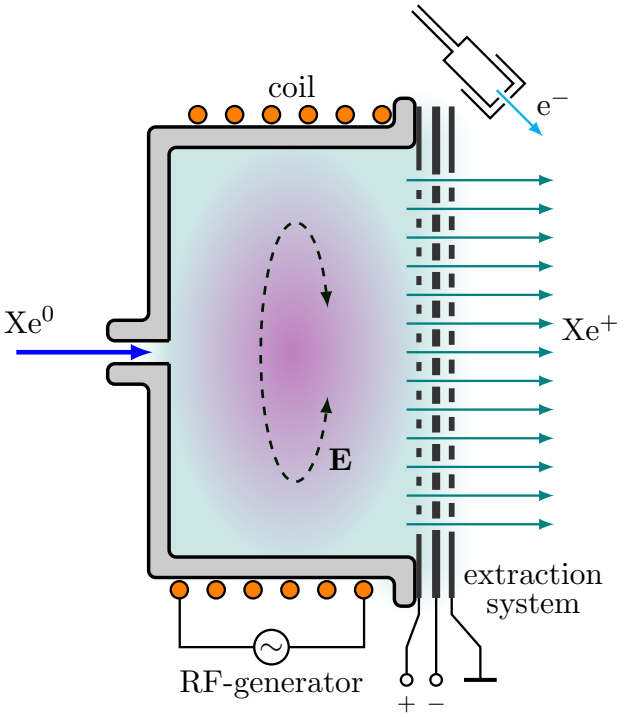

Figure 1.1: Principle of a radio-frequency ion thruster

with the vacuum permeability μ_0, the number of coil turns per unit length n_c, the coil current I_0 and the applied angular frequency ω.
Faraday's law of induction,

$$\boldsymbol{\nabla} \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \ , \tag{1.2}$$

is then solved by the azimuthal electric field

$$\boldsymbol{E}(r,t) = -\frac{i\,\omega\,r}{2}\, B_{\mathrm{z}0} \exp\left(i\,\omega\,t\right)\, \boldsymbol{e}_\varphi \ . \tag{1.3}$$

A direct consequence of this electrodeless method to sustain a so-called *inductively coupled plasma (ICP)* is that the lifetime of the thruster is only limited by the grid system. In a Kaufman thruster on the other hand, the electron emitting hollow cathode is directly exposed to the plasma, allowing for plasma etching to take place.
The lack of an electron emitter can be considered as a downside, as it complicates the ignition process. Since the heating mechanism, electron impact ionization, relies on free electrons to exist, they have to be created otherwise. However, it's possible to, e. g., utilize the neutralizer for this purpose by deactivating the voltage of the grid system.

Two or three grids are generally used to extract the ions. The *screen grid* has direct contact with the plasma and is positively biased on the order of one kilovolt compared to the rest of the spacecraft. Ions passing it are accelerated towards the *accel grid* at either ground (space) potential for two-grid systems or low-negative potential for three-grid systems. The *decel grid* can be used to positively influence thruster lifetime by preventing the backstreaming of electrons and charge-exchanged ions. However, the lifetime of the extraction system is primarily restricted due to erosion of the accel grid by high-energy ions deviating from their ideal trajectory. Since this is caused by collisions with the neutral gas or other ions (with possible charge exchange), it can hardly be prevented completely. For this reason, the accel grid is generally designed to be significantly thicker than the other grids.
Nevertheless, the reported maximal periods of operation for gridded ion thrusters ($\lesssim 5 \cdot 10^4$ h [3]) exceed those of Hall-effect thrusters ($\sim 10^4$ h [4]) many times over, which is attributed to erosion of the discharge chamber of HETs by energetic ions.
The grid system is furthermore an effective barrier for the unionized propellant gas, as a neutral particle entering an aperture is likely to be reflected back into the discharge chamber by diffuse reflection on a grid surface (free molecular flow regime). Positive ions on the other hand have a transmission coefficient very close to one because the extraction system's ion optics are designed to support a narrow range of trajectories. Consequently, a very high mass utilization efficiency of the propellant is possible ($\lesssim 90\,\%$ [5]), although the actual degree of ionization is very low ($\sim 1\,\%$).

The Child-Langmuir law

$$j = \frac{4}{9}\,\varepsilon_0 \left(\frac{2\,e}{m_\mathrm{i}}\right)^{1/2} \frac{U^{3/2}}{d^2} \tag{1.4}$$

and the breakdown voltage between two grids limit the ion flow j through an aperture of the extraction system (here, U is the voltage between screen and accel grid and d is the distance between them; m_i is the ion mass). For this reason, and to adapt to the different demands for propulsion systems, gridded ion thrusters are designed in various sizes, allowing for the number of grid apertures to vary greatly. Thus, the thrust range for which a thruster can be used varies likewise.

The radio-frequency ion thrusters developed in Gießen have the diameter of their discharge chamber (in cm) in their name. The original RIT-10 was both miniaturized (RIT-2.5 [6]) and scaled up (RIT-35 [7]), leading to a total thrust range of 50 μN to several hundred N of the RIT family.

For the purpose of optimizing the design of a given RIT, further insight into the properties of the plasma is required. Since diagnostic methods are either invasive (and therefore a complicated matter, especially for the small thrusters) or limited to characteristics that are accessible from outside the thruster (positional and energetic distribution of the ion beam, currents on the grids, temperatures, etc.), simulations are a viable alternative.

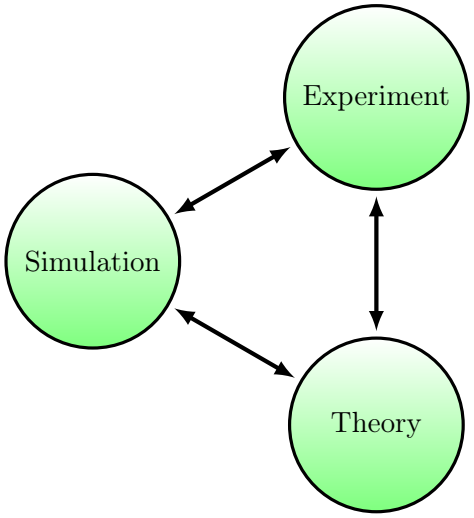

Figure 1.2: Interplay between theory, experiment and simulation

Scientific simulations are universally carried out to improve the understanding of how fundamental physical processes affect a broader context and to help interpret experiments for which, as in RITs, detailed information is hardly accessible. In the vast majority of cases, they also offer a faster and cheaper approach at designing and predicting new experimental setups. For this reason, they are generally not referred to as a part of either theoretical or experimental methods but as a distinct branch (figure 1.2). In the scope of his PhD thesis, Robert Henrich developed a powerful 3D plasma simulation tool, *PlasmaPIC*, capable of modeling the discharges in small thrusters of μN range in a reasonable time frame, facilitated by massive parallelization [8].
In order to simulate the low temperature plasmas present in a RIT, it utilizes the particle-in-cell (PIC) method, which is described in chapter 2 in further detail.

PlasmaPIC's great capabilities reach a limit when it comes to scaling the problem (i. e., the RIT) up to larger sizes. Since for a three-dimensional code, increasing the characteristic length of a problem equates to cubic scaling of the total system size, algorithmic scaling quickly becomes a dominant issue. If not all incorporated algorithms scale linearly with the system size (i. e., optimal), the additional computational costs can't always be balanced by utilizing more parallel processes and the total runtime is affected negatively. Linearly scaling algorithms on the other hand, if properly parallelized, can then be utilized to keep the simulation's total runtime constant. I. e., a doubling in system size can be compensated by running the simulation on twice as many processors. Since the eligible high-performance computing clusters can generally

be expected to grow in size, increasingly large systems can therefore be simulated in the years to come.
The development of a new field solver that ensures this parallel performance of PlasmaPIC can therefore be understood as both an immediate improvement to current simulations and an anticipatory measure.

1.3 Outline

In the following chapter, the particle-in-cell method is introduced and explained by the example of PlasmaPIC. The fundamentals of the multigrid methodology are then laid down in chapter 3, whereas the adjustments and modifications that were made to enable its application in PlasmaPIC are described in chapter 4. The performance of the final parallel multigrid field solver is then evaluated in chapter 5.

Chapter 2

The Particle-in-Cell Method by the Example of PlasmaPIC

The approaches to simulate a plasma can be categorized into two groups: *fluid* and *kinetic* models [9]. A fluid model, such as magnetohydrodynamics, describes the plasma as a continuum, using distribution functions that therefore need to be available. For low-temperature plasmas, such as those produced in a RIT, this is not the case, as the continuum approach breaks down.
Kinetic models on the other hand treat the plasma as a collection of particles with individual velocities. Here, either a continuous particle velocity distribution function is produced by solving a kinetic equation (e. g., the Boltzmann, Vlasov or Fokker-Planck equation) or the movement and interaction of individual simulated particles is tracked explicitly. For the latter variant to be practicable, the number of simulated particles usually needs to be much smaller than the number of particles in the actual plasma. This is achieved by introducing so-called *super-particles*, each representing numerous real particles, carrying their collective charge and mass.
These particle-based approaches can then again be grouped into particle-particle and particle-mesh methods [10]. The former approach involves calculating the forces between every pair of particles, which results in excessive computational costs as the number of simulated particles increases.
PlasmaPIC utilizes the latter approach, alternatively called *particle-in-cell* (*PIC*). Here, a mesh covers the simulation domain, defining cells and grid points (cf. figure 2.1). Instead of calculating the inter-particle forces directly, a discrete vector field is associated with the grid points and the force on each particle is calculated based on interpolation from its surrounding grid points. The movement of the particles is then simulated by updating position and velocity while advancing on discrete time steps Δt. After consideration of boundary effects and direct collisions, a new vector field can be calculated from the particles' distribution of position and velocity. For the direct collisions, PlasmaPIC utilizes Monte Carlo collisions, extending the method to a *particle-in-cell Monte Carlo collision* approach (*PIC-MCC*) [8].

In the following, the algorithm of one fundamental PIC cycle is described with an emphasis on the specific approaches of PlasmaPIC. The subsequent sections then provide a discussion of the numerical constraints of the PIC method and its parallelizability, as well as a comprehensive description of the features of PlasmaPIC not mentioned

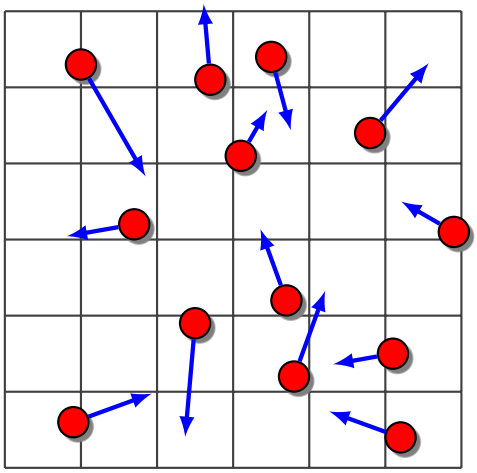

Figure 2.1: Principle of the particle-in-cell method: The particles are moving continuously through the grid cells

beforehand.

2.1 Algorithm

The procedure of a PIC simulation can be described as the processing of a number of modules of mutual dependency in cyclic succession, where a new time step is defined by an additional cycle through every module (figure 2.2).

2.1.1 Particle Weighting

Starting from a predefined distribution of particles with individual position and velocity, the electric charge density ρ and current density $\boldsymbol{j}$ are calculated for the grid points by assigning individual contributions of the particles. For this, various approaches are possible, with the simplest and computationally cheapest being the *nearest-grid-point (NGP)* method, where a particle's charge is assigned to the grid point within closest proximity. In contrast to this *zero-order weighting*, PlasmaPIC utilizes *first-order weighting*, where every particle is weighted linearly to the eight grid points that define the corners of the cell it's positioned in. This approach greatly reduces the statistic noise generated by a particle traversing a cell by preventing a sudden change of the associated grid point. Higher-order weighting schemes involving even more grid points per particle generally result in further improved accuracy but are rarely used due to the additional computational costs.

2.1.2 Solution of Maxwell's Equations

After the particle weighting, discretized fields for both charge (density) and current (density) are available and can be used to solve the Maxwell equations in order to derive the electromagnetic fields that define the force on every particle.

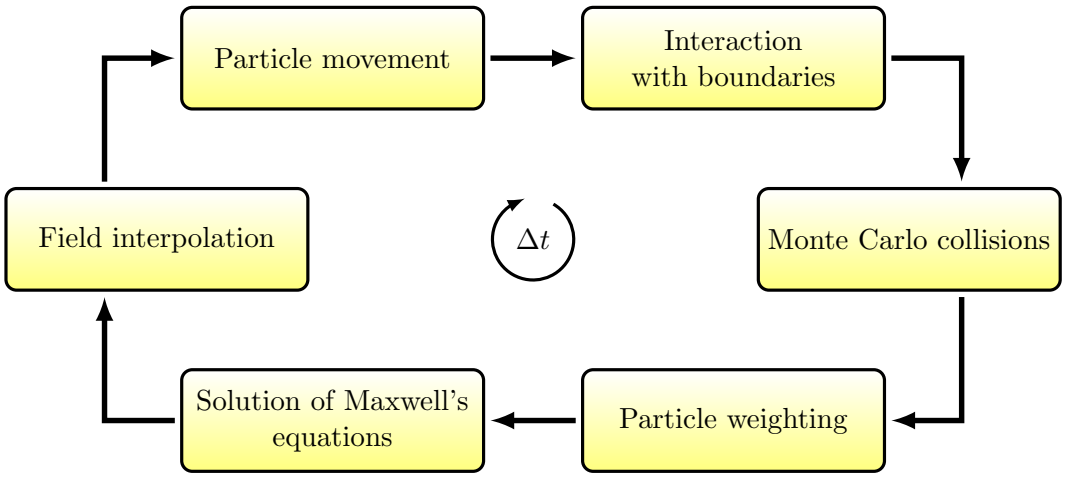

Figure 2.2: Particle-in-cell algorithm

A well-known expression for the electric field satisfying Maxwell's equations is

$$\boldsymbol{E}(\boldsymbol{r},t) = -\boldsymbol{\nabla}\Phi(\boldsymbol{r},t) - \frac{\partial}{\partial t}\boldsymbol{A}(\boldsymbol{r},t) \ , \tag{2.1}$$

which is a motivation to separate the solution into independently calculatable electrostatic and an electrodynamic components of the electric field [11]. Φ is the electrostatic potential, $\boldsymbol{A}$ the magnetic vector potential. The electrostatic term,

$$\boldsymbol{E}_{\text{el.stat.}}(\boldsymbol{r},t) = -\boldsymbol{\nabla}\Phi(\boldsymbol{r},t) \ , \tag{2.2}$$

can then be found by solving the Poisson equation,

$$\Delta\Phi(\boldsymbol{r},t) = -\frac{\rho(\boldsymbol{r},t)}{\varepsilon_0} \ , \tag{2.3}$$

and calculating the potential difference over two cells.

For the electrodynamic part, PlasmaPIC utilizes the common approach for inductively coupled plasmas of assuming the relevant physical quantities to be varying harmonically in time with the applied angular frequency ω [11, 12]:

$$\boldsymbol{E}_{\text{el.dyn.}}(\boldsymbol{r},t) = \hat{\boldsymbol{E}}(\boldsymbol{r})\, e^{i\,\omega\, t} \tag{2.4}$$
$$\boldsymbol{B}(\boldsymbol{r},t) = \hat{\boldsymbol{B}}(\boldsymbol{r})\, e^{i\,\omega\, t} \tag{2.5}$$
$$\boldsymbol{A}(\boldsymbol{r},t) = \hat{\boldsymbol{A}}(\boldsymbol{r})\, e^{i\,\omega\, t} \tag{2.6}$$
$$\boldsymbol{j}(\boldsymbol{r},t) = \hat{\boldsymbol{j}}(\boldsymbol{r})\, e^{i\,\omega\, t} \ . \tag{2.7}$$

The complex amplitudes are then time-independent and the simple relation

$$\boldsymbol{E}_{\text{el.dyn.}}(\boldsymbol{r},t) = -i\,\omega\,\boldsymbol{A}(\boldsymbol{r},t) \tag{2.8}$$

holds. The amplitudes are furthermore only calculated once every RF cycle and subsequent values are determined by multiplication with the exponential term.
Equation (2.8) implies that in principle, the Biot-Savart law can be used to calculate the (electrodynamic) electric field for every grid point:

$$\hat{\boldsymbol{E}}(\boldsymbol{r}) = -i\,\omega\frac{\mu_0}{4\pi}\int d^3r'\,\frac{\hat{\boldsymbol{j}}(\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|} \ . \tag{2.9}$$

The total current density of a RIT system consists of contributions of both the electric current through the coil and moving charges in the plasma. Because the mobility of the electrons in the plasma is much higher than that of the ions, the ion current can furthermore be neglected:

$$\hat{\boldsymbol{j}}(\boldsymbol{r}) = \hat{\boldsymbol{j}}_{\text{coil}}(\boldsymbol{r}) + \hat{\boldsymbol{j}}_{\text{plasma}}(\boldsymbol{r}) \simeq \hat{\boldsymbol{j}}_{\text{coil}}(\boldsymbol{r}) + \hat{\boldsymbol{j}}_{\text{e}}(\boldsymbol{r}) \ . \tag{2.10}$$

However, as for each grid point to be calculated the current density $\hat{\boldsymbol{j}}$ of every other grid point and of the (discretized) coil is needed, the computational costs are still immense. Instead, equation (2.9) is only used to determine the boundary conditions.

The Coulomb gauge ($\boldsymbol{\nabla} \cdot \boldsymbol{A} = 0$) allows Ampère's circuital law to be rearranged:

$$\boldsymbol{\nabla} \times \boldsymbol{B} = \mu_0 \, \boldsymbol{j} + \mu_0 \, \varepsilon_0 \frac{\partial \boldsymbol{E}}{\partial t} \tag{2.11}$$

$$-\Delta \hat{\boldsymbol{A}} = \mu_0 \, \hat{\boldsymbol{j}} + i \, \omega \, \mu_0 \, \varepsilon_0 \, \hat{\boldsymbol{E}} \tag{2.12}$$

$$\left(\Delta + \mu_0 \, \varepsilon_0 \, \omega^2\right) \hat{\boldsymbol{E}} = i \, \omega \, \mu_0 \, \hat{\boldsymbol{j}} \ . \tag{2.13}$$

Due to its complex properties, equation (2.13) can be separated into a system of six independent equations, that are, like Poisson's equation (equation (2.3)), elliptic partial differential equations, which can be solved using the same numerical techniques.
By neglecting the term $\mu_0 \, \varepsilon_0 \, \omega^2 \ll 1$, all relevant equations have Poisson form (in the mathematical sense) and can therefore even be solved in the same way.

The magnetic field $\boldsymbol{B}$ can then be calculated using

$$\hat{\boldsymbol{B}}(\boldsymbol{r}) = \frac{i}{\omega} \, \boldsymbol{\nabla} \times \hat{\boldsymbol{E}}(\boldsymbol{r}) \ . \tag{2.14}$$

How to efficiently solve equation (2.3) and equation (2.13) is the central task of this work. After the fundamentals of the multigrid methods are introduced in chapter 3, chapter 4 describes the adjustments made to develop a highly efficient field solver.

2.1.3 Field Interpolation

The electric and magnetic fields need to be interpolated to every particle position. This is done complementary to the particle weighting scheme to avoid self-acceleration, i. e., via linear interpolation from the eight grid points within closest proximity.

2.1.4 Particle Movement

In order to update the particles' position and velocities, the two first-order differential equations

$$m \, \frac{d\boldsymbol{v}}{dt} = \boldsymbol{F} \quad \text{and} \tag{2.15}$$

$$\frac{d\boldsymbol{r}}{dt} = \boldsymbol{v} \tag{2.16}$$

are replaced by the finite-difference equations

$$m \, \frac{\boldsymbol{v}_{\text{new}} - \boldsymbol{v}_{\text{old}}}{\Delta t} = \boldsymbol{F}_{\text{old}} \quad \text{and} \tag{2.17}$$

$$\frac{\boldsymbol{r}_{\text{new}} - \boldsymbol{r}_{\text{old}}}{\Delta t} = \boldsymbol{v}_{\text{new}} \ . \tag{2.18}$$

This is the commonly used *leap-frog method*, which is time-centered and second-order accurate [9].
Furthermore, since the Lorentz force

$$\boldsymbol{F} = q\,(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}) \tag{2.19}$$

is not only dependent on the fields $\boldsymbol{E}$ and $\boldsymbol{B}$ and the electric charge q, but also on the particles' velocity, it needs to be used carefully to preserve accuracy. One way to achieve this is to use Boris' method [13], for which the respective particle is first accelerated for half of the time step according to the electric field. The intermediate velocity vector is then rotated by the magnetic field, before the particle is accelerated again by the electric field till the end of the time step.

2.1.5 Interaction with Boundaries

If the path between old and new position of a particle intersects with a boundary, a particle-boundary interaction must be carried out. Dependent on the type of boundary, various effects are possible. A striking particle can either increase the surface charge of a dielectric at the position of impact or be absorbed by a conducting material, where the object's overall charge remains unchanged (equipotential boundary) or increases (conductor or current driven). Either way, the particle itself is removed from the simulation, just as if it leaves through an exit.
However, secondary electron emission, which generates new particles, is possible with a predefined probability.

2.1.6 Monte Carlo Collisions

PlasmaPIC utilizes Monte Carlo collisions to describe the interaction between the plasma species and the neutral background gas. By this, essential plasma features like electron heating, the production of new ion-electron pairs via electron impact ionization, and energy losses through transfer to the neutral gas are enabled to take place in the simulation.
The neutral gas is assumed to be unaffected by the plasma, so that its distribution function (to be specified in the input card) doesn't change over time.

The probability P_{coll} of a plasma particle striking a neutral gas particle is given by

$$P_{\text{coll}} = 1 - \exp\left(-n(\boldsymbol{r})\,\sigma(E)\,v\,\Delta t\right)\ . \tag{2.20}$$

Here, $n(\boldsymbol{r})$ is the number density of the background gas and $\sigma(E)$ is the sum of all cross sections for collisions at the kinetic particle Energy E. Instead of applying the straightforward approach of comparing every particles value of P_{coll} with a random number between zero and one to check for collision, the *null collision process* is used in PlasmaPIC [14].
It essentially involves calculating a maximum number of collisions $N_{\max}$ which might occur inside a cell, randomly choosing $N_{\max}$ particles from the cell, and checking for each of those particles, if and what type of collision takes place. By only examining a fraction of the total number of particles, the required computations are reduced significantly.
Particles that undergo a collision are ultimately assigned a new velocity vector.

For a plasma that is sustained at such low pressures as in a RIT, interactions between charged particles, e. g., elastic Coulomb collisions or electron-ion recombination, don't need to be considered. The relevant interaction modes are elastic scattering, excitation, and ionization for electron-neutral collisions and elastic scattering for ion-neutral collisions (possibly including charge exchange).

2.2 Numerical Constraints

The accuracy of a PIC simulation depends on a few fundamental parameters: the length of a time step Δt, the size of a grid cell Δx and the number of simulated particles per Debye length N_D.

The highest temporal frequency of a plasma is commonly associated with the plasma frequency

$$\omega_\mathrm{p} = \left(\frac{e^2\, n_e}{\varepsilon_0\, m_\mathrm{e}} \right)^{1/2} . \tag{2.21}$$

To resolve it, the time step needs to be appropriately small. A simple assessment provides the relation $\omega_\mathrm{p}\, \Delta t < 2$.
However, practical applications require a stricter limit [9, 15], often set to

$$\omega_\mathrm{p}\, \Delta t \lesssim 0.2 . \tag{2.22}$$

The smallest distance of interest is the Debye length,

$$\lambda_\mathrm{D} = \left(\frac{\varepsilon_0\, k_\mathrm{B}\, T}{e\, n_\mathrm{e}} \right)^{1/2} , \tag{2.23}$$

for which

$$\Delta x \leq \lambda_\mathrm{D} \tag{2.24}$$

is a sufficient condition [15], although it's possible to successfully conclude a simulation with a looser requirement for the cell size, e. g., $\Delta x \leq 3.4\, \lambda_\mathrm{D}$ [10].

The number of simulated particles per Debye length N_D, if chosen too small, is not only directly correlated with numerical noise but also impedes the shielding characteristics of the plasma and produces unphysical heating [15]. A widely accepted range for N_D is 10 to 100 [10] but large deviations to higher values are necessary under certain circumstances [16].
Furthermore, N_D directly affects the ratio of real to simulated particles and the number of particles per cell and therefore the computational costs for most of the PIC modules.

2.3 Parallelizability

PlasmaPIC is parallelized by utilizing the domain decomposition method, which is the standard approach to partition problems that are defined on a spatial domain. The program is started on a (possibly very large) number of processors, each of which is assigned a (cuboidal) sub-space as its own local simulation domain. With the exception of the field solver, every implemented PIC module can be processed almost entirely independent on each sub-space, so that the computations necessary for the simulation are

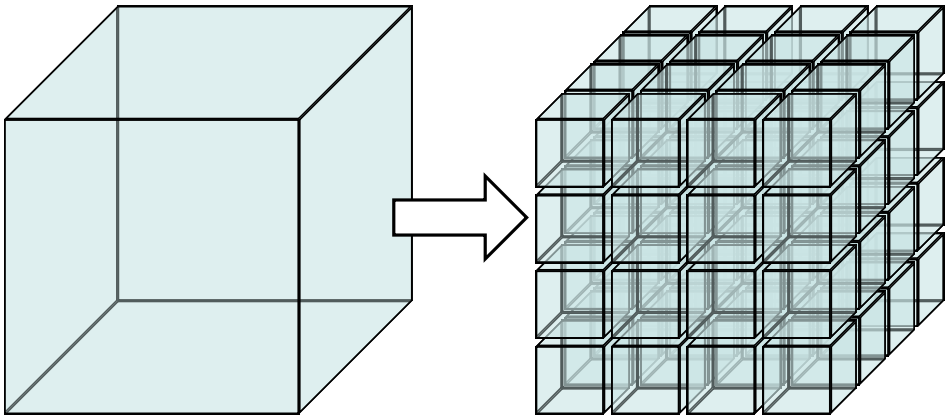

Figure 2.3: Application of the domain decomposition method

distributed very efficiently. In order to exchange boundary information to connect the numerous subdomains, the *message passing interface (MPI)* [17] is used. This communication standard for parallel computing provides the syntax and semantics for various library implementations that can be used for inter-process communication. Applications that utilize MPI can be designed to be scalable to both system size and number of involved processes on distributed memory architectures such as many high-performance computing clusters.
MPI further supports non-blocking send- and receive-operations to allow for messages to be exchanged while computations that are not immediately dependent on other processes' data are simultaneously performed. By this, it's possible for applications to completely hide the communication process so that the total runtime is not influenced by it. However, this is naturally only possible if the amount of information to be exchanged isn't too high compared to the number of potential concurrent computations and if the hardware available for communication (e. g., an InfiniBand network) can handle the data throughput in time.

Out of the six modules described in section 2.1, three don't require any communication at all because the preceding modules already made all necessary data available: The field interpolation process transfers data associated with the grid points into the grid cells, and the interactions of the particles with boundaries and the neutral background gas are restricted to the respective cell.
Weighting the particles' charge and current to the grid on the other hand is not possible without some communication because a grid point is assigned contributions of particles from the neighboring eight cells, which can, in principle, belong to up to eight different processes.
With the particle mover, communication needs to take place when particles are moved to a position outside of the process-local subdomain. The neighboring process then needs the six values for position and velocity of every crossing particle.

The module that requires the most communication is the field solver. While the other parts of PlasmaPIC show practically ideal scaling behavior for a wide range of setups, this can only be achieved for relatively large subdomains here, where the ratio of communication to computation (determined by the ratio of surface area of the subdomain to its volume) is favorable [8].
This becomes clear by acknowledging the fact that the electric potential at an arbitrary

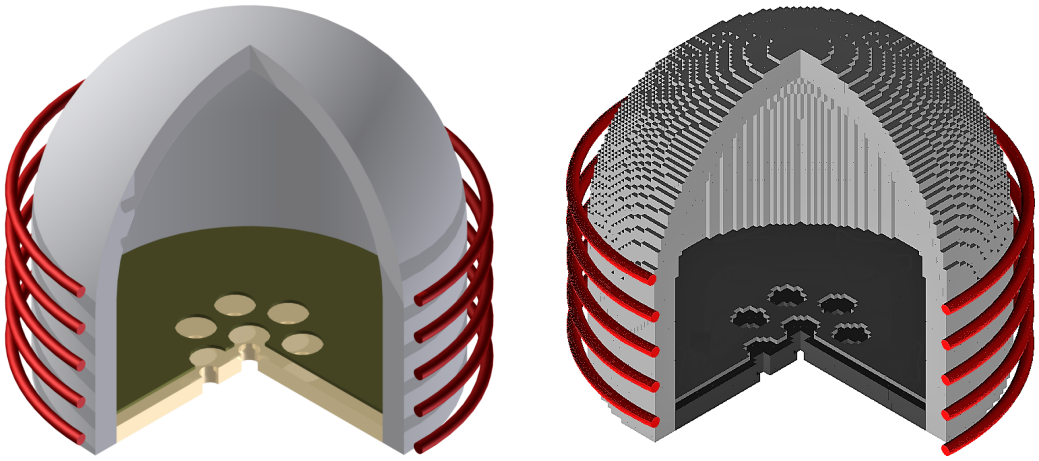

Figure 2.4: Sectional view of the RIT-1.0. Left: Generated from a CAD file. Right: Representation in PlasmaPIC [8].

position depends on charge distribution and boundary conditions of the whole simulation domain, so that simply exchanging boundary information can hardly be sufficient when done in a single step.
An efficient parallelization is therefore a significant requirement for the field solver developed in the context of this work.

2.4 Further Features of PlasmaPIC

For the purpose of setting the right general conditions, PlasmaPIC incorporates a *direct simulation Monte Carlo (DSMC)* tool to separately model the neutral gas. In principle, this allows for a number density distribution to be found so that every grid cell is assigned an individual value. Since the variations in density inside the discharge chamber of a RIT have shown to be rather small [8], the assumption of a homogeneous distribution with Maxwellian velocity distribution was justified, which reduces the computational costs of the MCC module.

It's furthermore possible to import (multiple) arbitrarily shaped geometries, created by using an external CAD program, to define the boundaries. For this, PlasmaPIC reads the predefined files and maps the triangular representation of surfaces onto the structured Cartesian grid.

Thus, PlasmaPIC is not limited to the simulation of radio-frequency ion thrusters, but is able to model a broad range of applications, including direct-current (DC) discharges, conductively coupled plasmas (CCP) and, of course, inductively coupled plasmas (ICP).

Chapter 3

Fundamentals of Multigrid

This chapter outlines the fundamentals and applications of the multigrid methods. The derivations are generally based on *vertex-centered* discretization, where the grid points are positioned at the vertices of the (cubical) cells into which the domain of interest is divided. This approach (as opposed to *cell-centered* discretization where each cell holds a single grid point at its center) is justified because it is the literature standard for outlining the multigrid basics and it is already utilized by PlasmaPIC throughout all modules. The multigrid methods can, however, be applied to cell-centered grids as well and without fundamental differences.

3.1 Preparations

The following remarks are based on the Poisson equation

$$\Delta u(\boldsymbol{r}) = -f(\boldsymbol{r}) \tag{3.1}$$

with the Laplace operator Δ and the real-valued functions $u(\boldsymbol{r})$ and $f(\boldsymbol{r})$ on $\mathbb{R}^3$ (of which f is given) to provide an example for the general numerical solution of elliptic partial differential equations.
Finding the solution of

$$\Delta\Phi(\boldsymbol{r}) = -\frac{\rho(\boldsymbol{r})}{\varepsilon} \ , \tag{3.2}$$

namely the electric potential $\Phi(\boldsymbol{r})$ for a given charge distribution $\rho(\boldsymbol{r})$ and permittivity ε, is the fundamental problem of electrostatics.

3.1.1 Finite Differences

The second derivative of a smooth (one-dimensional) function $u(x)$ can be approximated on a Cartesian grid with discrete values u_i and mesh size h by

$$\frac{\partial^2 u(x)}{\partial x^2} \approx \frac{u(x-h) - 2\,u(x) + u(x+h)}{h^2} = \frac{u_{i-1} - 2\,u_i + u_{i+1}}{h^2} \ , \tag{3.3}$$

which can be derived by adding the two Taylor approximations to third order $u(x_0+\Delta x)$ and $u(x_0-\Delta x)$ [18, 19]:

$$u(x_0 \pm \Delta x) = u(x_0) \pm \Delta x\, u'(x_0) + \frac{1}{2}\Delta x^2\, u''(x_0) \pm \frac{1}{6}\Delta x^3\, u'''(x_0) + \mathcal{O}(\Delta x^4) \tag{3.4}$$

$$u''(x_0) = \frac{u(x_0+\Delta x) - 2\,u(x_0) + u(x_0-\Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^2)\ . \tag{3.5}$$

The so-called *truncation* or *discretization error* due to approximating the continuous space by a finite grid is therefore of order $\mathcal{O}(\Delta x^2) = \mathcal{O}(h^2)$.
Alternatively, this can be obtained via the second derivative of a simple polynomial interpolation of $u(x)$ with its neighboring grid points $u(x-h)$ and $u(x+h)$ (three-point approximation for the second-derivative).

Consequently, the three-dimensional Laplace operator can then be depicted as

$$\begin{aligned}\Delta u(x,y,z) \approx \frac{1}{h^2}\,(&u(x-h,y,z) + u(x,y-h,z) + u(x,y,z-h)\\ &+ u(x+h,y,z) + u(x,y+h,z) + u(x,y,z+h)\\ &- 6\,u(x))\ .\end{aligned} \tag{3.6}$$

Applying this to Poisson's equation then yields

$$\frac{1}{h^2}\,(u_{i\text{-}1jk} + u_{ij\text{-}1k} + u_{ijk\text{-}1} - 6\,u_{ijk} + u_{i+1jk} + u_{ij+1k} + u_{ijk+1}) = -f_{ijk} \tag{3.7}$$

as its general discrete form and

$$\frac{1}{h^2}\,(\Phi_{i\text{-}1jk} + \Phi_{ij\text{-}1k} + \Phi_{ijk\text{-}1} - 6\,\Phi_{ijk} + \Phi_{i+1jk} + \Phi_{ij+1k} + \Phi_{ijk+1}) = -\frac{\rho_{ijk}}{\varepsilon} \tag{3.8}$$

in particular for the electrostatic case.
A solution is acceptable only if equation (3.7) is satisfied for every grid point (i,j,k) that is not part of the boundary conditions.
A straightforward approach is to merge the above expression for all grid points to a linear system of equations

$$A\,\boldsymbol{u} = \boldsymbol{b}\ , \tag{3.9}$$

which describes the Poisson equation for

$$u_{i'} \quad \Leftrightarrow \quad u_{ijk} \text{ or } \Phi_{ijk} \quad \text{and} \tag{3.10}$$

$$b_{i'} \quad \Leftrightarrow \quad f_{ijk} \text{ or } \frac{\rho_{ijk}}{\varepsilon} \tag{3.11}$$

and where the rows of the (sparse) quadratic matrix A have a form similar to

$$\frac{1}{h^2}\begin{pmatrix} -1 & \dots & -1 & \dots & -1 & 6 & -1 & \dots & -1 & \dots & -1 & \dots & \dots \\ \dots & -1 & \dots & -1 & \dots & -1 & 6 & -1 & \dots & -1 & \dots & -1 & \dots \\ \dots & \dots & -1 & \dots & -1 & \dots & -1 & 6 & -1 & \dots & -1 & \dots & -1 \end{pmatrix}\ ,$$

dependent on how the grid points are assigned to elements of $\boldsymbol{u}$ and $\boldsymbol{b}$. Assuming lexicographic ordering of the grid points in a three-dimensional grid, the matrix consists of exactly seven diagonals that contain non-zero entries. Grid points that don't need to satisfy equation (3.7) (or equation (3.8)) can furthermore be included by modifying

their respective row in the matrix. For example, a fixed value Φ for the electric potential (*Dirichlet boundary condition*) can be accounted for by replacing its row in the matrix with that of the identity matrix (multiplied by $1/h^2$ for consistency) and setting the element in the right-handed side vector to $h^{-2}\,\Phi$.

Methods to solve these kinds of systems, derived from approximating derivatives by finite differences, are consequently called *finite difference methods* (*FDM*).

It's evident that a direct matrix inversion, even though theoretically possible, is not a viable option to obtain the solution $\boldsymbol{u}$, as A^{-1} would generally be dense and impractically large even for relatively small grids.
For example, the inverse matrix for a system of $100 \times 100 \times 100$ grid points usually requires 7.3 TByte of memory (in double precision arithmetic) and it takes on the order of 10^{12} operations for the respective matrix-vector multiplication, that, due to the dense nature of the inverse, can hardly be parallelized.
In fact, even if A^{-1} were to be obtainable at no cost and to be stored in a practicable way, the number of arithmetic operations to calculate $\boldsymbol{u}$ by matrix-vector multiplication ($\mathcal{O}(n^2)$ for n grid points) would still be higher than with many numerical methods.

3.1.2 Conventional Methods to Solve Large Sparse Systems of Linear Equations

The actual methods of solution for this type of problem can be classified into two categories: *direct* and *iterative*.
Direct methods, with Gaussian elimination being a prominent example, are used to determine an exact (up to machine precision) solution in a finite number of arithmetic steps. Efficient implementations often use variations of the fast Fourier transform or the method of cyclic reduction, and require at least $\mathcal{O}(n \log n)$ operations (n being the system size), which is nearly optimal for small systems, but are not generally applicable as they require certain types of boundary conditions or specific system sizes [20]. Furthermore, they are hardly possible to parallelize as they require the whole set of data for computation.

Iterative methods on the other hand gradually improve an approximation of the exact solution starting from an initial guess. However, as the approximation converges to the exact solution, machine precision is generally achievable as well.
The basic iterative methods arise from applying

$$u_{ijk} \coloneqq \frac{1}{6}\left(u_{i\text{-}1jk} + u_{ij\text{-}1k} + u_{ijk\text{-}1} + u_{i+1jk} + u_{ij+1k} + u_{ijk+1} - h^2\, f_{ijk}\right) \;, \tag{3.12}$$

which is just a conversion of equation (3.7), repeatedly to every grid point (boundary conditions barred).
For the *Jacobi iterative method*, the new value for each grid point is calculated using only the values for the respective neighboring grid points that were obtained at the previous iteration, so

$$u^{m+1}_{ijk} \coloneqq \frac{1}{6}\left(u^{m}_{i\text{-}1jk} + u^{m}_{ij\text{-}1k} + u^{m}_{ijk\text{-}1} + u^{m}_{i+1jk} + u^{m}_{ij+1k} + u^{m}_{ijk+1} - h^2\, f_{ijk}\right) \tag{3.13}$$

is used for the $(m+1)^{\text{th}}$ iteration.
The number of iterations (sweeps over the whole grid) required to reach a given accuracy

is dependent on the number of grid points per dimension, i. e., for a three-dimensional cubic grid of size $n = N \times N \times N$, convergence is obtained after $\mathcal{O}(N^2)$ iterations [21]. Since every iteration requires $\mathcal{O}(n)$ operations, the overall computational cost is $\mathcal{O}(n^{1+2/d})$, and accordingly, $\mathcal{O}(n^{5/3})$ in three dimensions.

By using the updated values of neighboring grid points as soon as they are calculated, the *Gauss-Seidel method* is applied. This implicates that the outcome of a single iteration depends on the order in which the grid points are swept through, which enables different variations of the method like *symmetric Gauss-Seidel* (alternate between ascending and descending order) or *red-black Gauss-Seidel* ("color" the grid in a checkerboard pattern, so a "red" point is surrounded by "black" points and vice versa, then first update the red points and use their new values for the black points).
Assuming a sweep to be in lexicographic order, each calculation can be expressed by

$$u_{ijk}^{m+1} \coloneqq \frac{1}{6}\left(u_{i\text{-}1jk}^{m+1} + u_{ij\text{-}1k}^{m+1} + u_{ijk\text{-}1}^{m+1} + u_{i+1jk}^{m} + u_{ij+1k}^{m} + u_{ijk+1}^{m} - h^2\, f_{ijk}\right) . \tag{3.14}$$

Like the Jacobi method, the Gauss-Seidel method converges after $\mathcal{O}(n^{1+2/d})$ operations ($\mathcal{O}(n^{5/3})$ in 3D), but generally twice as fast [21].

Both methods can be expanded by a simple modification. Instead of using the output of equation (3.13) or (3.14) to update a grid point, it is effectively used as an intermediate value $\tilde{u}_{ijk}^{m+1}$. The difference between intermediate and old value, $\Delta u_{ijk}^{m\to\mathrm{m+1}} = \tilde{u}_{ijk}^{m+1} - u_{ijk}^{m}$, is then weighted with a factor ω (relaxation parameter) before it is added onto the old value:

$$u_{ijk}^{m+1} \coloneqq u_{ijk}^{m} + \omega\, \Delta u_{ijk}^{m\to\mathrm{m+1}} \tag{3.15}$$

While the *weighted* or *damped Jacobi method* usually employs $0 < \omega < 1$ to either allow convergence for systems that are diagonally dominant to a lesser extent or to selectively damp certain error modes (see section 3.1.3 for further discussion), the weighted Gauss-Seidel method, primarily referred to as *successive over-relaxation* (*SOR*), greatly improves the convergence rate for $1 < \omega < 2$ [22] (cf. figure 3.1). In fact, by using the optimal relaxation parameter, only $\mathcal{O}(N)$ iterations are needed for convergence, resulting in $\mathcal{O}(n^{1+1/d})$ ($\mathcal{O}(n^{4/3})$ in 3D) overall operations [23].
Analogous to equation (3.14), the updating process for the SOR method can be expressed by

$$\begin{aligned} u_{ijk}^{m+1} \coloneqq (1-\omega)\; u_{ijk}^{m} + \frac{\omega}{6}\Big(&u_{i\text{-}1jk}^{m+1} + u_{ij\text{-}1k}^{m+1} + u_{ijk\text{-}1}^{m+1} + u_{i+1jk}^{m} + u_{ij+1k}^{m} \\ &+ u_{ijk+1}^{m} - h^2\, f_{ijk}\Big) \ . \end{aligned} \tag{3.16}$$

The optimal relaxation parameter for the SOR method can generally be found by systematic trial and error. An analytic expression exists for rectangular grids with homogeneous boundary conditions in one, two, or three dimensions [24]:

$$\omega_{\mathrm{opt}} = \begin{cases} \frac{2}{1+\sin\left(\frac{\pi}{M-1}\right)} & \text{1D; size } n = M \\ \frac{2}{1+\sqrt{1-\frac{1}{4}\left[\cos\left(\frac{\pi}{M-1}\right)+\cos\left(\frac{\pi}{N-1}\right)\right]^2}} & \text{2D; size } n = M \times N \\ \frac{2}{1+\sqrt{1-\frac{1}{9}\left[\cos\left(\frac{\pi}{M-1}\right)+\cos\left(\frac{\pi}{N-1}\right)+\cos\left(\frac{\pi}{L-1}\right)\right]^2}} & \text{3D; size } n = M \times N \times L \end{cases} \tag{3.17}$$

Both the 2D and 3D expression reduce to

$$\omega_{\mathrm{opt}} = \frac{2}{1+\sin\left(\frac{\pi}{M-1}\right)} \tag{3.18}$$

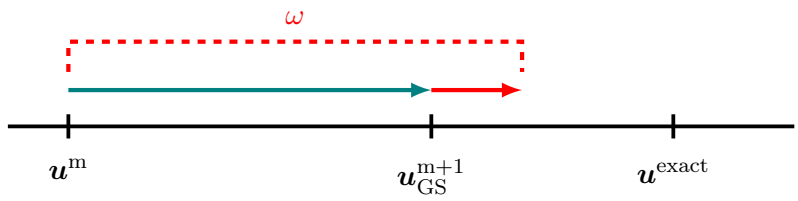

Figure 3.1: Illustration of the effect of over-relaxation. The improvement made by the Gauss-Seidel method is weighted by a factor $\omega > 1$ in order to decrease the difference to the exact solution.

for a square or cubic grid respectively.

3.1.3 Smoothing Properties of Basic Iterative Solvers

By monitoring the development of the solution vector during the iterative process, an important property of the aforementioned solvers can be observed. Provided that the right-handed side of the discretized Poisson equation (equation (3.7)) isn't too oscillatory along successive grid points (i. e., the mesh size is small enough to resolve the curve properly), the exact solution can be expected to be a relatively smooth curve. If the initial guess for the solution vector isn't smooth to begin with, this generally changes after just a few iterations, so that a smooth curve slowly aligns to the exact solution.
Briggs, Henson and McCormick [20] provide a simple example to visualize this:
In order to solve the linear system $A\,\boldsymbol{u} = \boldsymbol{0}$ representing Poisson's equation on a one-dimensional grid of size $n{+}1 = 65$ with boundary conditions $u_0 = u_n = 0$, one or a linear combination of multiple Fourier modes $v_j = \sin\left(\frac{j\,k\,\pi}{n}\right)$ $(0 \leq j \leq n,\ 1 \leq k \leq n\text{-}1)$ is chosen as the initial guess for the approximate solution $\boldsymbol{v}$. The integer j donates the vector component, k is the wavenumber of the respective Fourier mode.
Since the exact solution is trivial for this special case, the error $\boldsymbol{e} = \boldsymbol{u} - \boldsymbol{v}$ is simply $-\boldsymbol{v}$.

By testing the basic iterative methods with various different initial guesses, the following statements of general validity can be made:

1. Initial guesses constructed with low wavenumbers (from this point on referred to as *high-frequency/small-wavelength* or *oscillatory* error modes) converge substantially faster than those constructed with high wavenumbers (*low-frequency* or *smooth* error modes).

2. In linear combinations of Fourier modes with diverse wavenumbers, the high-frequency/small-wavelength modes are "smoothed out" quickly, while errors of longer wavelength remain nearly unchanged.

3. Which wavenumbers can be labeled low-frequency and high-frequency depends on the grid size. For example, a smooth error on 64 grid points can appear oscillatory on 32 grid points.

4. The SOR method's smoothing properties are inferior to the other methods, as it primarily aims to eliminate smooth errors more efficiently by "overshooting" [25].

Figure 3.2(a) shows the development of the maximum error for the model problem for $v_j = \sin\left(\frac{2\,j\,\pi}{n}\right)$, $v_j = \sin\left(\frac{16\,j\,\pi}{n}\right)$, and $v_j = \frac{1}{2}\left[\sin\left(\frac{2\,j\,\pi}{n}\right) + \sin\left(\frac{16\,j\,\pi}{n}\right)\right]$ during the first

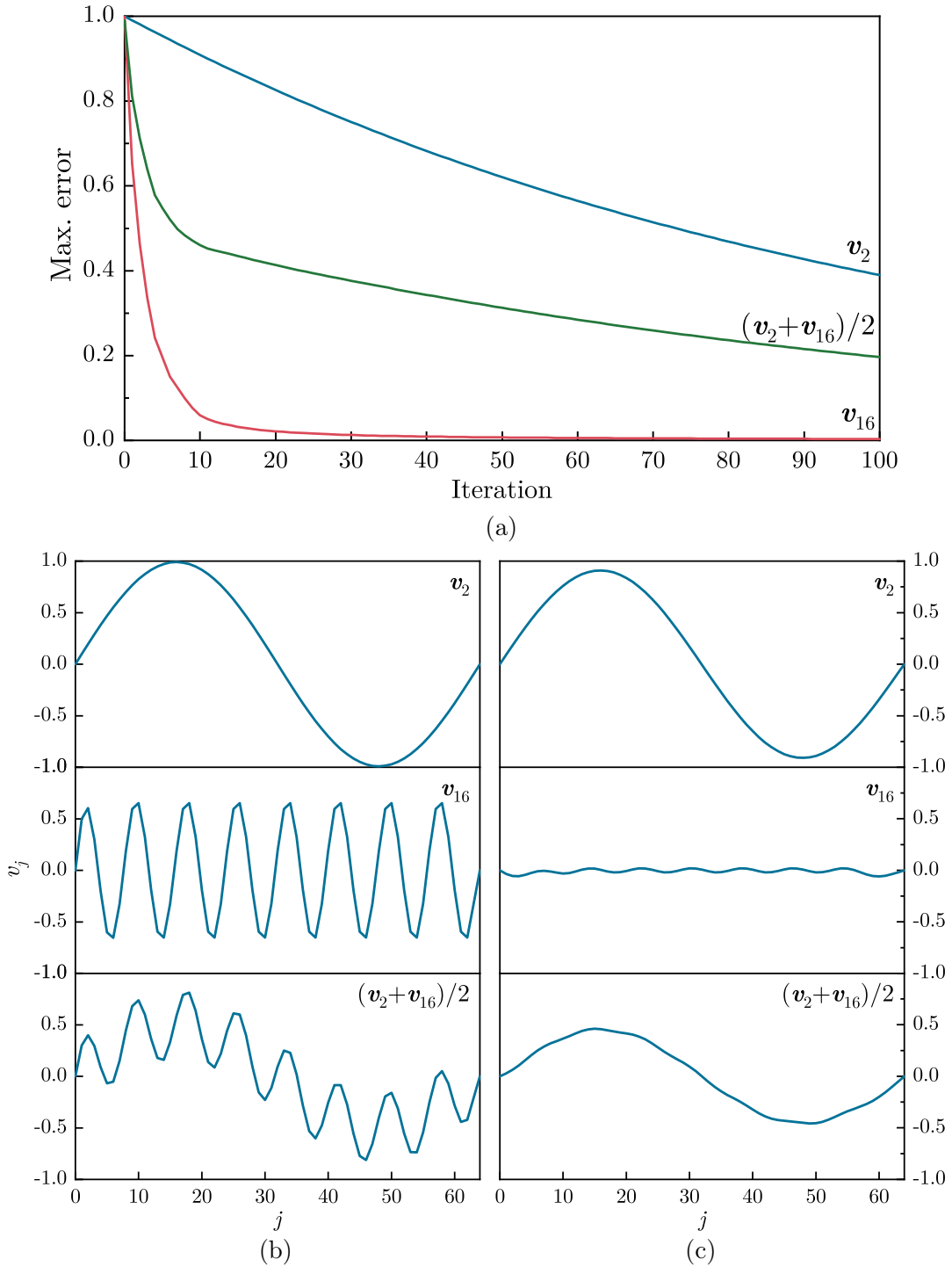

Figure 3.2: The Gauss-Seidel method applied to the model problem: (a) Maximum error of the model problem with initial guesses $\boldsymbol{v}_2$, $\boldsymbol{v}_{16}$, and $(\boldsymbol{v}_2 + \boldsymbol{v}_{16})/2$ plotted against the iteration number. Note that $\boldsymbol{v}_2 + \boldsymbol{v}_{16}$ would asymptotically approach the $\boldsymbol{v}_2$ curve. (b) Approximation to trivial solution after one iteration, plotted against grid point index j. (c) Approximation after ten iterations.
These figures were recreated analogous to figure 2.3 and 2.9 in [20].

100 iterations of the regular Gauss-Seidel method. Figures 3.2(b) and (c) show the actual approximate solution after one and ten iterations.

For every problem on which the basic iterative methods described here can be applied, the error of the approximate solution can be decomposed into high-frequency and low-frequency components. The inability to efficiently eliminate the latter is a major disadvantage, but the resulting smoothing property is a cornerstone to the multigrid methods.

3.2 The Multigrid V-Cycle

A smooth error on a grid Ω^h with mesh size h appears more oscillatory when projected onto a grid Ω^{2h} with doubled mesh size $2h$. Hence the motivation behind the multigrid approach is to utilize the efficient elimination of high-frequency errors by the basic iterative solvers on different length scales. If the problem can be restricted to a coarser grid without losing substantial information, convergence can be accelerated significantly, and furthermore if this concept can be transferred to even coarser grids.
Such a hierarchy of grids with decreasing point density can easily be constructed for structured grids, as the next coarser grid solely consists of the points with even-numbered indices in every direction on the respective finer grid, so that the total number of grid points reduces by a factor of 2 (1D, figure 3.3), 4 (2D), or 8 (3D).

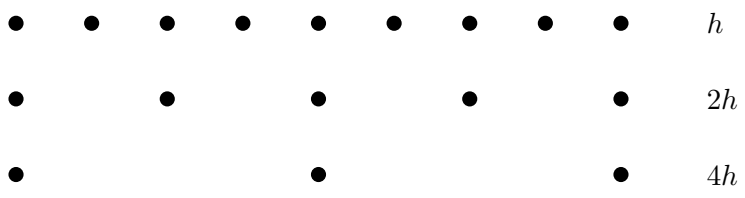

Figure 3.3: One-dimensional grid hierarchy

Coarse-grid acceleration had been proposed even before the first digital computers were developed (1935 [26]) and was first properly described by R. P. Fedorenko in 1961 [27] as an actual multigrid method.
Further progress on the subject is mainly attributed to A. Brandt [28], who then pioneered the development of advanced methods for solving PDEs and other problems with large numbers of unknowns.

One way to achieve the multigrid acceleration described above is the so-called *correction scheme* procedure, for which the problem $A\,\boldsymbol{u} = \boldsymbol{b}$ with initial guess/approximate solution $\boldsymbol{v}$ is reformulated using the residual

$$\boldsymbol{r} = \boldsymbol{b} - A\,\boldsymbol{v} \tag{3.19}$$

and the error

$$\boldsymbol{e} = \boldsymbol{u} - \boldsymbol{v}\ . \tag{3.20}$$

For an iterative process, $\boldsymbol{r}$ approaches $\boldsymbol{0}$ as $\boldsymbol{v}$ approaches the exact solution $\boldsymbol{u}$ and is (unlike $\boldsymbol{e}$ and $\boldsymbol{u}$) easily computable.
A straightforward derivation then gives the *residual equation*

$$A\,\boldsymbol{e} = \boldsymbol{r}\ , \tag{3.21}$$

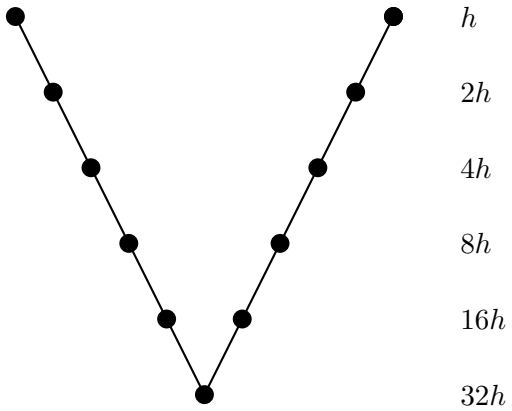

Figure 3.4: Illustration of the V-cycle. The pattern is traversed from left to right; the nodes in the left arm denote pre-smoothing, those in the right arm post-smoothing. The tip of the "V" marks the solution process on the coarsest grid (32 times the finest mesh size in this case). The line sections denote restriction on the left and prolongation on the right arm.

where only $\boldsymbol{e}$ is unknown. The solution of this linear system of equations coincidentally solves the original problem via equation (3.20).
For the correction scheme, the following strategy is used: First, the high-frequency errors are eliminated by applying one of the basic iterative solvers (*smoothing/relaxation*). The updated residual is then projected onto a coarser grid (*restriction*), where it is used as the right-handed side of a new linear system of equations $A^{2h}\,\boldsymbol{u}^{2h} = \boldsymbol{b}^{2h}$. Logically, the consistent initial guess is the zero vector.
By proceeding recursively, error modes of increasing wavelength can be suppressed until a reasonably small system is reached that can be solved to no noteworthy costs.
The so-obtained error is thereon interpolated back to the respective finer grid (*interpolation/prolongation*) where it is added onto the current approximate solution as a correction. Additional smoothing can now be used to compensate any new errors arising from restriction and prolongation.

Since the whole process starts at a fine grid and migrates along a hierarchy of coarser grids, it can be represented by a simple diagram showing the schedule in which the different grid levels are visited. Because the pattern resembles the letter "V" (figure 3.4), this basic routine is called *V-cycle*. The number of iterations for pre- and post-smoothing usually doesn't surpass three.

The correction scheme is commonly applied if the PDE that is supposed to be solved is discretized to a structured grid. Multigrid methods capable of handling unstructured grids are briefly discussed in section 3.5.

The affiliated algorithm and the individual components of the multigrid V-cycle are further elaborated in the following.

3.2.1 Algorithm

The correction scheme V-cycle can be put in concrete terms by using pseudo-code, as shown in algorithm 1.

Algorithm 1 Correction Scheme V-Cycle

```
1: function V-Cycle(ℓ, v^h, b^h)
2:     if ℓ = ℓ_max then
3:         v^h := Solve(v^h, b^h)
4:     else
5:         for i := 1, n_pre do
6:             v^h := Smooth(v^h, b^h)
7:         end for
8:         r^h := b^h − A^h v^h
9:         b^{2h} := Restrict(r^h)
10:        v^{2h} := 0
11:        v^{2h} := V-Cycle(ℓ + 1, v^{2h}, b^{2h})
12:        v^h := v^h + Prolongate(v^{2h})
13:        for i := 1, n_post do
14:            v^h := Smooth(v^h, b^h)
15:        end for
16:    end if
17:    return v^h
18: end function
```

Here, the finest grid is identified by the integer ℓ being zero.
The functions Restrict($\boldsymbol{r}^h$) and Prolongate($\boldsymbol{v}^{2h}$) could basically be described by operators I_h^{2h} and I_{2h}^h that transform vectors into their coarse-grid and fine-grid counterpart.

If any smooth error components remain after a V-cycle, it can be applied again, i. e., as an iterative solver.

More details on the respective subroutines, especially concerning three-dimensional structured grids, are given in the following.

Smoothing

For smoothing, a suitable iterative solver (usually one of those described in subsection 3.1.2) is applied for a small number of iterations n_{pre} or n_{post} respectively. The sufficient number of sweeps depends on the actual problem and is subject to optimization, but n_{pre} is generally chosen to be not smaller than n_{post}.

Calculation of Residuals

In terms of the expressions used in subsection 3.1.2, equation (3.19) can be rewritten as

$$r_{ijk} \coloneqq \frac{1}{h_\ell^2} \left(6\, u_{ijk} - u_{i\text{-}1jk} - u_{ij\text{-}1k} - u_{ijk\text{-}1} - u_{i+1jk} - u_{ij+1k} - u_{ijk+1}\right) + f_{ijk} \tag{3.22}$$

for the 3D Poisson equation. Since all values that are used were previously calculated, the order in which the grid points are swept through is of no further importance.

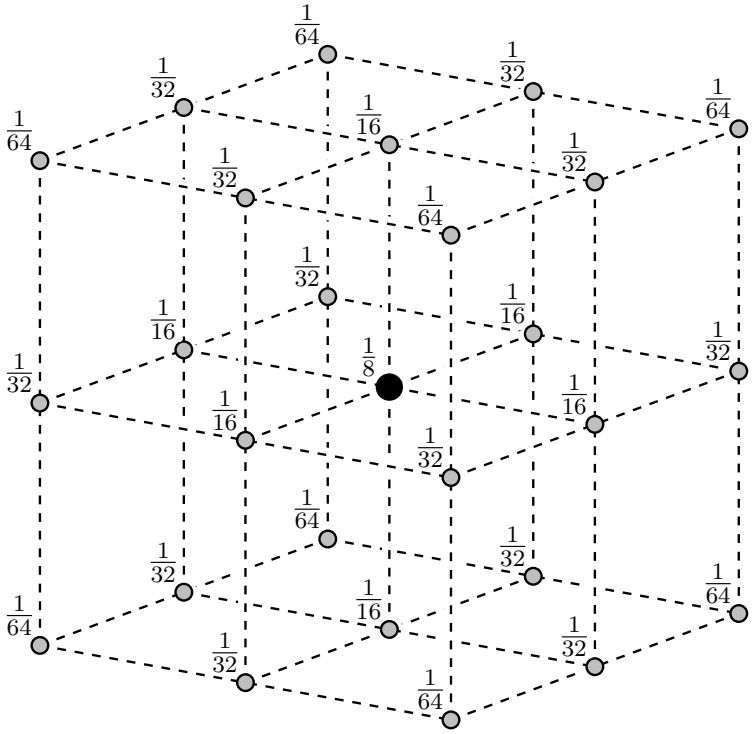

Figure 3.5: Three-dimensional restriction with 27-point full weighting. The black node marks the only grid point matching the coarse grid. The weighting factor is given for each node.

Restriction

Because the coarsening strategy to generate the hierarchy of grids generally involves doubling the mesh size, each point on a coarse grid has a geometrically corresponding point on the next finer grid (cf. fig 3.3).

The simplest method to project a vector $\boldsymbol{v}^h$ onto a vector $\boldsymbol{v}^{2h}$ is therefore a direct *injection* of these fine-grid vector components into the respective coarse-grid vector components:

$$v_j^{2h} \coloneqq v_{2j}^h \text{ (1D)}. \tag{3.23}$$

However, a more robust alternative is *full weighting*,

$$v_j^{2h} \coloneqq \frac{1}{4}\left(v_{2j\text{-}1}^h + 2\,v_{2j}^h + v_{2j+1}^h\right) \text{ (1D)}, \tag{3.24}$$

for which the surrounding grid points are considered as well via algebraic weighting.

For higher dimensions, not only the nearest neighboring points are considered for full weighting, but also those that define the smallest square or cube around the coarse grid point. The weighting factors are assigned depending on position, i. e., the central point has the maximum weighting factor, the four (2D) or six (3D) nearest neighbors half of that and the next nearest neighbors a quarter. For the three-dimensional case, this is visualized in figure 3.5. Calculating a coarse-grid value in 3D therefore involves 27 fine-grid values with weighting factors $\frac{1}{8}$ (1×), $\frac{1}{16}$ (6×), $\frac{1}{32}$ (12×), and $\frac{1}{64}$ (8×).
This approach has the remarkable quality that every fine-grid point is weighted to

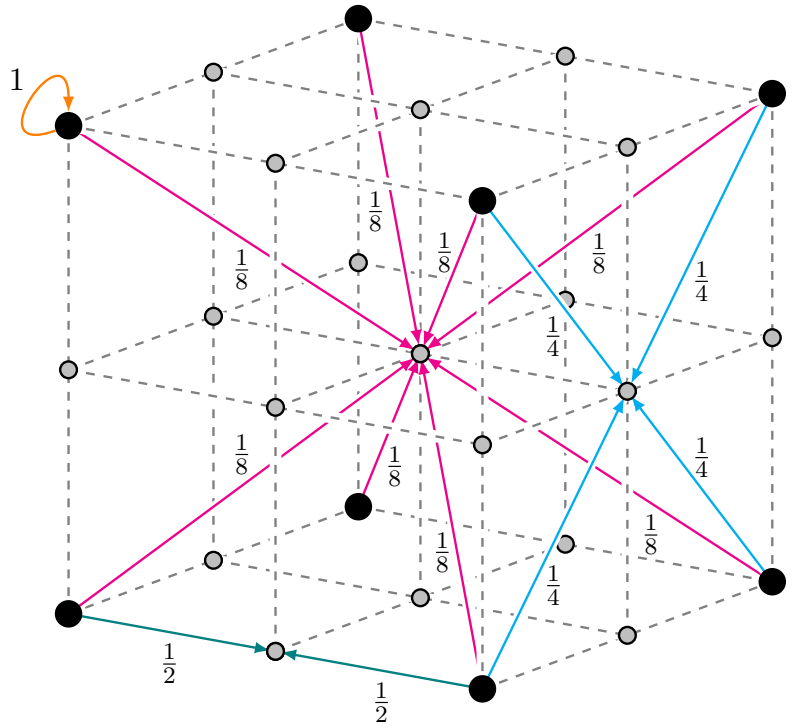

Figure 3.6: Linear, bilinear and trilinear interpolation on a three-dimensional grid. Note that the shown grid section is shifted compared to figure 3.5.

the same extent, because every point without corresponding coarse grid counterpart is weighted multiple times.
Full weighting can be attenuated to *half weighting* by ignoring the furthermost neighbors in order to trade accuracy for speed.

Prolongation

In order to apply the coarse-grid correction, linear interpolation is generally used, as it is quite effective and normally produces a sufficiently smooth function.
Since the PDE is discretized onto a structured grid, this simply implies calculating the mean value of the geometrically nearest coarse grid points:

$$\begin{aligned} v_{2j}^{h} &:= v_{j}^{2h} \\ v_{2j+1}^{h} &:= \frac{1}{2}\left(v_{j}^{2h} + v_{j+1}^{2h}\right) \quad . \end{aligned} \qquad \text{(1D)} \tag{3.25}$$

In two and three dimensions, this expands to bilinear and trilinear interpolation (cf. figure 3.6).

Solving

Since the coarsest grid in the hierarchy can be chosen to be almost arbitrarily small, the algorithm's runtime complexity is of minor importance. Instead, methods to solve linear systems have to be evaluated for their performance regarding small systems.

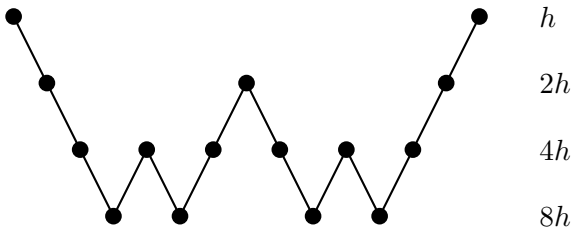

Figure 3.7: Illustration of the W-cycle ($\mu = 2$), analogous to figure 3.4.

Literature on the subject therefore often encourages a direct solver [19, 20, 29]. However, as the coarseness of the grid also increases the truncation error, which adds to the errors introduced via restriction and prolongation, the coarse grid solution actually doesn't need to be exact, but appropriately accurate. Hence, iterative solvers, if efficient, are a viable option, also because they are comparably easy to implement.

3.2.2 Derivatives

The basic V-cycle can be modified for various benefits, not only limited to the number of pre- and post-smoothing steps. To improve the convergence behavior of one iteration, the recursive self-calling of the V-cycle routine can be extended to form the so-called μ-cycle. On the other hand, if no initial guess is available, it can be beneficial to start the solution process on the coarsest grid level.

μ-Cycle

If the number of smoothing steps necessary to eliminate certain error modes surpasses a limit or another V-cycle would be needed to achieve a given convergence threshold, it can be faster to use the self-calling of the V-cycle function on coarser grid levels another time. By replacing line 11 in the pseudo-code algorithm above (algorithm 1) with

for $i \coloneqq 1$, μ **do**
  $\boldsymbol{v}^{2h} \coloneqq \mu$-CYCLE($\mu$, $\ell + 1$, $\boldsymbol{v}^{2h}$, $\boldsymbol{b}^{2h}$)
end for

and renaming the function to μ-CYCLE(μ, ℓ, $\boldsymbol{v}^h$, $\boldsymbol{b}^h$), the *μ-cycle* is generated. It shifts the computational load towards the coarser grids by recursively calling itself μ times and greatly improves the convergence rate of a single iteration, although not automatically run-time till convergence.
As can be deduced from figure 3.7, the number of times the coarsest grid is visited doubles with every additional grid level, practically limiting the μ-cycle to $\mu \leq 2$, which leaves the V- and the W-cycle (named analogously).

However, it's noteworthy that a value for μ can be assigned to each grid level individually, allowing further optimization.

Nested Iteration

In absence of an eligible initial guess (including the zero vector), it's advisable to start the solution process on the coarsest grid. A simple, yet often sufficient, approach is to

Algorithm 2 Full Multigrid V-Cycle

```
1: function FMG(ν, ℓ, v^h, b^h)
2:     if ℓ = ℓ_max then
3:         v^h := 0
4:         v^h := SOLVE(v^h, b^h)
5:     else
6:         b^{2h} := RESTRICT(b^h)
7:         v^{2h} := FMG(ν, ℓ + 1, v^{2h}, b^{2h})
8:         v^h := PROLONGATE(v^{2h})
9:         for i := 1, ν do
10:            v^h := V-CYCLE(ℓ, v^h, b^h)
11:        end for
12:    end if
13:    return v^h
14: end function
```

solve the problem on the coarsest grid and to prolongate the solution to the increasingly finer levels, on each of which only some additional smoothing is performed.
This implicates that in this case, a coarse grid solution is not a correction for the next finer grid, but is rather a rough approximation to the actual solution, being of the same order of magnitude of a physical quantity as the exact solution.

Full Multigrid

By combining nested iteration and the V-cycle, a powerful algorithm capable of solving many problems in a single run arises.
The simple smoothing described above is replaced by a whole V-cycle (or a multitude of) that starts on the current grid level, efficiently providing an accurate solution to the problem on the corresponding degree of discretization (figure 3.8).

With the V-cycle being established, the algorithm is rather simple (algorithm 2). It includes the restriction of the fine-grid right-handed side vector to the coarser grids, hence the initial call should be FMG(ν, $\ell_{\max}$, $\boldsymbol{v}^h$, $\boldsymbol{b}^h$).
Here again, the parameter ν that indicates the number of V-cycle iterations on every level can be assigned individually and is subject to optimization.

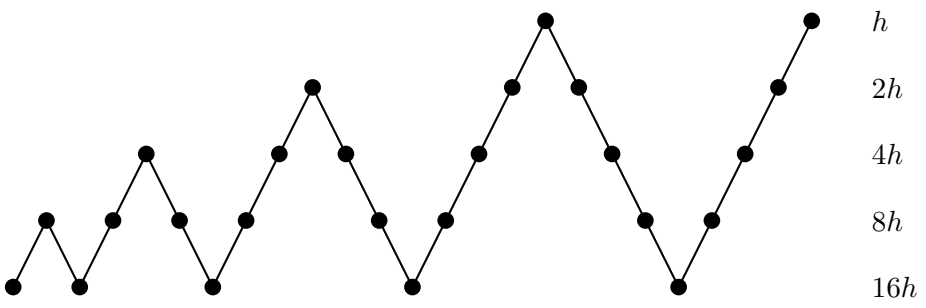

Figure 3.8: Full multigrid (FMG) scheme with $\nu = 1$

3.3 Boundary Conditions

Up to this point, a grid point was considered to be surrounded by and its numeric value to be dependent on other grid points. Since computations can only be performed on a finite grid, some grid points consequentially define the sides of the (box-shaped) domain and need to be treated differently. Their finite-difference equations, deviating from equation (3.7), must correspond to the boundary conditions needed for a unique solution to a physical differential equation (boundary value problem). Most commonly, either *Dirichlet* or *Neumann boundary conditions* are used.
For Dirichlet boundary conditions, the solution of the differential equation on the domain Ω, u, has to comply with a given function g on the boundaries $\partial\Omega$:

$$u(\boldsymbol{r}) = g(\boldsymbol{r}) \quad \forall\, \boldsymbol{r} \in \partial\Omega. \tag{3.26}$$

Neumann boundary conditions on the other hand only define the normal derivative of u on $\partial\Omega$:

$$\frac{\partial u(\boldsymbol{r})}{\partial \boldsymbol{n}} = g(\boldsymbol{r}) \quad \forall\, \boldsymbol{r} \in \partial\Omega. \tag{3.27}$$

It is further possible to enforce both types simultaneously (*Cauchy boundary conditions*) or as a linear combination of the form

$$c_1\, u(\boldsymbol{r}) + c_2\, \frac{\partial u(\boldsymbol{r})}{\partial \boldsymbol{n}} = g(\boldsymbol{r}) \quad \forall\, \boldsymbol{r} \in \partial\Omega \tag{3.28}$$

with constants c_1 and c_2 (*Robin boundary conditions*). However, since there is generally no physical reason to use the latter two with Poisson's equation, they are ignored in the further course of this work.
Additionally, it must be pointed out that neither type can be used to straightforwardly model boundary conditions at infinity, e. g.,

$$u(\boldsymbol{r}) = 0 \;\text{ for }\; \boldsymbol{r} \to \infty\,. \tag{3.29}$$

Overcoming this limitation without introducing significant inaccuracies is often a critical part of the numerical solution process.

While the finite-difference equation for Dirichlet boundary points is exceptionally simple,

$$u_{ijk} = g_{ijk} \tag{3.30}$$

with g_{ijk} often being a constant, the counterpart of equation (3.7) for Neumann boundary conditions depends on the side of the domain on which the respective point is located. For a point on the side in negative x direction (with index $i = 0$), this is

$$u_{0j\text{-}1k} + u_{0jk\text{-}1} - 6\, u_{0jk} + 2u_{1jk} + u_{0j+1k} + u_{0jk+1} = -h^2\, f_{0jk} + 2h\, g_{0jk}\,. \tag{3.31}$$

Here, the central difference

$$\frac{\partial u(i\,h, j\,h, k\,h)}{\partial x} = \frac{u_{i+1jk} - u_{i\text{-}1jk}}{2h} + \mathcal{O}(h) \tag{3.32}$$

was used.

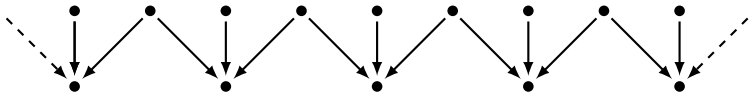

Figure 3.9: One-dimensional restriction with homogeneous Neumann boundary conditions. The outermost points on the coarser grid "see" the next fine grid point in inward direction mirrored to outside the domain.

These considerations can be adapted to all non-transfer grid operations of the multigrid V-cycle. For homogeneous Neumann boundary conditions ($\frac{\partial u(\boldsymbol{r})}{\partial \boldsymbol{n}} = 0$), the corresponding full weighting restriction operation is given by

$$v_0^{2h} \coloneqq \frac{1}{4}\left(2\,v_0^h + 2\,v_1^h\right) \text{ (1D)}, \tag{3.33}$$

where the weighting factor of v_1^h is doubled compared to equation (3.24) and the (non-existant) point at $i = -1$ is left out [20]. A simple visualization is given in figure 3.9.

In order to avoid ambiguities, the boundary conditions discussed here are referred to as *domain boundary conditions* in the further course of this thesis, in contrast to *irregular boundary conditions* imposed by points of fixed value distributed over the whole grid (additional Dirichlet boundary conditions).

3.4 Variable Coefficients

In its general form, the electrostatic Poisson equation allows for various different dielectric media to be present by letting the relative permittivity ε_r be dependent on $\boldsymbol{r}$:

$$\boldsymbol{\nabla}\left[\varepsilon_r(\boldsymbol{r})\boldsymbol{\nabla}\Phi(\boldsymbol{r})\right] = -\frac{\rho(\boldsymbol{r})}{\varepsilon_0}\ . \tag{3.34}$$

The one-dimensional discrete system resulting from this,

$$\frac{\varepsilon_{i+\frac{1}{2}}\frac{\Phi_{i+1}-\Phi_i}{h} - \varepsilon_{i-\frac{1}{2}}\frac{\Phi_i-\Phi_{i-1}}{h}}{h} = -\frac{\rho_i}{\varepsilon_0} \tag{3.35}$$

$$\varepsilon_{i+\frac{1}{2}}\,\Phi_{i+1} - \left(\varepsilon_{i-\frac{1}{2}} + \varepsilon_{i+\frac{1}{2}}\right)\Phi_i + \varepsilon_{i-\frac{1}{2}}\,\Phi_{i-1} = -h^2\,\frac{\rho_i}{\varepsilon_0}\ , \tag{3.36}$$

requires the relative permittivity to be defined between the grid points, i. e., at position $i - \frac{1}{2}$ and $i + \frac{1}{2}$. It's therefore common practice to define a variable coefficient such as ε_r as a cell-centered quantity. In two and three dimensions, however, this still requires additional averaging. For example, the value $\varepsilon_{i+\frac{1}{2}\,j\,k}$ needed in the three-dimensional case can be calculated via

$$\varepsilon_{i+1\,j\,k} = \frac{1}{4}\left(\varepsilon_{i+\frac{1}{2}\,j-\frac{1}{2}\,k-\frac{1}{2}} + \varepsilon_{i+\frac{1}{2}\,j-\frac{1}{2}\,k+\frac{1}{2}} + \varepsilon_{i+\frac{1}{2}\,j+\frac{1}{2}\,k-\frac{1}{2}} + \varepsilon_{i+\frac{1}{2}\,j+\frac{1}{2}\,k+\frac{1}{2}}\right)\ , \tag{3.37}$$

where the available cell-centered values are used.
The general variable coefficient problem $\boldsymbol{\nabla}\left[a(\boldsymbol{r})\boldsymbol{\nabla}u(\boldsymbol{r})\right] = -f(\boldsymbol{r})$ then consequently has the discrete form

$$\begin{aligned}\frac{1}{h^2}\,&(c_1\,u_{i-1\,j\,k} + c_2\,u_{i\,j-1\,k} + c_3\,u_{i\,j\,k-1}\\ &+c_4\,u_{i+1\,j\,k} + c_5\,u_{i\,j+1\,k} + c_6\,u_{i\,j\,k+1} - c_7\,u_{i\,j\,k}) = -f_{i\,j\,k}\end{aligned} \tag{3.38}$$

with precalculatable coefficients $c_1, \ldots, c_7$, equivalent to equation (3.7).

A simple approach to extend a multigrid method's capabilities to the solution of variable coefficient differential equations is to average the fine-grid coefficients of cells that match the same coarse-grid cell.
However, compared to a constant coefficient problem, the convergence rate can generally be expected to be inferior, because the numerical effects of varying coefficients are the same as those encountered with non-uniform grids, where smoothing is not equally effective over the whole grid and the interpolation routines need to be adapted [20].

3.5 Alternative Multigrid Variants

The multigrid correction scheme used up to this point is limited to linear problems on structured grids. Since the idea to use a multilevel hierarchy to solve a problem is too elemental, other variants were developed to widen the range of application.

The *full approximation storage scheme* (*FAS*) introduced by A. Brandt in 1977 [28] is similar to the correction scheme in that it uses the same basic operators and is applicable to μ-cycle and FMG as well.
One FAS V-cycle requires more arithmetic operations than with the correction scheme, because the coarse system is solved for the full approximation $\boldsymbol{u}^{2h} = I_h^{2h}\boldsymbol{v}^h + \boldsymbol{e}^{2h}$ instead of only the error $\boldsymbol{e}^{2h}$ (algorithm 3).

This enables the treatment of problems for which the residual equation (3.21) has no solution, i. e., nonlinear problems, with A being a nonlinear operator.
Another positive feature is the fact that every grid holds an actual approximation to the fine grid solution, which on the one hand allows visualization of the multigrid progress

Algorithm 3 FAS V-Cycle

```
1: function FAS_V-Cycle(ℓ, v^h, b^h)
2:     if ℓ = ℓ_max then
3:         v^h := Solve(v^h, b^h)
4:     else
5:         for i := 1, n_pre do
6:             v^h := Smooth(v^h, b^h)
7:         end for
8:         v^{2h} := Restrict(v^h)
9:         r^h := b^h − A^h v^h
10:        b^{2h} := A^{2h} v^{2h} + Restrict(r^h)
11:        v^{2h} := FAS_V-Cycle(ℓ + 1, v^{2h}, b^{2h})
12:        e^{2h} := v^{2h} − Restrict(v^h)
13:        v^h := v^h + Prolongate(e^{2h})
14:        for i := 1, n_post do
15:            v^h := Smooth(v^h, b^h)
16:        end for
17:    end if
18:    return v^h
19: end function
```

throughout a single iteration (cf. figure 3.10; not possible when using the correction scheme) and on the other hand is a valuable prerequisite for an adaptive grid approach. Problems that require a high level of discretization only on a limited subset of a given domain (e. g., because the solution varies over multiple magnitudes) can benefit from a global coarse grid that is only refined on local patches where a high resolution is needed. This can be implemented using the FAS method and appropriate strategies for transferring data at fine-grid edges.
A further logical step is the development of *adaptive mesh refinement* methods, for which the solution process is started on a very coarse grid that is subsequently refined where an evaluation of the current solution indicates inflated errors [30].

The inapplicability to linear systems based on highly unstructured grids or systems without any underlying geometric grid led to the development of *algebraic multigrid* (*AMG*) [31, 32, 33].
This method uses multigrid principles, but is independent of the problem geometry, as it produces coarse-grid equations and inter-grid transfer operators directly from the system matrix A. Hence the underlying problem doesn't have to be of geometric nature at all.
Assuming conventional, classical, AMG, the following coarsening strategy is used: For each line i of A, the non-zero values a_{ij}, $i \neq j$, are grouped into strongly and weakly influencing the solution u_i, dependent on how the absolute value compares to a_{ii}. Coarse-grid points (*C-points*) are then selected based on the premise that each fine-grid point (*F-point*) that is not a C-point should only be strongly influenced by C-points or other F-points that are in turn strongly influenced by at least one C-point themselves. By avoiding C-points being strongly dependent on other C-points, this produces a subset of the fine grid suitable to be used for further computation.
The interpolation of the error $\boldsymbol{e}$ from C- to F-points is performed using the interpolation operator I_{2h}^{h}:

$$\left(I_{2h}^{h}\boldsymbol{e}\right)_i = \begin{cases} e_i & \text{, if } i \text{ } C\text{-point.} \\ \sum_{j \in C_i} \omega_{ij}\, e_j & \text{, if } i \text{ } F\text{-point.} \end{cases} \tag{3.39}$$

Here, C_i is the set of C-points strongly influencing i, called *coarse interpolatory set*. The interpolation weights ω_{ij} are calculated from both the a_{ij} that indicate strong influence and weak influence.
The restriction operator is then

$$I_{h}^{2h} = \left(I_{2h}^{h}\right)^{T} \tag{3.40}$$

and the coarse-grid matrix is given by

$$A^{2h} = I_{h}^{2h} A^{h} I_{2h}^{h} \,, \tag{3.41}$$

the so-called *Galerkin condition* [20]. A correction scheme V-cycle can then be applied analogously.
Algebraic multigrid is generally the most flexible multigrid method and can, once established, be used as a black-box solver for linear systems represented by certain classes of sparse matrices. However, the classical, geometric, multigrid methods can be implemented without explicitly storing the system matrix and require less arithmetic operations per V-cycle.

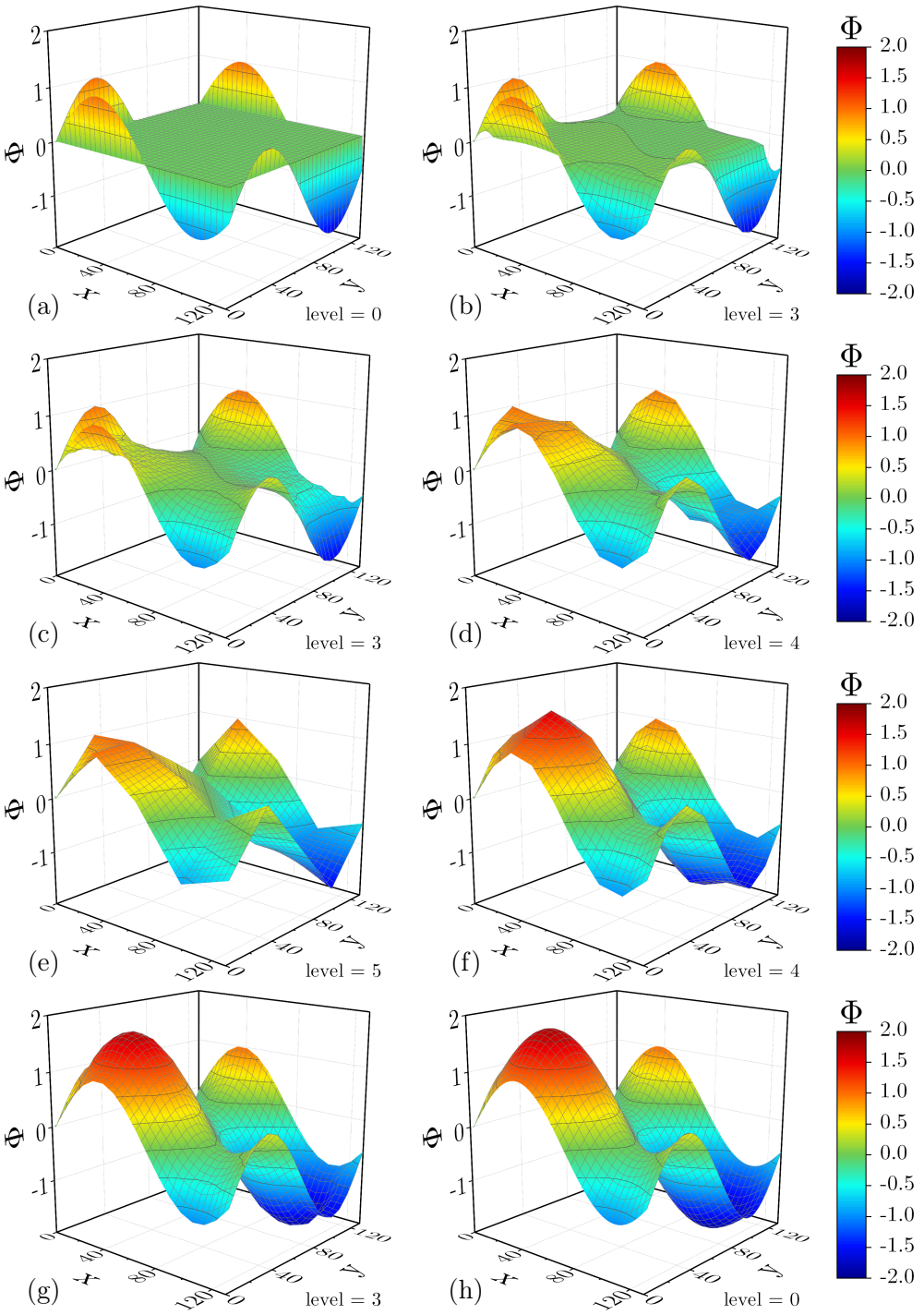

Figure 3.10: 2D model problem for the FAS V-cycle: The Poisson equation with Dirichlet boundary conditions is solved on a square with side length a (discretized to 129×129 grid points) for the potential $\Phi(x, y) = \sin\left(\frac{9}{5}\pi\,\frac{x}{a}\right) + \sin\left(2\pi\,\frac{y}{a}\right)$. Starting with a zero guess (a), notable changes emerge only after a couple restriction steps, when smoothing affects a wider range. On grid level 3 (17×17 grid points), the current approximation before (b) and after (c) smoothing are clearly distinguishable and further progress to grid level 4 (d) improves it even more. The exact solution (e) on level 5 (5×5 grid points) is then passed on to the finer grids, where the approximation is smoothed out (f, g, h).

3.6 Complexity

Following Briggs et al. [20], the computational cost to calculate new values for the finest grid (one smoothing sweep) is assumed to be one *work unit* (WU). The total work done on the grid incorporates pre- and post-smoothing ($(n_{\text{pre}}+n_{\text{post}})$ WU) and restriction and interpolation (both amounting a relatively fixed fraction χ of one WU) and is therefore proportional to a work unit. Accordingly, the work done on the first coarsened grid is more or less exactly $1/2^d$ times the work done on the finest grid, because the number of grid points is reduced by this fraction.
To simplify the following considerations, n_{pre} and n_{post} are set to 1 and χ is neglected. The total computation cost for a V-cycle on ℓ grids is then $2\,(1 + 2^{-d} + 2^{-2d} + \cdots + 2^{-(l-1)d})$ WU, because the coarsest grid is chosen to be small enough for the solution process to be negligible in this consideration. No matter how large the finest grid is and how many grid levels are therefore used, this series doesn't exceed an upper bound given by the geometric series:

$$\sum_{i=0}^{\infty} 2^{-i\,d} = \frac{1}{1-2^{-d}} = \begin{cases} 2 & \text{for } d = 1. \\ \frac{4}{3} & \text{for } d = 2. \\ \frac{8}{7} & \text{for } d = 3. \end{cases} \tag{3.42}$$

The computation cost of a V-cycle is therefore a constant multiplicative of a work unit and scales linearly with the system size n.
Similarly, a simple full multigrid cycle with $\nu = 1$ takes only up to $2\,(1-2^{-d})^{-1}\,(1 + 2^{-d} + 2^{-2d} + \cdots + 2^{-(l-1)d})$ WU $< 2\,(1-2^{-d})^{-2}$ WU for completion, since a V-cycle started on a coarse grid costs 2^{-d} of a V-cycle on the next finer grid, and hence the same $\mathcal{O}(n)$ scaling behavior is achieved for a cycle.

However, one cycle is not generally sufficient for a converged solution, so further considerations are needed.
Convergence theory shows that an appropriately set up V-cycle reduces the error by a convergence factor bound $\gamma < 1$, that is independent of the mesh size h [20]. To reduce the error of a zero initial guess ($\mathcal{O}(1)$) on a grid of size $n = N^d$ to the order of the discretization error ($\mathcal{O}(h^2) = \mathcal{O}(N^{-2})$),

$$\nu = \mathcal{O}(\log N) \tag{3.43}$$

iterations are needed, since

$$\gamma^{\nu} = \mathcal{O}(N^2) \tag{3.44}$$

needs to be satisfied.
Using the V-cycle iteratively therefore results in computation costs of $\mathcal{O}(n \log N)$ arithmetic operations to reach convergence.

FMG on the other hand has the potential to efficiently reduce every error component throughout the progression from the coarsest to the finest grid. It can be shown that for sufficiently small γ, only $\mathcal{O}(1)$ iterations of the last V-cycles (meaning those on the finest grid) are needed, which results in $\mathcal{O}(n)$ costs [20].
In other words, the initial guess FMG provides for the finest grid enables optimal behavior of the algorithm.

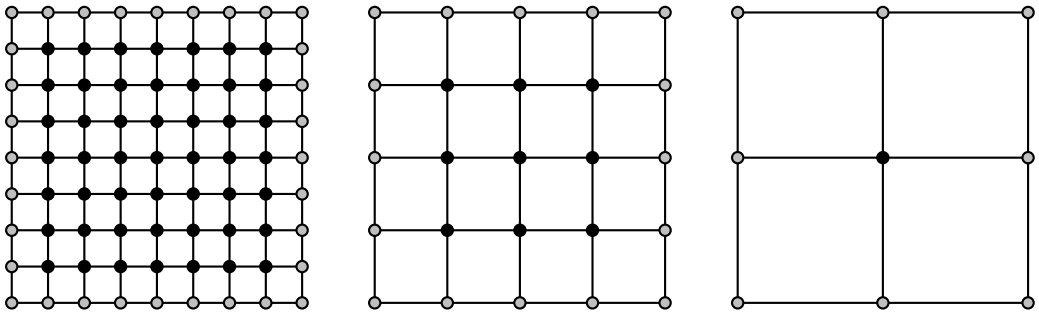

Figure 3.11: Coarsening of a two-dimensional square sub-grid. The ratio of inner to outer grid points decreases.

3.7 Parallelizability

For solving partial differential equations on spatially discretized domains, the *domain partitioning* method is the natural choice for parallelization, especially for iterative solvers. Assuming MPI utilization, the grid is partitioned into subsets of preferably equal size, each of which is assigned to an individual MPI process. Every process can then perform calculations on its local grid, but needs to communicate with the processes holding the neighboring subdomains to exchange boundary data. Since MPI allows non-blocking communication, for a single iteration, a properly adjusted solver first calculates the values of the (inner) grid points that don't depend on data held by other processes, while concurrently exchanging information with its neighbors, before the outer grid points can be swept through. Nonetheless, communication is a major performance bottleneck, since further parallelization of a given problem only increases the amount of boundary data that needs to be transferred. It's therefore necessary to minimize the surface area of the individual subdomains to maximize the ratio of computation and communication, i e., to preferably use square (2D) or cubic (3D) subdomains for their surface-to-volume ratio.
For the optimal case, the time needed to send and receive the boundary values equals the time needed to calculate new values for the inner grid points. This, of course, depends on the specific hardware, i. e., the network connection (latency and bandwidth), the CPU (computational power and cache size), and the random access memory (latency and bandwidth).

For the multigrid methods, this approach is still valid, but leads to some drawbacks. As the coarsening of the grids proceeds along the grid hierarchy, the ratio of inner to outer grid points decreases (cf. figure 3.11) and thus communication time increasingly dominates overall runtime, with latency becoming relevant. In practice, high performance requirements arise for the network connection, and perfect parallelization, where a program running on N_p cores is N_p times faster than when running on one core, is hardly ever achievable.
Nevertheless, it is a method without any considerable alternatives and can be optimized greatly [34].

However, multigrid variants that shift the computational load towards the coarser grids, i. e., the W-cycle, become less desirable and are only applied if there's a very clear advantage over multiple V-cycles in the serial case.

3.7.1 Order-Independent Smoothing

For obvious reasons, the domain partitioning method doesn't fit well with lexicographically ordered Gauss-Seidel smoothing. Other methods, like weighted Jacobi or red-black Gauss-Seidel are better suited, since the former needs updated boundary values only once at the beginning of an iteration and the latter enables processing the boundary values at an arbitrary point throughout a sweep.
One iteration of the parallel red-black Gauss-Seidel method involves updating the inner red grid points, while the black boundary values are exchanged with the neighboring processes. This enables calculating the new values for the outer red points, which then can be exchanged while the inner black points are updated. Afterwards, the update of the outer black grid points can proceed, using the previously received red boundary values.
By this, the total amount of transferred data is the same as with the Jacobi method, although an additional communication step is used.

3.7.2 U-Cycle vs. Coarse Grid Agglomeration

By coarsening the grids as much as possible, as it's preferable with serial computation, parallel multigrid solvers can experience a huge performance cut because communication eventually cannot be hidden completely, so the calculation process is interrupted by waiting times. This can actually deteriorate the parallel performance so much that additional cores only worsen the performance, or even that the program runs faster on a single core (without any communication) than in a parallel environment.
This is particularly the case if the number of processes approaches or even surpasses the total number of grid points, which is within easy reach considering massive parallelization and very coarse grids.

There are two strategies to overcome the issue: First, the coarsest grid can be chosen to be significantly finer than usual to restrict calculations to the few grid levels where parallelization is effective. Since this approach effectively cuts off the tip of the "V" in graphical representations like figure 3.4, it is called *U-cycle*. The coarse-grid solution process then takes place on a grid of non-negligible size, so the solver's overall performance is again influenced by the choice of the direct or iterative solver used on the coarsest grid. Although this affects the optimal runtime complexity in the serial case, it can still be used to achieve great parallel scaling behavior [35].
Second, the coarser grids can be completely transferred to a subset of the involved processes, so that the problem can be treated by an efficient parallel environment (*coarse grid agglomeration*). By this means, it's possible to process every coarse grid on the number of processes that provides the fastest computation. However, this is generally not the best approach, because in that case, the data of whole grids needs to be transferred over the network, which takes a significant amount of time itself. It's therefore advisable to reduce the number of involved processes only if the benefit of better on-grid performance clearly outweighs the additional communication costs.

It's noteworthy that while the U-cycle aims for better aggregate Mflop/s rates, the method of coarse grid agglomeration accepts temporarily idle cores to achieve faster runtime.

3.8 Alternative Linear Solvers

Multigrid methods, if applicable, are generally considered to be the fastest algorithms for solving large sparse systems of equations, but often fail due to limited capabilities in smoothing efficiently or projecting the problem to coarser grids. Additionally, a multilevel method for a given problem might not even be available. For these reasons, an optimal and effective multigrid solver can't be used in many practical cases [36].
Under such circumstances, another class of algorithms is generally used: the *Krylov subspace methods* [22, 37, 38, 39, 40].
These methods utilize that the solution $\boldsymbol{u}$ of the linear system $A\,\boldsymbol{u} = \boldsymbol{b}$ of size n (A being nonsingular) is an element of a Krylov subspace

$$\mathcal{K}_m(A, \boldsymbol{b}) = \operatorname{span}\left\{\boldsymbol{b}, A\,\boldsymbol{b}, A^2\boldsymbol{b}, \ldots, A^{m-1}\boldsymbol{b}\right\} \tag{3.45}$$

of order $m \leq n$ (see, e. g., [22] for the proof) and can further be written as

$$\boldsymbol{u} = p(A)\,\boldsymbol{b}\ , \tag{3.46}$$

where p is a polynomial of a degree not exceeding n - 1.
A Krylov subspace method approaches $\boldsymbol{u}$ iteratively by calculating an approximation $\boldsymbol{u}_m \in \mathcal{K}_m(A, \boldsymbol{b})$ on iteration m that fulfils a certain condition upon the resultant residual $\boldsymbol{r}_m = \boldsymbol{b} - A\,\boldsymbol{u}_m$. The subsequent iteration then builds upon this and constructs a new approximation in the Krylov subspace of order m+1. Using exact arithmetic, this ultimately leads to the exact solution after a maximum of n iterations, although this is hardly possible for practical cases. The process is usually sensitive regarding rounding errors, so only a limited, but generally sufficient accuracy can be reached after a comparably small number of iterations.
It's furthermore possible to adopt an initial guess $\boldsymbol{u}_0 \neq \boldsymbol{0}$. The solution is then sought on the affine subspace $\boldsymbol{u}_0 + \mathcal{K}_m(A, \boldsymbol{r}_0)$.
A valuable property of the Krylov methods is that the resulting algorithms' mathematically most complex operations are simple matrix-vector multiplications, mostly involving only A.

The imposed condition on the residual $\boldsymbol{r}_m$ is usually formulated as an orthogonal projection,

$$\boldsymbol{r}_m = \boldsymbol{b} - A\,\boldsymbol{u}_m \perp \mathcal{L}_m\ , \tag{3.47}$$

with $\mathcal{L}_m$ being another, appropriate, subspace of dimension m. The various different Krylov subspace methods arise from the choice of $\mathcal{L}_m$ (for the most generally used and well known methods, $\mathcal{L}_m$ is either $\mathcal{K}_m$ or $A\,\mathcal{K}_m$) and the utilization of special properties of the matrix A that simplify the construction of orthogonal vectors. A good choice of $\mathcal{L}_m$ enables finding an accurate approximation rather quickly, after $m \ll n$ iterations.

A prime example here is the *conjugate gradient method (CG)* [41], which uses the properties of symmetric positive definite matrices to construct a set of mutually A-conjugate vectors $\boldsymbol{P} = \{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_{n\text{-}1}\}$ that form a basis for $\mathbb{R}^n$, so that the solution $\boldsymbol{u}$ can be expressed as

$$\boldsymbol{u} = \boldsymbol{u}_0 + \sum_{i=0}^{n-1} \alpha_i\,\boldsymbol{p}_i\ . \tag{3.48}$$

It can be shown that span $\{\boldsymbol{P}\} = \mathcal{K}_m(A, \boldsymbol{r}_0)$.
The method is based on the observation that for positive definite matrices A, solving the system $A\,\boldsymbol{u} = \boldsymbol{b}$ for $\boldsymbol{u}$ is equivalent to minimizing the function

$$f(\boldsymbol{u}) = \frac{1}{2}\boldsymbol{u}^T A\,\boldsymbol{u} - \boldsymbol{b}^T\boldsymbol{u} \ . \tag{3.49}$$

This is done by a successive one-dimensional minimization,

$$\boldsymbol{u}_{m+1} = \boldsymbol{u}_m + \alpha_m\,\boldsymbol{p}_m \ , \tag{3.50}$$

where α_m is chosen such that it satisfies

$$f(\boldsymbol{u}_m + \alpha_m\,\boldsymbol{p}_m) = \min_{\alpha\in\mathbb{R}} f(\boldsymbol{u}_m + \alpha\,\boldsymbol{p}_m) \ , \tag{3.51}$$

namely

$$\alpha_m = \frac{\boldsymbol{r}_m^T\,\boldsymbol{p}_m}{\boldsymbol{p}_m^T A\,\boldsymbol{p}_m} \ . \tag{3.52}$$

By requiring the vectors of $\boldsymbol{P}$ to be mutually A-conjugate, $\boldsymbol{r}_{m+1}^T\,\boldsymbol{p}_j = 0$ for $j = 0, 1, \ldots, m$ can be derived, which in turn enables constructing them via

$$\boldsymbol{p}_{m+1} \coloneqq \boldsymbol{r}_{m+1} + \sum_{i=0}^{m} \beta_{m,i}\,\boldsymbol{p}_i \tag{3.53}$$

with

$$\beta_{m,i} = -\frac{\boldsymbol{r}_{m+1}^T A\,\boldsymbol{p}_i}{\boldsymbol{p}_i^T A\,\boldsymbol{p}_i} \ , \tag{3.54}$$

starting from

$$\boldsymbol{p}_0 \coloneqq -\boldsymbol{\nabla} f(\boldsymbol{u}_0) = \boldsymbol{r}_0 \ . \tag{3.55}$$

The mutual A-conjugacy further gives $\boldsymbol{r}_{m+1}^T\,\boldsymbol{r}_j = 0$, $j = 0, 1, \ldots, m$, which leads to $\beta_{m,i} = 0$, $i = 0, 1, \ldots, m-1$ and confirms $\mathcal{L}_m = \mathcal{K}_m$.
Some final transformations then simplify the coefficients α_m and β_m to

$$\alpha_m = \frac{\boldsymbol{r}_m^T\,\boldsymbol{r}_m}{\boldsymbol{p}_m^T A\,\boldsymbol{p}_m} \qquad \text{and} \tag{3.56}$$

$$\beta_m = \frac{\boldsymbol{r}_{m+1}^T \boldsymbol{r}_{m+1}}{\boldsymbol{r}_m^T \boldsymbol{r}_m} \ . \tag{3.57}$$

A more detailed derivation of the conjugate gradient method can be found, among others, in [22].
The resulting algorithm features relatively few operations per iteration and is exemplarily specified in algorithm 4.

A variety of other Krylov subspace methods are based on the conjugate gradient method and aim to generalize its concept of using conjugate vectors for systems that are not necessarily positive definite or symmetric. This is generally achieved at the expanse of the convergence rate and increases the number of computational steps per iteration, so being able to work with a symmetric positive definite matrix is a great advantage.
The *biconjugate gradient method (BiCG)* utilizes a second, related, Krylov subspace, $\mathcal{K}_m(A^T, \tilde{\boldsymbol{b}})$, to replace the orthogonal sequence of residuals $\boldsymbol{r}_m$ by two mutually orthogonal sequences, so that $\tilde{\boldsymbol{r}}_i^T\,\tilde{\boldsymbol{r}}_j = 0$ and $\tilde{\boldsymbol{p}}_i^T A\,\tilde{\boldsymbol{p}}_j = 0$ for $i \neq j$. The method is applicable

Algorithm 4 Conjugate Gradient Method

```
1: function CG(A, u_0, b)
2:     r_0 := b − A u_0
3:     p_0 := r_0
4:     for m := 0, 1, ..., n do
5:         α_m := (r_m^T r_m) / (p_m^T A p_m)
6:         u_{m+1} := u_m + α_m p_m
7:         r_{m+1} := r_m − α_m A p_m
8:         if CONVERGED(r_m) then
9:             return u_{m+1}
10:        end if
11:        β_m := (r_{m+1}^T r_{m+1}) / (r_m^T r_m)
12:        p_{m+1} := r_{m+1} + β_m p_m
13:    end for
14: end function
```

to general square (non-symmetric) matrices, but is considered to be quite unstable. For improved stability, the *biconjugate gradient stabilized method (BiCG-STAB)* is used, which introduces additional efforts to minimize the residuals.
By imposing the condition $\boldsymbol{r}_i^T A\,\boldsymbol{r}_j = 0$ for $i \neq j$, i. e., that the residuals are not only mutually orthogonal, but also A-conjugated, the *conjugate residual method* is found, which is very similar to the conjugate gradient method both in algorithm and convergence properties. The additional computational costs allow for the method to be used on symmetric (indefinite) matrices [37].

Another family of Krylov methods is spawned by the approach of minimizing the norm of the residual $\boldsymbol{r}$ on a Krylov subspace at every step of the iterative process. The most prominent example here is the *generalized minimal residual method (GMRES)* [42] for general (unsymmetric) matrices.
Here, the *Arnoldi iteration* [43] is used to form an orthonormal basis $\boldsymbol{V} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\}$ of $\mathcal{K}_m$, so that the approximation $\boldsymbol{u}_m$ can be written as

$$\boldsymbol{u}_m = \boldsymbol{u}_0 + V_m\,\boldsymbol{y}_m\ , \tag{3.58}$$

where the $n \times m$ matrix V_m is defined by the columns $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$ and the coefficients $\boldsymbol{y}_m$ form a vector of dimension m. In order to determine the components of $\boldsymbol{y}_m$, the transformation

$$\|\boldsymbol{r}_m\| = \|\boldsymbol{b} - A\,\boldsymbol{u}_m\| = \|\|\boldsymbol{r}_0\|\ \boldsymbol{e}_1 - H_m\,\boldsymbol{y}_m\| \tag{3.59}$$

is used (derivation, e. g., in [22]), so the goal is reformulated as minimizing the right-handed side expression, which is a linear least squares problem. The coefficients of the $(m{+}1) \times m$ upper Hessenberg matrix H_m are given by the Arnoldi iteration and $\boldsymbol{e}_1 = (1, 0, \ldots, 0)^T \in \mathbb{R}^{m+1}$.
The computational costs of the GMRES method increase with every iteration by an additional scalar product and every basis vector has to be kept available. Hence, there's a practical limit on the number of feasible iterations, which encourages using a restarted version of the algorithm, where the still insufficient approximation $\boldsymbol{u}_m$ is used as the initial guess of a new GMRES run.
Other minimal residual methods are tailored for either special matrices (*MINRES* [44]) or increased robustness (*deflated GMRES* [45]).

An in-depth convergence analysis of the Krylov subspace methods (e. g., [22, 37, 38]) leads to the general conclusion that their convergence rate strongly depends on the condition number κ of the system matrix A. I. e., the rate at which one iteration improves the approximation (minimizes the error) compared to the respective previous one is a decreasing function of the condition number

$$\kappa(A) = \left| \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)} \right| \geq 1 \tag{3.60}$$

(for normal matrices; $\lambda_{\max}$ and $\lambda_{\min}$ are the extreme eigenvalues of A). This implies that a small condition number (close to one) is a preferable property of A.
However, since A defines the linear system, decreasing the condition number to improve convergence is only achievable by reformulating the whole problem. This is done by applying the Krylov subspace method of choice to a preconditioned system of the following forms:

$$M_L^{-1} A \, \boldsymbol{u} = M_L^{-1} \boldsymbol{b} \tag{3.61}$$

$$A \, M_R^{-1} \tilde{\boldsymbol{u}} = \boldsymbol{b} \,, \qquad \boldsymbol{u} = M_R^{-1} \tilde{\boldsymbol{u}} \tag{3.62}$$

$$M_L^{-1} A \, M_R^{-1} \tilde{\boldsymbol{u}} = M_L^{-1} \boldsymbol{b} \,, \; \boldsymbol{u} = M_R^{-1} \tilde{\boldsymbol{u}} \,. \tag{3.63}$$

Solving these systems is referred to as *left*, *right*, and *symmetric preconditioning*, respectively.
Upon inclusion in a Krylov method's algorithm, the operations $M_L^{-1} A$, $A \, M_R^{-1}$, and $M_L^{-1} A \, M_R^{-1}$ are almost never performed explicitly. Instead, M_L^{-1} and M_R^{-1} are applied to a given vector and the regular matrices M_L and M_R don't actually need to be known at all. In fact, only the effects of applying M_L^{-1} and M_R^{-1} to a vector, i. e., the resulting vector, need to be known, so the preconditioner can in principle be implemented in a completely matrix-free fashion.
Once the preconditioner is established, the unpreconditioned algorithm needs to be edited only slightly. As an example, algorithm 5 shows the (left) preconditioned conjugate gradient method.
However, for such Krylov methods that require A to have special properties, e. g., being symmetric positive definite, the preconditioner matrix usually must have the same properties.

To reduce the condition number, a preconditioner effectively approximates A^{-1} to a practical extent, i. e., within reasonable additional efforts. It is therefore often based on a method that itself can be used to solve the linear system, but is altered to only yield a rough approximation of the exact solution. Thus, for example, the LU decomposition (a derivative of Gaussian elimination and therefore a direct method) that factorizes A into the product of a lower and an upper triangular Matrix L and U so that the linear system can be solved by simple forward and backward substitution, is reduced to an *incomplete LU factorization*, where only a limited number of matrix elements (usually those matching the sparsity pattern of A) are calculated. This, of course, results in an inaccurate solution for itself, but can enable a Krylov subspace method to converge within a greatly reduced number of iterations.
It is furthermore possible to utilize the stationary iterative solvers described in subsection 3.1.2 and even multigrid methods as preconditioners by imposing an insufficient tolerance.
Actually, good iterative convergence up to an insufficient limit is often observed for

Algorithm 5 Preconditioned Conjugate Gradient Method

function PCG(A, $\boldsymbol{u}_0$, $\boldsymbol{b}$)
  $\boldsymbol{r}_0 \coloneqq \boldsymbol{b} - A\,\boldsymbol{u}_0$
  $\boldsymbol{p}_0 \coloneqq M_L^{-1}\,\boldsymbol{r}_0$
  $\boldsymbol{z}_0 \coloneqq \boldsymbol{p}_0$
  for $m \coloneqq 0, 1, \ldots, n$ **do**
    $\alpha_m \coloneqq \frac{\boldsymbol{r}_m^T \boldsymbol{z}_m}{\boldsymbol{p}_m^T A\,\boldsymbol{p}_m}$
    $\boldsymbol{u}_{m+1} \coloneqq \boldsymbol{u}_m + \alpha_m\,\boldsymbol{p}_m$
    $\boldsymbol{r}_{m+1} \coloneqq \boldsymbol{r}_m - \alpha_m\,A\,\boldsymbol{p}_m$
    if CONVERGED($\boldsymbol{r}_m$) **then**
      return $\boldsymbol{u}_{m+1}$
    end if
    $\boldsymbol{z}_{m+1} \coloneqq M_L^{-1}\,\boldsymbol{r}_{m+1}$
    $\beta_m \coloneqq \frac{\boldsymbol{r}_{m+1}^T \boldsymbol{z}_{m+1}}{\boldsymbol{r}_m^T \boldsymbol{z}_m}$
    $\boldsymbol{p}_{m+1} \coloneqq \boldsymbol{z}_{m+1} + \beta_m\,\boldsymbol{p}_m$
  end for
end function

multigrid cycles. Therefore, a multigrid method that fails to converge on its own might nevertheless be an excellent preconditioner. In fact, for solving elliptic PDEs like the Poisson equation, multigrid methods are almost always the best choice.

Which combination of Krylov subspace method and preconditioner is going to perform the best for a given problem is nevertheless a complicated matter, as theoretical predictions are rare and not necessarily reliable [37].

Chapter 4

Development of a Parallel Multigrid Field Solver

For the parallel field solver required by PlasmaPIC, a three-dimensional geometric multigrid solver was written from scratch. This chapter presents the general properties of the implemented algorithm and emphasizes the customizations made to fit in to the pre-set particle-in-cell environment.

4.1 Challenges

While a lot of the basic settings, like grid size and domain partitioning, are predefined by the PlasmaPIC framework, they also need to work flawlessly on the solver. It's clear that the process on which the particle operations for a certain subdomain are processed and where the respective charge array is calculated is best suited to represent that subgrid in the field solver as well. Hence, the assignment of a process's location and of the neighboring processes is in the range of responsibilities of PlasmaPIC.
The size of the global grid is dictated by the simulated problem and the discretization necessary for the expected plasma density. The field solver therefore needs to be able to handle any given grid, which is not a straightforward transposition with multigrid, since this affects the coarsening strategy.

Another critical obstacle is the geometrical representation of arbitrarily shaped objects in PlasmaPIC either as irregular (Dirichlet) boundary conditions on the grid or as cells with a relative dielectric permittivity larger than one. By applying the standard methods described in chapter 3, an object's shape is not automatically projected onto the coarser grids and without a proper strategy it can vanish completely due to insufficient resolution. This generally deteriorates convergence behavior up to the point of actual divergence.

In addition, the performance of a multigrid solver in a particle-in-cell environment can't be expected to reach that of a variant designed to solve a continuous problem where the right-handed side vector $\boldsymbol{b}$ is given by a smooth forcing function $f(x, y, z)$. The inevitable fluctuations in the charge distribution, caused by weighting a finite number of particles to the grid, force the potential to be oscillatory as well, hence the attempts to smooth its approximation during a V-cycle are obstructed [46].

The exact solution should still follow a rough course that can be described by a smooth approximation and that is mappable to the coarser grids, but textbook efficiency is not necessarily within the realm of possibility.

All of this needs to be implemented with the capability of massive parallelization. Since the field solver is otherwise a limiting factor, proper parallel scaling must be achieved.

4.2 Adjustments to the Standard Algorithm

From here on, the finest grid in the hierarchy will be identified by $\ell = 0$. Any coarser grid up to $\ell = \ell_{\max} > 0$ then has the regular mesh size

$$h_\ell = 2^\ell \cdot h_0 \ . \tag{4.1}$$

For simplification, the following considerations are mostly based on a one-dimensional grid, but can easily be expanded to two or three dimensions.

4.2.1 Arbitrary Grid Sizes

Building a hierarchy of refining (vertex-centered) grids is comparably easy when starting from an existing coarse grid, because the number of points in every direction on the respective finer grid can then be chosen to be one less then double the number of grid points on the parental coarse grid:

$$N_{\text{fine}} = 2 \cdot N_{\text{coarse}} - 1 \ . \tag{4.2}$$

Whereas coarsening a given fine grid is delicate due to the fact that by far not every number N_{fine} has an associated (natural) number N_{coarse} that follows

$$N_{\text{coarse}} = \frac{(N_{\text{fine}} + 1)}{2} \ . \tag{4.3}$$

Since this limits the coarsening process to grid sizes that follow the sequence $N_{\text{coarsest}} \cdot (2^n - 1)$, where the number of grid points on every level above the coarsest grid is dividable by $2^n - 1$ (which is a common prerequisite in the pertinent literature), the standard (vertex-centered) coarsening is not an adequate approach to deal with arbitrarily sized grids.
Since there's no single right way to solve this problem (a projection of a fine grid to a coarse grid and the inverse operation need to exist; neither method nor grid size is set in stone, although the usual doubling in mesh size is proven to be a robust concept), even a very simple approach can be expected to be sufficient.
Considering this, the standard strategy for cell-centered multigrid methods is promising. Here, two (1D), four (2D) or eight (3D) cells are directly merged by averaging the respective grid values. The center (and therefore the grid point's position) of the new, larger, cell then doesn't align with the original fine grid, but the mesh size doubles nevertheless. The prolongation of coarse-grid values back to the fine grid then consequently is plain linear (1D) bilinear (2D) or trilinear (3D) interpolation (cf. fig 4.1).

Cell-centered coarsening relies on every grid size being divisible by two, which again is a significant limitation.

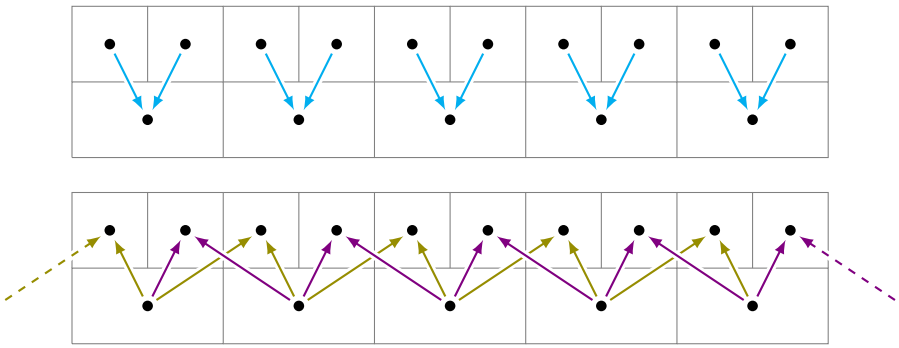

Figure 4.1: One-dimensional restriction (top) and prolongation (bottom) for cell-centered coarsening.

However, alternating both coarsening strategies is entirely possible, so that vertex-centered coarsening is used if the number of grid points is odd and cell-centered coarsening otherwise [47, 48].
In order to preserve the position of the domain boundaries, a minor modification to cell-centered coarsening has to be made: The outermost points on the left and right side, respectively, are kept unchanged and only the inner grid points are merged. As a consequence, the distance between the outermost grid points and their neighbors in inward direction on the respective coarse grid is smaller than the regular mesh size h_ℓ, which has to be accounted for on all grid operations (restriction, prolongation, smoothing, solving on coarsest grid). A simple, one-dimensional, example for this, including alternating coarsening concepts, is given in figure 4.2.

By using this alternation strategy, the number of grid points N_{gp}^ℓ on grid level $\ell > 0$ is given by

$$N_{\text{gp}}^\ell = \text{ceil}\left(\frac{N_{\text{gp}}^0 - 1}{2^\ell} + 1\right) , \tag{4.4}$$

which coincides with successively halving even numbers and using equation (4.3) on odd numbers.
The irregular distance d_{irr}^ℓ, $\ell > 0$, between the two outermost grid points on each side, respectively, is given by

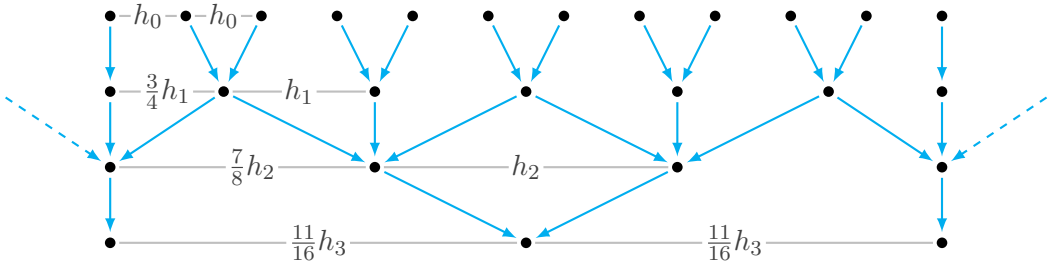

Figure 4.2: Example of a one-dimensional grid hierarchy with alternating coarsening strategy. If the number of grid points is even, cell-centered coarsening is used instead of vertex-centered coarsening. Between the first two and the last two grid points, respectively, the distance is smaller than the regular mesh size, which doubles on every successive grid.

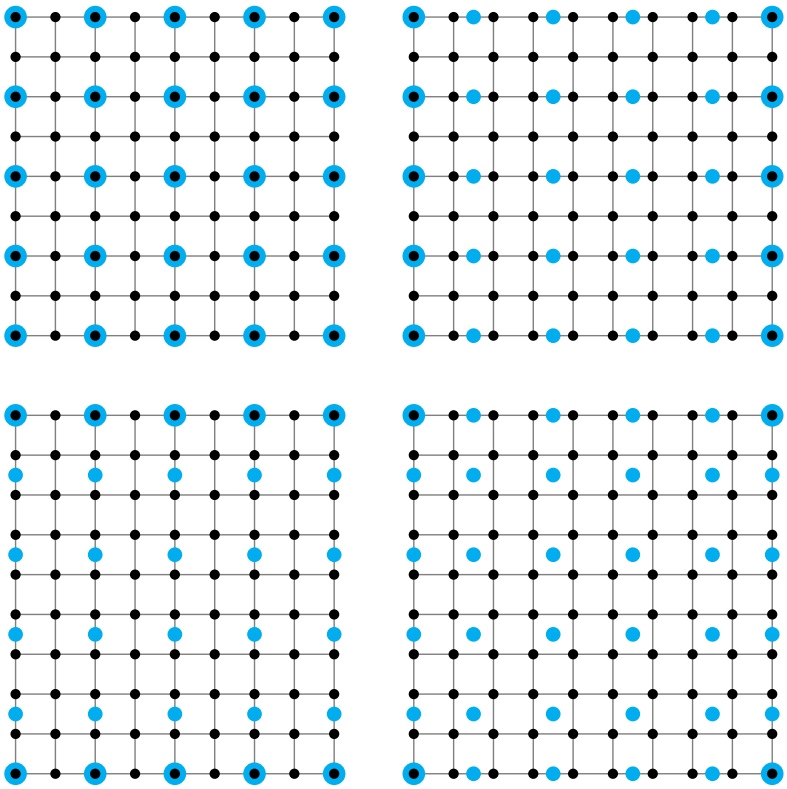

Figure 4.3: Possible combinations of how a fine (black) and a coarse (blue) grid can be aligned in 2D when using the mixed coarsening strategy. Top left: $9 \times 9 \rightarrow 5 \times 5$ grid points. Top right: $10 \times 9 \rightarrow 6 \times 5$ grid points. Bottom left: $9 \times 10 \rightarrow 5 \times 6$ grid points. Bottom right: $10 \times 10 \rightarrow 6 \times 6$ grid points. Note that the irregular distances between coarse grid points can deviate from the depicted if the fine grid already has them.

$$d_{\text{irr}}^{\ell} = h_{\ell} \cdot \begin{cases} 1 & \text{, if } \ell = 0. \\ \dfrac{d_{\text{irr}}^{\ell-1} + 1}{2} & \text{, if } \ell > 0 \text{ and } N_{\text{gp}}^{\ell-1} \text{ odd.} \\ \dfrac{d_{\text{irr}}^{\ell-1} + \frac{1}{2}}{2} & \text{, if } \ell > 0 \text{ and } N_{\text{gp}}^{\ell-1} \text{ even.} \end{cases} \tag{4.5}$$

In the two- and three-dimensional case, a fine grid can consist of an even number of grid points in one direction and an odd number in another one, thus both coarsening schemes have to be mixed. This increases the number of possibilities of how the fine and the coarse grid points align and affects both restriction and prolongation. A coarse-grid point can then be located directly on a fine-grid point, or in between two, four, or eight (only in 3D). Figure 4.3 illustrated this for the two-dimensional case.

The Restriction Scheme

The algebraic weighting described in section 3.2 can be derived the following way: The coarse-grid point "touches" eight fine-grid cells, each of which with eight vertices. The associated values of these eight fine-grid points are used to find an average value for the respective cell. Further averaging of these eight new cell-centered values then yields the

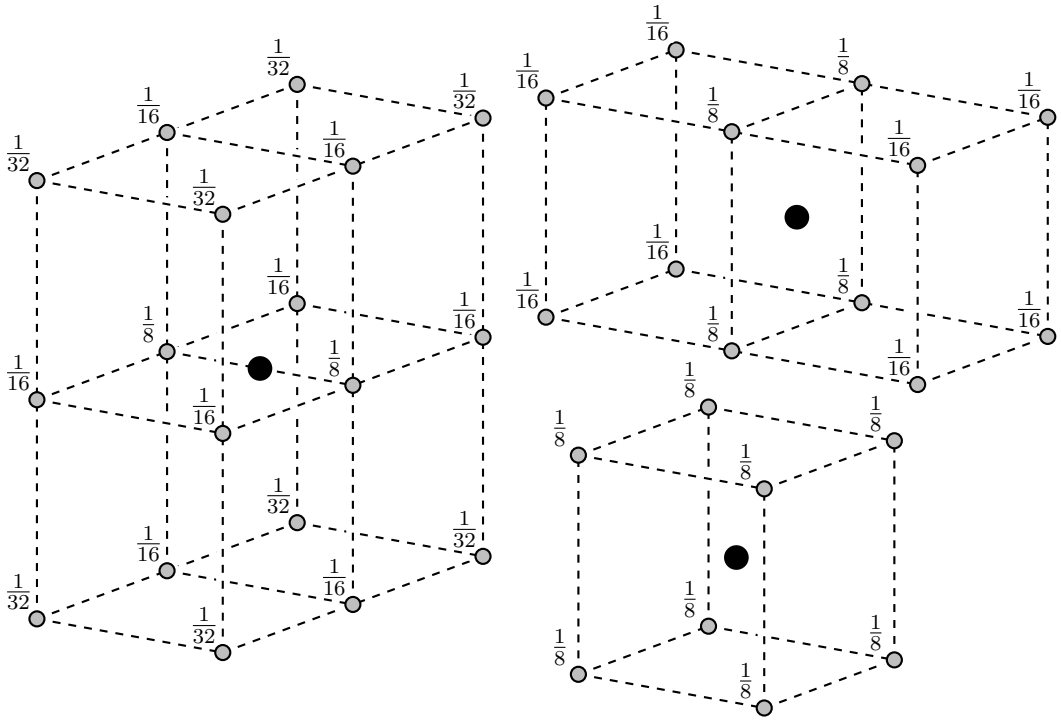

Figure 4.4: Variations of how a coarse-grid point can lie within the fine grid. In each case, the weighting factors for restriction are given.

coarse-grid value. The various weighting factors are obtained by combining these two steps and merging multiple occurrences.
When using the mixed coarsening approach, the coarse-grid point may "touch" fewer fine-grid cells. In fact, there are eight different ways of how the fine and the coarse grid can align (relevant examples are given in figure 4.4). By using the same two steps of averaging on only the connected cells, the unique weighting factors for each variant can be determined.
Furthermore, this procedure offers the possibility to adequately handle the restriction at irregularities in mesh size, which occurs at Neumann domain boundaries (cf. figure 4.2). Here, the cell-centered values obtained after the first step are calculated the same way, but are then weighted based on their distance to the coarse-grid point.

The Prolongation Scheme

Similarly to the restriction scheme, the increased number of possibilities of how a pair of fine and coarse grids can align necessitates a modification of the interpolation from the coarse to the fine grid.
The significant change is that a fine-grid point within a coarse-grid cell is now no longer necessarily centered between two, four, or eight coarse-grid points, but can be located anywhere reachable by shifts of quarter the coarse mesh size in any spatial direction.
Consequently, the weighting factors for linear, bilinear, and trilinear interpolation change according to the respective distances. An example is given in figure 4.5.
Furthermore, irregular distances can can now easily be taken into account.

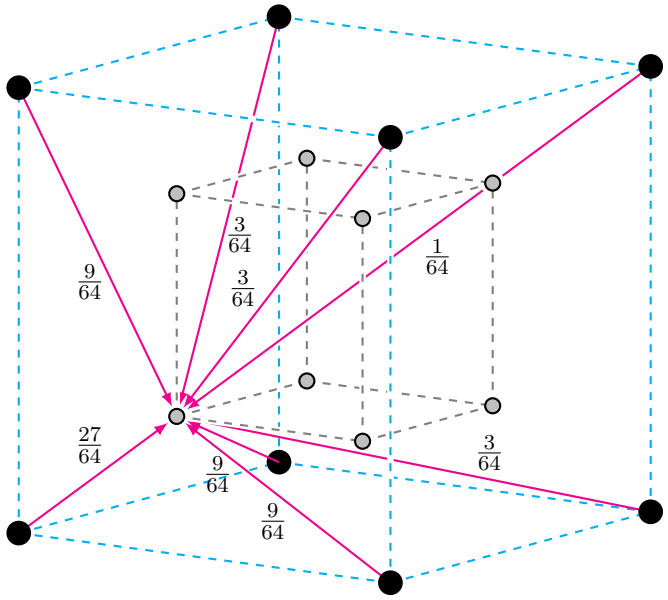

Figure 4.5: Example of how trilinear interpolation is performed in the case of cell-centered coarsening. Other variations including linear and bilinear interpolation to fine-grid points on the edges or sides of the coarse-grid cell are possible, if the fine grid has an odd number of grid points in at least one direction.

The Poisson Equation

The introduction of irregular distances between grid points affects the discretization of partial differential equations in general and therefore of Poisson's equation as well. Since for equation (3.7) to be applicable, a constant distance h is needed, so a more general formulation has to be used. Assuming the point u_{i-1} is at distance $p \cdot h$ from u_i and u_{i+1} is at distance $q \cdot h$ $(p, q < 1)$, the required expression can be derived using central differences for both derivatives:

$$(\Delta u)_i = \frac{\frac{u_{i+1}-u_i}{q\,h} - \frac{u_i-u_{i-1}}{p\,h}}{\frac{1}{2}\,(p+q)\,h} \tag{4.6}$$

$$= \frac{1}{h^2}\left[\frac{2}{p\,(p+q)}\,u_{i-1} - \frac{2}{p\,q}\,u_i + \frac{2}{q\,(p+q)}\,u_{i+1}\right] . \tag{4.7}$$

The full equivalent to equation (3.7) then is

$$\begin{aligned}(\Delta u)_{ijk} = \frac{1}{h^2}\Bigg[&\frac{2\,u_{i\text{-}1jk}}{p_x\,(p_x+q_x)} + \frac{2\,u_{ij\text{-}1k}}{p_y\,(p_y+q_y)} + \frac{2\,u_{ijk\text{-}1}}{p_z\,(p_z+q_z)} \\ &+ \frac{2\,u_{i+1jk}}{q_x\,(p_x+q_x)} + \frac{2\,u_{ij+1k}}{q_y\,(p_y+q_y)} + \frac{2\,u_{ijk+1}}{q_z\,(p_z+q_z)} \\ &- \left(\frac{2}{p_x\,q_x} + \frac{2}{p_y\,q_y} + \frac{2}{p_z\,q_z}\right) u_{ijk}\Bigg] = -f_{ijk} \ ,\end{aligned} \tag{4.8}$$

which leads to

$$u_{ijk} \coloneqq \left(\frac{1}{p_x\,q_x} + \frac{1}{p_y\,q_y} + \frac{1}{p_z\,q_z}\right)^{-1} \cdot \left(\frac{u_{i\text{-}1jk}}{p_x\,(p_x+q_x)} + \frac{u_{ij\text{-}1k}}{p_y\,(p_y+q_y)} + \frac{u_{ijk\text{-}1}}{p_z\,(p_z+q_z)} + \frac{u_{i+1jk}}{q_x\,(p_x+q_x)} + \frac{u_{ij+1k}}{q_y\,(p_y+q_y)} + \frac{u_{ijk+1}}{q_z\,(p_z+q_z)} - \frac{1}{2}\,h_\ell^2\,f_{ijk}\right) \tag{4.9}$$

for the Gauss-Seidel method. The calculation of the residual and an SOR iteration is done analogously.

The same approach as for equation (4.8) can be used to include variable coefficients. Compared to equation (3.38), the coefficients $c_1, \ldots, c_7$ are firstly calculated by weighted average and secondly multiplied by those in the above equation. Overall, the coefficients therefore remain calculatable at initialization.

4.2.2 Arbitrary Geometries

Introducing irregular Dirichlet boundary conditions in the form of fixed-value grid points is a straightforward approach for the standard iterative solvers such as SOR, as they can be implemented by simply not calculating new values for the respective grid points. For a geometric multigrid solver on the other hand, the existence of a hierarchy of coarser grids complicates the matter.

Three approaches to handle irregular boundaries shall be discussed here in the context of a simple model problem: Two opposing sides of a grid consisting of $64 \times 64 \times 64$ grid points define a plate capacitor at constant voltage (0 V and 100 V). The remaining domain boundaries are defined as homogeneous Neumann. At the center of the grid, a spherical charge distribution of -1 $\mathrm{mC/m^3}$, optionally enclosed by a spherical shell of variable thickness on a fixed potential of 50 V, is located.

First, a multigrid method where the irregular boundaries (the spherical shell in the model problem) are only accounted for on the finest grid is considered. Here, the smoothing process is limited to the non-boundary points and the residual is assumed to be zero for the boundaries. The restriction process is then applied to all residual values, independent on whether a point defines a boundary or not. Upon prolongation from the coarse grids, corrections are only added to non-boundary points.
The approach is then tested without a shell, with a thin shell, and with a thicker shell. Figure 4.7(a) shows the development of the Euclidean norm of the residual values over 20 V-cycles with two pre- and two post-smoothing steps, respectively. It can generally be expected that the lower the value of $\|\boldsymbol{r}\|$ is, the better the approximation to the exact solution. Without irregular boundaries, the residual norm decreases rapidly and the approximation even approaches machine precision (recognizable by the eventual bend in the curve). With boundaries, however, the solution process diverges, independent of the shell's thickness.

For the second approach, coarse grid points are marked as boundary points if their respective position in the finest grid coincides with one. For purely vertex-based coarsening, this is easily done by comparing each coarse-grid point with the respective matching point on the finest grid. If cell-centered coarsening is used, which is the case for 64 grid

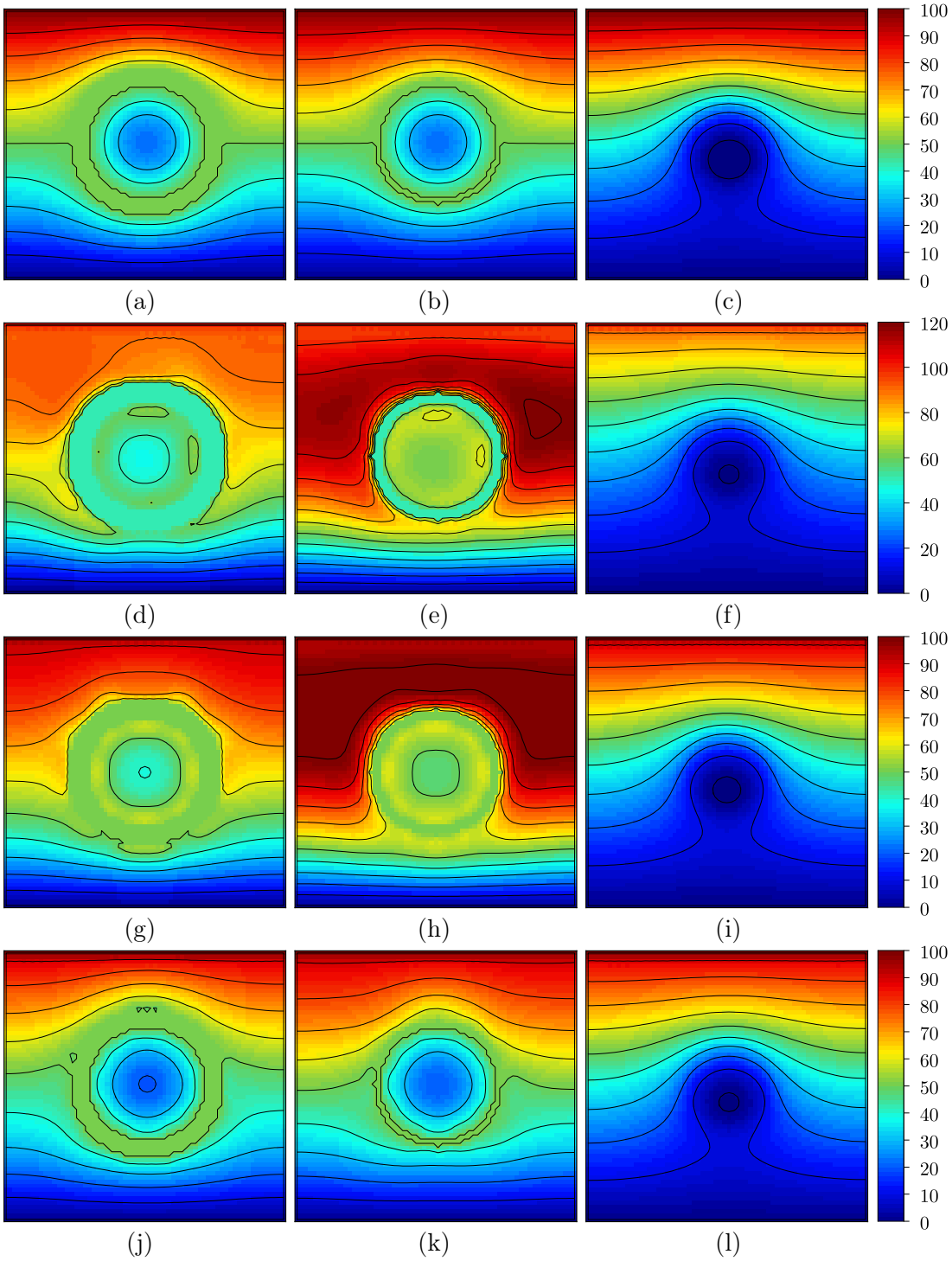

Figure 4.6: Simple model Poisson problem to demonstrate the necessity and effectiveness of the modifications to the geometric multigrid method presented here. Between two capacitor plates, a spherical homogeneous charge distribution is either enclosed within a thick or a thin shell (defined by Dirichlet boundary points), or not further influenced. (a-c) Cross-sectional view of the exact solution for the electric potential for the three cases. (d-f) Approximate solution after one V-cycle of a geometric multigrid approach that only maps the irregular boundaries to the coarser grids, applied to a zero initial guess. (g-i) Same as (d-e), but the system size is increased from 64^3 to 65^3. (j-l) Approximate solution after one V-cycle of the approach described here.

points per direction, the coarse-grid points don't necessarily align with the finest grid. Here, the fine-grid points that surround a coarse-grid point's position in the domain need to be considered.
Since the point density decreases with every additional grid level, this approach doesn't completely conserve the shape of the boundaries over the grid hierarchy.
As shown in figure 4.7(c), the residual norm decreases steadily, but the gradient of the curve depends strongly on the irregular boundaries. The convergence rate is better for the case where a thick shell is used, which can be explained by a better representation of the boundaries on the coarser grids. As expected, the results are identical to the first approach with no shell.
The convergence rate of this approach is further dependent on the specific coarsening scheme. With $64 \times 64 \times 64$ grid points, cell-centered coarsening is used on every grid level. By increasing the system size to $65 \times 65 \times 65$ grid points, vertex-based coarsening is enforced. In this case, the reduction of the residual norm, shown in figure 4.7(d), is generally better, but still dependent on the thickness of the shell.
A closer investigation of the approximate solution after one V-cycle further reveals that the charges inside the shell are not completely shielded from the outside during the solution process, which is recognizable by a short drop of the potential at the outer surface of the shell (figs. 4.6(d,e,g,h)).

An algebraic multigrid solver could principally be used to circumvent the problem of handling irregular boundaries. However, by using the coarsening strategy described in section 3.5, every boundary point would be transferred to every coarse grid, because its value in the solution vector is not influenced by any other grid points. Although resolving this issue is feasible by using a customized algorithm, the additional computational effort only adds to the list of disadvantages compared to geometric multigrid methods, along with generally poor parallel performance. Therefore, a further sophisticated strategy is needed to adjust geometric multigrid to this issue.

In a cell-centered multigrid approach, McAdams et al. [49] set whole cells, defined by eight vertices, as boundaries and pass this property to a cell on a coarser grid if any of its eight parental cells is a boundary. In iteratively called V-Cycles, the arising geometrical discrepancies lead to highly oscillatory or divergent behavior that can only be overcome by massive additional smoothing along the boundaries, thus devaluing the approach as a pure solver, limiting it to being used as a preconditioner.
For their adaptive mesh refinement solver, implemented as part of a fluid dynamics code for astrophysical applications, Guillet et al. [50] reconstruct the fine-grid boundaries by assigning a mask value m, $-1 \leq m \leq 1$, to the cell-centered grid points. In this configuration, the boundary surface is defined by the cell faces, so a point's value of m is either -1 (boundary) or $+1$ (non-boundary) on the finest grid. On the coarser grids, m is averaged from the eight parental grid points; a value $m \leq 0$ indicates a boundary and $m > 0$ otherwise. If a point is then part of a boundary, its neighboring non-boundary points use its m value to determine the actual boundary position and use the second order boundary reconstruction by Gibou et al. [51] to calculate an individual boundary value via linear extrapolation.
Here again, small and thin structures may not appear on the coarse grids, because averaging evens out the values of m, making them overall positive. On complex geometries, this results in divergence, which the authors propose to overcome by enforcing boundary conditions to coarse points that mark a local minimum in the distribution of m, which then, however, negatively affects the convergence rate.

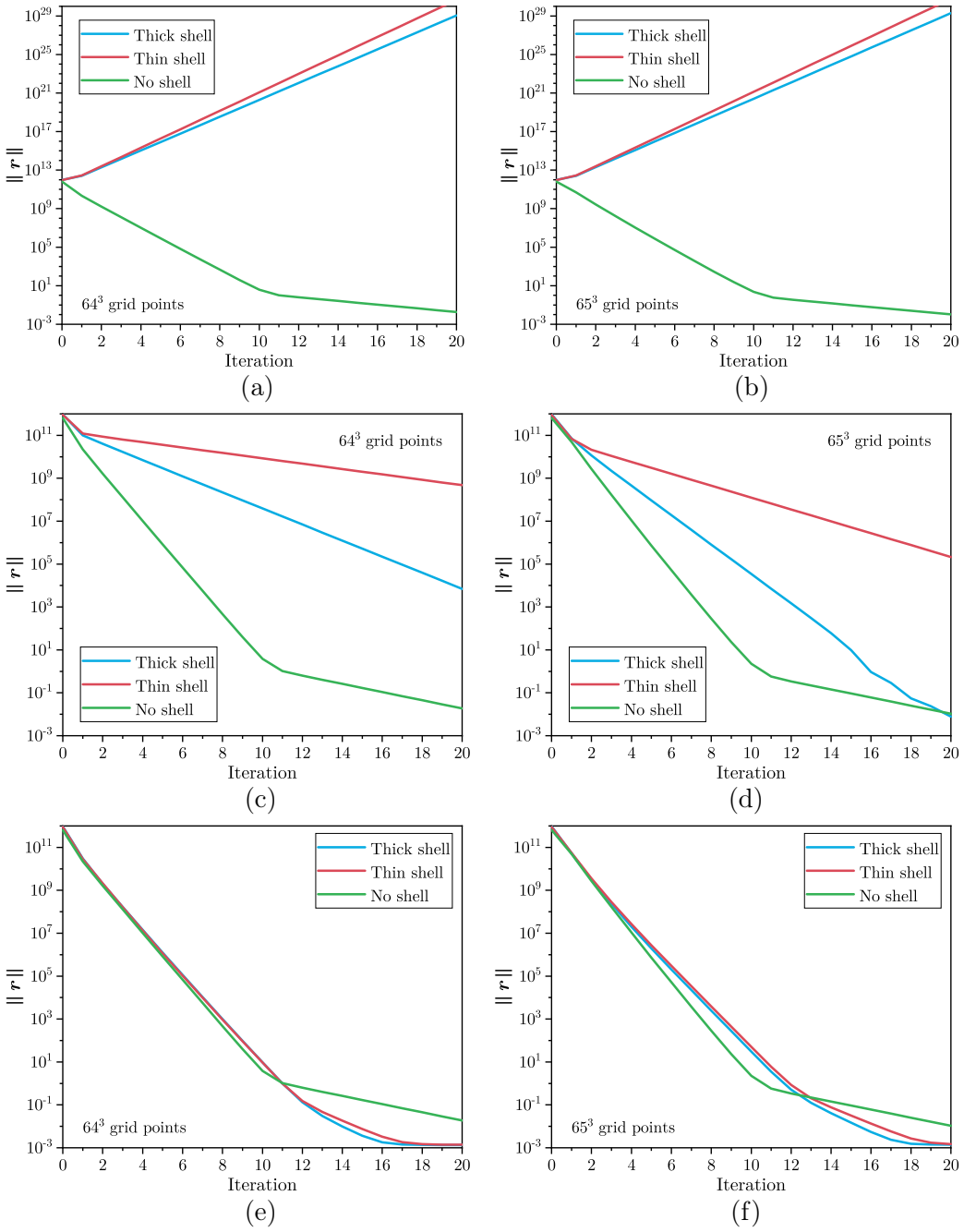

Figure 4.7: Course of the residual norm over 20 V-cycle iterations on the model problems for the three approaches. (a) First approach (64^3 grid points). (b) First approach (65^3 grid points). (c) Second approach (64^3 grid points). (d) Second approach (65^3 grid points). (e) Third and final approach (64^3 grid points). (f) Third approach (65^3 grid points).

Many other approaches utilize the finite volume method [52, 53, 54, 55, 56]. Here, the discrete system representing the respective partial differential equation is derived by evaluating fluxes through the surface of finite volumes (cells) enclosing each grid point. For example, the Poisson equation is first transformed by applying a volume integral,

$$\int_{\text{cell}} \nabla^2 u(\boldsymbol{r})\,\mathrm{d}V = -\int_{\text{cell}} f(\boldsymbol{r})\,\mathrm{d}V \quad , \tag{4.10}$$

which is then converted to a surface integral using the divergence theorem:

$$\int_S \boldsymbol{\nabla} u(\boldsymbol{r})\, \boldsymbol{n}\, \mathrm{d}S = -\int_{\mathrm{cell}} f(\boldsymbol{r})\, \mathrm{d}V \ . \tag{4.11}$$

On a Cartesian grid with mesh size h, equation (3.7) follows if

$$\begin{aligned} (\boldsymbol{\nabla} u)_{ijk} \ = \ & \frac{u_{i\text{-}1jk} - 2\,u_{ijk} + u_{i+1jk}}{2\,h}\,\boldsymbol{e}_i + \frac{u_{ij\text{-}1k} - 2\,u_{ijk} + u_{ij+1k}}{2\,h}\,\boldsymbol{e}_j \\ & + \frac{u_{ijk\text{-}1} - 2\,u_{ijk} + u_{ijk+1}}{2\,h}\,\boldsymbol{e}_k \end{aligned} \tag{4.12}$$

is used. Along with cell-centered multigrid, this formalism allows for irregular boundaries to be evaluated on the coarse grids by treating them as additional interfaces that restrict the cell volume if they cut through a coarse-grid cell.
While this can be used to perfectly conserve the shape of an embedded boundary, it also introduces a lot of challenges. If the boundary's surface is not smooth, e. g., because it is aligned with the finest grid (which is the case in PlasmaPIC), evaluating the passing fluxes can become quite complex. Additionally, the grid point may no longer be centered in the considered volume, which invalidates equation (4.12), or be located outside of it.

In the context of this work, a different approach, the third one discussed here, is chosen that similarly aims to reconstruct the exact shape of boundaries, but with finite differences. First, like for the second approach, if the coarse-grid points are spatially located in the same position as points on the finest grid, which is only the case if the number of grid points on the finest grid is odd in every direction, one is marked as a boundary point if (and only if) the matching point is already marked as such. If the coarse points are displaced relative to the finest grid, this is done if all of the two, four, or eight surrounding fine-grid points are boundaries. This prevents the boundaries from becoming increasingly dominant along the different grid levels and maximizes the number of non-boundary points.
Second, the use of a regular mesh size in the used routines is diminished by introducing varying distances to the discretized Laplace operator of grid points that are located next to a boundary point on the coarse grids. By counting the steps of length h_0 it takes to reach the boundary on the finest grid, the actual distance is obtained and can be used for calculation. Furthermore, the property of a grid point of neighboring a boundary is passed on to a matching point on the next coarser grid, which prevents boundaries from not being resolved, as long as the reason for this to occur is the boundary being a thin layer (one-dimensional thinness). In other words, a boundary must stretch beyond the size of a coarse-grid cell in at least two dimensions or otherwise small features can still end up not being resolved. Here again, a displacement of the coarse-grid points relative to the finest grid is accounted for; in this case by counting half-steps $h_0/2$ to a boundary or a line of length h_0, $\sqrt{2}\,h_0$, or $\sqrt{3}\,h_0$, that connects two boundary points.
A coarse-grid point can therefore be marked as being in close proximity to a boundary even though all its neighboring points on the coarse grid are regular non-boundary points. In this case, the respective neighbor's value v_{ijk} cannot be used for calculation and it's simply assumed to be zero. This is supported by the fact that the coarse-grid system $A^{2h}\,\boldsymbol{u}^{2h} = \boldsymbol{b}^{2h}$ solves for the error of the fine-grid system (correction scheme), which is zero on Dirichlet boundaries.
In this manner, the geometry of fixed boundaries can be conserved with adequate accuracy for every calculation (cf. figure 4.8).

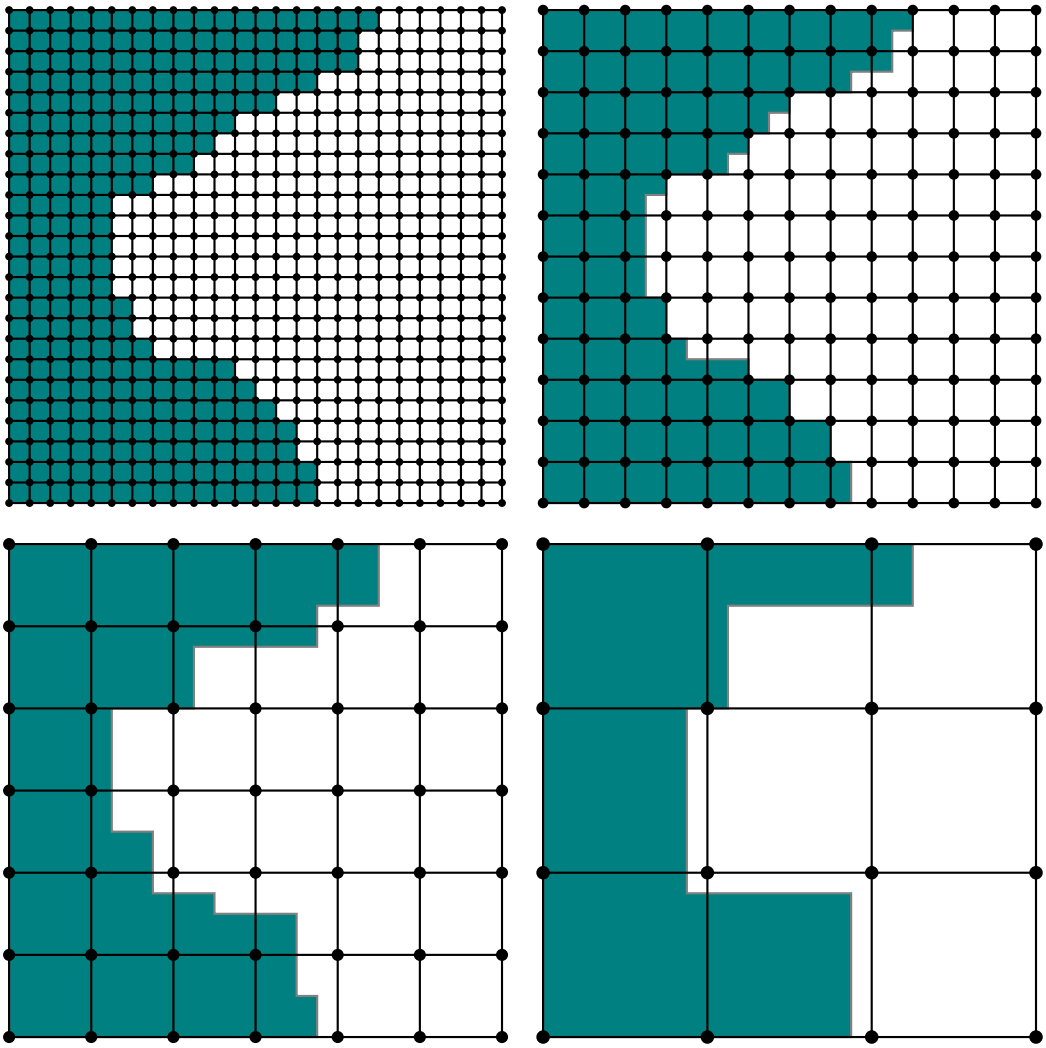

Figure 4.8: Reconstruction of a not further specified object's surface on coarser grids, as done for the field solver. The colored area indicates how the boundary is perceived by the grid points located outside of it.

Similarly to subsection 4.2.1, the subroutines need to be adapted accordingly, using the same formulas.

This method of accounting for an interface that doesn't align with the grid was first described by G. H. Shortley and R. Weller in 1938 [57, 58] (for two-dimensional grids). They applied it to the Gauss-Seidel method, which they additionally sped up by starting the solution process on coarse grids and gradually refining the obtained solution. They therefore also utilized a premature multigrid technique, best described as a variation of the nested iteration method, where the smoothing process on each grid is pursued until convergence is reached.
For them, however, the interface was defined by a continuous function not aligning with the finest grid and further implications for the various multigrid routines were naturally not considered.

Restriction and prolongation are kept unchanged except for the Dirichlet boundary conditions where the residual $\boldsymbol{r} = \boldsymbol{b} - A\,\boldsymbol{v}$ is zero (because the error is zero). The values

for the right-handed side vector $\boldsymbol{b}$ of matching coarse-grid points are directly assumed to be zero as well, without weighting in the surrounding grid points. By this, a sharp distinction between regular and Dirichlet points is achieved. Since the respective fine-grid points are fixed, interpolating a coarse-grid correction is distorting and therefore not performed as well.

As shown in figures 4.7(e) and (f), this third approach consistently reduces the residual norm at a rate very similar to the case without irregular boundaries (for which all three approaches produce the same values). This is independent of both the thickness of the shell in the model problem and the coarsening scheme.
This is achieved by thoroughly separating the inside and the outside of the spherical shell (indicated in figs. 4.6(j) and (k)), or, more generally, the two sides of a thin object.

4.3 The Coarse Grid Solver

The following considerations are solely based on the number of floating point operations needed to perform a task and ignore memory allocation and accesses, assuming similar consequences.

Assembling the system matrix A for the coarsest grid is barely more difficult than for the finest grid, as the integer values 1 and 6 in equation (3.7) are simply replaced by coefficients for equation (4.8). Hence the matrix A has the same non-zero structure that is independent of the right-handed side vector $\boldsymbol{b}$. It is therefore possible to use an arbitrary method for solving sparse linear systems, including direct methods such as variations of Gaussian elimination (decomposition methods) and even direct matrix inversion (and subsequent multiplication of the inverse with $\boldsymbol{b}$).
Since only $\boldsymbol{b}$ changes for successive iterations of the particle-in-cell method, the process of assembling the matrices associated with a solution method only has to be performed once and therefore carries no significant computational weight. Using the inverse matrix to solve the coarse system of n unknowns therefore only consists of calculating n scalar products of n-dimensional vectors ($2n^2$ floating point operations in total).
An alternative, and widely used for this purpose, is LU decomposition, a modification of Gaussian elimination. Here, the system matrix A is factorized into an upper triangular matrix U and a lower triangular matrix L, so that

$$A = L\,U\ . \tag{4.13}$$

The solution of the linear system $A\,\boldsymbol{x} = \boldsymbol{b}$ is then calculated by first using forward substitution to solve

$$L\,\boldsymbol{z} = \boldsymbol{b} \tag{4.14}$$

for $\boldsymbol{z}$ and then backward substitution to solve

$$U\,\boldsymbol{x} = \boldsymbol{z}\ . \tag{4.15}$$

Element m of the vectors $\boldsymbol{z}$ and $\boldsymbol{x}$ is calculated using

$$z_m = b_m - \sum_{i=1}^{m-1} \ell_{m,i}\, z_i \quad \text{and} \tag{4.16}$$

$$x_m = \frac{z_m - \sum_{i=m+1}^{n} u_{m,i}\, x_i}{u_{m,m}} \tag{4.17}$$

(the elements on the main diagonal of either L or U can be chosen to be one), which amounts to $2\,n^2 - n$ floating point operations in total once L and U are known (less than multiplication with the inverse of the system matrix).

However, since only an approximation of finite accuracy is strictly required, an iterative method such as SOR can offer competitive behavior. A single iteration involves eight multiplications and seven additions (using $\boldsymbol{u}$ as the solution vector again):

$$\begin{aligned} u_{ijk} \coloneqq (1-\omega)\, u_{ijk} + \omega \left(\frac{1}{p_x\, q_x} + \frac{1}{p_y\, q_y} + \frac{1}{p_z\, q_z} \right)^{-1} \cdot \left(\frac{u_{i\text{-}1jk}}{p_x\,(p_x + q_x)} \right. \\ + \frac{u_{ij\text{-}1k}}{p_y\,(p_y + q_y)} + \frac{u_{ijk\text{-}1}}{p_z\,(p_z + q_z)} + \frac{u_{i+1jk}}{q_x\,(p_x + q_x)} + \frac{u_{ij+1k}}{q_y\,(p_y + q_y)} \\ \left. + \frac{u_{ijk+1}}{q_z\,(p_z + q_z)} - \frac{1}{2}\, h_\ell^2\, f_{ijk} \right) \ . \end{aligned} \tag{4.18}$$

So even if the coefficients introduced in section 4.2 are used for every grid point, one sweep over all coarse-grid points only involves $15\,n$ floating point operations. The computational work required for solving the coarsest system with LU decomposition can therefore be equated to a certain number of SOR iterations, as shown in table 4.1.

Size of coarsest grid	flops (LU)	SOR iterations (equiv.)
$3 \times 3 \times 3$	$1,431$	3.5
$4 \times 4 \times 4$	$8,128$	8.5
$5 \times 5 \times 5$	$31,125$	16.6
$6 \times 6 \times 6$	$93,096$	28.7
$7 \times 7 \times 7$	$234,955$	45.7
$8 \times 8 \times 8$	$523,776$	68.2
$9 \times 9 \times 9$	$1,062,153$	97.1
$10 \times 10 \times 10$	$1,999,000$	133.3

Table 4.1: Number of floating point operations and equivalent number of SOR iterations on various coarsest grids

Empirically, fewer SOR iterations are necessary for the outcome of a V-cycle to be indistinguishable from one where a direct solver was used for the coarsest grid (which means that the interpolation process introduces an error larger than that of the approximative solution).
Additionally, LU decomposition is an inherently sequential algorithm and parallelizing it is quite a complex task [59].
However, since the concept of coarse grid agglomeration in principle allows for the coarsest grid to be solved on a single core, both SOR and LU decomposition can be further optimized for this case by exploiting lexicographic ordering of the grid points.
The special structure of A, being sparse with non-zero entries only on the main and six secondary diagonals, affects the properties of both L and U. By associating element i' of the vectors $\boldsymbol{x}$ and $\boldsymbol{b}$ with grid point (i, j, k) via

$$i' = i + j \cdot N_i + k \cdot N_i \cdot N_j \ , \tag{4.19}$$

where N_i and N_j are the number of grid points in x- and y-direction, the secondary diagonals are defined by the matrix entries a_{mo} with $o = m \pm 1$, $o = m \pm N_i$, and

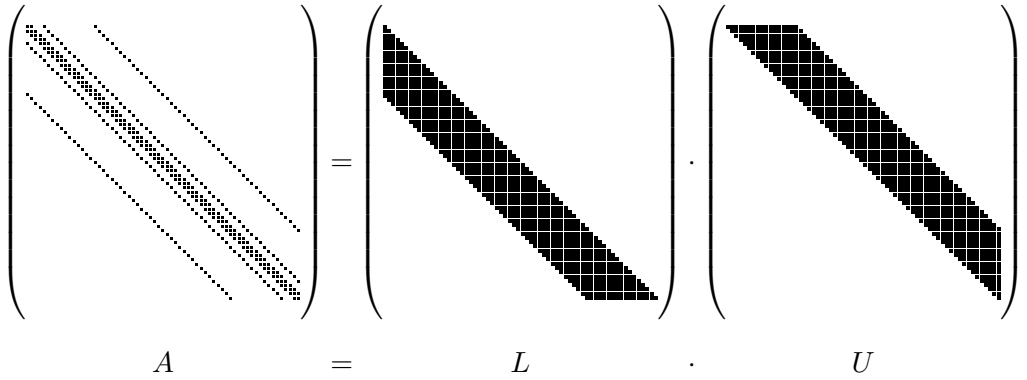

Figure 4.9: Example for the non-zero structure of the matrices A, L, and U for a three-dimensional finite-difference problem. The system size is $n = 4 \cdot 4 \cdot 4 = 64$ and no Dirichlet boundaries exist on the grid. The outermost secondary diagonals of A define the bandwidth of L and U.

$o = m \pm N_i \cdot N_j$. The non-zero entries of L are then limited to the elements ℓ_{mo} with $o \geq m - N_i \cdot N_j$ (and of course $o \leq m$). Analogous to this, the non-zero entries of U are confined to the band defined by $o \leq m + N_i \cdot N_j$ and $o \geq m$ (cf. figure 4.9).
This allows for the forward and backwards substitutions to be implemented much more efficiently regarding both number of floating point operations and memory usage. Because the number of non-zero entries in every line of L and U is then at most $N_i \cdot N_j + 1$, solving for $\boldsymbol{x}$ therefore involves exactly $2\,N_i\,N_j \cdot (2n - N_i\,N_j - 1) + n$ floating point operations (which equals to $4\,n^{5/3} - 2\,n^{4/3} + n - 2\,n^{2/3}$ for cubic grids).

While this is still not competitive with SOR for large systems ($\mathcal{O}(n^{5/3})$ compared to $\mathcal{O}(n^{4/3})$), the computational efforts for completion on small systems are nevertheless reduced so much that it's a matter of how accurate the SOR method needs to be for it to be faster (cf. table 4.2).

Size of coarsest grid	flops (LU)	SOR iterations (equiv.)
$3 \times 3 \times 3$	819	2.0
$4 \times 4 \times 4$	3,616	3.8
$5 \times 5 \times 5$	11,325	6.0
$6 \times 6 \times 6$	28,656	8.8
$7 \times 7 \times 7$	62,671	12.2
$8 \times 8 \times 8$	123,264	16.0
$9 \times 9 \times 9$	223,641	20.5
$10 \times 10 \times 10$	380,800	25.4

Table 4.2: Number of floating point operations after optimization for processing on a single processor and equivalent number of SOR iterations on various coarsest grids

For the SOR method, using only one core for solving the coarsest grid enables the abandonment of the alternation between red and black colored grid points and replacing it with sweeping in ascending order, which leads to slightly faster convergence.
However, for both the sequential and parallel case, the analytical expression for the optimal relaxation parameter ω_{opt} (equation (3.17)) is unlikely to be the best choice, firstly because point-individual coefficients are used to handle irregular distances and

secondly because Dirichlet boundary conditions may effectively reduce the system size. For example, if the discharge chamber of a RIT is defined by Dirichlet points, its interior is shielded from the rest of the domain and the volume of interest only covers a subsection of the whole grid. In this case, the best value for ω is actually smaller than ω_{opt} [60] and can only be found by systematic trial and error.
For small systems such as the coarsest grid, this is a simple task and can be done in a further negligible amount of time, once during initialization (if SOR is used in the first place).

Following the considerations above, the SOR iterative method was chosen for the parallel case and specialized sequential variants of both SOR and LU decomposition were implemented. The user can then either chose directly (and set a desired accuracy for the SOR solver) or let the benchmark module described in the following section find the fastest approach.

4.4 Measures for Efficient Parallel Performance

A standard approach to parallelize FDM solvers that is utilized here as well is the use of *ghost nodes* that each process allocates specifically to hold the boundary information of its neighbors. Their values are only updated during the communication phase and are only used to calculate new values for the process's outer grid points. The allocated grids of the respective processes therefore overlap, so the overall memory consumption actually increases the more processes are used. Since this doesn't really affect performance, it's rather insignificant.

A simple, yet valuable, modification to decrease the required bandwidth per communication step is to use single-precision floating-point data types instead of the double-precision data types used otherwise. This effectively halves the transmission time of every sent message, therefore enabling a more convenient ratio of communication and computation time, which comes in handy especially when progressing to the coarser grid levels.
The influence on the solver's precision is consequently negative, but shows to be negligible for the relatively high tolerances of PlasmaPIC (further discussed in chapter 5). However, if high accuracy is needed, the data type for communication can simply be switched back to double-precision.

Further far-reaching measures that were implemented to improve parallel performance are discussed in the following.

4.4.1 Utilizing Red-Black Ordered Gauss-Seidel Smoothing

Although their smoothing properties are similar, the (weighted) Jacobi method has the disadvantage of requiring twice as much memory as the Gauss-Seidel method at implementation because the outcome of every iteration has to be collectively preserved for every calculation of the next iteration.
As pointed out in, for example, [61], using the correction scheme in combination with red-black ordered Gauss-Seidel smoothing entails some additional advantages over other methods that are easy to utilize:

After a full sweep over both colors, the residual $\boldsymbol{r} = \boldsymbol{b} - A\,\boldsymbol{v}$ of the black points is automatically zero, which results from the unweighted Gauss-Seidel method being equivalent to successively setting each component of the residual to zero and the black points only being dependent on the red points. Therefore, only half of the residuals have to be calculated.
For the restriction step, this implies that of the maximal number of 27 fine-grid points, only 13 have a non-zero residual. Since only the residuals are used to calculate the coarse-grid right-handed side vector for the correction scheme, this can be taken advantage of to reduce the computational effort and number of memory accesses by approximately half.
Similarly, the prolongation process only needs to be applied to the black fine-grid points, because the red values are updated in the first post-smoothing step without being used themselves.

By applying all of this, another feature becomes apparent: The residual of the red points can be stored in the same memory location as the respective components of the vector $\boldsymbol{v}$, because these values aren't used again afterwards. This implies that out of the three vectors that are held for each grid, $\boldsymbol{v}$, $\boldsymbol{b}$, and $\boldsymbol{r}$, only two need to actually be allocated, which reduces the memory consumption significantly. Usable values for the red grid points are simply produced again at the first post-smoothing sweep.

4.4.2 Coarse Grid Agglomeration

As already briefly discussed in section 3.7, two problems arise at parallelizing the multigrid algorithm. First, the domain partitioning method cannot be used without proper adjustments if the number of parallel threads approaches the number of grid points on the coarser grid levels, resulting in zero assigned points for certain threads. Second, with decreasing system size along the different grid levels, the ratio of computation and communication shifts to the disadvantage of computation, which leads to increasingly poorer parallel performance, as the communication can hardly be hidden by simultaneous calculations.
Both problems can be solved by the method of coarse grid agglomeration, which essentially boils down to the realization that program runtime can actually be improved by a decrease of parallelism. Between certain successive grid levels, to be specified at initialization of the solver, the inter-grid transfer operators, restriction and prolongation, additionally transfer all needed data to a subset of the involved processes.
This subset is chosen in a way that the processes are evenly spread among the allocated computational nodes to utilize maximal bandwidth, unless they fit on a single node, in which case the communication can take place without the use of the cluster network.

Since this method requires whole coarse-grid vectors to be sent over the network, which reduces performance, it shouldn't be applied to every fine-to-coarse transition. On which grid levels and to how many processes this redistribution of the problem is performed is therefore a critical issue for optimization.

4.4.3 Determination of Optimal Parameters

There's a variety of parameters that each influence the parallel performance of the multigrid solver and that all have to be set at initialization: It has to be determined on

which grid levels the number of participating processes is reduced (and therefore how often this occurs), what the actual number of processes that are then used is, and of how many grid levels the multigrid hierarchy is supposed to consist of. The latter matters for the parallel case because using one or two grids less than possible can actually improve the runtime of a V-cycle if the saved communication time surpasses a certain limit.

The optimal configuration depends on the hardware, i. e., how the communication speed (influenced primarily by network latency and bandwidth) compares to computational power (dependent on CPU speed, cache size, memory bandwidth), but also on more variable parameters, like the number of cores available or the size of the finest grid. Therefore, the setup has to be reevaluated every time any of the aforementioned factors changes, even if the program is restarted on a different number of cores.

Since the relationship between hardware and program performance is not a seizable and quantifiable property, actual runtime measurements need to be used to determine an optimal configuration of how many processes are involved on each grid level and how many grids are used. A sophisticated benchmark module was written based on the following considerations:
Ignoring the point that a grid of a certain size is unsuited to be processed by too many processes, the total number of possible ways to distribute N_p processes over N_g grids in a way so that the finest grid is assigned the total number of processes and every coarse grid in the hierarchy is assigned not more than its affiliated fine grid is given by

$$\sum_{i_{(N_\mathrm{g}\text{-}2)}=1}^{N_\mathrm{p}} \cdots \sum_{i_2=1}^{i_3} \sum_{i_1=1}^{i_2} i_1 = \frac{1}{(N_\mathrm{g}-1)!} \prod_{i=1}^{N_\mathrm{g}-1} (N_\mathrm{p}+i-1) \tag{4.20}$$

for $N_\mathrm{g} > 2$, which approaches $N_\mathrm{p}^{N_\mathrm{g}-1}$ for large N_p. For example, 1000 processes handling a 5-grid hierarchy can be distributed in approximately $4 \cdot 10^{10}$ ways, which illustrates that not every possible configuration can be tested for performance.
Instead, a number of restrictions must be applied to vastly reduce the number of configurations to be tested by skillfully excluding unfavorable configurations. This primarily involves limiting the number of processes per grid level on coarse grids N_p^ℓ to cubic numbers (of the form $N_\mathrm{p}^\ell = N^3$) and cubic-like numbers (of the form $N_\mathrm{p}^\ell = N^2 \cdot (N-1)$), which then for the mostly cubical simulation domains enforces the use of local sub-grids with favorable ratio of surface to volume (and therefore communication to calculation). Other exclusion criteria that are implemented in the benchmark deal with the size of the process-local grids. I. e., configurations enabling local grids on any grid level to fall below or above a certain size limit are rejected for testing, as they deviate too far from the optimal relationship between communication and calculation. Similarly, the transition from one set of processes handling a fine grid to another set handling the next coarser grid is only allowed for significantly differently sized sets of processes.

To further increase the benchmark's efficiency, the remaining configurations are not tested individually. Since the time for on-level operations (smoothing, calculating the residual and solving on the coarsest grid) and transfer operations (restriction and prolongation) can be expected to vary only with the number of involved processes on the respective grids, redundant measurements can be minimized by identifying a subset of configurations which contains every viable combination of process numbers for fine-to-coarse transitions, and subsequently using the results of corresponding partial time measurements to estimate V-cycle runtimes for all remaining configurations.

Hence the most time-consuming parts of the benchmark, initializing and running different variants of coarse grid agglomeration, are reduced to a convenient minimum.

The resultant benchmark can then be conveniently called as part of the program's initialization. It's outcome, however, is not absolute in that numerous configurations can be comparably fast and fluctuations due to peaks in the network load or activation of random system processes can cause a decisive lag.

4.5 Practical Limitations

The multigrid solver described here is designed for solving the Poisson equation on cuboidal grids that are similarly sized in each direction of space. Large deviations of this assumption not only affect the benchmark as described above, but also lead to a premature stop of the coarsening process, as the smallest practicable grid size is achieved in one or two, but not all, directions. An extreme case here would be the simulation of a long, thin, (cylindrical) plasma vessel, where the vessel's interior can only be resolved on a very small number of coarse grids. In such a case, and if performance is an issue, a more sophisticated coarsening approach involving directional coarsening or semi coarsening (leading to individual mesh sizes h_x, h_y and h_z) needs to be implemented.

On another note, the applied approach to handle arbitrarily shaped irregular boundaries does not allow for the full multigrid method to be used effectively. As described in subsection 3.2.2, this variant first projects the right-handed side vector $\boldsymbol{b}$ of the system $A\,\boldsymbol{u} = \boldsymbol{b}$ onto the coarsest grid and improves the solution obtained there on increasingly finer grids. This is merely possible in the presence of boundary points because the restriction process (algebraic weighting of up to 27 fine-grid values) is then impeded by the values of the boundaries being zero.
In contrast, restricting the residual $\boldsymbol{r}$ near boundaries is unproblematic, as the respective values for the residual can be expected to be small after the smoothing process and the residuals of boundary points are always zero.
Therefore, the full multigrid method (with optimal algorithmic scaling) can't be used to its full potential. Instead, V-cycles are used iteratively and the solution of the respective previous time step of the PIC simulation is utilized as the initial guess. If the electric potential of the simulated plasma doesn't fluctuate too strongly and the V-cycles are properly tuned regarding pre- and post-smoothing, the preferable optimal algorithmic behavior can be achieved nevertheless.

4.6 Integration into PlasmaPIC

The multigrid solver is designed to fit into the modular framework of PlamsaPIC. Upon initialization, it is passed the information about global and local grid size and the position of the respective thread in the partitioned domain, which is then used to allocate the required memory. The coefficients used by the various multigrid subroutines are subsequently calculated based on the boundary characteristics provided by PlasmaPIC. The user can furthermore provide a suitable configuration for the coarse grid agglomeration via the input card or let the benchmark described in subsection 4.4.3 find an "optimal" distribution of the involved threads to the coarse grids.

During the actual simulation, the multigrid module gets passed the updated charge distribution of every new time step and performs a sufficient amount of V-cycles, starting with the solution of the respective previous time step as the initial guess for the electrostatic potential. The number of V-cycles and pre- and post-smoothing steps is dynamically adjusted by using the convergence criterion

$$\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|} \leq r_{\text{tol}} \ , \tag{4.21}$$

with $\|\cdot\|$ being the Euclidean norm of the vectors $\boldsymbol{r}$ and $\boldsymbol{b}$ and r_{tol} being the relative convergence tolerance, arbitrarily specifiable in the input card. r_{tol} should typically be small, e. g., 10^{-5}, but a sufficient value is generally dependent on the problem.

Similarly, equation 2.13 is solved once in each RF cycle.

Chapter 5

Assessment of Capabilities and Performance

As discussed in section 4.1, there are some properties of the particle-in-cell method that possibly have a negative impact on the convergence behavior and parallel scaling of a multigrid solver. In order to thoroughly demonstrate the capabilities of the developed solver, performance tests are therefore applied to both generic problems with irregular boundaries and actual plasma simulations. The results are then compared to computations performed with the previously used SOR method and the PETSc software suite [62, 63, 64].

5.1 Solving Generic Elliptic PDEs

The convergence behavior of an iterative solver is usually investigated by calculating some norm (often the Euclidean norm $\|\cdot\|_2$) of the residual vector $\boldsymbol{r}^m = \boldsymbol{b} - \boldsymbol{u}^m$ after each iteration m. The residual reduction factor $\|\boldsymbol{r}^m\| \,/\, \left\|\boldsymbol{r}^{m-1}\right\|$ is then a measure for the quality of the solver.
Usage of this approach is justified by the general unavailability of a better alternative, i. e., the actual error $\boldsymbol{e}^m = \boldsymbol{u}^{\text{exact}} - \boldsymbol{u}^m$.

For the purpose of setting a reference, an "ideal" problem is first considered.

5.1.1 Textbook Case

The Poisson equation

$$\begin{aligned} \Delta u_1(\boldsymbol{r}) &= -f(x, y, z) \\ &= 5\,\sin(5\pi\,x) + \cos(2\pi\,y) + (z^2 - z - 0.5)\,\mathrm{e}^{-(z-0.5)^2} \end{aligned} \tag{5.1}$$

is solved for u_1 on a cubic grid representing the unit cube with Dirichlet domain boundaries and with no additional geometry to be considered. The fixed values for the boundary conditions are chosen such that an analytical solution for u_1 exists:

$$u_1(\boldsymbol{r}) = -\frac{1}{5\pi^2}\sin(5\pi\,x) - \frac{1}{4\pi^2}\cos(2\pi\,y) + \frac{1}{4}\,\mathrm{e}^{-(z-0.5)^2} \quad \forall\ \boldsymbol{r} \in \partial\Omega. \tag{5.2}$$

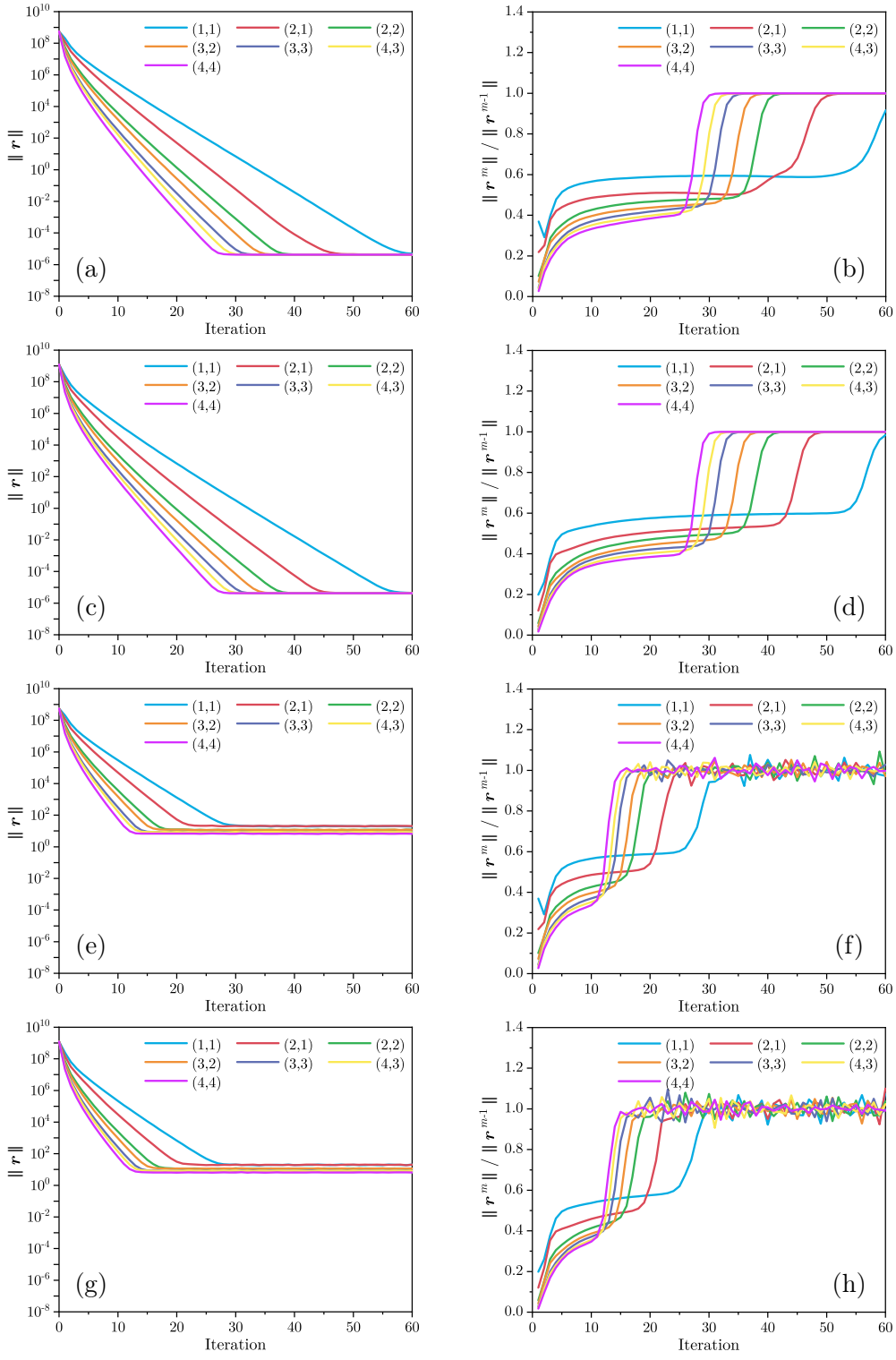

Figure 5.1: Textbook efficiency of an iterative multigrid solver over 60 V-cycles for the model Poisson problem (equation (5.1)) with varying number of pre- and post-smoothing steps (n_{pre},n_{post}) on a grid of size 1025^3. (a) The course of the total residual norm $\|\boldsymbol{r}\|_2$. (b) The course of the residual reduction factor $\|\boldsymbol{r}^m\|_2 / \left\|\boldsymbol{r}^{m-1}\right\|_2$. (c) and (d) Same as (a) and (b), but with a thin-walled hollow sphere as an additional irregular boundary interface embedded in the domain. (e) to (h) Same as (a) to (d), but with single-precision floating point communication. All calculations were performed on 192 processes.

For grid sizes where equation (4.3) can be applied sufficiently often (i. e., for those of the form $(2^n + 1)^3$, $(3 \cdot 2^n + 1)^3$ etc. for not too small integer values of n), this is a problem that can be treated by the textbook vertex-based multigrid method described in section 3.2 without further adjustments. Moreover, the multigrid field solver developed here reduces to a parallelized version of that method and can therefore be used to measure "textbook" efficiency.
Absolute values obtained this way are of little significance because they are problem-dependent. A qualitative assessment on the other hand is valid much more universally.

Figures 5.1(a) and (b) show the course of the residual norm and the residual reduction factor over 60 V-cycle iterations on 1025^3 grid points for calculations with varying number of smoothing steps, starting with the zero vector as initial guess. After the first few iterations, the residual drops at a relatively constant rate until a maximum accuracy (for calculations with double-precision floating-point arithmetic) is reached. The residual norm then approaches a constant value and the reduction factor increases to approximately one (within very small fluctuations). While the gradient of the seceding parts of the curves in figure 5.1(a) depends on the number of smoothing steps, the benefit of an additional step decreases the more smoothing steps are used. This corresponds to the observation that the smoothing process only effectively eliminates errors of specific frequency modes, while other modes remain relatively unaffected (cf. subsection 3.1.3).
The comparably poor performance over the first iterations can similarly be attributed to insufficient smoothing, as the V-cycle is not effective as long as the initial high-frequency errors are not properly reduced.
A very similar performance can be observed if an additional boundary object, "activating" the modifications based on the Shortley-Weller discretization scheme, is added to the grid. For a thin hollow sphere spanning the whole domain, this is shown in figures 5.1(c) and (d). With the exception of the very first iteration with a low number of smoothing steps, the residual reduction rate is generally (slightly) better than in the previous case (which can be attributed to the system size being effectively reduced). This demonstrates that the multigrid solver is capable of handling these kinds of problems without deterioration of convergence. Deactivating the respective features leads to the behavior illustrated in figure 5.2. Here, the resultant vectors of the iterative process diverge from the exact solution unless a tremendous amount of smoothing steps is applied.

This first model problem can also be used to investigate the influence of the radical measure of using single-precision floating-point numbers for inter-process communication. Figures 5.1(e) to (h) were created analogously to figures 5.1(a) to (d) with the only difference being the data type of the communication buffers. This change most obviously affects the maximum accuracy of the iterative process, which is significantly worse. Up until the point of stagnation, however, the convergence behavior is nearly identical to that of the prior case, which justifies the switch made in order to better utilize the limited bandwidth for communication, as long as the required accuracy of the obtained solution allows it.

Another relationship worth being considered is the one between residual $\boldsymbol{r}$ and error $\boldsymbol{e}$. Since an analytical solution to equation (5.1) is available, this can be done exemplary for the model problem. Figures 5.3(a) and (b) show the curves corresponding to figures 5.1(a) and (b). The error norm drops steadily until a maximum accuracy

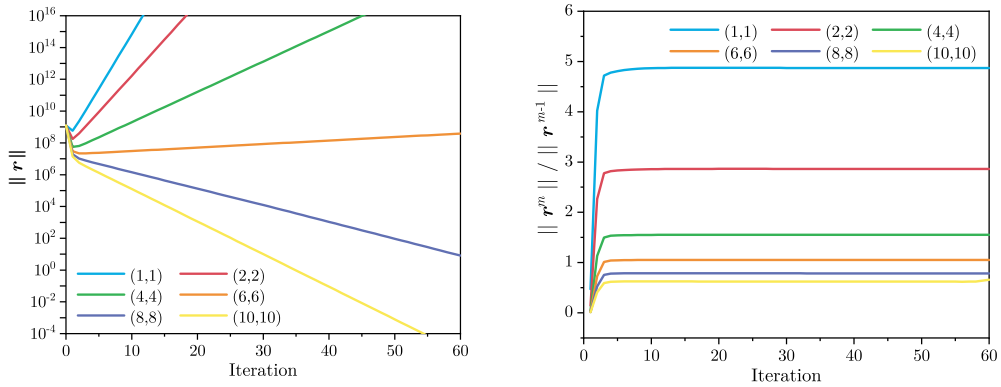

Figure 5.2: Behavior of the unmodified textbook geometric multigrid solver for a domain with irregular boundaries. The problem is the same as the one used for figure 5.1(c) and (d) and the curves were generated analogously.

is reached. A striking difference to the residual norm's behavior is that this minimal value is reached after far fewer iterations, which indicates that the maximum possible accuracy, limited by the so-called discretization error resulting from discretizing a continuous problem, is reached before the effects of the limited machine precision start to manifest. This relation is not universally valid. For a much further refined grid, the discretization error can be expected to be smaller than the one introduced by finite-precision arithmetic.
The residual norm on the other hand decreases independently of the discretization error as the iterative process converges to a solution that is slightly off the analytically exact solution $\boldsymbol{u}^{\text{exact}}$.
The respective plots for communication with single-precision floating-point numbers (figure 5.3(c) and (d)) are almost congruent to their double-precision counterpart, except for minor fluctuations. The maximum accuracy here is therefore still better than the limit caused by the discretization process, which further justifies the use of low-precision communication.

5.1.2 Convergence

Although Poisson's equation is a fundamental form for elliptic partial differential equations, applicability and convergence behavior of a solution method don't necessarily conform universally, especially in three dimensions. For this reason, a number of other elliptic PDEs are considered and analyzed analogously:

$$\Delta u_2 + c \cdot u_2 = -f(x,y,z) \tag{5.3}$$

$$\Delta u_3 + 2\,\frac{\partial u_3}{\partial x} + \frac{\partial u_3}{\partial y} = -f(x,y,z) \tag{5.4}$$

$$-(x+2z)\,\Delta u_4 - \mathrm{e}^{-(x^2+y^2)}\frac{\partial u_4}{\partial x} + \ln z\,\frac{\partial u_4}{\partial y} + 2x \cdot u_4 = -f(x,y,z) \tag{5.5}$$

$$\mathrm{e}^x\,\frac{\partial^2 u_5}{\partial x^2} + (y^3 + \frac{1}{2})\,\frac{\partial^2 u_5}{\partial y^2} + 2\cos(z)\,\frac{\partial^2 u_5}{\partial z^2} - x\frac{\partial u_5}{\partial z} = -f(x,y,z)\ . \tag{5.6}$$

Since the left handed side for each of these PDEs is altered compared to equation (5.1) and the error norm is not further considered, an analytical solution is neither available

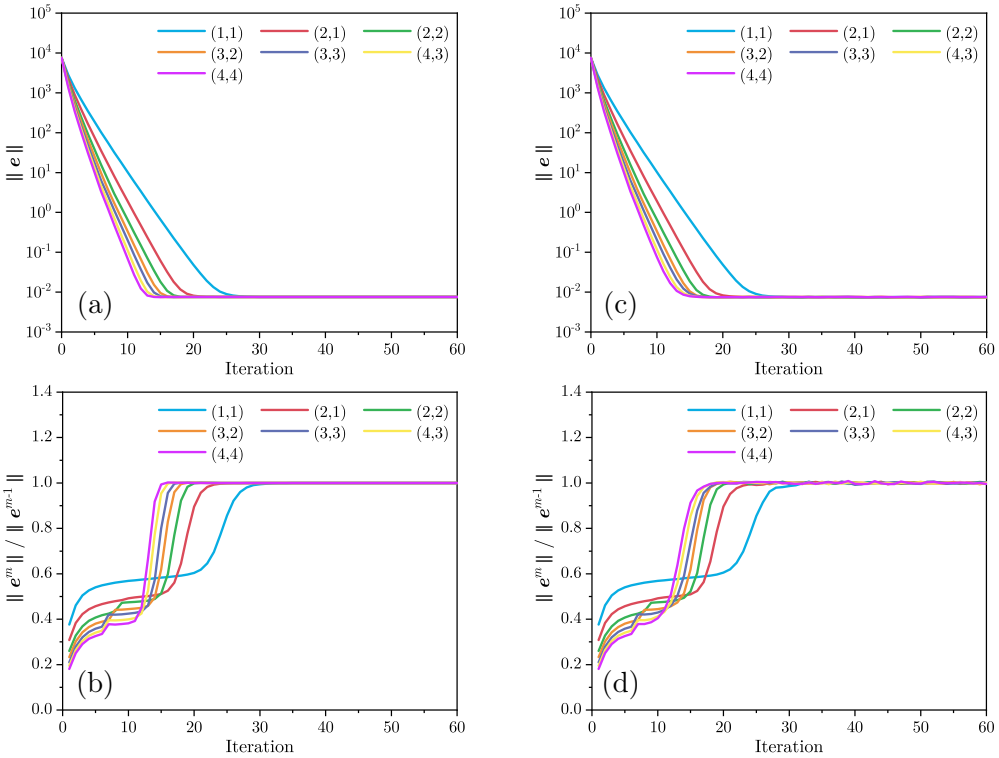

Figure 5.3: Behavior of the error norm for the model problem. (a) Course of $\|\boldsymbol{e}\|_2$ over 60 iterations. (b) Course of the error reduction $\|\boldsymbol{e}^m\|_2 / \|\boldsymbol{e}^{m-1}\|_2$. (c), (d) Same, but with single-precision floating-point numbers used for communication.

nor required. The boundary conditions can therefore be assigned an arbitrary value, e. g., $u(\boldsymbol{r}) = 0 \; \forall \; \boldsymbol{r} \in \partial\Omega$.
Each PDE is solved both on a cubic domain and on an embedded irregular domain of individual shape. Graphical representations of those shapes are given in figure 5.4. For simplification, only the $(2, 2)$-smoothing case is considered.
In figure 5.5(a), the residual norm of the approximate solution for u_2 (the Helmholtz equation) is plotted against the iteration number for various values of c, spanning over multiple magnitudes. It's notable that the maximal accuracy is reached after increasingly fewer iterations the less the Laplace operator influences the solution, i. e., the larger the constant c is. For $c = 0$, equation (5.3) is identical to the Poisson equation. Introducing irregular boundary conditions furthermore only vaguely alters the outcome of this approach, as can be seen in figure 5.5(b).

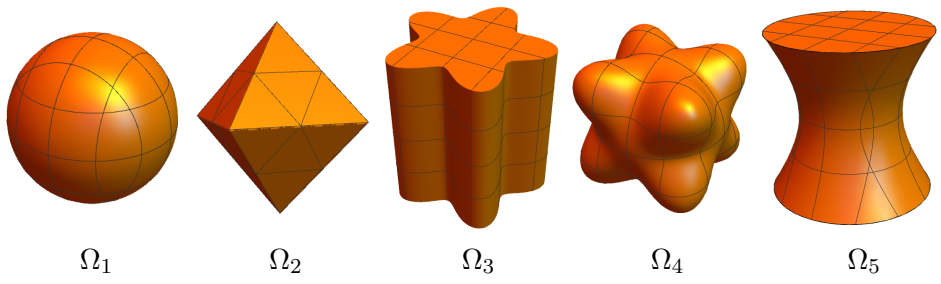

Figure 5.4: Shapes of the domains used for the model elliptic PDEs

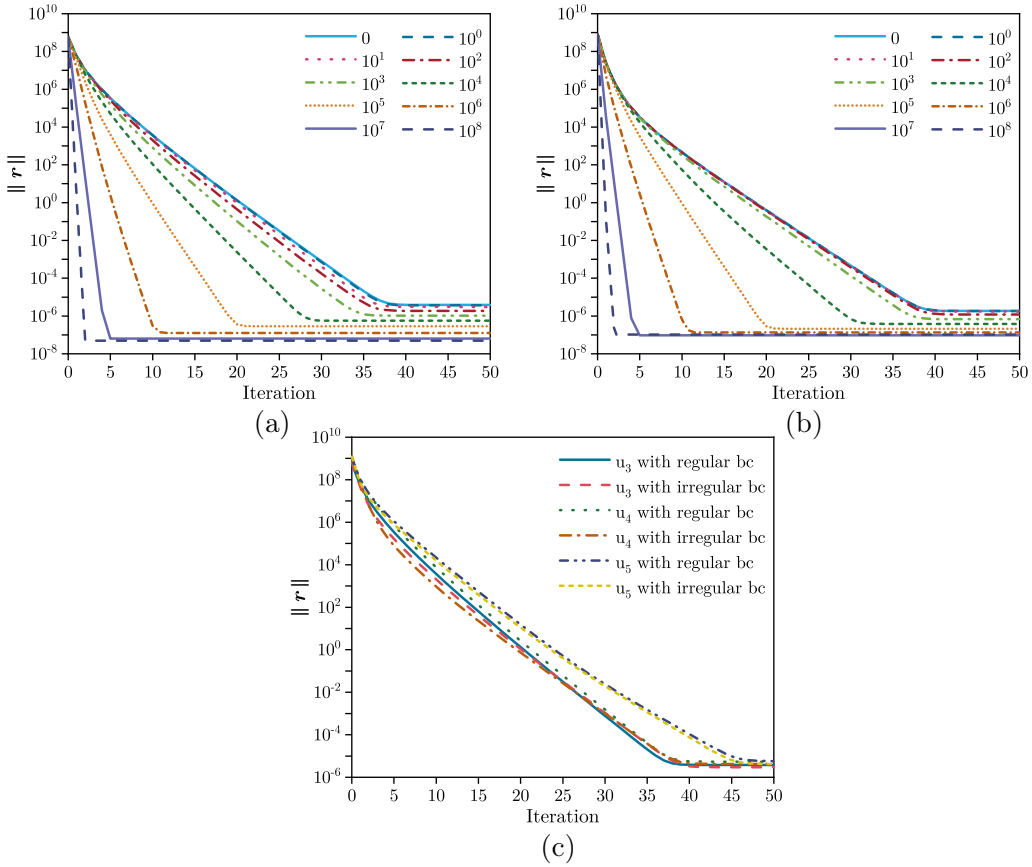

Figure 5.5: Decrease of the residual of the four additional elliptic partial differential equations. (a) The iterative development of the residual of u_2 for varying values of the constant c in a cubic domain of size 1025^3 (50 iterations, (2,2)-smoothing). (b) Equivalent to (a), but the effective domain is reduced to smaller cube that is wedged inside the original domain (cf. figure 5.4). (c) Course of the residuals of u_3, u_4 and u_5 with and without their respective irregularly shaped boundary interfaces.

Moreover, this observation holds true for u_3, u_4 and u_5, as shown in figure 5.5(c). All used PDEs converge similarly using V-cycles with (2,2)-smoothing, relatively independent of the shape of the boundary interface.

5.1.3 Scalability

Since the overall residual norm depends on the system size, it doesn't make sense to analyze the scaling behavior of how its absolute value decreases over multiple V-cycle iterations. Its relative reduction over several iterations on the other hand can be compared and analyzed over an arbitrary range of system sizes.
Since the increasing system size allows for more and more grid levels to be used, an additional grid is introduced whenever coarsening the coarsest grid yields a system of at least $3 \times 3 \times 3$ grid points (with the outermost grid points being defined as Dirichlet boundary points, this implies a minimal system size of 5^3).
For each of the elliptic PDEs (5.1) and (5.3) to (5.6), the ratio of the residual norm $\|\boldsymbol{r}^m\|$ after $m = 1$, $m = 2$, $m = 5$, and $m = 10$ iterations to the initial norm $\left\|\boldsymbol{r}^0\right\|$ is

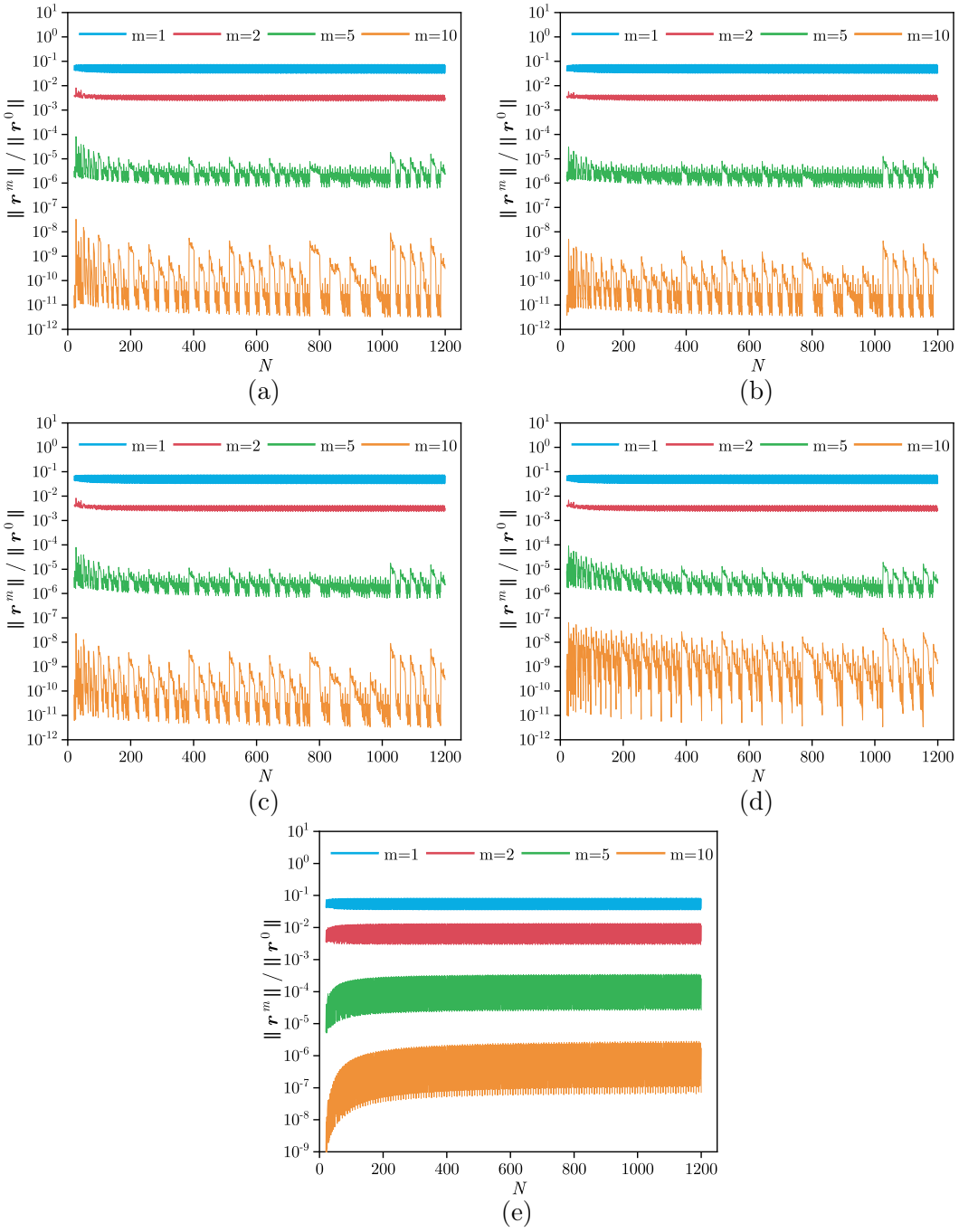

Figure 5.6: Scaling of the residual reduction rate for the solution of the five model PDEs. (a) Equation (5.1). (b) Equation (5.3). (c) Equation (5.4). (d) Equation (5.5). (e) Equation (5.6). For each, the ratios $\|\boldsymbol{r}^1\| / \|\boldsymbol{r}^0\|$, $\|\boldsymbol{r}^2\| / \|\boldsymbol{r}^0\|$, $\|\boldsymbol{r}^5\| / \|\boldsymbol{r}^0\|$ and $\|\boldsymbol{r}^{10}\| / \|\boldsymbol{r}^0\|$ are plotted against the number of grid points per spatial direction N. The total system size is $n = N^3$.

calculated for systems as small as 20^3 up to comparably huge systems of with 1200^3 grid points. This choice of size range is based on applicability of the multigrid method utilizing multiple grid levels and usage of a reasonable amount of computational resources and time.
The five residual ratios are then plotted against the number of grid points per spatial direction (figure 5.6).

Several observations can be made for every PDE:
While an averaged curve, obtained by calculating mean values over larger intervals of N, would produce a remarkably constant course of the residual norm ratios, the actual curve, incorporating every grid size, shows strong fluctuations, specifically between neighboring data points, where one may lie below the local average and the next above. The relative magnitude of these fluctuations is furthermore enhanced by subsequent iterations, so that they can be examined best using the $\left\|\boldsymbol{r}^{10}\right\| / \left\|\boldsymbol{r}^{0}\right\|$ ratio. In this case, and for the PDEs investigated here, sudden jumps over four orders of magnitude can be observed.

However, a closer examination over a shorter range of N reveals the residual norm ratio to actually be subject to several concurrent quasi-oscillations with distinguished periodicity. The top half of figure 5.7 depicts a cutout of the $\left\|\boldsymbol{r}^{10}\right\| / \left\|\boldsymbol{r}^{0}\right\|$ curve of figure 5.6(a) in appropriate detail. At 26, 50, and 98 grid points per spatial direction respectively, an additional grid is introduced, which is indicated by the three vertical lines. With the exception of u_5, which generally shows a reduced convergence rate and pure alternating jumps between "good" and "bad" residual reduction, this holds true for the other PDEs as well.
While the lines' positions on the x-axis coincide with local maximums of the norm ratio curve, this is not necessarily a correlation, since other maximums of strong manifestation exist (although the subsequent maximums are smaller) and suppressing the introduction of a new grid only changes the height of the maximum (not shown).

Rather, the deviation from a relatively constant minimal value is related to the abundance and the values of the irregular distances d_{irr}^{ℓ} between the two outermost grid points in every direction on the coarse grids, as introduced in chapter 4. The bottom half of figure 5.7 shows the values of $d_{\text{irr}}^{\ell}(N)/h_\ell$ for the systems examined in the top half. While the values jump between 1 and 0.75 on grid level $\ell = 1$, they increase linearly on the coarser grids until the value 1 is reached, after which the number of grid points increases by one and the irregular distance is reset to the minimal value.
Whenever all grids align perfectly so that no irregularities are necessary (which is the case for systems with $4 \cdot 2^{N_\text{g}-1} + 1$, $5 \cdot 2^{N_\text{g}-1} + 1$, and $6 \cdot 2^{N_\text{g}-1} + 1$ grid points per dimension under these prerequisites), a local minimum is reached. Another minimum can be found directly inbetween those, where only the coarsest grid doesn't align with the other grids.
The highest local maximums on the other hand can be found right after the minimums, coinciding with the minimal values of d_{irr}^{ℓ} on every grid. In these cases, cell-based coarsening is used throughout all grids, which indicates that although the two variants are not applicable to the same systems, vertex-based coarsening can be considered superior.

These observations imply that the mixed-coarsening approach described in subsection 4.2.1 thus fails to completely eliminate this consequence of arbitrary grid sizes. However, since the fluctuations of the (absolute) residual norm only surpass one order of magnitude after numerous iterations, this can be considered as acceptable.

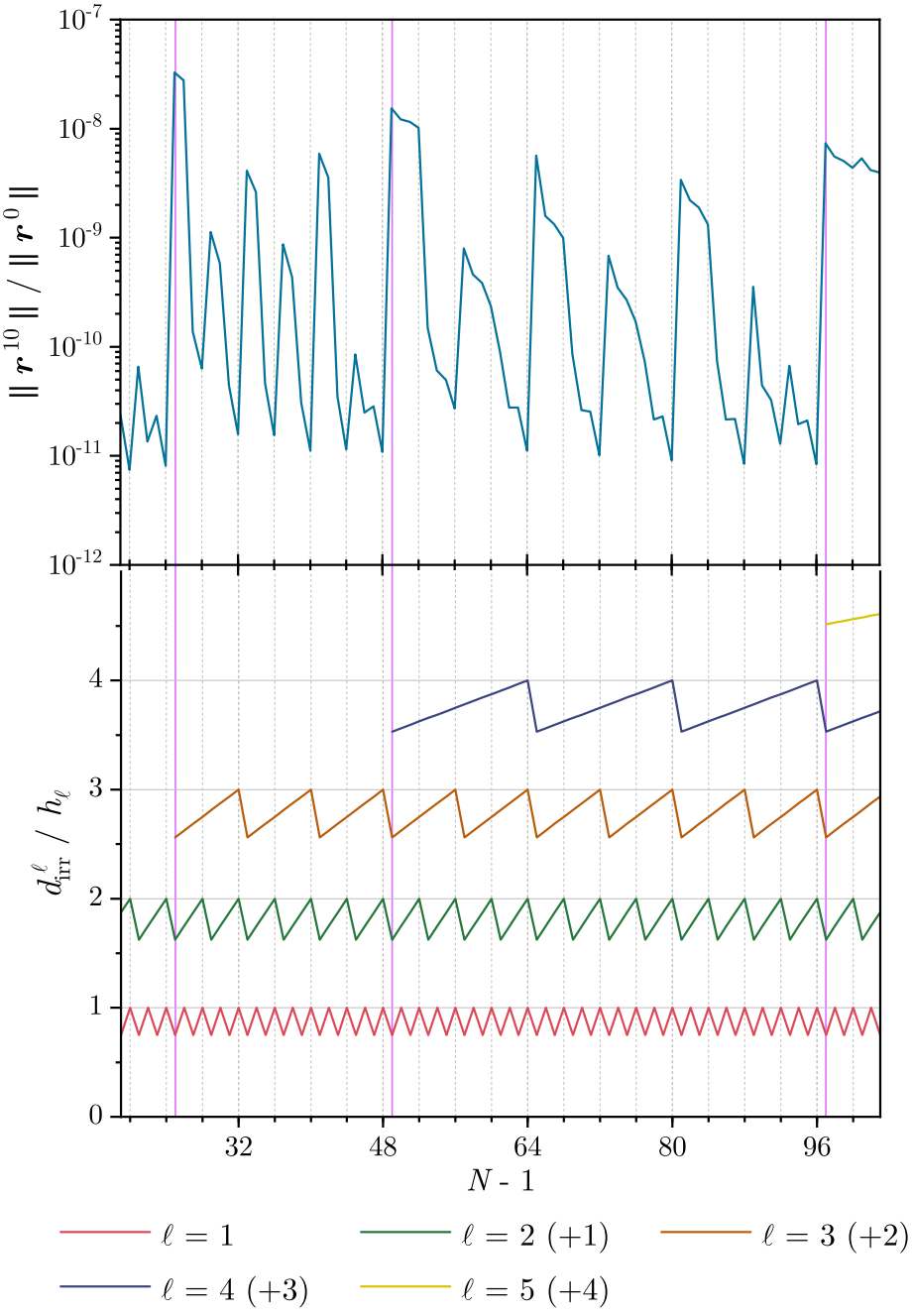

Figure 5.7: Top: Detailed view of the $\|\boldsymbol{r}^{10}\| \,/\, \|\boldsymbol{r}^{0}\|$ curve of figure 5.6(a). Up to $N = 26$ grid points per spatial direction, a total of three grid levels are used. At $N = 26,\ 50,\ 98$, an additional grid is introduced respectively.
Bottom: Comparative plot of $d^{\ell}_{\mathrm{irr}}(N)/h_{\ell}$ for the same range of grid sizes. The lines for the various grid levels ℓ are stacked by a constant offset of one. However, the maximal value for each line is one.
For both graphs, the x-axis is offset by one so that the vertical grid lines coincide with values divisible by four.

Apart from these effects, the multigrid solver shows remarkably constant reduction rates over all investigated system sizes, that are furthermore easily one order of magnitude per iteration (with the exception of u_5, for which it is still acceptable), which is generally considered to be textbook efficiency of a multigrid method.
Combined with the availability of a sufficiently accurate initial guess, this characterizes the solver to reach a desired accuracy within a number of iterations independent on the system size, which is a prerequisite to scale optimally (cf. section 3.6). Since a V-cycle's computational costs are directly proportional to the system size (in the serial case), it's therefore foremost the matter of efficient parallelization that determines the performance in the practical case (i. e., on a parallel computer).
In order to combine such an evaluation with the application within a PIC simulation, it is postponed to the following section.

5.2 Performance within PlasmaPIC

Domain partitioning methods for solving partial differential equations in parallel can never scale perfectly for every system. Rather, the speedup reached with a certain number of processor cores always depends on the system size. A very small system might be solved the fastest on a single core, i. e., the speedup of adding another core could be smaller than one. For a larger system on the other hand, additional cores should generally improve runtime, until a threshold is reached, at which communication costs surpass computation (assuming non-blocking communication) and the speedup deteriorates. It is therefore desirable to find and utilize a "sweet spot", at which an optimal number of cores works on a given system size.
This approach at investigating the dependency of the performance on the number of cores is commonly referred to as *strong scaling*.
While this is a suitable method to optimize the utilization of resources to reduce runtime for a given system, its conclusions have only vague meaning in respect of performance for systems of other size. Varying the system size for a set number of cores is of minor interest because it's clear that the effects of parallelizing the problem diminish as the ratio of computation to communication shifts favorable and the complexity of the underlying algorithm becomes more influential.
Instead, the so-called *weak scaling* has more significance. Here, both processor count and total system size are scaled up simultaneously, such that the system size per core remains constant. In this context, a perfectly parallelized algorithm that scales linearly with system size finishes computation in constant runtime, independent of the respective system. Such a behavior, although being a theoretical ideal, is the ultimately desired outcome for the multigrid solver developed in the context of this thesis. While both strong and weak scaling of the solver, among other characteristics, are investigated in the following subsections, the latter therefore represents the solver's central requirements the best.

5.2.1 Influence of Network Speed

The computations for this thesis were primarily performed on a cluster that utilizes the *InfiniBand* technology for fast inter-process communication. Although generally preferable to slower alternatives like simple Ethernet connections, this is an especially crucial

part of the hardware for the multigrid solver. Considering that a parallel multigrid method necessarily either deviates from the optimal ratio of (ideally hidden) communication to computation or sends the data of whole grids between intersecting sets of processes (both of which is utilized in the solver described here and an "optimal" configuration can be found with the benchmark), any measurements of parallel scaling have to be considered in the context of the specific hardware that was used.

In order to assess how different hardware configurations influence performance, the runtime of a single V-cycle with two pre- and two post-smoothing steps was measured on four different setups, of which three only vary in the CPU type and the associated number of cores per node:

HPC	CPU	Cores/node	Network
LOEWE-CSC	AMD "Magny-Cours" Opteron 6172	24	InfiniBand
LOEWE-CSC	Intel Xeon Ivy Bridge E5-2670 v2	20	InfiniBand
LOEWE-CSC	Intel Xeon Broadwell E5-2640 v4	20	InfiniBand
Yacana	Intel Xeon Ivy Bridge E5-2630 v2	12	Ethernet

Table 5.1: Comparison of the various cluster setups used for this thesis

For each, the runtime of a V-cycle during a PlasmaPIC simulation was measured on multiples of 24 processes on two different systems, namely the RIT-1.0 ($100 \times 100 \times 97$ grid points) and the RIT-2.5 ($233 \times 233 \times 220$ grid points). A preceding run of the benchmark module was furthermore used to determine a configuration of coarse grid agglomeration.
Additionally, the time spent on every individual grid and on the transfer operations between the grids was measured separately to provide further insight into what share of the total time is spent where and how this distribution develops if the number of processes is increased.

The left columns of diagrams in figures 5.8 (RIT-1.0) and 5.9 (RIT-2.5) show these measurements as stacked column graphs. On the right, the coarse grid agglomeration is visualized by associating the grid levels with the respective number of cores used there. All configurations have in common that the finest and the second finest grid are processed by the full number of cores used in the respective case and that the solution of the coarsest grid system is obtained by a single core (illustrating the effectiveness of the measures described in section 4.3). The total number of grids, however, fluctuates between six and seven for the RIT-1.0 and seems to be explicitly dependent on the network connection for the RIT-2.5, indicating that there are various agglomeration configurations with similar performance and that it's computationally cheaper to use less grids and solve on a larger coarsest grid if the network speed is rather slow.
For the Ivy Bridge/Ethernet configuration, the V-cycle runtime actually worsens if more than 24 processors (two nodes) are used on the RIT-1.0 system. The extra time needed by higher processor counts is mostly (but not exclusively) spent on the coarser grids, where communication is more expensive in relative terms.
An apparent difference between the calculations on the two systems is that for this number range, the benefit of adding more cores quickly diminishes with the RIT-1.0, whereas runtime keeps decreasing with the RIT-2.5, where the local subdomains are larger. This is a clear indication that for simulations of the RIT-1.0, the network reaches its limits and communication dominates runtime. The fact that overall runtime for a V-cycle

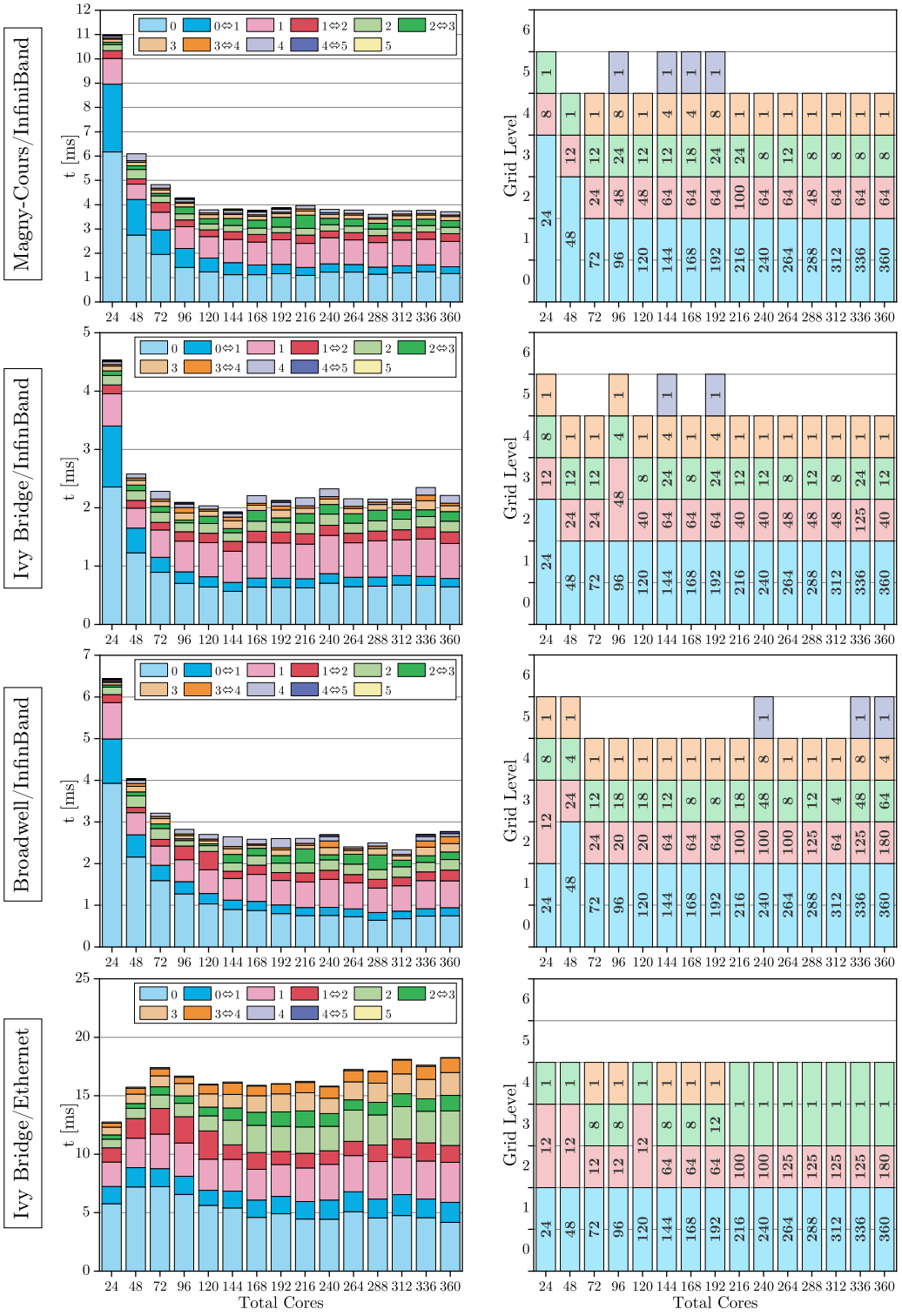

Figure 5.8: Analysis of the runtime of a single V-cycle for the RIT-1.0 system. Left side: Time spent on the different grid levels and respective transfer operations for various numbers of processors, measured on four different hardware configurations. Right side: The respective associated distribution of how many processors are used for the calculations on each grid level.

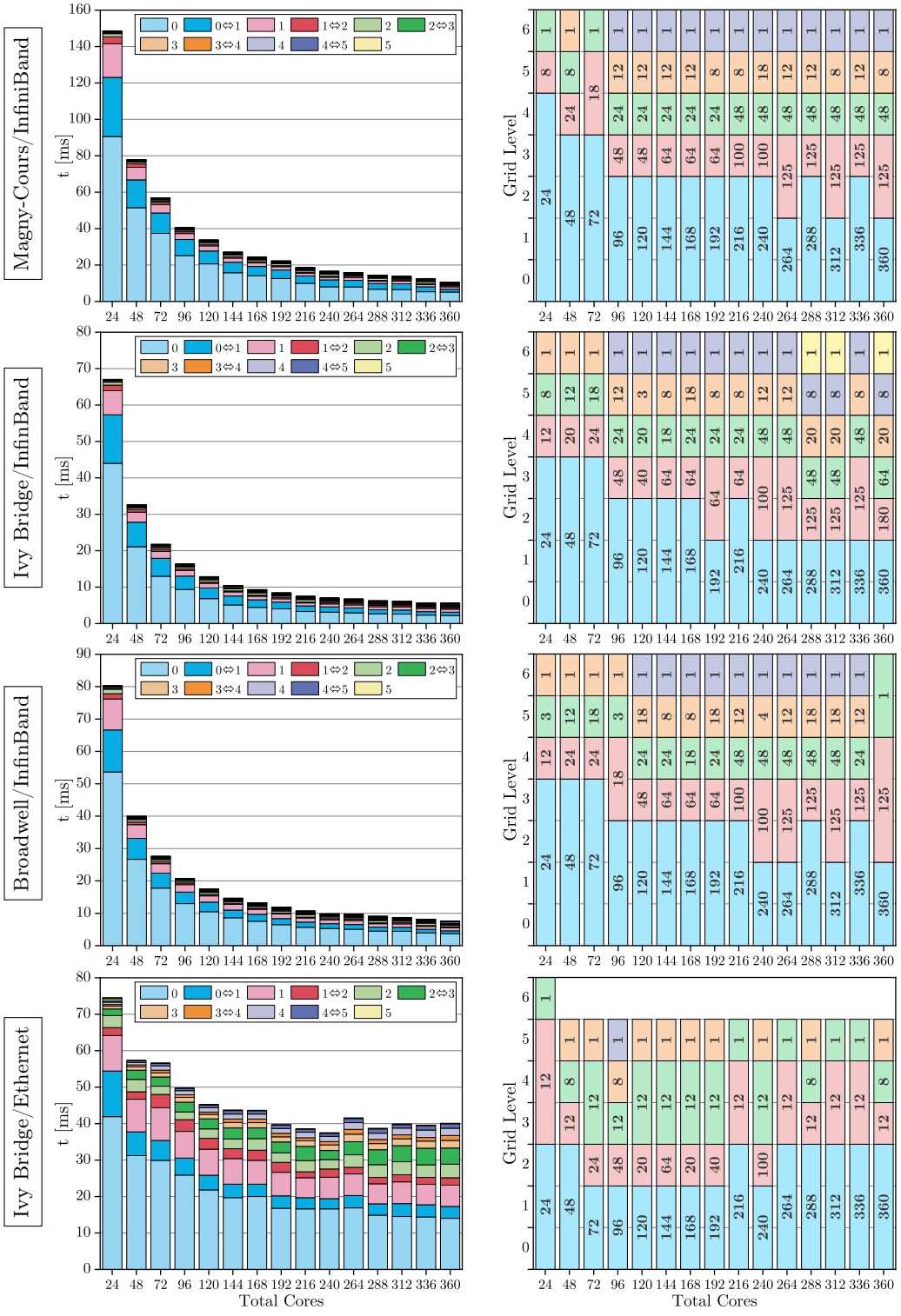

Figure 5.9: Analysis of the runtime of a single V-cycle for the RIT-2.5 system. The graphs were created analogous to figure 5.8. The RIT-2.5 system is approximately 12.3 times larger than the RIT-1.0 system.

further remains relatively constant and that the time spent on the second finest grid quickly approaches that on the finest points towards latency being the limiting factor. Overall, the finest grid isn't processed much faster than in one millisecond for either hardware configuration.

By increasing the number of processors, the size of the individual subdomains shrinks (the volume is inversely proportional to the number of sub-cells). While the surface area of a subdomain doesn't decrease proportionally to its volume, it decreases nevertheless and a sole dependency on the bandwidth per processor would result in further reduced runtimes. A significant latency on the other hand sets a lower limit for communication time that is relatively independent of the message size.
Although the InfiniBand network offers very low latencies of principally under one microsecond [8], various factors can introduce an additional overhead. These may include the numerous memory accesses to prepare buffers and to initiate (non-blocking) communication, the protocol overheads of the MPI implementation and further difficulties in the realization of bidirectional communication of every process with (up to) six neighboring processes. In fact, since of those six neighbors, most are usually located on different nodes, the number of outgoing messages from one node is quite high, surpassing the number that fits into the queue pair cache of the InfiniBand connects [65]. The resulting cache misses cause a significant overhead [66].

Figure 5.10 shows the result of three simple network performance tests. In the case that only two processes communicate with each other, they can utilize the network bandwidth very efficiently and large data transfer rates are possible if an appropriate message size is chosen. Small messages on the other hand need at least $1.7 \cdot 10^{-5}$ s to be delivered.
If all processes of a node send and receive messages with a partner-process, the transmission time varies widely for short messages and the maximum data throughput is significantly smaller than for the prior case. While the latter is simply a consequence of more processes sharing the same bandwidth, the former has a direct impact on the field solver's performance. Although the minimum transmission time is very similar to the case of only two processors communicating, the actual program runtime is primarily affected by the maximum time. Particularly if communication is slow at one point in the network even for only a short period of time, this affects all other processes, as they eventually have to wait to receive updated values for their own calculations. This indirect synchronization of all processes without a special rendezvous point is an unavoidable necessity of parallel computation.
The minimum transmission time further increases as a stepwise constant function where the width of the constant intervals is exactly 64 kB. This indicates that messages larger than a *maximum transfer unit* (MTU) are partitioned by the MPI implementation and then sent piecewise [67]. The curve then aligns more and more with that of the average transmission time.
If each process communicates with more than one partner-process (four in the case being), the curve for the minimum transmission time becomes smoother and the time for one transmission step further increases. Small messages then require up to 10^{-4} s. Since on each grid level, smoothing and calculating the residual require a total of nine communication steps for the V-cycle configuration used above (two, for red and black points separately, for each smoothing step and one to calculate the residual values), this provides an explanation for the lower limit of the runtime of the V-cycle on the finest grids of approximately 1 ms.

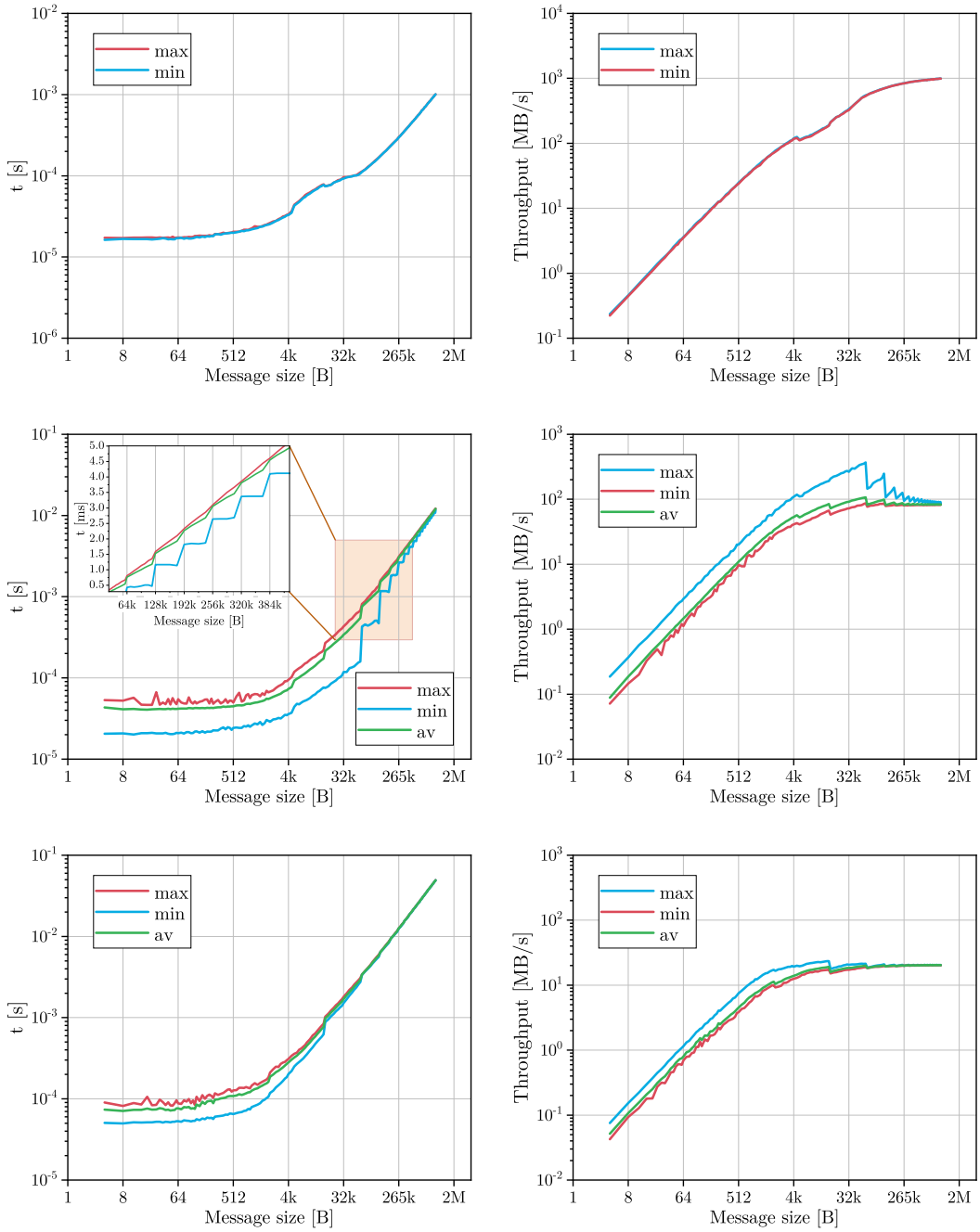

Figure 5.10: A simple test of the network performance for bidirectional communication. Top: One processor sends and receives a message to and from a processor on a different node. The time to complete this process is plotted against the message size in the left graph. In the right graph, the associated data throughput is plotted against the message size. Middle: All 24 processes of one node send and receive messages to and from a distinct partner-process on another node. The left diagram further contains a zoomed inset with linear axis scaling. Bottom: All 24 processes of one node communicate with four processes on another node, respectively, for a total of 96 bidirectional communication channels (graphs analogous).
Note that the total number of simultaneously sent messages differs for the three cases.

It is further noteworthy that the measurements of the runtime of a single V-cycle do not reflect the pure strong scaling of the multigrid solver. Firstly because a V-cycle represents only one iteration of the solution process, which stops only after meeting the convergence criterion, and secondly because gradually increasing the number of processes by 24 consequentially changes the aspect ratios of the individual subdomains, which affects the ratio of data to be exchanged and data to be processed locally.

5.2.2 Weak Scaling

Investigating weak scaling has relevance mostly for parallelized algorithms that ideally have $\mathcal{O}(n)$ scaling. If the time to solution then shows to deviate from constant although the system size per processor is fixed, the algorithm is either not scaling linearly or communication costs somehow increase. Both can be the case for PlasmaPIC's multigrid solver, as a larger plasma discharge may have slightly different properties (i. e., the electric potential may change more from one time step to the next, which influences the quality of the initial guess for the first V-cycle) and the multigrid approach with coarse grid agglomeration isn't limited to a constant message size between neighboring processes.
Unfortunately, the hardware configuration can affect the outcome, too. If all processes are located on the same computational node (i. e., on the same motherboard), interprocess messages don't need to be sent over the cluster network and a potential communication overhead is smaller. Similarly, if the processes are distributed over only two nodes, a comparatively large number of neighboring processes don't need to use the network to exchange information and the remaining send/receive operations can utilize the network bandwidth more efficiently than if more nodes are used.

In order to best measure the scaling behavior of the multigrid solver, simulating actual RITs is furthermore not a suitable approach. First, not every characteristic length of a RIT component grows proportionally in size if the diameter of the discharge chamber is increased, e. g., the wall thickness. Second, larger thrusters typically have more (and especially not larger) grid apertures, whose total number, however, doesn't scale directly. Third, the general physical conditions required for stable operation, e. g., neutral gas pressure and coil current, vary with the thruster's size and are likely to have an impact on the field solver's task. Keeping them constant for the sake of comparability on the other hand wouldn't result in meaningful simulations anyway.
In order to best capture the true scaling behavior of the multigrid solver (and, incidentally, the other PIC modules), a hollow cubic box, serving as plasma discharge chamber, is the better choice. For the setup used here, the walls have a fixed electric potential and a coil is wrapped around the whole domain.
In addition to keeping the system size per processor constant, the shape of the individual subdomains is preserved as well to avoid a shift in the ratio of communication to computation. By consistently using only cubic sections of the equally cubic simulation domain, the number of processes used to obtain a weak scaling data point is then limited to cube numbers so that $3 \times 3 \times 3$, $4 \times 4 \times 4$, etc. subdomains are used.
The only remaining degree of freedom then is the system size per processor.

A disadvantage of weak scaling is that a communication overhead may be a constant addition to the total runtime and therefore not apparent. Optimizations with the goal of eliminating or at least reducing it may then not appear necessary even if they are eligible.

For this, strong scaling generally provides better insight but it is possible to compare the weak scaling curves of measurements with varying system size per processor.

Following the above explanations, weak scaling measurements were performed with 3^3, 4^3, ..., 15^3 processes on systems with the suitable size to provide exactly 15^3, 20^3, 25^3, 30^3, 35^3, and 40^3 grid points per processor. For each combination, the runtime of the field solver module was measured over $10,000$ time steps of the simulation and then converted to the runtime of a single time step. The corresponding average time for the various particle operations, involving loops over all particles in the respective subdomain, additionally serves as a reference value.
All computations were performed on nodes with AMD Magny-Cours CPUs, which allowed for the most cores to be available simultaneously to a single user on LOEWE-CSC at the time of the measurements.
Coincidentally, this CPU type is relatively slow compared to the Intel CPUs that are available alternatively. Combined with the InfiniBand network, this minimizes the ratio of communication (due to high bandwidth) to computation (due to slow CPUs) among the available hardware configurations and therefore offers a "best case scenario" where communication bandwidth only plays a relatively minor (but non-negligible) role. However, while weak scaling (and especially strong scaling) on a faster CPU may be worse due to communication becoming a performance bottleneck more quickly, it is important to point out that the performance for every individual measurement won't deteriorate.

Figure 5.11 shows the results of these measurements with individual graphs for each curve (in ascending order by system size per processor).
A first and obvious observation is that runtime is not constant but increases towards larger systems/more cores. While this is not the pursued outcome, various properties of the displayed curves relativize this deviation from the ideal at least to some degree.
For all curves, including those associated to the particle operations, a cutoff towards the smaller systems that are computed by relatively low numbers of processors can be observed. This is an indication of increased efficiency regarding the utilization of the network bandwidth (with more neighboring subdomains processed on the same node). Neglecting this effect still leaves a general increase in runtime for all curves, which demonstrates that even the particle operations don't scale perfectly. This can partially be explained by the increasing characteristic length of the plasma vessel, due to which the mean free path of the particles increases as well. Since the relative share of particles undergoing particle-wall interactions decreases, the total particle number increases disproportionately.
However, the possibility that communication additionally causes an increasing overhead toward the largest systems can't be ruled out.

Within each curve, the gradient increases furthermore consistently after the 6^{th} to 7^{th} data point (more apparent for the field solver curves). Moreover, at that point, a short interval of relatively constant runtime ends and the general trend towards linear increase starts.
Since the respective system size at that point is different for every graph, this is unlikely to be connected to an increasing workload per processor. In fact, the parameters that the solver dynamically adjusts during the simulation, namely the number of V-cycles n_V and the number of smoothing steps $n_{\text{smooth}} = n_{\text{pre}} + n_{\text{post}}$ (logged separately), barely change over the course of the weak scaling measurements. For the 30^3 grid points per processor case, they are given in table 5.2. Here, the number of V-cycles

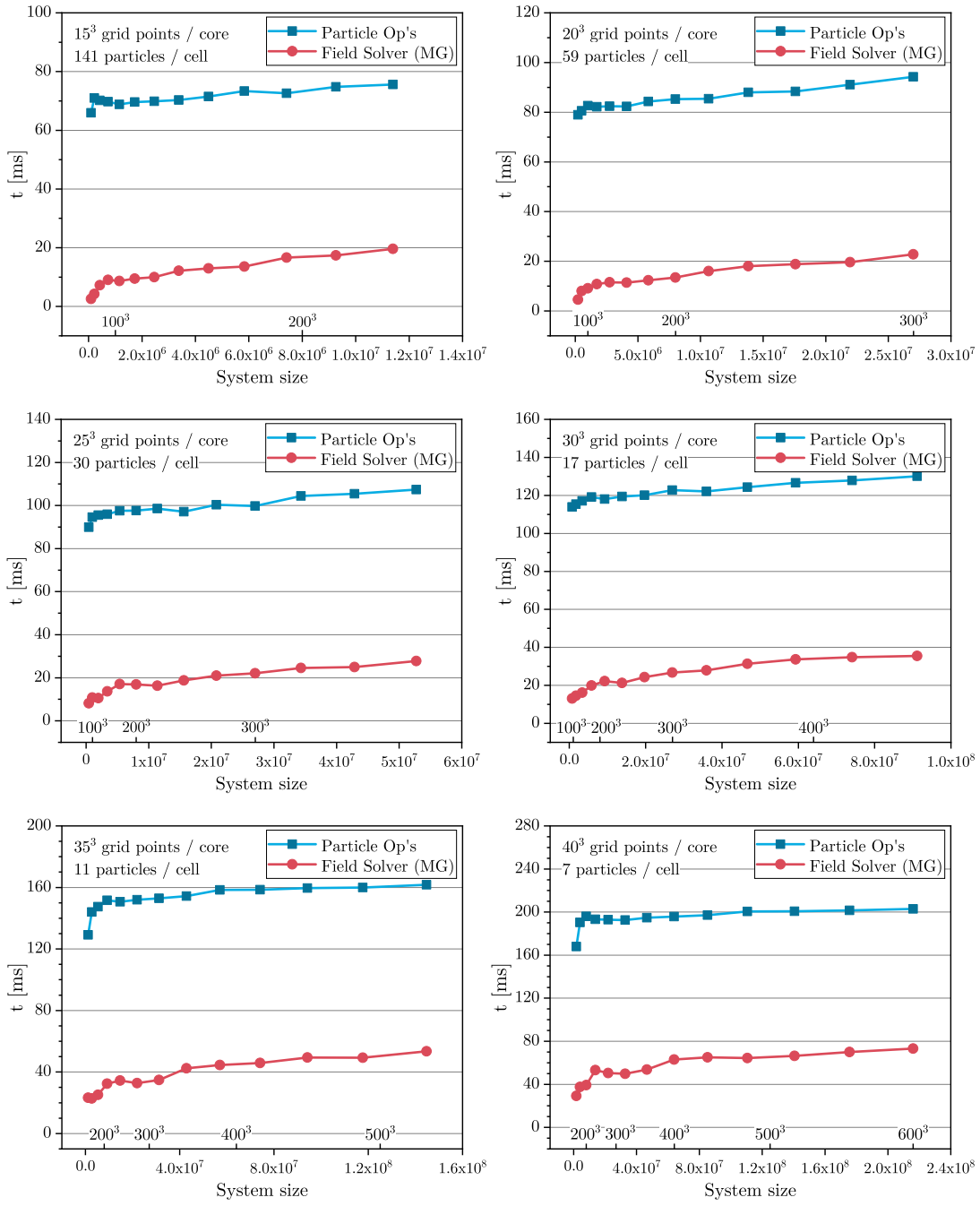

Figure 5.11: Weak scaling of the multigrid field solver within PlasmaPIC, generated by measuring the cumulative time to solution over 10,000 time steps. Each graph displays the measurements corresponding to a single time step for a distinctive number of grid points per processor. The 13 data points were consequently generated by computations on successive cube numbers of processors, ranging from 27 to 3375. For comparison purposes, the time PlasmaPIC spends on the particle operations is displayed as well.

Cores	System size	n_V	n_{pre}	n_{post}
27	90^3	2.00	2.00	2.00
64	120^3	2.00	2.00	2.00
125	150^3	2.00	2.00	2.00
216	180^3	2.02	2.02	1.98
343	210^3	2.00	2.00	2.00
512	240^3	2.01	2.03	1.99
729	270^3	2.03	2.07	1.97
1000	300^3	2.03	2.21	1.99
1331	330^3	2.00	2.15	2.04
1728	360^3	2.02	2.13	1.98
2197	390^3	2.03	2.18	1.98
2744	420^3	2.05	2.23	1.97
3375	450^3	2.04	2.26	2.00

Table 5.2: Average values for the dynamically adjusted parameters of the multigrid solver at 30^3 grid points per processor

remains very constant and the number of smoothing steps (then roughly proportional to computational workload) overall increases by marginal 6.5 % over a range of system sizes that vary by a factor of 125.
Given that the runtime approximately doubles between the calculations on 216 and 3375 cores, this can't be the main contributing factor.

Rather, the increase can be associated with the change in the number of processors used. A crucial property of the InfiniBand network of LOEWE-CSC is that it is configured as a 2:1 blocking fat-tree. In that case, several compute nodes are connected to the same network switch and can communicate with the full network bandwidth. However, with 2:1 blocking, the number of communication channels to the rest of the network is only half that between node and switch level. If all nodes connected to a switch send messages to nodes on different switches, these messages therefore need to be queued, implicating that for two separate messages of same length it can take twice as long for one to be delivered than for the other. Additionally, the network bandwidth becomes a partially shared resource, meaning that communication-heavy compute jobs started by different users may interfere with each other performance-wise.
For a problem that is parallelized using the domain decomposition approach, such a blocking network won't affect performance in that runtime shows a sharp increase when the number of processors is scaled up. Considering that each process has up to six neighboring processes with which it communicates bidirectionally and that received messages can be used directly to continue with calculations, communicating with a single processor to which it has reduced bandwidth can easily be compensated for by the time it takes to fully process the other neighbors' messages. As more and more neighbors are not connected to the same network switch, the performance therefore worsens.
To confirm this conclusion, the weak scaling test would have to be repeated on a cluster with a non-blocking network with otherwise similar capabilities, which was unfortunately not available for this thesis.

However, these weak scaling measurements show that by using the multigrid field solver, PlasmaPIC's overall runtime won't be dominated by it, independent of system size and number of processors.

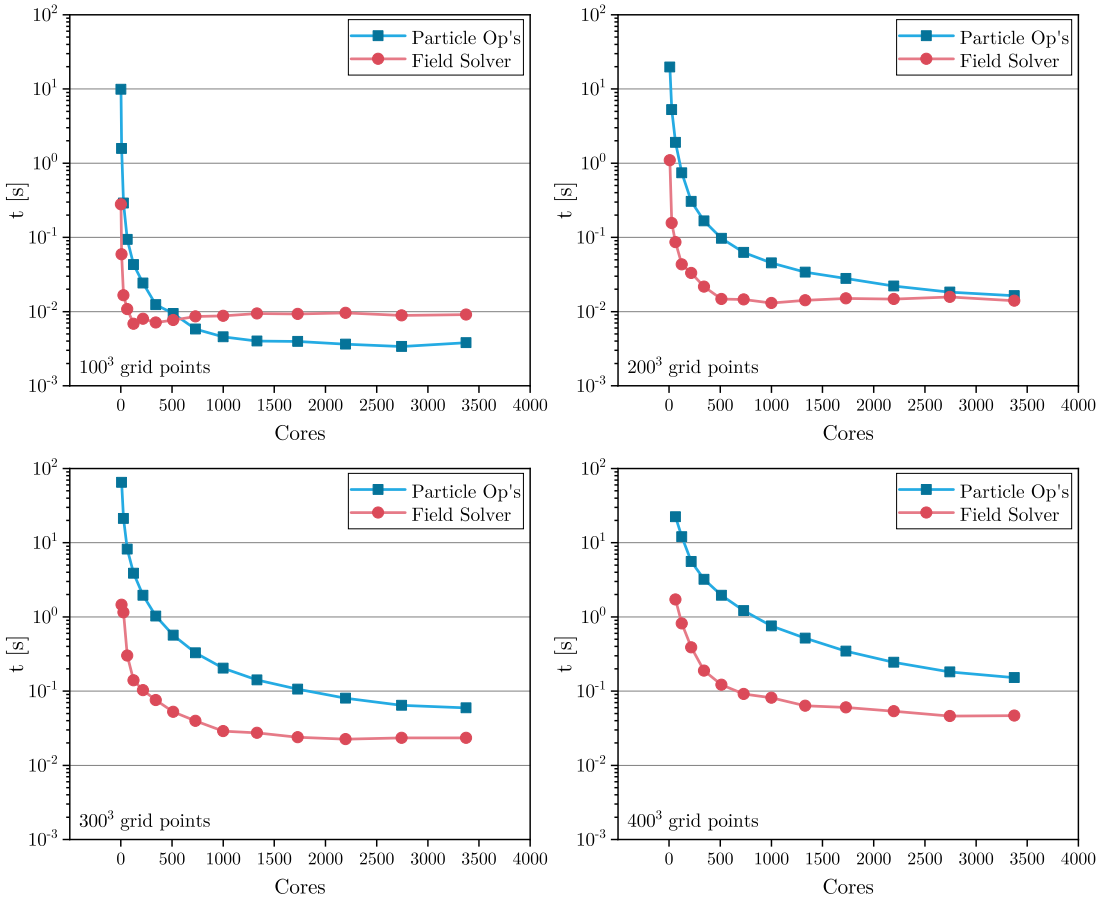

Figure 5.12: Average runtime on systems of fixed size (100^3, 200^3, 300^3, 400^3) over 10,000 time steps of field solver and particle operations plotted against the number of processors

5.2.3 Strong Scaling

While not being as relevant to the goal of this thesis as weak scaling, measurements of the strong scaling behavior offer a different perspective on the performance of the multigrid field solver.
Using the same box-shaped plasma discharge chambers as for the weak scaling tests, runtime of both field solver and particle operations were measured on four different systems (100^3, 200^3, 300^3, 400^3 grid points) for cube numbers of processors in the range from 1 to 3375. Due to the limited memory size of a single compute node, the minimal number of processors had to be increased step by step, so the largest system could only be simulated on at least 64 processors. The results are displayed in figure 5.12.

As already observed in section 5.2.1, the field solver's runtime quickly approaches a constant value for small systems. It furthermore surpasses that of the particle operations, which keeps decreasing for all configurations.
For the larger systems, the scaling behavior is generally better. This can be visualized by rescaling the runtime plots to a reference value, generating a unitless speedup value for each data point (figure 5.13). For comparability reasons, the reference value for each system is the respective data point measured with 64 processors. This is a distinction from the standard approach, where speedup is the ratio of runtime on one core to that

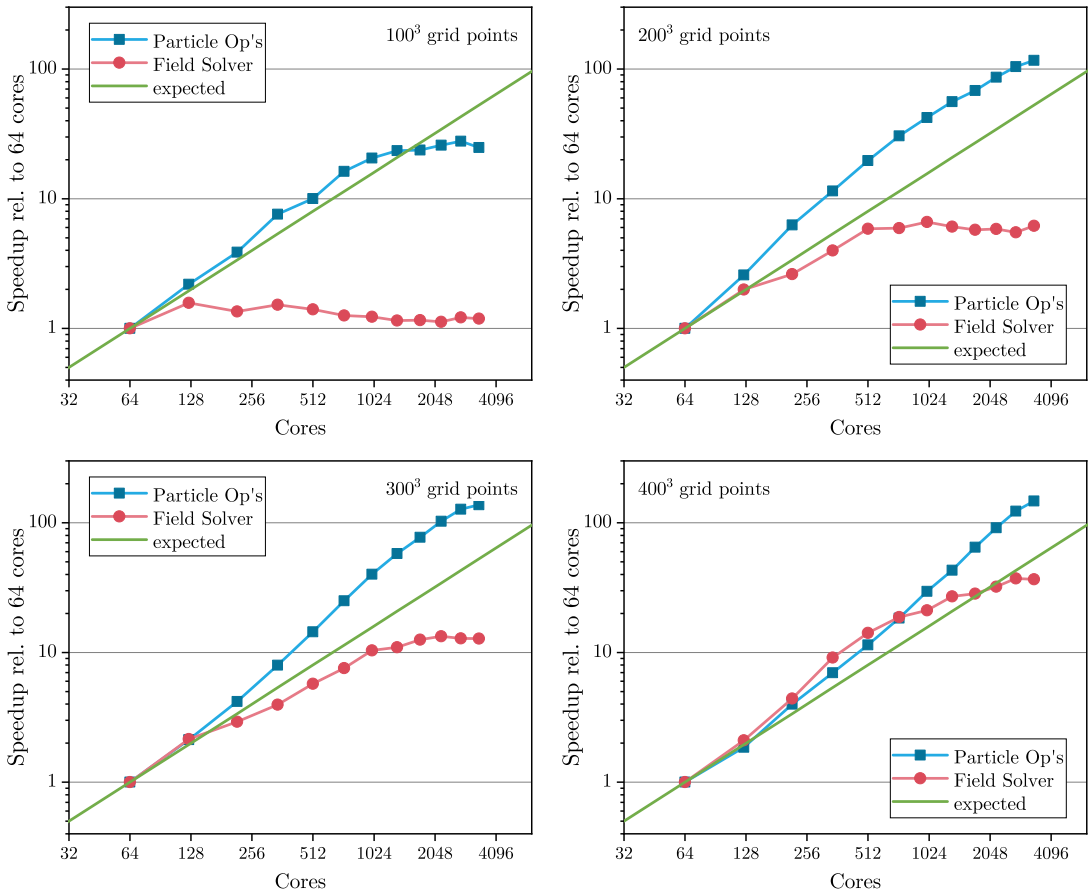

Figure 5.13: Speedup of both the field solver and the particle operations. The reference value is chosen to be at 64 processors for all four systems.

on N_p cores.
In theory, the expected ideal speedup as a function of the number of processors is a simple linear function with a gradient of 1, i. e., the same problem would be solved twice as fast on two processors than on one and eight times faster on 512 processors than on 64. In practice, however, and if the underlying algorithm is relatively independent of communication, speedup values greater than these expected values can be achieved. This is due to the fact that with more processors, increasingly larger portions of the fixed-size problem fit into the respective CPU caches, allowing for increasingly efficient computational operations.

Such characteristics, exceeding the expected results, can be observed for the particle operations in PlasmaPIC, as shown in figure 5.13, and were already described in [8]. Only for the 100^3 system the increase in speedup eventually deteriorates.
The multigrid field solver on the other hand is far more dependent on communication and caps at a constant speedup whose value increases with system size. Only on the largest system, communication is not the determining factor and speedup values greater than the expected are achieved (on less than 2000 processors).
This fundamental difference between the performances of the field solver and the particle operations that can't be observed by only measuring weak scaling illustrates a significant algorithmic imbalance within PlasmaPIC: The particle operations benefit more from

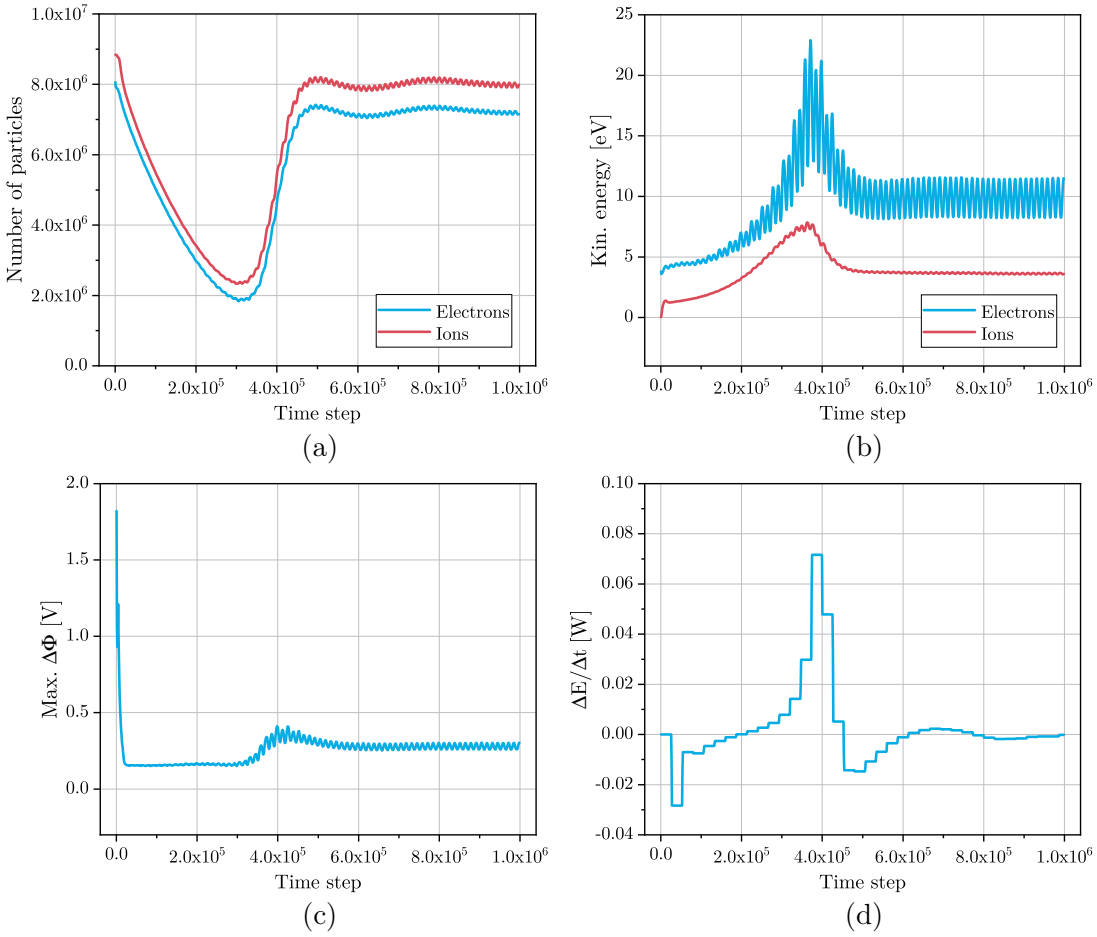

Figure 5.14: Evolution of the number of particles (a), the average kinetic energy (b), the maximal change in the electric potential over one time step (c), and the net energy deposited into the system (d) over 10^6 time steps, as observed for a simulation of the RIT-1.0 for which the accuracy of the multigrid field solver is very high.

using more processors and the communication-heavy field solver is ultimately still the bottleneck of overall strong scaling. However, the process of choosing an appropriate number of processes for a given simulation can be simplified to only considering the expected runtime of the particle operations, as the field solver basically always completes in minimal time.
This is especially important for simulations for which the number of particles is higher than in the cases discussed here (higher neutral gas density and/or higher power input). While the workload for the particle operations is directly proportional to the number of particles (electrons and ions) and therefore to the densities n_e and n_i as well, the field solver's runtime is mostly dependent on the mesh size Δx, which only needs to be scaled with $n_\mathrm{e}^{-1/2}$ (cf. equations (2.23) and (2.24)).

5.2.4 Influence of the Field Solver's Accuracy

At the start of every PlasmaPIC simulation, a value for r_tol, the multigrid field solver's convergence criterion (equation (4.21)), is read from the input card. Principally, a

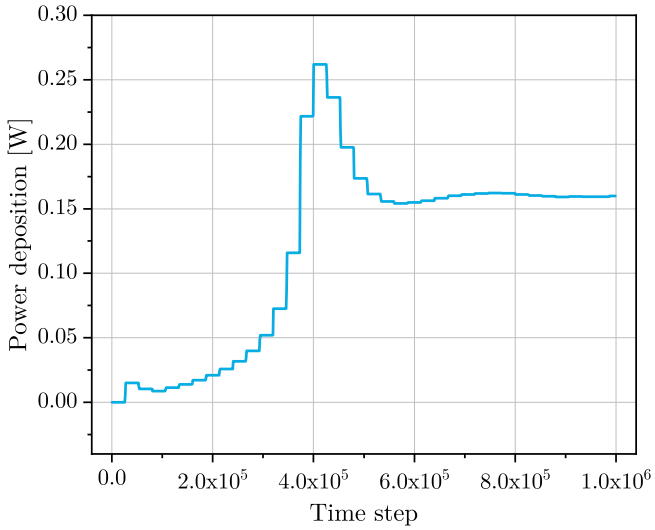

Figure 5.15: Power deposition into the simulated plasma by the electromagnetic fields induced by the coil that is wrapped around the thruster. The target value defined in the input card is 0.16 W. The curve is stepwise constant because the coil current is only adjusted every RF period ($T = 0.2\mu\text{s} \mathrel{\hat{=}} 26,667$ time steps).

simulation runs smoothly for a wide range of r_{tol}, with the lower limit resulting from the finite machine precision. I. e., if r_{tol} is set too small, an iterative process may never complete because the stop criterion is unreachable.
The upper limit on the other hand is determined by the effects the accuracy has on the behavior of the simulated plasma. An inaccurate solution for the electric potential results in unphysical acceleration of the charged particles which not only violates the conservation of energy over a single time step but may ultimately invalidate the whole simulation.
Since the pursued accuracy directly affects the field solver's runtime, it is however desirable to use a value that is both close to the upper limit and far enough away from it to allow for a stable and physically accurate simulation.

In order to investigate the influence of the field solver's accuracy on PlasmaPIC as a whole, simulations of both the RIT-1.0 and the RIT-2.5 were started with various different values for r_{tol}. By comparing certain quantities with what can be observed for high-accuracy runs, an appropriate value can then be found.

Figure 5.14 shows the evolution of four different quantities over the course of 10^6 time steps for the case of high accuracy ($r_{\text{tol}} = 10^{-8}$, RIT-1.0): the number of particles in the simulation domain (electrons and ions, separately), the average kinetic energy of the particles, the maximal change in the electric potential on any grid point compared to the respective previous time step, and the net energy lost or added to the system per RF cycle.
The shape of all displayed curves is related to the process of establishing a stable plasma discharge from a homogeneous charge distribution. At first, many electrons and ions strike the various surfaces in the domain and are removed from the simulation. After the plasma sheath has formed and the amount of power that is deposited into the plasma is gradually being increased (cf. fig 5.15), their total number rises again. How strong the average kinetic energy and the electric potential further fluctuate depends on the

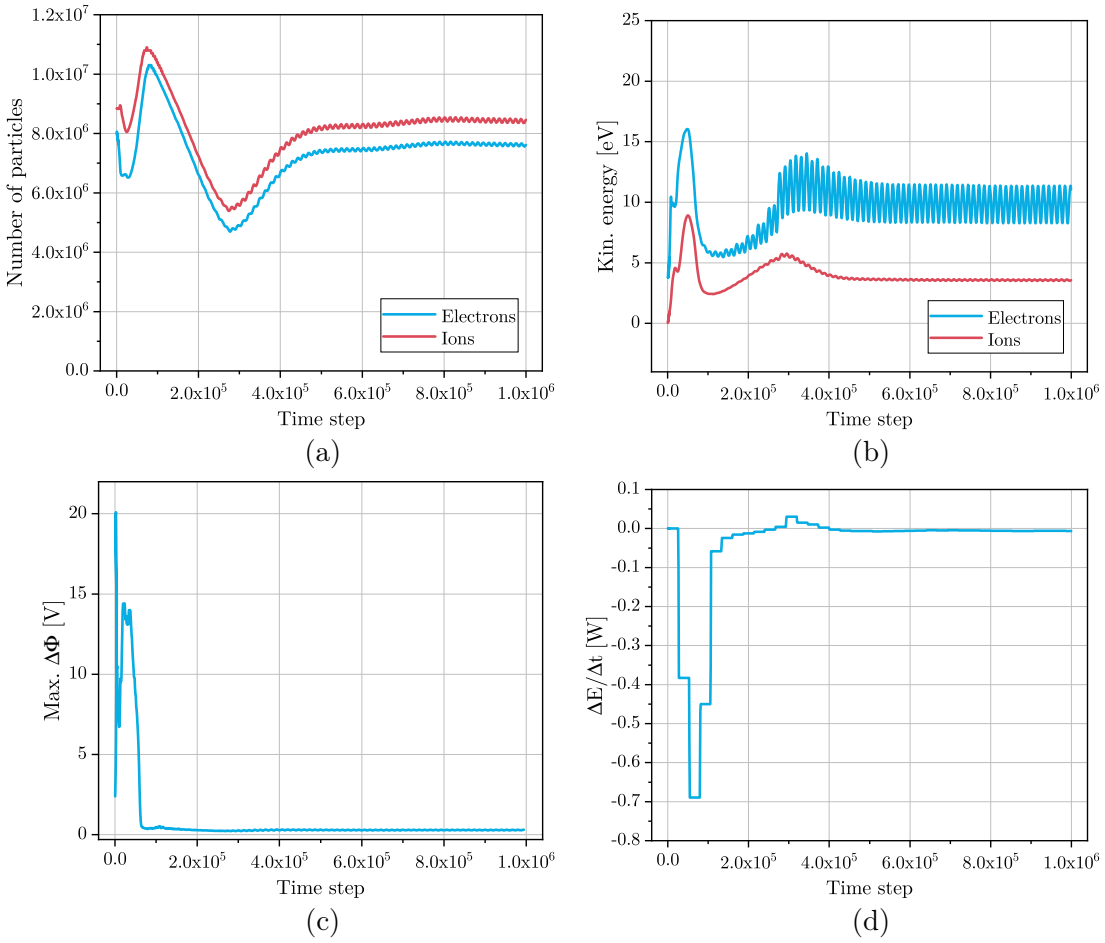

Figure 5.16: Evolution of the number of particles (a), the average kinetic energy (b), the maximal change in the electric potential over one time step (c), and the net energy deposited into the system (d) over 10^6 time steps, as observed for a simulation of the RIT-1.0 for which the accuracy of the multigrid field solver is very low.

deposited power as well. The distance between successive maximums and minimums depends on the frequency of the alternating coil current (the charged particles are accelerated and decelerated twice per RF period).
The bottom right curve describes the difference between the energy deposited into the plasma and the energy "lost" due to interactions with a wall (particles and their energy are removed from the simulation; only their charge may reside on surfaces) or the neutral background gas (ionization, excitation, charge exchange) and is only supposed to be zero once a stable plasma discharge is achieved and the particles' mean kinetic energy over one RF period remains constant. The state of the plasma can then be described as a dynamic equilibrium, where the number of particles being removed from the simulation is equal to the number of particles generated from the neutral background gas by ionization (over one RF period).

Repeating the same simulation with a high value for r_{tol} yields the curves displayed in figure 5.16. Here, r_{tol} was set to $8 \cdot 10^{-3}$. Simulations started with a greater value aborted prematurely due to particles leaving the simulation through domain boundaries that were not defined as exit planes (which can only happen if they are accelerated strong

enough to pass through a wall element).
The reduced accuracy seems to primarily affect approximately the first 10^5 time steps. There is an additional maximum in the number of particles and their kinetic energy, as well as significantly bigger changes in the electric potential. Since this coincides with a net loss of energy, unphysical processes must occur in the simulation.
However, after this initial phase, the system reaches an equilibrium state that is very similar to that achieved with $r_{\text{tol}} = 10^{-8}$. As shown in figure 5.17, even the distribution of particles and the electrostatic potential are very similar, so there is no clear indication that the state of the simulated plasma is actually a result of unphysical accelerations. Only the total number of particles stabilizes at a visibly higher value. Setting r_{tol} to 10^{-3} fixes this discrepancy and the simulated plasma's state of equilibrium is relatively equivalent. This indicates that the field solver's accuracy may be lowered after a certain number of time steps without affecting the outcome, which is a potential approach for future optimizations.

Figure 5.18 shows for four higher values of r_{tol} how the various quantities deviate from a run with $r_{\text{tol}} = 10^{-8}$. Apart from the net energy influx, for which the value of a stationary plasma is ideally zero, all curves represent the relative deviation as percentage values.
While the general assumption that the results align closer with the high-accuracy simulation the lower r_{tol} is chosen holds true, the general long-term behavior of the simulation at $r_{\text{tol}} = 10^{-3}$ is already very close to that case. Although deviations in the low single-digit percentage range can be observed, this is unlikely to affect the overall ability of

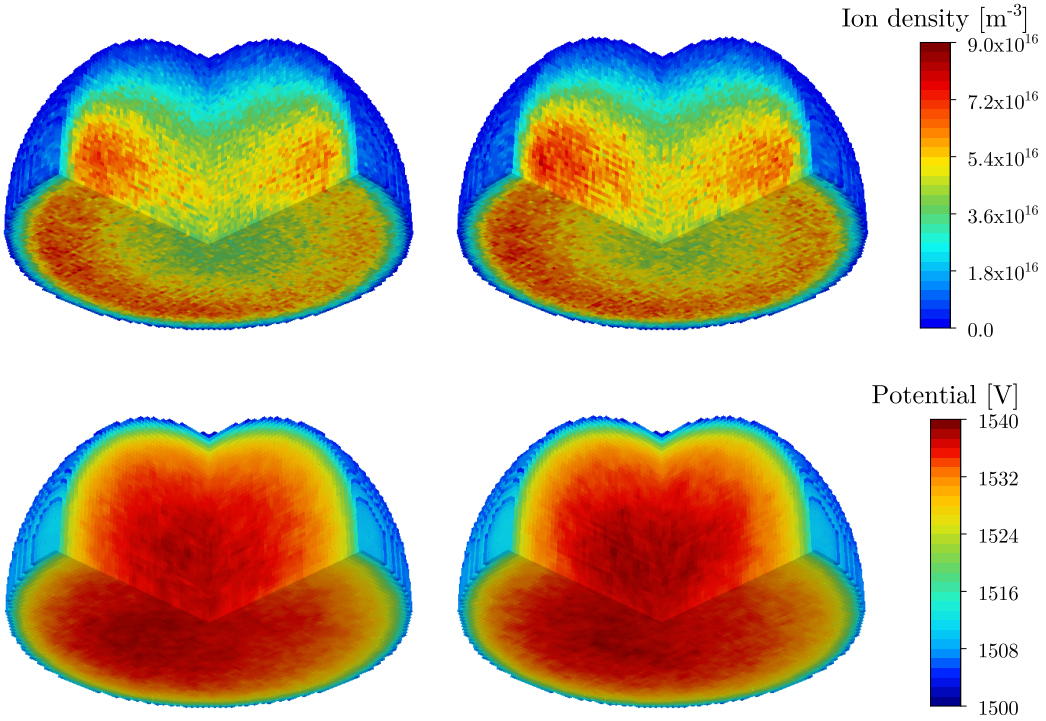

Figure 5.17: Sectional view of the ion density n_{i} (top) and the electrostatic potential Φ (bottom) inside the RIT-1.0, simulated with high accuracy of the field solver (left) and low accuracy (right). The depicted scales are simultaneously valid for the left and right side. Both simulations result in the asymmetric density distribution described in [8].

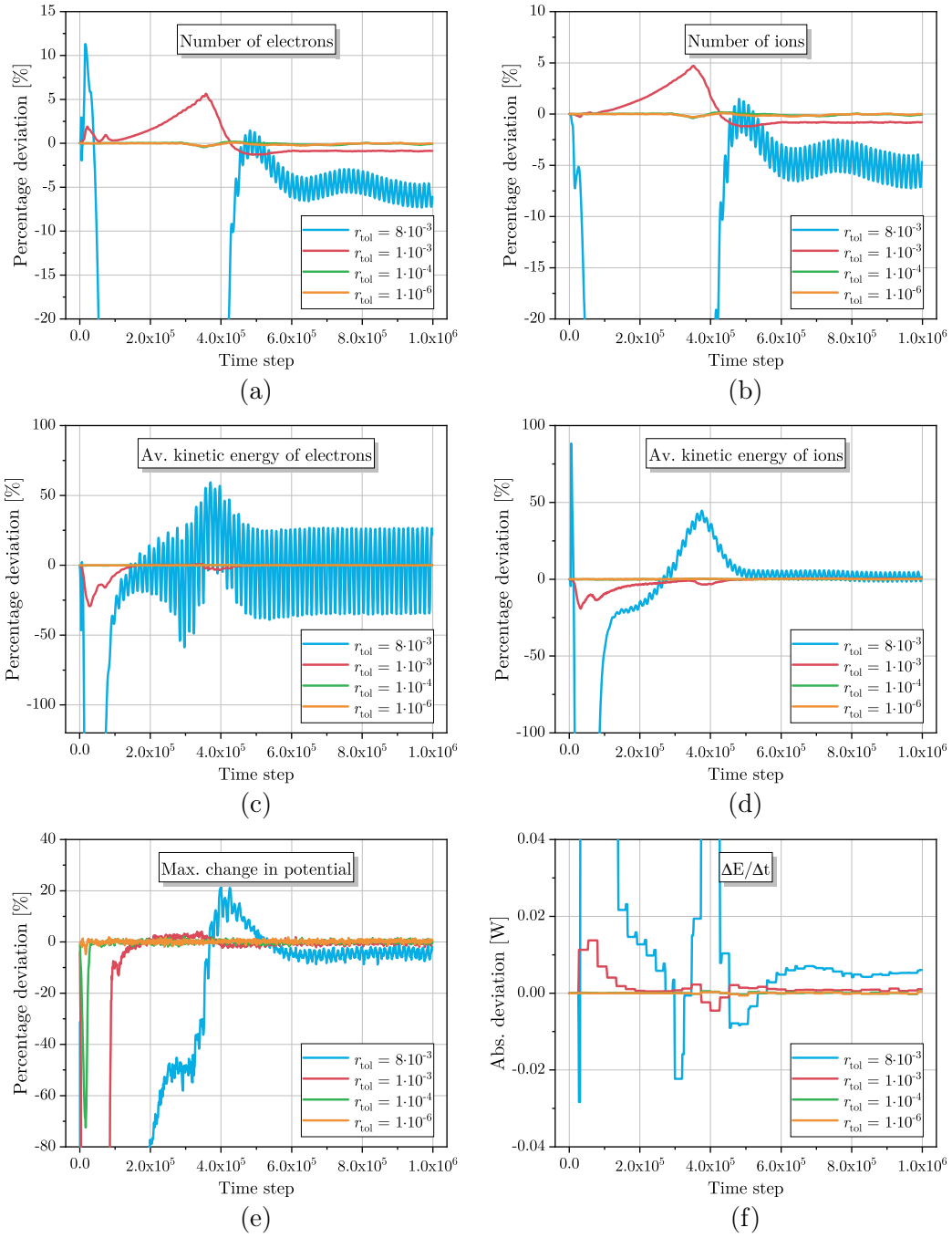

Figure 5.18: Deviation from the values obtained from a simulation run with $r_{\text{tol}} = 10^{-8}$ of the considered quantities for $r_{\text{tol}} = 8 \cdot 10^{-3}$, $r_{\text{tol}} = 10^{-3}$, $r_{\text{tol}} = 10^{-4}$, and $r_{\text{tol}} = 10^{-6}$, all plotted against the simulation time step. (a) The total number of electrons in the system. (b) The total number of ions. (c) The average kinetic energy of the electrons. (d) The average kinetic energy of the ions. (d) The maximal change in the electric potential from one time step to the next. (e) The net energy influx, calculated by subtracting the energy of conversion events PlasmaPIC doesn't keep further track of from the energy deposited by the electromagnetic fields induced by the coil current.
The displayed ranges of values are chosen such that the stable behavior to which the simulated plasma settles after approximately $5 \cdot 10^6$ time steps is shown in appropriate detail. This entails not fully covering the initial fluctuations of the low-accuracy runs.

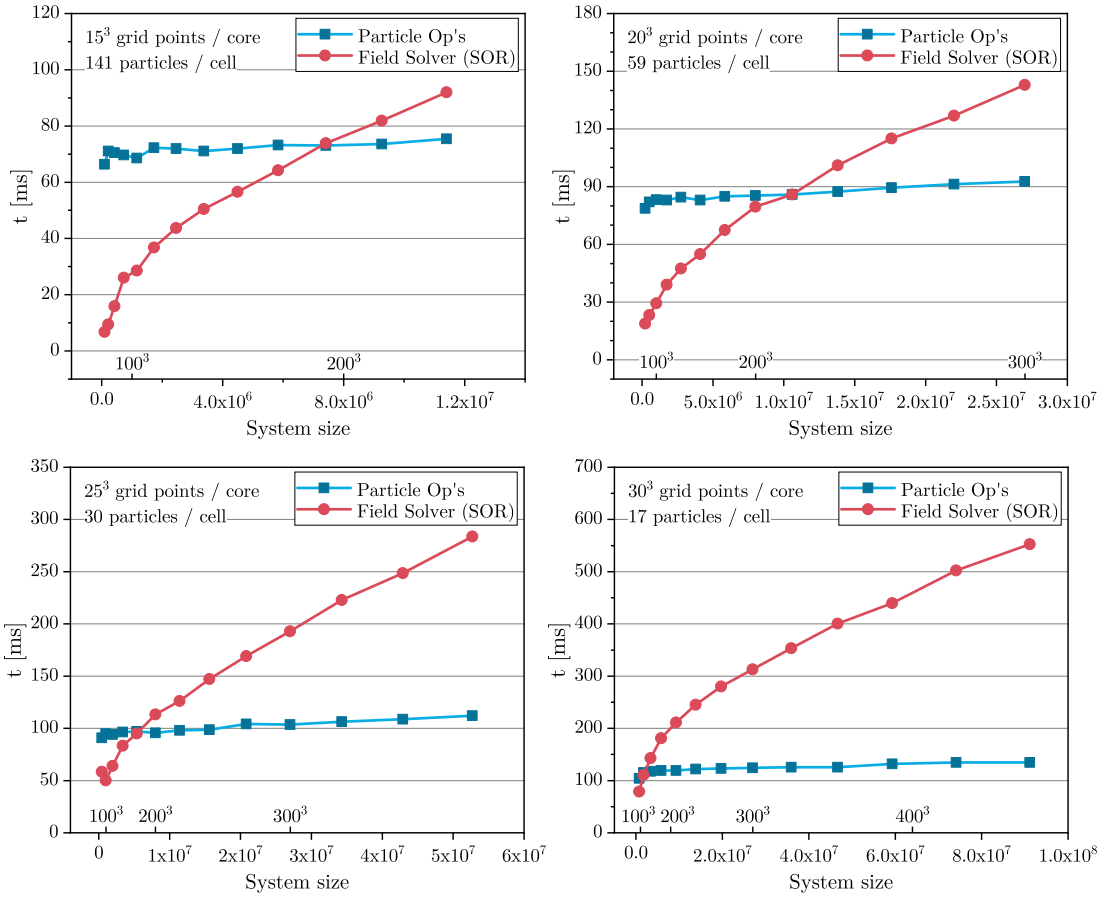

Figure 5.19: Weak scaling of the SOR field solver within PlasmaPIC, generated by measuring the cumulative time to solution over 10,000 time steps. The measurements were performed completely analogous to those for the multigrid solver (figure 5.11), but only up to 30^3 grid points per processor. The displayed timings correspond to a single time step.

PlasmaPIC to mirror a real plasma, considering that simplifying assumptions made to model a physical problem always introduce some form of error.
However, a more accurate solution of the field solver can always be enforced so that its accuracy can be further investigated and possibly ruled out as a problem.

For the RIT-2.5, the initial fluctuations of the considered quantities are generally bigger and a value of r_{tol} of not more than 10^{-4} must be chosen. Whether this is a trend for increasingly larger systems or simply an advantage of such a small system as the RIT-1.0 remains to be investigated.

5.3 Comparison with the SOR Method

Before the development of the multigrid solver, a parallel red-black SOR solver was used as PlasmaPIC's field solver. Motivation for this was the relatively simple implementation and the method's applicability to small systems. In order to demonstrate the improvements made by the switch, the weak scaling measurements discussed above were performed for the SOR method as well, using the same systems and the same

convergence criterion.
Figure 5.19 shows these measurements for up to 30^3 grid points per processor. It's evident that using larger local grids doesn't offer further insights, as the field solver's runtime surpasses that of the particle operations after the first two data points (< 100 processors) with 30^3 grid points per processor.

The shape of the measured curves are in very good compliance with what can be expected when parallelizing a $\mathcal{O}(n^{4/3})$ algorithm. Weak scaling imposes a relation of proportionality between n and the number of processors N_p, so ideally, the solver scales with

$$\mathcal{O}(n^{4/3} \cdot \frac{1}{N_p}) = \mathcal{O}(n^{1/3}) \ . \tag{5.7}$$

The original $\mathcal{O}(n^{4/3})$ scaling can further be reproduced by adding up the CPU time used by the individual processes (figure 5.20). This consequentially produces linear curves for the particle operations.

Figure 5.19 demonstrates the absolute necessity of a solver with proper scaling behavior. While SOR is very competitive for the smallest systems, its workload then quickly increases and passes that of the particle operations. A direct comparison with the

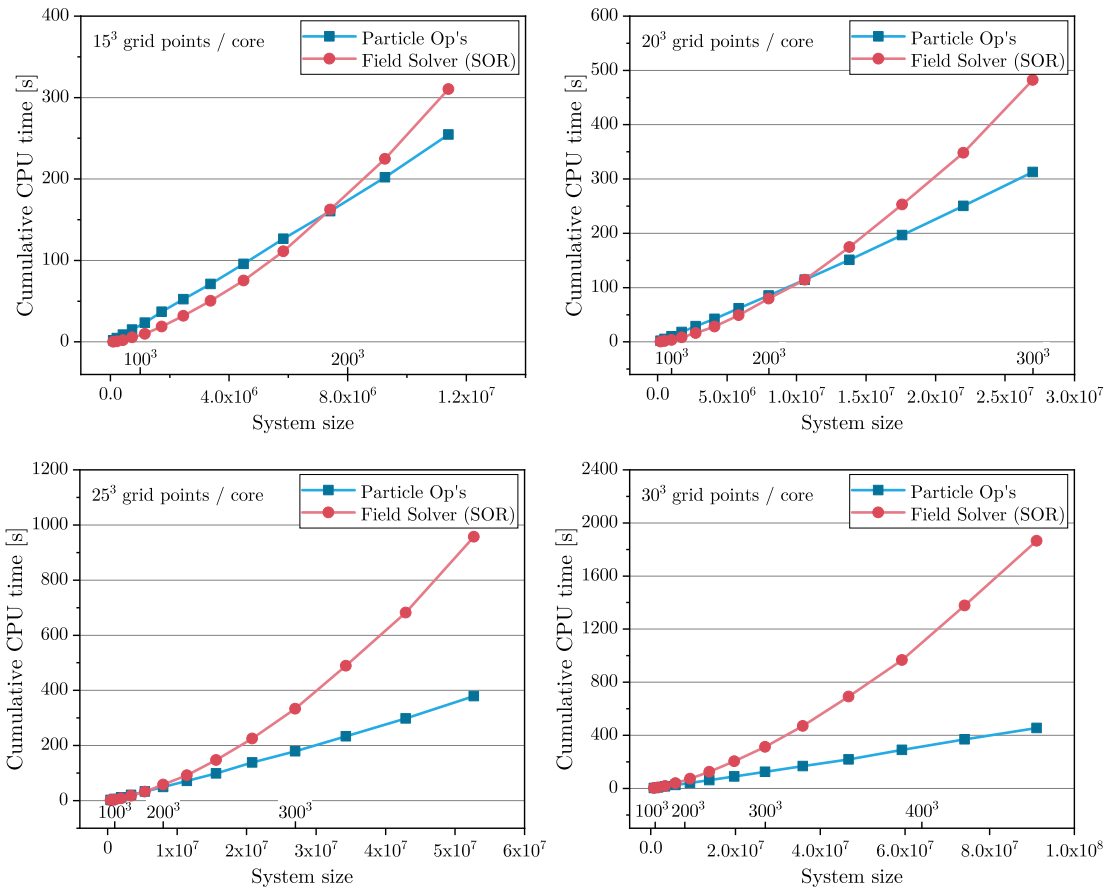

Figure 5.20: Cumulative CPU time used for the SOR weak scaling measurements, plotted against the total system size for 15^3, 20^3, 25^3, and 30^3 grid points per processor. Each data point represents the time needed for one time step (calculated by averaging the time for $10,000$ time steps).

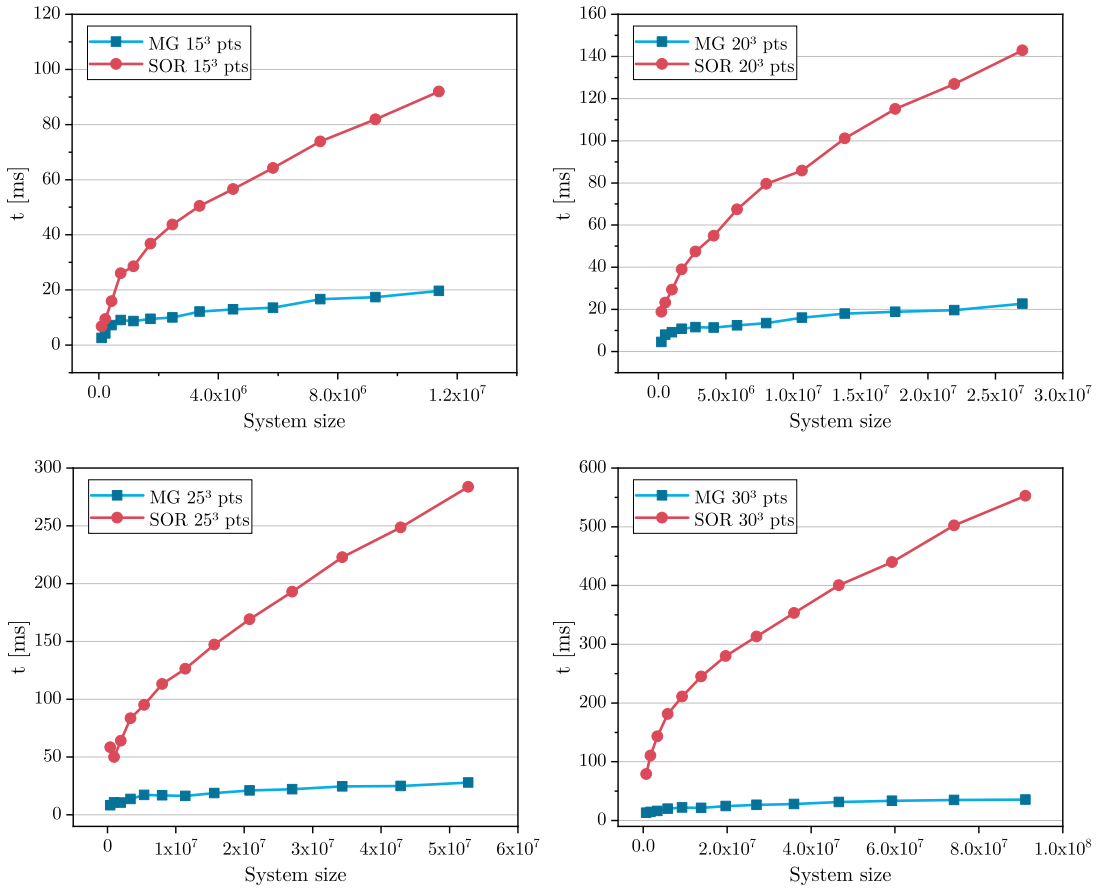

Figure 5.21: Weak scaling of both the SOR solver and the multigrid (MG) solver for 15^3, 20^3, 25^3, and 30^3 grid points per processor. The multigrid solver is faster for every considered system size.

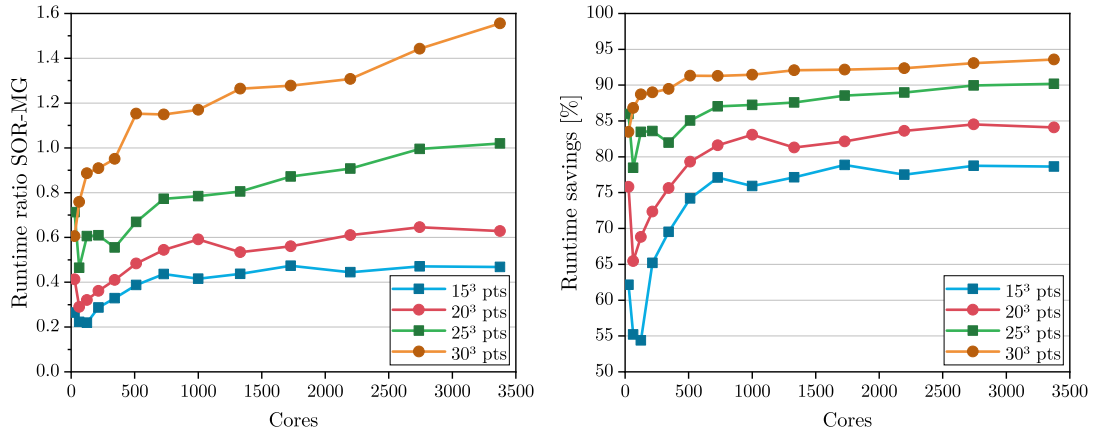

Figure 5.22: Improvements made by the multigrid solver in comparison to the original SOR solver, depicted by the metrics of runtime ratio (left) and relative runtime savings (right).

multigrid solver (figure 5.21) furthermore shows that not only was the scaling behavior improved vastly by the new solver, the total runtime also benefits for every measurement, even for systems as small as 45^3 grid points (computed with 27 cores). This was not a requirement for the multigrid solver but adds to the list of advantages over other solvers.
For figure 5.22, the runtime measurements of the two methods were further compared directly by displaying their ratios (left graph) and the percentage of the runtime of the SOR solver that is saved by the switch to the multigrid solver.

5.4 Comparison with PETSc

The widely used PETSc (*Portable, Extensible Toolkit for Scientific Computation*) software suite, developed by the Argonne National Laboratory, offers a wide range of data structures and routines for the direct and iterative numerical solution of partial differential equations. Like the multigrid solver developed in the scope of this thesis, it supports MPI and aims to be generally scalable but has the advantage of being applicable more universally.
For the solution of linear systems such as those arising from discretizing the Poisson equation, it provides the tools to assemble parallel vector and (sparse or dense) matrix objects, which can then be used in conjunction with numerous Krylov subspace methods and preconditioners (discussed in section 3.8).
Particularly, it allows for the linear system to be set up without further arrangements for a specific solution method and the combination of Krylov subspace method and preconditioner, including all additional configuration parameters, can be chosen at runtime.
This is beneficial for problems for which a suitable approach is not initially known or for optimizing the solution process in time-critical applications.

In order to assess PETSc's capabilities as PlasmaPIC's field solver, it therefore not only has to be implemented using the provided interfaces, a specific solution method also needs to be found. For this, the wide range of available Krylov subspace methods and preconditioners was systematically tested for performance on three of the cubic plasma vessels used for the weak scaling tests in this thesis. In all three cases, the lowest runtime was achieved with the *flexible GMRES* method, preconditioned with classical algebraic multigrid. This result is in accordance with the general consensus that the Poisson equation is solved most efficiently with a multigrid method.
As any multigrid method, the algebraic multigrid preconditioner implemented in PETSc is configurable with a vast variety of options, including the smoothing method, the number of smoothing steps, and the choice of coarse grid solver. The extensive tests revealed that the choice of these options only affects the number of total iterations if they add a significant overhead to a V-cycle (i. e., if the number of smoothing steps surpasses three or an elaborate smoother is used) and the plainest choice of options (i. e., one pre- and one post-smoothing step of block Gauss-Seidel) results in the lowest runtimes.

The choice of the Krylov subspace method is furthermore not a vital factor for performance, as other methods such as the conjugate gradient method or the generalized conjugate residual method achieve convergence in the same number of iterations (as

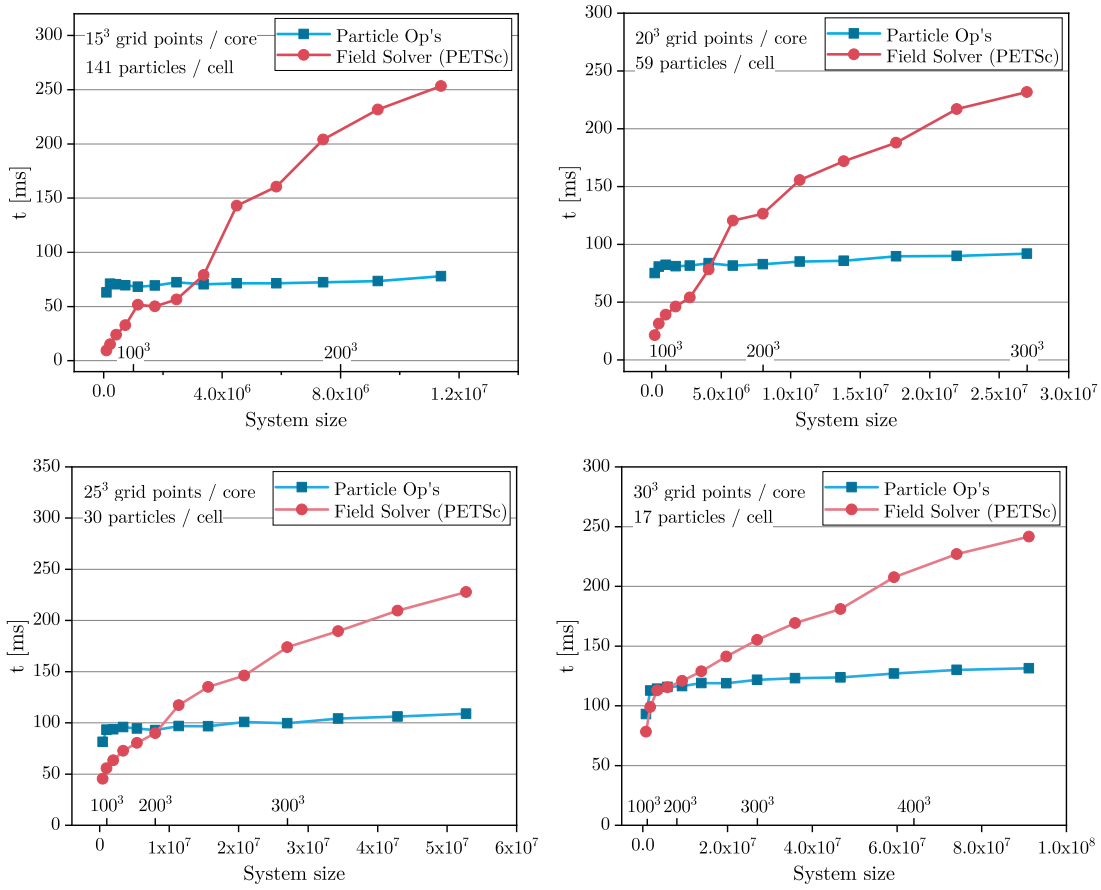

Figure 5.23: Weak scaling of the implemented PETSc solver for 15^3, 20^3, 25^3, and 30^3 grid points per processor.

long as the algebraic multigrid preconditioner is used) and perform close to the used method runtime-wise. It is, however, necessary to speed up convergence. While the algebraic multigrid method eventually does converge by itself, the addition of a Krylov method reduces the number of iterations significantly.

With an efficient PETSc solver being implemented, the weak scaling measurements previously done for the multigrid and SOR solver (that were developed specifically for PlasmaPIC) were performed a third time. The results are shown in figure 5.23. Again, the runtime of the particle operations serves as a reference.
Several observations can be made from these measurements. First, although the number of iterations remains very constant throughout the wide range of system sizes (between three, for the smaller systems, and four for the larger ones), the runtime is not even close to being constant but increases significantly. Second, while the smallest systems take longer to be solved the more grid points per processor are used (which is expected), the respective largest systems are solved in a very similar amount of time although they deviate by a factor of eight in size (225^3 compared to 450^3). As a consequence, the PETSc solver actually performs worse than the SOR solver for 15^3 and 20^3 grid points per processor.
A direct comparison with the multigrid solver, analogous to figure 5.22, is given by figure 5.24. While both SOR and the PETSc solver perform worse than the multigrid

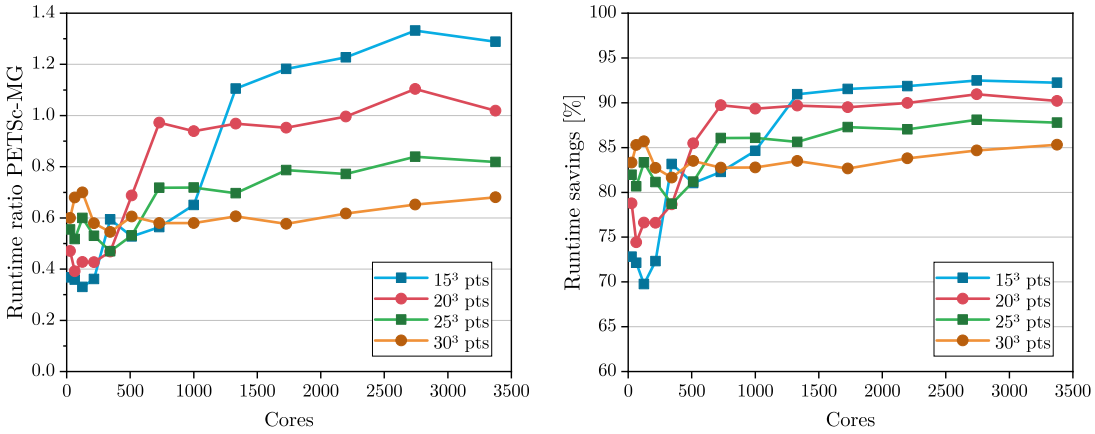

Figure 5.24: Advantage of using the multigrid solver developed for this thesis over a solver from an external software suite, depicted by the metrics of runtime ratio (left) and runtime savings (right).

solver for all measurements, the order in which the curves can be ranked as performing best compared to the multigrid solver is reversed, i. e., PETSc performs better relatively, if the number of grid points per processor is high.
In fact, the developers recommend at least 10,000 unknowns per processor, which corresponds to subdomains with approximately 22^3 grid points.

There are various reasons for PETSc's comparatively poor weak scaling which can be supported by the above observations and by a review of the log files generated by the PETSc environment.
The fact that the runtime on large numbers of processors is relatively independent of the number of grid points per processor indicates that not necessarily the size of the messages sent between the processors is the problem but rather their total number. The queuing of messages to be sent from a node could then be a contributing factor, more so if the number of communication steps per iteration is higher than for the other methods. The log files actually report a severe imbalance in the execution time of the communication routines of the order of a factor of ten.
The various routines called by the algebraic multigrid preconditioner further show a similar load-imbalance, which can be caused by the number of irregular Dirichlet boundaries not being fully reduced when the coarse grid systems are generated or by an inefficient execution of the coarse grid agglomeration approach.
The PETSc developers (personal communication, May to June 2018) further suggest insufficient memory bandwidth and low network latency as possible causes.

Nevertheless, the multigrid solver developed in the scope of this thesis shows to be far less restricted by these possible hardware problems and meets the requirements that were set as the main goal.

Chapter 6

Conclusions

Within the scope of this thesis, a parallel geometric multigrid field solver with great scaling capabilities was developed. This completes the efforts to optimize all algorithms used in PlasmaPIC, the 3D plasma simulation tool developed at the University of Gießen, towards applicability on (in theory) arbitrarily sized systems.

Multigrid methods, in general, spread the solution process over a hierarchy of increasingly coarser grids in order to efficiently reduce short- and long-range error components alike. By this, a powerful class of algorithms arises that offers the capability to solve a linear system of size n in $\mathcal{O}(n)$ operations.

The multigrid solver developed for PlasmaPIC distinguishes itself by being able to solve second order elliptic partial differential equations with variable coefficients on arbitrarily sized systems with irregular Dirichlet boundaries, while being parallelized very efficiently.
This is achieved firstly by combining the coarsening schemes of cell-centered and vertex-centered multigrid and applying the Shortley-Weller discretization scheme to the hierarchy of coarse grids and secondly by utilizing the concept of coarse grid agglomeration to prevent ineffective communication involving too many processors for a given set of data. A fast benchmark module further optimizes parallel performance for any new problem and hardware.

Thorough measurements of the weak scaling behavior show that by increasing the number of processors used for computation proportionally to the system size, the solver's runtime and therefore the time the simulation needs to progress by one time step can be kept almost constant. This is a crucial prerequisite for future simulations of large-scale plasma discharges such as those present in the radio-frequency ion thrusters (RITs) that are developed and investigated at the University of Gießen. Neither the previously used SOR field solver nor a solver based on the PETSc software suite achieved a similar performance, both regarding scaling behavior and runtime on any specific system size. While the former is restricted by its algorithmic scaling, the latter brought the network for inter-process communication of the used HPC cluster to its limit.
For example, a simulation of a RIT-1.0 on 96 processors that formerly took 36 hours to complete 10^6 time steps with the SOR solver is now approximately 44 % faster (20 hours). In this case, the number of simulated time steps per second increased from 7.6 to 13.7.

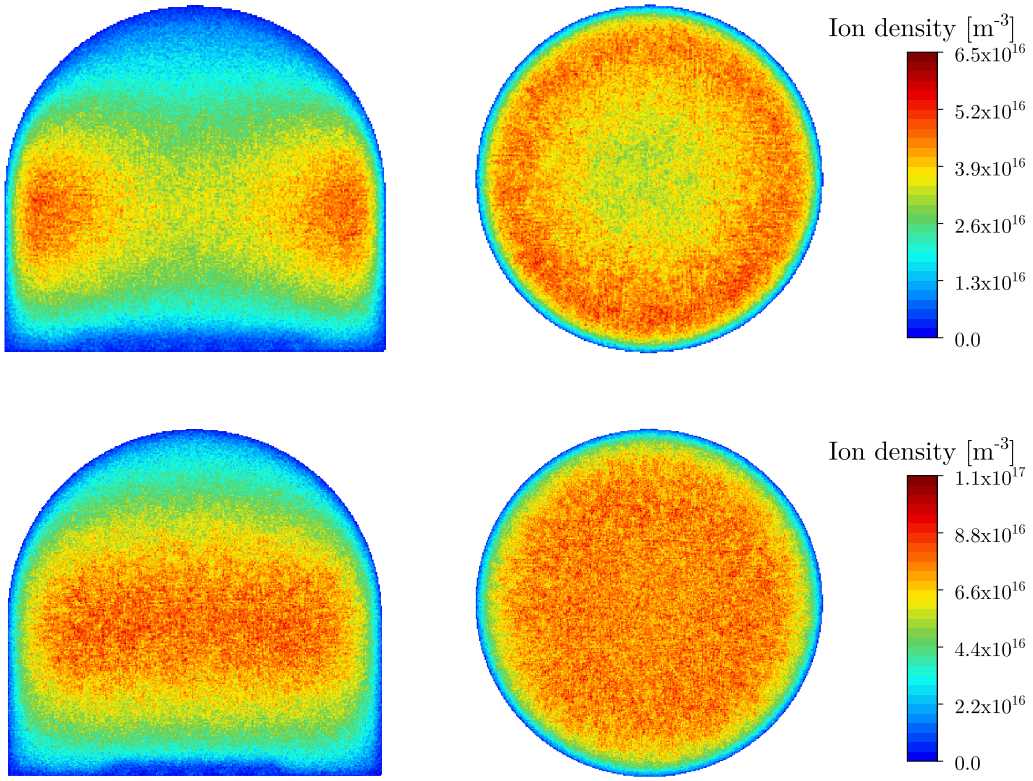

Figure 6.1: Cross-sectional views of the ion density in a RIT-2.5 simulation after $3{\cdot}10^6$ time steps. Two simulations were run in order to demonstrate the influence of the density n_0 of the neutral background gas (Xenon). Top: $n_0 = 2.5 \cdot 10^{19}\,\mathrm{m}^{-3}$. Bottom: $n_0 = 5 \cdot 10^{19}\,\mathrm{m}^{-3}$. The plane displayed on the left side, respectively, divides the domain into halves with respect to the y-axis. The plane on the right side corresponds to a slice perpendicular to the z-axis.

As expected, a similar comparison for the RIT-2.5 yields a far more significant performance improvement. Here, using the SOR solver leads to a total runtime of 136 hours on 1331 processors (2.0 time steps per second). The multigrid solver reduces the runtime by a factor of 4.6 to 29 hours (9.5 time steps per second). The increase in total runtime despite the nearly identical ratio of the total system size to the number of processors compared to the simulation of the RIT-1.0 can be attributed to, among other reasons, a disproportional increase in the total number of simulated particles.

The progress made here qualifies PlasmaPIC as a valuable tool for analyzing and optimizing RITs. Its application is further only limited by the available computational power.
However, since a large plasma discharge can be expected to take longer to form a dynamic equilibrium than a small one, the number of time steps that need to be simulated increases as well. This can't be compensated by using better-scaling algorithms and emphasizes the necessity to further improve and optimize PlasmaPIC.
For instance, the simulation of a RIT-2.5 reaches a dynamic equilibrium after close to $1.8 \cdot 10^6$ time steps, which is approximately three times more than it takes for the RIT-1.0. Moreover, the computations performed in the scope of this thesis were mostly set up such that relatively small plasma densities emerged as the dynamic equilibrium. In practice, however, RITs need to operate at higher densities in order to generate a signif-

icant thrust. This is achieved by higher pressures of the neutral gas and by increasing the power that is supplied to the plasma by inductive coupling.
As shown in figure 6.1, this affects not only the total number of charged particles in the simulation, but also their distribution. A brief discussion and a more detailed visualization of the plasmas generated from two different neutral gas densities can be found in the appendix.
Overall, future simulations of plasma discharges are likely to require disproportionally more particles in the domain of interest. Since this affects the particle operations (direct proportionality of the computational work) more than the field solver (the mesh size of the discretized domain Δx that dictates the system size n should be inversely proportional to the square root of the electron number density n_e), the latter's relative share of the total runtime should further decrease and a simulation's practical feasibility remains foremost a matter of available computational resources.

Appendix

Simulation of the RIT-2.5: Influence of the Neutral Gas Density

Two simulations of the RIT-2.5 were carried out, one with a neutral gas density n_0 of $2.5 \cdot 10^{19}\,\mathrm{m}^{-3}$ (which corresponds to a pressure of 10.4 mbar at a temperature of 300 K), the other with double that value. All other configuration options influencing the plasma were kept the same. Particularly, the target value for the electric power deposition was set to 0.8 W and the frequency of the alternating coil current to 2.86 MHz.

Figure A.1 shows a comparison of the development of various simulation parameters over $4 \cdot 10^6$ time steps. As can be expected, a higher gas pressure also increases the plasma density due to higher ionization rates. Similarly, the average kinetic energy of both electrons and ions decreases due to a higher abundance of potential collision partners, which restricts the acceleration by the electromagnetic fields.
A higher neutral gas density furthermore reduces the coil current that is necessary to reach the same power deposition. This is directly connected to the total number of charged particles that are accelerated by the electromagnetic fields. Since the kinetic energy of more particles is being increased, the individual energy transfer must reduce on average. Therefore, the magnitude of the electromagnetic fields induced by the coil current is decreased.

In good agreement with the observations made in [8], the density of the background gas also affects the distribution of the densities n_i and n_e of the ions and electrons. At low density of the neutral gas, a torus-shaped maximum of n_i and n_e evolves around the centered z-axis of the RIT. At higher density on the other hand, the particles distribute more evenly and form a wide maximum at the center of the plasma vessel. Sectional views of the distributions of n_i, n_e, and the electrostatic potential Φ are shown in figures A.2 and A.3.

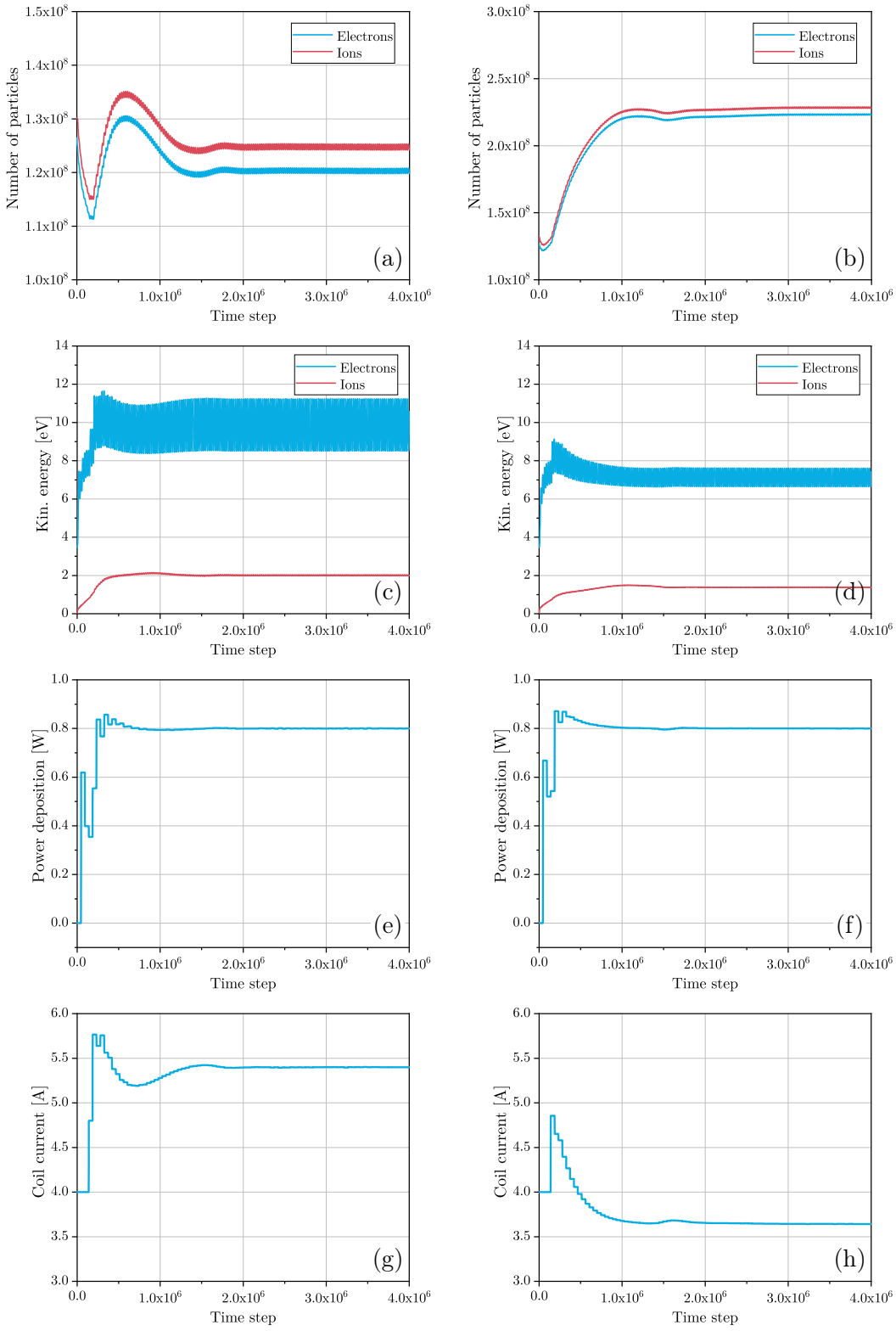

Figure A.1: Comparison of two simulations of the RIT-2.5, differing only by the number density n_0 of the neutral background gas. Left column of diagrams: $n_0 = 2.5 \cdot 10^{19}\,\mathrm{m}^{-3}$. Right column: $n_0 = 5{\cdot}10^{19}\,\mathrm{m}^{-3}$. Various parameters are plotted against the simulation time step: (a) and (b) Total number of ions and electrons in the simulation. (c) and (d) Average kinetic energy of the particles. (e) and (f) Power deposition. (g) and (h) Amplitude of the coil current.

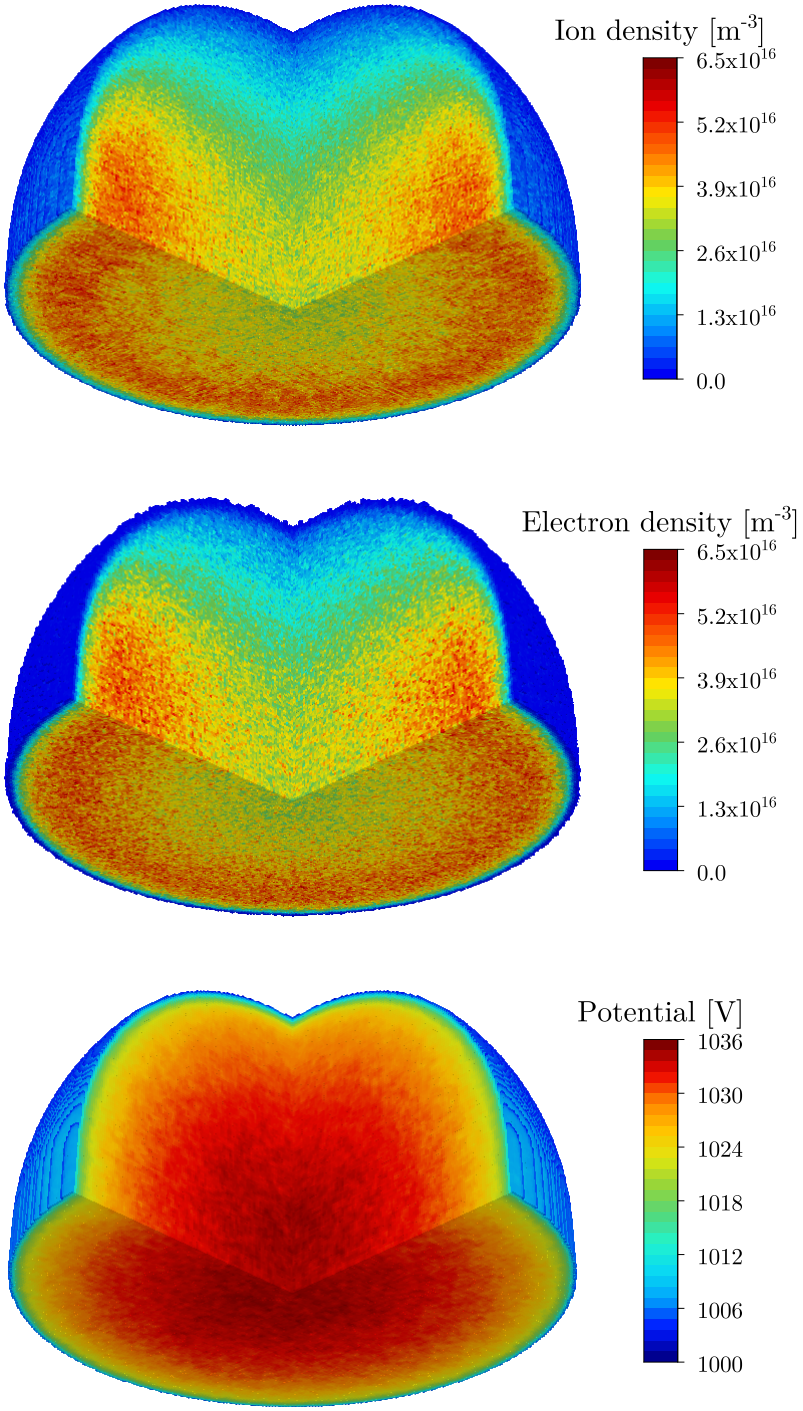

Figure A.2: Sectional views of the ion density n_i (top), the electron density n_e (middle), and the electrostatic potential Φ (bottom) after $3{\cdot}10^6$ time steps in a simulated RIT-2.5. A power of 0.8 W is deposited into the plasma by an alternating current (2.86 MHz) through the external coil. The density of the neutral background gas (Xenon) is $2.5 \cdot 10^{19}\,\mathrm{m}^{-3}$.

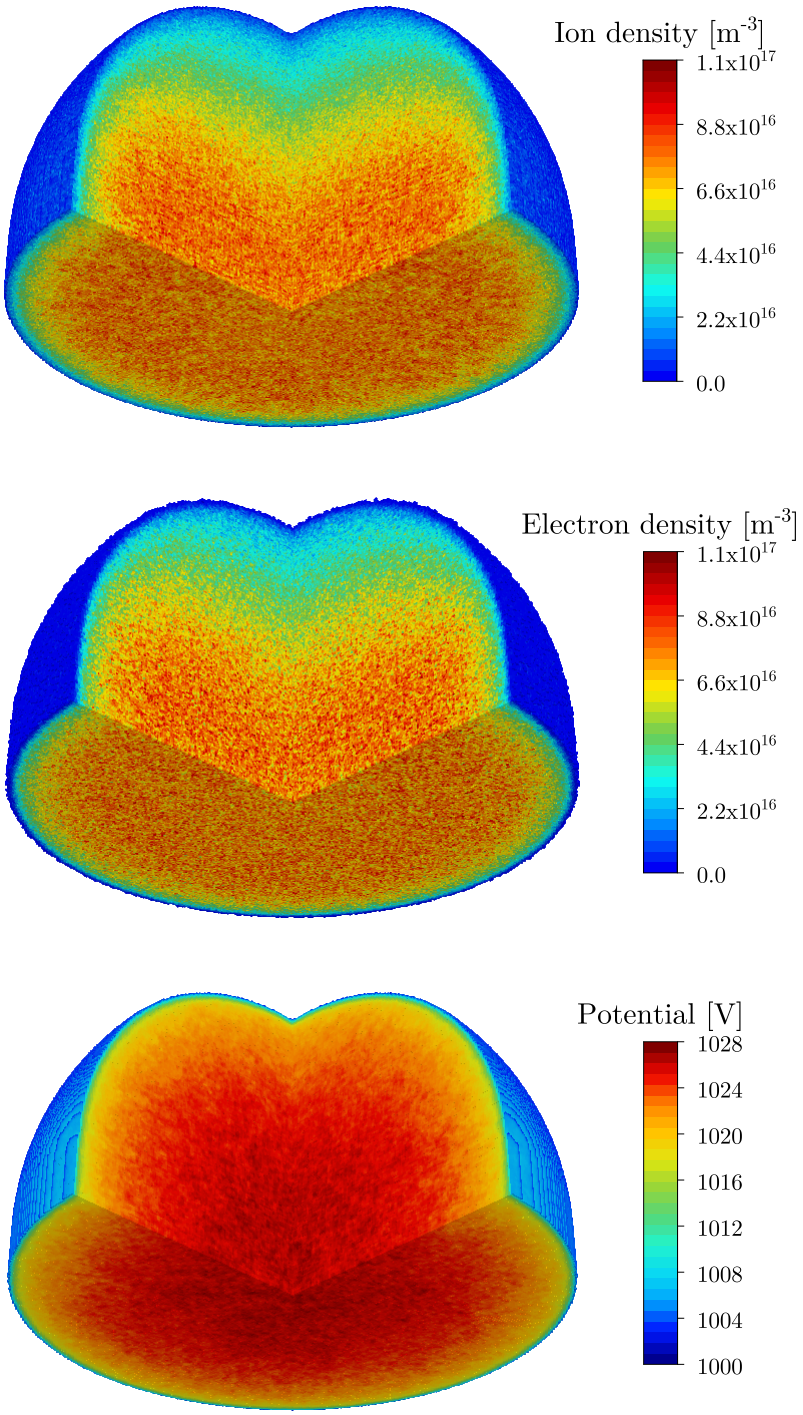

Figure A.3: Sectional views of the ion density n_i (top), the electron density n_e (middle), and the electrostatic potential Φ (bottom) after $3 \cdot 10^6$ time steps in a simulated RIT-2.5. A power of 0.8 W is deposited into the plasma by an alternating current (2.86 MHz) through the external coil. The density of the neutral background gas (Xenon) is $5 \cdot 10^{19}\,\mathrm{m}^{-3}$.

References

[1] D. M. Goebel and I. Katz. *Fundamentals of Electric Propulsion: Ion and Hall Thrusters*. JPL Space Science and Technology Series. John Wiley & Sons, 2008.

[2] H. W. Löb. Ein Elektrostatisches Raketentriebwerk mit Hochfrequenzionenquelle. *Astronautica Acta*, VIII(1):49, 1962.

[3] J. L. Van Noord. Lifetime assessment of the NEXT ion thruster. *43rd AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, 2007.

[4] O. A. Mitrofanova, R. Y. Gnizdor, V. M. Murashko, A. I. Koryakin, and A. N. Nesterenko. New generation of SPT-100. *32nd International Electric Propulsion Conference, Wiesbaden, Germany*, 2011.

[5] J. R. Brophy. NASA's Deep Space 1 ion engine (plenary). *Review of Scientific Instruments*, 73(2):1071–1078, 2002.

[6] D. Feili, B. Lotz, S. Bonnet, B. K. Meyer, H. W. Loeb, and N. Puetmann. μNRIT-2.5 - A new optimized microthruster of Giessen University. *31st International Electric Propulsion Conference, Ann Arbor, MI, USA*, 2009.

[7] K. H. Groh, H. W. Loeb, J. Mueller, W. Schmidt, and B. Schuetz. RIT-35 Rf-ion thruster - design and performance. *19th International Electric Propulsion Conference, Colorado Springs, USA*, 1987.

[8] R. Henrich. *Development of a Plasma Simulation Tool for Radio Frequency Ion Thrusters*. PhD thesis, Justus-Liebig-Universität Gießen, 2013.

[9] C. K. Birdsall and A. B. Langdon. *Plasma Physics via Computer Simulation*. Series in Plasma Physics and Fluid Dynamics. Taylor & Francis, 2004.

[10] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. CRC Press, 1988.

[11] Y. Takao, N. Kusaba, K. Eriguchi, and K. Ono. Two-dimensional particle-in-cell Monte Carlo simulation of a miniature inductively coupled plasma source. *Journal of Applied Physics*, 108, 2010.

[12] B. W. Yu and S. L. Girshick. Modeling inductively coupled plasmas: The coil current boundary condition. *Journal of Applied Physics*, 69(656), 1991.

[13] J. P. Boris. Relativistic plasma simulation-optimization of a hybrid code. *Proceeding of Fourth Conference on Numerical Simulations of Plasmas*, 1970.

[14] V. Vahedi and M. Surendra. A Monte Carlo collision model for the particle-in-cell method: applications to argon and oxygen discharges. *Computer Physics Communications*, 87(1):179 – 198, 1995.

[15] D. Tskhakaya, K. Matyash, R. Schneider, and F. Taccogna. The particle-in-cell method. *Contributions to Plasma Physics*, 47(8-9):563–594, 2007.

[16] M. M. Turner. Kinetic properties of particle-in-cell simulations compromised by Monte Carlo collisions. *Physics of Plasmas*, 13(3):033506, 2006.

[17] L. Clarke, I. Glendinning, and R. Hempel. The MPI message passing interface standard. In K.M. Decker and R.M. Rehmann, editors, *Programming Environments for Massively Parallel Distributed Systems. Monte Verità (Proceedings of the Centro Stefano Franscini Ascona)*, pages 213–218. Birkhäuser Basel, Basel, 1994.

[18] M. N. O. Sadiku. *Numerical Techniques in Electromagnetics, Second Edition.* Taylor & Francis, 2000.

[19] N. Köckler. *Mehrgittermethoden: Ein Lehr- und Übungsbuch.* SpringerLink : Bücher. Vieweg+Teubner Verlag, 2012.

[20] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial: Second Edition.* Society for Industrial and Applied Mathematics, 2000.

[21] U. M. Ascher and C. Greif. *A First Course on Numerical Methods.* Computational Science and Engineering. Society for Industrial and Applied Mathematics, 2011.

[22] C. Kanzow. *Numerik linearer Gleichungssysteme: Direkte und iterative Verfahren.* Springer-Lehrbuch. Springer Berlin Heidelberg, 2007.

[23] D. M. Young. Iterative methods for solving partial difference equations of elliptic type. *Transactions of the American Mathematical Society*, 76(1), 1954.

[24] S. Yang and M. K. Gobbert. The optimal relaxation parameter for the SOR method applied to the Poisson equation in any space dimensions. *Applied Mathematics Letters*, 22(3):325 – 331, 2009.

[25] W. Hackbusch. *Multi-Grid Methods and Applications.* Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2013.

[26] R. V. Southwell. Stress-calculation in frameworks by the method of "systematic relaxation of constraints". I and II. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 151(872):56–95, 1935.

[27] R. P. Fedorenko. A relaxation method for solving elliptic difference equations. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 1(5):922–927, 1961. Engl. translation published in *USSR Computational Mathematics and Mathematical Physics*, 1(4):1092 - 1096, 1962.

[28] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31(138):333–390, 1977.

[29] A. M. Bruaset and A. Tveito, editors. *Numerical Solution of Partial Differential Equations on Parallel Computers.* Lecture Notes in Computational Science and Engineering. Springer Berlin Heidelberg, 2006.

[30] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64–84, 1989.

[31] A. Brandt, S. F. McCormick, and J. Ruge. Algebraic multigrid (AMG) for sparse matrix equations. In D. J. Evans, editor, *Sparsity and its Applications*, pages 257–284. Cambridge University Press, Cambridge, UK, 1984.

[32] A. Brandt. Algebraic multigrid theory: The symmetric case. *Applied Mathematics and Computation*, 19(1):23 – 56, 1986.

[33] J. W. Ruge and K. Stüben. Algebraic multigrid. In S.F. McCormick, editor, *Multigrid Methods*, volume 3 of *Frontiers in Appied Mathematics*, pages 73–130. Society for Industrial and Applied Mathematics, Philadelphia, 1987.

[34] F. Hülsemann, M. Kowarschik, M. Mohr, and U. Rüde. Parallel geometric multigrid. In A. M. Bruaset and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume 51 of *Lecture Notes in Computational Science and Engineering*, pages 165–208. Springer Berlin Heidelberg, 2006.

[35] D. Xie and L. R. Scott. The parallel U-cycle multigrid method. *In Proceedings of the 8th Copper Mountain Conference on Multigrid Methods*, 1997.

[36] A. Brandt and O. E. Livne. *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, Revised Edition.* Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 2011.

[37] Y. Saad. *Iterative Methods for Sparse Linear Systems.* Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, second edition, 2003.

[38] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations.* Applied Mathematical Sciences. Springer International Publishing, 2016.

[39] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* SIAM, 1994.

[40] I. C. F. Ipsen and C. D. Meyer. The idea behind Krylov methods. *American Mathematical Monthly*, 105:889–899, 1997.

[41] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

[42] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[43] W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9(1):17–29, 1951.

[44] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, 1975.

[45] J. Erhel, K. Burrage, and B. Pohl. Restarted GMRES preconditioned by deflation. *Journal of Computational and Applied Mathematics*, 69(2):303–318, 1996.

[46] R. Hrach, M. Lahuta, Z. Pekarek, and J. Simek. Multi-dimensional codes for particle modelling of plasma-solid interaction at higher pressures. *Czechoslovak Journal of Physics*, 56(Suppl. 2):990–995, 2006.

[47] C. Vuik, J. J. I. M. van Kan, and P. Wesseling. A black box multigrid preconditioner for second order elliptic partial differential equations. In *European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS*, 2000.

[48] S. Bergler. Multigrid methods for arbitrary mesh sizes with application to quantum chemistry. Diploma thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2007.

[49] A. McAdams, E. Sifakis, and J. Teran. A parallel multigrid Poisson solver for fluids simulation on large grids. *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 65–74, 2010.

[50] T. Guillet and R. Teyssier. A simple multigrid scheme for solving the Poisson equation with arbitrary domain boundaries. *Journal of Computational Physics*, 230(12):4756–4771, 2011.

[51] F. Gibou, R. P. Fedkiw, L.-T. Cheng, and M. Kang. A second-order-accurate symmetric discretization of the Poisson equation on irregular domains. *Journal of Computational Physics*, 176(1):205–227, 2002.

[52] H. Johansen and P. Colella. A Cartesian grid embedded boundary method for Poisson's equation on irregular domains. *Journal of Computational Physics*, 147:60–85, 1998.

[53] P. Schwartz, M. Barad, P. Colella, and T. Ligocki. A Cartesian grid embedded boundary method for the heat equation and Poisson's equation in three dimensions. *Journal of Computational Physics*, 211:531–550, 2006.

[54] M. Oevermann and R. Klein. A Cartesian grid finite volume method for elliptic equations with variable coefficients and embedded interfaces. *Journal of Computational Physics*, 219(2):749–769, 2006.

[55] L. Botto. A geometric multigrid Poisson solver for domains containing solid inclusions. *Computer Physics Communications*, 184, 2013.

[56] D. Trebotich, M. F. Adams, S. Molins, C. I. Steefel, and C. Shen. High-resolution simulation of pore-scale reactive transport processes associated with carbon sequestration. *Computing in Science & Engineering*, 16(6):22–31, 2014.

[57] G. H. Shortley and R. Weller. The numerical solution of Laplace's equation. *Journal of Applied Physics*, 9(5):334–348, 1938.

[58] N. Matsunaga and T. Yamamoto. Superconvergence of the Shortley-Weller approximation for Dirichlet problems. *Journal of Computational and Applied Mathematics*, 116(2):263–273, 2000.

[59] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM Journal on Scientific Computing*, 34(1):A206–A239, 2012.

[60] J. F. Schäfer. Algorithmen zur numerischen Lösung der Poisson-Gleichung. Bachelor's thesis, Justus-Liebig-Universität Gießen, 2014.

[61] M. Stürmer. Optimierung von Mehrgitteralgorithmen auf der IA-64 Rechnerarchitektur. Diploma thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2006.

[62] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, K. Rupp, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2016.

[63] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.7, Argonne National Laboratory, 2016.

[64] S. Balay, W. D. Gropp, L. Curfman McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[65] A. Cohen. A performance analysis of 4X InfiniBand data transfer operations. In *Proceedings International Parallel and Distributed Processing Symposium*, 2003.

[66] S. Sur, M. J. Koop, and D. K. Panda. High-performance and scalable MPI over InfiniBand with reduced memory usage: An in-depth performance analysis. *Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing*, 2006.

[67] H. Subramoni, S. Chakraborty, and D. K. Panda. Designing dynamic and adaptive MPI point-to-point communication protocols for efficient overlap of computation and communication. In *International Supercomputing Conference*, pages 334–354, 2017.

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die mich in so vielerlei Hinsicht bei der Arbeit an dieser Dissertation unterstützt haben.
Zunächst ist dabei Herr Professor Dr. Christian Heiliger zu nennen, der sich bereit erklärt hat, meine Doktorarbeit zu betreuen und damit ein Projekt weiter zu fördern, das außerhalb seines eigentlichen Fachgebiets liegt. Nichtsdestotrotz waren unsere Besprechungen und Diskussionen immer ergiebig und führten zu neuen Ansätzen, eröffneten neue Perspektiven oder brachten mich dazu, eine bestimmte Richtung verstärkt weiterzuverfolgen.
Mein weiterer Dank gilt Herrn Dr. Robert Henrich, der mit seiner Arbeit an PlasmaPIC gewissermaßen die Grundvoraussetzungen für mein Promotionsprojekt geschaffen und auch in anderen Aspekten wertvolle Vorarbeit geleistet hat. Bei Fragen zur Performanceoptimierung und zur strategischen Vorgehensweise bei der Implementierung war er immer ansprechbar und brachte seine eigene Expertise ein.
Bei Detailfragen zur Arbeit mit HPC-Clustern und zum parallelen Programmieren mit MPI konnte ich zudem mehrfach auf die Hilfe von Herrn Dr. Michael Feldmann zurückgreifen, der sein Wissen bereitwillig geteilt hat.
Desweiteren möchte ich mich auch bei den restlichen Mitgliedern der AG Heiliger - sowohl bei den aktuellen wie auch bei den ehemaligen - für die zuvorkommende, freundliche und heitere Arbeitsatmosphäre bedanken.
Außerdem bedanke ich mich bei meinen Korrekturlesern für ihren Einsatz.
Abschließend möchte ich mich ganz herzlich bei meiner Familie für die ausdauernde Unterstützung während meines gesamten Studiums bedanken.

Eidesstattliche Erklärung

Ich erkläre: Ich habe die vorgelegte Dissertation selbständig und ohne unerlaubte fremde Hilfe und nur mit den Hilfen angefertigt, die ich in der Dissertation angegeben habe.
Alle Textstellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen sind, und alle Angaben, die auf mündlichen Auskünften beruhen, sind als solche kenntlich gemacht.
Ich stimme einer evtl. Überprüfung meiner Dissertation durch eine Antiplagiat-Software zu.
Bei den von mir durchgeführten und in der Dissertation erwähnten Untersuchungen habe ich die Grundsätze guter wissenschaftlicher Praxis, wie sie in der "Satzung der Justus-Liebig-Universität Gießen zur Sicherung guter wissenschaftlicher Praxis" niedergelegt sind, eingehalten.

(Ort, Datum)	(Unterschrift)